Mathematics of Language

Marcus Kracht
II. Mathematisches Institut
Freie Universität Berlin
Arnimallee 3
D – 14195 Berlin
kracht@math.fu-berlin.de

Preliminary Version

April 29, 2002

Preface

The present book developed from lectures held at the Freie Universität Berlin and seminars at the University of Potsdam on formal languages and mathematical linguistics. I wish to thank in particular the Department of Mathematics at the FU Berlin as well as the FU Berlin for their support and the always favourable conditions under which I was allowed to work.

Among my principal supporters I name Hans–Martin Gärtner, Ed Keenan, Hap Kolb and Uwe Mönnich. Without them I would not have had the energy to pursue this work and fill so many pages with symbols that create so much headache. They always encouraged me to go on.

Further, I wish to thank Helmut Alt, Christian Ebert, Benjamin Fabian, Stefanie Gehrke, Timo Hanke, Wilfrid Hodges, Gerhard Jäger, Franz Koniecny, Thomas Kosiol, Zsuzsanna Lipták, Jens Michaelis, István Németi, Stefan Salinger, Harald Stamm, Peter Staudacher, Wolfgang Sternefeld and Ngassa Tchao for their help in preparing this manuscript.

Berlin, 29th April 2002

Marcus Kracht

Introduction

This book is — as the title suggests — a book about the mathematical study of language, that is, about the description of language and languages with mathematical methods. It is intended for students of mathematics, linguistics, computer science, and computational linguistics, and also for all those who need or wish to understand the structure of language. It is a mathematical book; it cannot and does not intend to replace a genuine introduction to linguistics. For those who are not acquainted with general linguistics we recommend (Lyons, 1968), which is a bit outdated but still worth its while. No linguistic theory is discussed here in detail. This text only provides the mathematical background that will enable the reader to fully grasp the implications of these theories and understand them more thoroughly than before. Several topics of mathematical character have been omitted: there is for example no statistics, no learning theory, and no optimality theory. All these topics probably merit a book of their own. On the linguistic side the emphasis is on syntax and formal semantics, though morphology and phonology do play a role. These omissions are mostly due to my limited knowledge.

The main mathematical background is algebra and logic on the semantic side and strings on the syntactic side. In contrast to most introductions to formal semantics we do not start with logic — we start with strings and develop the logical apparatus as we go along. This is only a pedagogical decision. Otherwise, the book would start with a massive theoretical preamble after which the reader is kindly allowed to see some examples. Thus we have decided to introduce logical tools only when needed, not as overaching concepts, also since logic plays a major role only in semantics.

We do not distinguish between natural and formal languages. These two types of languages are treated completely alike. For a start it should not matter in principle whether what we have is a natural or an artificial product. Chemistry applies to naturally occurring substances as well as artificially produced ones. All we want to do here is study the structure of language. Noam Chomsky has repeatedly claimed that there is a fundamental difference between natural languages and nonnatural languages. Be this the case or not, this difference should not matter at the outset. To the contrary, the methods established here might serve as a tool in identifying what the difference is or might be. The present book also is not an introduction into the theory of formal languages; rather, it is an introduction into the mathematical theory of linguistics. The reader will therefore miss a few topics that are treated in depth in books on formal languages on the grounds that they are rather insignificant in linguistic theory. On the other hand, this book does treat subjects that are hardly found anywhere else in this form. The main characteristic of our approach is that we do not treat languages as sets of strings but as algebras of signs. This is much closer to the linguistic reality. We shall briefly sketch this approach, which will be introduced in detail in Chapter 3

A **sign** σ is defined here as a triple $\langle E, C, M\rangle$, where E is the **exponent of** σ, which typically is a string, C the (**syntactic**) **category of** σ, and M its **meaning**. By this convention a string is connected via the language with a set of meanings. Given a set Σ of signs, E **means** M **in** Σ if and only if there is a category C such that $\langle E, C, M\rangle \in \Sigma$. Seen this way, the task of language theory is not only to state which are the legitimate exponents of signs (as we find in the theory of formal languages as well as many treatises on generative linguistics which generously define language to be just syntax) but it must also say which string can have what meaning. The heart of the discussion is formed by the principle of compositionality, which in its weakest formulation says that the meaning of a a string (or other exponent) is found by homomorphically mapping its analysis into the semantics. Compositionality shall be introduced in Chapter 3 and we shall discuss at length its various ramifications. We shall also deal with Montague Semantics, which arguably was the first to state and execute this principle.

Once again, the discussion will be rather abstract, focussing on mathematical tools rather than the actual formulation of the theory. Anyhow, there are good introductions into the subject which eliminate the need to include details. One such book is (Dowty *et al.*, 1981) and the book by the collective of authors (Gamut, 1991a). A **system of signs** is a partial algebra of signs. This means that it is a pair $\langle \Sigma, M\rangle$, where Σ is a set of signs and M a finite set, the set of so called **modes of composition**. Standardly, one assumes M to have only one mode, a binary function $\bullet$, which allows to form a sign $\sigma_1 \bullet \sigma_2$ from two signs σ_1 and σ_2. The modes are generally partial operations. The action of $\bullet$ is explained by defining its action on the three components of the respective signs. We give a simple example. Suppose we have the following signs.

$$\begin{array}{rcl} \text{‘}\mathtt{runs}\text{’} & = & \langle \mathtt{runs}, v, f\rangle \\ \text{‘}\mathtt{Paul}\text{’} & = & \langle \mathtt{Paul}, n, p\rangle \end{array}$$

Here, v and n are the syntactic categories *(intransitive) verb* and *proper name*, respectively. p is a constant, which denotes an individual, namely Paul, and f is a function from individuals to the set of truth values, namely $\{0, 1\}$. (Furthermore, $f(x) = 1$ if x is running.) On the level of exponents we choose word concatenation, which is string composition with an interspersed blank. (Perfectionists will also add the period at the end ...) On the level of meanings we choose function application. Finally, let $\bullet_t$ be a partial function which is only defined if the first argument is n and the second is v and which in this case yields the value t. Now we put

$$\langle E_1, C_1, M_1\rangle \bullet \langle E_2, C_2, M_2\rangle := \langle E_1\ E_2, C_1 \bullet_1 C_2, M_2(M_1)\rangle$$

Then ‘`Paul`’ $\bullet$ ‘`runs`’ is a sign, and it has the following form.

$$\text{‘}\mathtt{Paul}\text{’} \bullet \text{‘}\mathtt{runs}\text{’} := \langle \mathtt{Paul\ runs}, t, f(p)\rangle$$

We shall say that this sentence is true if and only if $f(p) = 1$; otherwise we say that it is false. We hasten to add that ‘`Paul`’ $\bullet$ ‘`Paul`’ is *not* a sign. So, $\bullet$ is indeed a partial operation.

The key construct is the free algebra generated by the constant modes alone. This algebra is called the **algebra of structure terms**. The structure terms can be generated by a simple context free grammar. However, not every structure term names a sign. Since the algebras of exponents, categories and meanings are partial algebras it is in general not possible to define a homomorphism from the algebra of structure terms into the algebras of signs. All we can get is a partial homomorphism. In addition, the exponents are not always strings and the operations between them not only concatenation. Hence the defined languages can be very complex (indeed, every recursively enumerable language Σ can be so generated).

Before one can understand all this in full detail it is necessary to start off with an introduction into classical formal language theory using semi Thue–systems and grammars in the usual sense. This is what we shall do in the Chapter 1. It constitutes the absolute minimum one must know about these matters. Furthermore, we have added some sections containing basics from algebra, set theory, computability and linguistics. In Chapter 2 we study regular and context free languages in detail. We shall deal with the recognizability of these languages by means of automata, recognition and analysis problems, parsing, complexity, and ambiguity. At the end we shall discuss Parikh's Theorem.

In Chapter 3 we shall begin to study languages as systems of signs. Systems of signs and grammars of signs are defined in the first section. Then we shall concentrate on the system of categories and the so called categorial grammars. We shall introduce both the Ajdukiewicz–Bar Hillel Calculus and the Lambek–Calculus. We shall show that both can generate exactly the context free string languages. For the Lambek–Calculus this was for a long time an open problem, that was solved in the early 90's by Mati Pentus.

Chapter 4 deals with formal semantics. We shall develop some basic concepts of algebraic logic, and then deal with boolean semantics. We discuss the possibilities and limitations of Montague

Semantics, give a run down of Discourse Representation Theory, and of Dynamic Semantics. Furthermore, we shall introduce the reader to the theory of referent systems.

In the fifth chapter we shall treat so called PTIME languages. These are languages for which the parsing problem is decidable deterministically in polynomial time. The question whether or not natural languages are context free was considered settled negatively until the 1980. However, it was shown that most of the arguments were based on errors, and it seemed that none of them was actually tenable. Unfortunately, the conclusion that natural languages are actually all context free turned out to be premature again. It now seems that natural languages, at least some of them, are not context free. However, all known languages seem to be in PTIME. Moreover, the so called **weakly context sensitive languages** also belong to this class. A characterisation of this class in terms of a generating device was established by Bill Rounds, and in a different way by Annius Groenink, who introduced the notion of a literal movement grammar. In the final two sections we shall return to the question of compositionality in the light of the Leibniz' Principle, and then propose a new kind of grammars, de Saussure grammars, which eliminates the duplication of typing information found in categorial grammar.

The sixth chapter is devoted to the logical description of language. Also this approach has been introduced in the 1980s. The close connection between this approach and the so called *constraint*–programming is not accidental. It was proposed to view grammars not as generating devices but as theories of correct syntactic descriptions. This is very far from the tradition of generative grammar advocated by Chomsky, who always insisted on language containing an actual generating device (though on the other hand he characterizes this as a theory of competence). However, it turns out that there is a method to convert descriptions of syntactic structures into syntactic rules. This goes back to ideas by Büchi, Wright as well as Thatcher and Donner on theories of strings and theories of trees in monadic second order logic.

However, the reverse problem, extracting principles out of rules, is actually very hard, and its solvability depends on the strength of the description language. This opens the way into a logically based language hierarchy, which indirectly also reflects a complexity hierarchy. Chapter 6 ends with an overview of the major syntactic theories that have been introduced in the last 20 years.

NOTATION. A last word concerns our notational conventions. We use typewriter font for true characters in print. For example: `Maus` is the German word for 'mouse'. Its English counterpart appears in (English) texts either as `mouse` or as `Mouse`, depending on whether or not it occurs at the beginning of a sentence. Standard books on formal linguistics often ignore these points, but since strings are integral parts of signs we cannot afford this here. In between true characters in print we also use so called *metavariables* (placeholders) such as a (which denotes a single letter) and $\vec{x}$ (which denotes a string). The notation $\mathtt{c}_i$ is also used, which is short for the true letter `c` followed by the binary code of i (written with the help of appropriately chosen characters, for example `0` and `1`). When defining languages as sets of strings we distinguish between brackets that appear in print (these are `(` and `)`) and those which are just used to help the eye. People are used to employ abbreviatory conventions, for example `5+7+4` in place of `(5+(7+4))`. Also in logic one writes $\varphi \wedge \mathtt{(}\neg\chi\mathtt{)}$ or even $\varphi \wedge \neg\chi$ in place of $\mathtt{(}\varphi \wedge \mathtt{(}\neg\chi\mathtt{))}$. We shall follow that usage when the material shape of the formula is immaterial, but in that case we avoid using the true brackets `(`, `)` and use '(' and ')' instead. For $\varphi \wedge \mathtt{(}\neg\chi\mathtt{)}$ is actually *not* the same as $\mathtt{(}\varphi \wedge \mathtt{(}\neg\chi\mathtt{))}$. To an ordinary logician our notation may appear overly pedantic. However, since the character of the representation is part of what we are studying, syntactical issues play a vital role, and simply cannot be ignored. By contrast to brackets, $\langle$ and $\rangle$ are truly metalinguistic symbols that are used to define sequences. We use sans serife fonts for terms in formalized and computer languages, and attach a prime to refer to its denotation (or meaning). For example, the computer code for a while a loop is written $\mathsf{while}\ i < 100\ \mathsf{do}$

$x := x \times (x + i)$ od. This is just a string of symbols. However, the notation $\mathsf{see}'(\mathsf{john}', \mathsf{paul}')$ denotes the proposition that John sees Paul, not the sentence expressing that.

Contents

Chapter 1

Fundamental Structures

1.1 Algebras and Structures

In this section we shall provide definitions of basic terms and structures which we shall need throughout this book. Among them are the terms *algebra* and *structure*. Readers for whom these terms are entirely new are advised to read this section only cursorily and return to it only when they hit upon something for which they need background information.

We presuppose some familiarity with mathematical thinking, in particular some knowledge of elementary set theory, and proof techniques such as induction. For basic concepts in set theory see (Vaught, 1995) or (Just and Weese, 1996; Just and Weese, 1997); for background in logic see (Goldstern and Judah, 1995). Concepts from algebra (especially universal algebra) can be found in (Burris and Sankappanavar, 1981) and (Grätzer, 1968); for general background for lattices and order see (Grätzer, 1971) and (Davey and Priestley, 1990).

We use the symbols $\cup$ for the union, $\cap$ for the intersection of two sets. Instead of the difference symbol $M\backslash N$ we use $M - N$. $\varnothing$ denotes the empty set. $\wp(M)$ is the set of subsets of M, $\wp(M)$ the set of finite subsets of M. Sometimes it is necessary to take the union of two sets that does not identify the common symbols from the different sets. In that case one uses $+$. We define $M + N := M \times \{0\} \cup N \times \{1\}$. This is called the **disjoint union**. For reference, we fix the background theory of sets that we are using. This is the theory ZFC (Zermelo Fraenkel Set Theory with Choice). It is essentially a first order theory with only two two place relation symbols, $\in$ and $\doteq$. (See Section 3.8 for a definition of first order logic.) Its axioms are as follows (see (Vaught, 1995), (Just and Weese, 1996; Just and Weese, 1997) for the basics).

1. Singleton Set Axiom. $(\forall x)(\exists y)(\forall z)(x \in y \leftrightarrow z \doteq x)$.
   This makes sure that for every x we have a set $\{x\}$.
2. Powerset Axiom. $(\forall x)(\exists y)(\forall z)(z \subseteq x \leftrightarrow z \in y)$.
   This ensures that for every x the power set $\wp(x)$ of x exists.
3. Set Union. $(\forall x)(\exists y)(\forall z)(z \in y \leftrightarrow (\exists u)(z \in u \wedge u \in x))$.
   This makes sure that for every x the union $\bigcup_{z \in x} z$ exists, which we shall also denote by $\bigcup x$.
4. Extensionality. $(\forall xy)(x \doteq y \leftrightarrow (\forall z)(z \in x \leftrightarrow z \in y))$.
5. Replacement. If f is a function with domain x then the direct image of x under f is a set. (See below for a definition of *function*.)
6. Foundation. $(\forall x)(x \neq \varnothing \rightarrow (\exists y)(y \in x \wedge (\forall z)(z \in x \rightarrow z \notin y)))$.
   This says that in every set there exists an element that is minimal with respect to $\in$.
7. Comprehension. If x is a set and φ a first order property then $\{y : y \in x \wedge \varphi(y)\}$ also is a set.
8. Axiom of Infinity. There exists an x and an injective function $f : x \rightarrow x$ such that the direct image of x under f is not equal to x.
9. Axiom of Choice. If x is a set of sets then there exists a function $f : x \rightarrow \bigcup x$ with $f(y) \in y$ for all $y \in x$.

We remark here that in everyday discourse, comprehension is generally applied to all collections of sets, not just elementarily definable ones. This difference will hardly matter here; we only mention that if we employ monadic second order logic, we can express this as an axiom, as well as a true axiom of foundation. (Foundation is usually defined as follows: there is no infinite chain $x_0 \ni x_1 \ni x_2 \ni \ldots$.) In mathematical usage, one often forms certain collections of sets that can be shown not to be sets themselves. One example is the collection of all finite sets. The reason that it is not a set is as follows. For every set x, $\{x\}$ also is a set. The function $x \mapsto \{x\}$ is injective (by extensionality), and so there are as many finite sets as there are sets. If the collection of finite sets were a set, say y, its powerset has strictly more elements than y by a theorem of Cantor, but this is impossible, since y has the size of the universe. Nevertheless, mathematicians do use these collections (for example, the class of Ω–algebras), and they want to avail themselves of them. This is not a problem. Just notice that they are classes, and that classes are not members of sets, and no contradiction arises.

In set theory, numbers are defined as follows.

$$\begin{array}{lcl} 0 & := & \varnothing \,, \\ n+1 & := & \{k : k < n\} = \{0, 1, 2, \ldots, n-1\} \,. \end{array}$$

The set of so–constructed numbers is denoted by ω. It is the set of **natural numbers**. In general, an **ordinal** (**number**) is a set that is transitively and linearly ordered by $\in$. (See below for these concepts.) For two ordinals κ and λ, either $\kappa \in \lambda$ (for which we also write $\kappa < \lambda$) or $\kappa = \lambda$ or $\lambda \in \kappa$. The finite ordinals are exactly the natural numbers defined above. A **cardinal** (**number**) is an ordinal κ such that for every ordinal $\lambda < \kappa$ there is no injective map $f : \kappa \to \lambda$. The **cardinality** of a set M is the unique cardinal number κ such that there is a bijective function $f : M \to \kappa$. We denote κ by $|M|$. We distinguish between ω and its cardinality, $\aleph_0$. By definition, $\aleph_0$ is actually identical to ω so that it is not really necessary to distinguish the two. However, we shall do so here for reasons of clarity. (For example, infinite cardinals have a different arithmetic from ordinals.) If M is finite its cardinality is a natural number. If $|M| = \aleph_0$, M is called **countable**. If M has cardinality κ, the cardinality of $\wp(M)$ is denoted by 2^κ. $2^{\aleph_0}$ is the cardinality of the set of all real numbers. $2^{\aleph_0}$ is strictly greater than $\aleph_0$. Sets of this cardinality are uncountable. We remark here that the set of finite sets of natural numbers is countable.

If M and N are sets, $M \times N$ denotes the set of all pairs $\langle x, y\rangle$, where $x \in M$ and $y \in N$. First we have to define $\langle x, y\rangle$. A definition, which goes back to Kuratowski and Wiener, is as follows.

$$\langle x, y\rangle := \{x, \{x, y\}\}$$

Lemma 1.1.1 *$\langle x, y\rangle = \langle u, v\rangle$ if and only if $x = u$ and $y = v$.*

Proof. By extensionality, if $x = u$ and $y = v$ then $\langle x, y\rangle = \langle u, v\rangle$. Now assume that $\langle x, y\rangle = \langle u, v\rangle$. Then either $x = u$ or $x = \{u, v\}$, and $\{x, y\} = u$ or $\{x, y\} = \{u, v\}$. Now assume that $x = u$. Then $u \neq \{x, y\}$ since otherwise $x = \{x, y\}$, so $x \in x$, in violation to foundation. Hence we have $\{x, y\} = \{u, v\}$. We already know that $x = u$. Then certainly we must have $y = v$. This finishes the first case. Now we assume that $x = \{u, v\}$. Then $\{x, y\} = \{\{u, v\}, v\}$. Now either $\{x, y\} = \{\{u, v\}, v\} = u$ or $\{x, y\} = \{\{u, v\}, v\} = \{u, v\}$. Both contradict foundation. Hence this case cannot arise. So, $x = u$ and $y = v$, as promised. □

With these definitions, $M \times N$ is a set if M and N are sets. A **relation** from M to N is a subset of $M \times N$. We write $x\,R\,y$ if $\langle x, y\rangle \in R$. Particularly interesting is the case $M = N$. A relation $R \subseteq M \times M$ is called **reflexive** if $x\,R\,x$ for all $x \in M$; **symmetric** if from $x\,R\,y$ follows that $y\,R\,x$. R is called **transitive** if from $x\,R\,y$ and $y\,R\,z$ follows $x\,R\,z$. An **equivalence relation** on M is a reflexive, symmetric and transitive relation on M. A pair $\langle M, <\rangle$ is called an **ordered set** if M is a set and $<$ a transitive, irreflexive binary relation on M. $<$ is then called a (**strict**) **ordering on** M and M is then called **ordered by** $<$. $<$ is **linear** if for any two elements $x, y \in M$ either $x < y$ or $x = y$ or $y < x$. A

partial ordering is a relation which is reflexive, transitive and antisymmetric; the latter means that from $x\,R\,y$ and $y\,R\,x$ follows $x = y$.

If $R \subseteq M \times N$ is a relation, we write $R^{\smile} := \{\langle x, y\rangle : y\,R\,x\}$ for the so called **converse relation of** R. This is a relation from N to M. If $S \subseteq N \times P$, $T \subseteq M \times N$ are relation, put

$$\begin{array}{lcl} R \circ S & := & \{\langle x, y\rangle : \text{ for some } z : x\,R\,z\,S\,y\}\ , \\ R \cup T & := & \{\langle x, y\rangle : x\,R\,y \text{ or } x\,T\,y\}\ . \end{array}$$

We have $R \circ S \subseteq M \times P$ and $R \cup T \subseteq M \times N$. In case $M = N$ we still make further definitions. We put $\Delta_M := \{\langle x, x\rangle : x \in M\}$ and call this set the **diagonal on** M. Now put

$$\begin{array}{lcl} R^0 & := & \Delta_M\ , \\ R^{n+1} & := & R \circ R^n\ , \\ R^+ & := & \bigcup_{0<i\in\omega} R^i\ , \\ R^* & := & \bigcup_{i\in\omega} R^i\ . \end{array}$$

R^+ is the smallest transitive relation which contains R. It is therefore called the **transitive closure of** R. R^* is the smallest reflexive and transitive relation containing R.

A **function** from M to N is a relation $f \subseteq M \times N$ such that if $x\,f\,y$ and $x\,f\,z$ then $y = z$. We write $y = f(x)$ to say that $x\,f\,y$ and $f : M \to N$ to say that f is a function from M to N. If $P \subseteq M$ then $f \restriction P := f \cap (P \times N)$. Further, $f : M \twoheadrightarrow N$ abbreviates that f is a **surjective** function, that is, every $y \in N$ is of the form $y = f(x)$ for some $x \in M$. And we write $f : M \rightarrowtail N$ to say that f is **injective**, that is, for all $x, x' \in M$, if $f(x) = f(x')$ then $x = x'$. f is **bijective** if it is injective as well as surjective. Finally, we write $f : x \mapsto y$ if $y = f(x)$. If $X \subseteq M$ then $f[X] := \{f(x) : x \in X\}$ is the so called **direct image of** X **under** f. We warn the reader of the difference between $f(X)$ and $f[X]$. Let for example the function $S : \omega \to \omega : x \mapsto x + 1$ be given. Then according to the definition of natural numbers above we have $S(4) = 5$ and $S[4] = \{1, 2, 3, 4\}$, since $4 = \{0, 1, 2, 3\}$. Let

M be an arbitrary set. There is a bijection between subsets of M and functions from M to $2 = \{0, 1\}$ which is defined as follows. For $N \subseteq M$ we call $\chi_N : M \to \{0, 1\}$ the **characteristic function** of N if $\chi_N(x) = 1$ if and only if $x \in N$. Let $y \in N$ and $Y \subseteq N$; then put $f^{-1}(y) := \{x : f(x) = y\}$ and $f^{-1}[Y] := \{x : f(x) \in Y\}$. If f is injective, $f^{-1}(y)$ denotes the unique x such that $f(x) = y$ (if that exists). We shall see to it that this overload in notation does not give rise to confusions.

M^n, $n \in \omega$, denotes the set of n–tuples of elements from M. We can precisify this as follows.

$$\begin{array}{lcl} M^1 & := & M \ , \\ M^{n+1} & := & M^n \times M \ . \end{array}$$

In addition $M^0 := 1(= \{\varnothing\})$. Then an n–tuple of elements from M is an element of M^n. Depending on need we shall write $\langle x_i : i < n\rangle$ or $\langle x_0, x_2, \ldots, x_{n-1}\rangle$ for an n–tuple over M.

An n–**ary relation** on M is a subset of M^n, an n–**ary function** on M is a function $f : M^n \to M$. Here $n = 0$ is of course an option. A 0–ary relation is a subset of 1, hence it is either the empty set or the set 1 itself. A 0–ary function on M is a function $c : 1 \to M$. We also call it a **constant**. The **value** of this constant is the element $c(\varnothing)$. Let R be an n–ary relation and $\vec{x} \in M^n$. Then we write $R(\vec{x})$ in place of $\vec{x} \in R$.

Now let F be a set and $\Omega : F \to \omega$. The pair $\langle F, \Omega\rangle$, also denoted by Ω alone, is called a **signature** and F the set of **function symbols**.

Definition 1.1.2 *Let $\Omega : F \to \omega$ be a signature and A a nonempty set. Further, let Π be a mapping which assigns to every $f \in F$ an $\Omega(f)$–ary function on A. Then we call the pair $\mathfrak{A} := \langle A, \Pi\rangle$ an Ω–* ***algebra****. Ω–algebras are in general denoted by upper case German letters.*

In order not to get drowned in notation we adopt the following general usage. If $\mathfrak{A}$ is an Ω–algebra, we write $f^{\mathfrak{A}}$ for the function

$\Pi(f)$. In place of denoting $\mathfrak{A}$ by the pair $\langle A, \Pi\rangle$ we shall denote it by $\langle A, \{f^{\mathfrak{A}} : f \in F\}\rangle$. We warn the reader that the latter notation may give rise to confusion since functions of the same arity can be associated with different function symbols. We shall see to it that these problems shall not arise.

The set of Ω–terms is the smallest set Tm_Ω for which the following holds.

(†) If $f \in F$ and $t_i \in \mathrm{Tm}_\Omega$, $i < \Omega(f)$, also $f(t_0, \ldots, t_{\Omega(f)-1}) \in \mathrm{Tm}_\Omega$.

Terms are abstract entities; they are not to be equated with functions nor with the strings by which we denote them. To begin with we define the **level** of a term. If $\Omega(f) = 0$, then $f()$ is a term of level 0, which we also denote by 'f'. If t_i, $i < \Omega(f)$, are terms of level n_i, then $f(t_0, \ldots, t_{\Omega(f)-1})$ is a term of level $1 + max\{n_i : i < \Omega(f)\}$. Many proofs run by induction on the level of terms, we therefore speak about *induction on the construction of the term*. Two terms u and v are equal, in symbols $u = v$, if they have identical level and either they are both of level 0 and there is an $f \in F$ such $u = v = f()$ or there is an $f \in F$, and terms s_i, t_i, $i < \Omega(f)$, such that $u = f(s_0, \ldots, s_{\Omega(f)-1})$ and $v = f(t_0, \ldots, t_{\Omega(f)-1})$ as well as $s_i = t_i$ for all $i < \Omega(f)$.

An important example of an Ω–algebra is the so called *term algebra*. We choose an arbitrary set X of symbols, which must be disjoint to F. The signature is extended to $F \cup X$ such that the symbols of X have arity 0. The terms over this new signature are called **Ω–terms over X**. The set of Ω–terms over X is denoted by $\mathrm{Tm}_\Omega(X)$. Then we have $\mathrm{Tm}_\Omega = \mathrm{Tm}_\Omega(\varnothing)$. For many purposes (indeed most of the purposes of this book) the terms Tm_Ω are sufficient. For we can always resort to the following trick. For each $x \in X$ add a 0–ary function symbol $\underline{x}$ to F. This gives a new signature Ω_X, also called the **constant expansion of** Ω by X. Then Tm_{Ω_X} can be canonically identified with $\mathrm{Tm}_\Omega(X)$.

The terms are made the objects of an algebra, and the function symbols are interpreted by functions. Namely, we put:

$$\Pi(f) : \langle t_i : i < \Omega(f)\rangle \mapsto f(t_1, \ldots, t_{\Omega(f)-1}) \; .$$

Then $\langle \mathrm{Tm}_\Omega(X), \Pi\rangle$ is an Ω–algebra, called the **term algebra generated by** X. It has the following property. For any Ω–algebra $\mathfrak{A}$ and any map $v : X \to A$ there is exactly one homomorphism $\overline{v} : \mathrm{Tm}_\Omega(X) \to \mathfrak{A}$ such that $\overline{v} \restriction X = v$.

Definition 1.1.3 *Let $\mathfrak{A}$ be an Ω–algebra and $X \subseteq A$. We say that X **generates** $\mathfrak{A}$ if A is the smallest subset which contains X and which is closed under all functions $f^{\mathfrak{A}}$. If $|X| = \kappa$ we say that $\mathfrak{A}$ is κ–**generated**. Let $\mathcal{K}$ be a class of Ω–algebras and $\mathfrak{A} \in \mathcal{K}$. We say that $\mathfrak{A}$ is **freely generated by** X if for every $\mathfrak{B} \in \mathcal{K}$ and maps $v : X \to B$ there is exactly one homomorphism $\overline{v} : \mathfrak{A} \to \mathfrak{B}$ such that $\overline{v} \restriction X = v$. If $|X| = \kappa$ we say that $\mathfrak{A}$ is **freely** κ–**generated in** $\mathcal{K}$.*

Proposition 1.1.4 *Let Ω be a signature, and let X be disjoint from F. Then the termalgebra over X, $\mathfrak{Tm}_\Omega(X)$ is freely generated by X the class of all Ω–algebras.*

The following is left as an exercise. It is the justification for writing $\mathfrak{Fr}_{\mathcal{K}}(\kappa)$ for the (up to isomorphism unique) freely κ–generated algebra of $\mathcal{K}$.

Proposition 1.1.5 *Let $\mathcal{K}$ be a class of Ω–algebras and κ a cardinal number. If $\mathfrak{A}$ and $\mathfrak{B}$ are both freely κ–generated in $\mathcal{K}$ they are isomorphic.*

Maps of the form $\sigma : X \to \mathrm{Tm}_\Omega(X)$, as well as their homomorphic extensions are called **substitutions**. If t is a term over X, we also write $\sigma(t)$ in place of $\overline{\sigma}(t)$. Another notation, frequently employed in this book, is as follows. Given terms s_i, $i < n$, we write $[s_i/x_i : i < n]t$ in place of $\sigma(t)$, where σ is defined as follows.

$$\sigma(y) := \begin{cases} s_i & \text{if } y = x_i, \\ y & \text{else.} \end{cases}$$

(Most authors write $t[s_i/x_i : i < n]$, but this notation will cause confusion with other notation that we use.)

Terms induce term functions on a given Ω–algebra $\mathfrak{A}$. Let t be a term with variables x_i, $i < n$. (None of these variables need to occur in the term.) Then $t^{\mathfrak{A}} : A^n \to A$ is defined inductively as follows (with $\vec{a} = \langle a_i : i < \Omega(f)\rangle$).

1. $x_i^{\mathfrak{A}} : \langle a_i : i < n\rangle \mapsto a_i$.
2. $(f(t_0, \dots, f_{\Omega(f)-1}))^{\mathfrak{A}}(\vec{a}) := f^{\mathfrak{A}}(t_0^{\mathfrak{A}}(\vec{a}), \dots, t_{\Omega(f)-1}^{\mathfrak{A}}(\vec{a}))$.

We denote by $\mathbf{Clo}_n(\mathfrak{A})$ the set of n–ary term functions on $\mathfrak{A}$. This set is also called the **clone of n–ary term functions of $\mathfrak{A}$**. A **polynomial of $\mathfrak{A}$** is a term function over an algebra that is like $\mathfrak{A}$ but additionally has a constant for each element for A. (So, we form the constant expansion of the signature with every $a \in A$. Moreover, $\underline{a}$ (more exactly, $\overline{a}()$) shall have value a in A.) The clone of n–ary term functions of this algebra is called $\mathbf{Pol}_n(\mathfrak{A})$. For example, $((x_0 + x_1) \cdot x_0)$ is a term and denotes a binary term function in an algebra for the signature containing only $\cdot$ and $+$. However, $(2 + (x_0 \cdot x_0))$ is a polynomial but not a term. Suppose that we add a constant $\mathbf{1}$ to the signature, with denotation 1 in the natural numbers. Then $(2 + (x_0 \cdot x_0))$ is still not a term of the expanded language (it lacks the symbol 2), but the associated function actually is a term function, since it is identical with the function induced by the term $((\mathbf{1} + \mathbf{1}) + (x_0 \cdot x_0))$.

Definition 1.1.6 *Let $\mathfrak{A} = \langle A, \{f^{\mathfrak{A}} : f \in F\}\rangle$ and $\mathfrak{B} = \langle B, \{f^{\mathfrak{B}} : f \in F\}\rangle$ be Ω–algebras and $h : A \to B$. h is called a* ***homomorphism*** *if for every $f \in F$ and every $\Omega(f)$–tuple $\vec{x} \in A^{\Omega(f)}$ we have*

$$h(f^{\mathfrak{A}}(x_0, x_1, \dots, x_{\Omega(f)-1})) = f^{\mathfrak{B}}(h(x_0), h(x_1), \dots, h(x_{\Omega(f)-1})) \ .$$

We write $h : \mathfrak{A} \to \mathfrak{B}$ if h is a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. Further, we write $h : \mathfrak{A} \twoheadrightarrow \mathfrak{B}$ if h is a surjective homomorphism and $h : \mathfrak{A} \rightarrowtail \mathfrak{B}$ if h is an injective homomorphism. h is an

isomorphism *if h is injective as well as surjective.* $\mathfrak{B}$ *is called* ***isomorphic*** *to* $\mathfrak{A}$ *if there is an isomorphism from* $\mathfrak{A}$ *to* $\mathfrak{B}$*; we then write* $\mathfrak{A} \cong \mathfrak{B}$*. If* $\mathfrak{A} = \mathfrak{B}$ *then we call h an* ***endomorphism of*** $\mathfrak{A}$*; if h is additionally bijective then h is called an* ***automorphism*** *of* $\mathfrak{A}$.

If $h : A \to B$ is an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ then $h^{-1} : B \to A$ is an isomorphism from $\mathfrak{B}$ to $\mathfrak{A}$.

Definition 1.1.7 *Let* $\mathfrak{A}$ *be an* Ω*–algebra and* Θ *a binary relation on A.* Θ *is called a* ***congruence relation on*** $\mathfrak{A}$ *if* Θ *is an equivalence relation and for all $f \in F$ and all* $\vec{x}, \vec{y} \in A^{\Omega(f)}$ *we have:*

$$(\ddagger) \qquad \text{If } x_i \,\Theta\, y_i \text{ for all } i < \Omega(f) \text{ then } f^{\mathfrak{A}}(\vec{x}) \,\Theta\, f^{\mathfrak{A}}(\vec{y}).$$

If Θ is an equivalence relation put

$$[x]\Theta := \{y : x \,\Theta\, x\} \ .$$

We call $[x]\Theta$ the **equivalence class of** x. Then for all x and y we have either $[x]\Theta = [y]\Theta$ or $[x]\Theta \cap [y]\Theta = \varnothing$. Further, we always have $x \in [x]\Theta$. If Θ additionally is a congruence relation, then the following holds: if $y_i \in [x_i]\Theta$ for all $i < \Omega(f)$ then $f^{\mathfrak{A}}(\vec{y}) \in [f^{\mathfrak{A}}(\vec{x})]\Theta$. Therefore the following definition is independent of representatives.

$$[f^{\mathfrak{A}}]\Theta([x_0]\Theta, [x_1]\Theta, \ldots, [x_{\Omega(f)-1}]\Theta) := [f^{\mathfrak{A}}(x_0, x_1, \ldots, x_{\Omega(f)-1})]\Theta \ .$$

Namely, let $y_0 \in [x_0]\Theta$, …, $y_{\Omega(f)-1} \in [x_{\Omega(f)-1}]\Theta$. Then $y_i \;\Theta\; x_i$ for all $i < \Omega(f)$. Then because of $(\ddagger)$ we immediately have $f^{\mathfrak{A}}(\vec{y}) \;\Theta\; f^{\mathfrak{A}}(\vec{x})$. This simply means $f^{\mathfrak{A}}(\vec{y}) \in [f^{\mathfrak{A}}(\vec{x})]\Theta$. Put $A/\Theta := \{[x]\Theta : x \in A\}$. We denote the algebra $\langle A/\Theta, \{[f^{\mathfrak{A}}]\Theta : f \in F\}\rangle$ by $\mathfrak{A}/\Theta$. We call $\mathfrak{A}/\Theta$ the **factorization of** $\mathfrak{A}$ **by** Θ. The map $h_\Theta : x \mapsto [x]\Theta$ is easily proved to be a homomorphism.

Conversely, let $h : \mathfrak{A} \to \mathfrak{B}$ be a homomorphism. Then put $ker(h) := \{\langle x, y\rangle \in A^2 : h(x) = h(y)\}$. $ker(h)$ is a congruence relation on $\mathfrak{A}$. Furthermore, $\mathfrak{A}/ker(h)$ is isomorphic to $\mathfrak{B}$ if h is surjective. A set $B \subseteq A$ is **closed under** $f \in F$ if for all $\vec{x} \in B^{\Omega(f)}$ we have $f^{\mathfrak{A}}(\vec{x}) \in B$.

Definition 1.1.8 *Let $\langle A, \{f^{\mathfrak{A}} : f \in F\}\rangle$ be an Ω–algebra and $B \subseteq A$ closed under all $f \in F$. Put $f^{\mathfrak{B}}(\vec{x}) := f^{\mathfrak{A}}(\vec{x})$. Then $f^{\mathfrak{B}} : B^{\Omega(f)} \to B$. The pair $\langle B, \{f^{\mathfrak{B}} : f \in F\}\rangle$ is called a* ***subalgebra*** *of $\mathfrak{A}$.*

Given algebras $\mathfrak{A}_i$, $i \in I$, we form the product of these algebras in the following way. The carrier set is the set of functions $\alpha : I \to \bigcup_{i \in I} A_i$ such that $\alpha(i) \in A_i$ for all $i \in I$. Call this set P. For an n–ary function symbol f the function $f^{\mathfrak{P}}$ is defined as follows.

$$f^{\mathfrak{P}}(\alpha_0, \dots, \alpha_{n-1})(i) := \langle f^{\mathfrak{A}_i}(\alpha_0(i)), f^{\mathfrak{A}_i}(\alpha_1(i)), \dots, f^{\mathfrak{A}_i}(\alpha_{n-1}(i))\rangle$$

The resulting algebra is denoted by $\prod_{i \in I} \mathfrak{A}_i$. One also defines the product $\mathfrak{A} \times \mathfrak{B}$ in the following way. The carrier set is $A \times B$ and for an n–ary function symbol f we put

$$\begin{aligned} & f^{\mathfrak{A}\times\mathfrak{B}}(\langle a_0, b_0\rangle, \dots, \langle a_{n-1}, b_{n-1}\rangle) := \\ & \qquad \langle f^{\mathfrak{A}}(a_0, \dots, a_{n-1}), f^{\mathfrak{B}}(b_0, \dots, b_{n-1})\rangle \ . \end{aligned}$$

The algebra $\mathfrak{A} \times \mathfrak{B}$ is isomorphic to the algebra $\prod_{i \in 2} \mathfrak{A}_i$, where $\mathfrak{A}_0 := \mathfrak{A}$, $\mathfrak{A}_1 := \mathfrak{B}$. However, the two algebras are not identical. (Can you verify this?)

A particularly important concept is that of a **variety** or **equationally definable class of algebras**.

Definition 1.1.9 *Let Ω be a signature. A class of Ω–algebras is called a* ***variety*** *if it is closed under isomorphic copies, subalgebras, homomorphic images, and taking (possibly infinite) products.*

Let $V := \{x_i : i \in \omega\}$ be the set of variables. An **equation** is a pair $\langle s, t\rangle$ of Ω–terms (involving variables from V). We simply write $s \doteq t$ in place of $\langle s, t\rangle$. An algebra $\mathfrak{A}$ satisfies the equation $s \doteq t$ if and only if for all maps $v : V \to A$, $\overline{v}(s) = \overline{v}(t)$. We write $\mathfrak{A} \vDash s \doteq t$. A class $\mathcal{K}$ of Ω–algebras satisfies this equation if every algebra of $\mathcal{K}$ satisfies it. We write $\mathcal{K} \vDash s \doteq t$.

Proposition 1.1.10 *The following holds for all classes $\mathcal{K}$ of Ω–algebras.*

1. *$\mathcal{K} \vDash s \doteq t$.*

2. *If $\mathcal{K} \vDash s \doteq t$ then $\mathcal{K} \vDash t \doteq s$.*

3. *If $\mathcal{K} \vDash s \doteq t; t \doteq u$ then $\mathcal{K} \vDash s \doteq u$.*

4. *If $\mathcal{K} \vDash s_i \doteq t_i$ for all $i < \Omega(f)$ then $\mathcal{K} \vDash f(\vec{s}) \doteq f(\vec{t})$.*

5. *If $\mathcal{K} \vDash s \doteq t$ and $\sigma : V \to \mathrm{Tm}_\Omega(V)$ is a substitution, then $\mathcal{K} \vDash \sigma(s) \doteq \sigma(t)$.*

The verification of this is routine. It follows from the first three facts that equality is an equivalence relation on the algebra $\mathfrak{Tm}_\Omega(V)$, and together with the fourth that the set of equations valid in $\mathcal{K}$ form a congruence on $\mathfrak{Tm}_\Omega(V)$. There is a bit more we can say. Call a congruence Θ on $\mathfrak{A}$ **fully invariant** if for all endomorphisms $h : \mathfrak{A} \to \mathfrak{A}$: if $x \ \Theta \ y$ then $h(x) \ \Theta \ h(y)$. The next theorem follows immediately once we observe that the endomorphisms of $\mathfrak{Tm}_\Omega(V)$ are exactly the substitution maps. To this end, let $h : \mathfrak{Tm}_\Omega(V) \to \mathfrak{Tm}_\Omega(V)$. Then h is uniquely determined by $h \upharpoonright V$, since $\mathfrak{Tm}_\Omega(V)$ is freely generated by V. It is easily computed that h is the substitution defined by $h \upharpoonright V$. Moreover, every map $v : V \to \mathfrak{Tm}_\Omega(V)$ induces a (unique) homomorphism $\overline{v} : \mathfrak{Tm}_\Omega(V) \to \mathfrak{Tm}_\Omega(V)$. Now write $\mathrm{Eq}(\mathcal{K}) := \{\langle s, t\rangle : \mathcal{K} \vDash s \doteq t\}$.

Corollary 1.1.11 *Let $\mathcal{K}$ be a class of Ω–algebras. Then* $\mathrm{Eq}(\mathcal{K})$ *is a fully invariant congruence on $\mathfrak{Tm}_\Omega(V)$.*

Let E be a set of equations. Then put

$$\mathrm{Alg}(E) := \{\mathfrak{A} : \text{ for all } s \doteq t \in E : \mathfrak{A} \vDash s \doteq t\}$$

This is a class of Ω–algebras. Classes of Ω–algebras that have the form $\mathrm{Alg}(E)$ for some E are called **equationally definable**. The next theorem asserts that equationally definable classes are varieties.

Proposition 1.1.12 *Let E be a set of equations. Then* $\mathrm{Alg}(E)$ *is a variety.*

We state without proof the following result.

Theorem 1.1.13 (Birkhoff) *Every variety is an equationally definable class. Furthermore, there is a biunique correspondence between varieties and fully invariant congruences on the algebra* $\mathfrak{Tm}_\Omega(V)$.

The idea for the proof is as follows. It can be shown that a variety has free algebras. For every cardinal number κ, $\mathfrak{Fr}_{\mathcal{K}}(\kappa)$ exists. Moreover, a variety is uniquely characterized by $\mathfrak{Fr}_{\mathcal{K}}(\aleph_0)$. In fact, every algebra is a subalgebra of a direct image of some product of $\mathfrak{Fr}_{\mathcal{K}}(\aleph_0)$. Thus, we need to investigate the equations that hold in the latter algebra. The other algebras will satisfy these equations, too. The free algebra is the image of $\mathfrak{Tm}_\Omega(V)$ under the map $x_i \mapsto i$. The induced congruence is fully invariant, by the freeness of $\mathfrak{Fr}_{\mathcal{K}}(\aleph_0)$. Hence, this congruence simply *is* the set of equations valid in the free algebra, hence in the whole variety. Finally, if E is a set of equations, we write $E \vDash t \doteq u$ if $\mathfrak{A} \vDash t \doteq u$ for all $\mathfrak{A} \in \mathsf{Alg}(E)$.

Theorem 1.1.14 (Birkhoff) $E \vDash t \doteq u$ *if and only if* $t \doteq u$ *can be derived from* E *by means of the rules given in Proposition 1.1.10.*

The relation between classes of algebras and sets of equations is called a **Galois–correspondence**. It is useful to know a few facts about such correspondences. Let A, B be sets and $R \subseteq A \times B$ (it is easier but not necessary to just look at sets here). The triple $\langle A, B, R\rangle$ is called a **context**. Now define the following operator:

$$^{\uparrow} : \wp(A) \to \wp(B) : O \mapsto \{y \in B : \text{ for all } x \in O : x\ R\ y\}$$

One calls $O^{\uparrow}$ the **intent of** O. Similarly, we define the **extent** of $P \subseteq B$:

$$^{\downarrow} : \wp(B) \to \wp(A) : P \mapsto \{x \in A : \text{ for all } y \in P : x\ R\ y\}$$

Theorem 1.1.15 *Let* $\langle A, B, R\rangle$ *be a context. Then the following holds for all* $O, O' \subseteq A$ *and all* $P, P' \subseteq B$.

1. *$O \subseteq P^{\downarrow}$ if and only if $O^{\uparrow} \supseteq P$.*

2. *If $O \subseteq O'$ then $O^{\uparrow} \supseteq O'^{\uparrow}$.*

3. *If $P \subseteq P'$ then $P^{\downarrow} \supseteq P'^{\downarrow}$.*

4. *$O \subseteq O^{\uparrow\downarrow}$.*

5. *$P \subseteq P^{\downarrow\uparrow}$.*

Proof. Notice that if $\langle A, B, R\rangle$ is a context, $\langle B, A, R^{\smile}\rangle$ also is a context, and so we need only show the claims (1), (2) and (4). (1). $O \subseteq P^{\downarrow}$ if and only if every $x \in O$ stands in relation to every member of P if and only if $P \subseteq O^{\uparrow}$. (2). If $O \subseteq O'$ and $y \in O'^{\uparrow}$, then for every $x \in O'$: $x \mathrel{R} y$. This means that for every $x \in O$: $x \mathrel{R} y$, which is the same as $y \in O^{\uparrow}$. (4). Notice that $O^{\uparrow} \supseteq O^{\uparrow}$ by (1) implies $O \subseteq O^{\uparrow\downarrow}$. □

Definition 1.1.16 *Let M be a set and $H : \wp(M) \to \wp(M)$ a function. H is called a* ***closure operator on*** *M if for all $X, Y \subseteq M$ the following holds.*

1. *$X \subseteq H(X)$.*

2. *If $X \subseteq Y$ then $H(X) \subseteq H(Y)$.*

3. *$H(X) = H(H(X))$.*

Proposition 1.1.17 *Let $\langle A, B, R\rangle$ be a context. Then $O \mapsto O^{\uparrow\downarrow}$ and $P \mapsto P^{\downarrow\uparrow}$ are closure operators on A and B, respectively. The closed sets are the sets of the form $P^{\downarrow}$ for the first, and of the form $O^{\uparrow}$ for the second operator.*

Proof. We have $O \subseteq O^{\uparrow\downarrow}$, from which $O^{\uparrow} \supseteq O^{\uparrow\downarrow\uparrow}$. On the other hand, $O^{\uparrow} \subseteq O^{\uparrow\downarrow\uparrow}$, so that we get $O^{\uparrow} = O^{\uparrow\downarrow\uparrow}$. Likewise, $P^{\downarrow} = P^{\downarrow\uparrow\downarrow}$ is shown. The claims now follow easily. □

Definition 1.1.18 *Let $\langle A, B, R\rangle$ be a context. A pairs $\langle O, P\rangle \in \wp(A) \times \wp(B)$ is called a* ***concept*** *if $O = P^{\downarrow}$ and $P = O^{\uparrow}$.*

Theorem 1.1.19 *Let $\langle A, B, R\rangle$ be a context. The concepts are exactly the pairs of the form $\langle P^{\downarrow}, P^{\downarrow\uparrow}\rangle$, $P \subseteq B$, or, alternatively, the pairs of the form $\langle O^{\uparrow\downarrow}, O^{\uparrow}\rangle$, $O \subseteq A$.*

As a particular application we look again at the connection between classes of Ω–algebras and sets of equations over Ω–terms. (It suffices to take the set of Ω–algebras of size $< \kappa$ for a suitable κ to make this work.) Let Alg_Ω denotes the class of Ω–algebras, Eq_Ω the set of equations. The triple $\langle \mathsf{Alg}_\Omega, \mathsf{Eq}_\Omega, \vDash\rangle$ is a context, and the map $^{\uparrow}$ is nothing but Eq and the map $^{\downarrow}$ nothing but Alg. The classes $\mathsf{Alg}(E)$ are the equationally definable classes, $\mathsf{Eq}(\mathcal{K})$ the equations valid in $\mathcal{K}$. Concepts are pairs $\langle \mathcal{K}, E\rangle$ such that $\mathcal{K} = \mathsf{Alg}(E)$ and $E = \mathsf{Eq}(\mathcal{K})$.

Often we shall deal with structures in which there are also relations in addition to functions. The definitions, inasfar as they still make sense, are carried over analogously. However, the notation becomes more clumsy.

Definition 1.1.20 *Let F and G be disjoint sets and $\Omega : F \to \omega$ as well as $\Xi : G \to \omega$ functions. A pair $\mathfrak{A} = \langle A, \mathfrak{I}\rangle$ is called an $\langle \Omega, \Xi\rangle$–**structure** if for all $f \in F$ $\mathfrak{I}(f)$ is an $\Omega(f)$–ary function on A and for each $g \in G$ $\mathfrak{I}(g)$ is a $\Xi(g)$–ary relation on A. Ω is called the **functional signature**, Ξ the **relational signature** of $\mathfrak{A}$.*

Whenever we can afford it we shall drop the qualification '$\langle \Omega, \Xi\rangle$' and simply talk of 'structures'. If $\langle A, \mathfrak{I}\rangle$ is an $\langle \Omega, \Xi\rangle$–structure, then $\langle A, \mathfrak{I} \restriction F\rangle$ is an Ω–algebra. An Ω–algebra can be thought of in a natural way as a $\langle \Omega, \varnothing\rangle$–structure, where $\varnothing$ is the empty relational signature. We use a convention similar to that of algebras. Furthermore, we denote relations by upper case Roman letters such as R, S and so on. Now let $\mathfrak{A} = \langle A, \{f^{\mathfrak{A}} : f \in F\}, \{R^{\mathfrak{A}} : R \in G\}\rangle$ and $\mathfrak{B} = \langle B, \{f^{\mathfrak{B}} : f \in F\}, \{R^{\mathfrak{B}} : R \in G\}\rangle$ be structures of the same signature. A map $h : A \to B$ is called an **isomorphism** from $\mathfrak{A}$ to $\mathfrak{B}$, if h is bijective and for all $f \in F$ and all $\vec{x} \in A^{\Omega(f)}$ we have

$$h(f^{\mathfrak{A}}(\vec{x})) = f^{\mathfrak{B}}(h(\vec{x})) ,$$

as well as for all $R \in G$ and all $\vec{x} \in A^{\Xi(R)}$

$$R^{\mathfrak{A}}(x_0, x_1, \ldots, x_{\Xi(R)-1}) \quad \Leftrightarrow \quad R^{\mathfrak{B}}(h(x_0), h(x_1), \ldots, h(x_{\Xi(R)-1})) \ .$$

In general, there is no good notion of a homomorphism. It is anyway not needed for us.

Exercise 1. Determine the sets 0, 1, 2, 3 and 4. Draw them by representing each member by a vertex and drawing an arrow from x to y if $x \in y$. What do you see?

Exercise 2. Let $f : M \to N$ and $g : N \to P$. Show that if $g \circ f$ is surjective, g is surjective, and that if $g \circ f$ is injective, f is injective. Give in each case an example that the converse fails.

Exercise 3. In set theory, one writes ${}^N M$ for the set of functions from N to M. Show that if $|N| = n$ and $|M| = m$, then $|{}^N M| = m^n$. Deduce that $|{}^N M| = |M^n|$. Can you find a bijection between these sets?

Exercise 4. Show that for relations $R, R' \subseteq M \times N$, $S, S' \subseteq N \times P$ we have

$$\begin{array}{rcl} (R \cup R') \circ S & = & (R \circ S) \cup (R' \circ S) \\ R \circ (S \cup S') & = & (R \circ S) \cup (R \circ S') \end{array}$$

Show by giving en example that analogous laws for $\cap$ do not hold.

Exercise 5. Let $\mathfrak{A}$ and $\mathfrak{B}$ be Ω–algebras for some signature Ω. Show that if $h : \mathfrak{A} \twoheadrightarrow \mathfrak{B}$ is a surjective homomorphism then $\mathfrak{B}$ is isomorphic to $\mathfrak{A}/\Theta$ with $x \ \Theta \ y$ if and only if $h(x) = h(y)$.

Exercise 6. Show that every Ω–algebra $\mathfrak{A}$ is the homomorphic image of a term algebra. *Hint.* Take X to be the set underlying $\mathfrak{A}$.

Exercise 7. Show that $\mathfrak{A} \times \mathfrak{B}$ is isomorphic to $\prod_{i \in I} \mathfrak{A}_i$, where $\mathfrak{A}_0 = \mathfrak{A}$, $\mathfrak{A}_1 = \mathfrak{B}$. Show also that $(\mathfrak{A} \times \mathfrak{B}) \times \mathfrak{C}$ is isomorphic to $\mathfrak{A} \times (\mathfrak{B} \times \mathfrak{C})$.

Exercise 8. Prove Proposition 1.1.5.

1.2 Semigroups and Strings

In formal language theory, languages are sets of strings over some alphabet. We assume throughout that an alphabet is a finite, nonempty set, usually called A. It has no further structure (but see Section 1.3), it only defines the material of primitive letters. We do not make any further assumptions on the size of A. The Latin alphabet consists of 26 letters, which actually exist in two variants (upper and lower case), and we also use a few punctuation marks and symbols as well as the blank. On the other hand, the Chinese 'alphabet' consists of several thousand letters!

Strings are very fundamental structures. Without a proper understanding of their workings one could not read this book, for example. A string over A is nothing but the result of successively placing elements of A after each other. It is not necessary to always use a fresh letter. If, for example, $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, then `abb`, `bac`, `caaba` are strings over A. We agree to use typewriter font to mark an actual symbol (piece of ink), while letters in different font are only proxy for letters (technically, they are variables for letters). Strings are denoted by a vector arrow, for example $\vec{w}$, $\vec{x}$, $\vec{y}$ and so on, to distinguish them from individual letters. Since paper is of bounded length, strings are not really written down in a continuous line, but rather in several lines, and on several pieces of paper, depending on need. The way a string is cut up into lines and pages is actually immaterial for its abstract constitution (unless we speak of paragraphs and similar textual divisions). We wish to abstract from these details. Therefore we define strings formally as follows.

Definition 1.2.1 *Let A be a set. A **string over** A is a function $\vec{x} : n \to A$ for some natural number n. n is called the **length of** $\vec{x}$ and is denoted by $|\vec{x}|$. $\vec{x}(i)$, $i < n$, is called the i**th segment** or the i**th letter of** $\vec{x}$. The unique string of length 0 is denoted by ε. If $\vec{x} : m \to A$ and $\vec{y} : n \to A$ are strings over A then $\vec{x} \cdot \vec{y}$ denotes* *the unique string of length $m+n$ for which the following holds:*

$$(\vec{x}\cdot\vec{y})(j) := \begin{cases} \vec{x}(j) & \text{if } j < m, \\ \vec{y}(j-m) & \text{else.} \end{cases}$$

*We often write $\vec{x}\vec{y}$ in place of $\vec{x}\cdot\vec{y}$. In connection with this definition the set A is called the **alphabet**, an element of A is also referred to as a **letter**. A is, unless otherwise stated, finite and not empty.*

So, a string may also be written using simple concatenation. Hence we have `abc`$\cdot$`baca` $=$ `abcbaca`. Note that there is no blank inserted between the two string; the blank is a *letter*. We denote it by $\Box$. Two words of a language are usually separated by a blank possibly using additional punctuation marks. That the blank is a symbol is felt more clearly when we use a typewriter. If we want to have a blank, we need to press down a key in order to get it. For purely formal reasons we have added the empty string to the set of strings. It is not visible (unlike the blank). Hence, we need a special symbol for it, which is ε, in some other books also λ. We have

$$\vec{x}\cdot\varepsilon = \varepsilon\cdot\vec{x} = \vec{x}\ .$$

We say, the empty string is the **neutral element** or **unit** with respect to concatenation. For any triple of strings $\vec{x}$, $\vec{y}$ and $\vec{z}$ we have

$$\vec{x}\cdot(\vec{y}\cdot\vec{z}) = (\vec{x}\cdot\vec{y})\cdot\vec{z}\ .$$

We therefore say that concatenation, $\cdot$, is **associative**. More on that later. We define the notation $\vec{x}^i$ by induction on i.

$$\begin{array}{lcl} \vec{x}^0 & := & \varepsilon\ , \\ \vec{x}^{i+1} & := & \vec{x}^i\cdot\vec{x}\ . \end{array}$$

Furthermore, we define $\prod_{i<n}\vec{x}_i$ inductively.

$$\prod_{i<0}\vec{x}_i := \varepsilon, \qquad \prod_{i<n+1} := (\prod_{i<n}\vec{x}_i)\cdot\vec{x}_n\ .$$

Note that the letter $\mathtt{a}$ is technically distinct from the string $\vec{x} : 1 \to A : 0 \mapsto \mathtt{a}$. They are nevertheless written in the same way, namely $\mathtt{a}$. If $\vec{x}$ is a string over A and $A \subseteq B$, then $\vec{x}$ is a string over B. The set of all strings over A is denoted by A^*.

Let $<$ be a linear order on A. We define the so called **lexicographical ordering (with respect to** $<$**)** as follows. Put $\vec{x} <_L \vec{y}$ if there exist $\vec{u}$, $\vec{v}$ and $\vec{w}$ as well as a and b such that $\vec{x} = \vec{u} \cdot a \cdot \vec{v}$, $\vec{y} = \vec{u} \cdot b \cdot \vec{w}$ and $a < b$. Notice that $\vec{x} <_L \vec{y}$ can obtain even if $\vec{x}$ is longer than $\vec{y}$. Another important ordering is the following one. Let $\mu(a) := k$ if a is the kth symbol of A in the ordering $<$. Further, put $n := |A|$. If $\vec{x} = x_0 x_1 \dots x_{p-1}$ we associate the following the number with $\vec{x}$.

$$Z(\vec{x}) := \sum_{i=0}^{p-1} (\mu(x_i) + 1)(n+1)^{p-i-1}$$

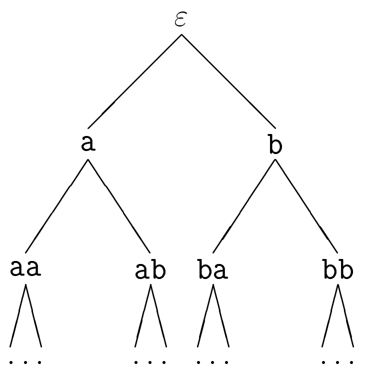
Now put $\vec{x} <_N \vec{y}$ if and only if $Z(\vec{x}) < Z(\vec{y})$. This ordering we call the **numerical ordering**. We emphasize that both orderings depend on the ordering $<$ chosen on A. We shall illustrate these orderings with an alphabet containing two letters, $A = \{\mathtt{a}, \mathtt{b}\}$. Let $\mathtt{a} < \mathtt{b}$. Then the numerical ordering is as follows.

$\vec{x}$	ε,	a,	b,	aa,	ab,	ba,	bb,	aaa,	aab,	aba, ...
$Z(\vec{x})$	0	1	2	4	5	7	8	13	14	16

This ordering is linear. The map sending $i \in \omega$ to the ith element in this element is known as the **dyadic representation** of the numbers. In the dyadic representation, 0 is represented by the empty string, 1 by $\mathtt{a}$, 2 by $\mathtt{b}$, 3 by $\mathtt{aa}$ and so on. (Actually, if one wants to avoid using the empty string here, one may start with $\mathtt{a}$ instead.)

The lexicographical ordering is somewhat more complex. We illustrate it for words with at most four letters.

Figure 1.1: The tree A^*



ε,	a,	aa,	aaa,	aaaa,	aaab,
aab,	aaba,	aabb,	ab,	aba,	abaa,
abab,	abb,	abba,	abbb,	b,	ba,
baa,	baaa,	bab,	baba,	babb,	bb,
bba,	bbaa,	bbab,	bbb,	bbba,	bbbb

In the lexicographical as well as the numerical ordering ε is the smallest element. Now look at the ordered tree based on the set A^*, which is similar to the tree domain based on the set $\{0, 1, \ldots, n-1\}^*$ to be discussed below. Then the lexicographical ordering corresponds to the linearization obtained by depth–first search in this tree, while the numerical ordering corresponds to the linearization obtained by breadth–first search (see Section 2.2).

A **monoid** is a triple $\mathfrak{M} = \langle M, 1, \circ \rangle$ where $\circ$ is a binary operation on M and 1 an element such that for all $x, y, z \in M$ the following holds.

$$\begin{array}{lcl} x \circ 1 & = & x \\ 1 \circ x & = & x \\ x \circ (y \circ z) & = & (x \circ y) \circ z \end{array}$$

A monoid is therefore an algebra with signature $\Omega : 1 \mapsto 0, \cdot \mapsto 2$, which in addition satisfies the above equations. An example is the algebra $\langle 4, 0, max\rangle$ (recall that $4 = \{0, 1, 2, 3\}$). Another very important example is the following.

Proposition 1.2.2 *Let* $\mathfrak{Z}(A) := \langle A^*, \varepsilon, \cdot\rangle$*. Then* $\mathfrak{Z}(A)$ *is a monoid.*

Subalgebras of a monoid $\mathfrak{M}$ are called **submonoids** of $\mathfrak{M}$. Submonoids of $\mathfrak{M}$ are uniquely determined by their underlying set, since the operations are derived by restriction from those of $\mathfrak{M}$. The function which assigns to each string its length is a homomorphism from $\mathfrak{Z}(A)$ onto the monoid $\langle\omega, 0, +\rangle$. It is surjective, since A is always assumed to be nonempty. A homomorphism $h : \mathfrak{Z}(A) \to \mathfrak{M}$ is already uniquely determined by its restriction on A. Moreover, any map $v : A \to M$ determines a unique homomorphism $\overline{v} : \mathfrak{Z}(A) \to \mathfrak{M}$.

Proposition 1.2.3 *The monoid* $\mathfrak{Z}(A)$ *is freely generated by* A*.*

Proof. Let $\mathfrak{N} = \langle N, 1, \circ\rangle$ be a monoid and $v : A \to N$ an arbitrary map. Then we define a map $\overline{v}$ as follows.

$$\begin{array}{lcl} \overline{v}(\varepsilon) & := & 1 \\ \overline{v}(a) & := & v(a) \\ \overline{v}(\vec{x} \cdot a) & := & \overline{v}(\vec{x}) \circ v(a) \end{array}$$

This map is surely well defined if in the last line we assume that $\vec{x} \neq \varepsilon$. For the defining clauses are mutually exclusive. Now we must show that this map is a homomorphism. To this end, let $\vec{x}$ and $\vec{y}$ be words. We shall show that

$$\overline{v}(\vec{x} \cdot \vec{y}) = \overline{v}(\vec{x}) \circ \overline{v}(\vec{y}) \ .$$

This will be established by induction on the length of $\vec{y}$. If it is 0, the claim is evidently true. For we have $\vec{y} = \varepsilon$, and hence $\overline{v}(\vec{x} \cdot \vec{y}) = \overline{v}(\vec{x}) = \overline{v}(\vec{x}) \circ 1 = \overline{v}(\vec{x}) \circ \overline{v}(\vec{y})$. Now let $|\vec{y}| > 0$. Then

$\vec{y} = \vec{w} \cdot a$ for some $a \in A$.

$$\begin{array}{rcll} \overline{v}(\vec{x} \cdot \vec{y}) & = & \overline{v}(\vec{x} \cdot \vec{w} \cdot a) & \\ & = & \overline{v}(\vec{x} \cdot \vec{w}) \circ v(a) & \text{by definition} \\ & = & (\overline{v}(\vec{x}) \circ \overline{v}(\vec{w})) \circ v(a) & \text{by induction hypothesis} \\ & = & \overline{v}(\vec{x}) \circ (\overline{v}(\vec{w}) \circ v(a)) & \text{since } \mathfrak{N} \text{ is a monoid} \\ & = & \overline{v}(\vec{x}) \circ \overline{v}(\vec{y}) & \text{by definition} \end{array}$$

This shows the claim. □

The set A is the only set that generates $\mathfrak{Z}(A)$ freely. For a letter cannot be produced from anything longer that a letter. The empty string is always dispensable, since it occurs anyway in the signature. Hence any generating set must contain A, and since A generates A^* it is the only minimal set that does so. A nonminimal generating set can never freely generate a monoid. For example, let $X = \{\mathtt{a}, \mathtt{b}, \mathtt{bba}\}$, then X generates $\mathfrak{Z}(A)$, but it is not minimal. Hence it does not generate $\mathfrak{Z}(A)$ freely. For example, let $v : \mathtt{a} \mapsto \mathtt{a}, \mathtt{b} \mapsto \mathtt{b}, \mathtt{bba} \mapsto \mathtt{a}$. Then there is no homomorphism that extends v to A^*. For then on the one hand $\overline{v}(\mathtt{bba}) = \mathtt{a}$, on the other $\overline{v}(\mathtt{bba}) = v(\mathtt{b}) \cdot v(\mathtt{b}) \cdot v(\mathtt{a}) = \mathtt{bba}$.

The fact that A generates $\mathfrak{Z}(A)$ freely has various noteworthy consequences. First, a homomorphism from $\mathfrak{Z}(A)$ into an arbitrary monoid need only be fixed on A in order to be defined. Moreover, *any* such map can be extended to a homomorphism into the target monoid. As a particular application we get that every map $v : A \to B^*$ can be extended to a homomorphism from $\mathfrak{Z}(A)$ to $\mathfrak{Z}(B)$. Furthermore, we get the following result, which shows that the monoids $\mathfrak{Z}(A)$ are up to isomorphism the only freely generated monoids (if at this point A is allowed to be infinite as well). They reader may note that the proof is completely general, it works in fact for algebras of any signature.

Theorem 1.2.4 *Let $\mathfrak{M} = \langle M, \circ, 1\rangle$ and $\mathfrak{N} = \langle N, \circ, 1\rangle$ be freely generated monoids. Then either of (a) and (b) obtains.*

(a) There is an injective homomorphism $i : \mathfrak{M} \rightarrowtail \mathfrak{N}$ and a surjective homomorphism $h : \mathfrak{N} \twoheadrightarrow \mathfrak{M}$ such that $h \circ i = 1_M$.

(b) There exists an injective homomorphism $i : \mathfrak{N} \rightarrowtail \mathfrak{M}$ *and a surjective homomorphism* $h : \mathfrak{M} \twoheadrightarrow \mathfrak{N}$ *such that* $h \circ i = 1_N$*.*

Proof. Let $\mathfrak{M}$ be freely generated by X, $\mathfrak{N}$ freely generated by Y. Then either $|X| \leq |Y|$ or $|Y| \leq |X|$. Without loss of generality we assume the first. Then there is an injective map $p : X \rightarrowtail Y$ and a surjective map $q : Y \twoheadrightarrow X$ such that $q \circ p = 1_X$. Since X generates $\mathfrak{M}$ freely, there is a homomorphism $\overline{p} : \mathfrak{M} \to \mathfrak{N}$ with $\overline{p} \restriction X = p$. Likewise, there is a homomorphism $\overline{q} : \mathfrak{N} \to \mathfrak{M}$ such that $\overline{q} \restriction Y = q$, since $\mathfrak{N}$ is freely generated by Y. The restriction of $\overline{q} \circ \overline{p}$ to X is the identity. (For if $x \in X$ then $\overline{q} \circ \overline{p}(x) = \overline{q}(p(x)) = q(p(x)) = x$.) Since, once again, X freely generates $\mathfrak{M}$, there is only one homomorphism which extends 1_X on $\mathfrak{M}$ and this is the identity. Hence $\overline{q} \circ \overline{p} = 1_M$. It immediately follows that $\overline{q}$ is surjective and $\overline{p}$ injective. Hence we are in case (a). Had $|Y| \leq |X|$ been the case, (b) would have obtained instead. □

Theorem 1.2.5 *In* $\mathfrak{Z}(A)$ *the following cancellation laws hold.*

1. *If* $\vec{x} \cdot \vec{u} = \vec{y} \cdot \vec{u}$*, then* $\vec{x} = \vec{y}$*.*
2. *If* $\vec{u} \cdot \vec{x} = \vec{u} \cdot \vec{y}$*, then* $\vec{x} = \vec{y}$*.*

$\vec{x}^T$ is defined as follows.

$$\left(\prod_{i<n} x_i\right)^T := \prod_{i<n} x_{n-1-i}$$

$\vec{x}^T$ is called the **mirror string** of $\vec{x}$. It is easy to see that $(\vec{x}^T)^T = \vec{x}$.

The reader is asked to convince himself that the map $\vec{x} \mapsto \vec{x}^T$ is *not* a homomorphism if $|A| > 1$.

Definition 1.2.6 *Let* $\vec{x}, \vec{y} \in A^*$*. Then* $\vec{x}$ *is a* ***prefix of*** $\vec{y}$ *if* $\vec{y} = \vec{x} \cdot \vec{u}$ *for some* $\vec{u} \in A^*$*.* $\vec{x}$ *is called a* ***postfix*** *or* ***suffix of*** $\vec{y}$ *if* $\vec{y} = \vec{u} \cdot \vec{x}$ *for some* $\vec{u} \in A^*$*.* $\vec{x}$ *is called a* ***substring of*** $\vec{y}$ *if* $\vec{y} = \vec{u} \cdot \vec{x} \cdot \vec{v}$ *for some* $\vec{u}, \vec{v} \in A^*$*.*

It is easy to see that $\vec{x}$ is a prefix of $\vec{y}$ exactly if $\vec{x}^T$ is a postfix of $\vec{y}^T$.

Notice that a given string can have several occurrences in another string. For example, `aa` occurs four times is `aaaaa`. The occurrences are in addition not always disjoint. An occurrence of $\vec{x}$ in $\vec{y}$ can be defined in several ways. We may for example assign *positions* to each letters. In a string $x_0 x_1 \ldots x_{n-1}$ the numbers $< n+1$ are called **positions**. The positions are actually thought of as the spaces between the letters. The ith letter, x_i, occurs between the position i and the position $i+1$. The substring $\prod_{i \leq j < k} x_i$ occurs between the positions i and k. The reason for doing it this way is that it allows us to define occurrences of the empty string as well. For each i, there is an occurrence of ε between position i and position i. We may interpret positions as time points in between which certain events take place, here the utterance of a given sound. Another definition of an occurrence is via the context in which the substring occurs.

Definition 1.2.7 *A* ***context*** *is a pair $C = \langle \vec{y}, \vec{z} \rangle$ of strings. A* ***substitution of*** $\vec{x}$ ***into*** C*, in symbols $C(\vec{x})$, is defined to be the string $\vec{y} \cdot \vec{x} \cdot \vec{z}$. We say that $\vec{x}$* ***occurs in*** $\vec{v}$ ***in the context*** C *if $\vec{v} = C(\vec{x})$. Every occurrence of $\vec{x}$ in a string $\vec{v}$ is uniquely defined by its context. We call C a* ***substring occurrence of*** $\vec{x}$ ***in*** $\vec{v}$.

Actually, given $\vec{x}$ and $\vec{v}$, only one half of the context would be enough. However, as shall become clear, contexts defined in this way allow for rather concise statements of facts in many cases. Now consider two substring occurrences C, D in a given word $\vec{z}$. Then there are various ways in which the substrings may be related with respect to each other.

Definition 1.2.8 *Let $C = \langle \vec{u}_1, \vec{u}_2 \rangle$ and $D = \langle \vec{v}_1, \vec{v}_2 \rangle$ be occurrences of the strings $\vec{x}$ and $\vec{y}$, respectively, in $\vec{z}$. We say that C* ***precedes*** *D if $\vec{u}_1 \vec{x}$ is a prefix of $\vec{v}_1$. C and D* ***overlap*** *if C does not precede D and D does not precede C. C* ***is contained in*** *D if $\vec{v}_1$ is a prefix of $\vec{u}_1$ and $\vec{v}_2$ is a suffix of $\vec{u}_2$.*

Notice that $\vec{x}$ can be a substring of $\vec{y}$ and every occurrence of $\vec{y}$ contains an occurrence of $\vec{x}$ but not every occurrence of $\vec{x}$ need be contained in an occurrence of $\vec{x}$.

Definition 1.2.9 *A* ***(string) language over the alphabet*** *A is a subset of A^*.*

This definition admits that $L = \varnothing$ and that $L = A^*$. Moreover, we may have $\varepsilon \in L$. The admission of ε is often done for technical reasons (like the introduction of a zero) and causes trouble from time to time (for example in the definition of grammars). Nevertheless, not admitting it will not ameliorate the situation, so we have opted for the streamlined definition here.

Theorem 1.2.10 *If A is not empty and countable, there are exactly $2^{\aleph_0}$ languages.*

This is folklore. For notice that $|A^*| = \aleph_0$. (This follows from the fact that we can enumerate A^*.) Hence, there are as many languages as there are subsets of $\aleph_0$, namely $2^{\aleph_0}$ (the size of the continuum, that is, the set of real numbers). One can prove this rather directly using the following result.

Theorem 1.2.11 *Let $C = \{c_i : i < p\}$, $p > 2$, be an arbitrary alphabet and $A = \{\mathtt{a}, \mathtt{b}\}$. Further, let $\overline{v}$ be the homomorphic extension of $v : c_i \mapsto \mathtt{a}^i \cdot \mathtt{b}$. The map $S \mapsto \overline{v}[S] : \wp(C^*) \to \wp(A^*)$ defined by $V(S) = \overline{v}[S]$ is a bijection between $\wp(C^*)$ and those languages which are contained in the direct image of $\overline{v}$.*

The proof is an exercise. The set of all languages over A is closed under $\cap$, $\cup$, and $-$, the relative complement with respect to A^*. Furthermore, we can define the following operations on languages.

$$\begin{array}{lcl}
L \cdot M & := & \{\vec{x} \cdot \vec{y} : \vec{x} \in L, \vec{y} \in M\} \\
L^0 & := & \{\varepsilon\} \\
L^{n+1} & := & L^n \cdot L \\
L^* & := & \bigcup_{n\in\omega} L^n \\
L^+ & := & \bigcup_{0<n\in\omega} L^n \\
L/M & := & \{\vec{y} \in A^* : (\exists \vec{x} \in M)(\vec{y} \cdot \vec{x} \in L)\} \\
M\backslash L & := & \{\vec{y} \in A^* : (\exists \vec{x} \in M)(\vec{x} \cdot \vec{y} \in L)\}
\end{array}$$

* is called the **Kleene star**. For example, L/A^* is the set of all strings which can be extended to members of L; this is exactly the set of prefixes of members of L. We call this set the **prefix closure** of L, in symbols L^P. Analogously, $L^S := A^*\backslash L$ is the **suffix** or **postfix closure** of L. It follows that $(L^P)^S$ is nothing but the substring closure of L.

Let L be a language over A, $C = \langle \vec{x}, \vec{y} \rangle$ a context and $\vec{u}$ a string. We say that C **accepts** $\vec{u}$ **in** L if $C(\vec{u}) \in L$. The triple $\langle A^*, A^* \times A^*, \dashv_L \rangle$, where $\dashv_L$ is the inverse of the acceptance relation, is a context in the sense of the previous section. Let $M \subseteq A^*$ and $P \subseteq A^* \times A^*$. Then denote by $C_L(M)$ the set of all C which accept all strings from M in L (intent); and denote by $Z_L(P)$ the set of all strings which are accepted by all contexts from P in L (extent). We call M (L–)**closed** if $M = Z_L(C_L(M))$. The closed sets form the so called **distribution classes** of strings in a language. $Z_L(C_L(M))$ is called the **Sestier–closure** of M and the map $S_L : M \mapsto Z_L(C_L(M))$ the **Sestier–operator**. From Proposition 1.1.17 we immediately get this result.

Proposition 1.2.12 *The Sestier–operator is a closure operator.*

For various reasons, identifying terms with strings that represent them is a dangerous affair. As is well–known, conventions for writing down terms can be misleading, they may lead to ambiguities. Hence we regard the term as an abstract entity (which we could define rigorously, of course), and treat the string only as a representative of that term.

Definition 1.2.13 *Let Ω be a signature. A* ***representation of terms (by means of strings over*** *A) is a relation $R \subseteq Tm_\Omega \times A^*$ such that for each term t there exists a string $\vec{x}$ with $\langle t, \vec{x} \rangle \in R$. $\vec{x}$ is called a* ***representative*** *or* ***representing string*** *of t* ***with respect to*** *R. $\vec{x}$ is called* ***unambiguous*** *if from $\langle t, \vec{x} \rangle, \langle u, \vec{x} \rangle \in R$ it follows that $t = u$. R is called* ***unique*** *or* ***uniquely readable*** *if every $\vec{x} \in A^*$ is unambiguous.*

R is uniquely readable if and only if it is an injective function from Tm_Ω to A^* (and therefore its converse a partial injective function). We leave it to the reader to verify that the representation defined in the previous section is actually uniquely readable. This is not self evident. It could be that a term possesses several representing strings. Our usual way of denoting terms is not necessarily unique. For example, one writes $2 + 3 + 4$ even though this could be a representative of the term $+(+(2,3),4)$ or of the term $+(2,+(3,4))$. The two terms do have the same value, but as terms they are different. This convention is useful, but it is not uniquely readable.

There are many more conventions for writing down terms. We give a few examples. (a) A binary symbol is typically written in between its arguments (this is called the **infix notation**). So, we do not write `+(2,3)` but `(2+3)`. (b) Outermost brackets may be omitted. `(2+3)` denotes the same term as `2+3`. (c) The multiplication sign binds stronger than `+`. So, the following strings all denote the same term.

`(2+(3·5))`   `2+(3·5)`   `(2+3·5)`   `2+3·5`

In logic we also use dots in place of brackets. The shorthand $p \wedge q. \to .p$ abbreviates $(p \wedge q) \to p$. The dots are placed to the left and right (sometimes just to the right) of the main operation sign.

Since the string `(2+3)·5` represents a different term than `2+3·5` (and both have a different value) the brackets are needed. That we can do without brackets is an insight we owe to the Polish logician Jan Łukasiewicz. In his notation, which is also called **Polish Notation** (**PN**), the function symbol is always placed in front of its arguments. Alternatively, the function symbol may be consistently placed behind its arguments (this is the so called **Reverse Polish Notation**, **RPN**). There are some calculators (in addition to the programming language FORTH) which have implemented RPN. In place of the (optional) brackets there is a key called '`enter`'. It is needed to separate two successive operands. For in RPN, the two arguments of a function follow each other immediately. If nothing is put in between them, both the terms $+(13,5)$ and

$+(1, 35)$ would both be written `135+`. To prevent this, `enter` is used to separate the first from the second input string. You therefore need to enter into the computer `13`$\boxed{\texttt{enter}}$`5+`. (Here, the box is the usual way in computer handbooks to turn a sequence into a 'key'. In Chapter 3 we shall deal extensively with the problem of writing down numbers.) Notice that the choice between Polish and Reverse Polish Notation only affects the position of the function symbol, not the way in which arguments are placed. For example, suppose there is a binary symbol `exp` to denote the exponential function. Then what is `2`$\boxed{\texttt{enter}}$`3`$\boxed{\texttt{exp}}$ in RPN is actually $\boxed{\texttt{exp}}$`2`$\boxed{\texttt{enter}}$`3=` in PN or 2^3 in ordinary notation. Hence, the relative order between base and exponent remains. This effect is also noted in natural languages: the subject precedes the object in the overwhelming majority of languages irrespective of the place of the verb. The mirror image of an VSO language is an SOV language, not OSV.

Now we shall show that the Polish Notation is uniquely readable. Let F be a set of symbols and Ω a signature over F, as defined in the previous section. Each symbol $f \in F$ is assigned an arity $\Omega(f)$. Next, we define a set of strings over F, which we assign to the various terms of Tm_Ω. PN_Ω is the smallest set M of strings over F for which the following holds.

> For all $f \in F$ and for all $\vec{x}_i \in M$, $i < \Omega(f)$, $f \cdot \vec{x}_0 \cdot \ldots \cdot \vec{x}_{\Omega(f)-1} \in M$.

(Notice the special case $n = 0$. Further, notice that no special treatment is needed for variables, by the remarks of the preceding section.) This defines the set PN_Ω, members of which are called **well–formed strings**. Next we shall define which string represents which term. The string 'f', $\Omega(f) = 0$, represents the term 'f'. If $\vec{x}_i$ represents the term t_i, $i < \Omega(f)$, then $f \cdot \vec{x}_0 \cdot \ldots \cdot \vec{x}_{\Omega(f)-1}$ represents the term $f(t_0, \ldots, t_{\Omega(f)-1})$. We shall now show that this relation, called Polish Notation, is bijective. (A different proof than the one used here can be found in Section 2.4, proof of Theorem 2.4.4.) Here we use an important principle, namely induction over the generation of the string. We shall prove inductively:

1. No proper prefix of a string is a well–formed string is a well–formed string.

2. If $\vec{x}$ is a well–formed string then $\vec{x}$ has length at least 1 and the following holds.

   (a) If $|\vec{x}| = 1$, then $\vec{x} = f$ for some $f \in F$ with $\Omega(f) = 0$.

   (b) If $|\vec{x}| > 1$, then there are f and $\vec{y}$ such that $\vec{x} = f \cdot \vec{y}$, and $\vec{y}$ is the concatenation of exactly $\Omega(f)$ many uniquely defined well–formed strings.

The proof is as follows. Let t and u be terms represented by $\vec{x}$. Let $|\vec{x}| = 1$. Then t and u are terms of the form a, $a \in X$ or $a \in F$ with $\Omega(a) = 0$. It is clear that $t = u$. A proper prefix is the empty string, which is clearly not well formed. Now for the induction step. Let $\vec{x}$ have length at least 2. Then there is an $f \in F$ and a sequence $\vec{y}_i$, $i < \Omega(f)$, of well–formed strings such that

$$(\ddagger) \qquad \vec{x} = f \cdot \vec{y}_0 \cdot \ldots \cdot \vec{y}_{\Omega(f)-1} \ .$$

There are therefore terms b_i, $i < \Omega(f)$, which are represented by $\vec{y}_i$. By inductive hypothesis, these terms are unique. Furthermore, the symbol f is unique. Now let $\vec{z}_i$, $i < \Omega(f)$, be well–formed strings with

$$\vec{x} = f \cdot \vec{z}_0 \cdot \ldots \cdot \vec{z}_{\Omega(f)-1} \ .$$

Then $\vec{y}_0 = \vec{z}_0$. For no proper prefix of $\vec{z}_0$ is a well–formed term, and no proper prefix of $\vec{y}_0$ is a term. But they are prefixes of each other, so they cannot be proper prefixes of each other, that is to say, they are equal. If $\Omega(f) = 1$, we are done. Otherwise we carry on in the same way, establishing by the same argument that $\vec{y}_1 = \vec{z}_1$, $\vec{y}_2 = \vec{z}_2$, and so on. The fragmentation of the string in $\Omega(f)$ many well–formed strings is therefore unique. By inductive hypothesis, the individual strings uniquely represent the terms b_i. So, $\vec{x}$ uniquely represents the term $f(b_0, \ldots, b_{\Omega(f)-1})$.

Finally, we shall establish that no proper prefix of $\vec{x}$ is a well–formed string. Look again at the decomposition (‡). If $\vec{u}$ is a well–formed prefix, then $\vec{u} \neq \varepsilon$. Hence $\vec{u} = f \cdot \vec{v}$ for some $\vec{v}$ which can be decomposed into $\Omega(f)$ many well–formed strings $\vec{w}_i$. As before we shall argue that $\vec{w}_i = \vec{x}_i$ for every $i < \Omega(f)$. Hence $\vec{u} = \vec{x}$, which shows that no proper prefix of $\vec{x}$ is well–formed.

Exercise 9. Prove Theorem 1.2.11.

Exercise 10. Put $Z^*(\vec{x}) := \sum_{i<p} \mu(x_i) n^{p-i-1}$. Now put $\vec{x} <_{N^*} \vec{y}$ if and only if $Z^*(\vec{x}) < Z^*(\vec{y})$. Show that $<_{N^*}$ is transitive and irreflexive, but not total.

Exercise 11. Show that the postfix relation is a partial ordering, likewise the prefix and the subword relation. Show that the subword relation is the transitive closure of the union of the postfix relation with the prefix relation.

Exercise 12. Let F, X and $\{(,)\}$ be three pairwise disjoint sets, Ω a signature over F. We define the following function from Ω–terms into strings over $F \cup X \cup \{(,)\}$:

$$\begin{array}{lcl} x^+ & := & x \\ f(t_0, \ldots, t_{\Omega(f)-1})^+ & := & f \cdot (\cdot t_0^+ \cdot \ldots \cdot t_{\Omega(f)-1}^+ \cdot) \end{array}$$

(To be clear: we represent terms by the string that we have used in Section 1.1 already.) Prove the unique readability of this notation. Notice that this does not already follow from the fact that we have chosen this notation to begin with. (We might just have been mistaken ...)

Exercise 13. Give an exact upper bound on the number of prefixes (postfixes) of a given string of length n, n a natural number. Also give a bound for the number of subwords. What can you say about the exactness of these bounds in individual cases?

Exercise 14. Let $L, M \subseteq A^*$. Define

$$\begin{array}{lcl} L//M & := & \{\vec{y} : (\forall \vec{x} \in M)(\vec{y} \cdot \vec{x} \in L)\} \\ M\backslash\backslash L & := & \{\vec{y} : (\forall \vec{x} \in M)(\vec{x} \cdot \vec{y} \in L)\} \end{array}$$

Show the following for all $L, M, N \subseteq A^*$:

$$M \subseteq L\backslash\backslash N \qquad \Leftrightarrow \qquad L \cdot M \subseteq N \qquad \Leftrightarrow \qquad L \subseteq N//M$$

Exercise 15. Show that not all equivalences are valid if in place of $\backslash\backslash$ and $//$ we had chosen $\backslash$ and $/$. Which implications remain valid?

1.3 Fundamentals of Linguistics

In this section we shall say some words about our conception of language and introduce some linguistic terminology. Since we cannot define all the linguistic terms, this section is more or less meant to fix the reader on our particular interpretation of them and to acquaint those readers with them who wish to read the book without going through an introduction into linguistics proper. (However, it is recommended to have such a book at hand.)

A central tool in linguistics is that of postulating abstract units and hierarchization. Language is thought to be more than a mere relation between sounds and meanings. In between the two realms we find a rather rich architecture that hardly exists in formal languages. This architecture is most clearly articulated in (Harris, 1963) and also (Lamb, 1966). Even though linguists might disagree with many details, the basic architecture is assumed even in most current linguistic theories. We shall outline what we think is basic consensus. Language is organized in four levels or layers, which are also called **strata**, see Figure 1.2: the phonological stratum, the morphological stratum, the syntactic stratum and the semantical stratum. Each stratum possesses elementary units and rules of combination. The phonological stratum and the morphological stratum are adjacent, the morphological stratum and the syntactic stratum are adjacent, and the syntactic stratum and the semantic stratum are adjacent. Adjacent strata are interconnected by so called **rules of realization**. On the **phonological**

Figure 1.2: The Strata of Language

Semantical Stratum

Syntactical Stratum

Morphological Stratum

Phonological Stratum

stratum we find the mere representation of the utterance in its phonetic and phonological form. The elementary units are the *phones.* An utterance is composed from phones (more or less) by concatenation. The terms phone, syllable, accent and tone refer to this stratum. In the **morphological stratum** we find the elementary signs of the language (see Section 3.1), which are called **morphs**. These are defined to be the smallest units that carry meaning, although the definition of 'smallest' may be difficult to give. They are different from words. The word `sees` is a word, but it is the combination of two morphs, the root `see` and the ending of the third person singular present, `s`. The units of the **syntactical stratum** are the words, also called **lexes**. The units of the **semantical stratum** are the **semes**.

On each stratum we distinguish concrete from abstract units. The abstract units are sets of concrete ones. The abstraction is done in such a way that the concrete member of each class that appears in a construction is defined by its context, and that substitution of another member results simply in a non well–formed unit (or else in a virtually identical one). This definition is deliberately vague; it is actually hard to make precise. The interested reader is referred to the excellent (Harris, 1963) for the ins and outs of the structural method. The abstract counterpart of a phone is a **phoneme**. A phoneme is simply a set of phones. The sounds of a single language are a subset of the entire space of human sounds, partitioned into phonemes. This is to say that two distinct phonemes of a languages are disjoint. We shall deal with the relationship between phones and phonemes in Section 6.3. We use the following notation. We enclose phonemes in slashes while square brackets are used to name phones. So, if [p] is a sound then /p/ is the phoneme containing [p]. (Clearly, there are infinitely many sounds that may be called [p], but we pick just one of them.) An index is used to make reference to the language, for phonemes are strictly language internal. It makes little sense to compare phonemes across languages. Languages cut up the sound continuum in a different way. For example, let [p] and [ph] be two distinct sounds, where [p] is the sound corresponding to the letter `p` in `spit`, [ph] the sound corresponding to the letter `p` in `put`. Sanscrit distinguishes these two sounds as instantiations of different phonemes: $/\mathrm{p}/_S \cap /\mathrm{ph}/_S = \varnothing$. English does not. So, $/\mathrm{p}/_E = /\mathrm{ph}/_E$. Moreover, the context determines whether what is written `p` is pronounced either as [p] or as [ph]. Actually, in English there is no context in which both will occur. Finally, French does not have the sound [ph]. We give another example. The combination of letters `ch` is pronounced in two noticeably distinct ways in German. After [i], it sounds like [ç], for example in `Licht` [lıçt], but after [a] it sounds like [x] as in `Nacht` [naxt]; the choice between these two variants is conditioned solely by the preceding vowel. It is therefore assumed that German does not possess two but one phonemes written `ch`, which is pronounced in these two ways depending on the context.

In the same way one assumes that German has only one plural **morpheme** even though there is a fair number of individual plural morphs. Table 1.1 shows some possibilities of forming the plural

Table 1.1: German Plural

singular	plural	
`Wagen`	`Wagen`	*car*
`Auto`	`Autos`	*car*
`Bus`	`Busse`	*bus*
`Licht`	`Lichter`	*light*
`Vater`	`Väter`	*father*
`Nacht`	`Nächte`	*night*

in German. The plural can be expressed either by no change, or by adding an `s`–suffix, an `e`–suffix (the reduplication of `s` in `Busse` is a phonological effect and needs no accounting for in the morphology), an `er`–suffix, or by Umlaut or a combination of Umlaut together with an `e`–suffix. (Umlaut is another name for the change of certain vowels when inflectional or derivational suffixes are added. In writing, Umlaut is the following change: `a` becomes `ä`, `o` becomes `ö`, and `u` becomes `ü`. All other vowels remain the same under Umlaut.) All these are clearly different morphs. But they belong to the same morpheme. We therefore call them **allomorphs** of the plural morpheme. The differentiation into strata allows to abstract away from disturbing irregularities. Moving up one stratum, the different members of an abstraction class are not distinguished. The different plural morphs for example, are defined as sequences of phonemes, not of phones. To decide which phone is be inserted is the job of the phonological stratum. Likewise, the word `Lichter` is 'known' to the syntactical stratum only as a plural nominative noun. That it consists of the root morph `Licht` together with the morph `-er` rather than any other plural morph is not visible in the syntactic stratum. The difference between concrete and abstract carries over in each stratum in the distinction between a **surface** and a **deep** sub–stratum. The morphotaxis has at deep level only the root `Licht` and the plural morpheme. At the surface, the latter gets realized as `-er`. The step from deep to surface can be quite complex. For example, the plural `Nächte` of `Nacht` is formed by umlauting the root vowel and adding the suffix `-e`. (The way the umlauted root is actually formed must be determined by the phonological stratum. For example, the plural of `Altar` (*altar*) is `Altäre` not `Ältare` or `Ältäre`!) As we have already said, on the so–called deep morphological (sub–)stratum we find only the combination of two morphemes, the morpheme `Nacht` and the plural morpheme. On the syntactical stratum (deep or surface) nothing of that decomposition is visible. We have one lex(eme), `Nächte`. On the phonological stratum we find a sequence of 5 (!) phonemes, which in writing correspond to `n`, `ä`, `ch`, `t` and `e`. This is the deep phonological representation. On the surface, we find the allophone [ç] for the phoneme (written as) `ch`.

In Section 3.1 we shall propose an approach to language by means of signs. This approach distinguishes only 3 strata: a sign has a *realization*, it has a combinatorics and it has a *meaning*. While the meaning is uniquely identifiable to belong to the semantic stratum, for the other two this is not clear. The combinatorics may be seen as belonging to the syntactical stratum. The realization of a sign, finally, could be spelled out either as a sequence of phonemes, as a sequence of morphemes or as a sequence of lexemes. Each of these choices is legitimate and yields interesting insights. However, notice that choosing sequences of morphemes or lexemes is somewhat incomplete since it further requires an additional algorithm that realizes these sequences in writing or speaking.

Language is not only spoken, it is also written. However, one must distinguish between letters and sounds. The difference between them is foremost a physical one. They use a different *channel*. A **channel** is a physical medium in which the message is manifested. Language manifests itself first and foremost acoustically, even though a lot of communication is done by writing. We principally learn a language by hearing and speaking it. Mastery of writing is achieved only after we are fully fluent. (Languages for the deaf form an exception that will not be dealt with here.) Each channel allows — by its mere physical properties — a different means of combination. A piece of paper is a two dimensional thing, and we are not forced to write down symbols linearly, as we are with acoustical signals. Think for example of the fact that Chinese characters are composite entities which contain parts in them. These are combined typically by juxtaposition, but characters are aligned vertically. Moreover, the graphical composition internally to a sign is of no relevance for the actual sound that goes with it. To take another example, Hindi is written in a syllabic script, which is called **Devanagari**. Each simple consonantal letter denotes a consonant plus `a`. Vowel letters may be added to these in case the vowel is different from `a`. (There are special characters for word initial vowels.) Finally, to denote consonantal clusters, the consonantal characters are melted into each other in a particular way. (There is only a finite number of consonantal clusters and the way the consonants are melted is fixed. The individual consonants are usually recognizable from the graphical complex. In typesetting there is a similar phenomenon known as **ligature**. The graphemes `f` and `i` melt into one when the first is before the second: 'fi'. Typewriters have no ligature for obvious reasons: `fi`.) Also in mathematics the possibilities of the graphical channel are widely used. We use indices, superscripts, subscripts, underlining, arrows and so on. Many diagrams are therefore not so easy to linearize. (For example, $\widehat{x}$ is spelled out as `x hat`, $\overline{x}$ as `x bar`.) Sign languages also make use of the three–dimensional space, which proves to require different perceptual skills than spoken language.

While the acoustic manifestation of language is in some sense essential for human language, its written manifestation is typically secondary, not only for the individual human being, as said above, but also from a cultural historic point of view. The sounds of the language and the pronunciation of words is something that comes into existence naturally, and they can hardly be fixed or determined arbitrarily. Attempts to stop language from changing are simply doomed to failure. Writing systems, on the other hand, are cultural products, and subject to sometimes severe regimentation. The effect is that writing systems show much greater variety across languages than sound systems. The number of primitive letters varies between some two dozen and a few thousand. This is so since some languages have letters for sounds (more or less) like Finnish (English is a moot point), others have letters for syllables (Devanagari) and yet others have letters for words (Chinese). It may be objected that in Chinese a character always stands for a syllable, but words may consist of several syllables, hence of several characters. Nevertheless, the difference with Devanagari is clear. The latter shows you how the word sounds like, the former does not, unless you know character by character how it is pronounced. If you were to introduce a new syllable into Chinese you would have create a new character, but not so in Devanagari. But all this has to be taken with care. Although French uses the Latin alphabet it becomes quite similar to Chinese. You may still know how to pronounce a word that you see written down, but from hearing it you are left in the dark as to how to spell it. For example, the following words are pronounced completely alike: `au`, `haut`, `eau`, `eaux`; similarly `vers`, `vert`, `verre`, `verres`.

In what is to follow, language will be written language. This is the current practice in such books as this one is but at least requires comment. We are using the so called Latin alphabet. It is used in almost all European countries, while each country typically uses a different set of symbols. The difference is slight, but needs accounting for (for example, when you wish to produce keyboards or design fonts). Finnish, Hungarian and German, for example, use `ä`, `ö` and `ü`. The letter `ß` is used in the German alphabet (but not in Switzerland). In French, one uses `ç`, also accents, and so on. The resource of single characters, which we call **letters**, is for the European languages somewhere between 60 and 100. We have besides each letter in upper and lower case letters also the punctuation marks and some extra symbols (not to forget the ubiquitous blank).

The counterpart of a letter in the spoken languages is the phoneme. Every language utterance can be analyzed into a sequence of phonemes (plus some residue about which we will speak briefly below). There is generally no biunique correspondence between phonemes and letters. The connection between the visible and the audible shape of language is everything but predictable or unique. English is a perfect example. There is hardly any letter that can unequivocally be related to a phoneme. For example, the letter `g` represents in many cases the phoneme [g] unless it is followed by `h`, in which case the two typically together represent a sound that can be zero (as in `sought` ([sɔ:t]), or `f` as in `laughter` ([la:ftə]). To add to the confusion, the letters represent different phones in different languages. (Note that it makes no sense to speak of the same *phoneme* in two different languages, as phonemes are abstractions that are formed within a single language.) The letter `u` has many different manifestations in English, German and French that are hardly compatible. This has prompted the invention of an international standard, the so called **International Phonetic Alphabet** (**IPA**, see (IPA, 1999)). Ideally, every sound of a given language can be uniquely transcribed into IPA such that anyone who is not acquainted with the language can reproduce the utterances correctly. The transcription of a word into this alphabet therefore changes whenever its sound manifestation changes, irrespective of the spelling norm.

The carriers of meaning are however not the sounds or letters (there is simply not enough of them); it is certain sequences thereof. Sequences of letters that are not separated by a blank or a punctuation mark other than '`-`' are called **words**. Words are units which can be analyzed further, for example into letters, but which for the most part we shall treat as units. This is the reason why the alphabet A in the technical sense will often *not* be the alphabet in the sense of 'stock of letters' but in the sense of 'stock of words'. However, since most languages have infinitely many words (due to compounding), and since the alphabet A must be finite, some care must be exercised in choosing the base.

We have analyzed words into sequences of letters or sounds, and sentences as sequences of words. This implies that sentences and words can always be so analyzed. This is what we shall assume throughout this book. The individual occurrences of sounds (letters) are called **segments**. For example, the letters `n`, `o`, and `t` are segments of `not`. The fact that words can be segmented is called **segmentability property**. At closer look it turns out that segmentability is an idealization. For example, a question differs from an assertion in its **intonation contour**. This contour may be defined as the rise and fall of the pitch. The contour shows distribution over the whole sentence but follows specific rules. It is again different in different languages. (Falling pitch at the end of a sentence, for example, may accompany questions in English, but not in German.) Because of its nature, intonation contour is called a **suprasegmental feature**. There are more, for example emphasis. Segmentability differs also with the channel. In writing, a question is marked by a segmental feature (the question mark), but emphasis is not. Emphasis is typically marked by underlining or italics. For example, if we want to emphasize the word 'board', we write <u>`board`</u> or *board*. As can be seen, every letter is underlined or set in italics, but underlining or italics is usually not something that is meant to emphasize those letters that are marked by it; rather, it marks emphasis of the entire word that is composed from them. We could have used a segmental symbol, just like quotes, but the fact of the matter is that we do not. Disregarding this, language typically is segmentable.

However, even if this is true, the conception that the morphemes of the language are sequences of letters is to a large extent fictitious. To give an extreme example, the plural is formed in Bahasa Indonesia by reduplicating the noun. For example, the word `anak` means 'child', the word `anak-anak` therefore means 'children', the word `orang` means 'man', and `orang-orang` means 'men'. Clearly, there is no sequence of letters or phonemes that can be literally said to constitute a plural morph. Rather, it is the function $f : A^* \to A^* : \vec{x} \mapsto \vec{x}\text{-}\vec{x}$, sending each string to its duplicate (with an interspersed hyphen). Actually, in writing the abbreviation `anak2` and `orang2` is commonplace. Here, `2` is a segmentable marker of plurality. However, notice that the words in the singular or the plural are each fully segmentable. Only the marker of plurality cannot be identified with any of the segments. This is to some degree also the case in German, where the rules are however much more complex, as we have seen above. The fact that morphs are (at closer look) not simply strings will be of central concern in this book.

Finally, we have to remark that letters and phonemes are not unstructured either. Phonemes consist of various so called **distinctive features**. These are features that distinguish the phonemes from each other. For example, [p] is distinct from [b] in that it is voiceless, while [b] is voiced. Other voiceless consonants are [k], [t], while [g] and [d] are once again voiced. Such features can be relevant for the description of language. There is a rule of German (and other languages, for example Russian) that forbids voiced consonants to occurs at the end of a syllable. For example, the word `Jagd` 'hunting' is pronounced [ˈja:kt], not [ˈja:gd]. This is so since [g] and [d] may not occurr at the end of the syllable, since they are voiced. Now, first of all, why do we not write `Jakt` then? This is so since inflection and derivation show that when these consonants occur non–finally in the syllable they are voiced: we have `Jagden` [ˈja:kden] 'huntings', with [d] now in fact being voiced, and also `jagen` [ˈja:gən] 'to hunt'. Second: why do we not propose that voiceless consonants become voiced when syllable initial? Because there is plenty of evidence that this does not happen. Both voiced and voiceless sounds may appear at the beginning of the syllable, and those ones that are analyzed as underlyingly voiceless remain so in whatever position. Third: why bother writing the underlying consonant rather than the one we hear? Well, first of all, since we know how to pronounce the word anyway, it does not matter whether we write [d] or [t]. On the other hand, if we know how to write the word, we also know a little bit about its morphological behaviour. What this comes down to is that to learn how to write a language is to learn how the language works. Now, once this is granted, we shall explain why we find [k] in place of [g] and [t] in place of [d]. This is because of the internal organisation of the phoneme. The phoneme is a set of distinctive features, one of which (in German) is [$\pm$voiced]. The rule is that when the voiced consonant may not occur, it is only the feature [+voiced] that is replaced by [$-$voiced]. Everything else remains the same. A similar situation is the relationship between upper and lower case letters. The rule says that a sentence may not begin with a lower case letter. So, when the sentence begins, the first letter is changed to its upper case counterpart if necessary. Hence, letters too contain distinctive features. Once again, we write the words in the dictionary always as if they would appear elsewhere. Notice by the way that although each letter is by itself an upper or a lower case letter, written language attributes the distinction upper versus lower case to the word not to the initial letter. Disregarding some modern spellings in advertisements (like in Germany `InterRegio`, `eBusiness` and so on) this is a reasonable strategy. However, it is nevertheless not illegitimate to call it a suprasegmental feature.

In the previous section we have talked extensively about representations of terms by means of strings. In linguistics this is a big issue, which appears typically under the name *word order*. Let us give an example. Disregarding word classes, each word of the language has one (or several) arities. The finite verb see has arity 2. The proper names Paul and Marcus on the other hand have arity 0. Any symbol of arity > 0 is called a **functor** with respect to its argument. In syntax one also speaks of **head** and **complement**. These are relative notions. In the term $\mathsf{see}(\mathsf{Marcus}, \mathsf{Paul})$, the functor or head is see, and its arguments are Paul and Marcus. To distinguish these arguments from each other, we use the terms *subject* and *object*. Marcus is the **subject** and Paul is the **object** of the sentence. The notions 'subject' and 'object' denote so called **grammatical relations**. The correlation between argument places and grammatical relations is to a large extent arbitrary, and is a central concern in syntactical theory. Notice also that not all arguments are complements. Here, syntactical theories diverge as to which of the arguments may be called 'complement'. In generative grammar, for example, it is assumed that only the direct object is a complement.

Now, how is a particular term represented? The representation of see is `sees`, that of Marcus is `Marcus` and that of Paul is `Paul`. The whole term (1.1) is represented by the string (1.2).

(1.1) see(Marcus, Paul)
(1.2) `Marcus sees Paul.`

So: the verb appears after the subject, which in turn precedes the object. At the end, a period is placed. However, to spell out the relationship between a language and a formal representation is not as easy as it appears at first sight. For first of all, we wish to let the term be something that does not depend on the particular language we choose and which gives us the full meaning of the term (so it is like a language of thought or an interlingua, if you wish). So the above term shall mean that Marcus sees Paul. We could translate the English sentence (1.2) by choosing a different representation language, but the choice between languages of representation should actually be immaterial as long as they serve the purpose. This is a very rudimentary picture but works well for our purposes. We shall return to the idea of producing sentences from terms in Chapter 3. Now look first at the representatives of the basic symbols in some other languages.

	see	Marcus	Paul
German	`sieht`	`Marcus`	`Paul`
Latin	`vidit`	`Marcus`	`Paulus`
Hungarian	`látja`	`Marcus`	`Pál`

Here is (1.1) given in these languages.

(1.3) `Marcus sieht Paul.`
(1.4) `Marcus Paulum vidit.`
(1.5) `Marcus látja Pált.`

English is called an SVO–language, since in transitive constructions the subject precedes the verb, and the verb in turn the object. This is exactly the infix notation. (However, notice that languages do not make use of brackets.) One uses the mnemonic symbols 'S', 'V' and 'O' to define the following basic 6 types of languages: SOV, SVO, VSO, OSV, OVS, VOS. These names tell us how the subject, verb and object follow each other in a basic transitive sentence. We call a language of type VSO or VOS **verb initial**, a language of type SOV or OSV **verb final** and a language of type SVO or OVS **verb medial**. By this definition, German is SVO, Hungarian too, hence both are verb medial and Latin is SOV, hence verb final. These types are not equally distributed. Depending on count 40 – 50 % of the world's languages are SOV languages, up to 40 % SVO languages and another 10 % are VSO languages. This means that in the vast majority of languages the order of the two arguments is: subject before object. This is why one does not generally emphasize the relative order of the subject with respect to the object. There is a bias against placing the verb initially (VSO), and a slight bias to put it finally (SOV) rather than medially (SVO).

One speaks of a **head final** (**head initial**) language if a head is consistently put at the end behind all of its arguments (at the beginning, before all the arguments). One denotes the type of order by XH (HX), X being the complement, H the head. Generally, a **head medial** language is not defined, particularly since most heads have at most one complement. It is defined that the direct object is the complement of the verb. Hence, word orders SVO and VOS are head initial, OVS and SOV head final. (The orders VSO and OSV are problematic since the verb is not adjacent to its object.) A verb is a head, however a very important one, since it basically builds the clause. Nevertheless, different heads may place their arguments differently, so a language that is verb initial need not be head initial, a language that is verb final need not be head final. Indeed, there are few languages that are consistently head initial (medial, final). Japanese is rather consistently head final, so is Turkish. Even a relative clause precedes the noun it modifies. Hungarian is a mixed case: adjectives precede nouns, there are no prepositions, only postpositions, but the verb tends to precede its object.

For the interested reader we give some more information on the languages shown above. First, Latin was initially an SOV language, however word order was not really fixed (see (Lehmann, 1993)). In fact, any of the six permutations of the sentence (1.4) is grammatical. Hungarian is more complex, again the word order shown in (1.5) is the least marked, but the rule is that discourse functions determine word order. German is another special case. Against all appearances there is all reason to believe that it is actually an SOV language. You can see this by noting first that only the carrier of inflection appears in second place, for example only the auxiliary if present. Second, in a subordinate clause all parts of the verb including the carrier of inflection are at the end.

(1.6) `Marcus sieht Paul.`
Marcus sees Paul.

(1.7) `Marcus will Paul sehen.`
Marcus wants to see Paul.

(1.8) `Marcus will Paul sehen können.`
Marcus wants to be able to see Paul.

(1.9) `..., weil Marcus Paul sieht.`
..., because Marcus sees Paul.

(1.10) `..., weil Marcus Paul sehen will.`
..., because Marcus wants to see Paul.

(1.11) `..., weil Marcus Paul sehen können will.`
..., because Marcus wants to be able to see Paul.

So, the main sentence is not always a good indicator of the word order. Some languages allow for alternative word orders, like Latin and Hungarian. This is not to say that all variants have the same meaning or significance; it is only that they are equal as representatives of (1.1). We therefore speak of Latin as having **free word order**. However, this only means that the head and the argument can assume any order with respect to each other, not that simply all permutations of the words means the same.

Now, notice that subject and object are coded by means of so called **cases**. In Latin, the object carries accusative case, so we find `Paulum` instead of `Paulus`. Likewise, in Hungarian we have `Pált` in place of `Pál`, the nominative. So, the way a representing string is arrived at is rather complex. We shall return again to case marking in Chapter 5.

Natural languages also display so called **polyvalency**. We say that a word is polyvalent if can have several arities (even with the same meaning). The verb `to roll` can be unary (= intransitive) as well as binary (transitive). This is not allowed in our definition of signature. However, it can easily be modified to account for polyvalent symbols.

Notes on this section. The rule that spells out the phoneme $/x/_G$ in German is more complex than the above explications show. For example, it is [x] in `fauchen` but [ç] in `Frauchen`. This may have two reasons: (a) There is a morpheme boundary between `u` and `ch` in the second word but not in the first. This morpheme boundary induces the difference. (b) The morpheme `chen`, although containing the phoneme $/x/_G$ will always realize this phoneme as [ç]. The difference between (a) and (b) is that while (a) defines a realization rule that uses only the phonological representation, (b) uses morphological information to define the realization. Mel'čuk defines the realization rules as follows. In each stratum, there are rules that define how deep representations get mapped to surface representations. Across strata, going down, the surface representations of the higher stratum get mapped into abstract representations of the lower stratum. (For example, a sequence of morphemes is first realized as a sequence of morphs and then spelled out as a sequence of phonemes, until, finally, it gets mapped onto a sequence of phones.) Of course, one may also reverse the process. However, adjacency between (sub-)strate remains as defined.

Exercise 16. Show that in Polish Notation, unique readability is lost when there exist polyvalent function symbols.

Exercise 17. Show that if you have brackets, unique readability is guaranteed even if you have polyvalency.

Exercise 18. We have argued that German is a verb final language. But is it strictly head final? Examine the data given in this section as well the data given below.

(1.12) `Josef pflückt eine schöne Rose für Maria.`
Josef is–plucking a beautiful rose for Mary

(1.13) `Heinrich ist dicker als Josef.`
Heinrich is fatter than Josef

Knowing that long ago the ancestor of German (and English) was consistently head final, can you give an explanation for these facts?

Exercise 19. Even if languages do not have brackets, there are elements that indicate clearly the left or right periphery of a constituent. Such elements are the determiners `the` and `a(n)`. Can you name more? Are there elements in English indicating the right periphery of a constituent? How about demonstratives like `this` or `that`?

Exercise 20. By the definitions, `Unix` is head initial. For example, the command `lpr` precedes its unique argument. Now study the way in which optional arguments are encoded. (If you are sitting behind a computer on which `Unix` runs, type `man lpr` and you get a synopsis of the command and it syntax.) Does it guarantee unique readability? (For the more linguistic minded reader: which type of marking strategy does `Unix` employ? Which natural language corresponds to it?)

1.4 Trees

Strings can also be defined as pairs $\langle \mathfrak{L}, \ell \rangle$ where $\mathfrak{L} = \langle L, < \rangle$ is a finite linearly ordered set and $\ell : L \to A$ a function, called the **labelling function**. Since L is finite we have $\langle L, < \rangle \cong \langle n, \in \rangle$ for

$n := |L|$. (Recall that n is defined to be a set and is linearly ordered by $\in$.) Replacing $\langle L, <\rangle$ by the isomorphic $\langle n, \in\rangle$, and eliminating the redundant $\in$, a string is a pair $\langle n, \ell\rangle$, where n is a natural number. In what is to follow, we will very often have to deal with extensions of relational structures (over a given signature Ξ) by a labelling function. They have the general form $\langle M, \mathfrak{I}, \ell\rangle$, where M is a set, $\mathfrak{I}$ an interpretation and ℓ a function from M to A. These structures shall be called **structures over** A or A–**structures**.

A very important notion in the analysis of language is that of a *tree*. A tree is a special case of a directed graph. A **directed graph** is a structure $\langle G, <\rangle$, where $< \subseteq G^2$ is a binary relation. As is common usage, we shall write $x \leq y$ if $x < y$ or $x = y$. Also, x and y are called **comparable** if $x \leq y$ or $y \leq x$. A (**directed**) **chain of length** k is a sequence $\langle x_i : i < k+1\rangle$ such that $x_i < x_{i+1}$ for all $i < k$. An **undirected chain of length** k is a sequence $\langle x_i : i < k + 1\rangle$ where $x_i < x_{i+1}$ or $x_{i+1} < x_i$ for all $i < k$. A directed graph is called **connected** if for every two elements x and y there is an undirected chain from x to y. A directed chain of length k is called **cycle of length** k, if $x_k = x_0$. A binary relation is called **cycle free** if it only has cycles of length 0. A **root** is an element r such that for every x $x <^* r$, where $<^*$ is the reflexive, transitive closure of $<$.

Definition 1.4.1 *A* ***directed acyclic transitive graph*** *(a* ***dag****) is a pair $\mathfrak{G} = \langle G, <\rangle$ such that $< \subseteq G^2$ is an acyclic transitive relation on G.*

Definition 1.4.2 *$\mathfrak{G} = \langle G, <\rangle$ is called a* ***forest*** *if $<$ is transitive and irreflexive and if $x < y, z$ then y and z are comparable. A forest with a root is called a* ***tree****.*

In a connected dag a root is comparable with every other element since the relation is transitive. Furthermore, in presence of transitivity $<$ is cycle free if and only if it is irreflexive. For if $<$ is reflexive it has a cycle of length 1. Conversely, if there is a cycle

$\langle x_i : i < k+1 \rangle$ of length $k > 0$, we immediately have $x_0 < x_k = x_0$, by transitivity.

If $x < y$ and there is no z such that $x < z < y$, x is called a **daughter of** y, and y the **mother of** x. We write $x \prec y$ to say that x is a daughter node of y. The following is easy to see.

Lemma 1.4.3 *Let $\langle T, < \rangle$ be a finite tree. If $x < y$ then there exists a $\widehat{x}$ such that $x \leq \widehat{x} \prec y$ and a $\widehat{y}$ such that $x \prec \widehat{y} \leq y$. $\widehat{x}$ and $\widehat{y}$ are uniquely determined by x and y.* $\square$

In infinite trees this need not hold. We define $x \circ y$ by $x \leq y$ or $y \leq x$ and say that x and y **overlap**. The following is easy to show.

Lemma 1.4.4 (Predecessor Lemma) *Let $\mathfrak{T}$ be a finite tree and x and y nodes which do not overlap. Then there exist uniquely determined u, v and w, such that $x \leq u \prec w$, $y \leq v \prec w$ and $v \neq u$.*

A node **branches** n**–times downwards** if it has exactly n daughters; and it branches n–**times upwards** if it has exactly n mothers. We say that a node **branches upwards** (**downwards**) if it branches upwards or downwards at least 2 times. A finite forest is characterized by the fact that it is transitive, irreflexive and no node branches upwards. Therefore, in connection with trees and forests we shall speak of 'branching' when we mean 'downward branching'. x is called a **leaf** if there is no $y < x$, that is, if x branches 0 times. The set of leaves of $\mathfrak{G}$ is denoted by $b(\mathfrak{G})$.

Further, we define the following notation.

$$\begin{array}{rcl} \downarrow x & := & \{y : y \leq x\}\ , \\ \uparrow x & := & \{y : y \geq x\}\ . \end{array}$$

By definition of a forest, $\uparrow x$ is linearly ordered by $<$. Also, $\downarrow x$ together with the restriction of $<$ to $\downarrow x$ is a tree.

A set $P \subseteq G$ is called a **path** if it is linearly ordered by $<$ and convex, that is to say, if $x, y \in P$ then $z \in P$ for every z such that

$x < z < y$. The **length** of P is defined to be $|P| - 1$. A **branch** is a maximal path with respect to set inclusion. The **height of** x in a dag, in symbols $h_{\mathfrak{G}}(x)$ or simply $h(x)$, is the maximal length of a branch in $\downarrow x$. It is defined inductively as follows.

$$\begin{array}{lcll} h(x) & := & 0 & \text{if } x \text{ is a leaf,} \\ h(x) & := & 1 + max\{h(y) : y \prec x\} & \text{otherwise.} \end{array}$$

Dually we define the **depth** in a dag.

$$\begin{array}{lcll} d(x) & := & 0 & \text{if } x \text{ is a root,} \\ d(x) & := & 1 + max\{d(y) : y \succ x\} & \text{otherwise.} \end{array}$$

For the entire dag $\mathfrak{G}$ we set

$$h(\mathfrak{G}) := \{h(x) : x \in T\} ,$$

and call this the **height of** $\mathfrak{G}$. (This is an ordinal, as is easily verified.)

Definition 1.4.5 *Let $\mathfrak{G} = \langle G, <_G \rangle$ and $\mathfrak{H} = \langle H, <_H \rangle$ be directed graphs and $G \subseteq H$. Then $\mathfrak{G}$ is called a **subgraph of** $\mathfrak{H}$ if $<_G = <_H \cap G^2$.*

If $\mathfrak{G}$ and $\mathfrak{H}$ are dags, forests or trees, then $\mathfrak{G}$ is a **sub–dag**, **subforest** and **subtree**, respectively. A subtree of $\mathfrak{H}$ with underlying set $\downarrow x$ is called a **constituent of** $\mathfrak{H}$.

Definition 1.4.6 *Let A be an alphabet. A **dag over** A (or an A–**dag**) is a pair $\langle \mathfrak{G}, \ell \rangle$ such that $\mathfrak{G} = \langle G, < \rangle$ is a dag and $\ell : G \to A$ an arbitrary function.*

Alternatively, we speak of **dags with labels in** A, or simply of labelled dags if it is clear which alphabet is meant. Similarly with trees. The notions of substructures are extended analogously.

The tree structure in linguistic representations encodes the hierarchical relations between elements and not their spatial or temporal relationship. The latter have to be added explicitly. This is done by extending the signature by another binary relation symbol, $\sqsubset$. We say that x is **before** y and that y is **after** x if $x \sqsubset y$ is the case. We say that x **dominates** y if $x > y$. The relation $\sqsubset$ articulates the temporal relationship between the segments. This is first of all defined on the leaves, and it is a linear ordering. (This reflects the insistance on segmentability. It will have to be abandoned once we do not assume segmentability.) Each node x in the tree has the physical span of its segments. This allows to define an ordering between the hierarchically higher elements as well. We simply stipulate that $x \sqsubset y$ if and only if all leaves below x are before all leaves below y. This is not unproblematic if nodes can branch upwards, but this situation we shall rarely encounter in this book. The following is an intrinsic definition of these structures.

Definition 1.4.7 *An* ***ordered tree*** *is a triple* $\langle T, <, \sqsubset\rangle$ *such that the following holds.*

(ot1) $\langle T, <\rangle$ *is a tree.*

(ot2) $\sqsubset$ *is a linear, strict ordering on the leaves of* $\langle T, <\rangle$.

(ot3) If $x \sqsubset z$ *and* $y < x$ *then also* $y \sqsubset z$.
If $x \sqsubset z$ *and* $y < z$ *then also* $x \sqsubset y$.

(ot4) If x *is not a leaf and for all* $y < x$ $y \sqsubset z$ *then also* $x \sqsubset z$.
If z *is not a leaf and for all* $y < z$ $x \sqsubset y$ *then also* $x \sqsubset z$.

The condition (ot2) requires that the ordering is coherent with the ordering on the leaves. It ensures that $x \sqsubset y$ only if all leaves below x are before all leaves below y. (ot3) is a completeness condition ensuring that if the latter holds, then indeed $x \sqsubset y$.

We agree on the following notation. Let $x \in G$ be a node. Then put $[x] := \downarrow x \cap b(\mathfrak{G})$. We call this the **extension of** x. $[x]$ is linearly ordered by $\sqsubset$. We write $k(x) := \langle [x], \sqsubset\rangle$ and call this the **associated string of** x. It may happen that two nodes have the same associated string. The string associated with the entire tree is $k(\mathfrak{G}) := \langle b(\mathfrak{G}), \sqsubset\rangle$. A constituent is called **continuous** if the associated string is convex with respect to $\sqsubset$. A set M is **convex (with respect to** $\sqsubset$) if for all $x, y, z \in M$: if $x \sqsubset z \sqsubset y$ then $z \in M$ as well.

For sets M, N of leaves put $M \sqsubset N$ if and only if for all for all $x \in M$ and all $y \in N$ we have $x \sqsubset y$. From (ot4) and (ot3) we derive the following:

$$(\dagger) \quad x \sqsubset y \qquad \Leftrightarrow \qquad [x] \sqsubset [y]$$

This property shows that the orderings on the leaves alone determines the relation $\sqsubset$ uniquely.

Theorem 1.4.8 *Let $\langle T, <\rangle$ be a tree and $\sqsubset$ a linear ordering on its leaves. Then there exists exactly one relation $\sqsubset' \supseteq \sqsubset$ such that $\langle T, <, \sqsubset'\rangle$ is an ordered tree.*

We emphasize that the ordering $\sqsubset'$ cannot be linear if the tree has more than one element. It may even happen that $\sqsubset' = \sqsubset$. One can show that overlapping nodes can never be comparable with respect to $\sqsubset$. For let $x \circ y$, say $x \leq y$. Let $u \leq x$ be a leaf. Assume $x \sqsubset y$; then by (ot3) $u \sqsubset y$ as well as $u \sqsubset u$. This contradicts the condition that $\sqsubset$ is irreflexive. Likewise $y \sqsubset x$ cannot hold. So, nodes can only be comparable if they do not overlap. We now ask: is it possible that they are comparable exactly when they do not overlap? In this case we call $\sqsubset$ **exhaustive**. Theorem 1.4.9 gives a criterion on the existence of exhaustive orderings. Notice that if M and N are convex sets, then so is $M \cap N$. Moreover, if $M \cap N = \varnothing$ then either $M \sqsubset N$ or $N \sqsubset M$. Also, M is convex if and only if for all u: $u \sqsubset M$ or $M \sqsubset u$.

Theorem 1.4.9 *Let $\langle T, <\rangle$ be a tree and $\sqsubset$ a linear ordering on the leaves. There exists an exhaustive extension of $\sqsubset$ if and only if all constituents are continuous.*

Proof. By Theorem 1.4.8 there exists a unique extension, $\sqsubset'$. Assume that all constituents are continuous. Let x and y are nonoverlapping nodes. Then $[x] \cap [y] = \varnothing$. Hence $[x] \sqsubset [y]$ or $[y] \sqsubset [x]$. since both sets are convex. So, by (†) we have $x \sqsubset' y$ or $y \sqsubset' x$. The ordering is exhaustive. Conversely, assume that $\sqsubset'$ is exhaustive. Pick x. We show that $[x]$ is convex. Let u be a leaf and $u \notin [x]$. Then u does not overlap with x. By hypothesis, $u \sqsubset' x$ or $x \sqsubset' u$, whence $[u] \sqsubset [x]$ or $[x] \sqsubset [u]$, by (†). This means nothing but that either $u \sqsubset y$ for all $y \in [x]$ or $y \sqsubset u$ for all $y \in [x]$. Hence $[x]$ is convex. □

Lemma 1.4.10 (Constituent Lemma) *Assume $\langle T, <, \sqsubset, \ell \rangle$ is an exhaustively ordered A–tree. Furthermore, let $p < q$. Then there is a context $C = \langle \vec{u}, \vec{v} \rangle$ such that*

$$k(q) = C(k(p)) = \vec{u} \cdot k(p) \cdot \vec{v} \, .$$

The converse does not hold. Furthermore, it may happen that $C = \langle \varepsilon, \varepsilon \rangle$ — in which case $k(q) = k(p)$ — without $q < p$.

Proposition 1.4.11 *Let $\langle T, <, \sqsubset \rangle$ be an ordered tree. For every x, x is 1–branching if and only if $[x] = [y]$ for some $y < x$.*

Proof. Let x be a 1–branching node with daughter y. Then we have $[x] = [y]$ but $x \neq y$. So, the condition is necessary. Let us that is sufficient. Let x be minimally 2–branching. Let $u < x$. There is a daughter $z \prec x$ such that $u \leq z$, and there is $z' \prec x$ different from z. Then $[u] \subseteq [z] \subseteq [x]$ as well as $[z'] \subseteq [x]$. All sets are nonempty and $[z'] \cap [z] = \varnothing$. Hence $[z] \subsetneq [x]$ and so also $[u] \subsetneq [x]$. □

We now say that a tree is **properly branching** if it has no 1–branching nodes.

There is a slightly different method of defining trees. Let T be a set and $\prec$ a cycle free relation on T such that for every x there is at most one y such that $x \prec y$. And let there be exactly one x which has no $\prec$–successor (the root). Then put $< := \prec^+$. $\langle T, < \rangle$ is a tree. And $x \prec y$ if and only if x is the daughter of y. Let $D(x)$ be the set of daughters of x. Now let P be a relation such that (a) $y \; P \; z$ only if y and z are sisters, (b) P^+, the transitive closure of

P, is a relation that linearly orders $D(x)$ for every x, (c) for every y there is at most one z such that $y \; P \; z$ and at most one z' such that $z' \; P \; y$. Then put $x \sqsubset y$ if and only if there is z such that (a) $x < \widehat{x} \prec z$ for some $\widehat{x}$, (b) $y < \widehat{y} \prec y$ for some $\widehat{y}$, (c) $\widehat{x} \; P^+ \; \widehat{y}$. $\prec$ und P are the immediate neighbourhood relations in the tree.

Proposition 1.4.12 *Let $\langle T, <, \sqsubset \rangle$ be an exhaustively ordered tree. Then $x \sqsubset y$ if and only if there are $x' \geq x$ and $y' \geq y$ which are sisters and $x' \sqsubset y'$.*

Finally we mention a further useful concept, that of a constituent structure.

Definition 1.4.13 *Let M be a set. A* ***constituent structure over*** *M is a system $\mathfrak{C}$ of subsets of M with the following properties.*

(cs1) $\{x\} \in \mathfrak{C}$ for every $x \in M$,

(cs2) $\varnothing \notin \mathfrak{C}$, $M \in \mathfrak{C}$,

(cs3) if $S, T \in \mathfrak{C}$ and $S \nsubseteq T$ as well as $T \nsubseteq S$ then $S \cap T = \varnothing$.

Proposition 1.4.14 *Let M be a nonempty set. There is a biunique correspondence between finite constituent structures over M and finite properly branching trees whose set of leaves is $\{\{x\} : x \in M\}$.*

Proof. Let $\langle M, \mathfrak{C} \rangle$ be a constituent structure. Then $\langle \mathfrak{C}, \subsetneq \rangle$ is a tree. To see this, one has to check that $\subsetneq$ is irreflexive and transitive and that it has a root. This is easy. Further, assume that $S \subsetneq T, U$. Then $U \cap T \supseteq S \neq \varnothing$, because of condition (cs2). Moreover, because of (cs3) we must have $U \subseteq T$ or $T \subseteq U$. This means nothing else than that T and U are comparable. The set of leaves is exactly the set $\{\{x\} : x \in M\}$. Conversely, let $\mathfrak{T} = \langle T, < \rangle$ be a properly branching tree. Put $M := b(\mathfrak{T})$ and $\mathfrak{C} := \{[x] : x \in T\}$. We claim that this is a constituent structure. For (cs1), we have that for every $u \in b(\mathfrak{G})$, $[u] = \{u\} \in \mathfrak{C}$. Further, for every x $[x] \neq \varnothing$, since the tree is finite. There is a root r of $\mathfrak{T}$, and we have $[r] = M$. This shows (cs2). Now we show (cs3). Assume that $[x] \nsubseteq [y]$ and $[y] \nsubseteq [x]$. Then x and y are incomparable (and different). Let u be a leaf and $u \in [x]$, then we have $u \leq x$. $u \leq y$ cannot hold since $\uparrow u$ is linear, and then x and y would be comparable. Likewise we see that from $u \leq y$ we get $u \nleq x$. Hence $[x] \cap [y] = \varnothing$. □

In general we can assign to every tree a constituent structure, but only if the tree is properly branching it can be properly reconstructed from this structure. The notion of a constituent structure can be extended to the notion of an ordered constituent structure, and we can introduce labellings. The terminology is completely as expected.

We shall now discuss the representation of terms by means of trees. There are two different methods, both widely used. Before we begin, we shall introduce the notion of a tree domain.

Definition 1.4.15 *Let $T \subseteq \omega^*$ be a set of finite sequences of natural numbers. T is called a **tree domain** if the following holds.*

(td1) If $\vec{x} \cdot i \in T$ then $\vec{x} \in T$.

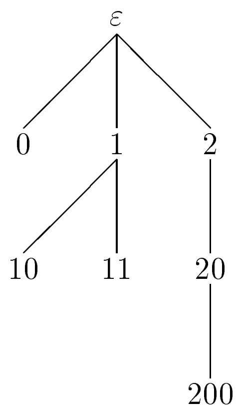
(td2) If $\vec{x} \cdot i \in T$ and $j < i$ then also $\vec{x} \cdot j \in T$.

We assign to a tree domain T an ordered tree in the following way. The set of nodes is T, (1) $\vec{x} < \vec{y}$ if and only if $\vec{y}$ is a proper prefix of $\vec{x}$ and (2) $\vec{x} \sqsubset \vec{y}$ if and only if there are numbers i, j and sequences $\vec{u}$, $\vec{v}$, $\vec{w}$ such that (a) $i < j$ and (b) $\vec{x} = \vec{u} \cdot i \cdot \vec{v}$, $\vec{y} = \vec{u} \cdot j \cdot \vec{w}$. Together with these relations, T is an exhaustively ordered finite tree, as is easily seen. Figure 1.3 shows the tree domain $T = \{\varepsilon, 0, 1, 2, 10, 11, 20, 200\}$. If T is a tree domain and $\vec{x} \in T$ then put

$$T/\vec{x} := \{\vec{y} : \vec{x} \cdot \vec{y} \in T\} .$$

This is the constituent below $\vec{x}$. (To be exact, it is not identical to this constituent, it is merely isomorphic to it. The (unique) isomorphism from $T/\vec{x}$ onto the constituent $\downarrow\vec{x}$ is the map $\vec{y} \mapsto \vec{x} \cdot \vec{y}$.)

Figure 1.3: A Tree Domain



Conversely, let $\langle T, <, \sqsubset \rangle$ be an exhaustively ordered tree. We define a tree domain T^β by induction on the depth of the nodes. If $t(x) = 0$, let $x^\beta := \varepsilon$. In this case x is the root of the tree. If x^β is defined, and y a daughter of x, then put $y^\beta := x^\beta \cdot i$, if y is the ith daughter of x counting from the left (starting, as usual, with 0). (Hence we have $|x^\beta| = t(x)$.) We can see quite easily that the so defined set is a tree domain. For we have $\vec{u} \in T^\beta$ as soon as $\vec{u} \cdot j \in T^\beta$ for some j. Hence (td1) holds. Further, if $\vec{u} \cdot i \in T^\beta$, say $\vec{u} \cdot i = y^\beta$ then y is the ith daughter of a node x. Take $j < i$. Then let z be the jth daughter of x (counting from the left). It exists, and we have $z^\beta = \vec{u} \cdot j$. Now we are not yet done. For we wish to show that the relations defined on the tree domain are exactly the ones that are defined on the tree. To put it differently, we wish to show that the map $x \mapsto x^\beta$ is an isomorphism. Obviously, it is enough if this map preserves the relations *is a daughter of* and *is left sister of*. However, the following is straightforward.

Theorem 1.4.16 *Let $\mathfrak{T} = \langle T, <, \sqsubset \rangle$ be a finite, exhaustively or-*

dered tree. The map $x \mapsto x^\beta$ is an isomorphism from $\mathfrak{T}$ onto the associated tree domain $\langle \mathfrak{T}^\beta, <, \sqsubset \rangle$. Furthermore, $\mathfrak{T} \cong \mathfrak{U}$ obtains if and only if $\mathfrak{T}^\beta = \mathfrak{U}^\beta$.

Terms can be translated into labelled tree domains. Each term t is assigned a tree domain t^b and a labelling function t^λ. The labelled tree domain associated with t is $t^m := \langle t^b, t^\lambda \rangle$. We start with the variables. $x^b := \{\varepsilon\}$, and $t^\lambda : \varepsilon \mapsto x$. Assume that the labelled tree domains t_i^m, $i < n - 1$, are defined, and put $n := \Omega(f)$. Let $s := f(t_0, \ldots, t_{n-1})$; then

$$s^b := \{\varepsilon\} \cup \bigcup_{i<n} \{i \cdot \vec{x} : \vec{x} \in t_i^b\} .$$

Then s^λ is defined as follows.

$$\begin{array}{lcl} s^\lambda(\varepsilon) & := & f \ , \\ s^\lambda(j \cdot \vec{x}) & := & t_j^\lambda(\vec{x}) \ . \end{array}$$

This means that s^m consists of a root named f which has n daughters, to which the labelled tree domains of t_0, ..., $t_{\Omega(f)-1}$ are isomorphic. We call the representation which sends t to t^m the **dependency coding**. This coding is more efficient that the following, which we call **structural coding**. We choose a new symbol, T, and define by induction to each term t a tree domain t^c and a labelling function t^μ. Put $x^c := \{\varepsilon, 0\}$, $x^\mu(\varepsilon) := \mathtt{T}$, $x^\mu(0) := x$. Further let for $s = f(t_0, \ldots, t_{n-1})$

$$\begin{array}{lcl} s^c & := & \{\varepsilon, 0\} \cup \bigcup_{0<i<n+1} \{i \cdot \vec{x} : \vec{x} \in t_i^c\} \ , \\ & & \\ s^\mu(\varepsilon) & := & \mathtt{T} \ , \\ s^\mu(0) & := & f \ , \\ s^\mu((j+1) \cdot \vec{x}) & := & t_j^\mu(\vec{x}) \ . \end{array}$$

(Compare the structural coding with the associated string in the notation without brackets.) In Figure 1.4 both codings are shown for the term `(3 + (5` $\times$ `7))` for comparison. The advantage of the structural coding is that the string associated to the labelled tree domain is also the string associated to the term (with brackets dropped, as the tree encodes the structure anyway).

Notes on this section. A variant of the dependency coding of syntactic structures has been proposed by Lucien Tesnière in (1982). He called tree representations *stemmata* (sg. *stemma*). See (**?**) for a survey. Unfortunately, the stemmata do not coincide with the dependency trees defined here, and this creates very subtle problems, see Mel'čuk op. cit. Noam Chomsky on the other hand proposed the more elaborate structural coding, which is by now widespread in linguistic theory.

Exercise 21. Define 'exhaustive ordering' on constituent structures. Show that a linear ordering on the leaves is extensible to an exhaustive ordering in a tree if and only if it is in the related constituent structure.

Exercise 22. Let $\mathfrak{T} = \langle T, < \rangle$ be a tree and $\sqsubset$ a binary relation such that $x \sqsubset y$ only if x, y are daughters of the same node (that is, they are sisters). Further, the daughter nodes of a given node shall be ordered linearly by $\sqsubset$. No other relations shall hold. Show that this ordering can be extended to an exhaustive ordering on $\mathfrak{T}$.

Exercise 23. Show that the number of binary branching exhaustively ordered tree on a given set is exactly

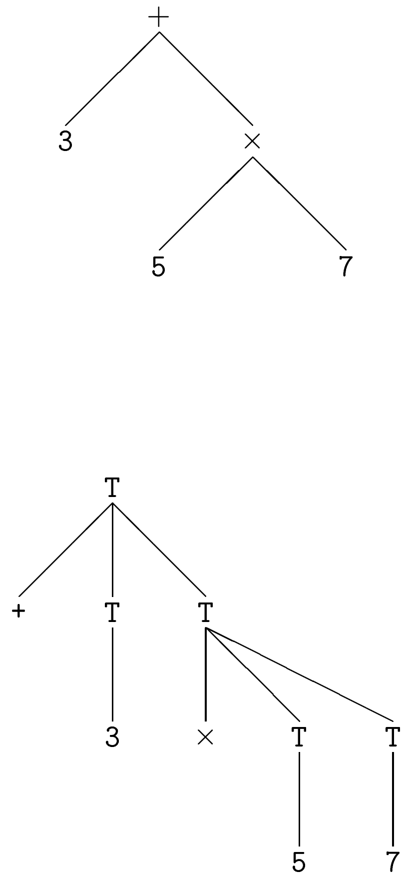
$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

These numbers are called **Catalan numbers**.

Exercise 24. Show that $C_n < \frac{1}{n+1}4^n$. (One can prove that $\binom{2n}{n}$ approximates the series $\frac{4^n}{\sqrt{\pi n}}$ in the limit. The latter even majorises the former. For the exercise there is an elementary proof.)

Exercise 25. Let L be finite with n elements and $<$ a linear ordering on L. Construct an isomorphism from $\langle L, < \rangle$ onto $\langle n, \in \rangle$.

Figure 1.4: Dependency coding and structural coding



1.5 Rewriting Systems

Languages are by Definition 1.2.9 arbitrary sets of strings over a (finite) alphabet. Nevertheless, languages that interest us here are those sets which can be described by finite means, particularly by finite processes. These can be processes which generate strings directly or by means of some intermediate structure (for example, labelled trees). The most popular approach is by means of rewrite systems on strings.

Definition 1.5.1 *Let A be a set. A* ***semi Thue system*** *over A is a finite set $T = \{\langle \vec{x}_i, \vec{y}_i \rangle : i < m\}$ of pairs of A–strings. If T is given, write $\vec{u} \Rightarrow^1_T \vec{v}$ if there are $\vec{s}, \vec{t}$ and some $i < m$ such that $\vec{u} = \vec{s} \cdot \vec{x}_i \cdot \vec{t}$ and $\vec{v} = \vec{s} \cdot \vec{y}_i \cdot \vec{t}$. We write $\vec{u} \Rightarrow^{n+1}_T \vec{v}$ if there is a $\vec{z}$ such that $\vec{u} \Rightarrow^1_T \vec{z} \Rightarrow^n_T \vec{v}$. Finally, we write $\vec{u} \Rightarrow^*_T \vec{v}$ if $\vec{u} \Rightarrow^n_T \vec{v}$ for some $n \in \omega$, and we say that $\vec{v}$ is* ***derivable in*** *T* ***from*** *$\vec{u}$.*

We can define $\Rightarrow^1_T$ also as follows. $\vec{u} \Rightarrow^1_T \vec{v}$ if and only if there exists a context C and a $\langle \vec{x}, \vec{y} \rangle \in T$ such that $\vec{u} = C(\vec{x})$ and $\vec{y} = C(\vec{y})$. Since the semi Thue system is finite it contains only finitely many letters. So, choosing A to be finite is no restriction. (See the exercises.) A semi Thue system T is called a **Thue system** if from $\langle \vec{x}, \vec{y} \rangle \in T$ follows $\langle \vec{y}, \vec{x} \rangle \in T$. In this case $\vec{v}$ is derivable from $\vec{u}$ if and only if $\vec{u}$ is derivable from $\vec{v}$. A **derivation of $\vec{y}$ from $\vec{x}$ in** T is a finite sequence $\langle \vec{v}_i : i < n+1 \rangle$ such that $\vec{v}_0 = \vec{x}$, $\vec{v}_n = \vec{y}$ and for all $i < n$ we have $\vec{v}_i \Rightarrow^1_T \vec{v}_{i+1}$. The **length** of this derivation is n. Sometimes we shall also allow the derivation to contain triples $\langle \vec{\alpha}, \vec{\alpha} \rangle \rangle$ even if there is no corresponding rule. This is done tacitly, but the reader hopefully has no problem in identifying those cases where the definition has been relaxed.

A *grammar* differs from a semi Thue system as follows. First, we introduce a distinction between the alphabet proper and an auxiliary alphabet, and secondly, the language is defined by means of a special symbol, the so called *start symbol.*

Definition 1.5.2 *A* ***grammar*** *is a quadruple $G = \langle S, N, A, R \rangle$ such that N, A are nonempty disjoint sets $S \in N$ and R a semi*

Thue system over $N \cup A$ such that $\langle \vec{\gamma}, \vec{\eta} \rangle \in R$ only if $\vec{\gamma} \notin A^*$. We call S the ***start symbol****, N the* ***nonterminal alphabet****, A the* ***terminal alphabet*** *and R the* ***set of rules****.*

Elements of the set N are also called **categories**. Notice that often the word 'type' is used instead of 'category', but this usage is dangerous for us in view of the fact that 'type' is reserved here for types in the λ–calculus. As a rule, we choose $S = \mathtt{S}$. This is not necessary. The reader is warned that $\mathtt{S}$ need not always be the start symbol. But if nothing else is said it is. As is common practice, nonterminals are denoted by upper case letters, terminals by lower case Roman letters. A lower case Greek letter signifies a letter that is either terminal or nonterminal. The use of vector arrows follows the practice established for strings. We write $G \vdash \vec{\gamma}$ or $\vdash_G \vec{\gamma}$ in case that $S \Rightarrow_R^* \vec{\gamma}$ and say that G **generates** $\vec{\gamma}$. Furthermore, we write $\vec{\gamma} \vdash_G \vec{\eta}$ if $\vec{\gamma} \Rightarrow_R^* \vec{\eta}$. The language generated by G is defined by

$$L(G) := \{\vec{x} \in A^* : G \vdash \vec{x}\} .$$

Notice that G generates strings which may contain terminal as well as nonterminal symbols. However, those that contain also nonterminals do not belong to the language that G generates. A grammar is therefore a semi Thue system in which it is defined how a derivation begins and how it ends.

Given a grammar G we call the **analysis problem** (or **parsing problem**) for G the problem (1) to say for a given string whether it is derivable in G and (2) to name a derivation in case that a string is derivable. The problem (1) alone is called the **recognition problem for** G.

A rule $\langle \vec{\alpha}, \vec{\beta} \rangle$ is often also called a **production** and is alternatively written as $\vec{\alpha} \to \vec{\beta}$. We call $\vec{\alpha}$ simply the **left hand side** and $\vec{\beta}$ the **right hand side** of the production. The **productivity** $p(\rho)$ of a rule $\rho = \vec{\alpha} \to \vec{\beta}$ is the difference $|\vec{\beta}| - |\vec{\alpha}|$. ρ is called **expanding** if $p(\rho) \geq 0$, **strictly expanding** if $p(\rho) > 0$ and **contracting** if $p(\rho) < 0$. A rule is **terminal** if it has the form $\vec{\alpha} \to \vec{x}$ (so, by our convention, $\vec{x} \in A^*$).

This notion of grammar is very general. There are only countably many grammars over a given alphabet — and hence only countably many languages generated by them —; nevertheless, the variety of these languages is bewildering. We shall see that every recursively enumerable language can be generated by some grammar. So, some more restricted notion of grammar is called for. Noam Chomsky has proposed the following hierarchy of grammar types. (Here, X_ε is short for $X \cup \{\varepsilon\}$.)

* A grammar is in the general case of **Type 0**.
* A grammar is said to be of **Type 1** or **context sensitive** if all rules are of the form $\vec{\delta}_1 X \vec{\eta}_2 \to \vec{\eta}_1 \vec{\alpha} \vec{\eta}_2$ and either (i) always $\vec{\alpha} \neq \varepsilon$ or (ii) $\mathtt{S} \to \varepsilon$ is a rule and $\mathtt{S}$ never occurs on the right hand side of a production.
* A grammar is said to be of **Type 2** or **context free** if it is context sensitive and all productions are of the form $X \to \vec{\alpha}$.
* A grammar is said to be of **Type 3** or **regular** if it is context free and all productions are of the form $X \to \vec{\alpha}$ where $\vec{\alpha} \in A_\varepsilon \cdot N_\varepsilon$.

A context sensitive rule $\vec{\eta}_1 X \vec{\eta}_2 \to \vec{\eta}_1 \vec{\alpha} \vec{\eta}_2$ is also written

$$X \to \vec{\alpha} / \vec{\eta}_1 \_\_ \vec{\eta}_2$$

One says that X can be rewritten into $\vec{\alpha}$ **in the context** $\vec{\eta}_1 \_\_ \vec{\eta}_2$. A *language* is said to be of **Type** i if it can be generated by a grammar of Type i. It is not relevant if there also exists a grammar of Type j, $j \neq i$, that generates this language in order for it to be of Type i.

We shall give examples of grammars of Type 3, 2 and 0.

Example 1. There are regular grammars which generate number expressions. Here a number expression is either a number, with or without sign, or a pair of numbers separated by a dot, again with or without sign. The grammar is as follows. The set of terminal symbols is $\{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}, \mathtt{+}, \mathtt{-}, \mathtt{.}\}$, the set of nonterminals is $\{\mathtt{V}, \mathtt{Z}, \mathtt{F}, \mathtt{K}, \mathtt{M}\}$. The start symbol is `V` and the productions are

$$\begin{array}{lcl}
\mathtt{V} & \to & \mathtt{+\,Z} \mid \mathtt{-\,Z} \mid \mathtt{Z} \\
\mathtt{Z} & \to & \mathtt{0\,Z} \mid \mathtt{1\,Z} \mid \mathtt{2\,Z} \mid \ldots \mid \mathtt{9\,Z} \mid \mathtt{F} \\
\mathtt{F} & \to & \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \ldots \mid \mathtt{9} \mid \mathtt{K} \\
\mathtt{K} & \to & \mathtt{.\,M} \\
\mathtt{M} & \to & \mathtt{0\,M} \mid \mathtt{1\,M} \mid \mathtt{2\,M} \mid \ldots \mid \mathtt{9\,M} \mid \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \ldots \mid \mathtt{9}
\end{array}$$

Here, we have used the following convention. The symbol '|' on the right hand side of a production indicates that the part on the left of this sign and the one to the right are alternatives. So, using the symbol '|' saves us from writing two rules expanding the same symbol. For example, `V` can be expanded either by `+Z`, `-Z` or by `Z`. The syntax of the language `Algol` has been written down in this notation, which became to be known as the **Backus–Naur Form**. The arrow was written '::='.

Example 2. The set of strings representing terms over a finite signature with finite set X of variables can be generated by a context free grammar. Let $F = \{\mathtt{F}_i : i < m\}$ and $\Omega(i) := \Omega(\mathtt{F}_i)$.

$$\mathtt{T} \quad \to \quad \mathtt{F}_i\mathtt{T}^{\Omega(i)} \qquad (i < m)$$

Since the set of rules is finite, so must be F. The start symbol is `T`. This grammar generates the associated strings in Polish Notation. Notice that this grammar reflects exactly the structural coding of the terms. More on that later. If we want to have dependency coding, we have to choose instead the following grammar.

$$\begin{array}{lcl}
\mathtt{S} & \to & \mathtt{F}_{j_0}\mathtt{F}_{j_1}\ldots\mathtt{F}_{j_{\Omega(i)-1}} \\
\mathtt{F}_i & \to & \mathtt{F}_{j_0}\mathtt{F}_{j_1}\ldots\mathtt{F}_{j_{\Omega(i)-1}} \,.
\end{array}$$

This is a scheme of productions. Notice that for technical reasons the root symbol must be `S`. We could dispense with the first kind of rules if we are allowed to have several start symbols. We shall return to this issue below.

EXAMPLE 3. Our example for a Type 0 grammar is the following, taken from (Salomaa, 1978).

$$\begin{array}{llcllcl}
(a) & \mathtt{X}_0 & \to & \mathtt{a}, & \mathtt{X}_0 & \to & \mathtt{aXX}_2\mathtt{Z}, \\
(b) & \mathtt{X}_2\mathtt{Z} & \to & \mathtt{aa}, & & & \\
(c) & \mathtt{Xa} & \to & \mathtt{aa}, & \mathtt{Ya} & \to & \mathtt{aa}, \\
(d) & \mathtt{X}_2\mathtt{Z} & \to & \mathtt{Y}_1\mathtt{YXZ}, & & & \\
(e) & \mathtt{XX}_1 & \to & \mathtt{X}_1\mathtt{YX}, & \mathtt{YX}_1 & \to & \mathtt{Y}_1\mathtt{YX}, \\
(f) & \mathtt{XY}_1 & \to & \mathtt{X}_1\mathtt{Y}, & \mathtt{YY}_1 & \to & \mathtt{Y}_1\mathtt{Y}, \\
(g) & \mathtt{aX}_1 & \to & \mathtt{aXXYX}_2, & & & \\
(h) & \mathtt{X}_2\mathtt{Y} & \to & \mathtt{XY}_2, & \mathtt{Y}_2\mathtt{Y} & \to & \mathtt{YY}_2, \\
 & & & & \mathtt{Y}_2\mathtt{X} & \to & \mathtt{YX}_2\,.
\end{array}$$

$\mathtt{X}_0$ is the start symbol. This grammar generates the language $\{\mathtt{a}^{n^2} : n > 0\}$. This can be seen as follows. To start, with (a) one can either generate the string $\mathtt{a}$ or the string $\mathtt{aXX}_2\mathtt{Z}$. Let $\vec{\gamma}_i = \mathtt{aX}\vec{\delta}_i\mathtt{X}_2\mathtt{Z}$, $\vec{\delta}_i \in \{\mathtt{X},\mathtt{Y}\}^*$. We consider derivations which go from $\vec{\gamma}_i$ to a terminal string. At the beginning, only (b) or (d) can be applied. Let it be (b). Then we can only continue with (c) and then we create a string of length $1 + 3 + |\vec{\gamma}_i|$. Since we have only one letter, the string is uniquely determined. Now assume that (d) has been chosen. Then we get the string $\mathtt{aX}\vec{\gamma}_i\mathtt{Y}_1\mathtt{YXZ}$. The only possibility to continue is (e) and (f). This moves the index 1 one place to the left. This procedure results in the placement of an occurrence of $\mathtt{Y}$ before every occurrence of an $\mathtt{X}$. We get a new string $\mathtt{aXXYX}_2\vec{\delta}_i'\mathtt{YYXZ}$. Now there is no other choice but to move the index 2 to the right with the help of (h). This gives a string $\vec{\gamma}_{i+1} = \mathtt{aX}\vec{\delta}_{i+1}\mathtt{X}_2\mathtt{Z}$ with $\vec{\delta}_{i+1} = \mathtt{XYX}\vec{\delta}_i'\mathtt{YY}$. We have

$$|\vec{\delta}_{i+1}| = |\vec{\delta}_i| + \ell_x(\vec{\delta}_i) + 5$$

where $\ell_x(\delta_i)$ counts the number of $\mathtt{X}$ in $\vec{\delta}_i$. Since $\ell_x(\vec{\delta}_{i+1}) = \ell_x(\vec{\delta}_i) + 2$, we conclude that $\ell_x(\vec{\delta}_i) = 2i$ and so $|\vec{\delta}_i| = (i+1)^2 - 4$, $i > 0$.

In the definition of a context sensitive grammar the following must be remembered. By intention, context sensitive grammars only consist of expanding rules. However, since we must begin with a start symbol, there would be no way to derive the empty string if no rule is contracting. Hence, we do admit the rule $\mathtt{S} \to \varepsilon$. But in order not to let other contracting uses of this rule creep in we require that $\mathtt{S}$ is not on the right hand side of any rule whatsoever. Hence, $\mathtt{S} \to \varepsilon$ can only be applied once, at the beginning of the derivation. The derivation immediately terminates. This condition is also in force for context free and regular grammars although we can show that it is technically not needed (see the exercises). For assume that in a grammar G with rules of the form $X \to \vec{\alpha}$ there are rules where S occurs on the right hand side of a production, and nevertheless replace S by Z in all rules which are not not of the form $S \to \varepsilon$. Add also all rules $S \to \vec{\alpha}'$, where $S \to \vec{\alpha}$ is a rule of G and $\vec{\alpha}'$ results from $\vec{\alpha}$ by replacing S by Z. This is a context free grammar which generates the same language, and even the same structures. (The only difference is with the nodes labelled S or Z.)

The class of regular grammars is denoted by RG, the class of all context free grammars by CFG, the class of context sensitive grammars by CSG and the class of Type 0 grammars by GG. The languages generated by these grammars is analogously denoted by RL, CFL, CSL and GL. The grammar classes form a proper hierarchy.

$$\mathrm{RG} \subsetneq \mathrm{CFG} \subsetneq \mathrm{CSG} \subsetneq \mathrm{GG}$$

This is not hard to see. It follows immediately that the languages generated by these grammar types also form a similar hierarchy. However, that the inclusions are likewise strict is not clear a priori. We claim that the following also holds.

$$\mathrm{RL} \subsetneq \mathrm{CFL} \subsetneq \mathrm{CSL} \subsetneq \mathrm{GL}$$

We shall prove each of the proper inclusions. In Section 1.7 (Theorem 1.7.14) we shall show that there are languages of Type 0 which are not of Type 1. Furthermore, from the Pumping Lemma (Theorem 1.6.13) for context free languages it follows that $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$ is not context free. However, it is context sensitive (which is left as an exercise in that section). Also, by Theorem 1.5.9 below, the language $\{\mathtt{a}^{n^2} : n \in \omega\}$ has a grammar of Type 1. However, this language is not semilinear, whence it is not of Type 2 (see Section 2.5). Finally, it will be shown that $\{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$ is context free but not regular. (See Exercise 2.1.) We should note that many definitions of context free grammars allow that they also contain rules of the form $X \to \varepsilon$. In the exercises the reader is asked to show that this definition is not more general with respect to the set of languages. However, contracting productions are actually not so pleasant to work with and therefore we have eliminated them from the definitions. Moreover, without this condition the grammar types would not form a hierarchy any more.

In order to speak about equivalence of grammars we very often have to speak about derivations. We shall propose the following terminology. We call a triple $A = \langle \vec{\alpha}, C, \vec{\zeta} \rangle$ an **instance of** ρ if C is an occurrence of $\vec{\gamma}$ in $\vec{\alpha}$ and also an occurrence of $\vec{\eta}$ in $\vec{\zeta}$. This means in particular that there exist $\vec{\kappa}_1$ and $\vec{\kappa}_2$ such that $C = \langle \vec{\kappa}_1, \vec{\kappa}_2 \rangle$ and $\vec{\alpha} = \vec{\kappa}_1 \cdot \vec{\gamma} \cdot \vec{\kappa}_2$ as well as $\vec{\zeta} = \vec{\kappa}_1 \cdot \vec{\eta} \cdot \vec{\kappa}_2$. We call C the **domain of** A. A **derivation of length** n is a sequence $\langle A_i : i < n \rangle$ of *instances* of rules from G such that $A_i = \langle \vec{\alpha}_i, C_i, \vec{\zeta}_i \rangle$ for $i < n$ and for every $j < n - 1$ $\vec{\alpha}_{j+1} = \vec{\zeta}_j$. $\vec{\alpha}_0$ is called the **start** of the derivation, $\vec{\zeta}_{n-1}$ the **end**. We denote by $\mathbf{der}(G, \vec{\alpha})$ the set of derivations G from the string $\vec{\alpha}$ and $\mathbf{der}(G) := \mathbf{der}(G, S)$. Let $\rho = \vec{\gamma} \to \vec{\eta}$.

This definition has been carefully chosen. Let $\langle A_i : i < n \rangle$, $A_i = \langle \vec{\alpha}_i, C_i, \vec{\alpha}_{i+1} \rangle$ $(i < n)$ be a derivation in G. Then we call $\langle \vec{\alpha}_i : i < n + 1 \rangle$ the (**associated**) **string sequence**. Notice that the string sequence is longer by one than the derivation. In what is to follow we shall often call the string sequence a derivation. However, this is not quite legitimate, since the string sequence does not determine the derivation uniquely. Here is an example.

Let G be the following grammar.

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{AB} \\ \mathtt{A} & \to & \mathtt{AA} \\ \mathtt{B} & \to & \mathtt{AB} \end{array}$$

Take the string sequence $\langle \mathtt{S}, \mathtt{AB}, \mathtt{AAB} \rangle$. There are two derivations corresponding to this sequence.

$$\begin{array}{ll} (a) & \langle \langle \mathtt{S}, \langle \varepsilon, \varepsilon \rangle, \mathtt{AB} \rangle, \langle \mathtt{AB}, \langle \varepsilon, \mathtt{B} \rangle, \mathtt{AAB} \rangle \rangle \\ (b) & \langle \langle \mathtt{S}, \langle \varepsilon, \varepsilon \rangle, \ mbox\mathtt{AB} \rangle, \langle \mathtt{AB}, \langle \mathtt{A}, \varepsilon \rangle, \mathtt{AAB} \rangle \rangle \end{array}$$

After application of a rule ρ, the left hand side $\vec{\gamma}$ is replaced by the right hand side, but the context parts $\vec{\kappa}_1$ and $\vec{\kappa}_2$ remain as before. It is intuitively clear that if we apply a rule to parts of the context, then this application could be permuted with the first. This is clarified in the following definition and theorem.

Definition 1.5.3 *Let $\langle \vec{\alpha}, \langle \vec{\kappa}_1, \vec{\kappa}_2 \rangle, \vec{\beta} \rangle$ be an instance of $\rho = \vec{\eta} \to \vec{\vartheta}$, and $\langle \vec{\beta}, \langle \vec{\mu}_1, \vec{\mu}_2 \rangle, \vec{\gamma} \rangle$ an instance of $\sigma = \vec{\zeta} \to \vec{\xi}$. We call the domains of these applications* ***disjoint*** *if either (a) $\vec{\kappa}_1 \cdot \vec{\vartheta}$ is a prefix of $\vec{\mu}_1$ or (b) $\vec{\vartheta} \cdot \vec{\kappa}_2$ is a suffix of $\vec{\mu}_2$.*

Lemma 1.5.4 (Commuting Instances) *Let $\langle \vec{\alpha}, \langle \vec{\kappa}_1, \vec{\kappa}_2 \rangle, \vec{\beta} \rangle$ be an instance of $\rho = \vec{\eta} \to \vec{\vartheta}$, and $\langle \vec{\beta}, \langle \vec{\mu}_1, \vec{\mu}_2 \rangle, \vec{\gamma} \rangle$ an instance of $\sigma = \vec{\zeta} \to \vec{\xi}$. Suppose that the instances are disjoint. Then there exists an instance $\langle \vec{\alpha}, C, \vec{\delta} \rangle$ of σ as well as an instance $\langle \vec{\delta}, D, \vec{\gamma} \rangle$ of ρ, and both have disjoint domains.*

The proof is easy and left as an exercise. Analogously, suppose that to the *same* string the rule ρ and the rule σ can be applied, although with disjoint domains. Then after applying one of them the domains remain disjoint, and the other can still be applied.

We give an example where this fails. Let the following rules be given.

$$\begin{array}{lclcllcl} \mathtt{AX} & \to & \mathtt{XA} & \qquad & \mathtt{XA} & \to & \mathtt{Xa} \\ \mathtt{XB} & \to & \mathtt{Xb} & \qquad & \mathtt{Xa} & \to & \mathtt{a} \end{array}$$

There are two instances of the rules applicable to $\mathtt{AXB}$. The first has domain $\langle \varepsilon, \mathtt{B}\rangle$, the second the domain $\langle \mathtt{A}, \varepsilon\rangle$. The domains overlap and indeed the first rule when applied destroys the domain of the second. However, if we apply the rule $\mathtt{AX} \to \mathtt{XA}$ we can reach a terminal string.

$$\mathtt{AXB} \Rightarrow \mathtt{XAB} \Rightarrow \mathtt{XaB}$$

If we apply the rule $\mathtt{XB} \to \mathtt{Xb}$ we get

$$\mathtt{AXB} \Rightarrow \mathtt{AXb} \Rightarrow \mathtt{XAb} \Rightarrow \mathtt{Xab} \Rightarrow \mathtt{ab}\ .$$

So much for noncommuting instances. Now take the string $\mathtt{AXXB}$. Again, the two rules are in competition. However, this time none destroys the applicability of the other.

$$\mathtt{AXXB} \Rightarrow \mathtt{AXXb} \Rightarrow \mathtt{XAXb}$$

$$\mathtt{AXXB} \Rightarrow \mathtt{XAXB} \Rightarrow \mathtt{XAXb}$$

As before we can derive the string $\mathtt{ab}$. Notice that in a context free grammar every pair of rules that are in competition for the same string can be used in succession with either order on condition that they do not compete for the same occurrence of a nonterminal.

A useful form for grammars is the following.

Definition 1.5.5 *A grammar is in* ***standard form*** *if all rules are of the form* $\vec{X} \to \vec{Y}$, $X \to x$ *or* $X \to \varepsilon$.

In other words, in a grammar in standard form the right hand side either consists entirely of terminals, or both left and right hand side consist of nonterminals only.

Lemma 1.5.6 *For every grammar* G *of Type* i *there exists a grammar* H *of Type* i *in standard form such that* $L(G) = L(H)$.

Proof. Put $N' := \{\mathtt{N}_a : a \in A\} \cup N$ and $h : a \mapsto \mathtt{N}_a, X \mapsto X : N \cup A \to N^1$. For each rule ρ let $h(\rho)$ be the result of applying $\overline{h}$ to both strings. Finally, let $R' := \{h(\rho) : \rho \in R\} \cup \{\mathtt{N}_a \to a : a \in A\}$, $H := \langle S, N', A, R'\rangle$. It is easy to verify, using the Commutating

Instances Lemma, that $L(H) = L(G)$. (The proof is standard and follows the pattern of some proofs below.) □

We shall now proceed to show that the conditions on Type 0 grammars are actually insignificant as regards the class of generated languages. First, we may assume a set of start symbols rather than a single one. Define the notion of a **grammar*** (**of Type** i) to be a quadruple $G = \langle \Sigma, N, A, R\rangle$ such that $\Sigma \subseteq N$ and for all $S \in \Sigma$ is a grammar (of Type i). Write $G \vdash \vec{\gamma}$ if there is an $S \in \Sigma$ such that $S \Rightarrow^*_R \vec{\gamma}$. We shall see that grammars* are not more general than grammars with respect to languages. Let G be a grammar*. Define $G^\heartsuit$ as follows. Let $S^\heartsuit \notin A \cup N$ be a new nonterminal and add the rules $S^\heartsuit \to X$ to R for all $X \in \Sigma$. It is easy to see that $L(G^\heartsuit) = L(G)$. (Moreover, the derivations differ minimally.) Notice also that we have not changed the type of the grammar.

The second simplification concerns the requirement that the set of terminals and the set of nonterminals be disjoint. We shall show that it too can be dropped without increasing the generative power. We shall sometimes work without this condition, as it can be cumbersome to deal with.

Definition 1.5.7 *A* ***quasi–grammar*** *is a quadruple* $\langle \mathtt{S}, N, A, R\rangle$ *such that* A *and* N *are finite and nonempty sets,* $\mathtt{S} \in N$*, and* R *a semi Thue system over* $N \cup A$ *such that if* $\langle \vec{\alpha}, \vec{\beta}\rangle \in R$ *then* $\vec{\alpha}$ *contains a symbol from* N*.*

Proposition 1.5.8 *For every quasi–grammar there exists a grammar which generates the same language.*

Proof. Let $\langle \mathtt{S}, N, A, R\rangle$ be a quasi–grammar. Put $N_1 := N \cap A$. Then assume for every $a \in N_1$ a new symbol $\mathtt{Y}_a$. Put $Y := \{\mathtt{Y}_a : a \in N_1\}$, $N^\circ := (N - N_1) \cup Y$, $A^\circ := A$. Now $N^\circ \cap A^\circ = \varnothing$. We put $\mathtt{S}^\circ := \mathtt{S}$ if $\mathtt{S} \notin A$ and $\mathtt{S}^\circ := \mathtt{Y}_\mathtt{S}$ if $\mathtt{S} \in A$. Finally, we define the rules. Let $\vec{\alpha}^\circ$ be the result of replacing every occurrence of an $a \in N_1$ by the corresponding $\mathtt{Y}_a$. Then let

$$R^\circ := \{\vec{\alpha}^\circ \to \vec{\beta}^\circ : \vec{\alpha} \to \vec{\beta} \in R\} \cup \{\mathtt{Y}_a \to a : a \in N_1\}\ .$$

Together with $G^\circ := \langle \mathtt{S}^\circ, N^\circ, A^\circ, R^\circ\rangle$ we claim that $L(G^\circ) = L(G)$. The reason is as follows. We define a homomorphism $h : (A \cup N)^* \to (A^\circ \cup N^\circ)^*$ by $h(a) := a$ for $a \in A - N_1$, $h(a) := \mathtt{Y}_a$ for $a \in N_1$ and $h(X) := X$ for all $X \in N - N_1$. Then $h(\mathtt{S}) = \mathtt{S}^\circ$ as well as $h(R) \subseteq R^\circ$. From this it immediately follows that if $G \vdash \vec{\alpha}$ then $G^\circ \vdash h(\vec{\alpha})$. This can be shown by induction on the length of a derivation. Since we can derive $\vec{\alpha}$ in G° from $h(\vec{\alpha})$, we certainly have $L(G) \subseteq L(G^\circ)$. For the converse we have to convince ourselves that an instance of a rule $\mathtt{Y}_a \to a$ can always be moved to the end of the derivation. For if $\vec{\alpha} \to \vec{\beta}$ is a rule then it is of type $\mathtt{Y}_b \to b$ and replaces a $\mathtt{Y}_b$ by b; and hence it commutes with that instance of the first rule. Or it is of a different form, namely $\vec{\alpha}^\circ \to \vec{\beta}^\circ$; since a does not occur in $\vec{\alpha}^\circ$, these two instances of rules commute. Now that this is shown, we conclude from $G^\circ \vdash \vec{\alpha}$ already $G^\circ \vdash \vec{\alpha}^\circ$. This implies $G \vdash \vec{\alpha}$. ☐

The last of the conditions, namely that the left hand side of a production must contain a nonterminal, is also no restriction. For let $G = \langle \mathtt{S}, N, A, R\rangle$ be a grammar which does not comply with this condition. Then for every terminal a let a^1 be a new symbol and let $A^1 := \{a^1 : a \in A\}$. Finally, for each rule $\rho = \vec{\alpha} \to \vec{\beta}$ let ρ^1 be the result of replacing every occurrence of an $a \in A$ by a^1 (on every side of the production). Now set $\mathtt{S}' := \mathtt{S}$ if $\mathtt{S} \notin A$ and $\mathtt{S}' := \mathtt{S}^1$ otherwise, $R' := \{\rho^1 : \rho \in R\} \cup \{a^1 \to a : a \in A\}$. Finally put

$$G' := \langle \mathtt{S}', N \cup A^1, A, R'\rangle\ .$$

It is not hard to show that $L(G') = L(G)$. These steps have simplified the notion of a grammar considerably. It smost general form is $\langle \Sigma, N, A, R\rangle$, where $\Sigma \subseteq N$ is the set of start symbols and $R \subset (N \cup A)^* \times (N \cup A)^*$ a finite set.

Next we shall show a general theorem for context sensitive languages. A grammar is called **noncontracting** if either no rule is contracting or only the rule $S \to \varepsilon$ is contracting and in this case the symbol S never occurs to the right of a production.

Theorem 1.5.9 *A language is context sensitive if and only if there is a noncontracting grammar that generates it.*

Proof. Since all context sensitive grammars are noncontracting, one direction is trivial. Let G be a noncontracting grammar. We shall construct a grammar $G^\spadesuit$ which is context sensitive and such that $L(G^\spadesuit) = L(G)$. To this end, let $\rho : X_0X_1 \dots X_{m-1} \to Y_0Y_1 \dots Y_{n-1}$ be a production. (As remarked above, we can reduce attention to such rules and rules of the form $X \to a$. Since the latter are not contracting, only the former kind needs attention.) We assume m new symbols, Z_0, Z_1, ..., Z_{m-1}. Let $\rho^\spadesuit$ be the following set of rules.

$$\begin{array}{lcl}
X_0X_1 \dots X_{m-1} & \to & Z_0X_1 \dots X_{m-1} \\
Z_0X_1X_2 \dots X_{m-1} & \to & Z_0Z_1X_2 \dots X_{m-1} \\
 & \dots & \\
Z_0Z_1 \dots Z_{m-2}X_{m-1} & \to & Z_0Z_1 \dots Z_{m-1} \\
Z_0Z_1 \dots Z_{m-1} & \to & Y_0Z_1 \dots Z_{m-1} \\
Y_0Z_1Z_2 \dots Z_{m-1} & \to & Y_0Y_1Z_2 \dots Z_{m-1} \\
 & \dots & \\
Y_0Y_1 \dots Y_{m-2}Z_{m-1} & \to & Y_0Y_1 \dots Y_{n-1}
\end{array}$$

Let $G^\spadesuit$ be the result of replacing all non context sensitive rules ρ by $\rho^\spadesuit$. The new grammar is context sensitive. Now let us be given a derivation in G. Then replace every instance of a rule ρ by the given sequence of rules in $\rho^\spadesuit$. This gives a derivation of the same string in $G^\spadesuit$. Conversely, let us be given a derivation in $G^\spadesuit$. Now look at the following. If somewhere the rule $\rho^\spadesuit$ is applied, and then a rule from $\rho_1^\spadesuit$ then they commute unless $\rho_1 = \rho$ and the second instance is inside that of that rule instance of $\rho^\spadesuit$. Thus, by suitably reordering the derivation is a sequence of segments, where each segment is a sequence of the rule $\rho^\spadesuit$ for some ρ, so that it begins with $\vec{X}$ and ends with $\vec{Y}$. This can be replaced by ρ. Do this for every segment. This yields a derivation in G. □

The following theorem shows that the languages of Type 1 are not closed under arbitrary homomorphism, if we anticipate that there exist languages which are of Type 0 but not of Type 1. (This is Theorem 1.7.14.)

Theorem 1.5.10 *Let* $\mathtt{a}, \mathtt{b} \notin A$ *be (distinct) symbols. For every language* L *over* A *of Type 0 there is a language* M *over* $A \cup \{\mathtt{a}, \mathtt{b}\}$ *of Type 1 such that for every* $\vec{x} \in L$ *there is an* i *with* $\mathtt{a}^i\mathtt{b}\vec{x} \in M$ *and every* $\vec{y} \in M$ *has the form* $\mathtt{a}^i\mathtt{b}\vec{x}$ *with* $\vec{x} \in L$.

Proof. We put $N^♣ := N \cup \{\mathtt{A}, \mathtt{B}, \mathtt{S}^♣\}$. Let

$$\rho : X_0 X_1 \ldots X_{m-1} \to Y_0 Y_1 \ldots Y_{n-1}$$

be a contracting rule. Then put

$$\rho^♣ := X_0 X_1 \ldots X_{m-1} \to \mathtt{A}^{m-n} Y_0 Y_1 \ldots Y_{n-1} \, .$$

$\rho^♣$ is certainly not contracting. If ρ is not contracting then put $\rho^♣ := \rho$. Let $R^♣$ consist of all rules of the form $\rho^♣$ for $\rho \in R$ as well as the following rules.

$$\begin{array}{lcll} \mathtt{S}^♣ & \to & \mathtt{BS} & \\ X\mathtt{A} & \to & \mathtt{A}X & (X \in N^♣) \\ \mathtt{BA} & \to & \mathtt{aB} & \\ \mathtt{B} & \to & \mathtt{b} & \end{array}$$

Let $M := L(G^♣)$. Certainly, $\vec{y} \in M$ only if $\vec{y} = \mathtt{a}^i\mathtt{b}\vec{x}$ for some $\vec{x} \in A^*$. For strings contain $\mathtt{B}$ (or $\mathtt{b}$) only once. Further, $\mathtt{A}$ can be changed into $\mathtt{a}$ only if it occurs directly before $\mathtt{B}$. After that we get $\mathtt{B}$ followed by $\mathtt{a}$. Hence $\mathtt{b}$ must occur after all occurrences of $\mathtt{a}$ but before all occurrences of $\mathtt{B}$. Now consider the homomorphism $\overline{v}$ defined by $v : \mathtt{A}, \mathtt{a}, \mathtt{B}, \mathtt{b}, \mathtt{S}^♣ \mapsto \varepsilon$ and $v : X \mapsto X$ for $X \in N$, $v : a \mapsto a$ for $a \in A$. If $\langle \vec{\alpha}_i : i < n \rangle$ is a derivation in $G^♣$ then $\langle \overline{v}(\vec{\alpha}_i) : 0 < i < n \rangle$ is a derivation in G (if we disregard repetitions). In this way one shows that $\mathtt{a}^i\mathtt{b}\vec{x} \in M$ implies $\vec{x} \in L(G)$. Next, let $\vec{x} \in L(G)$. Let $\langle \vec{\alpha}_i : i < n \rangle$ be a derivation of $\vec{x}$ in G. Then do the following. Define $\vec{\beta}_0 := S^♣$ and $\vec{\beta}_1 = \mathtt{BS}$. Further let $\vec{\beta}_{i+1}$ of the form $\mathtt{BA}^{k_i}\vec{\alpha}_i$ for some k_i which is determined inductively. It is easy to see that $\vec{\beta}_{i+1} \vdash_{G^\clubsuit} \vec{\beta}_{i+2}$, so that one can complete the sequence $\langle \vec{\beta}_i : i < n+1 \rangle$ to a derivation. From $\mathtt{BA}^{k_n}\vec{x}$ one can derive $\mathtt{a}^{k_n}\mathtt{b}\vec{x}$. This shows that $\mathtt{a}^{k_n}\mathtt{b}\vec{x} \in M$. □

Now let $v : A \to B^*$ be a map. v (as well as the generated homomorphism $\overline{v}$) is called ε–**free** if $v(a) \neq \varepsilon$ for all $a \in A$.

Theorem 1.5.11 *Let L_1 and L_2 be languages of Type i, $0 \leq i \leq 3$. Then the following are also languages of Type i.*

1. $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^*.
2. $\overline{v}[L_1]$, *where v is ε–free.*

If $i \neq 1$ then $\overline{v}[L_1]$ also is of Type i even if v is not ε–free.

Proof. Before we begin, we remark the following. If $L \subseteq A^*$ is a language and $G = \langle S, N, A, R \rangle$ a grammar over A which generates L then for an arbitrary $B \supseteq A$ $\langle S, N, B, R \rangle$ is a grammar over B which generates $L \subseteq B^*$. Therefore we may now assume that L_1 and L_2 are languages over the same alphabet. The first claim is seen as follows. We have $G_1 = \langle \mathtt{S}_1, N_1, A, R_1 \rangle$ and $G_2 = \langle \mathtt{S}_2, N_2, A, R_2 \rangle$ with $L(G_1) = L(G_2)$. By renaming the nonterminals of G_2 we can see to it that $N_1 \cap N_2 = \varnothing$. Now we put $N_3 := N_1 \cup N_2 \cup \{\mathtt{S}^\diamondsuit\}$ (where $\mathtt{S}^\diamondsuit \notin N_1 \cup N_2$) and $R := R_1 \cup R_2 \cup \{\mathtt{S}^\diamondsuit \to \mathtt{S}_1, \mathtt{S}^\diamondsuit \to \mathtt{S}_2\}$. This defines $G_3 := \langle \mathtt{S}^\diamondsuit, N_3, A, R_3 \rangle$. This is a grammar which generates $L_1 \cup L_2$. The second claim is proved similarly. We introduce a new start symbol $\mathtt{S}^\times$ together with the rules $\mathtt{S}^\times \to \mathtt{S}_1\mathtt{S}_2$ where $\mathtt{S}_1$ is the start symbol of G_1 and G_2 the start symbol of G_2. This yields a grammar of Type i except if $i = 3$. In this case the fact follows from the results of Section 2.1. It is however not difficult to construct a grammar which is regular and generates the language $L_1 \cdot L_2$. Now for L_1^*. Let $\mathtt{S}$ be the start symbol for a grammar G which generates L_1. Then introduce a new symbol $\mathtt{S}^+$ as well as a new start symbol $\mathtt{S}^*$ together with the rules

$$\begin{array}{lcl} \mathtt{S}^* & \to & \varepsilon \mid \mathtt{S} \mid \mathtt{SS}^+ \\ \mathtt{S}^+ & \to & \mathtt{S} \mid \mathtt{SS}^+ \end{array}$$

This grammar is of Type i and generates L_1^*. (Again the case $i = 3$ is an exception that can be dealt with in a different way.) The last is the closure under ε–free homomorphisms. Let v be a homomorphism. We extend it by putting $v(X) := X$ for all nonterminals X. Then replace the rules $\rho : \vec{\alpha} \to \vec{\beta}$ by $\overline{v}(\rho) : \overline{v}(\vec{\alpha}) \to \overline{v}(\vec{\beta})$. If $i = 0, 2$, this does not change the type. If $i = 1$ we must additionally require that v is ε–free. For if $\vec{\gamma} X \vec{\delta} \to \vec{\gamma}\vec{\alpha}\vec{\delta}$ is a rule and $\vec{\alpha}$ is a terminal string we may have $\overline{v}(\alpha) = \varepsilon$. This is however not the case if v is ε–free. If $i = 3$ again a different method must be used. For now — after applying the replacement — we have rules of the form $X \to \vec{x}Y$ and $X \to \vec{x}$, $\vec{x} = x_0 x_1 \dots x_{n-1}$. Replace the latter by $X \to x_0 Z_0$, $Z_i \to x_i Z_{i+1}$ and $Z_{n-2} \to x_{n-1} Y$ and $Z_{n-2} \to x_{n-1}$, respectively. □

Definition 1.5.12 *Let A be a (possibly infinite) set. A nonempty set $\mathcal{S} \subseteq \wp(A^*)$ is called an* ***abstract family of languages over*** *A if the following holds.*

1. *For every $L \in \mathcal{S}$ there is a finite $B \subseteq A$ such that $L \subseteq B^*$.*
2. *If $h : A^* \to A^*$ is a homomorphism and $L \in \mathcal{S}$ then also $h[L] \in \mathcal{S}$.*
3. *If $h : A^* \to A^*$ is a homomorphism and $L \in \mathcal{S}$, $B \subseteq A$ finite, then also $h^{-1}[L] \cap B^* \in \mathcal{S}$.*
4. *If $L \in \mathcal{S}$ and R is a regular language then $L \cap R \in \mathcal{S}$.*
5. *If $L_1, L_2 \in \mathcal{S}$ then also $L_1 \cup L_2 \in \mathcal{S}$ and $L_1 \cdot L_2 \in \mathcal{S}$.*

We still have to show that the languages of Type i are closed with respect to intersections with regular languages. A proof for the Types 3 and 2 is found in Section 2.1, Theorem 2.1.14. This proof can be extended to the other types without problems.

The regular, the context free and the Type 0 languages over a fixed alphabet form an abstract family of languages. The context sensitive languages fulfill all criteria except for the closure under homomorphisms. It is easy to show that the regular languages over A form the smallest abstract family of languages. More on this subject can be found in (Ginsburg, 1975).

Notes on this section. That languages are actually sets of strings is not self evident. Moreover, the idea that they can be defined by means of formal processes did not become apparent until the 1930s. The idea of formalizing rules for transforming strings appeared in (Thue, 1914). The observation that languages (in his case formal languages) could be seen as generated from semi Thue systems, is due Emil Post. Also, he has invented independently what is now known as the Turing machine and has shown that this machine does nothing but string transformations. The idea was picked up by Noam Chomsky and he defined the hierarchy which is now named after him (see for example (Chomsky, 1959), but the ideas have been circulating earlier). In view of Theorem 1.5.10 it is unclear, however, whether grammars of Type 0 or 1 have any relevance for natural language syntax, since there is no notion of a constituent that they define as opposed to context free grammars. There are other points to note about these types of grammars. (Langholm, 2001) voices clear discontentment with the requirement of a single start symbol, which is in practice anyway not complied with.

Exercise 26. Let T be a semi Thue system over A and $A \subseteq B$. Then T is also a semi Thue system T' over B. Characterize $\Rightarrow_{T'}^{*} \subseteq B^* \times B^*$ by means of $\Rightarrow_T^* \subseteq A^* \times A^*$. *Remark.* This exercise shows that with the Thue system we also have to indicate the alphabet on which it is based.

Exercise 27. Prove the Commutating Instances Lemma.

Exercise 28. Show that every finite language is regular.

Exercise 29. Let G be a grammar with rules of the form $X \to \vec{\alpha}$. Show that $L(G)$ is context free. Likewise show that $L(G)$ is regular if all rules have the form $X \to \alpha_0 \cdot \alpha_1$ where $\alpha_0 \in A \cup \{\varepsilon\}$ and $\alpha_1 \in N \cup \{\varepsilon\}$.

Exercise 30. Let G be a grammar in which every rule distinct from $X \to a$ is strictly expanding. Show that a derivation of a string of length n takes at most $2n$ steps.

Exercise 31. Show that the language $\{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$ is context free but not regular.

Exercise 32. Write a Type 1 grammar for the languages $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$ and one for $\{\vec{x} \cdot \vec{x} : \vec{x} \in A^*\}$.

1.6 Grammar and Structure

In contrast to processes that replace of strings by strings we may also consider processes that successively replace parts structures by structures. Such processes we shall look at in this section. Such algorithms can in principle generate any kind of structure but we will restrict our attention to algorithms that generate ordered trees. There are basically two kinds of algorithms: the first is like the grammars of the previous section, generating intermediate structures that are not proper structures of the language; and the second, which generates in each step a structure of the language.

First, we shall extend the notion of a graph. Instead of graphs we shall have to deal with so called *multigraphs*. A **directed multigraph** is a structure $\langle V, \langle K_i : i < n\rangle\rangle$ where is V a set, the set of **vertices** and $K_i \subseteq V \times V$ a disjoint set, the set of **edges** of type i. In our case edges are always directed. We shall not mention this fact explicitly later on as it is tacitly assumed. Ordered trees are one example among many of (directed) multigraphs. For technical reasons we shall not exclude the case $V = \varnothing$, so that $\langle \varnothing, \langle \varnothing : i < n\rangle\rangle$ also is a multigraph. Next we shall introduce a colouring on the vertices. A **vertex–colouring** is a function $\mu_V : V \to F_V$ where F_V is a nonempty set, the set of **vertex colours**. Think of the labelling as being a vertex colouring on the graph. The principal structures are therefore vertex coloured multigraphs. However, from a technical point of view the different edge relations can also be viewed as colourings on the edges.

Namely, if v and w are vertices, we colour the edge $\langle v, w\rangle$ by the set $\{i : \langle v, w\rangle \in K_i\}$. This set may be empty.

Definition 1.6.1 *An $\langle F_V, F_E\rangle$–**coloured multigraph** or simply a γ–**graph (over** F_V **and** F_E**)** is a triple $\langle V, \mu_V, \mu_E\rangle$, where V is a (possibly empty) set and $\mu_V : V \to F_V$ as well as $\mu_E : V \times V \to \wp(F_E)$ are functions.*

Now, in full analogy to the string case we shall distinguish terminal and nonterminal colours. Replacement is done here (though not necessarily) by replacing a single vertex by a graph. Replacing a vertex by another structure means embedding a structure into some other structure. We need to be told how to do so. Before we begin we shall say something about the replacement of graphs in general. The reader is asked to look at Figure 1.5. The graph $\mathfrak{G}_3$ is the result of replacing in $\mathfrak{G}_1$ the encircled dot by $\mathfrak{G}_2$. The edge colours are 1 and 2 (the vertex colours pose no problems, so they are omitted here for clarity).

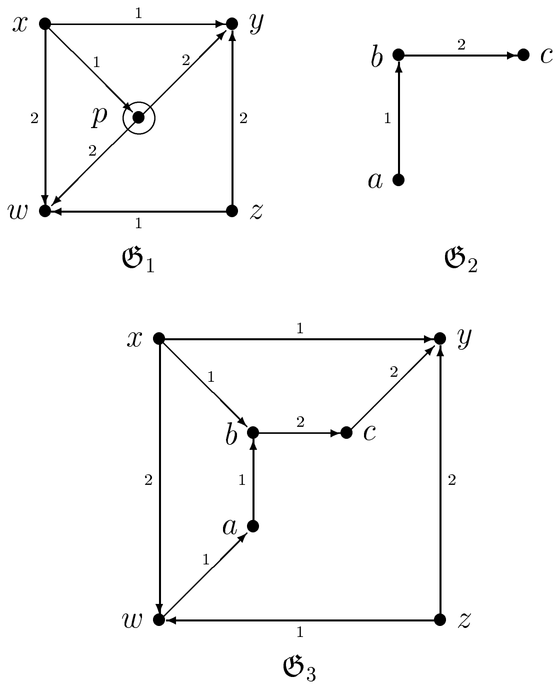
Let $\mathfrak{G} = \langle E, \mu_E, \mu_K\rangle$ be a γ–graph. Furthermore, let M_1 and M_2 be subsets of E such that $M_1 \cap M_2 = \varnothing$ and $M_1 \cup M_2 = E$. Then we define on M_1 and M_2 subgraphs $\mathfrak{M}_i = \langle M_i, \mu_V^i, \mu_E^i\rangle$ by $\mu_V^i := \mu_V \restriction M_i$ and $\mu_E^i := \mu_E \restriction M_i \times M_i$. These graphs do not completely determine $\mathfrak{G}$ since there is no information on the edges between them. We therefore define functions $in, out : M_2 \times F_E \to \wp(M_1)$, which for every vertex of M_2 and every edge colour name the set of all vertices of M_1 which lie on an edge with the vertex that either is directed into M_1 or goes outwards.

$$\begin{array}{lcl} in(x, f) & := & \{y \in M_1 : f \in \mu_E(\langle y, x\rangle)\}, \\ out(x, f) & := & \{y \in M_1 : f \in \mu_E(\langle x, y\rangle)\}. \end{array}$$

It is clear that $\mathfrak{M}_1$, $\mathfrak{M}_2$ and the functions in and out determine $\mathfrak{G}$ completely. In our example we have

$$\begin{array}{lcllcl} in(p, 1) & = & \{x\}, & in(p, 2) & = & \varnothing, \\ out(p, 1) & = & \varnothing, & out(p, 2) & = & \{w, y\}. \end{array}$$

Figure 1.5: A Graph–Replacement



Now assume that we wish to replace $\mathfrak{M}_2$ by a different graph $\mathfrak{H}$. Then not only do we have to know $\mathfrak{H}$ but also the functions $in, out : H \times F_E \to \wp(M_1)$. This, however, is not the way we wish to proceed here. We want to formulate rules of replacement that are general in that they do not presuppose exact knowledge about the embedding context. We shall only assume that the functions $in(x,f)$ and $out(x,f)$, $x \in H$, are systematically defined from the sets $in(y,g)$, $out(y,g)$, $y \in M_2$. We shall therefore only allow to specify how the sets of the first kind are formed from the sets of the second kind. This we do by means of four so called *colour functionals*. A **colour functional from $\mathfrak{H}$ to $\mathfrak{M}_2$** is a map

$$\mathfrak{F} : H \times F_E \to \wp(M_2 \times F_E)$$

In our case a functional is a function from $\{a,b,c\} \times \{1,2\}$ to $\wp(\{p\} \times \{1,2\})$. We can simplify this to a function from $\{a,b,c\} \times \{1,2\}$ to $\wp(\{1,2\})$. The colour functionals are called $\mathfrak{II}$, $\mathfrak{IO}$, $\mathfrak{OI}$ and $\mathfrak{OO}$. They are arranged into the following 2×2–matrix:

$$\mathfrak{F} := \begin{pmatrix} \mathfrak{II} & \mathfrak{IO} \\ \mathfrak{OI} & \mathfrak{OO} \end{pmatrix}$$

This yields for the example of Figure 1.5 the following (we only give values when the functions do not yield $\varnothing$).

$$\begin{array}{lcl} \mathfrak{II} & : & \langle b,1\rangle \mapsto \{1\} \\ \mathfrak{IO} & : & \varnothing \\ \mathfrak{OI} & : & \langle a,2\rangle \mapsto \{1\} \\ \mathfrak{OO} & : & \langle c,2\rangle \mapsto \{2\} \end{array}$$

The result of substituting $\mathfrak{M}_2$ by $\mathfrak{H}$ by means of the colour functionals from $\mathfrak{F}$ is denoted by $\mathfrak{G}[\mathfrak{H}/\mathfrak{M}_2 : \mathfrak{F}]$. This graph is the union of $\mathfrak{M}_1$ and $\mathfrak{H}$ together with the functions in^+, out^+, which are defined as follows.

$$\begin{array}{lcl} in^+(x,f) & := & \bigcup\langle in(x,g) : g \in \mathfrak{II}(x,f)\rangle \\ & & \cup \bigcup\langle out(x,g) : g \in \mathfrak{OI}(x,f)\rangle \\ out^+(x,f) & := & \bigcup\langle out(x,g) : g \in \mathfrak{OO}(x,f)\rangle \\ & & \cup \bigcup\langle in(x,g) : g \in \mathfrak{IO}(x,f)\rangle \end{array}$$

If $g \in \mathfrak{I}\mathfrak{I}(x, f)$ we say that an edge with colour f into x is *transmitted as an ingoing edge of colour g to y*. If $g \in \mathfrak{O}\mathfrak{I}(x, f)$ we say that an edge with colour f going out from x is *transmitted as an ingoing edge with colour g to y*. Analogously for $\mathfrak{I}\mathfrak{O}$ and $\mathfrak{O}\mathfrak{O}$. So, we do allow for an edge to change colour and to change direction when being transmitted. If edges do not change direction, we only need the functionals $\mathfrak{I}\mathfrak{I}$ and $\mathfrak{O}\mathfrak{O}$, which are then denoted simply by $\mathfrak{I}$ and $\mathfrak{O}$. Now we look at the special case where M_2 consists of a single element, say x. In this case a colour functional simply is a function $\mathfrak{F} : H \times F_E \to \wp(F_E)$.

Definition 1.6.2 *A **context free graph grammar with edge replacement** — a **context free** γ**–grammar** for short — is a quintuple of the form*

$$\Gamma = \langle \mathfrak{S}, F_V, F_V^T, F_E, R \rangle$$

*in which F_V is a finite set of vertex colours, F_E a finite of edge colours, $F_V^T \subseteq F_V$ a set of so called **terminal vertex colours**, $\mathfrak{S}$ a γ–graph over F_V and F_E, the so called **start graph**, and finally R a finite set of triples $\langle X, \mathfrak{H}, \mathfrak{F} \rangle$ such that $X \in F_V - F_V^T$ is a nonterminal vertex colour, $\mathfrak{H}$ a γ–graph over F_V and F_E and $\mathfrak{F}$ is a matrix of colour functionals.*

A **derivation** in a γ–grammar Γ is defined as follows. For γ–graphs $\mathfrak{G}$ and $\mathfrak{H}$ with the colours F_V and F_E the symbol $\mathfrak{G} \Rightarrow_R^1 \mathfrak{H}$ means that there is $\langle X, \mathfrak{M}, \mathbb{F} \rangle \in R$ such that $\mathfrak{H} = \mathfrak{G}[\mathfrak{M}/\mathfrak{X} : \mathbb{F}]$, where $\mathfrak{X}$ is a subgraph consisting of a single vertex x having the colour X. Further we define $\Rightarrow_R^*$ to be the reflexive and transitive closure of $\Rightarrow_R^1$ and finally we put $\Gamma \vdash \mathfrak{G}$ if $\mathfrak{S} \Rightarrow_R^* \mathfrak{G}$. A derivation **terminates** if there is no vertex with a nonterminal colour. We write $L_\gamma(\Gamma)$ for the class of γ–graphs that can be generated from Γ. Notice that the edge colours only the vertex colours are used to steer the derivation.

We also define the productivity of a rule as the difference between the cardinality of the replacing graph and the cardinality of the graph being replaced. The latter is 1 in context free γ–grammars, which is the only type we shall study here. So, the productivity is always ≥ -1. It equals -1 if the replacing graph is the empty graph. In an exercise the reader is asked to verify that such rules can always be eliminated. A rule has productivity 0 if the replacing graph consists of a single vertex. In an exercise the reader will be asked to verify that we can dispense with rules of this kind as well.

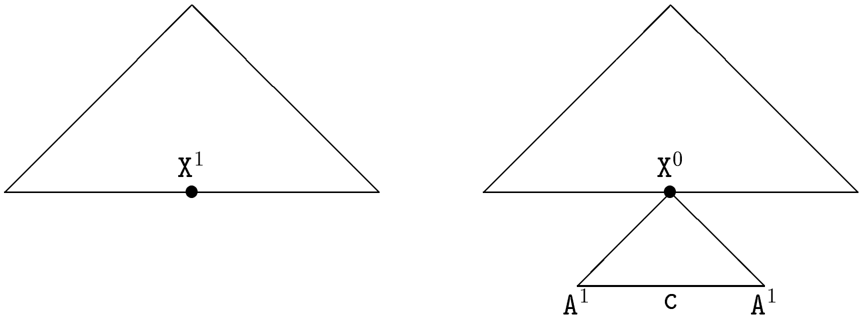
Now we shall define two types of context free γ–grammars. Both are context free as γ–grammars but the second type can generate non context free languages. This shows that the concept of γ–grammar is more general. We shall begin with ordinary context free grammars. We can view them alternatively as grammars for string replacement or as grammars that replace trees by trees. For that we shall now assume that there are no rules of the form $X \to \varepsilon$. (For such rules generate tree whose leaves are not necessarily marked by letters from A. This case can be treated if we allow labels to be in $A_\varepsilon = A \cup \{\varepsilon\}$. We shall however not deal with this case.) Let $G = \langle S, A, N, R\rangle$ be such a grammar. We put $F_V := A \cup (N \times 2)$. We write X^0 for $\langle X, 0\rangle$ and X^1 for $\langle X, 1\rangle$. $F_V^T := A \cup N \times \{0\}$. $F_E := \{<, \sqsubset\}$. Furthermore, the start graph consists of a single vertex labelled S^1 and no edge. The rules of replacement are as follows. Let $\rho = X \to \alpha_0\alpha_1 \ldots \alpha_{n-1}$ be a rule from G, and none of the α_i is ε. Then we define a γ–graph $\mathfrak{H}_\rho$ as follows. $H_\rho := \{y_i : i < n\} \cup \{x\}$. $\mu_V(x) = X^0$, $\mu_V(y_i) = \alpha_i$ if $\alpha_i \in A$ and $\mu_V(y_i) = \alpha_i^1$ if $\alpha_i \in N$.

$$\begin{array}{rcl} \mu_E^{-1}(\{<\}) & := & \{\langle y_i, x\rangle : i < n\}, \\ \mu_E^{-1}(\{\sqsubset\}) & := & \{\langle y_i, y_j\rangle : i < j < n\}. \end{array}$$

This defines $\mathfrak{H}_\rho$. Now we defined the colour functionals. For $u \in n$ we put

$$\begin{array}{rclcrcl} \mathfrak{I}_\rho(u, \sqsubset) & := & \{\sqsubset\} & \qquad & \mathfrak{O}_\rho(u, \sqsubset) & := & \{\sqsubset\} \\ \mathfrak{I}_\rho(u, <) & := & \{<\} & & \mathfrak{O}_\rho(u, <) & := & \{<\} \end{array}$$

Figure 1.6: Replacement in a Context Free Grammar



Finally we put $\rho^\gamma := \langle X, \mathfrak{H}_\rho, \{\mathfrak{I}_\rho, \mathfrak{O}_\rho\}\rangle$. $R^\gamma := \{\rho^\gamma : \rho \in R\}$.

$$\gamma G := \langle \mathfrak{S}, F_E, F_E^T, F_T, R^\gamma \rangle \ .$$

We shall show that this grammar yields exactly those trees that we associate with the grammar G. Before we do so, a few remarks are in order. The previous nonterminals are now from a technical viewpoint, terminals since they are also part of the structure that we are generating. In order to have any derivation at all we define two equinumerous sets of nonterminals. Each nonterminal N is split into the nonterminal N^1 (which is nonterminal in the new grammar) and N^0 (which is now a terminal vertex colour). We call the first kind **active**, **nonactive** the second. Notice that the rules are formulated in such a way that only the leaves of the generated trees carry active nonterminals. A single derivation step is displayed in Figure 1.6. In it, the rule $\mathtt{X} \to \mathtt{AcA}$ has been applied to the tree to the left. The result is shown on the right hand side. It is easy to show that in each derivation only leaves carry active nonterminals. This in turn shows that the derivations of the γ–grammar are in one to one correspondence with the derivations of the context free grammar. We denote by $L_B(G)$ the class of trees generated by γG, however under the identification of X^0 by X. The rules of G can therefore be interpreted as conditions on labelled ordered trees in the following way. $\mathfrak{C}$ is called a **local subtree** of $\mathfrak{B}$ if (i) it is has height 2 (so it does not possess inner nodes) and (ii) $\mathfrak{C}$ contains with each node also all its sister nodes. For a rule $\rho = X \to Y_0 Y_1 \ldots Y_{n-1}$ we define $L_\rho := \{y_i : i < n\} \cup \{x\}$, $<_\rho := \{\langle y_i, x\rangle : i < n\}$, $\sqsubset_\rho := \{\langle y_i, y_j\rangle : i < j < n\}$, and finally $\ell_\rho(x) := X$, $\ell(y_i) := Y_i$. $\mathfrak{L}_\rho := \langle L_\rho, <_\rho, \sqsubset_\rho, \ell_\rho\rangle$. Now, an isomorphism between labelled ordered trees $\mathfrak{B} = \langle B, <_\mathfrak{B}, \sqsubset_\mathfrak{B}, \ell_\mathfrak{B}\rangle$ and $\mathfrak{C} = \langle C, <_\mathfrak{C}, \sqsubset_\mathfrak{C}, \ell_\mathfrak{C}\rangle$ is a bijective map $h : B \to C$ such that $h[<_\mathfrak{B}] = <_\mathfrak{C}$, $h[\sqsubset_\mathfrak{B}] = h[\sqsubset_\mathfrak{C}]$ and $\ell_\mathfrak{C}(h(x)) = \ell_\mathfrak{B}(x)$ for all $x \in B$.

Proposition 1.6.3 *Let $G = \langle S, N, A, R\rangle$. $\mathfrak{B} \in L_B(G)$ if and only if every local tree of $\mathfrak{B}$ is isomorphic to a $\mathfrak{L}_\rho$ such that $\rho \in R$.*

Theorem 1.6.4 *Let $\boldsymbol{B}$ be a set of trees over an alphabet $A \cup N$ with terminals from A. $\boldsymbol{B} = L_B(G)$ holds for a context free grammar G if and only if there is a finite set $\{\mathfrak{L}_i : i < n\}$ of trees of height 2 and an S such that $\mathfrak{B} \in \boldsymbol{B}$ exactly if*

1. *the root carries label S,*
2. *a label is terminal if and only if the node is a leaf, and*
3. *every local tree is isomorphic to some $\mathfrak{L}_i$.*

We shall derive a few useful consequences from these considerations. It is clear that γG generates trees that do not necessarily have leaves with terminal symbols. However, we do know that the leaves carry labels either from A or from $N^1 := N \times \{1\}$ while all other nodes carry labels from $N^0 := N \times \{0\}$. For a labelled tree we define the associated string sequence $k(\mathfrak{B})$ in the usual way. This is an element of $(A \cup N^1)^*$. Let $v : A \cup (N \times 2) \to A \cup N$ be defined by $v(a) := a$, $a \in A$ and $v(X^0) := v(X^1) := X$ for $X \in N$.

Lemma 1.6.5 *Let $\gamma G \vdash \mathfrak{B}$ and $\vec{\alpha} = k(\mathfrak{B})$. Then $\vec{\alpha} \in (A \cup N^1)^*$ and $G \vdash \overline{v}(\vec{\alpha})$.*

Proof. Induction over the length of the derivation. If the length is 0 then $\vec{\alpha} = S^1$ and $\overline{v}(S^1) = S$. Since $G \vdash S$ this case is settled. Now let $\mathfrak{B}$ be the result of an application of some rule ρ^γ on $\mathfrak{C}$ where $\rho = X \to \vec{\beta}$. We then have $k(\mathfrak{C}) \in (A \cup N^1)^*$. The rule ρ^γ has been applied to a leaf; to this leaf corresponds an occurrence of X^1 in $k(\mathfrak{C})$. Therefore we have $k(\mathfrak{C}) = \vec{\delta}_1 \cdot X^1 \cdot \vec{\delta}_2$. Then $k(\mathfrak{B}) = \vec{\delta}_1 \cdot \vec{\beta} \cdot \vec{\delta}_2$. $k(\mathfrak{B})$ is the result of a single application of the rule ρ from $k(\mathfrak{C})$. □

Definition 1.6.6 *Let $\mathfrak{B}$ be a labelled ordered tree. A* ***cut through*** *$\mathfrak{B}$ is a maximal set S such that it contains no two elements comparable by $<$. If $\mathfrak{B}$ is exhaustively ordered, a cut is linearly ordered and labelled, and then we also call the string associated to this set a* ***cut****.*

Proposition 1.6.7 *Let $\gamma G \vdash \mathfrak{B}$ and let $\vec{\alpha}$ be a cut through $\mathfrak{B}$. Then $G \vdash \overline{v}(\vec{\alpha})$.*

This theorem shows that the tree provides all necessary information. If you have the tree, all essential details of the derivation can be reconstructed (up to commuting applications of rules). Now let us be given a tree $\mathfrak{B}$ and let $\vec{\alpha}$ be a cut. We say that an occurrence C of $\vec{\gamma}$ in $\vec{\alpha}$ is a **constituent of category X in $\mathfrak{B}$** if this occurrence of $\vec{\gamma}$ in $\vec{\alpha}$ is that cut defined by $\vec{\alpha}$ on $\downarrow x$ where x carries the label X. This means that $\vec{\alpha} = \vec{\delta}_1 \cdot \vec{\gamma} \cdot \vec{\delta}_2$, $C = \langle \vec{\delta}_1, \vec{\delta}_2 \rangle$ and $\downarrow x$ contains exactly those nodes that do not belong to $\vec{\delta}_1$ or $\vec{\delta}_2$. Further, let G be a context free grammar. A substring occurrence of $\vec{\gamma}$ is a G–constituent of category X in $\vec{\alpha}$ if there is a γG–tree for which there exists a cut $\vec{\alpha}$ such that the occurrence $\vec{\gamma}$ is a constituent of category X. If G is clear from the context, we shall omit it.

Lemma 1.6.8 *Let $\mathfrak{B}$ be a γG–tree and $\vec{\alpha}$ a cut through $\mathfrak{B}$. Then there exists a tree $\mathfrak{C}$ with associated string $\vec{\beta}$ and $\overline{v}(\vec{\beta}) = \overline{v}(\vec{\alpha})$.*

Lemma 1.6.9 *Let $G \vdash \vec{\alpha}_1 \cdot \vec{\gamma} \cdot \vec{\alpha}_2$, $C = \langle \vec{\alpha}_1, \vec{\alpha}_2 \rangle$ an occurrence of $\vec{\gamma}$ as a G–constituent of category X. Then C is a G–constituent occurrence of X in $C(X) = \vec{\alpha}_1 \cdot X \cdot \vec{\alpha}_2$.*

For a proof notice that if $\vec{\alpha}_1 \cdot \vec{\gamma} \cdot \vec{\alpha}_2$ is a cut and $\vec{\gamma}$ is a constituent of category X therein then $\vec{\alpha}_1 \cdot X \cdot \vec{\alpha}_2$ also is a cut.

Theorem 1.6.10 (Constituent Substitution) *Suppose that C is an occurrence of $\vec{\beta}$ as a G–constituent of category X. Furthermore, let $X \vdash_G \vec{\gamma}$. Then $G \vdash C(\vec{\gamma}) = \vec{\alpha}_1 \cdot \vec{\gamma} \cdot \vec{\alpha}_2$ and C is a G–constituent occurrence of $\vec{\gamma}$ of category X.*

Proof. By assumption there is a tree in which $\vec{\beta}$ is a constituent of category X in $\vec{\alpha}_1 \cdot \vec{\beta} \cdot \vec{\alpha}_2$. Then there exists a cut $\vec{\alpha}_1 \cdot X \cdot \vec{\alpha}_2$ through this tree, and by Lemma 1.6.8 there exists a tree with associated string $\vec{\alpha}_1 \cdot X \cdot \vec{\alpha}_2$. Certainly we have that X is a constituent in this tree. However, a derivation $X \vdash_G \vec{\gamma}$ can in this case be extended to a γG–derivation of $\vec{\alpha}_1 \cdot \vec{\gamma} \cdot \vec{\alpha}_2$ in which $\vec{\gamma}$ is a constituent. □

We shall now derive a rather important theorem. Suppose G is given. Then the theorem says that there is a number k such that any string of length at least k contains two constituents of identical category properly contained inside each other. More precisely, if $|\vec{x}| \geq k$ there is a tree $\mathfrak{B}$ and an $X \in N$ such that $\vec{x} = \vec{u} \cdot \vec{x} \cdot \vec{v} \cdot \vec{y} \cdot \vec{w}$ is the associated string and $\vec{x} \cdot \vec{v} \cdot \vec{y}$ as well $\vec{v}$ constituents of category X, and finally $\vec{x} \neq \varepsilon$ or $\vec{y} \neq \varepsilon$. As a preparation we show the following.

Lemma 1.6.11 *Let G be a context free grammar. Then there exists a number k_G such that for each derivation tree of a string of length $\geq k_G$ there are two constituents $\downarrow y$ and $\downarrow z$ of identical category such that $y \leq z$ or $z \leq y$, and the associated strings are different.*

Proof. To begin with, notice that nothing changes in our claim if we eliminate the unproductive rules. This does not change the constituent structure. Now let π be the maximum of all productivities of rules in G, and put $\nu := |N|$. Then let $k_G := (1+\pi)^\nu + 1$. We claim that this is the desired number. (We can assume that $\pi > 0$. Otherwise G only generates strings of length 1, and then $k_G := 2$ satisfies our claim.) For let $\vec{x}$ be given such that $|\vec{x}| \geq k_G$. Then there exists in every derivation tree a branch of length $> \nu$.

(If not, there can be no more than π^ν leaves.) On this branch we have two nonterminals with identical label. The strings associated to these nodes are different since we have no unproductive rules. □

We say, an occurrence C of $\vec{u}$ is a **left constituent part** if C is an occurrence of a prefix of a constituent. Likewise we define the notion of a **right constituent part**. $\vec{x}$ contains a left constituent part $\vec{z}$ if some suffix of $\vec{u}$ is a left constituent part. We also remark the following. If $\vec{u}$ is a left constituent part and a proper substring of $\vec{x}$ then $\vec{x} = \vec{v}\vec{v}_1\vec{u}$ with $\vec{v}_1$ a possibly empty sequence of constituents and $\vec{v}$ a right constituent part. This will be of importance in the sequel.

Lemma 1.6.12 *Let G be a context free grammar. Then there exists a number k'_G such that for every derivation tree of a string $\vec{x}$ and every occurrence in a derivable string $\vec{x}$ of a string $\vec{z}$ of length $\geq k'_G$ there exist two different left or two different right constituent parts $\vec{y}$ and $\vec{y}_1$ of $\vec{z}$ of constituents that have the same category. Also, necessarily $\vec{y}$ is a prefix of $\vec{y}_1$ or $\vec{y}_1$ a prefix of $\vec{y}$ in case that both are left constituent parts, and $\vec{y}$ is a suffix of $\vec{y}_1$ or $\vec{y}_1$ a suffix of $\vec{y}$ in case that both are right constituent parts.*

Proof. Let $\nu := |N|$ and let π be the maximal productivity of a rule from G. We can assume that $\pi \geq 2$. Put $k'_G := (2 + 2\pi)^\nu$. We show by induction on the number m that a string of length $\geq (2 + 2\pi)^m$ has at least m left or at least m right constituent parts that are contained in each other. If $m = 1$ the claim is trivial. Assume that it holds for $m \geq 1$. We shall show that it also holds for $m + 1$. Let $\vec{z}$ be of length $\geq (2 + 2\pi)^{m+1}$. Let $\vec{x} = \prod_{i<2\pi+2} \vec{x}_i$ for certain $\vec{x}_i$ with length at least $(2 + 2\pi)^m$. By induction hypothesis each $\vec{x}_i$ contains at least m constituent parts. Now we do not necessarily have $(2\pi + 2)m$ constituent parts in $\vec{x}$. For if $\vec{x}_i$ contains a left part then $\vec{x}_j$ with $j > i$ may contain the corresponding right part. (There is only one. The sections in between contain subwords of that constituent occurrence.) For each left constituent part we count at most one (corresponding) right constituent part. In total we have at least $(1+\pi)m \geq m+1$ constituent parts. However, we have to verify that at least $m+1$ of these are contained inside each other. Assume this is not the case, for all $\vec{x}_i$. Then $\vec{x}_i$, $i < 2\pi + 2$, contains exactly m left or exactly m right constituent parts. Case 1. $\vec{x}_0$ contains m left constituent parts inside each other. If $\vec{x}_1$ also contains m left constituent parts inside each other, we are done. Now suppose that this is not the case. Then $\vec{x}_1$ contains m right constituent parts inside each other. Then we obviously get m entire constituent stacked inside each other. Again, we would be done if $\vec{x}_2$ contained m right constituent parts inside each other. If not, then $\vec{x}_2$ contains exactly m left constituent parts. And again we would be done if these would not correspond to exactly m right part that $\vec{x}_3$ contains. And so on. Hence we get a sequence of length π of constituents which each contain m constituents stacked inside each other. Now three cases arise: (a) one of the constituents is a left part of some constituent, (b) one of the constituent is a right part of some constituent. (For if neither is the case, we have a rule of arity $> \pi$, a contradiction.) In case (a) we evidently have $m+1$ left constituent parts stacked inside each other, and in case (b) $m+1$ right constituent parts. Case 2. $\vec{x}_0$ contains m right hand constituents stacked inside each other. Similarly. This shows our auxiliary claim. Putting $m := \nu + 1$ the main claim now follows. □

Theorem 1.6.13 (Pumping Lemma) *Given a context free language L there exists a p_L such that for every string $\vec{z} \in L$ of length at least p_L and an occurrence of a string $\vec{r}$ of length at least p_L in $\vec{z}$, $\vec{z}$ possesses a decomposition*

$$\vec{z} = \vec{u} \cdot \vec{x} \cdot \vec{v} \cdot \vec{y} \cdot \vec{w}$$

such that the following holds.

1. $\vec{x} \cdot \vec{y} \neq \varepsilon$.
2. $\vec{u} \cdot \vec{w} \neq \varepsilon$.

3. *Either the occurrence of $\vec{x}$ or the occurrence of $\vec{y}$ is contained in the specified occurrence of $\vec{r}$.*

4. $\{\vec{u} \cdot \vec{x}^i \cdot \vec{v} \cdot \vec{y}^i \cdot \vec{w} : i \in \omega\} \subseteq L$.

(The last property is called the ***pumpability*** *of the substring occurrences of $\vec{x}$ and $\vec{y}$.) Alternatively, in place of 3. one may require that $|\vec{v}| \leq p_L$. Further we can choose p_L in such a way that every derivable string $\vec{\gamma}$ with designated occurrences of a string $\vec{\alpha}$ of length $\geq p_S$ can be decomposed in the way given.*

Proof. Let G be a grammar which generates L. Let p_L be the constant defined in Lemma 1.6.12. We look at a G–tree of $\vec{z}$ and the designated occurrence of $\vec{r}$. Suppose that $\vec{r}$ has length at least p_L. Then there are two left or two right constituent parts of identical category contained in $\vec{r}$. Without loss of generality we assume that $\vec{r}$ contains two left parts. Suppose that these parts are not fully contained in $\vec{r}$. Then $\vec{r} = \vec{s}\vec{x}\vec{s}_1$ where $\vec{x}\vec{s}_1$ and $\vec{s}_1$ are left constituent parts of identical category, say X. Now $|\vec{x}| > 0$. There are $\vec{s}_2$ and $\vec{y}$ such that $\vec{v} := \vec{s}_1\vec{s}_2$ and $\vec{x}\vec{s}_1\vec{s}_2\vec{y}$ are constituents of category X.

Hence there exists a decomposition

$$\vec{z} = \vec{u} \cdot \vec{x} \cdot \vec{v} \cdot \vec{y} \cdot \vec{w} \ ,$$

where $\vec{v}$ is a constituent of the same category as $\vec{x}\vec{v}\vec{y}$ having the properties 1., 2. and 3. By the Constituent Substitution Lemma we may replace the occurrence of $\vec{x}\vec{v}\vec{y}$ by $\vec{v}$ as well as $\vec{v}$ by $\vec{x}\vec{v}\vec{y}$. This yields, after an easy induction, Claim 4. Now let the smaller constituent part be contained in $\vec{r}$ but not the larger one. Then we have a decomposition $\vec{r} = \vec{s}\vec{x}\vec{v}\vec{s}_1$ such that $\vec{v}$ is a constituent part of category X and $\vec{x}\vec{v}\vec{s}_1$ left constituent part of a constituent of category X. Then there exists a $\vec{s}_2$ such that also $\vec{x}\vec{v}\vec{s}_1\vec{s}_2$ is a constituent of category X. Now put $\vec{y} := \vec{s}_1\vec{s}_2$. Then we also have $\vec{y} \neq \varepsilon$. The third case is if both parts are proper substrings of $\vec{r}$. Also here we find the desired decomposition. If we want to have in place of 3. that $\vec{v}$ is as small as possible then notice that $\vec{v}$

already is a constituent. If it has length $\geq (1+\pi)^\nu$ then there is a decomposition of $\vec{v}$ such that it contains pumpable substrings. Hence we may require in place of 3. that $|\vec{v}| \leq p_G$. ☐

The Pumping Lemma can be stated more concisely as follows. For every large enough derivable string $\vec{x}$ there exist contexts C, D, where $C \neq \langle \varepsilon, \varepsilon \rangle$, and a string $\vec{y}$ such $\vec{x} = D(C(\vec{x}))$, and we have $D(C^k(\vec{y})) \in L$ for every $k \in \omega$.

Notice that we may choose $i = 0$ as well. This means that not only we can pump 'up' the string so that it becomes longer except if $i = 1$, but we may also pump it 'down' ($i = 0$) so that the string becomes shorter. However, one can pump down only once. Using the Pumping Lemma we can show that the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$ is not context free.

For suppose the contrary. Then there is an m such that for all $k \geq m$ the string $\mathtt{a}^k\mathtt{b}^k\mathtt{c}^k$ can be decomposed into

$$\mathtt{a}^k\mathtt{b}^k\mathtt{c}^k = \vec{u} \cdot \vec{v} \cdot \vec{w} \cdot \vec{x} \cdot \vec{y} \,.$$

Furthermore there is an $\ell > k$ such that

$$\mathtt{a}^\ell\mathtt{b}^\ell\mathtt{c}^\ell = \vec{u} \cdot \vec{v}^2 \cdot \vec{w} \cdot \vec{x}^2 \cdot \vec{y}$$

The string $\vec{v} \cdot \vec{x}$ contains exactly $\ell - k$ times the letters $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$. It is clear that we must have $\vec{v} \subseteq \mathtt{a}^* \cup \mathtt{b}^* \cup \mathtt{c}^*$. For if $\vec{v}$ contains two distinct letters, say $\mathtt{b}$ and $\mathtt{c}$, then $\vec{v}$ contains an occurrence of $\mathtt{b}$ before an occurrence of $\mathtt{c}$ (certainly not the other way around). But then $\vec{v}^2$ contains an occurrence of $\mathtt{c}$ before an occurrence of $\mathtt{b}$, and that cannot be. Analogously it is shown that $\vec{y} \in \mathtt{a}^* \cup \mathtt{b}^* \cup \mathtt{c}^*$. But this is a contradiction. We shall meet this example of a non context free language quite often in the sequel.

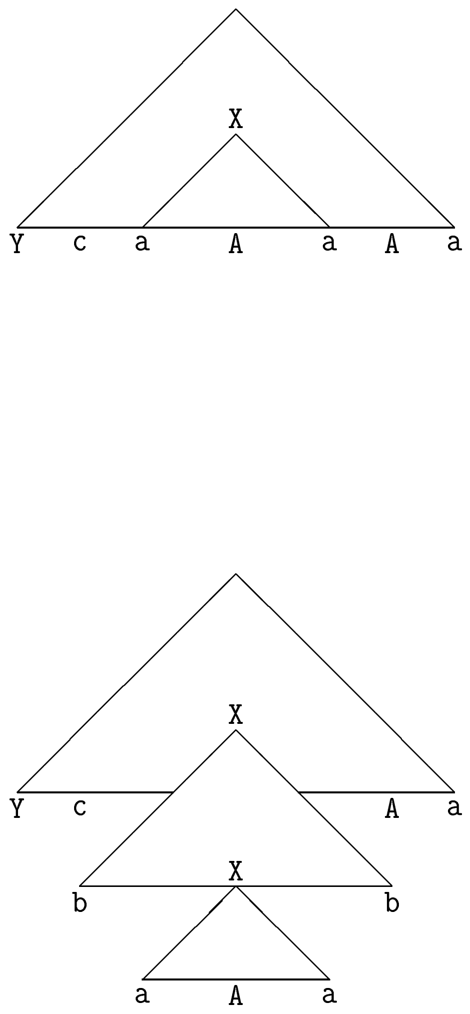
The second example of a context free graph grammar shall be the so called *tree adjunction grammars*. We take an alphabet A and a set N of nonterminals. Let $S \in N$ be the start symbol. A **center tree** is an ordered labelled tree over $A \cup N$ such that all leaves have labels from A all other nodes labels from N and the root has label S. An **adjunction tree** is an ordered labelled tree over $A \cup N$ which is distinct from ordinary trees in that of the leaves there is exactly one with a nonterminal label; this label is the same as that of the root. Interior nodes have nonterminal labels. We require that an adjunction tree has at least one leaf with a terminal symbol. An **unregulated tree adjunction grammar**, briefly UTAG, over N and A, is a quadruple $\langle \mathfrak{C}, N, A, \mathfrak{A}\rangle$ where $\mathfrak{C}$ is a finite set of center trees over N and A, and $\mathfrak{A}$ a finite set of adjunction trees over N and A. An example of a tree adjunction is given in Figure 1.7. The tree to the left is adjoined to a centre tree with root X and associated string `bXb`; the result is shown to the right. Tree adjunction can formally be defined as follows. Let $\mathfrak{B} = \langle B, <, \sqsubset, \ell\rangle$ be a tree and $\mathfrak{A} = \langle A, <, \sqsubset, m\rangle$ an adjunction tree. We assume that r is the root of $\mathfrak{A}$ and that s is the unique leaf such that $m(r) = m(s)$. Now let x be a node of B such that $\ell(x) = m(r)$. Then the replacement of x by $\mathfrak{B}$ is defined by naming the colour functionals. These are

$$\begin{array}{ll}
\mathfrak{I}\mathfrak{I}_\rho(y, \sqsubset) := \begin{cases} \{\sqsubset, <\}, & \text{if } s \sqsubset y, \\ \{\sqsubset\}, & \text{otherwise.} \end{cases} & \mathfrak{O}\mathfrak{I}_\rho(y, \sqsubset) := \varnothing \\
\mathfrak{I}\mathfrak{O}_\rho(y, \sqsubset) := \begin{cases} \{<\}, & \text{if } y \sqsubset s, \\ \varnothing, & \text{otherwise.} \end{cases} & \mathfrak{O}\mathfrak{O}_\rho(y, \sqsubset) := \{\sqsubset\} \\
\mathfrak{I}\mathfrak{I}_\rho(y, <) := \{<\} & \mathfrak{I}\mathfrak{O}_\rho(y, <) := \varnothing \\
\mathfrak{O}\mathfrak{I}\rho(y, <) := \varnothing & \mathfrak{O}\mathfrak{O}_\rho(y, <) := \{<\}
\end{array}$$

Two things may be remarked. First, instead of a single start graph we have a finite set of them. This can be remedied by standard means. Second, all vertex colours are terminal as well as nonterminal. One may end the derivation at any given moment. We have noticed in connection with grammars for strings that this can be remedied. In actual fact, we have defined not context free γ–grammars but context free quasi γ–grammars*. However, we shall refrain from being overly pedantic. Suffice it to note that the adjunction grammars do not define the same kind of generative process if defined exactly as above.

Finally we shall give a graph grammar which generates all strings of the form $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$, $n > 0$. The idea for this grammar is

Figure 1.7: Tree Adjunction



due to Uwe Mönnich (see (Mönnich, 1999)). We shall exploit the fact that we may think of terms as structures. We posit a ternary symbol, $\mathtt{F}$, which is nonterminal, and another ternary symbol, $\mathtt{f}$, which is terminal. Further, there is a binary terminal symbol $^\frown$. The rules are as follows. (To enhance readability we shall not write terms in Polish Notation but by means of brackets.)

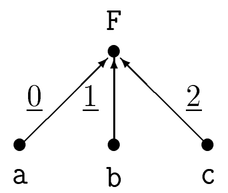
$$\begin{array}{lcl} \mathtt{F}(x,y,z) & \to & \mathtt{F}(\mathtt{a}^\frown x, \mathtt{b}^\frown y, \mathtt{c}^\frown z), \\ \mathtt{F}(x,y,z) & \to & \mathtt{f}(x,y,z). \end{array}$$

These rules constitute a so called **term replacement system**. The start term is $\mathtt{F}(\mathtt{a},\mathtt{b},\mathtt{c})$. Now suppose that $u \to v$ is a rule and that we have derived a term t such that u^σ occurs in t as a subterm. Then we may substitute this occurrence by v^σ. Hence we get the following derivations.

$$\begin{array}{llll} \mathtt{F}(\mathtt{a},\mathtt{b},\mathtt{c}) & \to \mathtt{f}(\mathtt{a},\mathtt{b},\mathtt{c}), & & \\ \mathtt{F}(\mathtt{a},\mathtt{b},\mathtt{c}) & \to \mathtt{F}(\mathtt{a}^\frown\mathtt{a},\mathtt{b}^\frown\mathtt{b},\mathtt{c}^\frown\mathtt{c}) & \to \mathtt{f}(\mathtt{a}^\frown\mathtt{a},\mathtt{b}^\frown\mathtt{b},\mathtt{c}^\frown\mathtt{c}) & \\ \mathtt{F}(\mathtt{a},\mathtt{b},\mathtt{c}) & \to \mathtt{F}(\mathtt{a}^\frown\mathtt{a},\mathtt{b}^\frown\mathtt{b},\mathtt{c}^\frown\mathtt{c}) & \to \mathtt{F}(\mathtt{a}^\frown(\mathtt{a}^\frown\mathtt{a}),\mathtt{b}^\frown(\mathtt{b}^\frown\mathtt{b}),\mathtt{c}^\frown(\mathtt{c}^\frown\mathtt{c})) & \\ & & \to \mathtt{f}(\mathtt{a}^\frown(\mathtt{a}^\frown\mathtt{a}),\mathtt{b}^\frown(\mathtt{b}^\frown\mathtt{b}),\mathtt{c}^\frown(\mathtt{c}^\frown\mathtt{c})) & \end{array}$$

Notice that the terms denote graphs here. We make use of the dependency coding. Hence the associated strings to these terms are $\mathtt{abc}$, $\mathtt{aabbcc}$ and $\mathtt{aaabbbccc}$.

In order to write a graph grammar which generates the graphs for these terms we shall have to introduce colours for edges. Put $F_E := \{\underline{0}, \underline{1}, \underline{2}, \sqsubset, <\}$, $F_V := \{\mathtt{F},\mathtt{f},\mathtt{a},\mathtt{b},\mathtt{c}\}$, and $F_V^T := \{\mathtt{f},\mathtt{a},\mathtt{b},\mathtt{c}\}$. The start graph is as follows. It has four vertices, p, q, r and s. ($<$ is empty (!), and $q \sqsubset r \sqsubset s$.) The labelling is $p \mapsto \mathtt{F}$, $q \mapsto \mathtt{a}$, $r \mapsto \mathtt{b}$ and $s \mapsto \mathtt{c}$.



There are two rules of replacement. The first can be written schematically as follows. The root, x, carries the label $\mathtt{F}$ and has three incoming edges; their colours are $\underline{0}$, $\underline{1}$ and $\underline{2}$. These come from three disjoint subgraphs, $\mathfrak{G}_0$, $\mathfrak{G}_1$ and $\mathfrak{G}_2$, which are ordered trees with respect to $<$ and $\sqsubset$ and in which there are no edges with colour $\underline{0}$, $\underline{1}$ and $\underline{2}$. In replacement, x is replaced by a graph consisting of seven vertices, p, q_i, r_i and s_i, $i < 2$, where $q_i \sqsubset r_j \sqsubset s_k$, $i, j, k < 2$, and $q \xrightarrow{0} p$, $r \xrightarrow{1} p$ and $s \xrightarrow{2} p$. $<= \{\langle q_1, q_0\rangle, \langle r_1, r_0\rangle, \langle s_1, s_0\rangle\}$. The colouring is

$$\begin{array}{lll@{\qquad}lll}
p & \mapsto & \mathtt{F}, & & & \\
q_0 & \mapsto & {}^\frown, & q_1 & \mapsto & \mathtt{a}, \\
r_0 & \mapsto & {}^\frown, & r_1 & \mapsto & \mathtt{b}, \\
s_0 & \mapsto & {}^\frown, & s_1 & \mapsto & \mathtt{c}.
\end{array}$$

(We reproduce with $\{p, q_0, r_0, s_0\}$ the begin situation.) The tree $\mathfrak{G}_0$ is attached to q_0 to the right of q_1, the tree $\mathfrak{G}_1$ is attached to r_0 to the right of r_1 and the tree $\mathfrak{G}_2$ is attached to s_0 to the right of s_1. Additionally, we put $x < p$ for all vertices x of the $\mathfrak{G}_i$. (So, the edge $\langle x, p\rangle$ has colour $<$ for all such x.) By this we see to it that in each step the union of the relations $<$, $\underline{0}$, $\underline{1}$ and $\underline{2}$ is the intended tree ordering and that there always exists an ingoing edge with colour $\underline{0}$, $\underline{1}$ and $\underline{2}$ into the root.

The second replacement rule replaces the root by a one vertex graph with label $\mathtt{f}$ at the root. This terminates the derivation. The edges with label $\underline{0}$, $\underline{1}$ and $\underline{2}$ are transmitted under the name $<$. This completes the tree. It has the desired form.

Exercise 33. Strings can also be viewed as multigraphs with only one edge colour. Show that a context free grammar for strings can also be defined as a context free γ–grammar on strings. We shall see in the next chapter that context free grammars can also be defined by unregulated tree adjunction grammars, but that the converse does not hold.

Exercise 34. Show that for every context free γ–grammar Γ there exists a context free γ–graph grammar Δ which has no rules of productivity -1 and which generates the same class of graphs.

Exercise 35. Show that for every context free γ–grammar there exists a context free γ–grammar with the same yield and no rules of productivity ≤ 0.

Exercise 36. Define in analogy to the unregulated tree adjunction grammars unregulated string adjunction grammars. Take note of the fact that these are quasi–grammars. Characterize the class of strings generated by these grammars in terms of ordinary grammars.

Exercise 37. Show that the language $\{\vec{w} \cdot \vec{w} : \vec{w} \in A^*\}$ is not context free but that it satisfies the Pumping Lemma.

1.7 Turing machines

We owe to (Turing, 1936) and (Post, 1936) the concept of a machine which is very simple and nevertheless capable of computing all functions that we know are computable. Without going into the details of what makes a function computable, it is nowadays agreed that there is no loss if we define 'computable' to mean *computable by a Turing machine*. The essential idea was that computation on numbers can be replaced by computation on strings. The number n can for example be represented by $n + 1$ successive strokes on a piece of paper. (So, the number 0 is represented by a single stroke. This is really necessary.) In addition to the stroke we have a blank, which is used to separate different numbers. The Turing machine, however powerful, takes a lot of time to compute even the most basic functions. Hence we agree from the start that it has an arbitrary, finite stock of symbols that it can use in addition to the blank. A Turing machine is a physical device, consisting of a tape which is infinite in both directions. That is, it contains cells numbered by the set of integers. Each cell may carry a symbol from an alphabet A or a blank. The machine possesses a read and write head, which can move between the cells, one at a time.

Finally, it has finitely many states, and can be programmed in the following way. We assign instructions for the machine that tell it what to do on condition that it is in state α and reads a symbol a from the tape. These instruction tell the machine whether it should write a symbol, then move the head one step or leave it at rest, and subsequently change to a state α'.

Definition 1.7.1 *A* ***(nondeterministic) Turing machine*** *is a quintuple $\langle A, L, Q, q_0, f\rangle$, where A is a finite set, the* ***alphabet****, $L \notin A$ is the so called* ***blank****, Q a finite set, the set of* ***(internal) states****, $q_0 \in Q$ the* ***initial state*** *and $f : A_L \times Q \to \wp(A_L \times \{-1, 0, 1\} \times Q)$, called the* ***transition function****. If for all $b \in A_L$ and $q \in Q$ $|f(b, q)| \leq 1$, the machine is called* ***deterministic****.*

Here, we have written A_L in place of $A \cup \{L\}$. Often, we use `L` or even $\Box$ as particular blanks. What this describes physically is a machine that has a two–sided infinite tape (which we can think of as a function $\tau : \mathbb{Z} \to A_L$), with a read/write head positioned on one of the cells. A **computation step** is as follows. Suppose the machine scans the symbol a in state q and is on cell $i \in \mathbb{Z}$. Then if $\langle b, 1, q'\rangle \in f(a, q)$, the machine writes b in place of a, advances to cell $i + 1$ and changes to state q'. If $\langle b, 0, q'\rangle \in f(a, q)$ the machine writes b in place of a, stays in cell i and changes to state q'. Finally, if $\langle b, -1, q'\rangle \in f(a, q)$, the machine writes b in place of a, moves to cell $i - 1$ and switches to state q'. Evidently, in order to describe the process we need (i) the tape, (ii) the position of the head of that tape, (iii) the state the machine is currently in. We assume throughout that the tape is almost everywhere filled by a blank. (The locution 'almost all' of 'almost everywhere' is often used in place 'all but finitely many'.) This means that the content of the tape plus the information on the machine may be coded by a single string, called *configuration*. Namely, if the tape is almost everywhere filled by a blank, there is a unique interval $[m, n]$ which contains all non–blank squares and the head of the machine. Suppose that the machine head is on Tape ℓ. Then let $\vec{x}_1$ be the string defined by the interval $[m, \ell - 1]$ (it may be empty), and $\vec{x}_2$ the string defined by the interval $[\ell, n]$. Finally, assume that the machine is in state q. Then the string $\vec{x}_1 \cdot q \cdot \vec{x}_2$ is the configuration corresponding to that phyical configuration. So, the state of the machine is simply written behind the symbol of the cell that is being scanned.

Definition 1.7.2 *Let* $T = \langle A, \mathtt{L}, Q, q_0, f\rangle$ *be a Turing machine. A* T–**configuration** *is a string* $\vec{x}q\vec{y} \in A_L^* \times Q \times A_L^*$ *such that* $\vec{x}$ *does not begin and* $\vec{y}$ *does not end with a blank.*

This configuration corresponds to a situation, where the tape is almost empty (that is, almost all occurrences of symbols on it are blanks). The nonempty part is a string $\vec{x}$, with the head being placed somewhere behind the prefix $\vec{u}$. Since $\vec{x} = \vec{u}\vec{v}$ for some $\vec{v}$, we insert the state the machine is in between $\vec{u}$ and $\vec{v}$. The configuration omits most of the blanks, whence we have agreed that $\vec{u}q\vec{v}$ is the same configuration as $\Box\vec{u}q\vec{v}$ and the same $\vec{u}q\vec{v}\Box$.

We shall now describe the working of the machine using configurations. We say, $\vec{x} \cdot q \cdot \vec{y}$ is **transformed by T in one step into** $\vec{x}' \cdot q' \cdot \vec{y}'$ and write $\vec{x} \cdot q \cdot \vec{y} \vdash_T \vec{x}' \cdot q' \cdot \vec{y}'$ if one of the following holds.

1. $\vec{x}' = \vec{x}$, and for some $\vec{v}$ and b and c we have $\vec{y} = b \cdot \vec{v}$ and $\vec{y}' = c \cdot \vec{v}$, as well as $\langle c, 0, q'\rangle \in f(b, q)$.
2. We have $\vec{x}' = \vec{x} \cdot c$ and $\vec{y} = b \cdot \vec{y}'$ as well as $\langle c, 1, q'\rangle \in f(b, q)$.
3. We have $\vec{x} = \vec{x}' \cdot c$ and $\vec{y}' = b \cdot \vec{y}$ as well as $\langle c, -1, q'\rangle \in f(b, q)$.

Now, for T–configurations Z and Z' we define $Z \vdash_T^n Z'$ inductively by (a) $Z \vdash_T^0 Z'$ if and only if $Z = Z'$ and (b) $Z \vdash_T^{n+1} Z'$ if and only if for some Z'' we have $Z \vdash_T^n Z'' \vdash_T Z'$.

It is easy to see that we can define a semi Thue system on configurations that mimicks the computation of T. The canonical Thue system, $C(T)$, is shown in Table 1.2. (x and y range over A_L and q and q' over Q.) Notice that we have to take care not to leave a blank at the left and right end of the strings. This is why

Table 1.2: The Canonical Thue System

$$
\begin{array}{rll}
C(T) := & & \{\langle \vec{u}qx\vec{v}, \vec{u}yq'\vec{v}\rangle : \langle y, 1, q'\rangle \in f(x, q); \vec{u} \neq \varepsilon \text{ or } y \in A; \\
& & \qquad\qquad\qquad\qquad \vec{v} \neq \varepsilon \text{ or } x \in A\} \\
& \cup & \{\langle \vec{u}q, \vec{u}yq'\rangle : \langle y, 1, q'\rangle \in f(\Box, q); \vec{u} \neq \varepsilon \text{ or } y \in A\} \\
& \cup & \{\langle qx\vec{v}, q'\vec{v}\rangle : \langle \Box, 1, q'\rangle \in f(x, q); \vec{v} \neq \varepsilon \text{ or } x \in A\} \\
& \cup & \{\langle q, q'\rangle : \langle \Box, \alpha, q'\rangle \in f(\Box, q), \alpha \in \{-1, 0, 1\}\} \\
& \cup & \{\langle \vec{u}xq\vec{v}, \vec{u}q'y\vec{v}\rangle : \langle y, -1, q'\rangle \in f(x, q); \\
& & \qquad\qquad \vec{u} \neq \varepsilon \text{ or } x \in A; \vec{v} \neq \varepsilon \text{ or } y \in A\} \\
& \cup & \{\langle q\vec{v}, q'y\vec{v}\rangle : \langle y, -1, q'\rangle \in f(\Box, q); \vec{v} \neq \varepsilon \text{ or } y \in A\} \\
& \cup & \{\langle \vec{u}xq, \vec{u}q'\rangle : \langle \Box, -1, q'\rangle \in f(x, q); \vec{u} \neq \varepsilon \text{ or } x \in A\} \\
& \cup & \{\langle \vec{u}qx\vec{v}, \vec{u}q'y\vec{v}\rangle : \langle y, 0, q'\rangle \in f(x, q); \\
& & \qquad\qquad\qquad\qquad \vec{v} \neq \varepsilon \text{ or } x, y \in A\}
\end{array}
$$

the definition is more complicated than expected. The alphabet of the semi Thue system is $(Q \cup A_L)^*$. The following is easily shown by induction.

Proposition 1.7.3 *Let T be a Turing machine, $C(T)$ be its associated semi Thue system. Then for all T–configurations Z and Z': $Z \vdash_T^n Z'$ if and only if $Z \Rightarrow_{C(T)}^n Z'$. Moreover, if Z is a T–configuration and $Z \Rightarrow_{C(T)}^n \vec{u}$ for an arbitrary string $\vec{u} \in (Q \cup A_L)^*$, then $\vec{u}$ is a T–configuration and $Z \vdash_T^n \vec{u}$.*

Of course, the semi Thue system defines transitions on strings that are not configurations, but this is not relevant for the theorem.

Definition 1.7.4 *Let T be a Turing machine, Z a configuration and $\vec{x} \in A^*$. Z is called an **end configuration** if there is no configuration Z' such that $Z \vdash_T Z'$. T **accepts** $\vec{x}$ if there is an end configuration Z such that $q_0 \cdot \vec{x} \vdash_T^* Z$. The **language accepted by** T, $L(T)$, is the set of all strings from A^* which are accepted by T.*

It takes time to get used to the concept of a Turing machine and the languages that are accepted by such machines. We suggest to the interested reader to play a little while with these machines and see if he can program them to compute a few very easy functions. A first example is the machine which computes the successor function on binary strings. Assume our alphabet is $\{\mathtt{0}, \mathtt{1}\}$. We want to build a machine which computes the next string for $\vec{x}$ in the numerical encoding (see Section 1.2 for its definition). This means that if the machine starts with $q_0 \cdot \vec{x}$ it shall halt in the configuration $q_0 \cdot \vec{y}$ where $\vec{y}$ is the word immediately following $\vec{x}$ in the numerical ordering. (If in the sequel we think of numbers rather than strings we shall simply think instead of the string $\vec{x}$ of the number n, where $\vec{x}$ occupies the nth place in the numerical ordering.)

How shall such a machine be constructed? We need four states, q_i, $i < 4$. First, the machine advances the head to the right end of the string, staying in q_0 until it reads $\Box$. Finally, when it hits $\Box$, it changes to state q_1 and starts moving to the left. As long as it reads $\mathtt{1}$, it changes $\mathtt{1}$ to $\mathtt{0}$ and continues in state q_1, moving to the left. When it hits $\mathtt{0}$, it replaces it by $\mathtt{1}$, moves left and changes to state q_2. When it sees a blank, that blank is filled by $\mathtt{0}$ and the machine changes to state q_3, the final state. In q_2, the machine simply keeps moving leftwards until it hits a blanks and then stops in state q_3. The machine is shown in Table 1.3. (If you want a machine that computes the successor in the binary encoding, you have to replace Line 6 by $\Box \mapsto \langle \mathtt{1}, -1, q_3 \rangle$.) In recursion theory the notions of computability are defined for functions on the set of natural numbers. By means of the function Z, which is bijective, these notions can be transferred to functions on strings.

Definition 1.7.5 *Let A and B be alphabets. Further, let $f : A^* \to B^*$ be a function. f is called* ***computable*** *if the exists a deterministic Turing machine T such that for every $\vec{x} \in A^*$ there is a $q_t \in Q$ such that $q_0 \cdot \vec{x} \vdash_T^* q_t \cdot f(\vec{x})$ and $q_t \cdot f(\vec{x})$ is an end configuration. Let $L \subseteq A^*$. L is called* ***recursively enumerable*** *if $L = \varnothing$ or there is a computable function $f : \{\mathtt{0}, \mathtt{1}\}^* \to A^*$ such*

Table 1.3: The Successor Machine

$$\begin{array}{llcl}
q_0 & \mathtt{0} & \mapsto & \langle \mathtt{0}, 1, q_0 \rangle \\
 & \mathtt{1} & \mapsto & \langle \mathtt{1}, 1, q_0 \rangle \\
 & \square & \mapsto & \langle \square, -1, q_1 \rangle \\
q_1 & \mathtt{0} & \mapsto & \langle \mathtt{1}, -1, q_2 \rangle \\
 & \mathtt{1} & \mapsto & \langle \mathtt{0}, -1, q_1 \rangle \\
 & \square & \mapsto & \langle \mathtt{0}, -1, q_3 \rangle \\
q_2 & \mathtt{0} & \mapsto & \langle \mathtt{0}, -1, q_2 \rangle \\
 & \mathtt{1} & \mapsto & \langle \mathtt{1}, -1, q_2 \rangle \\
 & \square & \mapsto & \langle \square, 0, q_3 \rangle \\
q_3 & & &
\end{array}$$

that $f[\{\mathtt{0}, \mathtt{1}\}^*] = L$. *$L$ is* ***decidable*** *if both L as well as $A^* - L$ are recursively enumerable.*

Lemma 1.7.6 *Let $f : A^* \to B^*$ and $g : B^* \to C^*$ be computable functions. Then $g \circ f : A^* \to C^*$ is computable as well.*

The proof is a construction of a machine U from machines T and T' computing f and g, respectively. Simply write T and T' using disjoint sets of states, and then take the union of the transition functions. However, make the transition function of T first such that it changes to the starting state of T' as soon as the computation by T is finished.

Lemma 1.7.7 *Let $f : A^* \to B^*$ be computable and bijective. Then $f^{-1} : B^* \to A^*$ also is computable (and bijective).*

Write a machine that generates all strings of A^* in successive order (using the successor machine, see above), and computes $f(\vec{x})$ for all these strings. As soon as the target string is found, the machine writes $\vec{x}$ and deletes everything else.

Lemma 1.7.8 *Let A and B be finite alphabets. Then there are computable bijections $f : A^* \to B^*$ and $g : B^* \to A^*$ such that $f = g^{-1}$.*

In this section we shall show that the enumerable sets are exactly the sets which are accepted by a Turing machine. Further, we shall show that these are exactly the Type 0 languages. This establishes the first correspondence result between types of languages and types of automata. Following this we shall show that the recognition problem for Type 0 languages is in general not decidable. The proofs proceed by a series of reduction steps for Turing machines. First, we shall generalize the notion of a Turing machine. A k–**tape Turing machine** is a quintuple $\langle A, L, Q, q_0, f\rangle$ where A, L, Q, and q_0 are as before but now

$$f : A_L^k \times Q \to \wp(A_L^k \times \{-1, 0, 1\} \times Q) .$$

This means, intuitively speaking, that the Turing machine manipulates k tapes in place of a single tape. There is a read and write head on each of the tapes. In each step the machine can move only one of the heads. The next state depends on the symbols read on all the tapes plus the previous internal state. The initial configuration is as follows. All tapes except the first are empty. The heads are anywhere on these tapes (we may require them to be in position 0). On the first tape the head is immediately to the left of the input. The k–tape machine has an additional $k-1$ tapes to record intermediate results. The reader may verify that we may also allow such configurations as initial configurations in which the other tapes are filled with some finite string, with the tapes immediately to the left of it. This does not increase the recognition power. However, it may help in defining a machine computing a function of several variables. We may also allow that the information to the right of the head consists in a sequence of strings each separated by a blank (so that when two successive blanks follow the machine knows that the input is completely read). Again, there is a way to recode these machines using a basic multitape Turing machine, modulo computable functions. We shall give a little more detail concerning the fact that also k–tape Turing machines (in whatever of the discussed forms) cannot compute more functions that 1–tape machines. For this define the following coding of the k tapes onto a single tape. We shall group $2k$ cells together to a macro cell. The (micro) cell $2kp+2m$ corresponds to the entry on cell p on Tape m. The (micro) cell number $2kp+2,+1$ only contains information whether the head of the machine is placed on cell p on tape m. (Hence, every second micro cell is filled only with, say, either `1` or `0`.) Now given a k–tape Turing machine T, we shall define a machine U that simulates T under the given coding. This machine operates as follows. For a single step of T it scans the actual string for the positions of the read and write heads and remembers the symbols on which they are placed (they can be found in the adjacent cell). Remembering this information requires only finite amount of memory, and can be done using the internal states. The machine scans the tape again for the read or write head that will have to be changed in position. (To identify it, the machine must be able to do calculations modulo $2k$. Again finite memory is sufficient.) It changes its position (if needed) and the content of the adjacent cell (again if needed). Now it changes into the appropriate state. Notice that each step of T costs $2k \cdot |\vec{x}|$ time for U to simulate, where $\vec{x}$ is the longest string on the tapes. If there is an algorithm taking $f(n)$ steps to compute then the simulating machine needs at most $2k(f(n)+n)^2$ time to compute that same function under simulation. (Notice that in $f(n)$ steps the string(s) may acquire length at most $f(n)+n$.)

We shall use this to show that the nondeterministic Turing machines cannot compute more functions than the deterministic ones.

Proposition 1.7.9 *Let $L = L(T)$ for a Turing machine. Then there is a deterministic Turing machine U such that $S = L(U)$.*

Proof. Let $L = L(T)$. Choose a number such that $|f(q,x)| < b$ for all $q \in Q$, $x \in A$. We fix an ordering on $f(q,x)$ for all x and q. We construct a 3–tape machine V as follows. On the first tape we write the input $\vec{x}$. On the second tape we generate all sequences $\vec{p}$ of numbers $< b$. These sequences describe the action sequences of T. With each new sequence $\vec{p} = a_0 a_1 \ldots a_{q-1}$ we write onto the third tape $\vec{x}$ (which we copy straight from Tape 1) and let V work as follows. The head on Tape 2 is to the left of the sequence $\vec{a}$.

In the first step we shall follow the a_0th alternative for machine T on the 3rd tape and advance head number 2 one step to the right. In the second step we follow the alternative a_1 in the transition set of T and execute it on Tape 3. Then the head of Tape 2 is advanced one step to the right. If for some i the a_ith alternative does not exist, we exit the computation on Tape 3. In this way we execute on Tape 3 a single computation of T for the input. By this, the machine is completely deterministic. The machine V halts only if for some k it halts on some alternative sequences of length k. The reader may think about the machine can know that this is the case. □

Lemma 1.7.10 *L is recursively enumerable if and only if $L = L(T)$ for a Turing machine T.*

Proof. The case $L = \varnothing$ has to be dealt with separately. It is easy to construct a machine that halts on no word. This shows the equivalence in this case. Now assume that $L \neq \varnothing$. Let L be recursively enumerable. Then there exists a function $f : \{0, 1\}^* \to A^*$ such that $f[\{\mathtt{0}, \mathtt{1}\}^*] = L$ and a Turing machine U which computes f. Now we construct a (minimally) 3–tape Turing machine V as follows. The input $\vec{x}$ will be placed on the first tape. On the second tape V generates all strings $\vec{y} \in \{\mathtt{0}, \mathtt{1}\}^*$ starting with ε, in the lexicographical order. In order to do this we use the machine computing the successors in this ordering. If we have computed the string $\vec{y}$ on the second tape the machine computes the value $f(\vec{y})$ on the third tape. (Thus, we emulate machine T on the third tape, with input given on the second tape.) Since f is computable, V halts on Tape 3. Then it compares the string on Tape 3, $f(\vec{y})$ with $\vec{x}$. If they are equal, it halts, if not it computes the successor of $\vec{y}$ and starts the process over again. It is easy to see that $L = L(V)$. By the previous considerations, there is a one tape Turing machine W such that $L = L(W)$. Now conversely, let $L = L(T)$ for some Turing machine T. We wish to show that L is recursively enumerable. We may assume, by the previous theorem, that T is deterministic. We leave it to the reader to construct a machine U which computes a function $f : \{\mathtt{0}, \mathtt{1}\}^* \to A^*$ whose image is L. □

Theorem 1.7.11 *The following are equivalent.*

1. *L is of Type 0.*
2. *L is recursively enumerable.*
3. *$L = L(T)$ for a Turing machine T.*

Proof. We shall show (1) ⇒ (2) and (3) ⇒ (1). The theorem then follows with Lemma 1.7.10. Let L be of Type 0. Then there is a grammar $\langle \mathtt{S}, N, A, R\rangle$ which generates L. Now we have to construct a Turing machine which lists all strings that are derivable from $\mathtt{S}$. To this end we construct a machine with 3 tapes, of which the first contains the number n (in binary code, say) and which on Tape 2 first writes $\mathtt{S}$ and then applies in succession all possible rules to the string on Tape 2, keeping all terminal strings on that tape. If the tape contains n strings, the computation stops there. The last string is the desired result. Now, we shall have to make sure that the machine is capable to list all derivable strings from $\mathtt{S}$. To this end, it takes a string and adds to its left a symbol $\flat$. This symbol denotes the left edge of a domain for a rule instance. If immediately to the right of $\flat$ some left hand side of a production occurs, all possible rules are applied, the results written down. The previous string remains, but $\flat$ is advanced to the right on that string. Properly administrated this makes sure that all derivations are listed. Now let $L = L(T)$ for some Turing machine. Chose the following grammar G: in addition to the alphabet let $\mathtt{X}$ be the start symbol, $\mathtt{0}$ and $\mathtt{1}$ two nonterminals, and $\mathtt{Y}_q$ a nonterminal for each $q \in Q$. The rules are as follows.

$$\begin{array}{lcll}
\mathtt{X} & \to & \mathtt{X0} \mid \mathtt{X1} \mid \mathtt{Y}_{q_0} & \\
\mathtt{Y}_q b & \to & c\mathtt{Y}_r & \text{if } \langle c, 1, r\rangle \in f(b, q) \\
\mathtt{Y}_q b & \to & \mathtt{Y}_r c & \text{if } \langle c, 0, r\rangle \in f(b, q) \\
a\mathtt{Y}_q b & \to & \mathtt{Y}_r cb & \text{if } \langle c, -1, r\rangle \in f(b, q) \\
\mathtt{Y}_q b & \to & b & \text{if } f(b, q) = \varnothing
\end{array}$$

Starting with $\mathtt{X}$ this grammar generates strings of the form $\mathtt{Y}_{q_0}\vec{x}$, where $\vec{x}$ is a binary string. This codes the input for T. The additional rules code in a transparent way the computation of T on the string. If the computation stops, it is allowed to eliminate $\mathtt{Y}_q$. If the string is terminal it will be generated by G. In this way it is seen that $L(G) = L(T)$. □

Now we shall derive an important fact, namely that there exist undecidable languages of Type 0. We first of all note that Turing machines can be regarded as semi Thue systems, as we have done earlier. Now one can design a machine U which takes two inputs, one being the code of a Turing machine T and the other a string $\vec{x}$, and U computes what T computes on $\vec{x}$. Such a machine is called a **universal Turing machine**. The coding of Turing machines can be done as follows. We only use the letters $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$, which are, of course, also contained in the alphabet B. Let $A = \{\mathtt{a}_i : i < n\}$. Then let $\gamma(\mathtt{a}_i)$ be the number i in dyadic coding (over $\{\mathtt{a}, \mathtt{b}\}$, where $\mathtt{a}$ replaces $\mathtt{0}$ and $\mathtt{b}$ replaces $\mathtt{1}$). The number 0 is coded by $\mathtt{a}$ to distinguish it from ε. Furthermore, we associate the number n with the blank, $\mathtt{L}$. The states are coded likewise; we assume that $Q = \{0, 1, \dots, n-1\}$ for some n and that $q_0 = 0$. Now we still have to write down f. f is a subset of

$$A_L \times Q \times A_L \times \{-1, 0, 1\} \times Q$$

Each element $\langle a, q, b, m, r\rangle$ of f can be written down as follows.

$$\vec{x} \cdot \mathtt{c} \cdot \vec{u} \cdot \mathtt{c} \cdot \vec{\mu} \cdot \mathtt{c} \cdot \vec{y} \cdot \mathtt{c} \cdot \vec{v} \cdot \mathtt{c}$$

where $\vec{x} = \gamma(a)$, $\vec{u} = Z^{-1}(q)$, $\vec{y} = \gamma(b)$, $\vec{v} = Z^{-1}(r)$. Further, we have $\vec{\mu} = \mathtt{a}$ if $m = -1$, $\vec{\mu} = \mathtt{b}$ if $m = 0$ and $\vec{\mu} = \mathtt{ab}$ if $m = 1$. Now we simply write down f as a list, the entries being separated by cc. (This is not necessary, but is easier to handle.) We call the code of T $T^\spadesuit$. The set of all codes of Turing machines is decidable. (This is essential but not hard to see.) It should not be too hard to see that there is a machine U with two tapes, which for two strings $\vec{x}$ and $\vec{y}$ does the following. If $\vec{y} = T^\spadesuit$ for some T then U computes on $\vec{x}$ exactly as T does. If $\vec{y}$ is not the code of a machine, U moves into a special state and stops.

Suppose that there is a Turing machine V, which decides for given $\vec{x}$ and $T^\spadesuit$ wether or not $\vec{x} \in L(T)$. Now we construct a two tape machine W as follows. The input is $\vec{x}$, and it is given on both tapes. If $\vec{x} = T^\spadesuit$ for some T, then W computes T on $\vec{x}$. (This is done by emulating V.) If T halts on $\vec{x}$, we send W into an infinite loop. If T does not halt, W shall stop. (If $\vec{x}$ is not the code of a machine, the computation stops right away.) Now we have the following: $W^\spadesuit \in L(W)$ exactly if $W^\spadesuit \notin L(W)$. For $W^\spadesuit \in L(W)$ exactly when W stops if applied to $W^\spadesuit$. This however is the case exactly if W does *not* stop. If on the other hand $W^\spadesuit \notin L(W)$ then W does not stop if applied to $W^\spadesuit$, which we can decide with the help of machine V, and then W does halt on the input $W^\spadesuit$. Contradiction. Hence, V cannot exist. There is, then, no machine that can decide for any Turing machine (in code) and any input whether that machine halts on that string. It is still conceivable that this is decidable for every T, but we that we simply do not know how to extract such an algorithm for given T. Now, in order to show that this too fails, we use the universal Turing machine U, in its single tape version. Suppose that $L(U)$ is decidable. Then we can decide universally whether U halts on $\vec{x}\mathtt{L}T^\spadesuit$. Since U is universal, this means that we can decide for given T and given $\vec{x}$ whether T halts on $\vec{x}$. We have seen above that this is impossible.

Theorem 1.7.12 (Markov, Post) *There exists a recursively enumerable set which is not decidable.*

So we also shown that the Type 1 languages are properly contained in the Type 0 languages. For it turns out that the Type 1 languages are all decidable.

Theorem 1.7.13 (Chomsky) *Every Type 1 language is decidable.*

Proof. Let G be of Type 1 and let $\vec{x}$ be given. Put $n := |\vec{x}|$ and $\alpha := |A \cup N|$. If there is a derivation of $\vec{x}$ that has length $> \alpha^n$, there is a string that occurs twice in it, since all occurring strings must have length $\leq n$. Then there exists a shorter derivation for $\vec{x}$. So, $\vec{x} \in L(G)$ if and only if it has a G–derivation of length $\leq \alpha^n$. This is decidable. $\Box$

Corollary 1.7.14 $\mathrm{CSL} \subsetneq \mathrm{GL}$.

Chomsky credits in (Chomsky, 1959) Hilary Putnam with the observation that not all decidable languages are of Type 1. Actually, we can give a characterization of context sensitive languages as well. Say that a Turing machine is **linearly space bounded** if given input $\vec{x}$ it may use only $O(|\vec{x}|)$ on each of its tapes. Then the following holds.

Theorem 1.7.15 (Landweber, Kuroda) *A language L is context sensitive if and only if $L = L(T)$ for some linear space bounded Turing machine T.*

The proof can be assembled from Theorem 1.5.9 and the proof of Theorem 1.7.11.

We briefly discuss so called *word problems.* Recall from Section 1.5 the definition of a Thue process T. Let A be an alphabet. Consider the monoid $\mathfrak{Z}(A)$. The set of pairs $\langle s, t\rangle \in A^* \times A^*$ such that $s \Rightarrow_T^* t$ is a congruence on $\mathfrak{Z}(A)$. Denote the factor algebra by $\mathfrak{Mon}(T)$. (One calls the pair $\langle A, T\rangle$ a **presentation** of $\mathfrak{Mon}(T)$.) It can be shown to be undecidable whether $\mathfrak{Mon}(T)$ is the one element monoid. From this one deduces that it is undecidable whether or not $\mathfrak{Mon}(T)$ is a finite monoid, whether it is isomorphic to a given finite monoid, and many more.

Before we close this chapter we shall introduce a few measures for the complexity of computations. In what is to follow we shall often have to deal with questions of how fast and with how much space a Turing machine can compute a given problem. Let $f : \omega \to \omega$ be a function, T a Turing machine which computes a function $g : A^* \to B^*$. We say that T needs $O(f)$–**space** if there is a constant c such that for all but finitely many $\vec{x} \in A^*$ there is a computation of an accepting configuration $q_t \cdot g(\vec{x})$ from $q_0 \cdot \vec{x}$ in which every configuration has length $\leq c \times f(|\vec{x}|)$. For a multi tape machine we simply add the lengths of all words on the tapes. We say that T needs $O(f)$–**time** if for almost all $\vec{x} \in A^*$ there is a $k \leq c \times f(|\vec{x}|)$ such that $q_0 \cdot \vec{x} \vdash_T^k q_0 \cdot g(\vec{x})$. We denote by $\mathbf{DSPACE}(f)$ ($\mathbf{DTIME}(f)$) the set of all functions which for some k are computable by a deterministic k–tape Turing machine in $O(f)$–space ($O(f)$–time). Analogously the notation $\mathbf{NSPACE}(f)$ and $\mathbf{NTIME}(f)$ is defined for nondeterministic machines. We always have

$$\mathbf{DTIME}(f) \subseteq \mathbf{NTIME}(f) \subseteq \mathbf{NSPACE}(f)$$

as well as

$$\mathbf{DSPACE}(f) \subseteq \mathbf{NSPACE}(f) .$$

For a machine can write into at most k cells in k steps of computation, regardless of whether it is deterministic or nondeterministic. This applies as well for multi tape machines, since they can only write one cell and move one head at a time.

The reason for not distinguishing between the time complexity $f(n)$ and the $cf(n)$ (c a constant) is the following result.

Theorem 1.7.16 (Speed Up Theorem) *Let f be a computable and let T be a Turing machine which computes $f(\vec{x})$ in at most $g(|\vec{x}|)$ steps (using at most $h(|\vec{x}|)$ cells) where $\inf_{n\to\infty} g(n)/n = \infty$. Further, let c be an arbitrary real number > 0. Then there exists a Turing machine U which computes f in at most $c \cdot g(|\vec{x}|)$ steps (using at most $c \cdot h(|\vec{x}|)$ cells).*

The proof results from the following fact. In place of the original alphabet A_L we may introduce a new $B_L = A \cup B \cup \{L_1\}$, where each symbol from B corresponds to a sequence of length k of symbols from A_L. The symbol L_1 then corresponds to L^k. The alphabet A_L is still used for giving the input. The new machine upon receiving $\vec{x}$ recodes the input and calculates completely inside B_L.

Since to each single letter corresponds a block of k letters in the original alphabet, the space requirement shrinks by the factor k. (However, we need to ignore the length of the input.) Likewise, the time is cut by a factor k, since one move of the head simulates up to k moves. However, the exact details are not so easy to sum up. They can be found in (Hopcroft and Ullman, 1969).

Typically, one works with the following complexity classes.

Definition 1.7.17 ***PTIME*** *is the class of functions computable in deterministic polynomial time,* ***NP*** *the class of functions computable in nondeterministic polynomial time.* ***PSPACE*** *is the class of functions computable in polynomial space,* ***EXPTIME*** *(**NEXPTIME**) the class of functions computable in deterministic (nondeterministic) exponential time.*

Definition 1.7.18 *A language $L \subseteq A^*$ is in a complexity class $\mathcal{P}$ if and only if $\chi_L \in \mathcal{P}$.*

Notes on this section. In the mid 1930s, several people have independently studied the notion of feasibility. Alonzo Church and Stephen Kleene have defined the notion of λ–definablity and of a general recursive function, Emil Post and Alan Turing the notion of computability by a certain machine, now called the Turing machine. All three notions can be shown to identify the same class of functions, as these people have subsequently shown. It is known as Church's Thesis that these are all the functions that humans can compute, but for the purpose of this book it is irrelevant whether it is correct. We shall define the λ–calculus later in Chapter 3, without going into the details alluded to here, however. It is to be kept in mind that the Turing machine is a physical device. Hence, its computational capacities depend on the structure of the space–time continuum. This is not any more a speculation. Quantum computing exploits the different physical behaviour of quantum physics to do parallel computation. This radically changes the time complexity of problems (see (Deutsch *et al.*, 2000)). This asks us to be cautious not to attach too much significance to complexity theoretical results in connection with human behaviour since we do not know too well how the brain functions.

Exercise 38. Construct a Turing machine which computes the lexicographic predecessor of a string, and which returns ε for input ε.

Exercise 39. Construct a Turing machine which, given a list of strings (each string separated from the next by a single blank), moves the first string onto the end of the list.

Exercise 40. Let T be a Turing machine over the alphabet A. Show how to write a Turing machine over $\{\mathtt{0}, \mathtt{1}\}$ which computes the same partial function over A under a coding that assigns each letter of A a unique block of fixed length.

Exercise 41. In many definitions of a Turing machine the tape is only a one sided tape. Its cells can be numbered by natural numbers. This requires the introduction of a special symbol $\sharp$ that marks the left end of the tape, or of a predicate $\mathsf{left\text{-}end}$, which is true each time the head is at the left end of the tape. The transitions are different depending on whether the machine is at the left end of the tape or not. (There is an alternative, namely to stop the computation once that the left end is reached, but this is not recommended. Such a machine can compute only very uninteresting functions.) Show that for a Turing machine with a one sided tape there is a corresponding Turing machine our sense computing the same function, and that for each Turing machine in our sense there is a one sided machine computing the same function.

Exercise 42. Prove Lemma 1.7.8. *Hint.* Show first that it is enough to look at the case $|A| = 1$.

Exercise 43. A set $L \subseteq A^*$ shall be called **computable** if its characteristic function $\chi_L : A^* \to \{\mathtt{0}, \mathtt{1}\}$ is computable. Here, $\chi_L(\vec{x}) = 1$ if and only if $\vec{x} \in L$. Show that L is computable if and only if L is decidable.

Chapter 2

Context Free Languages

2.1 Regular Languages

Type 3 or regular grammars are the most simple grammars in the Chomsky Hierarchy. There are several characterizations of regular languages: by means of finite state automata, by means of equations over strings, and by means of so called regular expressions. Before we begin, we shall develop a simple form for regular grammars. First, all rules of the form $X \to Y$ can be eliminated. To this end, the new set of rules will be

$$\begin{array}{rcl} R^{\heartsuit} := & & \{X \to aY : X \vdash_G aZ\} \\ & \cup & \{X \to \vec{x} : X \vdash_G \vec{x}, \vec{x} \in A_\varepsilon\} \end{array}$$

It is easy to show that the grammar with $R^{\heartsuit}$ in place of R generates the same strings. We shall introduce another simplification. For each $a \in A$ we introduce a new nonterminal U_a. In place of the rules $X \to a$ we now add the rules $X \to aU_a$ as well as $U_a \to \varepsilon$. Now every rule with the exception of $U_a \to \varepsilon$ are strictly expanding. This grammar is therefore not regular if $\varepsilon \in L(G)$ but it generates the same language. However, the last kind of rules can be used only once, at the end of the derivation. For the derivable strings all have the form $\vec{x} \cdot Y$ with $\vec{x} \in A^*$ and $Y \in N$. If one applies a rule $Y \to \varepsilon$ then the nonterminal disappears and the derivation is terminated. We call a regular grammar **strictly binary** if there are only rules of the form $X \to aY$ or $X \to \varepsilon$.

Definition 2.1.1 *Let A be an alphabet. A* ***(partial) finite state automaton over*** *A is a quadruple $\mathfrak{A} = \langle Q, i_0, F, \delta\rangle$ such that $i_0 \in Q$, $F \subseteq Q$ and $\delta : Q \times A \to \wp(Q)$. Q is the set of* ***states****, i_0 is called* ***the initial state****, F the set of* ***accepting states*** *and δ the* ***transition function****. $\mathfrak{A}$ is called* ***deterministic*** *if $\delta(q, a)$ contains exactly one element for each $q \in Q$ and $a \in A$.*

δ can be extended to sets of states and strings in the following way ($S \subseteq Q$, $a \in A$).

$$\begin{array}{lll} \delta(S, \varepsilon) & := & S \\ \delta(S, a) & := & \bigcup\langle \delta(q, a) : q \in S\rangle \\ \delta(S, \vec{x} \cdot a) & := & \delta(\delta(S, \vec{x}), a) \end{array}$$

With this defined, we can now define the accepted language.

$$L(\mathfrak{A}) = \{\vec{x} : \delta(\{i_0\}, \vec{x}) \cap F \neq \varnothing\} \ .$$

$\mathfrak{A}$ is strictly partial if there is a state q and some $a \in A$ such that $\delta(q, a) = \varnothing$. An automaton can always be transformed into an equivalent automaton which is not partial. Just add another state $q_\varnothing$ and add to the transition function the following transitions.

$$\delta^+(q, a) := \begin{cases} \delta(q, a) & \text{if} \quad \delta(q, a) \neq \varnothing, q \neq q_\varnothing, \\ q_\varnothing & \text{if} \quad \delta(q, a) = \varnothing, q \neq q_\varnothing, \\ q_\varnothing & \text{if} \quad q = q_\varnothing. \end{cases}$$

Furthermore, $q_\varnothing$ shall *not* be an accepting state. In the case of a deterministic automaton we have $\delta(q, \vec{x}) = \{q'\}$ for some q'. In this case we think of the transition function as yielding states from states plus strings, that is, we now have $\delta(q, \vec{x}) = q'$. Then the definition of the language of an automaton $\mathfrak{A}$ can be refined as follows.

$$L(\mathfrak{A}) = \{\vec{x} : \delta(i_0, \vec{x}) \in F\} \ .$$

This is as general as we need it. For we can, for every given automaton, write down a deterministic automaton that accepts the same language. Put

$$\mathfrak{A}^d := \langle \wp(Q), \{i_0\}, F^d, \delta \rangle ,$$

where $F^d := \{G \subseteq Q : G \cap F \neq \varnothing\}$ and δ is the transition function of $\mathfrak{A}$ extended to sets of states.

Proposition 2.1.2 *$\mathfrak{A}^d$ is deterministic and $L(\mathfrak{A}^d) = L(\mathfrak{A})$. Hence every language accepted by a finite state automaton is a language accepted by a deterministic finite state automaton.*

The proof is straightforward and left as an exercise. Now we shall first show that a regular language is a language accepted by a finite state automaton. We may assume that G is (almost) strictly binary, as we have seen above. So, let $G = \langle S, N, A, R \rangle$. We put $Q_G := N$, $i_0 := S$, $F_G := \{X : X \to \varepsilon \in R\}$ as well as

$$\delta_G(X, a) := \{Y : X \to aY \in R\} .$$

Now put $\mathfrak{A}_G := \langle Q_G, i_0, F_G, \delta_G \rangle$.

Lemma 2.1.3 *For all $X, Y \in N$ and $\vec{x}$ of length > 0 we have $Y \in \delta(X, \vec{x})$ if and only if $X \Rightarrow_R \vec{x} \cdot Y$.*

Proof. Induction over the length of $\vec{x}$. The case $|\vec{x}| = \varepsilon$ is evident. Let $\vec{x} = a \in A$. Then $Y \in \delta_G(X, a)$ by definition if and only if $X \to aY \in R$, and from this we get $X \Rightarrow_R aY$. Conversely, from $X \Rightarrow_R aY$ follows that $X \to aY \in R$. For since the derivation uses only strictly expanding rules except for the last step, the derivation of aY from X must be the application of a single rule. This finishes the case of length 1. Now let $\vec{x} = \vec{y} \cdot a$. By definition of δ_G we have

$$\delta_G(X, \vec{x}) = \delta_G(\delta_G(X, \vec{y}), a) .$$

Hence there is a Z such that $Z \in \delta_G(X, \vec{y})$ and $Y \in \delta_G(Z, a)$. By induction hypothesis this is equivalent with $X \Rightarrow_R \vec{y} \cdot Z$ and

$Z \Rightarrow_R aY$. From this we get $X \Rightarrow_R \vec{y} \cdot a \cdot Y = \vec{x} \cdot Y$. Conversely, from $X \Rightarrow_R \vec{x} \cdot Y$ we get $X \Rightarrow_R \vec{y} \cdot Z$ and $Z \Rightarrow_R aY$ for some Z. This is so since the grammar is regular. Now, by induction hypothesis, $Z \in \delta_G(X, \vec{y})$ and $Y \in \delta_G(Z, a)$, and so $Y \in \delta_G(\vec{x}, X)$. $\Box$

Proposition 2.1.4 $L(\mathfrak{A}_G) = L(G)$.

Proof. It is easy to see that $L(G) = \{\vec{x} : G \vdash \vec{x} \cdot Y, Y \to \varepsilon \in R\}$. By Lemma 2.1.3 $\vec{x} \cdot Y \in L(G)$ if and only if $S \Rightarrow_R \vec{x} \cdot Y$. The latter is equivalent with $Y \in \delta_G(S, \vec{x})$. And this is nothing but $\vec{x} \in L(\mathfrak{A}_G)$. Hence $L(G) = L(\mathfrak{A}_G)$. $\Box$

If we now take a finite state automaton $\mathfrak{A} = \langle Q, i_0, F, \delta\rangle$ we can conversely define a grammar as follows. We put $N := Q$, $S := i_0$. R consists of all rules of the form $X \to aY$ where $Y \in \delta(X, a)$ as well as all rules of the form $X \to \varepsilon$ for $X \in F$. This defines the grammar $G_{\mathfrak{A}}$. It is strictly binary and we have $\mathfrak{A}_{G_{\mathfrak{A}}} = \mathfrak{A}$. Therefore we have $L(G_{\mathfrak{A}}) = L(\mathfrak{A})$.

Theorem 2.1.5 *The regular language are exactly those languages that are accepted by some deterministic finite state automaton.* $\Box$

Now we shall turn to a further characterization of regular languages. A **regular term over** A is a term which is composed from A with the help of the symbols 0 (0–ary), ε (0–ary), $\cdot$ (binary), $\cup$ (binary) and * (unary). A regular term defines a language over A as follows.

$$\begin{array}{lcl} L(0) & := & \varnothing \\ L(\varepsilon) & := & \{\varepsilon\} \\ L(a) & := & \{a\} \\ L(R \cdot S) & := & L(R) \cdot L(S) \\ L(R \cup S) & := & L(R) \cup L(S) \\ L(R^*) & := & L(R)^* \end{array}$$

(Commonly, one writes R in place of $L(R)$, a usage that we will follow in the sequel to this section.) Also, $R^+ := R^* \cdot R$ is an often used abbreviation. Languages which are defined by a regular term can also be viewed as solutions of some very simple system of equations. We introduce variables (say X, Y und Z) which are variables for subsets of A^* and we write down equations for the terms over these variables and the symbols 0, ε, a ($a \in A$), $\cdot$, $\cup$ and *. An example is the equation $X = \mathtt{b} \cup \mathtt{a}X$, whose solution is $X = \mathtt{a}^*\mathtt{b}$.

Lemma 2.1.6 *Assume $\varepsilon \notin L(R)$. Then R^* is the unique solution of*

$$X = \varepsilon \cup R \cdot X\ .$$

Proof. (The reader may reflect over the fact that this theorem also holds for $R = 0$. This is therefore now excluded.) The proof is by induction over the length of $\vec{x}$. $\vec{x} \in X$ means by definition that $\vec{x} \in \varepsilon \cup R \cdot X$. If $\vec{x} = \varepsilon$ then $\vec{x} \in R^*$. Hence let $\vec{x} \neq \varepsilon$; then $\vec{x} \in R \cdot X$ and so it is of the form $\vec{u}_0 \cdot \vec{x}_0$ where $\vec{u}_0 \in R$ and $\vec{x}_0 \in X$. Since $\vec{u}_0 \neq \varepsilon$, $\vec{x}_0$ has smaller length than $\vec{x}$. By induction hypothesis we therefore have $\vec{x}_0 \in R^*$. Hence $\vec{x} \in R^*$. The other direction is as easy. □

Lemma 2.1.7 *Let C, D be regular terms and $\varepsilon \notin L(D)$. The equation $X = C \cup D \cdot X$ has exactly one solution, namely $D^* \cdot C$.* □

We shall now show that regular languages can be seen as solutions of systems of equations. A general system of string equations is a set of equations of the form $X_j = Q \cup \bigcup_{i<m} T^i$ where Q is a regular term and the T^i have the form $R \cdot X_k$ where R is a regular term. Here is an example.

$$\begin{aligned} X_0 &= \mathtt{a}^* \cup \mathtt{c} \cdot \mathtt{a} \cdot \mathtt{b} \cdot X_1 \\ X_1 &= \mathtt{c} \cup \mathtt{c} \cdot \mathtt{b}^3 \cdot X_0 \end{aligned}$$

Notice that like in other systems of equations a variable need not occur to the right in every equation. Moreover, a system of equations contains any given variable only once on the left. The system is called **proper** if for all i and j we have $\varepsilon \notin L(T^i_j)$. We shall call a system of equations **simple** if it is proper and Q as well as the T^i_j consist only of terms made from elements of A using ε and $\cup$. The system displayed above is proper but not simple.

Let now $\langle S, N, A, R\rangle$ be a strictly binary regular grammar. Introduce for each nonterminal X a variable Q_X. This variable Q_X shall stand for the set of all strings which can be generated from X in this grammar, that is, all strings $\vec{x}$ for which $X \Rightarrow_R \vec{x}$. This latter set we denote by $[X]$. We claim that the Q_X so interpreted satisfy the following system of equations.

$$\begin{aligned} Q_Y = & \quad \textstyle\bigcup\{\varepsilon : Y \to \varepsilon \in R\} \\ & \cup \ \textstyle\bigcup\{a \cdot Q_X : Y \to aX \in R\} \end{aligned}$$

This system of equations is simple. We show $Q_Y = [Y]$ for all $Y \in N$. The proof is by induction over the length of the string. To begin, we show that $Q_Y \subseteq [Y]$. For let $\vec{y} \in Q_Y$. Then either $\vec{y} = \varepsilon$ and $Y \to \varepsilon \in R$ or we have $\vec{y} = a \cdot \vec{x}$ with $\vec{x} \in Q_X$ and $Y \to a \cdot \vec{x} \in R$. In the first case $Y \to \varepsilon \in R$, whence $\varepsilon \in [Y]$. In the second case $|\vec{x}| < |\vec{y}|$ and so by induction hypothesis $\vec{x} \in [X]$, hence $X \Rightarrow_R \vec{x}$. Then we have $Y \Rightarrow_R a \cdot \vec{x} = \vec{y}$, from which $\vec{y} \in [Y]$. This shows the first inclusion. Now we show that $[Y] \subseteq Q_Y$. To this end let $Y \Rightarrow_R \vec{y}$. Then either $\vec{y} = \varepsilon$ and so $Y \to \varepsilon \in R$ or $\vec{y} = a \cdot \vec{x}$ for some $\vec{x}$. In this case $\vec{y} \in Q_Y$, by definition. In the second case there must be an X such that $Y \to aX \in R$ and $X \Rightarrow_R \vec{x}$. Then $|\vec{x}| < |\vec{y}|$ and therefore by induction hypothesis $\vec{x} \in Q_X$. Finally, by definition of Q_Y, $\vec{y} \in Q_Y$, which had to be shown.

So, a regular language is the solution of a simple system of equations. Conversely, every simple system of equations can be rewritten into a regular grammar which generates the solution of this system. Finally, it remains to be shown that regular terms describe nothing but regular languages. What we shall establish is more general and derives the desired conclusion. We shall show that every proper system of equations which has as many equations as it has variables has as its solution for each variable a regular language. To this end, let such a system $X_j = \bigcup_{i<m_j} T_j^i$ be given. We begin by eliminating X_0 from the system of equations. We distinguish two cases. (1.) X_0 does appear in the equation $X_0 = \bigcup_{i<m_0} T_j^i$ only to the left. This equation is fixed, and called the **pivot equation for** X_0. Then a simple replacement of X_0 by the right hand sides of the other (!) equations does the trick. (2.) The equation is of the form $X_0 = C \cup D \cdot X_0$, C a regular term, which does not contain X_0, D free of variables and $\varepsilon \notin L(D)$. Then $X_0 = D^* \cdot C$ by Lemma 2.1.7. Now X_0 does not occur and we can replace X_0 in the other equations as in (1.). The system of equations that we get is not simple, even if it was simple at the beginning. We can proceed in this fashion und eliminate step by step the variables from the right hand side (and putting aside the corresponding pivot equations) until we reach the last equation. The solution for X_{n-1} does not contain any variables at all and is a regular term. The solution can be inserted into the other equations, and then we continue with X_{n-2}, then with X_{n-3}, and so on. As an example, we take the following system of equations.

$$
\begin{array}{llcrrr}
\text{(I)} & X_0 & = & \mathtt{a} \cup \mathtt{a} \cdot X_0 & \cup\, \mathtt{b} \cdot X_1 & \cup\, \mathtt{c} \cdot X_2 \\
& X_1 & = & \mathtt{c} \cdot X_0 & & \cup\, \mathtt{a} \cdot X_2 \\
& X_2 & = & \mathtt{b} \cup \mathtt{a} \cdot X_0 & \cup\, \mathtt{b} \cdot X_1 & \\
\\
\text{(II)} & X_0 & = & \mathtt{a}^+ & \cup\, \mathtt{a}^*\mathtt{b} \cdot X_1 & \cup\, \mathtt{a}^*\mathtt{c} \cdot X_2 \\
& X_1 & = & \mathtt{ca}^+ & \cup\, \mathtt{ca}^*\mathtt{b} \cdot X_1 & \cup\, (\mathtt{ca}^*\mathtt{c} \cup \mathtt{a}) \cdot X_2 \\
& X_2 & = & \mathtt{b} \cup \mathtt{aa}^+ & \cup\, (\mathtt{a}^+\mathtt{b} \cup \mathtt{b}) \cdot X_1 & \cup\, \mathtt{a}^*\mathtt{c} \cdot X_2
\end{array}
$$

$$
\begin{array}{llcl}
\text{(III)} & X_1 & = & (\mathtt{ca}^*\mathtt{b})^*\mathtt{ca}^+ \cup (\mathtt{ca}^*\mathtt{b})^*(\mathtt{ca}^*\mathtt{c} \cup \mathtt{a}) \cdot X_2 \\
& X_2 & = & (\mathtt{b} \cup \mathtt{aa}^+) \cup [\mathtt{a}^*\mathtt{b}(\mathtt{ca}^*\mathtt{b})^*(\mathtt{ca}^*\mathtt{c} \cup \mathtt{a}) \cup \mathtt{a}^*\mathtt{c}] \cdot X_2 \\
\\
\text{(IV)} & X_2 & = & [\mathtt{a}^*\mathtt{b}(\mathtt{ca}^*\mathtt{b})^*(\mathtt{ca}^*\mathtt{c} \cup \mathtt{a}) \cup \mathtt{a}^*\mathtt{c}]^*(\mathtt{b} \cup \mathtt{aa}^+)
\end{array}
$$

Theorem 2.1.8 (Kleene) *Let L be a language over A. Then the following are equivalent:*

1. *L is regular.*

2. *$L = L(\mathfrak{A})$ for a finite, deterministic automaton $\mathfrak{A}$ over A.*
3. *$L = L(R)$ for some regular term R over A.*
4. *L is the solution for X_0 of a simple system of equations over A with variables X_i, $i < m$.*

Further, there exist algorithms which (i) for a given automaton $\mathfrak{A}$ compute a regular term R such that $L(\mathfrak{A}) = L(R)$; (ii) for a given regular term R compute a simple system of equations Σ over $\vec{X}$ whose solution for a given variable X_0 is exactly $L(R)$; and (iii) which for a given simple system of equations Σ over $\{X_i : i < m\}$ compute an automaton $\mathfrak{A}$ such that $\vec{X}$ is its set of states and the solution for X_i is exactly the set of strings which send the automaton from state X_0 into X_i. □

This is the most important theorem for regular languages. We shall derive a few consequences. Notice that a finite state automaton is almost a Turing machine. What is missing is the accepting states. However, in case of nonacceptance we may send the machine into a loop. Otherwise, the machine halts at the first blank. Modulo accepting states, a Turing machine is a finite state automaton that may only move to the right and is not allowed to write. Therefore, the recognition problem for regular languages is in **DTIME**(n) and in **DSPACE**(n). This also applies to the parsing problem, as is easily seen.

Corollary 2.1.9 *The recognition and the parsing problem are in DTIME(n) and DSPACE(n).*

Corollary 2.1.10 *The set of regular languages over A is closed under intersection and relative complement. Further, for given regular terms R and S one can determine terms U and V such that $L(U) = A^* - L(R)$ and $L(V) = L(R) \cap L(S)$.*

Proof. It is enough to do this construction for automata. Using Theorem 2.1.8 it follows that we can do it also for the corresponding regular terms. Let $\mathfrak{A} = \langle Q, i_0, F, \delta\rangle$. Without loss of generality we may assume that $\mathfrak{A}$ is deterministic. Then let $\mathfrak{A}^- := \langle Q, i_0, Q - F, \delta\rangle$. We then have $L(\mathfrak{A}^-) = A^* - L(\mathfrak{A})$. This shows that for given $\mathfrak{A}$ we can construct an automaton which accepts the complement of $L(\mathfrak{A})$. Now let $\mathfrak{A}' = \langle Q', i_0', F', \delta'\rangle$. Put $\mathfrak{A} \times \mathfrak{A}' = \langle Q \times Q', \langle i_0, i_0'\rangle, F \times F', \delta \times \delta'\rangle$, where

$$(\delta \times \delta')(\langle q, q'\rangle, a) := \{\langle r, r'\rangle : r \in \delta(q, a), r' \in \delta'(q', a)\} .$$

It is easy to show that $L(\mathfrak{A} \times \mathfrak{A}') = L(\mathfrak{A}) \cap L(\mathfrak{A}')$. □
The proof of the next theorem is an exercise.

Theorem 2.1.11 *Let L and M be regular languages. Then so are L/M und $M\backslash L$. Likewise, with L also L^T, $L^P := L/A^*$ as well as $S^S := A^*\backslash S$ are regular.*

Furthermore, the following important consequence can be established.

Theorem 2.1.12 *Let $\mathfrak{A}$ and $\mathfrak{B}$ be finite state automata. Then the problem '$L(\mathfrak{A}) = L(\mathfrak{B})$' is decidable.*

Proof. Let $\mathfrak{A}$ and $\mathfrak{B}$ be given. By Theorem 2.1.8 we can compute a regular term R with $L(R) = L(\mathfrak{A})$ as well as a regular term S with $L(S) = L(\mathfrak{B})$. Then $L(\mathfrak{A}) = L(\mathfrak{B})$ if and only if $L(R) = L(S)$ if and only if $(L(R) - L(S)) \cup (L(S) - L(R)) = \varnothing$. By Corollary 2.1.10 we can compute a regular term U such that $L(U) = (L(R) - L(S)) \cup (L(S) - L(R))$. Hence $L(\mathfrak{A}) = L(\mathfrak{B})$ if and only if $L(U) = \varnothing$. This is decidable by Lemma 2.1.13. □

Lemma 2.1.13 *The problem '$L(R) = \varnothing$', where R is a regular term, is decidable.*

Proof. By induction on R. If $R = \varepsilon$ or $R = a$ then $L(R) \neq \varnothing$. If $R = 0$ then by definition $L(R) = \varnothing$. Now assume that the problems '$L(R) = \varnothing$' and '$L(S) = \varnothing$' are decidable. Then $L(R \cup S) = \varnothing$ if and only if $L(R) = \varnothing$ as well as $L(S) = \varnothing$; this is decidable. Furthermore, $L(R \cdot S) = \varnothing$ is decidable if and only if $L(R) = \varnothing$ or $L(S) = \varnothing$. This is evidently decidable. Finally, $L(R^*) = \varnothing$ if and only if $L(R) = \varnothing$, which is decidable. □

We conclude with the following theorem, which we have used already in Section 1.5.

Theorem 2.1.14 *Let L be a context free language and R be a regular language. Then $L \cap R$ is context free.*

Proof. Let be $G = \langle S, N, A, R\rangle$ be a context free grammar with $L(G) = L$ and $\mathfrak{A} = \langle n, 0, F, \delta\rangle$ a deterministic automaton consisting of n states such that $L(\mathfrak{A}) = R$. We may assume that rules of G are of the form $X \to a$ or $X \to \vec{Y}$. We define new nonterminals, which are all of the form ${}^iX^j$, where $i, j < n$ and $X \in N$. The interpretation is as follows. X stands for the set of all strings $\vec{\alpha} \in A^*$ such that $X \vdash_G \vec{\alpha}$. ${}^iX^j$ stands for the set of all $\vec{\alpha}$ such that $X \vdash_G \vec{\alpha}$ and $\delta(i, \vec{\alpha}) = j$. We have a set of start symbols, consisting of all ${}^0S^j$ with $j \in F$. As we already know, this does not increase the generative power. A rule of the form $X \to Y_0Y_1 \dots Y_{k-1}$ is now replaced by the set of all rules of the form

$$ {}^iX^j \to^i Y_0^{i_0} \, {}^{i_0}Y_1^{i_1} \, \dots \, {}^{i_{k-2}}Y_{k-1}^j . $$

Finally, we take all rules of the form ${}^iX^j \to a$, $\delta(i, a) = j$. This defines the grammar G_r. We shall show: $\vdash_{G^r} \vec{x}$ if and only $\vdash_G \vec{x}$ and $\vec{x} \in L(\mathfrak{A})$. ($\Rightarrow$) Let $\mathfrak{B}$ be a G^r–tree with associated string $\vec{x}$. The map ${}^iX^j \mapsto X$ turns $\mathfrak{B}$ into a G–tree. Hence $\vec{x} \in L(G)$. Further, it is easily shown that $\delta(0, x_0x_1 \dots x_j) = k_j$, where ${}^{k_{j-1}}X^{k_j}$ is the node dominating x_j. Also, if $|\vec{x}| = n$, then ${}^0S^{k_n}$ is the top node and by construction $k_n \in F$. Hence $\delta(\vec{x}, 0) \in F$ and so $\vec{x} \in L(\mathfrak{A})$. ($\Leftarrow$) Let $\vec{x} \in L(G)$ and $\vec{x} \in L(\mathfrak{A})$. We shall show that $\vec{x} \in L(G^r)$. We take a G–tree $\mathfrak{B}$ for $\vec{x}$. We shall now prove that one can replace the G–nonterminals in $\mathfrak{B}$ in such a way by G^r–nonterminals that we get a G_r–tree. The proof is by induction on the height of a node. We begin with nodes of height 1. Let $\vec{x} = \prod_{i<n} x_i$; and let X_i be the nonterminal above x_i. Further let $\delta(0, \prod_{i<j} x_i) = j_i$. Then $p_0 = 0$ and $p_n \in F$. We replace X_i by ${}^{p_i}X^{p_{i+1}}$. We say that

two nodes x and y **connect** if they are adjacent and for the labels ${}^iX^j$ of x and ${}^kY^\ell$ of y we have $j = k$. Let x be a node of height $n+1$ with label X and let x be mother of the nodes with labels $Y_0Y_1 \dots Y_{n-1}$ in G. We assume that below x all nodes carry labels from G^r in such a way that adjacent nodes connect. Then there exists a rule in G_r such that X can be labelled with superscripts, the left hand superscript of Y_0 to its left and the right hand superscript of Y_{n-1} to its right. All adjacent nodes of height $n+1$ connect, as is easily seen. Further, the leftmost node carries the left superscript 0, the rightmost node carries a right superscript p_n, which is an accepting state. Eventually, the root has superscripts as well. It carries the label ${}^0S^{p_n}$, and so we have a G_r–tree. $\Box$

Exercise 44. Prove Theorem 2.1.11.

Exercise 45. Show that a language is regular if and only if it can be generated by a grammar with rules of the form $X \to Y$, $X \to Ya$, $X \to a$ and $X \to \varepsilon$. Such a grammar is called **left regular**, in contrast to the grammars of Type 3, which we also call **right regular**. Show also that it is allowed to add rules of the form $X \to \vec{x}$ and $X \to Y\vec{x}$.

Exercise 46. Show that there is a grammar with rules of the form $X \to a$, $X \to aY$ and $X \to Ya$ which generates a nonregular language. This means that a Type 3 grammar may contain (in general) only left regular rules or only right regular rules, but not both.

Exercise 47. Show that if L and M are regular, then so are L/M und $M\backslash L$.

Exercise 48. Let L be a language over A. Define an equivalence relation $\sim_S$ over A^* as follows. $\vec{x} \sim_S \vec{y}$ if and only if for all $\vec{z} \in A^*$ we have $\vec{x} \cdot \vec{z} \in L \Leftrightarrow \vec{y} \cdot \vec{z} \in L$. L is said to have **finite index** if there are only finitely many equivalence classes with respect to $\sim_S$. Show that L is regular if and only if it has finite index.

Exercise 49. Show that the language $\{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$ does not have finite index. Hence it is not regular.

Exercise 50. Show that the intersection of a context sensitive language with a regular language is again context sensitive.

Exercise 51. Show that L is regular if and only if it is accepted by a read only 1–tape Turing machine.

2.2 Normal Forms for Context Free Grammars

In the remaining sections of this chapter we shall deal with context free grammars and languages. In view of the extensive literature about context free languages it is only possible to present an overview. In this section we shall deal in particular with normal forms. There are many normal forms for context free grammars, each having a different purpose. However, notice that the transformation of a grammar into a normal form necessarily destroys some of its properties. So, to say that a grammar can be transformed into another is meaningless unless we specify exactly what properties remain constant under this transformation. If, for example, we are only interested in the language generated then we can transform any context free grammar into Chomsky normal form. However, if we want to maintain the constituent structures, then only the so called standard form is possible. A good exposition of this problem area can be found in (Miller, 1999).

Before we deal with reductions of grammars we shall study the relationship between derivations, trees and sets of rules. To be on the safe side, we shall assume that every symbol occurs at least once in a tree, that is, that the grammar is slender in the sense of Definition 2.2.3. From the considerations of Section 1.6 we conclude that for any two context free grammars $G = \langle S, N, A, R\rangle$ and $G' = \langle S', N', A, R'\rangle$ $L_B(G) = L_B(G')$ if and only if $\mathbf{der}(G) = \mathbf{der}(G')$. Likewise we see that for all $X \in N \cup N'$ $\mathbf{der}(G, X) =$

$\mathbf{der}(H, X)$ if and only if $R = R'$. Now let $G = \langle S, N, A, R \rangle$ and a sequence $\Gamma = \langle \vec{\alpha}_i : i < n \rangle$ be given. In order to test whether Γ is a G–string sequence we have to check for each $i < n - 1$ whether $\vec{\alpha}_{i+1}$ can be derived from $\vec{\alpha}_i$ with a single application of a rule. To this end we have to choose an $\vec{\alpha}_i$ and apply a rule and check whether the string obtained equals $\vec{\alpha}_{i+1}$. Checking this needs $a_G \times |\vec{\alpha}_i|$ steps, where a_G is a constant which depends only on G. Hence for the whole derivation we need $\sum_{i<n} a_G |\vec{\alpha}_i|$ steps. This can be estimated from above by $a_G \times n \times |\vec{\alpha}_{n-1}|$ and if G is strictly expanding also by $a_G \times |\vec{\alpha}_{n-1}|^2$. It can be shown that there are grammars for which this is the best possible bound. In order to check for an ordered labelled tree whether it can be generated by γG we need less time. We only need to check for each node whether the local tree at x conforms to some rule of G. This can be done in constant time, depending only on G. The time therefore only linearly depends on the size of the tree.

There is a tight connection between derivations and trees. To begin, a derivation has a unique tree corresponding to it. Simply translate the derivation in G into a derivation in γG. Conversely, however, there may exist many derivations for the same tree. Their number can be very large. However, we can obtain them systematically in the following way. Let $\mathfrak{B}$ be an (exhaustively ordered, labelled) tree. Call $\lhd \subseteq B^2$ a **linearisation** if $\lhd$ is an irreflexive, linear ordering and from $x > y$ follows $x \lhd y$. Given a linearisation, a derivation is found as follows. We begin with the element which is smallest with respect to $\lhd$. This is, as is easy to see, the root. The root carries the label S. Inductively, we shall construct cuts $\vec{\alpha}_i$ through $\mathfrak{B}$ such that the sequence $\langle \vec{\alpha}_i : i < n \rangle$ is a derivation of the associated string. (Actually, the derivation is somewhat more complex than the string sequence, but we shall not complicate matters beyond need here.) The beginning is clear: we put $\vec{\alpha}_0 := S$. Now assume that $\vec{\alpha}_i$ has been established, and that it is not identical to the associated string of $\mathfrak{B}$. Then there exists a node y with nonterminal label in $\vec{\alpha}_i$. (There is a unique correspondence between nodes of the cut and segments of the strings

$\vec{\alpha}_i$.) We take the smallest such node with respect to $\lhd$. Let its label be Y. Since we have a G–tree, the local tree with root y corresponds to a rule of the form $Y \to \vec{\beta}$ for some $\vec{\beta}$. In $\vec{\alpha}_i$ y defines a unique instance of that rule. Then $\vec{\alpha}_{i+1}$ is the result of replacing that occurrence of Y by $\vec{\beta}$. The new string is then the result of applying a rule of G, as desired.

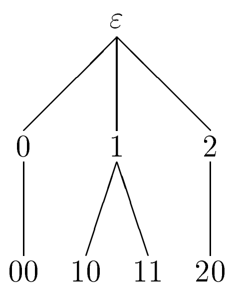
It is also possible to determine for each derivation a linearisation of the tree which yields that derivation in the described manner. However, there can be several linearisations that yield the same derivation.

Theorem 2.2.1 *Let G be context free and $\mathfrak{B} \in L_B(G)$. Further, let $\lhd$ be a linearisation of $\mathfrak{B}$. Then $\lhd$ determines a G–derivation $\boldsymbol{der}(\lhd)$ of the string which is associated to $\mathfrak{B}$. If $\blacktriangleleft$ is another linearisation of $\mathfrak{B}$ then $\boldsymbol{der}(\blacktriangleleft) = \boldsymbol{der}(\lhd)$ is the case if and only if $\blacktriangleleft$ and $\lhd$ coincide on the interior nodes of $\mathfrak{B}$.* $\Box$

Linearisations may also be considered as top down search strategies on a tree. We shall present examples. The first is a particular case of the so called **depth–first** search and the linearisation shall be called **leftmost linearisation**. It is as follows. $x \lhd y$ if and only if $x > y$ or $x \sqsubset y$. For every tree there is exactly one leftmost linearisation. We shall denote the fact that there is a leftmost derivation of $\vec{\alpha}$ from X by $X \vdash^{\ell}_{G} \vec{\alpha}$. We can generalize the situation as follows. Let $\blacktriangleleft$ be a linear ordering uniformly defined on the leaves of local subtrees. That is to say, if $\mathfrak{B}$ and $\mathfrak{C}$ are isomorphic local trees (that is, if they correspond to the same rule ρ) then $\blacktriangleleft$ orders the leaves $\mathfrak{B}$ linearly in the same way as $\lhd$ orders the leaves of $\mathfrak{C}$ (modulo the unique (!) isomorphism). In the case of the leftmost linearisation the ordering is the one given by $\sqsubset$. Now a minute's reflection reveals that every linearisation of the local subtrees of a tree induces a linearisation of the entire tree but not conversely (there are orderings which do not proceed in this way, as we shall see shortly). $X \vdash^{\blacktriangleleft}_{G} \vec{\alpha}$ denotes the fact that there is a derivation of $\vec{\alpha}$ from X determined by $\blacktriangleleft$. Now call π a **priorisation for** $G = \langle S, N, A, R\rangle$ if π defines a linearisation on the local tree $\mathfrak{H}_\rho$, for every $\rho \in R$. Since the root is always the first element in a linearisation, we only need to order the daughters of the root node, that is, the leaves. Let this ordering be $\blacktriangleleft$. We write $X \vdash^\pi_G \vec{\alpha}$ if $X \vdash^{\blacktriangleleft}_G \vec{\alpha}$ for the linearisation $\blacktriangleleft$ defined by π.

Proposition 2.2.2 *Let π be a priorisation. Then $X \vdash^\pi_G \vec{x}$ if and only if $X \vdash_G \vec{x}$.*

A different strategy is the *breadth–first search.* This search goes through the tree in increasing depth. Let S_n be the set of all nodes x with $d(x) = n$. For each n, S_n shall be ordered linearly by $\sqsubset$. The **breadth–first search** is a linearisation Δ, which is defined as follows. (a) If $d(x) = d(y)$ then $x \mathbin{\Delta} y$ if and only if $x \sqsubset y$, and (b) if $d(x) < d(y)$ then $x \mathbin{\Delta} y$. The difference between these search strategies, depth–first and breadth–first, can be made very clear with tree domains (see Section 1.4). The depth–first search traverses the tree domain in the lexicographical order, the breadth–first search in the numerical order. Let the following tree domain be given.



The depth–first linearisation is

$$\varepsilon, 0, 00, 1, 10, 11, 2, 20$$

The breadth–first linearisation, however, is

$$\varepsilon, 0, 1, 2, 00, 10, 11, 20$$

Notice that with these linearisations the tree domain ω^* cannot be enumerated. Namely, in the depth–first linearisation we get

$$\varepsilon, 0, 00, 000, 0000, \ldots$$

Also, in the breadth–first linearisation we get

$$\varepsilon, 0, 1, 2, 3, \ldots$$

On the other hand, ω^* is countable, so we do have a linearization, but it is more complicated than the given ones.

The first reduction of grammars we look at is the elimination of superfluous symbols and rules. Let $G = \langle S, A, N, R\rangle$ be a context free grammar. Call $X \in N$ **reachable** if $G \vdash \vec{\alpha} \cdot X \cdot \vec{\beta}$ for some $\vec{\alpha}$ and $\vec{\beta}$. X is called **completable** if there is a $\vec{x}$ such that $X \Rightarrow_R \vec{x}$.

$$\begin{array}{lllllll} \mathtt{S} & \to & \mathtt{AB} & \quad & \mathtt{A} & \to & \mathtt{CB} \\ \mathtt{B} & \to & \mathtt{AB} & & \mathtt{A} & \to & \mathtt{x} \\ \mathtt{D} & \to & \mathtt{Ay} & & \mathtt{C} & \to & \mathtt{y} \end{array}$$

In the given grammar $\mathtt{A}$, $\mathtt{C}$ und $\mathtt{D}$ are completable, and $\mathtt{S}$, $\mathtt{A}$, $\mathtt{B}$ and $\mathtt{C}$ are reachable. Since $\mathtt{S}$, the start symbol, is not completable, no symbol is both reachable and completable. The grammar generates no terminal strings.

Let N' be the set of symbols which are both reachable and completable. We assume that $\mathtt{S} \in N'$, so it is completable. If not, $L(G) = \varnothing$. In this case put $N' := \{\mathtt{S}\}$ and $R' := \varnothing$. Now let R' be the restriction of R to the symbols from $A \cup N'$. This defines $G' = \langle \mathtt{S}, N', A, R'\rangle$. It may be that throwing away rules may make some nonterminals unreachable or uncompletable. Therefore, this process must be repeated until $G' = G$, in which case every element is both reachable and completable. Call the resulting grammar G^s. Inductively, one can then show that $G \vdash \vec{\alpha}$ if and only if $G^s \vdash \vec{\alpha}$. Additionally, it can be shown that every derivation in G is a derivation in G^s and conversely.

Definition 2.2.3 *A context free grammar is called* ***slender*** *if either $L(G) = \varnothing$ and G has no nonterminals except for the start symbol and no rules; or $L(G) \neq \varnothing$ and every nonterminal is both reachable and completable.*

Two slender grammars have identical sets of derivations if and only if their rule sets are identical.

Proposition 2.2.4 *Let G and H be slender. Then $G = H$ if and only if $\boldsymbol{der}(G) = \boldsymbol{der}(H)$.*

Proposition 2.2.5 *For every context free grammar G there is an effectively constructable slender context free grammar $G^s = \langle \mathtt{S}, N^s, A, R^s \rangle$ such that $N^s \subseteq N$, which has the same set of derivations as G. In this case it also follows that $L_B(G^s) = L_B(G)$.* □

Next we shall discuss the role of the nonterminals. Since these symbols do not occur in $L(G)$, their name is irrelevant for the purposes of $L(G)$. To make this precise we shall introduce the notion of a rule simulation. Let G and G' be grammars with sets of nonterminals N and N'. Let $\sim \subseteq N \times N'$ be a relation. This relation can be extended to a relation $\approx \subseteq (N \cup A)^* \times (N' \cup A)^*$ by putting $\vec{\alpha} \approx \vec{\beta}$ if $\vec{\alpha}$ and $\vec{\beta}$ are of equal length and $\alpha_i \sim \beta_i$ for every i. A relation $\sim \subseteq N \times N'$ is called a **forward rule simulation** or **R–simulation** if (0) $\mathtt{S} \sim \mathtt{S}'$, (1) if $X \to \vec{\alpha} \in R$ and $X \sim Y$ then there exists a $\vec{\beta}$ such that $\vec{\alpha} \approx \vec{\beta}$ and $Y \to \vec{\beta} \in R'$, and (2) if $Y \to \vec{\beta} \in R'$ and $X \sim Y$ then there exists an $\vec{\alpha}$ such that $\vec{\alpha} \approx \vec{\beta}$ and $X \to \vec{\alpha} \in R$. A **backward simulation** is defined as follows. (0) From $\mathtt{S} \sim X$ follows $X = \mathtt{S}'$ and from $Y \sim \mathtt{S}'$ follows $Y = \mathtt{S}$, (1) if $X \to \vec{\alpha} \in R$ and $\vec{\alpha} \approx \vec{\beta}$ then $Y \to \vec{\beta} \in R'$ for some Y such that $X \sim Y$, (2) if $Y \to \vec{\beta} \in R'$ and $\vec{\beta} \approx \vec{\alpha}$ then $X \to \vec{\alpha} \in R$ for some X such that $X \sim Y$.

We give an example of a forward simulation. Let G and G' be the following grammars.

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{ASB} \mid \mathtt{AB} \\ \mathtt{A} & \to & \mathtt{b} \\ \mathtt{B} & \to & \mathtt{b} \end{array} \qquad \begin{array}{lcl} \mathtt{S} & \to & \mathtt{ATB} \mid \mathtt{ASC} \mid \mathtt{AC} \\ \mathtt{T} & \to & \mathtt{ATC} \mid \mathtt{AC} \\ \mathtt{A} & \to & \mathtt{a} \\ \mathtt{B} & \to & \mathtt{b} \\ \mathtt{C} & \to & \mathtt{b} \end{array}$$

The start symbol is $\mathtt{S}$ in both grammars. Then the following is an R–simulation.

$$\sim := \{\langle \mathtt{A}, \mathtt{A}\rangle, \langle \mathtt{B}, \mathtt{B}\rangle, \langle \mathtt{S}, \mathtt{S}\rangle, \langle \mathtt{B}, \mathtt{C}\rangle, \langle \mathtt{S}, \mathtt{T}\rangle\}$$

Together with $\sim$ also the converse relation $\sim^{\smile}$ is an R–simulation. If $\sim$ is an R–simulation and $\langle \vec{\alpha}_i : i < n+1\rangle$ is a G–derivation there exists a G'–derivation $\langle \vec{\beta}_i : i < n+1\rangle$ such that $\vec{\alpha}_i \approx \vec{\beta}_i$ for every $i < n+1$. We can say more exactly that if $\langle \vec{\alpha}_i, C, \vec{\alpha}_{i+1}\rangle$ is an instance of a rule from G where $C = \langle \kappa_1, \kappa_2\rangle$ then there is a context $D = \langle \lambda_1, \lambda_2\rangle$ such that $\langle \vec{\beta}_i, D, \vec{\beta}_{i+1}\rangle$ is an instance of a rule from G'. In this way we get that for every $\mathfrak{B} = \langle B, <, \sqsubset, \ell\rangle \in L_B(G)$ there is a $\mathfrak{C} = \langle B, <, \sqsubset, \mu\rangle \in L_B(G')$ such that $\ell(x) = \mu(x)$ for every leaf and $\ell(x) \sim \mu(x)$ for every nonleaf. Analogously to a rule simulation we may define a simulation of derivation by requiring that for every G–derivation Γ there is a G'–derivation Δ which is equivalent to it.

Proposition 2.2.6 *Let G_1 and G_2 be slender context free grammars and $\sim \subseteq N_1 \times N_2$ be an R–simulation. Then for every G_1–derivation $\langle \vec{\alpha}_i : i < n\rangle$ there exists a G_2–derivation $\langle \vec{\beta}_i : i < n\rangle$ such that $\vec{\alpha}_i \approx \vec{\beta}_i$, $i < n$.* □

We shall look at two special cases of simulations. Two grammars G and G' are called **equivalent** if there is a bijection $b : N \cup A \to N' \cup A$ such that $b(x) = x$ for every $x \in A$, $b(S) = S'$ and $\overline{b}$ induces a bijection between G–derivations and G'–derivations. This notion is more restrictive than the one which requires that $\overline{b}$ is a bijection between the set of rules. For it may happen that certain rules may never be used in a derivation. For given context free grammars we can easily decide whether they are equivalent. To begin, we bring them into a form in which all rules are used in a derivation, by removing all symbols that are not reachable and not completable. Such grammars are equivalent if there is a bijection b which puts the rules into correspondence. The existence of such a bijection is easy to check.

The notion of equivalence just proposed is too strict in one sense. There may be nonterminal symbols which cannot be distinguished. We say G is **reducible to** G' if there is a surjective function $b : N \cup A \twoheadrightarrow N' \cup A'$ such that $b(S) = S'$, $b(x) = x$ for every $x \in A$ and such that $\overline{b}$ maps every G–derivation onto a G'–derivation, while every preimage under $\overline{b}$ of a G'–derivation is a G–derivation. (We do not require however that the preimage of the start symbol from G' is unique; only that the start symbol from G has *one* preimage which is a start symbol of G'.)

Definition 2.2.7 *G is called **reduced** if every grammar G' such that G is reducible onto G' can itself be reduced onto G.*

Given G we can effectively construct a reduced grammar onto which it can be reduced. We remark that in our example above G' is not reducible onto G. For even though $\sim^{\smile}$ is a function (with $\mathtt{A} \mapsto \mathtt{A}, \mathtt{B} \mapsto \mathtt{B}, \mathtt{C} \mapsto \mathtt{B}, \mathtt{S} \mapsto \mathtt{S}, \mathtt{T} \mapsto \mathtt{S}$) and `ASB` can be derived from `S` in one step, `ATB` cannot be derived from `S` in one step. Given G and the function $\sim^{\smile}$ the following grammar is reduced onto G.

$$\begin{array}{lll}
\mathtt{S} & \to & \mathtt{ASB} \mid \mathtt{ATB} \mid \mathtt{ASC} \mid \mathtt{ATC} \mid \mathtt{AB} \mid \mathtt{AC} \\
\mathtt{T} & \to & \mathtt{ASB} \mid \mathtt{ATB} \mid \mathtt{ASC} \mid \mathtt{ATC} \mid \mathtt{AB} \mid \mathtt{AC} \\
\mathtt{A} & \to & \mathtt{a} \\
\mathtt{B} & \to & \mathtt{b} \\
\mathtt{C} & \to & \mathtt{b}
\end{array}$$

Now let G be a context free grammar. We add to A two more symbols, namely `(` and `)`, not already contained in A. Subsequently, we replace every rule $X \to \vec{\alpha}$ by the rule $X \to \mathtt{(} \cdot \vec{\alpha} \cdot \mathtt{)}$. The so constructed grammar is denoted by G^b.

G			G^b		
S	$\to$	AS | SB | AB	S	$\to$	(AS) | (SB) | (AB)
A	$\to$	a	A	$\to$	(a)
B	$\to$	b	B	$\to$	(b)

The grammar G generates the language $\mathtt{a}^+\mathtt{b}^+$. The string aabb has several derivations, which correspond to different trees.

$$\langle \mathtt{S}, \mathtt{AS}, \mathtt{ASB}, \mathtt{AABB}, \dots, \mathtt{aabb} \rangle$$
$$\langle \mathtt{S}, \mathtt{SB}, \mathtt{ASB}, \mathtt{AABB}, \dots, \mathtt{aabb} \rangle$$

If we look at the analogous derivations in G^b we get the strings

$$\mathtt{((a)(((a)(b))(b)))}, \qquad \mathtt{(((a)((a)(b)))(b))}$$

These are obviously distinct. Define a homomorphism $\overline{e}$ by $\overline{e}(a) := a$, if $a \in A$, $\overline{e} : \mathtt{)} \mapsto \varepsilon$ and $\overline{e} : \mathtt{)} \mapsto \varepsilon$. Then it is not hard to see that

$$L(G) = \overline{e}[L(G^b)] \ .$$

Now look at the class of trees $L(G)$ and forget the labels of all nodes which are not leaves. Then the structure obtained shall be called a **bracketing analysis** of the associated strings. The reason is that the bracketing analyses are in one–to–one correspondence with the strings which $L(G^b)$ generates. Now we will ask ourselves whether for two given grammars G and H it is decidable whether they generate the same bracketing analyses. We ask ourselves first what the analogon of a derivation of G is in G^b. Let $\vec{\gamma}X\vec{\eta}$ be derivable in G, and let the corresponding G^b–string in this derivation be $\vec{\gamma}^b X \vec{\eta}^b$. In the next step X is replaced by α. Then we get $\vec{\gamma}\vec{\alpha}\vec{\eta}$, and in G^b the string $\vec{\gamma}^b \mathtt{(}\vec{\alpha}\mathtt{)} \vec{\delta}^b$. If we have an R–simulation to H then it is also an R–simulation from G^b to H^b provided that it sends the opening bracket of G^b to the opening bracket of H^b and the closing bracket of G^b to the closing bracket of H^b. It follows that if there is an R–simulation from G to H then not only we have $L(G) = L(H)$ but also $L(G^b) = L(H^b)$.

Theorem 2.2.8 *We have $L(G^b) = L(H^b)$ if there is an R–simulation from G to H.*

The bracketing analysis is too strict for most purposes. First of all it is not customary to put a single symbol into brackets. Further, it makes no sense to distinguish between $\texttt{((}\vec{x}\texttt{))}$ and $\texttt{(}\vec{x}\texttt{)}$, since both strings assert that $\vec{x}$ is a constituent. We shall instead use what we call **constituent analyses**. These are pairs $\langle \vec{x}, \mathfrak{C}\rangle$ in which $\vec{x}$ is a string and $\mathfrak{C}$ an exhaustively ordered constituent structure defined over $\vec{x}$. We shall denote by $L_c(G)$ the class of all constituent analyses generated by G. In order to switch from bracketing analyses to constituent analyses we only have to eliminate the unary rules. This can be done as follows. First of all every rule $\rho = Y \to \vec{\alpha}$ is replaced by the set ρ^2 of all rules in which Y is replaced in $\vec{\alpha}$ by a Z such that $Z \Rightarrow Y^+$. $R^2 := \bigcup\langle \rho^2 : \rho \in R\rangle$. $R^>$ is the result of eliminating all rules of the form $X \to Y$ from R^2. Finlly let $G^> := \langle \mathtt{S}, N, A, R^>\rangle$. Every rule is strictly productive and we have $L_c(G) = L_c(G^>)$. (If necessary, we shall assume that $G^>$ is slender.)

Definition 2.2.9 *A context free grammar is in* ***standard form*** *if every rule different from $\mathtt{S} \to \varepsilon$ has the form $X \to \vec{Y}$ with $|\vec{Y}| > 1$ or the form $X \to a$. A grammar is in* ***2–standard form*** *or* ***Chomsky normal form*** *if every rule is of the form $\mathtt{S} \to \varepsilon$, $X \to Y_0Y_1$ or $X \to a$.*

(Notice that by our conventions a context free grammar in standard form contains the rule $X \to \varepsilon$ for $X = \mathtt{S}$, but this only if $\mathtt{S}$ is not on the right hand side of a rule.) We already have proved that the following holds.

Theorem 2.2.10 *For every context free grammar G one can construct a slender context free grammar G^n in standard form which generates the same constituent structures as G.*

Theorem 2.2.11 *For every context free grammar G we can construct a slender context free grammar G^c in Chomsky normal form such that $L(G^c) = L(G)$.*

Proof. WE may assume that G is in standard form. Let $\rho = X \to Y_0 Y_1 \ldots Y_{n-1}$ be a rule with $n > 2$. Let $Z_0^\rho, Z_1^\rho, \ldots, Z_{n-2}^\rho$ be new nonterminals. Replace ρ by the rules

$$\begin{aligned}\rho_0^c &:= X \to Y_0 Z_0^\rho,\ \rho_1^c := Z_0^\rho \to Y_1 Z_1^\rho, \ldots, \\ \rho_{n-2}^c &:= Z_{n-3}^\rho \to Y_{n-2} Y_{n-1}\end{aligned}$$

Every derivation in G of a string $\vec{\alpha}$ can be translated into a derivation in G^c by replacing every instance of ρ by a sequence $\rho_0^c, \rho_1^c, \ldots, \rho_{n-1}^c$. For the converse we introduce the following priorisation π on the rules. Let Z_i^ρ be always before Y_i. However, in $Z_{n-3}^\rho \to Y_{n-2} Y_{n-1}$ we choose the leftmost priorisation. We show $G \vdash^\ell \vec{x}$ if and only if $G^c \vdash^\pi \vec{x}$. For if $\langle \alpha_i : i < p+1 \rangle$ is a leftmost derivation of $\vec{x}$ in G, then replace every instance of a rule ρ by the sequence ρ_0^c, ρ_1^c, and so on until ρ_{n-2}^c. This is a G^c–derivation, as is easily checked. It is also a π–derivation. Conversely, let $\langle \beta_j : j < q+1 \rangle$ be a G^c–derivation which is priorised with π. If β_{i+1} is the result of an application of the rule ρ_k^c, $k < n-2$, then $i+2 < q+1$ and β_{i+2} is the result of an application of ρ_{k+1}^c on β_{i+1}, which replaced exactly the occurrence Z_k of the previous instance. This means that every ρ_k^c in a block of instances of ρ_0^c, ρ_1^c, …, ρ_{n-2}^c corresponds to a single instance of ρ. There exists a G–derivation of $\vec{x}$, which can be obtained by backward replacement of the blocks. It is a leftmost derivation. □

For example, the right hand side grammar is the result of the conversion of the left hand grammar into Chomsky normal form.

`S`	$\to$	`ASBBT` | `ABB`	`S`	$\to$	`AX` | `AV`
			`V`	$\to$	`BB`
			`X`	$\to$	`SY`
			`Y`	$\to$	`BZ`
			`Z`	$\to$	`BT`
`T`	$\to$	`CTD` | `CD`	`T`	$\to$	`CW` | `CD`
			`W`	$\to$	`TD`
`A`	$\to$	`a`	`A`	$\to$	`a`
`B`	$\to$	`b`	`B`	$\to$	`b`
`C`	$\to$	`c`	`C`	$\to$	`c`
`D`	$\to$	`d`	`D`	$\to$	`d`

Definition 2.2.12 *A context free grammar is called **invertible** if from $X \to \vec{\alpha} \in R$ and $Y \to \vec{\alpha} \in R$ it follows that $X = Y$.*

For an invertible grammar the labelling on the leaves uniquely determines the labelling on the entire tree. We propose an algorithm which creates an invertible grammar from a context free grammar. For simplicity a rule is of the form $X \to \vec{Y}$ or $X \to \vec{x}$. Now we choose our nonterminals from the set $\wp(N) - \{\varnothing\}$. The terminal rules are now of the form $\mathsf{X} \to \vec{x}$, where $\mathsf{X} = \{X : X \to \vec{x} \in R\}$. The nonterminal rules are of the form $\mathsf{X} \to \mathsf{Y}_0\mathsf{Y}_1 \ldots \mathsf{Y}_{n-1}$ with

$$\mathsf{X} = \{X : X \to Y_0 Y_1 \ldots Y_{n-1} \in R \text{ for some } Y_i \in \mathsf{Y}_i\} .$$

Further, we choose a start symbol, Σ, and we take the rules $\Sigma \to \vec{\mathsf{X}}$ for every $\vec{X}$, for which there are $X_i \in \mathsf{X}_i$ with $S \to \vec{X} \in R$. This grammar we call G^i. It is not difficult to show that G^i is invertible. For let $\mathsf{Y}_0\mathsf{Y}_1 \ldots \mathsf{Y}_{n-1}$ be the right hand side of a production. Then there exist $Y_i \in \mathsf{Y}_i$, $i < n$, and an X such that $X \to \vec{Y}$ is a rule in G. Hence there is an X such that $\mathsf{X} \to \vec{\mathsf{Y}}$ is in G^i. X is uniquely determined. Further, G^i is in standard form (Chomsky normal form), if this is the case with G.

Theorem 2.2.13 *Let G be a context free grammar. Then we can construct an invertible context free grammar G^i which generates the same bracketing analyses as G.* $\square$

The advantage offered by invertible grammars is that the labelling can be reconstructed from the labellings on the leaves. The reader may reflect on the fact that G is invertible exactly if G^b is.

Definition 2.2.14 *A context free grammar is called **perfect** if it is in standard form, slender, reduced and invertible.*

It is instructive to see an example of a grammar which is invertible but not reduced.

$$\begin{array}{lcl@{\qquad}lcl}
 & G & & & H & \\
\mathtt{S} & \to & \mathtt{AS} \mid \mathtt{BS} \mid \mathtt{A} \mid \mathtt{B} & \mathtt{S} & \to & \mathtt{CS} \mid \mathtt{C} \\
\mathtt{A} & \to & \mathtt{a} & \mathtt{C} & \to & \mathtt{a} \mid \mathtt{b} \\
\mathtt{B} & \to & \mathtt{b} & & &
\end{array}$$

G is invertible but not reduced. To this end look at H and the map $\mathtt{A} \mapsto \mathtt{C}$, $\mathtt{B} \mapsto \mathtt{C}$, $\mathtt{S} \mapsto \mathtt{S}$. This is an R–simulation. H is reduced and invertible.

Theorem 2.2.15 *For every context free grammar we can construct a perfect grammar which generates the same constituent structures.*

Finally we shall turn to the so called *Greibach normal form*. This form most important for algorithms recognizing languages by reading the input from left to right. Such algorithms have problems with rules of the form $X \to Y \cdot \vec{\alpha}$, in particular if $Y = X$.

Definition 2.2.16 *Let $G = \langle \mathtt{S}, N, A, R\rangle$ be a context free grammar. G is in **Greibach (normal) form** if every rule is of the form $\mathtt{S} \to \varepsilon$ or of the form $X \to x \cdot \vec{Y}$.*

Proposition 2.2.17 *Let G be in Greibach normal form and $X \vdash_G \vec{\alpha}$. $\vec{\alpha}$ has a leftmost derivation from X in G if and only if $\vec{\alpha} = \vec{y} \cdot \vec{Y}$ for some $\vec{y} \in A^*$ and $\vec{Y} \in N^*$ and $\vec{y} = \varepsilon$ only if $\vec{Y} = X$.*

The proof is not hard. It is also not hard to see that this property characterizes the Greibach form uniquely. For if there is a rule of the form $X \to Y \cdot \vec{\gamma}$ then there is a leftmost derivation of $Y \cdot \vec{\gamma}$ from X, but not in the desired form. Here we assume that there are no rules of the form $X \to X$.

Theorem 2.2.18 (Greibach) *For every context free grammar one can effectively construct a grammar G^g in Greibach normal form with $L(G^g) = L(G)$.*

Before we start with the actual proof, we shall prove some auxiliary statements. We call ρ an X–**production** if $\rho = X \to \vec{\alpha}$ for some $\vec{\alpha}$. Such a production is called **left recursive** if it has the form $X \to X \cdot \vec{\beta}$. Let $\rho = X \to \vec{\alpha}$ be a rule; define $R^{-\rho}$ as follows. For every factorisation $\vec{\alpha} = \vec{\alpha}_1 \cdot Y \cdot \vec{\alpha}_2$ of $\vec{\alpha}$ and every rule $Y \to \vec{\beta}$ add the rule $X \to \vec{\alpha}_1 \cdot \vec{\beta} \cdot \vec{\alpha}_2$ to R and finally remove the rule ρ. Now let $G^{-\rho} := \langle \mathtt{S}, N, A, R^{-\rho} \rangle$. Then $L(G^{-\rho}) = L(G)$. We call of this construction as **skipping** the rule ρ. The reader may convince himself that the tree for $G^{-\rho}$ can be obtained in a very simple way from trees for G simply by removing all nodes x which dominate a local tree corresponding to the rule ρ, that is to say, which are isomorphic to $\mathfrak{H}_\rho$. (This has been defined in Section 1.6.) This technique works only if ρ is not an $\mathtt{S}$–production. In this case we proceed as follows. Replace ρ by all rules of the form $\mathtt{S} \to \vec{\beta}$ where $\vec{\beta}$ derives from $\vec{\alpha}$ by applying a rule. Skipping a rule does not necessarily yield a new grammar. This is so if there are rules of the form $X \to Y$ (in particular rules like $X \to X$).

Lemma 2.2.19 *Let $G = \langle \mathtt{S}, N, A, R \rangle$ be a context free grammar and let $X \to X \cdot \vec{\alpha}_i$, $i < m$, be all left recursive X–productions as well as $X \to \vec{\beta}_j$, $j < n$, all non left recursive X–productions. Now let $G^1 := \langle \mathtt{S}, N \cup \{Z\}, A, R^1 \rangle$, where $Z \notin N \cup A$ and R^1 consists of all Y–productions from R with $Y \neq X$ as well as the productions*

$$\begin{array}{llll} X \to \vec{\beta}_j, & j < n, & Z \to \vec{\alpha}_i, & i < m, \\ X \to \vec{\beta}_j \cdot Z, & j < n, & Z \to \vec{\alpha}_i \cdot Z, & i < m. \end{array}$$

Then $L(G^1) = L(G)$.

Proof. We shall prove this lemma rather extensively since the method is relatively tricky. We consider the following priorisation on G^1. In all rules of the form $X \to \vec{\beta}_j$ and $Z \to \vec{\alpha}_i$ we take the natural ordering (that is, the leftmost ordering) and in all rules $X \to \vec{\beta}_j Z$ as well as $Z \to \vec{\alpha}_i Z$ we also put the left to right ordering except that Z precedes all elements from $\vec{\alpha}_j$ and $\vec{\beta}_i$, respectively. This defines the linearisation $\blacktriangleleft$. Now, let $M(X)$ be the set of all $\vec{\gamma}$ such that there is a leftmost derivation from X in G in such a way that $\vec{\gamma}$ is the first element not of the form $X \cdot \vec{\delta}$. Likewise, we define $P(X)$ to be the set of all $\vec{\gamma}$ which can be derived from X priorised by $\blacktriangleleft$ in G^1 such that $\vec{\gamma}$ is the first element which does not contain Z. We claim that $P(X) = M(X)$. It can be seen that

$$M(X) = \bigcup_{j<n} \vec{\beta} j \cdot (\bigcup_{i<m} \vec{\alpha}_i)^* = P(X)$$

From this the desired conclusion follows thus. Let $\vec{x} \in L(G)$. Then there exists a leftmost derivation $\Gamma = \langle A_i : i < n+1 \rangle$ of $\vec{x}$. (Recall that the A_i are instances of rules.) This derivation is cut into segments Σ_i, $i < \sigma$, of length k_i, such that

$$\Sigma_i = \langle A_j : \sum_{p<i} k_p \leq j < 1 + \sum_{p<i+1} k_i \rangle \ .$$

This partitioning is done in such a way that each Σ_i is a maximal portion of Γ of X–productions or a maximal portion of Y–productions with $Y \neq X$. The X–segments can be replaced by a $\blacktriangleleft$–derivation $\widehat{\Sigma}_i$ in G^1, by the previous considerations. The segments which do not contain X–productions are already G^1–derivations. For them we put $\widehat{\Sigma}_i := \Sigma_i$. Now let $\widehat{\Gamma}$ be result of stringing together the $\widehat{\Sigma}_i$. This is well defined, since the first string of $\widehat{\Sigma}_i$ equals the first string of Σ_i, as the last string of $\widehat{\Sigma}_i$ equals the last string of Σ_i. $\widehat{\Gamma}$ is a G^1–derivation, priorised by $\blacktriangleleft$. Hence $\vec{x} \in L(G^1)$. The converse is analogously proved, by beginning with a derivation priorised by $\blacktriangleleft$. $\Box$

Now to the proof of Theorem 2.2.18. We may assume at the outset that G is in Chomsky normal form. We choose an enumeration of N as $N = \{X_i : i < p\}$. We claim first that by taking in new nonterminals we can see to it that we get a grammar G^1 such that $L(G^1) = L(G)$ in which the X_i–productions have the form $X_i \to x \cdot \vec{Y}$ or $X_i \to X_j \cdot \vec{Y}$ with $j > i$. This we prove by induction on i. Let i_0 be the smallest i such that there is a rule $X_i \to X_j \cdot \vec{Y}$ with $j \leq i$. Let j_0 be the largest j such that $X_{i_0} \to X_j \cdot \vec{Y}$ is a rule. We distinguish two cases. The first is $j_0 = i_0$. By the previous lemma we can eliminate the production by introducing some new nonterminal symbol Z_{i_0}. The second case is $j_0 < i_0$. Here we apply the induction hypothesis on j_0. We can skip the rule $X_{i_0} \to X_{j_0} \cdot \vec{Y}$ and introduce rules of the form (a) $X_{i_0} \to X_k \cdot \vec{Y}'$ with $k > j_0$. In this way the second case is either eliminated or reduced to the first.

Now let $P := \{Z_i : i < p\}$ be the set of newly introduced nonterminals. It may happen that for some j Z_j does not occur in the grammar, but this does not disturb the proof. Let finally $P_i := \{Z_j : j < i\}$. At the end of this reduction we have rules of the form

$$\begin{array}{lll}
\text{(a)} & X_i \to X_j \cdot \vec{Y} & (j > i) \\
\text{(b)} & X_i \to x \cdot \vec{Y} & (x \in A) \\
\text{(c)} & Z_i \to \vec{W} & (\vec{W} \in (N \cup P_i)^+ \cdot (\varepsilon \cup Z_i))
\end{array}$$

It is clear that every X_{p-1}–production already has the form $X_{p-1} \to x \cdot \vec{Y}$. If some X_{p-2}–production has the form $X_{p-2} \to X_{p-1} \cdot \vec{Y}$ then we can skip this rule and get rules of the form $X_{p-2} \to \vec{x}\vec{Y}'$. Inductively we see that all rules of the form can be eliminated in favour of rules of the form (b). Now finally the rules of the last line. Also these rules can be skipped, and then we get rules of the form $Z \to x \cdot \vec{Y}$ for some $x \in A$, as desired.

For example, let the following grammar be given.

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{SDA} \mid \mathtt{CC} \\ \mathtt{D} & \to & \mathtt{DC} \mid \mathtt{AB} \\ \mathtt{A} & \to & \mathtt{a} \\ \mathtt{B} & \to & \mathtt{b} \\ \mathtt{C} & \to & \mathtt{c} \end{array}$$

The production $\mathtt{S} \to \mathtt{SDA}$ is left recursive. We replace it according to the above lemma by

$$\mathtt{S} \to \mathtt{CCZ}, \quad \mathtt{Z} \to \mathtt{DA}, \quad \mathtt{Z} \to \mathtt{DAZ}\ .$$

Likewise we replace the production $\mathtt{D} \to \mathtt{DC}$ by

$$\mathtt{D} \to \mathtt{ABY}, \quad \mathtt{Y} \to \mathtt{C}, \quad \mathtt{Y} \to \mathtt{CY}$$

With this we get the grammar

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{CC} \mid \mathtt{CCZ} \\ \mathtt{Z} & \to & \mathtt{DA} \mid \mathtt{DAZ} \\ \mathtt{D} & \to & \mathtt{AB} \mid \mathtt{ABY} \\ \mathtt{Y} & \to & \mathtt{C} \mid \mathtt{CY} \\ \mathtt{A} & \to & \mathtt{a} \\ \mathtt{B} & \to & \mathtt{b} \\ \mathtt{C} & \to & \mathtt{c} \end{array}$$

Next we skip the $\mathtt{D}$–productions.

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{CC} \mid \mathtt{CCZ} \\ \mathtt{Z} & \to & \mathtt{ABA} \mid \mathtt{ABYA} \mid \mathtt{ABAZ} \mid \mathtt{ABYAZ} \\ \mathtt{D} & \to & \mathtt{AB} \mid \mathtt{ABY} \\ \mathtt{Y} & \to & \mathtt{C} \mid \mathtt{CY} \\ \mathtt{A} & \to & \mathtt{a} \\ \mathtt{B} & \to & \mathtt{b} \\ \mathtt{C} & \to & \mathtt{c} \end{array}$$

Next D can be eliminated (since it is not reachable) and we can replace on the right hand side of the productions the first nonterminals by terminals.

$$\begin{array}{lcl} \mathtt{S} & \rightarrow & \mathtt{cC} \mid \mathtt{cCZ} \\ \mathtt{Z} & \rightarrow & \mathtt{aBA} \mid \mathtt{aBYA} \mid \mathtt{aBAZ} \mid \mathtt{aBYZ} \\ \mathtt{Y} & \rightarrow & \mathtt{c} \mid \mathtt{cY} \end{array}$$

Now the grammar is in Greibach normal form.

Exercise 52. Show that for a context free grammar G it is decidable (a) whether $L(G) = \varnothing$, (b) whether $L(G)$ is finite, (c) whether $L(G)$ is infinite.

Exercise 53. Let G^i be the invertible grammar constructed from G as defined above. Show that the relation $\sim$ defined by

$$\mathsf{X} \sim Y \quad \Leftrightarrow \quad Y \in \mathsf{X}$$

is a backward simulation from G^i to G.

Exercise 54. Let $\langle B, <, \sqsubset, \ell\rangle$ be an ordered labelled tree. If x is a leaf then $\uparrow x$ is a branch and can be thought of in a natural way as a string $\langle\uparrow x, >, \ell\rangle$. Since the leaf x plays a special role, we shall omit it. We say, a **branch expression of $\mathfrak{B}$** is a string of the form $\langle\uparrow x - \{x\}, >, \ell\rangle$, x a leaf of $\mathfrak{B}$. We call it $\zeta(x)$. Show that

$$Z(G) = \{\zeta(x) : \zeta(x) \text{ is a branch expression of } \mathfrak{B}, \mathfrak{B} \in L_B(G)\}$$

is regular.

Exercise 55. Let G be in Greibach normal form and $\vec{x}$ a terminal string of length $n > 0$. Show that every derivation of $\vec{x}$ has exactly the length n. Can you name a number for arbitrary strings $\vec{\alpha}$?

2.3 Recognition and Analysis

Context free languages can be characterized using a special kind of automata, just like regular languages. A finite automaton is a machine which steers its action using a finite memory. Since there are context free languages that are not regular, automata that recognize them cannot all be finite state automata. They must have an infinite memory. The special way such a memory is organized and manipulated differentiates the various kinds of non-regular languages. Context free languages can be recognized by so called *pushdown automata*. These automata have a memory in the form of a stack onto which they can put symbols and remove (and read them) one by one. However, the automaton only has access to the symbol added most recently. A *stack* over the alphabet D is a string over D. We shall agree that the first letter of the string is the highest entry in the stack and the last letter corresponds to the lowest entry. To denote the end of the stack, we need a special symbol, which we denote by $\#$. (See Exercise 1.7.)

A *pushdown automaton* steers its actions by means of the highest entry of the stack and the momentary memory state. Its actions consist of three successive steps. (1) The disposal or removal of a symbol on the stack. (2) The moving or not moving of the read head to the right. (3) The change into a memory state (possibly the same one). If the automaton does not move the head in (2) we call the action an ε–**move**. We write A_ε in place of $A \cup \{\varepsilon\}$.

Definition 2.3.1 *A* ***pushdown automaton over*** *A is a septuple*

$$\mathfrak{K} = \langle Q, i_0, A, F, D, \#, \delta\rangle,$$

where Q and D are finite sets, $i_0 \in Q$, $\# \in D$ and $F \subseteq Q$, as well as δ a function $\delta : Q \times D \times A_\varepsilon \to \wp(Q \times D^*)$ such that $\delta(q, a, d)$ is always finite. We call Q the set of ***states****, i_0 the* ***initial state****, F the set of* ***accepting states****, D the* ***stack alphabet****, $\#$ the* ***beginning of the stack*** *and δ the* ***transition function****.*

We call $\mathfrak{z} := \langle q, \vec{d}\rangle$, where $q \in Q$ and $\vec{d} \in D^*$, a **configuration**. We now write

$$\langle p, \vec{d}\rangle \xrightarrow{x} \langle p', \vec{d}'\rangle$$

if for some $\vec{d}_1$ $\vec{d} = Z \cdot \vec{d}_1$, $\vec{d}' = \vec{e} \cdot \vec{d}_1$ and $\langle p', \vec{e}\rangle \in \delta(p, Z, x)$. We call this a **transition**. We extend the function δ to configurations and also write $\langle p', \vec{d'}\rangle \in \delta(p, \vec{d}, x)$. Notice that in contrast to a pushdown automaton a finite state automaton may not change into a new state without reading a new symbol. For a pushdown automaton this is necessary in particular if the automaton wants to clear the stack. If the stack is empty then the automaton cannot work further. This means, however, that the pushdown automaton is necessarily partial. The transition function can now analogously be extended to strings. Likewise, we can define it for sets of states. Put

$$\mathfrak{z} \xrightarrow{\vec{x}\cdot\vec{y}} \mathfrak{z}' \quad \Leftrightarrow \quad \text{there exists } \mathfrak{z}'' \text{ with } \mathfrak{z} \xrightarrow{\vec{x}} \mathfrak{z}'' \xrightarrow{\vec{y}} \mathfrak{z}' \ .$$

If $\mathfrak{z} \xrightarrow{\vec{x}} \mathfrak{z}'$ we say that there is a $\mathfrak{K}$–**computation for $\vec{x}$ from $\mathfrak{z}$ to $\mathfrak{z}'$**. Now

$$L(\mathfrak{K}) := \{\vec{x} : \textit{for some } q \in F \textit{ and some } \vec{z} \in D^* : \langle i_0, \#\rangle \xrightarrow{\vec{x}} \langle q, \vec{z}\rangle\} \ .$$

We call this the language which is **accepted by $\mathfrak{K}$ by state**. We call a pushdown automaton **simple** if from $\langle q, \vec{z}\rangle \in \delta(p, Z, a)$ follows $|\vec{z}| \leq 1$. It is an exercise to prove the next theorem.

Proposition 2.3.2 *For every pushdown automaton $\mathfrak{K}$ there is a simple pushdown automaton $\mathfrak{L}$ such that $L(\mathfrak{L}) = L(\mathfrak{K})$.*

For this reason we shall tacitly assume that the automaton does not write arbitrary strings but a single symbol. In addition to $L(\mathfrak{K})$ there also is a language which is **accepted by $\mathfrak{K}$ by stack**.

$$L^s(\mathfrak{K}) := \{\vec{x} : \textit{for some } q \in Q : \langle i_0, \#\rangle \xrightarrow{\vec{x}} \langle q, \varepsilon\rangle\}$$

Now the languages $L(\mathfrak{K})$ and $L^s(\mathfrak{K})$ are not necessarily identical for given $\mathfrak{K}$. However, the set of all languages of the form $L(\mathfrak{K})$ for some pushdown automaton equals the set of all languages of the form $L^s(\mathfrak{K})$ for some pushdown automaton. This is the content of the following theorem.

Proposition 2.3.3 *For every pushdown automaton $\mathfrak{K}$ there is an $\mathfrak{L}$ with $L(\mathfrak{K}) = L^s(\mathfrak{L})$ as well as a pushdown automaton $\mathfrak{M}$ with $L^s(\mathfrak{K}) = L(\mathfrak{M})$.*

Proof. Let $\mathfrak{K} = \langle Q, i_0, F, D, \#, \delta\rangle$ be given. We add to Q two states, q_i and q_f. q_i shall be the new initial state and $F^{\mathfrak{L}} := \{q_f\}$. Further, we add a new symbol $\flat$ which is the beginning of the stack of $\mathfrak{L}$. We define $\delta^{\mathfrak{L}}(q_i, \flat, \varepsilon) := \{\langle i_0, \# \cdot \flat\rangle\}$. There are no more $\delta^{\mathfrak{L}}$–transitions exiting q_i. For $q \neq q_i, q_f$ and $Z \neq \flat$ $\delta^{\mathfrak{L}}(q, Z, \vec{x}) := \delta^{\mathfrak{K}}(q, Z, x)$, $x \in A$. Further, if $q \in F$ and $Z \neq \flat$, $\delta^{\mathfrak{K}}(q, Z, \varepsilon) := \delta^{\mathfrak{K}}(q, Z, \varepsilon) \cup \{\langle q_f, \varepsilon\rangle\}$ and otherwise $\delta^{\mathfrak{L}}(q, Z, \varepsilon) := \delta^{\mathfrak{K}}(q, Z, \varepsilon)$. Finally, let $\delta^{\mathfrak{L}}(q_f, Z, x) := \varnothing$ for $x \in A$ and $\delta^{\mathfrak{L}}(q_f, Z, \varepsilon) := \{\langle q_f, \varepsilon\rangle\}$ for $Z \in D \cup \{\flat\}$. Assume now $\vec{x} \in L(\mathfrak{K})$. Then there exists a $\mathfrak{K}$–computation $\langle i_0, \#\rangle \xrightarrow{\vec{x}} \langle q, \vec{d}\rangle$ with $q \in F$ and so there is an $\mathfrak{L}$–computation $\langle q_i, \flat\rangle \xrightarrow{\vec{x}} \langle q_f, \vec{d}\rangle$. Since $\langle q_f, \vec{d}\rangle \xrightarrow{\varepsilon} \langle q_f, \varepsilon\rangle$ we have $\vec{x} \in L^s(\mathfrak{L})$. Hence $L(\mathfrak{K}) \subseteq L^s(\mathfrak{L})$. Now, conversely, let $\vec{x} \in L^s(\mathfrak{L})$. Then $\langle q_i, \flat\rangle \xrightarrow{\vec{x}} \langle p, \varepsilon\rangle$ for a certain p. Then $\flat$ is deleted only at last since it happens only in q_f and so $p = q_f$. Further, we have $\langle q_i, \flat\rangle \xrightarrow{\vec{x}} \langle q, \vec{d}\cdot\flat\rangle$ for some state $q \in F$. This means that there is an $\mathfrak{L}$–computation $\langle i_0, \# \cdot \flat\rangle \xrightarrow{\vec{x}} \langle q, \vec{d} \cdot \flat\rangle$. This, however, is also a $\mathfrak{K}$–computation. This shows that $L^s(\mathfrak{L}) \subseteq L(\mathfrak{K})$ and so also the first claim. Now for the construction of $\mathfrak{M}$. We add two new states, q_f und q_i, and a new symbol, $\flat$, which shall be the begin of stack of $\mathfrak{M}$, and we put $F^{\mathfrak{M}} := \{q_f\}$. Again we put $\delta^{\mathfrak{M}}(q_i, \flat, x) := \varnothing$ for $x \in A$ and $\delta^{\mathfrak{M}}(q_i, \flat, x) := \{\langle i_0, \# \cdot \flat\rangle\}$ for $x = \varepsilon$. Also, we let $\delta^{\mathfrak{M}}(q, Z, x) := \delta^{\mathfrak{K}}(q, Z, x)$ for $Z \neq \flat$ and $\delta^{\mathfrak{M}}(q, \flat, \varepsilon) := \{\langle q_f, \varepsilon\rangle\}$, as well as $\delta^{\mathfrak{M}}(q, \flat, x) := \varnothing$ for $x \in A$. Further, let $\delta^{\mathfrak{M}}(q_f, Z, x) := \varnothing$. This defines $\delta^{\mathfrak{M}}$. Now consider an $\vec{x} \in L^s(\mathfrak{K})$. There is a $\mathfrak{K}$–computation $\langle i_0, \#\rangle \xrightarrow{\vec{x}} \langle p, \varepsilon\rangle$ for some p. Then there exists an $\mathfrak{L}$–computation

$$\langle q_i, \flat\rangle \xrightarrow{\vec{x}} \langle p, \flat\rangle \xrightarrow{\varepsilon} \langle q_f, \varepsilon\rangle\ .$$

Hence $\vec{x} \in L(\mathfrak{M})$. Conversely, let $\vec{x} \in L(\mathfrak{M})$. Then there exists an $\mathfrak{L}$–computation $\langle q_i, \flat\rangle \xrightarrow{\vec{x}} \langle q_f, \vec{d}\rangle$ for some $\vec{d}$. One can see quite easily that $\vec{d} = \varepsilon$. Further, this computation factors as follows.

$$\langle q_i, \flat\rangle \xrightarrow{\varepsilon} \langle i_0, \# \cdot \flat\rangle \xrightarrow{\vec{x}} \langle p, \flat\rangle \xrightarrow{\varepsilon} \langle q_f, \varepsilon\rangle$$

Here $p \in Q$, whence $p \neq q_f, q_i$. But every $\mathfrak{M}$–transition from i_0 to

p is also a $\mathfrak{K}$–transition. Hence there is a $\mathfrak{K}$–computation $\langle i_0, \# \rangle \xrightarrow{\vec{x}} \langle p, \varepsilon \rangle$. From this follows $\vec{x} \in L^s(\mathfrak{K})$, and so $L^s(\mathfrak{K}) = L(\mathfrak{M})$. □

Lemma 2.3.4 *Let L be a context free language over A. Then there exists a pushdown automaton $\mathfrak{K}$ such that $L = L^s(\mathfrak{K})$.*

Proof. We take a context free grammar $G = \langle \mathtt{S}, N, A, R \rangle$ in Greibach form with $L = L(G)$. We assume that $\varepsilon \notin G$. (If $\varepsilon \in L(G)$ then a modification of the automaton of the analogous case can be given.) The automaton possesses only one state, i_0, and uses N as its stack alphabet. The beginning of the stack is $\mathtt{S}$.

$$\delta(i_0, X, x) := \{\langle i_0, \vec{Y} \rangle : X \to x \cdot \vec{Y} \in R\}$$

This defines $\mathfrak{K} := \langle \{i_0\}, i_0, A, \{i_0\}, N, \mathtt{S}, \delta \rangle$. We show that $L = L^s(\mathfrak{K})$. To this end recall that for every $\vec{x} \in L(G)$ there is a leftmost derivation. In a grammar in Greibach form every leftmost derivation derives strings of the form $\vec{y} \cdot \vec{Y}$. Now one shows by induction that $G \vdash \vec{y} \cdot \vec{Y}$ if and only if $\langle i_0, \vec{Y} \rangle \in \delta(i_0, \mathtt{S}, \vec{y})$. □

Lemma 2.3.5 *Let $\mathfrak{K}$ be a pushdown automaton. Then $L^s(\mathfrak{K})$ is context free.*

Proof. Let $\mathfrak{K} = \langle Q, i_0, A, F, D, \#, \delta \rangle$ be a pushdown automaton. Let $N = Q \times D \times (Q \cup \{\mathtt{S}\})$, where $\mathtt{S}$ is a new symbol. $\mathtt{S}$ shall also be the start symbol. We write a general element of N in the form $[q, A, p]$. Now we define $R := R^s \cup R^\delta \cup R^\varepsilon$, where

$$\begin{array}{rcl} R^s & := & \{\mathtt{S} \to [i_0, \#, q] : q \in Q\} \\ R^\delta & := & \{[p, Z, q] \to x[q_0, Y_0, q_1][q_1, Y_1, q_2] \ldots [q_{m-1}, Y_{m-1}, q] : \\ & & \qquad \langle q_0, Y_0 Y_1 \ldots Y_{m-1} \rangle \in \delta(p, Z, x)\} \\ R^\varepsilon & := & \{[p, Z, q] \to [p, Y_0, q_1][q_1, Y_1, q_2] \ldots [q_{m-1}, Y_{m-1}, q] : \\ & & \qquad \langle p, Y_0 Y_1 \ldots Y_{m-1} \rangle \in \delta(p, Z, \varepsilon)\} \end{array}$$

The grammar thus defined is called $G(\mathfrak{A})$. Notice that in the definition of R^δ the case $m = 0$ is admitted; in this case we get a rule of the form $[p, Z, q] \to x$ for every p, q, x, Z with $\langle q, \varepsilon \rangle \in$

$\delta(p, Z, x)$. We now show that for every $\vec{x} \in A^*$, every $p, q \in Q$ as well as every $Z \in D$

$$(\dagger) \qquad [p, Z, q] \vdash_G \vec{x} \quad \Leftrightarrow \quad \langle q, \varepsilon \rangle \in \delta(p, Z, \vec{x})$$

This suffices for the proof. For if $\vec{x} \in L(G)$ then we have $[i_0, \#, q] \vdash_G \vec{x}$ and so because of (†) $\langle q, \varepsilon \rangle \in \delta(i_0, \#, \vec{x})$, which means nothing but $\vec{x} \in L^s(\mathfrak{K})$. And if the latter holds then we have $[i_0, \#, q] \vdash_G \vec{x}$ and so $\mathtt{S} \vdash_G \vec{x}$, which is nothing else but $\vec{x} \in L(G)$.

Now we show (†). It is clear that (†) follows from (‡).

$$(\ddagger) \qquad \begin{array}{l} [p, Z, q] \vdash_G^{\ell} \vec{y} \cdot [q_0, Y_0, q_1][q_1, Y_1, q_2] \ldots [q_{m-1}, Y_{m-1}, q] \\ \quad \Leftrightarrow \quad \langle q_0, Y_0 Y_1 \ldots Y_{m-1} \rangle \in \delta(p, Z, \vec{y}) \end{array}$$

(‡) is proved by induction. □

On some reflection it is seen that for every automaton $\mathfrak{K}$ there is an automaton $\mathfrak{L}$ with only one accepting state which accepts the same language. If one takes $\mathfrak{L}$ in place of $\mathfrak{K}$ then there is no need to use the trick with a new start symbol. Said in another way, we may choose $[i_0, \#, q]$ as a start symbol where q is the accepting state of $\mathfrak{L}$.

Theorem 2.3.6 (Chomsky) *The context free languages are exactly the languages which are accepted by a pushdown automaton, either by state or by stack.*

From this proof we can draw some further conclusions. The first conclusion is that for every pushdown automaton $\mathfrak{K}$ we can construct a pushdown automaton $\mathfrak{L}$ for which $L^s(\mathfrak{L}) = L^s(\mathfrak{K})$ and which contains no ε–moves. Also, there exists a pushdown automaton $\mathfrak{M}$ such that $L^s(\mathfrak{M}) = L^s(\mathfrak{K})$ and which contains only one state, which is at the same time an initial and an accepting state. For such an automaton these definitions reduce considerably. Such an automaton possesses as a memory only a string. The transition function can be reduced to a function ζ from $A \times D^*$ into finite subsets of D^*. (We do not allow ε–transitions.)

The pushdown automaton runs along the string from left to right. It recognizes in linear time whether or not a string is in the language. However, the automaton is nondeterministic. The concept of a deterministic automaton is defined as follows.

Definition 2.3.7 *A pushdown automaton $\mathfrak{K} = \langle Q, i_0, A, F, D, \#, \delta\rangle$ is* ***deterministic*** *if for every $q \in Q$, $Z \in D$ and $x \in A_\varepsilon$ we have $|\delta(q, Z, x)| \leq 1$ and for all $q \in Q$ and all $Z \in D$ either (a) $\delta(q, Z, \varepsilon) = \varnothing$ or (b) $\delta(q, Z, a) = \varnothing$ for all $a \in A$. A languages L is called* ***deterministic*** *if $L = L(\mathfrak{A})$ for a deterministic automaton $\mathfrak{A}$. The set of deterministic languages is denoted by Δ.*

Deterministic languages are such languages which are accepted by a deterministic automaton by state. Now, is it possible to build a deterministic automaton accepting that language just like regular languages? The answer is negative. To this end we consider the **mirror language** $\{\vec{x}\,\vec{x}^T : \vec{x} \in A^*\}$. This language is surely context free. There are, however, no deterministic automata that accept it. To see this one has to realize that the automaton will have to put into the stack the string $\vec{x}\,\vec{x}^T$ at least up to $\vec{x}$ in order to compare it with the remaining word, $\vec{x}^T$. The machine, however, has to guess when the moment has come to change from putting onto stack to removing from stack. The reader may reflect that this is not possible without knowing the entire word.

Theorem 2.3.8 *Deterministic languages are in* ***DTIME***(n).

The proof is left as an exercise.

We have seen that also regular languages are in $\mathbf{DTIME}(n)$. However, there are deterministic languages which are not regular. Such a language is $L = \{\vec{x}\mathtt{c}\vec{x}^T : \vec{x} \in \{\mathtt{a}, \mathtt{b}\}^*\}$. In contrast to the mirror language L is deterministic. For now the machine does not have to guess where the turning point is: it is right after the symbol $\mathtt{c}$.

Now there is the question whether a deterministic automaton can recognize languages using the stack. This is not the case. For let L be a language such that $S = L^s(\mathfrak{K})$, $\mathfrak{K}$ a deterministic automaton. Then, if $\vec{x}\,\vec{y} \in L$ for some $\vec{y} \neq \varepsilon$ then $\vec{x} \notin L$. We say that L is **prefix free** if it has this property. For if $\vec{x} \in S$ then there exists a $\mathfrak{K}$–computation from $\langle q_0, \#\rangle$ to $\langle q, \varepsilon\rangle$. Further, since $\mathfrak{K}$ is deterministic: if $\langle q_0, \#\rangle \xrightarrow{\vec{x}} \mathfrak{z}$ then $\mathfrak{z} = \langle q, \varepsilon\rangle$. However, if the stack has been emptied the automaton cannot work further. Hence $\vec{x}\,\vec{y} \notin L$. There are deterministic languages which are not prefix free. We present an important class of such languages, the *Dyck–languages.* Let A be an alphabet consisting of r symbols. For each symbol x let $\underline{x}$ be another symbol. We write $\underline{A} := \{\underline{x} : x \in A\}$. We introduce a congruence θ on strings. It is generated by the equations

$$a\underline{a} \;\theta\; \varepsilon$$

for all $a \in A$. (The analogous equations $\underline{a}a \;\theta\; \varepsilon$ are *not* included.) A string $\vec{x} \in (A \cup \underline{A})^*$ is called **balanced** if $\vec{x} \;\theta\; \varepsilon$. $\vec{x}$ is balanced if and only if $\vec{x}$ can be rewritten into ε by successively replacing substrings of the form $x\underline{x}$ into ε.

Definition 2.3.9 $\boldsymbol{D_r}$ *denotes the set of balanced strings over an alphabet consisting of* $2r$ *symbols. A languages is called* ***Dyck–language*** *if it has the form* D_r *for some* r *(and some alphabet* $A \cup \underline{A}$*).*

The language XML (**Extensible Markup Language**, an outgrowth of HTML) embodies like no other language the features of Dyck–languages. For every string $\vec{x}$ it allows to form a pair of tags <$\vec{x}$> (opening tag) and </$\vec{x}$> (closing tag). The syntax of XML is such that the tags always come in pairs. The tags alone (not counting the text in between) form a Dyck language. What distinguishes XML from other languages is that tags can be freely formed.

Proposition 2.3.10 *Dyck–languages are deterministic but not prefix free.*

The following grammars generate the Dyck–languages:

$$\mathtt{S} \to \mathtt{SS} \mid x\mathtt{S}\underline{x} \mid \varepsilon$$

Dyck–languages are therefore context free. It is easy to see that together with $\vec{x}, \vec{y} \in D_r$ also $\vec{x}\vec{y} \in D_r$. Hence Dyck–languages are not prefix free. That they are deterministic follows from some general results which we shall establish later. We leave it to the reader to construct a deterministic automaton which recognizes D_r. This shows that the languages which are accepted by a deterministic automaton by empty stack are a proper subclass of the languages which are accepted by an automaton by state. This justifies the following definition.

Definition 2.3.11 *A language L is called* ***strict deterministic*** *if there is a deterministic automaton $\mathfrak{K}$ such that $L = L^s(\mathfrak{K})$. The class of strict deterministic languages is denoted by Δ^s.*

Theorem 2.3.12 *L is strict deterministic if L is deterministic and prefix free.*

Proof. We have seen that strict deterministic languages are prefix free. Now let L be deterministic and prefix free. Then there exists an automaton $\mathfrak{K}$ which accepts L by state. Since L is prefix free, this holds for every $\vec{x} \in L$, and for every proper prefix $\vec{y}$ of $\vec{x}$ we have that if $\langle q_0, \#\rangle \xrightarrow{\vec{y}} \langle q, \vec{Y}\rangle$ then q is not an accepting state. Thus we shall rebuild $\mathfrak{K}$ in the following way. Let $\delta_1(q, Z, x) := \delta^{\mathfrak{K}}(q, Z, x)$ if q is not accepting. Further, let $\delta_1(q, Z, x) := \varnothing$ if $q \in F$ and $x \in A$; let $\delta_1(q, Z, \varepsilon) := \{\langle q, \varepsilon\rangle\}$, $Z \in D$. Finally, let $\mathfrak{L}$ be the automaton which results from $\mathfrak{K}$ by replacing $\delta^{\mathfrak{K}}$ with δ_1. $\mathfrak{L}$ is deterministic as is easily checked. Further, an $\mathfrak{L}$–computation can be factorised into an $\mathfrak{K}$–computation followed by a deletion of the stack. We claim that $L(\mathfrak{K}) = L^s(\mathfrak{L})$. The claim then follows. So let $\vec{x} \in L(\mathfrak{K})$. Then there exists a $\mathfrak{K}$–computation using $\vec{x}$ from $\langle q_0, \#\rangle$ to $\langle q, \vec{Y}\rangle$ where $q \in F$. For no proper prefix $\vec{y}$ of $\vec{x}$ there is a computation into an accepting state since L is prefix free. So there is an $\mathfrak{L}$–computation with $\vec{x}$ from $\langle q_0, \#\rangle$ to $\langle q, \vec{Y}\rangle$. Now $\langle q, \vec{Y}\rangle \xrightarrow{\varepsilon} \langle q, \varepsilon\rangle$ and so $\vec{x} \in L^s(\mathfrak{L})$. Conversely, assume $\vec{x} \in L^s(\mathfrak{L})$. Then there is a computation $\langle q_0, \#\rangle \xrightarrow{\vec{x}} \langle q, \varepsilon\rangle$. Let $\vec{Y} \in D^*$ be the longest string such that $\langle q_0, \#\rangle \xrightarrow{\vec{x}} \langle q, \vec{Y}\rangle$. Then the $\mathfrak{L}$–step before reaching $\langle q, \vec{Y}\rangle$ is a $\mathfrak{K}$–step. So there is a $\mathfrak{K}$–computation for $\vec{x}$ from $\langle q_0, \#\rangle$ to $\langle q, \vec{Y}\rangle$, and so $\vec{x} \in L(\mathfrak{K})$. □

The proof of this theorem also shows the following.

Theorem 2.3.13 *Let U be a deterministic context free language. Let L be the set of all $\vec{x} \in U$ for which no proper prefix is in U. Then L is strict deterministic.*

For the following definition we make the following agreement, which shall be used quite often in the sequel. We denote by ${}^{(k)}\vec{\alpha}$ the prefix of $\vec{\alpha}$ of length k in case $\vec{\alpha}$ has length at least k; otherwise ${}^{(k)}\vec{\alpha} := \vec{\alpha}$.

Definition 2.3.14 *Let $G = \langle \mathtt{S}, N, A, R\rangle$ be a grammar and $\Pi \subseteq \wp(N \cup A)$ a partition. We write $\alpha \equiv \beta$ if there is an $M \in \Pi$ such that $\alpha, \beta \in M$. Π is called **strict for G**, if the following holds.*

1. *$A \in \Pi$*
2. *For $C, C' \in N$ and $\vec{\alpha}, \vec{\gamma}_1, \vec{\gamma}_2 \in (N \cup A)^*$: if $C \equiv C'$ and $C \to \vec{\alpha}\,\vec{\gamma}_1$ as well as $C' \to \vec{\alpha}\,\vec{\gamma}_2 \in R$ then either*
   (a) *$\vec{\gamma}_1, \vec{\gamma}_2 \neq \varepsilon$ and ${}^{(1)}\vec{\gamma}_1 \equiv {}^{(1)}\vec{\gamma}_2$ or*
   (b) *$\vec{\gamma}_1 = \vec{\gamma}_2 = \varepsilon$ and $C = C'$.*

Definition 2.3.15 *A context free grammar G is called **strict deterministic** if there is a strict partition for G.*

We look at the following example (taken from (Harrison, 1978)):

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{aA} \mid \mathtt{aB} \\ \mathtt{A} & \to & \mathtt{aAa} \mid \mathtt{bC} \\ \mathtt{B} & \to & \mathtt{aB} \mid \mathtt{bD} \\ \mathtt{C} & \to & \mathtt{bC} \mid \mathtt{a} \\ \mathtt{D} & \to & \mathtt{bDc} \mid \mathtt{c} \end{array}$$

$\Pi = \{\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{\mathtt{S}\}, \{\mathtt{A}, \mathtt{B}\}, \{\mathtt{C}, \mathtt{D}\}\}$ is a strict partition. The language generated by this grammar is $\{\mathtt{a}^n\mathtt{b}^k\mathtt{a}^n, \mathtt{a}^n\mathtt{b}^k\mathtt{c}^k : k, n \geq 1\}$.

We shall now show that the languages generated by strict deterministic grammars are exactly the strict deterministic languages. This justifies the terminology in retrospect. To begin, we shall draw a few conclusions from the definitions. A strict deterministic grammar is in a sense invertible. If $G = \langle \mathtt{S}, N, A, R\rangle$ is strict deterministic and $R' \subseteq R$ then $G' = \langle \mathtt{S}, N, A, R'\rangle$ is strict deterministic as well. Therefore for a strict deterministic grammar we can construct a weakly equivalent strict deterministic slender grammar. We denote by $\vec{\alpha} \Rightarrow^n_L \vec{\gamma}$ the fact that there is a leftmost derivation of length n of $\vec{\gamma}$ from $\vec{\alpha}$.

Lemma 2.3.16 *Let G be a context free grammar with a strict partition Π. Then the following is true. For $C, C' \in N$ and $\vec{\alpha}, \vec{\gamma}_1, \vec{\gamma}_2 \in (N \cup A)^*$: if $C \equiv C'$ and $C \Rightarrow^n_L \vec{\alpha}\,\vec{\gamma}_1$ as well as $C' \Rightarrow^n_L \vec{\alpha}\,\vec{\gamma}_2$ then either*

1. *$\vec{\gamma}_1, \vec{\gamma}_2 \neq \varepsilon$ and ${}^{(1)}\vec{\gamma}_1 \equiv {}^{(1)}\vec{\gamma}_2$ or*

2. *$\vec{\gamma}_1 = \vec{\gamma}_2 = \varepsilon$ and $C = C'$.*

The proof is an easy induction over the length of the derivation.

Lemma 2.3.17 *Let G be a slender strict deterministic grammar. Then if $C \Rightarrow^+_L D\,\vec{\alpha}$ we have $C \not\equiv D$.*

Proof. Assume $C \Rightarrow^n_L D\,\vec{\alpha}$. Then because of Lemma 2.3.16 we have for all $k \geq 1$: $C \Rightarrow^{kn}_L D\,\vec{\gamma}$ for some $\vec{\gamma}$. From this it follows that there is no terminating leftmost derivation from C. This contradicts the fact that G is slender. □

It follows that a strict deterministic grammar is not left recursive, that is, $A \Rightarrow^+_L A\,\vec{\alpha}$ cannot hold. We can construct a Greibach form for G in the following way. Let $\rho = C \to \alpha\,\vec{\gamma}$ be a rule. If $\alpha \notin A$ then we skip ρ be replacing it with the set of all rules $C \to \vec{\eta}\,\vec{\gamma}$ such that $\alpha \to \vec{\eta} \in R$. Then Lemma 2.3.16 assures us that Π is a strict partition also for the new grammar. This operation we repeat as often as necessary. Since G is not left recursive, this process terminates.

Theorem 2.3.18 *For every strict deterministic grammar G there is a strict deterministic grammar H in Greibach form such that $L(G) = L(H)$.*

Now for the promised correspondence between strict deterministic languages and strict deterministic grammars.

Lemma 2.3.19 *Let L be strict deterministic. Then there exists a deterministic automaton with a single accepting state which accepts L by stack.*

Proof. Let $\mathfrak{A}$ be given. We add a new state q into which the automaton changes as soon as the stack is empty. ☐

Lemma 2.3.20 *Let $\mathfrak{A}$ be a deterministic automaton with a single accepting state. Then $G(\mathfrak{A})$ is strict deterministic.*

Proof. Let $\mathfrak{A} = \langle Q, i_0, A, F, D, \#, \delta \rangle$. By the preceding lemma we may assume that $F = \{q_f\}$. Now let $G(\mathfrak{A})$ defined as in Proposition 2.3.5. Put

$$\alpha \equiv \beta \qquad \Leftrightarrow \qquad \begin{cases} & \alpha, \beta \in A \\ or & \alpha = [q, Z, q'], \beta = [q, Z, q''] \\ & \quad \textit{for some } q, q', q'' \in Q, Z \in D. \end{cases}$$

We show that $\equiv$ is a strict partition. To this end, let $[q, Z, q'] \to \vec{\alpha}\vec{\gamma}_1$ and $[q, Z, q''] \to \vec{\alpha}\vec{\gamma}_2$ be two rules. Assume first of all $\vec{\gamma}_1, \vec{\gamma}_2 \neq \varepsilon$. Case 1. $\vec{\alpha} = \varepsilon$. Consider $\zeta_i := {}^{(1)}\vec{\gamma}_i$. If $\zeta_1 \in A$ then also $\zeta_2 \in A$, since $\mathfrak{A}$ is deterministic. If on the other hand $\zeta_1 \notin A$ then we have $\zeta_1 = [q, Y_0, q_1]$ and $\zeta_2 = [q, Y_0, q_1']$, and so $\zeta_1 \equiv \zeta_2$. Case 2. $\vec{\alpha} \neq \varepsilon$. Let then $\eta := {}^{(1)}\vec{\alpha}$. If $\eta \in A$, then we now have $\zeta_1 = [q_i, Y_i, q_{i+1}]$ and $\zeta_2 = [q_i, Y_i, q_{i+1}']$ for some $q_i, q_{i+1}, q_{i+1}' \in Q$. This completes this case.

Assume now $\vec{\gamma}_1 = \varepsilon$. Then $\vec{\alpha}\vec{\gamma}_1$ is a prefix of $\vec{\alpha}\vec{\gamma}_2$. Case 1. $\vec{\alpha} = \varepsilon$. Then $\vec{\alpha}\vec{\gamma}_2 = \varepsilon$, hence $\vec{\gamma}_2 = \varepsilon$. Case 2. $\vec{\alpha} \neq \varepsilon$. Then it is easy to see that $\vec{\gamma}_2 = \varepsilon$. Hence in both cases we have $\vec{\gamma}_2 = \varepsilon$, and so $q' = q''$. This shows the claim. ☐

Theorem 2.3.21 *Let L be a strict deterministic language. Then there exists a strict deterministic grammar G such that $L(G) = L$.*

The strategy to put a string onto the stack and then subsequently remove it from there has prompted the following definition. A **stack move** is a move where the machine writes a symbol onto the stack or removes a symbol from the stack. (So the stack either increases in length or it decreases.) The automaton is said to make a **turn** if in the last stack move it increased the stack and now it decreases it or, conversely, in the last stack move it diminishes the stack and now increases it.

Definition 2.3.22 *A language L is called an n–**turn language** if there is a pushdown automaton which recognizes every string from L with at most n turns. L is **ultralinear** if it is an n–turn language for some $n \in \omega$.*

Notice that a context free language is n–turn exactly if there is an automaton which accepts L and in which for every string $\vec{x}$ *every* computation needs at most n turns. For given any automaton $\mathfrak{K}$ which recognizes L, we build another automaton $\mathfrak{L}$ which has the same computations as $\mathfrak{K}$ except that they are terminated before the $n+1$st turn. This is achieved by adding a memory that counts the number of turns.

We shall not go into the details of ultralinear languages. One case is worth noting, that of 1–turn languages. A context free grammar is called **linear** if in every rule $X \to \vec{\alpha}$ the string $\vec{\alpha}$ contains at most one occurrence of a nonterminal symbol. A language is **linear** if it is generated by a linear grammar.

Theorem 2.3.23 *A context free language L is linear if and only if it is 1–turn.*

Proof. Let G be a linear grammar. Without loss of generality a rule is of the form $X \to aY$ or $X \to Ya$. Further, there are rules of the form $X \to \varepsilon$. We construct the following automaton. $D := \{\#\} \cup N$, where $\#$ is the beginning of the stack, $Q := \{+, -, q\}$,

$i_0 := +$, $F := \{q\}$. Further, for $x \in A$ we put $\delta(+, X, x) := \{\langle +, Y\rangle\}$ if $X \to xY \in R$ and $\delta(+, X, \varepsilon) := \{\langle +, Y\rangle\}$ if $X \to Yx \in R$; let $\delta(-, Y, x) := \{\langle -, \varepsilon\rangle\}$ if $X \to Yx \in R$. And finally $\delta(\langle +, X, x\rangle) := \{\langle -, \varepsilon\rangle\}$ if $X \to x \in R$. Finally, $\delta(-, \#, \varepsilon) := \{\langle q, \varepsilon\rangle\}$. This defines the automaton $\mathfrak{K}(G)$. It is not hard to show that $\mathfrak{K}(G)$ only admits computations without stack moves. For if the automaton is in state $+$ the stack may not decrease unless the automaton changes into the state $-$. If it is in $-$, the stack may not increase and it may only be changed into a state $-$, or, finally, into q. We leave it to the reader to check that $L(\mathfrak{K}(G)) = L(G)$. Therefore $L(G)$ is a 1–turn language. Conversely, let $\mathfrak{K}$ be an automaton which allows computations with at most one turn. It is then clear that if the stack is emptied the automaton cannot put anything on it. The automaton may only fill the stack and later empty it. Let us consider the automaton $G(\mathfrak{K})$ as defined above. Then all rules are of the form $X \to x\vec{Y}$ with $x \in A_\varepsilon$. Let $\vec{Y} = Y_0 Y_1 \ldots Y_{n-1}$. We claim that every Y_i–production for $i > 0$ is of the form $Y_i \to a$ or $Y_i \to X$. If not, there is a computation in which the automaton makes two turns, as we have indicated above. (This argument makes tacit use of the fact that the automaton possesses a computation where it performs a transition to $Y_i = [p, X, q]$ that is to say, that it goes from p to q where X is the topmost stack symbol. If this is not the case, however, then the transitions can be eliminated without harm from the automaton.) Now it is easy to eliminate the rules of the form $Y_i \to X$ by skipping them. Subsequent skipping of the rules $Y_i \to a$ yields a linear grammar. □

The automata theoretic analyses suggest that the recognition problem for context free languages must be quite hard. However, this is not the case. It turns out that the recognition and parsing problem are solvable in $O(n^3)$ steps. To see this, let a grammar G be given. We assume without loss of generality that G is in Chomsky normal form. Let $\vec{x}$ be a string of length n. As a first step we try to list all substrings which are constituents, together with their category. If $\vec{x}$ is a constituent of category S then $\vec{x} \in L(G)$;

if it is not, then $\vec{x} \notin L(G)$. In order to enumerate the substrings we make use an $(n+1) \times (n+1)$–matrix whose entries are subsets of N. Such a matrix is called a **chart**. Every substring is defined by a pair $\langle i, j\rangle$ of numbers, where $0 \leq i < j \leq n+1$. In the cell $\langle i, j\rangle$ we enter all $X \in N$ for which the substring $x_i x_{i+1} \dots x_{j-1}$ is a constituent of category X. In the beginning the matrix is empty. Now we start by filling the matrix inductively, in increasing length $d := j - i$. We have $0 < d < n+1$. So, we begin with the substrings of length 1, $i = 0$. Then we move on to $i := 1$, $i := 2$ and so on. Then we set $d := 2$ and $i := 0$, $i := 1$ etc. We consider the pair $\langle d, i\rangle$. The substring $x_i \dots x_{i+d}$ is a constituent of category X if and only if it decomposes into substrings $\vec{y} = x_i \dots x_{i+e}$ and $\vec{z} = x_{i+e+1} \dots x_{i+d}$ such that there is a rule $X \to YZ$ where $\vec{y}$ is a constituent of category Y and $\vec{z}$ is a constituent of category Z. This means, the set of all $X \in N$, which we enter at $\langle i, i+d\rangle$ is computed from all decompositions into substrings. There are $d-1 \leq n$ such decomposition. For every decomposition the computational effort is limited and depends only on a constant c_G whose value is determined by the grammar. For every pair we need $c_G \cdot (n+1)$ steps. Now there exist $\binom{n}{2}$ proper subwords. Hence the effort is bounded by $c_G \cdot n^3$.

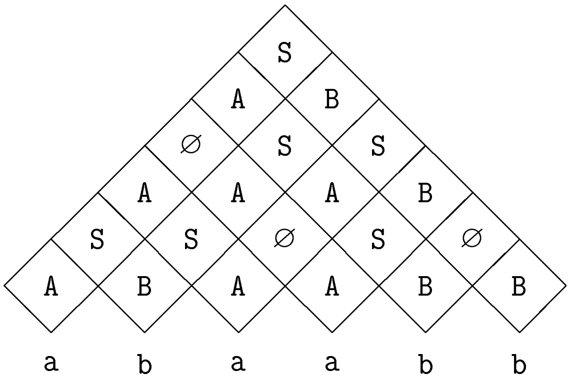
In Figure 2.1 we have shown the computation of a chart based on the word `abaabb`. Since the grammar is invertible any substring has at most one category. In general, this need not be the case. (Because of Theorem 2.2.13 we can always assume the grammar to be invertible.)

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{SS} \mid \mathtt{AB} \mid \mathtt{BA} \\ \mathtt{A} & \to & \mathtt{AS} \mid \mathtt{SA} \mid \mathtt{a} \\ \mathtt{B} & \to & \mathtt{BS} \mid \mathtt{SB} \mid \mathtt{b} \end{array}$$

The construction of the chart is as follows. Let $C_{\vec{x}}(i, j)$ be the set of all nonterminals X such that $X \vdash_G x_i x_{i+1} \dots x_{j-1}$. Further, for two nonterminals X and Y $X \odot Y := \{Z : Z \to XY \in R\}$ and for sets $\mathbb{U}, \mathbb{V} \subseteq N$ let

$$\mathbb{U} \odot \mathbb{V} := \bigcup \langle X \odot Y : X \in \mathbb{U}, Y \in \mathbb{V}\rangle$$

Figure 2.1: A Chart for `abaabb`



Now we can compute $C_{\vec{x}}(i, j)$ inductively. The induction parameter is $j - i$. If $j - i = 1$ then $C_{\vec{x}}(i, j) = \{X : X \to x \in R\}$. If $j - i > 1$ then the following equation holds.

$$C_{\vec{x}}(i, j) = \bigcup_{i<k<j} C_{\vec{x}}(i, k) \odot C_{\vec{x}}(k, j)$$

We always have $j - k, k - i < j - i$. Now let $\vec{x} \in L(G)$. How can we find a derivation for $\vec{x}$? To that end we use the fully computed chart. We begin with $\vec{x}$ and decompose it in an arbitrary way; since $\vec{x}$ has the category $\mathtt{S}$, there must be a rule $\mathtt{S} \to XY$ and a decomposition into $\vec{x}$ of category X and $\vec{y}$ of category Y. Or $\vec{x} = a \in A$ and $\mathtt{S} \to a$ is a rule. If the composition has been found, then we continue with the substrings $\vec{x}$ and $\vec{y}$ in the same way. Every decomposition needs some time, which only depends on G. A substring of length i has $i \leq n$ decompositions. In our analysis we have at most $2n$ substrings. This follows from the fact that in a properly branching tree with n leaves there are at most $2n$ nodes. In total we need time at most $d_G \cdot n^2$ for a certain constant

d_G which only depends on G.

From this it follows that in general, even if the grammar is not in Chomsky standard form the recognition and analysis only needs $O(n^3)$ steps where at the same time we only need $O(n^2)$ cells. For let G be given. Now transform G into 2–standard form into the grammar G^2. Since $L(G^2) = L(G)$, the recognition problem for G is solvable in the same amount of time as G^2. One needs $O(n^2)$ steps to construct a chart for $\vec{x}$. One also needs an additional $O(n^2)$ steps in order to create a G–tree for $\vec{x}$ and $O(n)$ steps to turn this into a derivation.

However, this is not already a proof that the problem is solvable in $O(n^3)$ steps and $O(n^2)$ space, for we need to find a Turing machine which solves the problem in the same time and space. This is possible; this has been shown independently by Cocke, Kasami and Younger.

Theorem 2.3.24 (Cocke, Kasami, Younger) *Context free languages have the following multitape complexity.*

1. $\mathrm{CFS} \subseteq \boldsymbol{DTIME}(n^3)$.
2. $\mathrm{CFS} \subseteq \boldsymbol{DSPACE}(n^2)$.

Proof. We construct a deterministic 3 tape Turing machine which only needs $O(n^2)$ space and $O(n^3)$ time. The essential trick consists in filling the tape. Also, in addition to the alphabet A we need an auxiliary alphabet consisting of `B` and `Q` as well as for every $U \subseteq N$ a symbol $[U]$ and a symbol $[U]^{\surd}$. On Tape 1 we have the input string, $\vec{x}$. Put $C(i, j) := C_{\vec{x}}(i, j)$. Let $\vec{x}$ have length n. On Tape 1 we construct a sequence of the following form.

$$\mathtt{QB}^n\mathtt{QB}^{n-1}\mathtt{Q}\ldots\mathtt{QBBQBQ}$$

This is the skeleton of the chart. We call a sequence of `B`s in between two `Q`s a **block**. The first block is being filled as follows. The string $\vec{x}$ is deleted step by step and the sequence $\mathtt{B}^n$ is being replaced by the sequence of the $C(i, i+1)$. This procedure requires

$O(n^2)$ steps. For every d from 1 to $n-1$ we shall fill the $d+1$th block. So, let d be given. On Tape 2 we write the sequence

$$\begin{aligned}&\mathtt{Q}[C(0,1)][C(0,2)]\ldots[C(0,d)]\\&\quad\mathtt{Q}[C(1,2)][C(1,3)]\ldots[C(1,d+1)]\\&\qquad\ldots\\&\quad\mathtt{Q}[C(n-d,n-d+1)][C(n-d,n-d+2)]\ldots[C(n-d,n)]\mathtt{Q}\end{aligned}$$

On Tape 3 we write the sequence

$$\begin{aligned}&\mathtt{Q}[C(0,d)][C(1,d)]\ldots[C(d-1,d)]\\&\quad\mathtt{Q}[C(1,d+1)][C(2,d+1)]\ldots[C(d,d+1)]\\&\qquad\ldots\\&\quad\mathtt{Q}[C(n-d,n)][C(n-d+1,n)]\ldots[C(n-1,n)]\mathtt{Q}\end{aligned}$$

From this sequence we can compute the $d+1$st block quite fast. The automaton has to traverse the first block on Tape 2 and the second block on Tape 3 cogradiently and memorize the result of $C(0,j)\odot C(j,d+1)$. When it reaches the end it has computed $C(0,d+1)$ and can enter it on Tape 1. Now it moves on to the next block on the second and the third tape and computes $C(1,d+2)$. And so on. It is clear that the computation is linear in the length of the Tape 2 (and the Tape 3) and therefore needs $O(n^2)$ time. At the end of this procedure Tape 2 and 3 are emptied. Also this needs quadratic time. At the end we need to consider that the filling of Tapes 2 and 3 needs $O(n^2)$ time. Then for every d the time consumption is at most $O(n^2)$ and in total $O(n^3)$. For this we first write $\mathtt{Q}$ and position the head of Tape 1 on the element $[C(0,1)]$. We write $[C(0,1)]$ onto Tape 2 and $[C(0,1)]^{\checkmark}$ onto Tape 1. (So, we 'tick off' the symbol. This helps us to remember what we did.) Now we advance to $[C(1,2)]$ copy the result onto Tape 2 and replace it by $[C(1,2)]^{\checkmark}$. And so on. This only needs linear time; for the symbols $[C(i,i+1)]$ we recognize because they are placed before the $\mathtt{Q}$. If we are ready we write $\mathtt{Q}$ onto Tape 2 and move on Tape 1 on to the beginning and then to the first symbol to the right of a 'ticked off' symbol. This is $[C(1,2)]$. We copy this symbol onto Tape 2 and tick it off. Now we move on to the next symbol to the right of the symbol which has been ticked off, copy it and tick it off. In this way Tape 2 is filled in quadratic time. At last the symbols that have been ticked off are being ticked 'on', which needs $O(n^2)$ time. Analogously the Tape 3 is filled. □

Exercise 56. Prove Proposition 2.3.2.

Exercise 57. Prove Theorem 2.3.8. *Hint.* Show that the number of ε–moves of an automaton $\mathfrak{A}$ in scanning of the string $\vec{x}$ is bounded by $k_{\mathfrak{A}} \cdot |\vec{x}|$, where $k_{\mathfrak{A}}$ is a number that depends only on $\mathfrak{A}$. Now code the behaviour of an arbitrary pushdown automaton using a 2–tape Turing machine and show that to every move of the pushdown automaton corresponds a bounded number of steps of the Turing machine.

Exercise 58. Show that a context free language is 0–turn if and only if it is regular.

Exercise 59. Give an algorithm to code a chart onto the tape of a Turing machine.

Exercise 60. Sketch the behaviour of a deterministic Turing machine which recognizes a given context free language using $O(n^2)$ space.

Exercise 61. Show the following: $\{\vec{w}\,\vec{w}^T : \vec{w} \in A^*\}$ is context free but not deterministic.

Exercise 62. Construct a deterministic automaton which recognizes a given Dyck–language.

Exercise 63. Prove Theorem 2.3.13.

2.4 Ambiguity, Transparency and Parsing Strategies

In this section we will deal with the relationship between strings and trees. As we have explained in Section 1.6, there is a bijective correspondence between derivations in G and derivations in the corresponding graph grammar γG. Moreover, every derivation $\Delta = \langle A_i : i < p \rangle$ of G defines an exhaustively ordered tree $\mathfrak{B}$ with labels in $N \cup A$ whose associated string is exactly $\vec{\alpha}_p$, where $A_{p-1} = \langle \vec{\alpha}_{p-1}, C_{p-1}, \vec{\alpha}_p \rangle$. If $\vec{\alpha}_p$ is not a terminal string, the labels of the leaves are also not all terminal. We call such a tree a **partial** G–**tree**.

Definition 2.4.1 *Let G be a context free grammar. $\vec{\alpha}$ is called a G–**constituent of category** A if $A \vdash_G \vec{\alpha}$. Let $\mathfrak{B}$ be a G–tree with associated string $\vec{x}$ and $\vec{y}$ a substring of $\vec{x}$. Assume further that $\vec{y}$ is a G–constituent of category A and $\vec{x} = D(\vec{y})$. The occurrence D of $\vec{y}$ in $\vec{x}$ is called an **accidental G–constituent of category** A **in** $\mathfrak{B}$ if it is not a G–constituent of category A in $\mathfrak{B}$.*

We shall illustrate this terminology with an example. Let G be the following grammar.

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{SS} \mid \mathtt{AB} \mid \mathtt{BA} \\ \mathtt{A} & \to & \mathtt{AS} \mid \mathtt{SA} \mid \mathtt{a} \\ \mathtt{B} & \to & \mathtt{BS} \mid \mathtt{SB} \mid \mathtt{b} \end{array}$$

The string $\vec{x}$ = abaabb has several derivations, which generate among other the following bracketing analyses.

$$(\mathtt{a}(\mathtt{b}(\mathtt{a}((\mathtt{ab})\mathtt{b})))), \quad ((\mathtt{ab})(((\mathtt{a}(\mathtt{ab}))\mathtt{b})))$$

We now list all G–constituents which occur in $\vec{x}$:

$$\begin{array}{l} \mathtt{A} : \mathtt{a}, \mathtt{aab}, \mathtt{aba}, \mathtt{baa}, \mathtt{abaab} \\ \mathtt{B} : \mathtt{b}, \mathtt{abb} \\ \mathtt{S} : \mathtt{ab}, \mathtt{aabb}, \mathtt{abaabb} \end{array}$$

Some constituents occur several times, for example ab in $\langle \varepsilon, \mathtt{aabb} \rangle$ and $\langle \mathtt{aba}, \mathtt{b} \rangle$. Now we look at the first bracketing, $(\mathtt{a}(\mathtt{b}(\mathtt{a}((\mathtt{ab})\mathtt{b}))))$. The constituents are a (context: $\langle \varepsilon, \mathtt{baabb} \rangle$, $\langle \mathtt{ab}, \mathtt{abb} \rangle$, $\langle \mathtt{aba}, \mathtt{bb} \rangle$), b, ab (for example in the context: $\langle \mathtt{aba}, \mathtt{b} \rangle$), abb in the context $\langle \mathtt{aba}, \varepsilon \rangle$, aabb, baabb and abaabb. These are the constituents of the tree. The occurrence $\langle \varepsilon, \mathtt{aabb}\rangle$ of $\mathtt{ab}$ in $\mathtt{ababb}$ is therefore an accidental occurrence of a G–constituent of category $\mathtt{S}$ in that tree. For although $\mathtt{ab}$ is a G–constituent, this occurrence in the tree is not a constituent occurrence of it. Notice that it may happen that $\vec{y}$ is a constituent of the tree $\mathfrak{B}$ but that as a G–constituent of category C it occurs accidentally since its category in $\mathfrak{B}$ is $D \neq C$.

Definition 2.4.2 *A grammar G is called* ***transparent*** *if no G–constituent occurs accidentally in a G–string. A grammar which is not transparent will be called* ***opaque****. A language for which no transparent grammar exists will be called* ***inherently opaque****.*

An example shall illustrate this. Polish Notation can be generated by a transparent grammar. For every f we add the rules

$$\begin{aligned} &\mathtt{S} \to \mathtt{F}_n \mathtt{S}^{\Omega(f)} \\ &\mathtt{F}_n \to f \qquad (\Omega(f) = n) \end{aligned}$$

This defines the grammar Π_Ω for PN_Ω. Moreover, given a string $\vec{x}$ generated by this grammar, the subterm occurrences of $\vec{x}$ under a given analysis are in one to one correspondence with the subconstituents of category $\mathtt{S}$. An occurrence of an n–ary function symbol is a constituent of type $\mathtt{F}_n$. We shall show that this grammar is not only unambiguous, it is transparent.

Let $\vec{x} = x_0 x_1 \dots x_{n-1}$ be a string. Then let $\gamma(\vec{x}) := \sum_{i<n} \gamma(x_i)$, where $\gamma(x) := -1$, $\gamma(f) := \Omega(f) - 1$.

Lemma 2.4.3 *$\vec{x} \in PN_\Omega(X)$ if and only if $\gamma(\vec{x}) = -1$ and if for every proper prefix $\vec{y}$ of $\vec{x}$ we have $\gamma(\vec{y}) \geq 0$.*

It follows from this theorem that no proper prefix of a term is a term. Now suppose that $\vec{x}$ contains an accidental occurrence of a term $\vec{y}$. Then this occurrence overlaps properly with a constituent $\vec{z}$. Without loss of generality $\vec{y} = \vec{u}\cdot\vec{v}$ and $\vec{z} = \vec{v}\cdot\vec{w}$ (with $\vec{u}, \vec{w} \neq \varepsilon$). It follows that $\gamma(\vec{v}) = \gamma(\vec{y}) - \gamma(\vec{u}) < 0$ since $\gamma(\vec{u}) \geq 0$. Hence there exists a proper prefix $\vec{u}_1$ of $\vec{u}$ such that $\vec{u}_1 = -1$. (In order to show this one must first conclude that the set $P(\vec{x}) := \{\gamma(\vec{p}) : \vec{p} \text{ is a prefix of } \vec{x}\}$ is a convex set for every term $\vec{x}$.)

Theorem 2.4.4 *The grammar Π_Ω for Polish Notation is transparent.*

Now look at the languages $\mathtt{a}^+\mathtt{b}$ and $\mathtt{a}^+$. Both are regular. A regular grammar which generates $\mathtt{a}^+\mathtt{b}$ is necessarily transparent. For constituents are: $\mathtt{a}$ and $\mathtt{a}^n\mathtt{b}$, $n \in \omega$. No constituent occurs accidentally. Now look at a regular grammar of $\mathtt{a}^+$. Any such grammar is opaque. For it generates all $\mathtt{a}^n$, which are also constituents. However, $\mathtt{aa}$ occurs in $\mathtt{aaa}$ accidentally. We can show that $\mathtt{a}^+$ not only possesses no transparent regular grammar, but that it is inherently opaque.

Proposition 2.4.5 $\mathtt{a}^+$ *is inherently opaque.*

Proof. Assume there is a context free grammar G which generates $\mathtt{a}^+$. Let $K := \{\mathtt{a}^k : \mathtt{a}^k \text{ is aG–constituent}\}$. There are at least two elements in K, say $\mathtt{a}^p$ and $\mathtt{a}^q$, $q > p$. Now there exist two occurrences of $\mathtt{a}^p$ in $\mathtt{a}^q$ which properly overlap. One of these occurrences must be accidental. □

It can be easily seen that if L is transparent and $\varepsilon \in L$, then $L = \{\varepsilon\}$. Also, a language over an alphabet consisting of a single letter can only be transparent if it contains no more than a single string. Many properties of context free grammars are undecidable. Transparency is different in this respect.

Theorem 2.4.6 (Fine) *Let G be a context free grammar. It is decidable whether or not G is transparent.*

Proof. The proof essentially follows from the theorems below. For let k_G be the constant from the Pumping Lemma (1.6.13). This constant can effectively be determined. By Lemma 2.4.7 there is an accidental occurrence of a constituent if and only if there is an accidental occurrence of a right hand side of a production. These are of the length $p + 1$ where p is the maximum productivity of a rule from G. Further, because of Lemma 2.4.9 we only need to check those constituents for accidental occurrences whose length does not exceed $p^2 + p$. This can be done in finite amount of time. □

Lemma 2.4.7 *G is opaque if and only if there is a production $\rho = A \to \vec{\alpha}$ such that $\vec{\alpha}$ has an accidental occurrence in a partial G–tree.*

Proof. Let $\vec{\varphi}$ be a string of minimal length which occurs accidentally. And let C be an accidental occurrence of $\vec{\varphi}$. Further, let $\vec{\varphi} = \vec{\gamma}_1\vec{\alpha}\vec{\gamma}_2$, and let $A \to \vec{\alpha}$ be a rule. Then two cases may occur. (A) The occurrence of $\vec{\alpha}$ is accidental. Then we have a contradiction to the minimality of $\vec{\varphi}$. (B) The occurrence of $\vec{\alpha}$ is not accidental. Then $\vec{\eta} := \vec{\gamma}_1 A\vec{\gamma}_2$ also occurs accidentally in $C(\vec{\eta})$! (We can undo the replacement $A \to \vec{\alpha}$ in the string $C(\vec{\varphi})$ since $\vec{\alpha}$ is a constituent.) Also this contradicts the minimality of $\vec{\varphi}$. So, $\vec{\varphi}$ is the right hand side of a production. □

Lemma 2.4.8 *Let G be a context free grammar without rules of productivity -1 and let $\vec{\alpha}$, $\vec{\gamma}$ be strings. Further, assume that $\vec{\gamma}$ is a G–constituent of category A in which $\vec{\alpha}$ occurs accidentally and in which $\vec{\gamma}$ is minimal in the following sense: there is no $\vec{\eta}$ of category A with (1) $|\vec{\eta}| < |\vec{\gamma}|$ and (2) $\vec{\eta} \vdash_G \vec{\gamma}$ and (3) $\vec{\alpha}$ occurs accidentally in $\vec{\eta}$. Then every constituent of length > 1 overlaps with the accidental occurrence of $\vec{\alpha}$.*

Proof. Let $\vec{\gamma} = \vec{\sigma}_1\,\vec{\eta}\,\vec{\sigma}_2$, $|\vec{\eta}| > 1$, and assume that the occurrence of $\vec{\eta}$ is a constituent of category A which does not overlap with $\vec{\alpha}$. Then $\vec{\alpha}$ occurs accidentally in $\vec{\delta} := \vec{\sigma}_1\,A\,\vec{\sigma}_2$. Further, $|\vec{\delta}| < |\vec{\gamma}|$, contradicting the minimality of $\vec{\gamma}$. □

Lemma 2.4.9 *Let G be a context free grammar where the productivity of rules is at least 0 and at most p, and let $\vec{\alpha}$ be a string of length n which occurs accidentally. Then there exists a constituent $\vec{\gamma}$ of length $\leq np$ in which $\vec{\alpha}$ occurs accidentally.*

Proof. Let $A \vdash_G \vec{\gamma}$ be minimal in the sense of the previous lemma. Then we have that every constituent of $\vec{\gamma}$ of length > 1 overlaps properly with $\vec{\alpha}$. Hence $\vec{\gamma}$ has been obtained by at most n applications of rules of productivity > 0. Hence $|\vec{\gamma}| \leq np$. □

The property of transparency is stronger than that of unique readability, also known as unambiguity. This is defined as follows.

Definition 2.4.10 *A grammar is called* ***unambiguous*** *if for every string $\vec{x}$ there is at most one G–tree whose associated string is $\vec{x}$. If G is not unambiguous, it is called* ***ambiguous****. A context free language L is called* ***inherently ambiguous*** *if every context free grammar generating it is ambiguous.*

Proposition 2.4.11 *Every transparent grammar is unambiguous.*

There exist inherently ambiguous languages. Here is an example.

Theorem 2.4.12 (Parikh) *The language* $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m : n, m \in \omega\} \cup \{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^n : n, m \in \omega\}$ *is inherently ambiguous.*

Proof. This language is context free and so there exists a context free grammar G such that $L(G) = L$. We shall show that G is ambiguous. There is a number k_G which satisfies the Pumping Lemma (1.6.13). Let $n := k_G!(:= \prod_{i=1}^{k_G} i)$. Then there exists a decomposition of $\mathtt{a}^{2n}\mathtt{b}^{2n}\mathtt{c}^{3n}$ into

$$\vec{u}_1\,\vec{x}_1\,\vec{v}_1\,\vec{y}_1\,\vec{z}_1$$

in such a way that the occurrences of $\vec{x}_1\,\vec{v}_1\,\vec{y}_1$ are no more than k_G apart from the left edge of the string. Furthermore, we can see to it that $\vec{v}_1$ has length at most k_G. As indicated earlier, $\vec{x}_1\,\vec{y}_1$ may not contain a, b and c at the same time. Also this string may not contain occurrences of c. Then we have $\vec{x}_1 = \mathtt{a}^p$ and $\vec{y} = \mathtt{b}^p$ for a certain p. We consider a maximal constituent of the form $\mathtt{a}^q\,\mathtt{b}^{q'}$. Such a constituent must exist. In it there is a constituent of the form $\mathtt{a}^{q-i}\,\mathtt{b}^{q'-i}$ for some $i < k_G$. This follows from the Pumping Lemma. Hence we can pump up $\mathtt{a}^i$ and $\mathtt{b}^i$ at the same time and get strings of the form

$$\mathtt{a}^{2p+ki}\,\mathtt{b}^{2p+ki}\,\mathtt{c}^{3q}$$

while there exists a constituent of the form $\mathtt{a}^{2p+ki-r}\,\mathtt{b}^{2p+ki-s}$ for certain $r, s \leq k_G$. In particular, for $k := p/i$ we get

$$\mathtt{a}^{3p}\,\mathtt{b}^{3p}\,\mathtt{c}^{3q}\ .$$

Now we form a decomposition of $\mathtt{a}^{3n}\mathtt{b}^{2n}\mathtt{c}^{2n}$ into

$$\vec{u}_2\,\vec{x}_2\,\vec{v}_2\,\vec{y}_2\,\vec{z}_2$$

in such a way that the occurrence of $\vec{x}_2\,\vec{v}_2\,\vec{y}_2$ is no more that k_G apart from the right edge of the string and $\vec{v}_2$ has length $\leq k_G$. Analogously we get a constituent of the form $\mathtt{b}^{2p-s'}\,\mathtt{c}^{2p-r'}$ for certain $r', s' \leq k_G$. If G is not ambiguous both are constituents in a tree for this string. But they overlap. For the left hand constituent contains $3p-s$ many occurrences of $\mathtt{b}$ and the right hand constituent contains $3p-s'$ many occurrences of $\mathtt{b}$. Since $3p-s+3p-s' = 6p-(s+s') > 3p$, these constituents must overlap. However, they are not equal. But this is impossible. So G is ambiguous. Since G was arbitrary, L is inherently ambigous. □

Now we discuss a property which is contrarious to the property of unambiguity. It says that if a right hand side occurs in a constituent, then under some different analysis this occurrence is actually a constituent occurrence.

Definition 2.4.13 *A context free grammar has the* ***NTS–property*** *if from $C \vdash_G \vec{\alpha}$, $\vec{\alpha} = \vec{\alpha}_1 \cdot \vec{\beta} \cdot \vec{\alpha}_2$ and $B \to \vec{\beta} \in R$ follows: $C \vdash_G \vec{\alpha}_1 \cdot B \cdot \vec{\alpha}_2$. A language is called an* ***NTS–language*** *if it has an NTS–grammar.*

The following grammar is not an NTS–grammar.

$$\mathtt{X} \to \mathtt{aX}, \quad \mathtt{X} \to \mathtt{a}$$

For we have $\mathtt{X} \vdash \mathtt{aa}$ but it does not hold that $\mathtt{X} \vdash \mathtt{Xa}$. In general, regular grammars are not NTS. However, we have

Theorem 2.4.14 *All regular languages are NTS–languages.*

Proof. Let L be regular. Then there exists a finite automaton $\mathfrak{A} = \langle Q, q_0, F, \delta\rangle$ such that $L = L(\mathfrak{A})$. Now let $N := \{S^\star\} \cup \{L(p,q) : p, q \in Q\}$, the start symbol being $\mathtt{S}^\star$. The rules are

$$\begin{array}{lcll}
\mathtt{S}^\star & \to & \mathtt{L}(q_0, q) & (q \in F) \\
\mathtt{L}(p,q) & \to & \mathtt{L}(p,r)\mathtt{L}(r,q) & \\
\mathtt{L}(p,q) & \to & a & (q \in \delta(p,a))
\end{array}$$

Then we have $\mathtt{L}(p,q) \vdash_G \vec{x}$ if and only if $q \in \delta(p,\vec{x})$, as is checked by induction. From this follows that $\mathtt{S}^\star \vdash_G \vec{x}$ if and only if $\vec{x} \in L(\mathfrak{A})$. Hence we have $L(G) = L$. It remains to see that G has the NTS–property. To this end let $\mathtt{L}(p,q) \vdash_G \vec{\alpha}_1 \cdot \vec{\beta} \cdot \vec{\alpha}_2$ and $\mathtt{L}(r,s) \vdash_G \vec{\beta}$. We have to show that $\mathtt{L}(p,q) \vdash_G \vec{\alpha}_1 \cdot \mathtt{L}(r,s) \cdot \vec{\alpha}_2$. In order to do this we extend the automaton $\mathfrak{A}$ to an automaton which reads strings from $N \cup A$. Here $q \in \delta(p, C)$ if and only if for every string $\vec{y}$ with $C \vdash_G \vec{y}$ we have $q \in \delta(p, \vec{y})$. Then it is clear that $q \in \delta(p, \mathtt{L}(p,q))$. Then it still holds that $\mathtt{L}(p,q) \vdash_G \vec{\alpha}$ if and only if $q \in \delta(p, \vec{\alpha})$. Hence we have $r \in \delta(p, \vec{\alpha}_1)$ and $q \in \delta(s, \vec{\alpha}_2)$. From this follows that $\mathtt{L}(p,q) \vdash_G \mathtt{L}(p,r)\mathtt{L}(r,s)\mathtt{L}(s,q)$ and finally $\mathtt{L}(p,q) \vdash_G \vec{\alpha}_1\mathtt{L}(r,s)\vec{\alpha}_2$. □

If a grammar has the NTS–property, strings can be recognized very fast. We sketch a pushdown automaton that recognizes $L(G)$. Scanning the string from left to right it puts the symbols into the stack. Using its states the automaton memorizes the content of the stack up to κ symbols deep, where κ is the length of a longest right hand side of a production. If the upper part of the stack matches a right hand side of a production $A \to \vec{\alpha}$ in the appropriate order, then $\vec{\alpha}$ is deleted from the stack and A is put on top of it. At this moment the automaton rescans the upper part of the stack up to κ symbols deep. This is done using a series of empty moves. The automaton pops κ symbols and then puts them back onto the stack. Then it continues the procedure above. It is important that the replacement of a right hand side by a left hand side is done whenever first possible.

Theorem 2.4.15 *Let G be an NTS–grammar. Then G is deterministic. Furthermore, the recognition and parsing problem are in* $\boldsymbol{DTIME}(n)$.

We shall deepen this result. To this end we abstract somewhat from the pushdown automata and introduce a calculus which manipulates pairs $\vec{\alpha} \vdash \vec{x}$ of strings separated by a turnstile. Here, we think of $\vec{\alpha}$ as the stack of the automaton and $\vec{x}$ as the string to the right of the reading head. It is not really necessary to have terminal strings on the right hand side; however, the generalization to arbitrary strings is easy to do. There are several operations. The first is called **shifting**. It simulates the reading of the first symbol.

$$\text{Shift:} \qquad \frac{\vec{\eta} \ \vdash x\vec{y}}{\vec{\eta}x \vdash \vec{y}}$$

Another operation is the so called **reduction**.

$$\text{Reduce } \rho\text{:} \qquad \frac{\vec{\eta}\vec{\alpha} \vdash \vec{x}}{\vec{\eta}X \vdash \vec{x}}$$

Here $\rho = X \to \vec{\alpha}$ must be a G–rule. This calculus shall be called the **shift–reduce–calculus for** G. The following theorem is easily proved by induction on the length of a derivation.

Theorem 2.4.16 *Let G be a context free grammar. $\vec{\alpha} \vdash_G \vec{x}$ if and only if there is a derivation of $\vec{\alpha} \vdash \varepsilon$ from $\varepsilon \vdash \vec{x}$ in the shift–reduce–calculus for G.*

This strategy can be applied to every language. We take the following grammar.

$$\begin{array}{lll} \mathtt{S} & \to & \mathtt{ASB} \mid \mathtt{c} \\ \mathtt{A} & \to & \mathtt{a} \\ \mathtt{B} & \to & \mathtt{b} \end{array}$$

Then we have $S \vdash_G$ `aacbb`. Indeed, we get a derivation shown in Table 2.1. Of course the calculus does not provide unique solutions. On many occasions we have to guess whether we want to shift or whether we want to reduce, and if the latter, then by what rule. Notice namely that if some right hand side of a production is a suffix of a right hand side of another production we have an option. We call a k–**strategy** a function f, which tells us for every pair $\vec{\alpha} \vdash \vec{x}$ whether or not we shall shift or reduce (and by which rule). Further, f shall only depend (1) on the reduction rules which can be at all applied to $\vec{\alpha}$ and (2) on the first k symbols of $\vec{x}$. We assume that in case of competition only one rule is chosen. So, a k–strategy is a map $R \times \bigcup_{i<k} A^i$ to $\{s, r\}$. If $\vec{\alpha} \vdash \vec{x}$ is given then we determine the next rule application as follows. Let $\vec{\beta}$ be

Table 2.1: A Derivation by Shifting and Reducing

ε	$\vdash$ `aacbb`
`a`	$\vdash$ `acbb`
`A`	$\vdash$ `acbb`
`Aa`	$\vdash$ `cbb`
`AA`	$\vdash$ `cbb`
`AAc`	$\vdash$ `bb`
`AAS`	$\vdash$ `bb`
`AASb`	$\vdash$ `b`
`AASB`	$\vdash$ `b`
`AS`	$\vdash$ `b`
`ASb`	$\vdash \varepsilon$
`ASB`	$\vdash \varepsilon$
`S`	$\vdash \varepsilon$

a suffix of $\vec{\alpha}$ which is reducible. If $f(\vec{\beta}, {}^{(k)}\vec{x}) = s$, then we shift; if $f(\vec{\beta}, {}^{(k)}\vec{x}) = r$ then we apply reduction $\vec{\beta}$. This is in fact not really unambigous. For a right hand side of a production may be the suffix of a right hand side of another production. Therefore, we look at another property.

($*$) If $\rho_1 = X_1 \to \vec{\beta}_1 \in R$ and $\rho_2 = X_2 \to \vec{\beta}_2 \in R$, $\rho_1 \neq \rho_2$, and if $\vec{y}$ has length $\leq k$ then $f(\vec{\beta}_1, \vec{y})$ or $f(\vec{\beta}_2, \vec{y})$ is undefined.

Definition 2.4.17 *A grammar G is called an **LR**(k)–**grammar** if not* $\mathtt{S} \Rightarrow^+ \mathtt{S}$ *and if for some $k \in \omega$ there is a k–strategy for the shift–and–reduce calculus for G. A language is called an LR(k)–**language** if it is generated by some LR(k)–grammar.*

Theorem 2.4.18 *A grammar is an LR(k)–grammar if the following holds: Suppose that $\vec{\eta}_1\vec{\alpha}_1\vec{x}_1$ and $\vec{\eta}_2\vec{\alpha}_2\vec{x}_2$ have a rightmost derivation, and with $p := |\vec{\eta}_1\vec{\alpha}_1| + k$ we have*

$${}^{(p)}\vec{\eta}_1\vec{\alpha}_1\vec{x}_1 = {}^{(p)}\vec{\eta}_2\vec{\alpha}_2\vec{x}_2 \ .$$

Then $\vec{\eta}_1 = \vec{\eta}_2$*,* $\vec{\alpha}_1 = \vec{\alpha}_2$ *and* ${}^{(k)}\vec{x}_1 = {}^{(k)}\vec{x}_2$*.*

This theorem is not hard to show. It says that the strategy may be based indeed only of the k–prefix of the string which is to be read. This is essentially the property $(*)$. One needs to convince oneself that a derivation in the shift–reduce–calculus corresponds to a rightmost derivation, provided reduction is scheduled as early as possible. In what is to follow, we shall prove two theorems which characterize $LR(k)$–languages rather exactly. For it turns out that every deterministic language is an $LR(1)$–language while the $LR(0)$–languages form a proper subclass. Since an $LR(k)$–language already is an $LR(k+1)$–language and every $LR(k)$–language is deterministic we only get two classes of languages: $LR(0)$– and $LR(1)$–languages.

Theorem 2.4.19 $LR(k)$*–languages are deterministic.*

We leave the proof of this fact to the reader. The task is to show how to extract a deterministic automaton from a strategy. The following is easy.

Lemma 2.4.20 *Every* $LR(k)$*–language is an* $LR(k{+}1)$*–language.*

So we have the following hierarchy.

$$LR(0) \subseteq LR(1) \subseteq LR(2) \subseteq LR(3) \ldots$$

This hierarchy is stationary already from $k = 1$.

Lemma 2.4.21 *Let* $k > 0$*. If* L *is an* $LR(k+1)$*–language then* L *also is an* $LR(k)$*–language.*

Proof. For a proof we construct an $LR(k)$–grammar $G^{>}$ from an $LR(k+1)$–grammar G. For simplicity we assume that G is in Chomsky normal form. The general case is easily shown in the same way. The idea behind the construction is as follows. A constituent of $G^{>}$ corresponds to a constituent of G which has been shifted one letter to the right. To implement this idea we introduce new symbols, $[a, X, b]$, where $a, b \in A$, $X \in N$, and $[a, X, \varepsilon]$, $a \in A$. The start symbol of $G^{>}$ is the start symbol of G. The rules are as follows, where a, b, c range over A.

$$\begin{array}{lcll}
\mathtt{S} & \to & \varepsilon, & \text{if } \mathtt{S} \to \varepsilon \in R, \\
\mathtt{S} & \to & a\,[a, \mathtt{S}, \varepsilon], & \\
[a, X, b] & \to & [a, Y, c]\;[c, Z, b], & X \to YZ \in R, \\
[a, X, \varepsilon] & \to & [a, Y, c]\;[c, Z, \varepsilon], & X \to YZ \in R, \\
[a, X, b] & \to & b, & X \to a \in R, \\
[a, X, \varepsilon] & \to & \varepsilon, & X \to a \in R.
\end{array}$$

By induction on the length of a derivation the following is shown.

$$[a, X, b] \vdash_{G^{>}} \vec{\alpha}\, b \qquad \Leftrightarrow \qquad X \vdash_G a\, \vec{\alpha}$$

$$[a, X, \varepsilon] \vdash_{G^{>}} \vec{\alpha} \qquad \Leftrightarrow \qquad X \vdash_G a\, \vec{\alpha}$$

From this we can deduce that $G^{>}$ is an $LR(k)$–grammar. To this end let $\vec{\eta}_1\vec{\alpha}_1\vec{x}_1$ and $\vec{\eta}_2\vec{\alpha}_2\vec{x}_2$ be rightmost derivable in $G^{>}$, and let $p := |\vec{\eta}_1\vec{\alpha}_1| + k$ as well as

$$^{(p)}\vec{\eta}_1\vec{\alpha}_1\vec{x}_1 = {}^{(p)}\vec{\eta}_2\vec{\alpha}_2\vec{x}_2 \; .$$

Then $a\vec{\eta}_1\vec{\alpha}_1\vec{x}_1 = \vec{\eta}_1'\vec{\alpha}_1' b\vec{x}_1$ for some $a, b \in A$ and some $\vec{\eta}_1'$, $\vec{\alpha}_1'$ with $a\vec{\eta}_1 = \vec{\eta}_1' c$ for $c \in A$ and $c\vec{\alpha}_1 = \vec{\alpha}_1' b$. Furthermore, we have $a\vec{\eta}_2\vec{\alpha}_2\vec{x}_2 = \vec{\eta}_2'\vec{\alpha}_2' b\vec{x}_2$, $a\vec{\eta}_2 = \vec{\eta}_2' c$ and $c\vec{\alpha}_2 = \vec{\alpha}_2'$ for certain $\vec{\eta}_2'$ and $\vec{\alpha}_2'$. Hence we have

$$^{(p+1)}\vec{\eta}_1'\vec{\alpha}_1' b\vec{x}_1 = {}^{(p+1)}\vec{\eta}_2'\vec{\alpha}_2' b\vec{x}_2$$

and $p + 1 = |\vec{\eta}_1'\vec{\alpha}_1'| + k + 1$. Furthermore, the left hand and the right hand string have a rightmost derivation in G. From this it follows, since G is an $LR(k)$–grammar, that $\vec{\eta}_1' = \vec{\eta}_2'$ and $\vec{\alpha}_1' = \vec{\alpha}_2'$, as well as $^{(k+1)}b\vec{x}_1 = {}^{(k+1)}b\vec{x}_2$. From this we get $\vec{\eta}_1 = \vec{\eta}_2$, $\vec{\alpha}_1 = \vec{\alpha}_2$ and $^{(k)}\vec{x}_1 = {}^{(k)}\vec{x}_2$, as required. □

Now we shall prove the following important theorem.

Theorem 2.4.22 *Every deterministic language is an* $LR(1)$*–language.*

The proof is relatively long winded. Before we begin we shall prove a few auxiliary theorems which establish that strictly deterministic languages are exactly the languages that are generated by strict deterministic grammars, and that they are unambiguous and in $LR(0)$. This will give us the key to the general theorem.

But now back to the strict deterministic languages. We still owe the reader a proof that strict deterministic grammars only generate strict deterministic languages. This is essentially the consequence of a property that we shall call **left transparency**. We say $\vec{\alpha}$ **occurs in** $\vec{\eta}_1\,\vec{\alpha}\,\vec{\eta}_2$ **with left context** $\vec{\eta}_1$.

Definition 2.4.23 *Let G be a grammar. G is called **left transparent** if a constituent may never occur in a string accidentally with the same left context. This means that if $\vec{x}$ is a constituent of category C in $\vec{y}_1\,\vec{x}\,\vec{y}_2$ and if $\vec{z} := \vec{y}_1\,\vec{x}\,\vec{y}_3$ is a G–string then $\vec{x}$ also is a constituent of category C in $\vec{z}$.*

For the following theorem we need a few definitions. Let $\mathfrak{B}$ be a tree and $n \in \omega$ a natural number. Then ${}^{(n)}\mathfrak{B}$ denotes the tree which consists of all nodes above the first n leaves from the left. Let P the set of leaves of $\mathfrak{B}$, say $P = \{p_i : i < q\}$, and let $p_i \sqsubset p_j$ if and only if $i < j$. Then put $N_n := \{p_i : i < n\}$, and $O_n := \uparrow N_n$. ${}^{(n)}\mathfrak{B} := \langle O_n, r, <, \sqsubset\rangle$, where $<$ and $\sqsubset$ are the relations relativized to O_n. If ℓ is a labelling function and $\mathfrak{T} = \langle \mathfrak{B}, \ell\rangle$ a labelled tree then let ${}^{(n)}\mathfrak{T} := \langle {}^{(n)}\mathfrak{B}, \ell \restriction O_n\rangle$. Again, we denote $\ell \restriction O_n$ simply by ℓ. We remark that the set $R_n := {}^{(n)}\mathfrak{B} - {}^{(n-1)}\mathfrak{B}$ is linearly ordered by $<$. We look at the largest element u from R_n. Two cases arise. (a) z has no right sister. (b) z has a right sister. In Case (a) the constituent of the mother of u is closed at the transition from ${}^{(n-1)}\mathfrak{B}$ to ${}^{(n)}\mathfrak{B}$. Say that y is at the **right edge** of $\mathfrak{T}$ if there is no z such that $y \sqsubset z$. Then $\uparrow R_n$ consists exactly of the elements which are at the right edge of ${}^{(n)}\mathfrak{B}$ and R_n consists of all those elements which are at the right edge of ${}^{(n)}\mathfrak{B}$ but not contained in ${}^{(n-1)}\mathfrak{B}$. Now the following holds.

Proposition 2.4.24 *Let G be a strict deterministic grammar.*

Then G is left transparent. Furthermore: let $\mathfrak{T}_1 = \langle \mathfrak{B}_1, \ell_1 \rangle$ and $\mathfrak{T}_2 = \langle \mathfrak{B}_2, \ell_2 \rangle$ be partial G–trees such that the following holds.

1. *If C_i is the label of the root of $\mathfrak{T}_i$ then $C_1 \equiv C_2$.*
2. *${}^{(n)}k(\mathfrak{T}_1) = {}^{(n)}k(\mathfrak{T}_2)$.*

Then there is an isomorphism $f : {}^{(n+1)}\mathfrak{B}_1 \twoheadrightarrow {}^{(n+1)}\mathfrak{B}_2$ such that $\ell_2(f(x)) = \ell_1(x)$ in case x is not at the right edge of ${}^{(n+1)}\mathfrak{B}_1$ and $\ell_2(f(x)) \equiv \ell_1(x)$ otherwise.

Proof. We show the theorem by induction on n. We assume that it holds for all $k < n$. If $n = 0$, it holds anyway. Now we show the claim for n. There exists by assumption an isomorphism $f_n : {}^{(n)}\mathfrak{B}_1 \to {}^{(n)}\mathfrak{B}_2$ satisfying the conditions given above. Again, put $R_{n+1} := {}^{(n+1)}\mathfrak{B}_1 - {}^{(n)}\mathfrak{B}_1$. At first we shall show that $\ell_2(f_n(x)) = \ell_1(x)$ for all $x \notin {\uparrow}R_{n+1}$. From this it immediately follows that $\ell_2(f_n(x)) \equiv \ell_1(x)$ for all $x \in {\uparrow}R_{n+1} - R_{n+1}$ since G is strict deterministic. This claim we show by induction on the height of x. If $h(x) = 0$, then x is a leaf and the claim holds because of the assumption that $\mathfrak{T}_1$ and $\mathfrak{T}_2$ have the same associated string. If $h(x) > 0$ then every daughter of x is in ${\uparrow}R_{n+1}$. By induction hypothesis therefore $\ell_2(f_n(y)) = \ell_1(y)$ for every $y \prec x$. Since G is strict deterministic, the label of x is uniquely fixed by this for $\ell_2(f_n(x)) \equiv \ell_1(x)$, by induction hypothesis. So we now have $\ell_2(f_n(x)) = \ell_1(x)$. This shows the first claim. Now we extend f_n to an isomorphism f_{n+1} from ${}^{(n+1)}\mathfrak{B}_1$ onto ${}^{(n+1)}\mathfrak{B}_2$ and show at the same time that $\ell_2(f_n(x)) \equiv \ell_1(x)$ for every $x \in {\uparrow}R_{n+1}$. This holds already by inductive hypothesis for all $x \notin R_{n+1}$. So, we only have to show this for $x \in R_{n+1}$. This we do as follows. Let u_0 be the largest node in R_{n+1}. Certainly, u_0 is not the root. So let v be the mother of u_0. f_n is defined on v and we have $\ell_2(f_n(v)) \equiv \ell_1(v)$. By assumption, $\ell_2(f_n(x)) = \ell_1(x)$ for all $x \sqsubset u$. So, we first of all have that there is a daughter x_0 of $f_n(v)$ which is not in the image of f_n. We choose x_0' minimal with this property. Then we put $f_{n+1}(u_0) := x_0$. Now we have $\ell_2(f_{n+1}(u_0)) \equiv \ell_1(u_0)$. We continue with u_0 in place of v. In this way we obtain a map f_{n+1} from

${}^{(n)}\mathfrak{B}_1 \cup R_{n+1} = {}^{(n+1)}\mathfrak{B}_1$ to ${}^{(n+1)}\mathfrak{B}_2$ with $\ell_2(f_{n+1}(x)) \equiv \ell_1(x)$, if $x \in R_{n+1}$ and $\ell_2(f_{n+1}(x)) = \ell_1(x)$ otherwise. That f_{n+1} is surjective is seen as follows. Suppose that u_k is the leaf of $\mathfrak{B}_1$ in R_{n+1}. Then $x_k = f_{n+1}(u_k)$ is not a leaf in $\mathfrak{B}_2$, and then there exists a x_{k+1} in ${}^{(n+1)}\mathfrak{B}_2 - {}^{(n)}\mathfrak{B}_2$. We have $\ell_2(f_{n+1}(x_k)) \equiv \ell_1(u_k)$. Let x_p be the leaf in L. By Lemma 2.3.17 $\ell_2(x_p) \not\equiv \ell_2(x_k)$ and therefore also $\ell_2(x_p) \not\equiv \ell_1(u_k)$. However, by assumption x_p is the $n+1$st leaf of $\mathfrak{B}_2$ and likewise u_k is the $n+1$st leaf of $\mathfrak{B}_1$, from which we get $\ell_1(u_k) = \ell_2(x_p)$ in contradiction to what has just been shown. □

Theorem 2.4.25 *Let G be a strict deterministic grammar. Then $L(G)$ is unambiguous. Further, G is an $LR(0)$–grammar and $L(G)$ is strict deterministic.*

Proof. The strategy of shifting and reducing can be applied as follows: every time we have identified a right hand side of a rule $X \to \vec{\mu}$ then this is a constituent of category X and we can reduce. This shows that we have a 0–strategy. Hence the grammar is an $LR(0)$–grammar. $L(G)$ is certainly unambiguous. Furthermore, $L(G)$ is deterministic, by Theorem 2.4.19. Finally, we have to show that $L(G)$ is prefix free for then by Theorem 2.3.12 is follows that $L(G)$ is strict deterministic. Now let $\vec{x}\,\vec{y} \in L(G)$. If also $\vec{x} \in L(G)$, then by Proposition 2.4.24 we must have $\vec{y} = \varepsilon$. □

At first sight it appears that Lemma 2.4.21 also holds for $k = 0$. The construction can be extended to this case without trouble. Indeed, in this case we get something of an $LR(0)$–grammar; however, it is to be noted that a strategy for $G^{>}$ does not only depend on the next symbol. Additionally, it depends on the fact whether or not the string that is yet to be read is empty. The strategy is therefore not entirely independent of the right context even though the dependency is greatly reduced. That $LR(0)$–languages are indeed more special than $LR(1)$–languages is the content of the next theorem.

Theorem 2.4.26 (Geller & Harrison) *Let L be a deterministic context free language. Then the following are equivalent.*

1. *L is an $LR(0)$–language.*

2. *If $\vec{x} \in L$, $\vec{x}\,\vec{v} \in L$ and $\vec{y} \in L$ then also $\vec{y}\,\vec{v} \in L$.*

3. *There are strict deterministic languages U and V such that $S = U \cdot V^*$.*

Proof. Assume that (1) holds, that is, let L be an $LR(0)$–language. Then there is an $LR(0)$–grammar G for L. Hence, if $X \to \vec{\alpha}$ is a rule and if $\vec{\eta}\,\vec{\alpha}\,\vec{y}$ is G–derivable then also $\vec{\eta}\,X\,\vec{y}$ is G–derivable. Using induction, this can also be shown of all pairs X, $\vec{\alpha}$ for which $X \vdash_G \vec{\alpha}$. Now let $\vec{x} \in L$ and $\vec{x}\,\vec{v} \in L$. Then $\mathtt{S} \vdash_G \vec{x}$, and so we have by the previous $\vdash_G \mathtt{S}\,\vec{v}$. Therefore we also have $\mathtt{S} \vdash_G \vec{y}$, and so $\vdash_G \vec{y}\,\vec{v}$. Hence (2) obtains. Assume now that (2) holds for L. Let U be the set of all $\vec{x} \in L$ such that $\vec{y} \notin L$ for every proper prefix $\vec{y}$ of $\vec{x}$. Let V be the set of all $\vec{v}$ such that $\vec{x}\,\vec{v} \in L$ for some $\vec{x} \in U$ but $\vec{x}\,\vec{w} \notin L$ for every $\vec{x} \in U$ and every proper prefix $\vec{w}$ of $\vec{v}$. Now, V is the set of all $\vec{y} \in V^* - \{\varepsilon\}$ for which no proper prefix is in $V^* - \{\varepsilon\}$. We show that $U \cdot V^* = L$. To this end let us prove first that $L \subseteq U \cdot V^*$. Let $\vec{u} \in S$. We distinguish two cases. (a) No proper prefix of $\vec{u}$ is in L. Then $\vec{u} \in U$, by definition of U. (b) There is a proper prefix $\vec{x}$ of $\vec{u}$ which is in L. We choose $\vec{x}$ minimally. Then $\vec{x} \in U$. Let $\vec{u} = \vec{x}\,\vec{v}$. Now two subcases arise. (A) For no proper prefix $\vec{w}_0$ of $\vec{v}$ we have $\vec{x}\,\vec{w}_0 \in L$. Then $\vec{v} \in V$, and we are done. (B) There is a proper prefix $\vec{w}_0$ of $\vec{v}$ with $\vec{x}\,\vec{w}_0 \in L$. Let $\vec{v} = \vec{w}_0\,\vec{v}_1$. Then, by Property (2), we have $\vec{x}\,\vec{v}_1 \in L$. (In (2), put $\vec{x}\,\vec{w}_0$ in place of $\vec{x}$ and in place of $\vec{y}$ put $\vec{x}$ and for $\vec{w}$ put $\vec{v}_1$.) $\vec{x}\,\vec{v}_1$ has smaller length than $\vec{x}\,\vec{v}$. Continue with $\vec{x}\,\vec{v}_1$ in the same way. At the end we get a partition of $\vec{v} = \vec{w}_0\,\vec{w}_1 \ldots \vec{w}_{n-1}$ such that $\vec{w}_i \in V$ for every $i < n$. Hence $L \subseteq U \cdot V^*$. We now show $U \cdot V^* \subseteq L$. Let $\vec{u} = \vec{x} \cdot \prod_{i<n} \vec{w}_i$. If $n = 0$, then $\vec{u} = \vec{x}$ and by definition of U we have $\vec{u} \in L$. Now let $n > 0$. With Property (2) we can show that $\vec{x} \cdot \prod_{i<n-1} \vec{w}_i \in S$. This shows that $\vec{u} \in L$. Finally, we have to show that U and V are deterministic. This follows for U from Theorem 2.3.13. Now let $\vec{x}, \vec{y} \in U$. Then by (2) $P := \{\vec{v} : \vec{x}\,\vec{v} \in L\} = \{\vec{v} : \vec{y}\,\vec{v} \in L\}$.

The reader may convince himself that P is deterministic. Now let V be the set of all $\vec{v}$ for which there is no prefix in $P - \{\varepsilon\}$. Then $P = V^*$ and because of Theorem 2.3.13 V is strict deterministic. This shows that (3) holds. Finally, assume that (3) holds. We have to show that L is an $LR(0)$–language. To this end, let $G_1 = \langle \mathtt{S}, N_1, A, R_1\rangle$ be a strict deterministic grammar which generates U and $G_2 = \langle \mathtt{S}_2, N_2, A, R_2\rangle$ a strict deterministic grammar which generates V. Then let $G_3 := \langle \mathtt{S}_3, N_1 \cup N_2 \cup \{\mathtt{S}_3, \mathtt{S}_4\}, A, R_3\rangle$ be defined as follows.

$$\begin{array}{rcl} R_3 := & & \{\mathtt{S}_3 \to \mathtt{S}_1, \mathtt{S}_3 \to \mathtt{S}_1\,\mathtt{S}_4, \mathtt{S}_4 \to \mathtt{S}_2, \mathtt{S}_4 \to \mathtt{S}_2\,\mathtt{S}_4\} \\ & \cup & R_1 \cup R_2 \end{array}$$

It is not hard to show that G_3 is an $LR(0)$–grammar and that $L(G_3) = L$. □

The decomposition in (3) is unique, if we exclude the possibility that $V = \varnothing$ and that $U = \{\varepsilon\}$ shall be the case only if $V = \{\varepsilon\}$. In this way we take care of the cases $L = \varnothing$ and $L = U$. The case $U = V$ may arise. Then $L = U^+$. The semi Dyck languages are of this kind.

Now we proceed to the proof of Theorem 2.4.22. Let L be deterministic. Then put $M := L \cdot \{\$\}$, where \$ is a new symbol. M is certainly deterministic; and it is prefix free and so strict deterministic. It follows that M is an $LR(0)$–language. Therefore there exists a strict deterministic grammar G which generates M. From the next theorem we now conclude that L is an $LR(1)$–language.

Lemma 2.4.27 *Let G be an $LR(0)$–grammar of the form $G = \langle \mathtt{S}, N \cup \{\$\}, A, R\rangle$ with $R \subseteq N \times ((N \cup A)^* \cup (N \cup A)^* \cdot \$)$ and $L(G) \subseteq A^*\$$, and assume that there is no derivation $\mathtt{S} \Rightarrow_R \mathtt{S}\$$ in G. Then let H be defined by $H := \langle \mathtt{S}, N, A, R_1 \cup R_2\rangle$, where*

$$\begin{array}{rcl} R_1 & := & \{A \to \vec{\alpha} : A \to \vec{\alpha} \in R, \vec{\alpha} \in (N \cup A)^*\}, \\ R_2 & := & \{A \to \vec{\alpha} : A \to \vec{\alpha}\,\$ \in R\}. \end{array}$$

Then H is an $LR(1)$–grammar and $L(H) \cdot \$ = L(G)$.

For a proof consider the following. We do not have $\mathtt{S} \Rightarrow^n_L \mathtt{S}$ in H. Further: if $\mathtt{S} \Rightarrow^+_L \vec{\alpha}$ in H then there exists a D such that $\mathtt{S} \Rightarrow^+_L \vec{\alpha}\, D$ in G, and if $\mathtt{S} \Rightarrow^+_L \vec{\beta}$ in G then we have $\vec{\beta} = \vec{\alpha}\, D$ and $\mathtt{S} \Rightarrow^+_L \vec{\alpha}$ in H. From this we can immediately conclude that H is an $LR(1)$–grammar.

Finally, let us return to the calculus of shifting and reducing. We generalize this strategy as follows. For every symbol α of our grammar we add a symbol $\underline{\alpha}$. This symbol is a formal inverse of α; it signals that at its place we look for an α but haven't identified it yet. This means that we admit the following transitions.

$$\frac{\vec{\eta}\underline{\alpha}\alpha \vdash \vec{x}}{\vec{\eta} \quad \vdash \vec{x}}$$

We call this rule **cancellation**. We write for strings $\vec{\alpha}$ also $\underline{\vec{\alpha}}$. This denotes the formal inverse of the entire string. If $\vec{\alpha} = \prod_{i<n} \alpha_i$ then $\underline{\alpha} = \prod_{i<n} \underline{\alpha_{n-i}}$. Notice that the order is reversed. For example $\underline{\mathtt{AB}} = \underline{\mathtt{B}} \cdot \underline{\mathtt{A}}$. These new strings allow to perform reductions on the left hand side even when only part of the right hand side of a production has been identified. The most general rule is this one.

$$\frac{\vec{\eta}\underline{X}\vec{\alpha} \vdash \vec{x}}{\vec{\eta}\underline{\vec{\delta}} \quad \vdash \vec{x}}$$

This rule is called the **LC–rule**. Here $X \to \vec{\alpha}\vec{\delta}$ must be a G–rule. This means intuitively speaking that if $\vec{\alpha}$ has already been found this string is an X if followed by $\vec{\delta}$. Since $\vec{\delta}$ is not yet there we have to write $\underline{\vec{\delta}}$. The **LC–calculus** consists of the rules shift, reduce and LC. Now the following holds.

Theorem 2.4.28 *Let G be a grammar. $\vec{\alpha} \vdash_G \vec{x}$ holds if and only if there is a derivation of $\varepsilon \vdash \varepsilon$ from $\underline{\vec{\alpha}} \vdash \vec{x}$ in the **LC**–calculus.*

A special case is $\vec{\alpha} = \varepsilon$. Here no part of the production has been identified, and one simply guesses a rule. If in place of the usual rules only this rule is taken, we get a strategy known as **top–down strategy**. In it, you may shift, reduce and guess a rule.

A grammar is called an $LL(k)$–grammar if it has a deterministic recognition algorithm using the top–down–strategy in which the next step depends on the first k symbols of $\vec{x}$. The case $k = 0$ is of little use (see the exercises).

This method is however too flexible to be really useful. However, the following is an interesting strategy. The right hand side of a production is divided into two parts, which are separated by a dot.

$$\begin{array}{lcl} \mathtt{S} & \to & \mathtt{A.SB} \mid \mathtt{c.} \\ \mathtt{A} & \to & \mathtt{a.} \\ \mathtt{B} & \to & \mathtt{b.} \end{array}$$

This dot fixes the part of the rule that must have been read when the corresponding LC–rule is triggered. A strategy of this form is called **generalized left corner strategy**. If the dot is at the right edge we get the bottom–up–strategy, if it is at the left edge we get the top–down–strategy.

Exercise 64. Show that a grammar G with $L(G) \neq \{\varepsilon\}$ is transparent only if it has no rules of the form $A \to \varepsilon$.

Exercise 65. Show Theorem 2.4.19.

Exercise 66. Prove Lemma 2.4.3. Show in addition: *If $\vec{x}$ is a term then the following set $P(\vec{x})$ is convex.*

$$P(\vec{x}) := \{\gamma(\vec{y}) : \vec{y} \text{ a prefix of } \vec{x}\}$$

Exercise 67. Show the following: *If L is deterministic then also $L/\{\vec{x}\}$ as well as $\{\vec{x}\}\backslash L$ are deterministic.* (See Section 1.2 for notation.)

Exercise 68. Show that a grammar is an $LL(0)$–grammar if it generates exactly one tree.

Exercise 69. Give an example of an NTS–language which is not an $LR(0)$–language.

Table 2.2: Generalized LC–Strategy

$\underline{\mathtt{S}}$	$\vdash \mathtt{aacbb}$
$\underline{\mathtt{S}}\,\mathtt{a}$	$\vdash \mathtt{acbb}$
$\underline{\mathtt{S}}\,\mathtt{A}$	$\vdash \mathtt{acbb}$
$\underline{\mathtt{B}}\,\underline{\mathtt{S}}$	$\vdash \mathtt{acbb}$
$\underline{\mathtt{B}}\,\underline{\mathtt{S}}\mathtt{a}$	$\vdash \mathtt{cbb}$
$\underline{\mathtt{B}}\,\underline{\mathtt{S}}\,\mathtt{A}$	$\vdash \mathtt{cbb}$
$\underline{\mathtt{B}}\,\underline{\mathtt{B}}\,\underline{\mathtt{S}}$	$\vdash \mathtt{cbb}$
$\underline{\mathtt{B}}\,\underline{\mathtt{B}}\,\underline{\mathtt{S}}\mathtt{c}$	$\vdash \mathtt{bb}$
$\underline{\mathtt{B}}\,\underline{\mathtt{B}}\,\underline{\mathtt{S}}\mathtt{S}$	$\vdash \mathtt{bb}$
$\underline{\mathtt{B}}\,\underline{\mathtt{B}}$	$\vdash \mathtt{bb}$
$\underline{\mathtt{B}}\,\underline{\mathtt{B}}\,\mathtt{b}$	$\vdash \mathtt{b}$
$\underline{\mathtt{B}}\,\underline{\mathtt{B}}\,\mathtt{B}$	$\vdash \mathtt{b}$
$\underline{\mathtt{B}}$	$\vdash \mathtt{b}$
$\underline{\mathtt{B}}\,\mathtt{b}$	$\vdash \varepsilon$
$\underline{\mathtt{B}}\,\mathtt{B}$	$\vdash \varepsilon$
ε	$\vdash \varepsilon$

2.5 Parikh's Theorem

In this section we shall prove a theorem which is due to Rohit Parikh (see (Parikh, 1961)). The proof consists of two parts. We shall first show that the trees which are generated by a context free grammar can also be generated by a tree adjunction grammar. Parikh's Theorem then follows quite directly from that latter result. First, however, we shall introduce the theorem itself.

Definition 2.5.1 *A* ***commutative monoid*** *or* ***commutative semigroup with unit*** *is a structure $\langle H, 0, +\rangle$ in which the following equations hold for every $x, y, z \in H$.*

$$\begin{array}{lcl} x + 0 & = & x \\ x + (y + z) & = & (x + y) + z \\ x + y & = & y + x \end{array}$$

We define the notation nx as follows. We have $0 \cdot x := 0$ and $(k + 1) \cdot x := k \cdot x + x$. We shall denote by $M(A)$ the monoid freely generated by A. By construction, $\mathfrak{M}(A) := \langle M(A), 0, +\rangle$ is a commutative semigroup with unit. What is more, $\mathfrak{M}(A)$ is freely generated by A as a commutative semigroup. We leave it to the reader to show this. We now look at the set ω^n of all n–long sequences of natural numbers, endowed with the operation $+$ defined by

$$\langle x_i : i < n\rangle + \langle y_i : i < n\rangle := \langle x_i + y_i : i < n\rangle \ .$$

This also forms a commutative semigroup with unit, which in this case is the sequence $\vec{0}$ consisting of n 0s. We denote this semigroup by Ω^n. For the following theorem we also need the so called **Kronecker symbol**.

$$\delta^i_j := \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 2.5.2 *Let $A = \{\mathsf{a}_i : i < n\}$. Let h be the map which assigns to each element a_i the sequence $\vec{e}_i = \langle \delta^i_j : j < n\rangle$. Then the* *homomorphism which extends h is already an isomorphism from $\mathfrak{M}(A)$ onto Ω^n.*

Proof. Let θ be the smallest congruence relation on $\mathrm{Tm}_\Omega(A)$ $(\Omega : 0 \mapsto 0, + \mapsto 2)$ which satisfies the following equations.

$$\begin{array}{ll} 0 + x & \theta\ x \\ x + (y + z) & \theta\ (x + y) + z \\ x + y & \theta\ y + x \end{array}$$

We shall show that for every $x \in T(A)$ there is a $y\ \theta\ x$ of the form

$$\sum_{i<n} k_i \cdot \mathsf{a}_i \ .$$

We show this by induction on x. If y has this form, then $h(x) = h(y) = \langle k_i : i < n\rangle$. This map is certainly surjective. It is also a homomorphism. For if $h(x) = \langle k_i : i < n\rangle$ and $h(y) = \langle \ell_i : i < n\rangle$ then we have

$$x + y\ \theta\ \sum_{i<n} k_i \cdot \mathsf{a}_i + \sum_{i<n} \ell_i \cdot \mathsf{a}_i\ \theta\ \sum_{i<n} (k_i + \ell_i) \cdot \mathsf{a}_i \ .$$

Therefore $h(x + y) = h(x) + h(y)$, as had to be shown. □

This theorem tells us that free commutative semigroups can be thought of as vectors of numbers. A general element of $M(A)$ can be written down as follows.

$$\sum_{i<n} k_i \cdot \mathsf{a}_i \ ,$$

where $k_i \in \omega$.

Now we shall define the map $\mu : A^* \to M(A)$ by

$$\begin{array}{lcl} \mu(1) & := & 0 \\ \mu(\mathsf{a}_i) & := & \mathsf{a}_i \\ \mu(\vec{x} \cdot \vec{y}) & := & \mu(\vec{x}) + \mu(\vec{y}) \end{array}$$

This map is a homomorphism of monoids and also surjective. It is not injective, except in the case where A consists of one element only. The map μ is called the **Parikh map**. We have

$$\mu\left(\prod_{i<k}\vec{x}_i\right) = \sum_{i<k}\mu(\vec{x}_i)\ .$$

Definition 2.5.3 *Two languages $L, M \subseteq A^*$ are called* ***letter equivalent*** *if $\mu[L] = \mu[M]$.*

Definition 2.5.4 *A set $U \subseteq M(A)$ is called* ***linear*** *if for some $\alpha \in \omega$ and some $u, v_i \in M(A)$*

$$U = \{u + \sum_{i<\alpha} k_i \cdot v_i : k_0, \ldots k_{\alpha-1} \in \omega\}\ .$$

The v_i are called ***cyclic vectors of*** *U. The smallest α for which U has such a representation is called the* ***dimension of*** *U. U is said to be* ***semilinear*** *if U is the finite union of linear sets. A language $L \subseteq A^*$ is called* ***semilinear*** *or a* ***Parikh language*** *if $\mu[S]$ is semilinear.*

We can denote semilinear sets rather compactly as follows. If U and V are subsets of $M(A)$ then write $U + V := \{x + y : x \in U, y \in V\}$. Further, let $x + U := \{x + y : y \in U\}$. So, elements are treated as singleton subsets. Also, we write $nU := \{nx : n \in \omega\}$. Finally, we denote by ωU the union of all nU, $n \in \omega$. With these definitions being made we write U as from the definition as follows.

$$U = u + \omega v_0 + \omega v_1 + \ldots + \omega v_{\alpha-1}\ .$$

This in turn we abbreviate by

$$U = u + \sum_{i<\alpha}\omega v_i$$

We draw a useful conclusion from the definitions.

Theorem 2.5.5 *Let A be a (possibly infinite) set. The set of semilinear languages over A form an AFL with the exception that the intersection of a semilinear language with a regular language need not be semilinear.*

Proof. Closure under union, star and concatenation are immediate. We have to show that semilinear languages are closed under homomorphisms and inverse homomorphisms. The latter is again trivial. Now let $v : A \to A^*$ be a homomorphism. v induces a map $\kappa_v : \mathfrak{M}(A) \to \mathfrak{M}(A)$. The image under κ_v of a semilinear set is semilinear. For given a string $\vec{x} \in A^*$ we have $\mu(\overline{v}(\vec{x})) = \kappa_v(\mu(\vec{x}))$, as is easily checked by induction on the length of $\vec{x}$. Let M be linear, say $M = u + \sum_{i<k} \omega \cdot v_i$. Then

$$\kappa_v[M] = \kappa_v(u) + \sum_{i<k} \omega \kappa_v(v_i) \ .$$

From this the claim follows. Hence we have $\mu[\overline{v}[L]] = \kappa_v[\mu[L]]$. The right hand side is semilinear as we have seen. Finally, take the language $L := \{\mathtt{a}^{2^i}\mathtt{b}^{2^i} : i \in \omega\} \cup \{\mathtt{b}^j\mathtt{a}^j : j \in \omega\}$. L is semilinear. $L \cap \mathtt{a}^*\mathtt{b}^* = \{\mathtt{a}^{2^i}\mathtt{b}^{2^i} : i \in \omega\}$ is not semilinear, however. ☐

From this it follows that if in a semilinear language we delete certain letters completely the result is a semilinear language again. Now we look first into semilinear languages over a single letter alphabet. Equivalently, we ask which subsets of ω are semilinear. A set $M \subseteq \omega$ is semilinear if there is an n_0 and a c such that for all $k > n_0$: $k \in M$ if and only if $k + c \in M$ if and only if $k - c \in M$. This means that up to finitely many exceptions M is the set of numbers of the form $a_0 + mc$ for some a_0 and c. This is a consequence of the Chinese Remainder Theorem (see for example (Jones, 1955) and (Ireland and Rosen, 1990)). For let $M = \{p_0 + \sum_{i<k} \lambda_i q_i : \lambda_i \in \omega\}$. Let c be the greatest common divisor of the q_i, $i < k$. By the Chinese Remainder Theorem c can be represented by

$$c = \sum_{i<k} \mu_i q_i,$$

with entire numbers μ_i. Since the μ_i need not be positive, we do the following. Let $\mu_0 < 0$. Denote by d the smallest common multiple of the q_i and assume that $d = q_0 r_0$ for some r_0. Then

$$c + d = (\mu_0 + r_0)q_0 + \sum_{0<i<k} \mu_i q_i \ .$$

If $(\mu_0 + r_0) < 0$, we repeat this step until the coefficient of q_0 is positive. This in turn is repeated for all $i < k$. Finally, we obtain a ξ such that

$$c + \xi d = \sum_{i<k} \nu_i q_i \ .$$

and for every $i \leq k$: $\nu_i \geq 0$. Likewise, we can represent all $2c+\xi' d$ as a positive sum for certain ξ'. If we choose i large enough then all numbers of the form $\gamma c + \xi d$ between ξd and $(\xi + 1)d$ have a positive representation. Since d has a positive representation, now every number of the form $p_0 + uc$ is in M, if only $uc \geq \xi d$. Since a number r which is not divisible by c is not representable as a sum of the q_i, we have $r \notin M$ if $r - p_0$ is not divisible by c. We call a set **arithmetical** if it has the form $p + \omega c$. This set is $= \{p\}$ if $c = 0$.

Theorem 2.5.6 *A subset of ω is semilinear if and only if it is the union of finitely many arithmetical sets.*

We can strengthen this as follows. We call U **almost periodical** if there is an n_0 and a p such that for all $n \in U$ such that $n \geq n_0$ we have $n + p \in U$. It is easily seen that a union of almost periodical sets is also almost periodical (again, this follows from the Chinese Remainder Theorem). Hence we have

Theorem 2.5.7 *A subset of ω is semilinear if and only if it is almost periodical.*

It follows that the complement of a semilinear subset of ω is again semilinear. We shall note without a proof that this also holds for arbitrary subsets of ω^n.

Theorem 2.5.8 *The complement of a semilinear subset of Ω^n is again semilinear. Hence, the intersection of two semilinear subsets of Ω^n is semilinear as well.*

Theorem 2.5.9 (Parikh) *A language is a Parikh language if and only if it is letter equivalent to a regular language.*

Proof. ($\Rightarrow$). Straightforward induction. ($\Leftarrow$). By induction on the length of the regular term R we shall show that $\mu[L(R)]$ is semilinear. This is clear for $R = \mathtt{a}_i$ or $R = \varepsilon$. It is also clear for $R = S_1 \cup S_2$. Now let $R = S_1 \cdot S_2$. If S_1, S_2 are linear, say

$$\mu[L(S_1)] = u + \sum_{i<\alpha} \omega v_i, \qquad \mu[L(S_2)] = u' + \sum_{i<\alpha'} \omega v'_i \,.$$

Then we have

$$\mu[L(R)] = (u + u') + \sum_{i<\alpha} \omega v_i + \sum_{j<\alpha'} \omega v'_j \,.$$

If S_1, S_2 are only semilinear then because of $(S \cup T) \cdot U = S \cdot U \cup T \cdot U$ and $U \cdot (S \cup T) = U \cdot S \cup U \cdot T$, R can be written as a union of products of linear sets, hence is semilinear. Now, finally, $R = S^*$. If $S = T \cup U$, then $R = (T^* \cdot U^*)^*$, so that we may restrict our attention again to the case that S is linear. Let $R = S^*$ with S semilinear, say

$$\mu[L(S)] = u + \sum_{i<\alpha} \omega v_i \,.$$

Then

$$\mu[L(R)] = \omega u + \sum_{i<\alpha} \omega v_i \,.$$

Hence R too is linear. This ends the proof. $\Box$

Now we shall turn to the already announced embedding of context free tree sets into tree sets generated by tree adjunction grammars. (The reader may wonder why we speak of sets and not of classes. In fact, we shall tacitly assume that trees are really tree domains, so that classes of finite trees are automatically sets.) Let $G = \langle S, N, A, R\rangle$ be a context free grammar. We want to define a tree adjunction grammar $\mathsf{Ad}_G = \langle \mathsf{C}_G, N, A, \mathsf{A}_G\rangle$ such that $L_B(G) = L_B(\mathsf{Ad}_G)$. We define C_G to be the set of all (ordered labelled) tree (domains) $\mathfrak{B}$ which can be generated by $L_B(G)$ and which are center trees and in which on no path not containing the root some nonterminal symbol occurs twice. Since there are only finitely many symbols, and the branching is finite, this set is actually finite. Now we define A_G. Let A_G contain all adjunction trees $\mathfrak{B}_X$, $X \in N$, such that (1) $\mathfrak{B}_X$ can be derived from X in γG, (2) no symbol occurs twice along a path that does contain the root. Also A_G is finite. Now it is not hard to show that $L_B(\mathsf{Ad}_G) \subseteq L_B(G)$. The reverse inclusion we shall show by induction on the number of nodes in the tree (domain). Let $\mathfrak{B}$ be in $L_B(G)$. Either there is a path not containing the root along which some symbol occurs twice, or there is not. In the second case the tree is in C_G. Hence $\mathfrak{B} \in L_B(\mathsf{Ad}_G)$ and we are done. In the first case we choose a $x \in B$ of minimal height such that there is a $y < x$ with identical label; let the label be X. Consider the subtree $\mathfrak{U}$ induced by the set $\downarrow x - \downarrow y \cup \{y\}$. We claim that $\mathfrak{U} \in \mathsf{A}_G$. For this we have to show the following. (a) $\mathfrak{U}$ is an adjunction tree, (b) $\mathfrak{U}$ can be deduced from X, (c) no symbol symbol occurs twice along a path which does not contain x. (a). A leaf of $\mathfrak{U}$ is either a leaf of $\mathfrak{B}$ or $= y$. In the first case the label is a terminal symbol in the second case it is identical to that of the root. (b). If $\mathfrak{B}$ is a tree of γG then $\mathfrak{U}$ can be derived from X. (c). Let π be a path which does not contain x and let $u, v \in \pi$ nodes with identical label and $u < v$. Then $v < x$, and this contradicts the minimality of x. Hence all three conditions are met. So we can disembed $\mathfrak{U}$. This means that there is a tree $\mathfrak{B}'$ such that $\mathfrak{B}$ is derived from $\mathfrak{B}'$ by adjoining $\mathfrak{U}$. We have $\mathfrak{B}' \in L_B(G)$ and by induction hypothesis $\mathfrak{B}' \in L_B(\mathsf{Ad}_G)$. Hence $\mathfrak{B} \in L_B(\mathsf{Ad}_G)$, which had to be shown.

Theorem 2.5.10 (Joshi & Levy & Takahashi) *Every set of labelled ordered tree domains generated by a context free grammar is also one generated by an unregulated tree adjunction grammar.* $\Box$

Now we shall prove Parikh's Theorem for unregulated tree adjunction grammars. To this end we define for a labelled tree and a letter α $\sigma_\alpha(\mathfrak{B})$ to be the number of nodes *excluding the root* whose label carry α. Now let $\langle \mathsf{C}, N, A, \mathsf{A}\rangle$ be a tree adjunction grammar and $\mathsf{C} = \{\mathfrak{C}_i : i < \alpha\}$, $\mathsf{A} = \{\mathfrak{A}_j : j < \beta\}$.

Lemma 2.5.11 *Let $\mathfrak{B}'$ result from $\mathfrak{B}$ by adjoining the tree $\mathfrak{A}$. Then $\sigma_\alpha(\mathfrak{B}') = \sigma_\alpha(\mathfrak{B}) + \sigma_\alpha(\mathfrak{A})$.*

The proof of this lemma is easy. From this it follows that we only need to know for an arbitrarily derived tree how many times which tree has been adjoined and what the starting tree was. So let $\mathfrak{B}$ be a tree which resulted from $\mathfrak{C}_i$ by adjoining $\mathfrak{A}_j$ p_j times, $j < \beta$. Then

$$\sigma_\alpha(\mathfrak{B}) = \sigma_\alpha(\mathfrak{C}_i) + \sum_{i<\beta} p_j \cdot \sigma_\alpha(\mathfrak{A}_j) \ .$$

Let now $\mu(\mathfrak{B}) := \sum_{a\in A} \sigma_a(\mathfrak{B}) \cdot a$. Then we have

$$\mu(\mathfrak{B}) = \mu(\mathfrak{C}_i) + \sum_{i<\beta} p_j \cdot \mu(\mathfrak{B}_j) \ .$$

We define the following sets

$$\Sigma_i := \{\mu(\mathfrak{C}_i) + \sum_{j<\beta} p_j \cdot \mu(\mathfrak{A}_j) : p_j \in \omega\} \ .$$

Then $\mu[L_B(\langle \mathsf{C}, \mathsf{A}\rangle)] \subseteq \bigcup_{i<n} \Sigma_i$. However, equality need not always hold. We have to notice the following problem. A tree $\mathfrak{A}_j$ can be adjoined to a tree $\mathfrak{B}$ only if its root label actually occurs in the tree $\mathfrak{B}$. Hence not all values of $\bigcup \Sigma_i$ are among the values under μ of a derived tree. However, if a tree can be adjoined *once* it can be adjoined any number of times and to all trees that result from this tree by adjunction. Hence we modify our starting set of trees somewhat. We consider the set D of all pairs $\langle k, W\rangle$ such that $k < \alpha$, $W \subseteq \beta$ and there is a derivation of a tree that starts with $\mathfrak{C}_k$ and uses exactly the trees from W. For $\langle k, W\rangle \in D$

$$L(k, W) = \{\mu(\mathfrak{C}_i) + \sum_{j\in W} k_j \cdot \mu(\mathfrak{A}_j) : k_j \in \omega\} \ .$$

Then $L := \bigcup\langle L(k,W) : \langle k,W\rangle \in D\rangle$ is semilinear. At the same time this is the set of all $\mu(\mathfrak{B})$ where $\mathfrak{B}$ is derivable from $\langle \mathsf{C}, N, A, \mathsf{A}\rangle$. Now we have the

Theorem 2.5.12 *Let L be the language of an unregulated tree adjunction grammar then L is semilinear.* □

Corollary 2.5.13 (Parikh) *Let L be a context free language. Then L is semilinear.* □

This theorem is remarkable is many respects. We shall meet it again several times. Semilinear sets are closed under complement, by Theorem 2.5.8 and hence also under intersection. We shall show, however, that this does not hold for semilinear languages.

Proposition 2.5.14 *There are context free language L_1 and L_2 such that $L_1 \cap L_2$ is not semilinear.*

Proof. Let $M_1 := \{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$ and $M_2 := \{\mathtt{b}^n\mathtt{a}^{2n} : n \in \omega\}$. Now put

$$\begin{array}{rcl} L_1 & := & \mathtt{b}M_1^*\mathtt{a}^*, \\ L_2 & := & M_2^+. \end{array}$$

Because of Theorem 1.5.11 L_1 and L_2 are context free. Now look at $L_1 \cap L_2$. It is easy to see that the intersection consists of the following strings.

$$\mathtt{ba}^2, \quad \mathtt{ba}^2\mathtt{b}^2\mathtt{a}^4, \quad \mathtt{ba}^2\mathtt{b}^2\mathtt{a}^4\mathtt{b}^4\mathtt{a}^8, \quad \mathtt{ba}^2\mathtt{b}^2\mathtt{a}^4\mathtt{b}^4\mathtt{a}^8\mathtt{b}^8\mathtt{a}^{16}, \ldots$$

The Parikh image is $\{(2^{n+1}-2)\mathtt{a} + (2^n-1)\mathtt{b} : n \in \omega\}$. This set is not semilinear, since the result of deleting the symbol $\mathtt{b}$ (that is, the result of applying the projection onto $\mathtt{a}^*$) is not almost periodical. □

We know that for every semilinear set $N \subseteq M(A)$ there is a regular grammar G such that $\mu[L(G)] = N$. However G can be relatively complex. Now the question arises whether the complete preimage $\mu^{-1}[N]$ under μ is at least regular or context free. This is not the case. However, we do have the following.

Theorem 2.5.15 *The full preimage of a semilinear set over a single letter alphabet is regular.*

This is best possible result. The theorem becomes false as soon as we have two letters.

Theorem 2.5.16 *The full preimage of* $\omega(\mathtt{a}+\mathtt{b})$ *is not regular; it is however context free. The full preimage of* $\omega(\mathtt{a}+\mathtt{b}+\mathtt{c})$ *is not context free.*

Proof. We show the second claim first. Let

$$W := \mu^{-1}[\omega(\mathtt{a}+\mathtt{b}+\mathtt{c})] .$$

Assume that W is context free. Then the intersection with the regular language $\mathtt{a}^*\mathtt{b}^*\mathtt{c}^*$ is again context free. This is precisely the set $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$. Contradiction. Now for the first claim. Denote by $b(\vec{x})$ the number of occurrences of $\mathtt{a}$ in $\vec{x}$ minus the number of occurrences of $\mathtt{b}$ in $\vec{x}$. Then $V := \{\vec{x} : b(\vec{x}) = 0\}$ is the full preimage of $\omega(\mathtt{a}+\mathtt{b})$. V is not regular; otherwise the intersection with $\mathtt{a}^*\mathtt{b}^*$ is also regular. However, this is $\{\mathtt{a}^n\mathtt{b}^n : n \in \omega\}$. Contradiction. V is context free. To show this we shall construct a context free grammar G which generates V. We have three nonterminals, $\mathtt{S}$, $\mathtt{A}$, and $\mathtt{B}$. The rules are

$$\begin{array}{lcll} \mathtt{S} & \to & \mathtt{SS} \mid \mathtt{AB} \mid \mathtt{BA} \mid x, & \quad x \in A - \{\mathtt{a},\mathtt{b}\}, \\ \mathtt{A} & \to & \mathtt{AS} \mid \mathtt{SA} \mid \mathtt{a}, & \\ \mathtt{B} & \to & \mathtt{BS} \mid \mathtt{SB} \mid \mathtt{b}. & \end{array}$$

The start symbol is $\mathtt{S}$. We claim: $\mathtt{S} \vdash_G \vec{x}$ if and only if $b(\vec{x}) = 0$, $\mathtt{A} \vdash_G \vec{x}$ if and only if $b(\vec{x}) = 1$ and $\mathtt{B} \vdash_G \vec{x}$ if and only if $b(\vec{x}) = -1$. The directions from left to right are easy to verify. It therefore follows that $V \subseteq L(G)$. The other directions we show by induction on the length of $\vec{x}$. It suffices to show the following claim.

> If $b(\vec{x}) \in \{1, 0, -1\}$ there are $\vec{y}$ and $\vec{z}$ such that $|\vec{y}|, |\vec{z}| < |\vec{x}|$ and such that $\vec{x} = \vec{y}\vec{z}$ as well as $|b(\vec{y})|, |b(\vec{z})| \leq 1$.

Hence let $\vec{x} = \prod_{i<n} x_i$ be given. Define $k(\vec{x}, j) := b({}^j\vec{x})$, and $K := \{k(\vec{x}, j) : j < n+1\}$. As is easily seen, this set is an interval $[m, m']$ with $m \leq 0$. Further, $k(\vec{x}, n) = b(\vec{x})$. (a) Let $b(\vec{x}) = 0$. Then put $\vec{y} := x_0$ and $\vec{z} := \prod_{0<i<n} x_i$. This satisfies the conditions. (b) Let $b(\vec{x}) = 1$. Case 1: $x_0 = \mathsf{a}$. Then put again $\vec{y} := x_0$ and $\vec{z} := \prod_{0<i<n} x_i$. Case 2: $x_0 = \mathsf{b}$. Then $k(\vec{x}, 1) = -1$ and there is a j such that $k(\vec{x}, j) = 0$. Put $\vec{y} := \prod_{i<j} x_i$, $\vec{z} := \prod_{j \leq i < n} x_i$. Since $0 < j < n$, we have $|\vec{y}|, |\vec{z}| < |\vec{x}|$. Furthermore, $b(\vec{y}) = 0$ and $b(\vec{z}) = 1$. (c) $b(\vec{x}) = -1$. Similar as (b). □

Exercise 70. Let $|A| = 1$. Show that $\mathfrak{Z}(A)$ is isomorphic to $\mathfrak{M}(A)$.

Exercise 71. Let $|A| = 1$ and Ad be an unregulated tree adjunction grammar. Show that the language generated by Ad over A^* is regular.

Exercise 72. Let $A = \{\mathsf{a}, \mathsf{b}\}$. Further, let $U := \mathsf{a}^* \cup \mathsf{b}^*$. Now let $N \subseteq M(A)$ be a set such that $N - U$ is infinite. Show that there are $2^{\aleph_0}$ many languages S with $\mu[S] = N$. *Hint.* The cardinality of A^* is $\aleph_0$, hence there can be no more than $2^{\aleph_0}$ such languages.

Exercise 73. Prove Theorem 2.5.15. *Hint.* Restrict your attention first to the case that $A = \{\mathsf{a}\}$.

Exercise 74. Let $N \subseteq M(A)$ be semilinear. Show that the full preimage is of Type 1 (that is, context sensitive). *Hint.* It is enough to show this for linear sets.

Exercise 75. In this exercise, we sketch an alternative proof of Parikh's Theorem. Let $A = \{\mathsf{a}_i : i < n\}$ be an alphabet. In analogy to the regular terms we define semilinear terms. (a) a_i, $i < n$, is a semilinear term, with interpretation $\{\vec{e}_i\}$. (b) If A and B are semilinear terms, so is $A \oplus B$ with interpretation $\{\vec{u} + \vec{v} : \vec{u} \in A, \vec{v} \in B\}$, $A \cup B$, with interpretation $\{\vec{u} : \vec{u} \in A \text{ or } \vec{u} \in B\}$ and ωA with interpretation $\{k\vec{u} : k \in \omega, \vec{u} \in A\}$. The first step is to translate a context free grammar into a set of equations of the form $X_i = C_i(X_0, X_1, \ldots, X_{q-1})$, q the number of nonterminals,

C_i semilinear terms. This is done as follows. Without loss of generality we can assume that in a rule $X \to \vec{\alpha}$, $\vec{\alpha}$ contains a given variable at most once. Now, for each nonterminal X let $X \to \vec{\alpha}_i$, $i < p$, be all the rules of G. Corresponding to these rules there is an obvious equation of the form

$$X = A \cup (B \oplus X) \text{ or } X = A$$

where A and B are semilinear terms that do not contain X. The second step is to prove the following lemma:

> *Let $X = A \cup (B \oplus X) \cup (C \oplus \omega X)$, with A, B and C semilinear terms not containing X. Then the least solution of that equation is $A \cup \omega B \cup \omega C$. If $B \oplus X$ is missing from the equation, the solution is $A \cup \omega C$, and if $C \oplus \omega X$ is missing the solution is $A \cup \omega B$.*

Using this lemma it can be shown that the system of equations induced by G can be solved by constant semilinear terms for each variable.

2.6 Are Natural Languages Context Free?

We shall finish our discussion of context free languages by looking at some naturally arising languages. We shall give examples of languages and constructions which are definitely not context free. The complexity of natural languages has been high on the agenda ever since the introduction of this hierarchy by Noam Chomsky. His intention was to discredit structuralism which he identified as a theory that postulated that natural languages always are context free. By contrast, he claimed that natural languages are not context free and gave many examples. It is still widely believed that Chomsky had won his case. (For an illuminating discussion read (Manaster-Ramer and Kac, 1990)).

It has emerged over the years that the arguments given by Noam Chomsky and Paul Postal against the context freeness of natural languages were faulty. Gerald Gazdar, Geoffrey Pullum and others have repeatedly found holes in the argumentation. This has finally led to the claim that to the contrary natural languages all are context free syntactically (see (Gazdar *et al.*, 1985)). The first to deliver a correct proof of the contrary was Riny Huybregts, only shortly later followed by Stuart Shieber. (See (Huybregts, 1984) and (Shieber, 1985).) Evidence from Bambara was given by (Culy, 1987). Of course, it was hardly doubted that from structural point of view natural languages are not context free (see the analyses of Dutch and German within GB, for example, or (Bresnan *et al.*, 1987)), but it was not shown decisively that they are not even weakly context free.

How can one give such a proof non context freeness of a language L? To this end, one takes a regular language R and intersects it with L. If L is context free, so is $L \cap R$. Now choose a homomorphism h and map the language $L \cap R$ onto a known non context free language. We give an example from the paper by Stuart Shieber. Look at (2.6.1) – (2.6.3) in Table 2.3. If one looks at the nested infinitives in Swiss German (first rows) we find that they are structured differently from English (last rows) and High German (middle rows). By asking who does what to whom (we let, the children help, Hans paints) we see that the constituents are quite different in the three languages. Subject and corresponding verb are together in English, in High German they are on either side of the embedded infinitive (this is called **nesting order**). In Swiss German however it is still different. The verbs follow each other in the reverse order as in German (so, they occur in the order of the subjects). This is called **crossing order**. Now we assume — this is an empirical assumption, to be sure — that this is the general pattern. It shall be emphasized that the processing of such sentences becomes difficult with four or five infinitives. Nevertheless, the resulting sentences are considered grammatical.

Now we proceed as follows. The verbs require accusative or dative on their complements. The following examples show that there is a difference between dative and accusative. In (2.6.4) `de`

Table 2.3: Infinitives in Germanic Languages

(2.6.1) `Jan säit, das Hans es huus aastricht.`
`Jan sagt, daß Hans das Haus anstreicht.`
`Jan says that Hans is painting the house.`

(2.6.2) `Jan säit, das mer em Hans es huus hälfed aastriche.`
`Jan sagt, daß wir Hans das Haus anstreichen helfen.`
`Jan says that we help Hans paint the house.`

(2.6.3) `Jan säit, das mer d'chind em Hans es huus lönd hälfe aastriche.`
`Jan sagt, daß wir die Kinder Hans das Haus anstreichen helfen lassen.`
`Jan says that we let the children help Hans paint the house.`

(2.6.4) `*Jan säit, das mer de Hans es huus hälfed aastriche.`

(2.6.5) `*Jan säit, das mer em chind em Hans es huus lönd hälfe aastriche.`

Hans is accusative and the complement of aastriche, which selects dative. The resulting sentence is ungrammatical. In (2.6.5), em chind is dative, while lönd selects accusative. Again the sentence is ungrammatical. We now define the following regular language.

$$\begin{aligned} R := \; & \texttt{Jan säit, das} \cdot (\texttt{em} \cup \texttt{d'} \cup \texttt{de} \cup \texttt{mer} \cup \texttt{Hans} \cup \texttt{es} \cup \texttt{huus})^* \\ & \cdot (\texttt{laa} \cup \texttt{lönd} \cup \texttt{hälfe}) \cdot \texttt{aastriche} \end{aligned}$$

Subsequently we define the following homomorphism v. v sends d', de, laa and lönd to a, em and hälfe to d, everything else is mapped to ε. The claim is that

$$h[S \cap R] = \{\vec{x}\,\vec{x} : \vec{x} \in \texttt{a} \cdot (\texttt{a} \cup \texttt{d})^*\}$$

To this end we remark that a verb is sent to d if it has a dative object and to a if it has an accusative object. An accusative object is of the form de N or d' N (N a noun) and is mapped to a by $\overline{v}$. A dative object has the form em N, N a noun, and is mapped onto d. Since the nouns are in the same order as the associated infinitives we get the desired result.

In mathematics we find a phenomenon similar to Swiss German. Consider the integral of a function. If $f(x)$ is a function, the integral of $f(x)$ in the interval $[a, b]$ is denoted by

$$\int_a^b f(x)dx$$

This is not in all cases well formed. For example, $\int_0^1 x^{-1}dx$ is ill formed, since there Riemann approximation leads to a sequence which is not bounded, hence has no limit. Similarly, $\lim_{n\rightarrow\infty}(-1)^n$ does not exist. Notice that the value range of x is written at the integral sign without saying with what variable the range is associated. For example, let us look at

$$\int_a^b \int_c^d f(x, y)dxdy$$

The rectangle over which we integrate the function is $a \leq x \leq b$ and $c \leq y \leq d$. Hence, the first integral sign corresponds to the operator dx, which occurs first in the list. Likewise for three integrals:

$$\int_{a_0}^{b_0} \int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x_0, x_1, x_2) dx_0 dx_1 dx_2$$

where the value range is $a_i \leq x_i \leq b_i$ for all $i < 3$. Consider the following functions:

$$f(x_0, \ldots, x_n) := \prod_{i<n} x_i^{\alpha_i}$$

with $\alpha_i \in \{-1, 1\}$, $i < n$. Further, we allow for the interval $[a_i, b_i]$ either $[0, 1]$ or $[1, 2]$. Then an integral expression

$$\int_{a_0}^{b_0} \int_{a_1}^{b_1} \cdots \int_{a_{n-1}}^{b_{n-1}} f(x_0, x_1, \ldots, x_{n-1}) dx_0 dx_1 \ldots dx_{n-1}$$

is well formed if and only if $a_i > 0$ for all $i < n$ such that $\alpha_i = -1$. The dependencies are crossing, and the order of elements is exactly as in Swiss German (considering the boundaries and the variables). The complication is the mediating function, which determines which of the boundary elements must be strictly positive.

The next example in our series is modelled after the proof of the non context freeness of `Algol`. It deals with a quite well known language, namely predicate logic. Predicate logic is defined as a language over a set of relation and function symbols of varying arity and a set of variables $\{x_i : i \in \omega\}$. In order to be able to conceive of predicate logic as a language in our sense, we code the variables as consisting of sequences $\mathtt{x}\vec{\alpha}$, where $\vec{\alpha} \in \{\mathtt{0}, \mathtt{1}\}^+$. We have $\mathtt{x}\vec{\alpha} = \mathtt{x}\vec{\beta}$ if and only if $\vec{\alpha} = \vec{\beta}$. (Leading zeros are not suppressed.) We restrict ourselves to the language of pure equality. The alphabet is $\{\forall, \exists, \mathtt{(}, \mathtt{)}, \doteq, \mathtt{x}, \mathtt{0}, \mathtt{1}, \wedge, \neg, \rightarrow\}$. The grammar rules are as follows.

$$\begin{array}{lcl}
\mathtt{F} & \to & \mathtt{QF} \mid \neg\mathtt{(F)} \mid \mathtt{(F} \wedge \mathtt{F)} \mid \mathtt{(F} \to \mathtt{F)} \mid \mathtt{P} \\
\mathtt{P} & \to & \mathtt{V} \doteq \mathtt{V} \\
\mathtt{Q} & \to & \forall\mathtt{VF} \mid \exists\mathtt{VF} \\
\mathtt{V} & \to & \mathtt{xZ} \\
\mathtt{Z} & \to & \mathtt{0Z} \mid \mathtt{1Z} \mid \mathtt{0} \mid \mathtt{1}
\end{array}$$

Here $\mathtt{F}$ stands for the set of formulae $\mathtt{P}$ for the set of prime formulae $\mathtt{Q}$ for the set of quantifier prefixes, $\mathtt{V}$ the set of variables and $\mathtt{E}$ for the set of strings over $\mathtt{0}$ and $\mathtt{1}$. Let $\vec{x}$ be a formula and C an occurrence of a variable $\mathtt{x}\vec{\alpha}$. We now say that this occurrence of a variable is **bound** in $\vec{x}$ if it is an occurrence D of a formula $Q\mathtt{x}\vec{\alpha}\vec{y}$ in $\vec{x}$ with $Q \in \{\forall, \exists\}$ which contains C. A formula is called a **sentence** if every occurrence of a variable is bound.

Theorem 2.6.1 *The set of sentences of predicate logic of pure equality is not context free.*

Proof. Let L be the set of sentences of pure equality of predicate logic. Assume this set is context free. Then by the Pumping Lemma there is a k such that every string of length $\geq k$ has a decomposition $\vec{u}\vec{x}\vec{v}\vec{y}\vec{w}$ such that $\vec{u}\vec{x}^i\vec{v}\vec{y}^i\vec{z} \in L$ for all i and $|\vec{x}\vec{v}\vec{y}| \leq k$. Define the following formulae.

$$\forall \mathtt{x}\vec{\alpha}\mathtt{x}\vec{\alpha} \doteq \mathtt{x}\vec{\alpha}\ .$$

All these formulae are sentences. If $\vec{\alpha}$ is sufficiently long (for example, longer than k) then there is a decomposition as given. Since $\vec{x}\vec{v}\vec{y}$ must have length $\leq k$ $\vec{x}$ and $\vec{y}$ cannot both be disjoint to all occurrences of $\vec{\alpha}$. On the other hand, it follows from this that $\vec{x}$ and $\vec{y}$ consist only of $\mathtt{0}$ and $\mathtt{1}$, and so necessarily they are disjoint to some occurrence of $\vec{\alpha}$. If one pumps up $\vec{x}$ and $\vec{y}$, necessarily one occurrence of a variable will end up being unbound. $\Box$

We can strengthen this result considerably.

Theorem 2.6.2 *The set of sentence of predicate logic of pure equality is not semilinear.*

Proof. Let P be the set of sentences of predicate logic of pure equality. Assume that P is semilinear. Then let P_1 be the set of sentences which contain only one occurrence of a quantifier, and let this quantifier by $\exists$. $\mu[P_1]$ is the intersection of $\mu[P]$ with the set of all vectors whose $\exists$–component is 1 and whose $\forall$–component is 0. This is then also semilinear. Now we consider the image of $\mu[P_1]$ under deletion of all symbols which are different from `x`, `0` and `1`. The result is denoted by Q_1. Q_1 is semilinear. By construction of P_1 there is an $\vec{\alpha} \in \{\mathtt{0}, \mathtt{1}\}^*$ such that every occurrence of a variable is of the form $\mathtt{x}\vec{\alpha}$. If this variable occurs k times and if $\vec{\alpha}$ contains p occurrences of `0` and q occurrences of `1` we get as a result the vector $k\mathtt{x} + kp\mathtt{0} + kq\mathtt{1}$. It is easy to see that k must be odd. For a variable occurs once in the quantifier and elswehere once to the left and once to the right of the equation sign. Now we have among others the following sentences.

$$\begin{array}{l}
\exists\mathtt{x}\vec{\alpha}\mathtt{x}\vec{\alpha} \doteq \mathtt{x}\vec{\alpha} \\
\exists\mathtt{x}\vec{\alpha}\mathtt{(x}\vec{\alpha} \doteq \mathtt{x}\vec{\alpha} \wedge \mathtt{x}\vec{\alpha} \doteq \mathtt{x}\vec{\alpha}\mathtt{)} \\
\exists\mathtt{x}\vec{\alpha}\mathtt{(x}\vec{\alpha} \doteq \mathtt{x}\vec{\alpha} \wedge \mathtt{(x}\vec{\alpha} \doteq \mathtt{x}\vec{\alpha} \wedge \mathtt{x}\vec{\alpha} \doteq \mathtt{x}\vec{\alpha}\mathtt{))}
\end{array}$$

Since we may choose any sequence $\vec{\alpha}$ we have

$$Q_1 = \{(2k+3)(\mathtt{x} + p\mathtt{0} + q\mathtt{1}) : k, p, q \in \omega\} .$$

Q_1 is an infinite union of planes of the form $(2k+3)(\mathtt{x}+\omega\mathtt{0}+\omega\mathtt{1})$. We show: no finite union of linear planes equals Q_1. From this we automatically get a contradiction. So, assume that Q_1 is the union of U_i, $i < n$, U_i linear. Then there exists a U_i which contains infinitely many vectors of the form $(2k+3)\mathtt{x}$. From this one easily deduces that U_i contains a cyclic vector of the form $m\mathtt{x}$, $m > 0$. (This is left as an exercise.) However, it is clear that if $v \in Q_1$ then we have $m\mathtt{x} + v \notin Q_1$, and then we have a contradiction. □

Now we shall present an easy example of a 'natural' language which is not semilinear. It has been proposed in somewhat different form by Arnold Zwicky. Consider the number names of English. The stock of primitive names for numbers is finite. It contains the names for digits (`zero` up to `nine`) the names for the multiples of ten (`ten` until `ninety`), the numbers from `eleven` and `twelve` until `nineteen` as well as some names for the powers of ten: `hundred`, `thousand`, `million`, `billion`, and a few more. (Actually, using Latin numerals we can go on to very high powers, but few people master these numerals, so they will hardly know more than these.) Assume without loos of generality that `million` is the largest of them. Then there is an additional recipe for naming higher powers, namely by stacking the word `million`. The number 10^{6k} is represented by the k–fold iteration of the word `million`. For example, the words `one million million million` names the number 10^{18}. (It is also called `quintillion` using Latin numerals.) For arbitrary numbers the schema is as follows. A number in digital expansion is divided from right to left into blocks of six. So, it is divied into

$$\alpha_0 + \alpha_1 \times 10^6 + \alpha_2 \times 10^{12} \ldots$$

Here, $\alpha_i < 10^6$. The associated number name is then as follows.

$$\ldots \vec{\eta}_2 \ \texttt{million million} \ \vec{\eta}_1 \ \texttt{million} \ \vec{\eta}_0$$

where $\vec{\eta}_i$ is the string representing α_i. If $\alpha_i = 0$ the ith block is omitted. Let Z be the set of number names. We define the mapping φ as follows. $\varphi(\texttt{million}) = \texttt{b}$; all other primitive names are mapped onto `a`. The Parikh image $\varphi[Z]$ of Z is denoted by W. Now we have

$$W = \{k_0\texttt{a} + k_1\texttt{b} : k_1 \geq \binom{\ulcorner k_0/9 \urcorner}{2}\} .$$

Here, $\ulcorner k \urcorner$ is the largest integer $\leq k$. We have left the proof of this fact to the reader. We shall show that W is not semilinear. This shows that Z is also not semilinear. Suppose that W is semilinear, say $W = \bigcup_{i<n} N_i$ where all the N_i are linear. Let

$$N_i = u_i + \sum_{j<p_i} \omega v^i_j$$

for certain u_i and $v_j^i = \lambda_j^i \mathtt{a} + \mu_j^i \mathtt{b}$. Suppose further that for some i and j we have $\lambda_j^i \neq 0$. Consider the set

$$P := u_i + \omega v_j^i = \{u_i + k\lambda_j^i \mathtt{a} + k\mu_j^i \mathtt{b} : k \in \omega\} .$$

Certainly we have $P \subseteq N_i \subseteq W$. Furthermore, we surely have $\mu_j^i \neq 0$. Now put $\zeta := \lambda_j^i / \mu_j^i$. Then

$$P = \{u_i + k\mu_j^i(\mathtt{a} + \zeta\mathtt{b}) : k \in \omega\} .$$

Lemma 2.6.3 *For every $\varepsilon > 0$ almost all elements of P are of the form $p\mathtt{a} + q\mathtt{b}$ where $q/p \leq \zeta + \varepsilon$.*

Proof. Let $u_i = x\mathtt{a} + y\mathtt{b}$. Then a general element is of the form $(x + k\lambda_j^i)\mathtt{a} + (y + k\mu_j^i)\mathtt{b}$. We have to show that for almost all k the inequality

$$\frac{x + k\lambda_j^i}{y + k\mu_j^i} \leq \varepsilon + \zeta$$

is satisfied. Indeed, if $k > \frac{x}{\mu_j^i \varepsilon}$, then

$$\frac{x + k\lambda_j^i}{y + k\mu_i^j} \leq \frac{x + k\lambda_j^i}{k\mu_j^i} = \zeta + \frac{x}{k\mu_j^i} < \zeta + \frac{x}{\mu_j^i x / \mu_j^i \varepsilon} = \zeta + \varepsilon$$

This holds for almost all k. ☐

Lemma 2.6.4 *Almost all points of P are outside of W.*

Proof. Let n_0 be chosen in such a way that $\binom{\lceil n_0/9 \rceil}{2} > n_0(\zeta + 1)$. Then for all $n \geq n_0$ we also have $\binom{\lceil n/9 \rceil}{2} > n(\zeta + 1)$. Let $p\mathtt{a} + q\mathtt{b} \in W$ with $p \geq n_0$. Then we have $\frac{q}{p} > \zeta + \varepsilon$, and therefore $p\mathtt{a} + q\mathtt{b} \notin P$. Hence P and W are disjoint on the set $H := \{p\mathtt{a} + q\mathtt{b} : p \geq n_0\}$. However $P \cap -H$ is certainly finite. ☐

Now have the desired contradiction. For on the one hand no vector is a multiple of $\mathtt{a}$; on the other hand there can be no vector $m\mathtt{a} + n\mathtt{b}$ with $n \neq 0$. Hence W is not semilinear.

Notes on this section. The question on the complexity of variable binding is discussed in (Marsh and Partee, 1987). It is shown there that the language of sentences of predicate logic is not context free (a result that was 'folklore') but that it is at least an indexed language. (Indexed languages neeed not be semilinear.) On the other hand, it is conjectured that if we take V to the set of formulae in which every quantifier binds at least one free occurrence of a variable, the language V is not even an indexed language. See also Section 5.6.

Exercise 76. Formalize the language of functions and integral expressions. Prove that the language of proper integral expressions is not context free.

Exercise 77. Show the following: *Let U be a linear set which contains infinitely many vectors of the form $k\mathtt{a}$. Then there exists a cyclic vector of the form $m\mathtt{a}$, $m > 0$.* *Hint.* Notice that the alphabet may consist of more than one letter.

Exercise 78. Show that W has the claimed form.

Exercise 79. Show that the set V is not semilinear.

$$V := \left\{ k_0\mathtt{a} + k_1\mathtt{b} : k_1 \leq \binom{k_0}{2} \right\}$$

Hint. Evidently, no linear set $\subseteq V$ may contain a vector $k\mathtt{b}$. Therefore the following is well defined.

$$\gamma := max \left\{ \frac{\mu^i_j}{\lambda^i_j} : i < n, j < p_i \right\} .$$

Show now that for every $\varepsilon > 0$ almost all elements of W are of the form $x\mathtt{a} + y\mathtt{b}$ where $y \leq (\gamma + \varepsilon)x$. If we put for example $\varepsilon = 1$ we now get a contradiction.

Exercise 80. Prove the unique readability of predicate logic. *Hint.* Since we have strictly speaking not defined terms, restrict yourself to proving that the grammar given above is not ambiguous (not even opaque).

Chapter 3

Categorial Grammar and Formal Semantics

3.1 Languages as Systems of Signs

Languages are certainly not sets of strings. They are systems for communication. This means in particular that the strings have meaning, a meaning which all speakers of the language understand (almost by definition). And since natural languages have potentially infinitely many strings, there must be a way to find out what meaning a given string has on the basis of finite information. An important principle in connection with this is the so called *Principle of Compositionality*. It says in simple words that the meaning of a string only depends on its analysis. More precisely: if $\rho = X \to \alpha_0\alpha_1 \ldots \alpha_{n-1}$ is a rule and $\vec{u}_i$ a string of category α_i then $\vec{v} := \vec{u}_0\vec{u}_1 \ldots \vec{u}_{n-1}$ is a string of category X and the meaning of $\vec{v}$ depends only on the meaning of the $\vec{u}_i$ and ρ. In this form the principle of compositionality is still rather vague, and we shall refine and precisify it in the course of this section. However, for now we shall remain with this definition. It appears that we have admitted only context free rules. This is a restriction, as we know. We shall see later how we can get rid of it.

To begin, we shall assume that meanings come from some set

M, which shall not be specified further. As before, exponents are members of A^*, where A is a finite alphabet.

Definition 3.1.1 *An* ***interpreted (string) language over the alphabet*** A *and* ***with meanings in*** M *is a relation* $\mathfrak{I} \subseteq A^* \times M$. *The* ***string language associated with*** $\mathfrak{I}$ *is*

$$L(\mathfrak{I}) := \{\vec{x} : \text{ there is } m \in M \text{ such that } \langle \vec{x}, m\rangle \in \mathfrak{I}\}$$

The meanings expressed by $\mathfrak{I}$ *are*

$$M(\mathfrak{I}) := \{m : \text{ there is } \vec{x} \in A^* \text{ such that } \langle \vec{x}, m\rangle \in \mathfrak{I}\}$$

We can change the definition slightly as follows. We may regard a language as a function from A^* to $\wp(M)$. Then $L(f) := \{\vec{x} : f(\vec{x}) \neq \varnothing\}$ is the string language associated with f and $M(f) := \bigcup_{\vec{x}\in A^*} f(\vec{x})$ the set of expressed meanings of f. Both definitions are equivalent, though we prefer to use the first.

We shall give an example, which shows the problems already clear enough. We consider the number terms as known from the everyday life as for example `((3` $+$ `5)` $\times$ `2)`. We shall write a grammar with which we can compute the value of a term as soon as its analysis is known. This means that we regard an interpreted language as a set of pairs $\langle t, x\rangle$ where t is an arithmetical term and x its value. Of course, the analysis does not directly reveal the value but we must in addition to the rules of the grammar specify in which way the value of the term is computed inductively over the analysis. Since the nodes correspond to the subterms this is straightforward. Let T be the following grammar.

$$\begin{array}{lcl} \mathtt{T} & \to & \mathtt{(T+T)} \mid \mathtt{(T-T)} \mid \mathtt{(T\times T)} \mid \mathtt{(T\div T)} \\ \mathtt{T} & \to & \mathtt{Z} \mid \mathtt{(-Z)} \\ \mathtt{Z} & \to & \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \ldots \mid \mathtt{9} \end{array}$$

(This grammar only generates terms which have ciphers in place of decimal strings. This must be seen as an example only.) Let now an arbitrary term be given. To this term corresponds a unique number (if for a moment we disregard division by 0). This number can indeed be determined by induction over the term. To this end we define a partial interpretation map I, which if defined assigns a number to a given term.

$$\begin{array}{lcl} I(\texttt{(}\vec{x}\texttt{+}\vec{y}\texttt{)}) & := & I(\vec{x}) + I(\vec{y}) \\ I(\texttt{(}\vec{x}\texttt{-}\vec{y}\texttt{)}) & := & I(\vec{x}) - I(\vec{y}) \\ I(\texttt{(}\vec{x}\times\vec{y}\texttt{)}) & := & I(\vec{x}) \times I(\vec{y}) \\ I(\texttt{(}\vec{x}\div\vec{y}\texttt{)}) & := & I(\vec{x}) \div I(\vec{y}) \\ I(\texttt{(-}\vec{x}\texttt{)}) & := & -I(\vec{x}) \\ I(\texttt{0}) & := & 0 \\ I(\texttt{1}) & := & 1 \\ & \cdots & \\ I(\texttt{9}) & := & 9 \end{array}$$

If a function f is undefined on x we write $f(x) = \star$. We may also regard $\star$ as a value. The rules for $\star$ are then as follows. If one argument is $\star$, so is the value. Additionally, $a/0 = \star$ for all a. If $\vec{x}$ is a term, then $I(\vec{x})$ is uniquely defined. For either $\vec{x}$ is a cipher from 0 to 9 or it is a negative cipher, or $\vec{x} = \texttt{(}\vec{y}_1 \odot \vec{y}_2\texttt{)}$ for some uniquely determined $\vec{y}_1$ and $\vec{y}_2$ and $\odot \in \{+, -, \times, \div\}$. In this way one can calculate $I(\vec{x})$ if one knows $I(\vec{y}_1)$ and $I(\vec{y}_2)$. The value of a term can be found by naming a derivation and then computing the value of each of its subterms. Notice that the grammar is transparent so that only one syntactical analysis can exist for each string.

The method just described has a disadvantage: the interpretation of a term is in general not unique, for example if a string is ambiguous. (For example, if we leave out all brackets then the term $\texttt{3+5}\times\texttt{2}$ has two values, 13 or 16.) As said above, we could define for each string a set of numbers as its meaning. If the language is unambiguous this set has at most one member. Further, we have $I(\vec{x}) \neq \varnothing$ only if $\vec{x}$ is a constituent. However, in general we wish to avoid taking this step. Different meanings should arise only from different analyses. There is a way to implement this idea no matter what the grammar is. Let U be the grammar which results from T by deleting the brackets of T.

$$\begin{array}{lcl}
\mathtt{T} & \to & \mathtt{T}+\mathtt{T} \mid \mathtt{T}-\mathtt{T} \mid \mathtt{T}\times\mathtt{T} \mid \mathtt{T}\div\mathtt{T} \\
\mathtt{T} & \to & \mathtt{Z} \mid -\mathtt{Z} \\
\mathtt{Z} & \to & \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \ldots \mid \mathtt{9}
\end{array}$$

The strings of U can be viewed as images of a canonical transparent grammar. This could be T. However, for some reasons we choose a different grammar. Intuitively, we think of the string as the image of a term which codes the derivation tree. This tree differs from the structure tree in that the intermediate symbols are not nonterminals but symbols for rules. The derivation tree we code as a term in Polish Notation. For each rule ρ we add a new symbol $\mathtt{R}_\rho$. In place of the rule $\rho = A \to \vec{\alpha}$ we now take the rule $A \to \mathtt{R}_\rho\vec{\alpha}$. This grammar, call it V, is transparent. $\vec{x} \in L(V)$ is called a **derivation term**. We define two maps ζ and ι. ζ yields a string for each derivation term, and ι yields an interpretation. Both maps shall be homomorphisms from the term algebra, though the concrete definition is defined over strings. ζ can be uniformly defined by deleting the symbols $\mathtt{R}_\rho$. However, notice that the rules below yield values only if the strings are derivation terms.

$$\begin{array}{lcl}
\zeta(\mathtt{R}_\rho\vec{\alpha}_0 \ldots \vec{\alpha}_{n-1}) & := & \zeta(\alpha_0)\cdot\zeta(\alpha_1)\cdot\ldots\cdot\zeta(\alpha_{n-1}) \\
\zeta(\alpha) & := & \alpha
\end{array}$$

Here, α is different from all $\mathtt{R}_\rho$. We have assumed here that the grammar has no rules of the form $A \to \varepsilon$ even though a simple adaptation can help here as well. Now on to the definition of ι. In our concrete case this is without problems.

$$\begin{array}{lcl}
\iota(\mathtt{R}_{+}\vec{\alpha}_0 + \vec{\alpha}_1) & := & \iota(\vec{\alpha}_0) + \iota(\vec{\alpha}_1) \\
\iota(\mathtt{R}_{-2}\vec{\alpha}_0 - \vec{\alpha}_1) & := & \iota(\vec{\alpha}_0) - \iota(\vec{\alpha}_1) \\
\iota(\mathtt{R}_{\times}\vec{\alpha}_0 \times \vec{\alpha}_1) & := & \iota(\vec{\alpha}_0) \times \iota(\vec{\alpha}_1) \\
\iota(\mathtt{R}_{\div}\vec{\alpha}_0 \div \vec{\alpha}_1) & := & \iota(\vec{\alpha}_0) \div \iota(\vec{\alpha}_1) \\
\iota(\mathtt{R}_{-1} - \vec{\alpha}) & := & -\iota(\vec{\alpha})
\end{array}$$

Here we have put the derivation term into Polish Notation. As we know they are uniquely readable. However, this only holds under the condition that every symbol is unique. Notice, namely, that some symbols can have different meanings — as in our example the minus symbol. To this end we have added an additional annotation of the symbols. Using a superscript we have distinguished between the unary minus and the binary one. Since the actual language does not do so (we write '$-$' without distinction), we have written $\mathtt{R}_{-1}$ if the rule for the unary symbol has been used, and $\mathtt{R}_{-2}$ if the one for the binary symbol has been used.

The mapping ι is a homomorphism of the algebra of derivation terms into the algebra of real numbers with $\star$, which is equivalent to a partial homomorphism from the algebra of terms to the algebra of real numbers. For example the symbol $\mathtt{R}_{+}$ is interpreted by the function $+ : \mathbb{R}_\star \times \mathbb{R}_\star \to \mathbb{R}_\star$, where $\mathbb{R}_\star := \mathbb{R} \cup \{\star\}$ and $\star$ satisfies the laws specified above. In principle this algebra can be replaced by any other which allows to interpret unary and binary function symbols. We emphasize that it is not necessary to have the interpreting functions be basic functions of the algebras. It is enough that they can be built from the basic functions. For example we can introduce a unary function symbol $\mathtt{d}$ whose interpretation is the duplication. Now $2x = x + x$, and hence the duplication is a polynomial function of the algebra $\langle \mathbb{R}, +, \cdot, 0, 1\rangle$, which is not a basic function.

This exposition motivates a terminology which sees meanings and strings as images of abstract signs under a homomorphism. We shall now develop this idea in full generality. The basis is formed by the notion of an algebra of signs.

Definition 3.1.2 *Let $\langle F, \Omega\rangle$ be a signature. A* ***partial*** *Ω*–***algebra*** *is a pair $\mathfrak{A} = \langle A, \Pi\rangle$ where A is an arbitrary nonempty set and for $f \in F$ $\Pi(f)$ is a partial $\Omega(f)$–ary function on A. We write $f^{\mathfrak{A}}$ in place of $\Pi(f)$. If $\mathfrak{B} = \langle B, \{f^{\mathfrak{B}} : f \in F\}\rangle$ is a partial Ω–algebra and $h : A \to B$ h is called a* ***homomorphism*** *from $\mathfrak{A}$ to $\mathfrak{B}$ if the following holds: if $\vec{a} \in A^{\Omega(f)}$ and if $f^{\mathfrak{A}}(\vec{a})$ is defined, then $f^{\mathfrak{B}}(h(\vec{a}))$ is also defined and*

$$f^{\mathfrak{B}}(h(\vec{a})) = h(f^{\mathfrak{A}}(\vec{a})) \ .$$

*We say, h is a **strong homomorphism** if $h(f^{\mathfrak{A}}(\vec{a}))$ is defined if and only if $f^{\mathfrak{B}}(h(\vec{a}))$ is defined. If $B \subseteq A$, then $\mathfrak{B}$ is a **subalgebra** of $\mathfrak{A}$ if the embedding $i : B \to A : x \mapsto x$ is a strong homomorphism.*

A subalgebra is determined by the set B. The functions on B are the restrictions of the respective functions on A. Notice that it is not allowed to partialize functions additionally. For example, $\langle A, \Xi \rangle$ with $\Xi(f) = \varnothing$ is not a subalgebra of $\mathfrak{A}$ unless $\Pi(f) = \varnothing$.

A **sign** is a triple $\sigma = \langle E, C, M \rangle$ where E is the exponent of σ, usually some kind of string over an alphabet A, C the category of σ and M its meaning. Abstractly, however, we shall set this up differently. We shall first define an algebra of signs as such, and introduce exponent, category and meaning as value of the signs under some homomorphisms. This will practically amount to the same, however. So, we start by fixing a signature $\langle F, \Omega \rangle$. In this connection the function symbols from F are called **modes**. Over this signature we shall define an algebra of signs, of exponents, of categories and meanings. An algebra of signs over $\langle F, \Omega \rangle$ is simply a 0–generated partial algebra $\mathfrak{A}$ over this signature together with certain homomorphisms, which will be defined later.

Definition 3.1.3 *A (partial) Ω–algebra $\mathfrak{A} = \langle A, \Pi \rangle$ is called n–**generated** if there is an n–element subset $X \subseteq A$ such that the smallest subalgebra containing X is $\mathfrak{A}$.*

Definition 3.1.4 *A **sign grammar over the signature** Ω is a quadruple $\langle \mathfrak{A}, \varepsilon, \gamma, \mu \rangle$ where $\mathfrak{A}$ is a 0–generated partial Ω–algebra and $\varepsilon : \mathfrak{A} \to \mathfrak{E}$, $\gamma : \mathfrak{A} \to \mathfrak{C}$ and $\mu : \mathfrak{A} \to \mathfrak{M}$ homomorphisms to certain partial Ω–algebras such that the homomorphism $\langle \varepsilon, \gamma, \mu \rangle$ is injective and strong. $\mathfrak{A}$ is called the **algebra of signs**, $\mathfrak{E}$ the **algebra of exponents**, $\mathfrak{C}$ the **algebra of categories** and $\mathfrak{M}$ the **algebra of meanings**.*

This means in particular:

∗ Every sign σ is uniquely characterized by three things:

– its so called **exponent** $\varepsilon(\sigma)$,

– its syntactical category $\gamma(\sigma)$ (which is also often called its **type**),

– its meaning $\mu(\sigma)$.

* To every function symbol $f \in F$ corresponds an $\Omega(f)$–ary function f^{ε} in $\mathfrak{E}$, an $\Omega(f)$–ary function f^{γ} in $\mathfrak{C}$ and an $\Omega(f)$–ary function f^{μ} in $\mathfrak{M}$.

* Signs can be combined with the help of the function $f^{\mathfrak{A}}$ any time their respective exponents can be combined with the help of f^{ε}, their respective categories can be combined with f^{γ} and their respective meanings with f^{μ}. (This corresponds to the condition of strongness.)

If σ is a sign, then $\langle \varepsilon(\sigma), \gamma(\sigma), \mu(\sigma)\rangle$ is uniquely defined by σ, and on the other hand it defines σ uniquely as well. We shall call the triple the **realization** of σ. Additionally, we can represent σ by a term in the free Ω–algebra. We shall now deal with the correspondences between these viewpoints.

Let $\mathfrak{S}$ be the Ω–algebra which is freely 0–generated algebra. Its domain consists of all terms, which we shall take to be strings in Polish Notation. The mode g is represented by the function $g^{\mathfrak{S}}$:

$$g^{\mathfrak{S}}(\vec{x}_0, \ldots, \vec{x}_{\Omega(g)-1}) := g \cdot \prod_{i<\Omega(g)} \vec{x}_i$$

So we put $\mathfrak{S} := \langle S, \{g^{\mathfrak{S}} : g \in F\}\rangle$. The elements of S are called **structure terms**. We use $\mathfrak{s}$, $\mathfrak{t}$ and so on as variables for structure terms. We give an example. Suppose that $\mathtt{N}$ is a 0–ary mode and $\mathtt{S}$ a unary mode. Then we get the following structure terms.

$$\mathtt{N}, \mathtt{SN}, \mathtt{SSN}, \mathtt{SSSN}, \ldots$$

Furthermore, we have $\mathtt{S}^{\mathfrak{S}} : \vec{x} \mapsto \mathtt{S} \cdot \vec{x}$.

We denote by $h : M \stackrel{p}{\to} N$ the fact that h is a partial function from M to N. We now define partial maps $\dot{\varepsilon} : S \stackrel{p}{\to} E$, $\dot{\gamma} : S \stackrel{p}{\to} C$ and $\dot{\mu} : S \stackrel{p}{\to} M$ in the following way.

$$\dot{\varepsilon}(g^{\mathfrak{S}}(\mathfrak{s}_0, \ldots, \mathfrak{s}_{\Omega(g)-1})) := g^{\varepsilon}(\dot{\varepsilon}(\mathfrak{s}_0), \ldots, \dot{\varepsilon}(\mathfrak{s}_{\Omega(g)-1}))$$

Here, the left hand side is defined if and only if the right hand side is and then the two are equal. If we have a 0–ary mode g, then it is a structure term $\dot{\varepsilon}(g) = g^{\varepsilon} \in E$. Likewise we define the other maps.

$$\begin{array}{rcl} \dot{\gamma}(g^{\mathfrak{S}}(\mathfrak{s}_0, \ldots, \mathfrak{s}_{\Omega(g)-1})) & := & g^{\gamma}(\dot{\gamma}(\mathfrak{s}_0), \ldots, \dot{\gamma}(\mathfrak{s}_{\Omega(g)-1})) \\ \dot{\mu}(g^{\mathfrak{S}}(\mathfrak{s}_0, \ldots, \mathfrak{s}_{\Omega(g)-1})) & := & g^{\mu}(\dot{\mu}(\mathfrak{s}_0), \ldots, \dot{\mu}(\mathfrak{s}_{\Omega(g)-1})) \end{array}$$

As remarked above, for every signs there is a structure term. The converse need not hold.

Definition 3.1.5 *We say, a structure term $\mathfrak{s}$ is **orthographically definite** if $\dot{\varepsilon}(\mathfrak{s})$ is defined. $\mathfrak{s}$ is **syntactically definite** if $\dot{\gamma}(\mathfrak{s})$ is defined and **semantically definite** if $\dot{\mu}(\mathfrak{s})$ is defined. Finally, $\mathfrak{s}$ is **definite** if $\mathfrak{s}$ is orthographically, syntactically as well as semantically definite.*

Definition 3.1.6 *The partial map $\zeta := \langle \dot{\varepsilon}, \dot{\gamma}, \dot{\mu} \rangle$ is called the **unfolding map**.*

$\mathfrak{A}$ is isomorphic to the partial algebra of all $\langle \dot{\varepsilon}(\mathfrak{s}), \dot{\gamma}(\mathfrak{s}), \dot{\mu}(\mathfrak{s}) \rangle$, where $\mathfrak{s}$ is a definite structure term. This we can attack also differently. Let D be the set of definite structure terms. This set becomes a partial Ω–algebra together with the partial functions $g^{\mathfrak{S}} \restriction D$. We denote this algebra by $\mathfrak{D}$. $\mathfrak{D}$ is usually not a subalgebra of $\mathfrak{S}$. For let $j : \mathfrak{s} \mapsto \mathfrak{s}$ be the identity map. Then we have $j(g^{\mathfrak{D}}(\mathfrak{s}_0, \ldots, \mathfrak{s}_{\Omega(g)-1})) = g^{\mathfrak{S}}(j(\mathfrak{s}_0), \ldots, j(\mathfrak{s}_{\Omega(g)-1}))$. The right hand side is always defined, the left hand side need not be.

The homomorphism $\zeta \restriction D$ (which we also denote by ζ) is however strong. Now look at the following relation $\Theta := \{\langle \sigma_0, \sigma_1 \rangle : \zeta(\sigma_0) = \zeta(\sigma_1)\}$. This is a congruence on $\mathfrak{D}$; this relation is an equivalence relation and if $\mathfrak{s}_i \;\Theta\; \mathfrak{t}_i$ for all $i < \Omega(f)$ then $f(\vec{\mathfrak{s}})$ is defined if and only if $f(\vec{\mathfrak{s}})$ is defined. And in this case we have $f(\vec{\mathfrak{s}}) \;\Theta\; f(\vec{\mathfrak{s}})$. We can now put:

$$f^{\mathfrak{A}}(\langle [\mathfrak{s}_i]_\Theta : i < \Omega(f)\rangle) := [f(\langle \mathfrak{s}_i : i < \Omega(f)\rangle)]_\Theta$$

This is well defined and we get an algebra, the algebra $\mathfrak{D}/\Theta$. The following is easy to see.

Proposition 3.1.7

$$\mathfrak{A} \cong \mathfrak{D}/\Theta\ .$$

So, $\mathfrak{D}/\Theta$ is isomorphic to the algebra of signs. It is to be noted that for every sign there is a structure term, but there can also be several. As an instructive example we look at the algebra of remainders modulo 60. The modes shall be $\mathtt{N}$ (the zero, 0–ary) and $\mathtt{S}$ (the successor function, unary). Then $\mathtt{S}$ is totally defined, but we have

$$\zeta(\mathtt{N}) = \zeta(\mathtt{S}^{60}\mathtt{N})\ .$$

Hence every sign has infinitely many structure terms, and so is inherently structurally ambiguous. This algebra arises naturally. Look at the hands of a clock. The minute hand in modern clocks advances once a minute. $\mathtt{N}$ denotes the vertical position, and $\mathtt{S}$ its advancement by one minute. The meaning shall be the number of minutes that have passed since the last full hour. Indeed, knowing only the position of the minute hand we only know the minutes, not the hour, and so every sign is structurally ambiguous. If instead we take as meanings the natural numbers and $\mathtt{N}^\mu := 0$ as well as $\mathtt{S}^\mu := \lambda n.n + 1$ then every structure term represents a different sign! However, still there are only 60 exponents. Only that every exponent has infinitely meanings.

We shall illustrate the concepts of a sign grammar by proceeding with our initial example. Our alphabet is now

$$R := \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}, \times, -, +, \div, \mathtt{(}, \mathtt{)}\}$$

The algebra $\mathfrak{E}$ consists of R^* together with some functions that we still have to determine. We shall now begin to determine the modes. They are $\mathtt{R}_+$, $\mathtt{R}_{-2}$, $\mathtt{R}_\times$, $\mathtt{R}_\div$, which are binary, $\mathtt{R}_{-1}$, $\mathtt{V}$, which are unary and finally ten 0–ary modes, namely $\mathtt{Z}_0$, $\mathtt{Z}_1$, ..., $\mathtt{Z}_9$.

We begin with the 0–ary modes. These are, by definition, signs. For their identification we only need to know the three components. For example, to the mode $\mathtt{Z}_0$ corresponds the triple $\langle \mathtt{0}, \mathtt{Z}, 0\rangle$. This means: the exponent of the signs $\mathtt{Z}_0$ (what we get to see) is the digit $\mathtt{0}$; its category is $\mathtt{Z}$ and its meaning the number 0. Likewise with the other 0–ary modes. Now on to the unary modes. These are operations taking signs to make new signs. We begin with $\mathtt{R}_{-1}$. On the level of strings we get the polynomial $\mathtt{R}^{\varepsilon}_{-1}$, which is defined as follows.

$$\mathtt{R}^{\varepsilon}_{-1}(\vec{x}) := \mathtt{(} - \vec{x}\mathtt{)}\ .$$

On the level of categories we get the function

$$\mathtt{R}^{\gamma}_{-1}(K) := \begin{cases} \mathtt{T}\ , & \text{if } K = \mathtt{Z}, \\ \star\ , & \text{otherwise.} \end{cases}$$

Here $\star$ is again the symbol for the fact that the function is not defined. Finally we have to define $\mathtt{R}^{\mu}_{-1}$. We put

$$\mathtt{R}^{\mu}_{-1}(x) := -x\ .$$

Notice that even if the function $x \mapsto -x$ is iterable, the mode $\mathtt{R}_{-1}$ is not. This is made impossible by the categorial assignment. This is an artefact of the example. We could have set things up differently. The mode $\mathtt{V}$ finally is defined by the following functions. $\mathtt{V}^{\varepsilon}(\vec{x}) := \vec{x}$, $\mathtt{V}^{\mu}(x) := x$ and $\mathtt{V}^{\gamma}(K) := \mathtt{R}_{-1}(K)$. Finally we turn to the binary modes. Let us look at $\mathtt{R}_\div$. We have that $\mathtt{R}^{\mu}_{\div}$ is the partial (!) binary function $\div$ on $\mathbb{R}$. Further, we put

$$\mathtt{R}^{\varepsilon}_{\div}(\vec{x}, \vec{y}) := \mathtt{(}\vec{x} \div \vec{y}\mathtt{)}\ .$$

as well as

$$\mathtt{R}^{\gamma}_{\div}(K, L) := \begin{cases} \mathtt{T} & \text{if } K = L = \mathtt{T}, \\ \star & \text{otherwise.} \end{cases}$$

The string

$$\mathtt{R}_\times\mathtt{R}_+\mathtt{Z}_3\mathtt{Z}_5\mathtt{Z}_7$$

defines — as is easily computed — a sign whose exponent is $\mathtt{((3+5)\times 7)}$. By contrast, $\mathtt{R}_\div\mathtt{Z}_2\mathtt{Z}_0$ does *not* represent a sign. For it has the exponent $\mathtt{(1 \div 0)}$. This is syntactically well formed but its meaning cannot exist since we may not divide by 0.

Definition 3.1.8 *A* ***linear system of signs over the alphabet*** *A, the set of categories C and the meanings M is a set $\Sigma \subseteq A^* \times C \times M$. Further, let $\mathtt{S}$ be a category. Then the interpreted language of Σ with respect to this category $\mathtt{S}$ is defined by*

$$\mathtt{S}(\Sigma) := \{\langle \vec{x}, m\rangle : \langle \vec{x}, \mathtt{S}, m\rangle \in \Sigma\} .$$

We added the qualifying phrase 'linear' to distinguish this from sign systems which do not generally take strings as exponents. (For example, pictograms are nonlinear, as discussed in Section 1.3.)

A system of signs is simply a set of signs. The question is whether one can define an algebra over it. This is always possible. Just take a 0–ary mode for every sign. Since this is certainly not as intended, we shall restrict the possibilities as follows.

Definition 3.1.9 *Let $\Sigma \subseteq E \times C \times M$ be a system of signs. We say that Σ is* ***compositional*** *if there is a finite signature Ω and partial Ω–algebras $\mathfrak{E} = \langle E, \{f^{\mathfrak{E}} : f \in F\}\rangle$, $\mathfrak{C} = \langle C, \{f^{\mathfrak{C}} : f \in F\}\rangle$, $\mathfrak{M} = \langle M, \{f^{\mathfrak{M}} : f \in F\}\rangle$ such that all functions are computable and Σ consists of exactly the set of 0–ary signs from $\mathfrak{E} \times \mathfrak{C} \times \mathfrak{M}$. Σ is* ***weakly compositional*** *if there is a compositional system $\Sigma' \subseteq E' \times C' \times M'$ such that $\Sigma = \Sigma' \cap E \times C \times M$.*

We remark that a partial function $f : M^n \xrightarrow{p} M$ in the sense of the definition above is a computable total function $f^\star : M_\star^n \to M_\star$ such that $f^\star \upharpoonright M^n = f$. So, the computation always halts, and we are told at its end whether or not the function is defined and if so what the value is.

Two conditions have been made: the signature has to be finite and the functions on the algebras computable. We shall show that however strong they appear, they do not really restrict the class of sign systems in comparison to weak compositionality.

We start by drawing some immediate conclusions from the definitions. If σ is a sign we say that $\langle \varepsilon(\sigma), \gamma(\sigma), \mu(\sigma)\rangle$ is its **realization** or **unfolding**. We have introduced the unfolding map ζ above.

Proposition 3.1.10 *Let $\langle \mathfrak{A}, \varepsilon, \gamma, \mu\rangle$ be a compositional sign grammar. Then the unfolding map is computable.*

For a proof notice that the unfolding of a structure term can be done inductively. This has the following immediate consequence.

Corollary 3.1.11 *Let Σ be compositional. Then Σ is recursively enumerable.*

This is remarkable inasmuch as the the set of all signs over $E \times C \times M$ need not even be enumerable. For typically M contains uncountably many elements (which can of course not all be named by a sign)! Now, a weak converse also holds.

Theorem 3.1.12 *A system of signs is weakly compositional if and only if it is recursively enumerable.*

Proof. Let $\Sigma \subseteq E \times C \times M$ be given. If Σ is weakly compositional, it also is recursively enumerable. Now, let us assume that Σ is recursively enumerable, say $\Sigma = \{\langle e_i, t_i, m_i\rangle : 0 < i \in \omega\}$. (Notice that we start counting with 1.) Now let $\mathtt{V}$ be a symbol and $\Delta := \{\langle \mathtt{V}^n, \mathtt{V}^n, \mathtt{V}^n\rangle : n \in \omega\}$ a system of signs. By properly choosing $\mathtt{V}$ we can see to it that $\Delta \cap \Sigma = \varnothing$ and that no $\mathtt{V}^n$ occurs in E, C or M. Let $F := \{\mathtt{Z}_0, \mathtt{Z}_1, \mathtt{Z}_2\}$, $\Omega(\mathtt{Z}_0) := 0$, $\Omega(\mathtt{Z}_1) := 1$ and $\Omega(\mathtt{Z}_2) := 1$.

$$\begin{aligned}
\mathtt{Z}_0 &:= \langle \mathtt{V}, \mathtt{V}, \mathtt{V}\rangle, \\
\mathtt{Z}_1(\sigma) &:= \begin{cases} \langle \mathtt{V}^{i+1}, \mathtt{V}^{i+1}, \mathtt{V}^{i+1}\rangle & \text{if } \sigma = \langle \mathtt{V}^i, \mathtt{V}^i, \mathtt{V}^i\rangle, \\ \star & \text{otherwise,} \end{cases} \\
\mathtt{Z}_2(\sigma) &:= \begin{cases} \langle e_i, t_i, m_i\rangle & \text{if } \sigma = \langle \mathtt{V}^i, \mathtt{V}^i, \mathtt{V}^i\rangle, \\ \star & \text{otherwise.} \end{cases}
\end{aligned}$$

This is well defined. Further, the functions are all computable. For example, the map $\mathtt{V}^i \mapsto e_i$ is computable since it is the concatenation of the computable functions $\mathtt{V}^i \mapsto i$, $i \mapsto \langle e_i, t_i, m_i\rangle$ with $\langle e_i, t_i, m_i\rangle \mapsto e_i$. We claim: the system of signs generated is exactly $\Delta \cup \Sigma$. For this we notice first that a structure term is definite if and only if has the following form. (a) $t = \mathtt{Z}_1^i\mathtt{Z}_0$, or (b) $t = \mathtt{Z}_2\mathtt{Z}_1^i\mathtt{Z}_0$. In Case (a) we get the sign $\langle \mathtt{V}^{i+1}, \mathtt{V}^{i+1}, \mathtt{V}^{i+1}\rangle$, in Case (b) the sign $\langle e_{i+1}, t_{i+1}, m_{i+1}\rangle$. Hence we generate exactly $\Delta \cup \Sigma$. So, Σ is weakly compositional. □

Notice that the algebra of exponents uses additional symbols which are used only to create new objects which are like natural numbers. The just presented algebra is certainly not very satisfying. (It is also not compositional.) Hence one has sought to provide a more systematic theory of categories and their meanings. A first step in this direction are the categorial grammars. To motivate them we shall give a construction for context free grammars that differs markedly from the one in Theorem 3.1.12. The starting point is once again an interpreted language $\mathfrak{I} = \{\langle \vec{x}, f(\vec{x})\rangle : \vec{x} \in S\}$, where S is now context free and f computable. Then let $G = \langle \mathtt{S}, N, A, R\rangle$ be a context free grammar with $L(G) = S$. Put $A' := A$, $C' := N \cup \{\mathtt{S}^\heartsuit\}$ and $M' := M \cup A^*$. We presuppose for simplicity's sake that G is already in Chomsky Normal Form. For every rule of the form $\rho : A \to \vec{x}$ we take a 0–ary mode $\mathtt{R}_\rho$, which is defined as follows:

$$\mathtt{R}_\rho = \langle \vec{x}, A, \vec{x}\rangle \ .$$

For every rule of the form $\rho : A \to B\ C$ we take a binary mode $\mathtt{R}_\rho$ defined by

$$\mathtt{R}_\rho(\langle \vec{x}, B, \vec{x}\rangle, \langle \vec{y}, C, \vec{y}\rangle) := \langle \vec{x}\,\vec{y}, A, \vec{x}\,\vec{y}\rangle \ .$$

Finally we choose a unary mode for $\mathtt{S}$:

$$\mathtt{S}(\langle \vec{x}, \mathtt{S}, \vec{x}\rangle) := \langle \vec{x}, \mathtt{S}^\heartsuit, f(\vec{x})\rangle \ .$$

Then $\mathfrak{I}$ is indeed the set of signs with category $\mathtt{S}^\heartsuit$. As one can see, this algebra of signs is more perspicuous. The strings are just concatenated. The meanings, however, are not the ones we expect to see. And the category assignment is unstructured. This grammar is not compositional, since it still uses non standard meanings. Hence once again a pathological example.

Definition 3.1.13 *Let $Z = \langle \mathfrak{A}, \varepsilon, \gamma, \mu \rangle$ be a system of signs. We say, Z* ***has natural numbers of category*** *α if the following holds.*

1. *$\Sigma_\alpha := \{\sigma : \gamma(\sigma) = \alpha\}$ is infinite.*

2. *$\Sigma_\alpha = \{\langle E_i, \alpha, M_i \rangle : i \in \omega\}$, where $E_i \mapsto i : E \to \omega$ and $M_i \mapsto i : M \to \omega$ are bijective, computable functions.*

Definition 3.1.14 *Let Δ be a language of signs. Δ is called* ***modularly decidable*** *if the following holds.*

1. *Δ is decidable.*

2. *For given $e \in E$ it is decidable whether there exists a $\sigma \in \Delta$ such that $\varepsilon(\sigma) = e$.*

3. *For given $c \in C$ it is decidable whether there exists a $\sigma \in \Delta$ such that $\gamma(\sigma) = c$.*

4. *For given $m \in M$ it is decidable whether there exists a $\sigma \in \Delta$ such that $\mu(\sigma) = m$.*

Proposition 3.1.15 *Let Σ be compositional, and let $\Delta \subseteq \Sigma$ be modularly decidable. Then Δ is compositional, too.*

Theorem 3.1.16 (Extension) *Let Σ be a recursively enumerable set of signs with finitely many categories. Let $\Delta \subseteq \Sigma$ be modularly decidable, compositional, and let Δ have natural numbers of category α. Then there exists a compositional grammar for Σ.*

Proof. By assumption, Δ has natural numbers. Hence there exists an α and computable bijective functions $z : E \to \omega$ and $m : M \to \omega$ such that

$$\Delta_\alpha := \{\sigma : \gamma(\sigma) = \alpha\} = \{\langle z^{-1}(j), \alpha, m^{-1}(j)\rangle : j \in \omega\} .$$

Further, Σ is recursively enumerable and there exist only finitely many categories. Then it is easily seen that for every category γ there is a pair of computable functions z_γ and m_γ such that

$$\Sigma_\gamma = \{\langle z_\gamma(j), \gamma, m_\gamma(j)\rangle : j \in \omega\} .$$

We take a compositional grammar for Δ with signature Ω_1. By Proposition 3.1.15 we may assume that the grammar generates only the signs of Δ. (This is not entirely trivial since Δ may be defined from a smaller alphabet, a smaller set of categories and so on, the algorithms are however defined on the bigger domains.) Then let $F := F_1 \mathbin{\dot\cup} \{\mathtt{E}_\gamma : \gamma \in C\}$ with $\Omega_1(\mathtt{E}_\gamma) := 1$.

$$\mathtt{E}_\gamma^\Sigma(\langle E, \alpha, M\rangle) := \langle z_\gamma(z(E)), \gamma, m_\gamma(m(M))\rangle .$$

Since m, z as well as m_γ and z_γ are computable also $z_\gamma \circ z : E \to E$ and $m_\gamma \circ m : M \to M$ are computable.

Now we must verify that this grammar generates Σ. To this end let $\sigma = \langle E, C, M\rangle$ be a sign and $\mathfrak{s}$ a structure term of σ. By induction on the length of $\mathfrak{s}$ we shall show that $\sigma \in \Sigma$. If $\mathfrak{s}$ has length 1, then $\mathfrak{s}$ is a 0–ary mode, hence in Ω. Then we even have $\sigma \in \Delta$, by assumption on Ω. Now on to the inductive step. (Case 1.) $\mathfrak{s} = \mathtt{E}_\gamma\mathfrak{t}$ for some $\mathfrak{t}$. By inductive hypothesis $\mathfrak{t}$ is the structure term of a sign $\sigma' = \langle E', C', M'\rangle \in \Sigma$. We have $C' = \alpha$ and so $\sigma' \in \Delta$. Then by construction $\sigma \in \Sigma$. (Case 2.) The main symbol of $\mathfrak{s}$ is in Ω. Then $\sigma \in \Delta$ since the modes from Ω only generate signs from Δ. This shows that our grammar does not generate too many meanings. That it also does not generate too few meanings is seen thus. Let $\sigma \in \Sigma$. Then $\sigma = \langle E, \gamma, M\rangle$ for some γ. Then there is a number j such that $\sigma = \langle z_\gamma(j), \gamma, m_\gamma(j)\rangle$. Put $E' := z^{-1}(j)$, $M' := m^{-1}(j)$. Then we have $\langle E', \alpha, M'\rangle \in \Delta$

and it possesses a structure term $\mathfrak{r}$. Then σ has the structure term $\mathtt{E}_\gamma\mathfrak{r}$, as can be checked. ☐

The twist of the construction is that all occurring signs are in Σ. But what happens is still suspicious. Namely, $\vec{x}$ is suddenly turned into the string $f(\vec{x})$, which may have nothing to do with $\vec{x}$ at all. Evidently, we need to further restrict our operations, for example, by not allowing arbitrary string manipulations. We shall deal with this problem in Section 5.7.

Compositionality in the weak sense defines semantics as an autonomous component of language. When a rule is applied, the semantics may not 'spy' into the phonological form or the syntax to see what it is supposed to do. Rather, it acts autonomously, without that knowledge. Its only input is the semantics of the argument signs and the mode that is being applied. In a similar way syntax is autonomous from phonology and semantics. That this is desirable has been repeatedly argued for by Noam Chomsky. It means that syntactic rules apply regardless of the semantics or the phonological form. It is worthwile to explain that our notion of compositionality not only makes semantics autonomous from syntax and phonology, but also syntax autonomous from phonology and semantics and phonology autonomous from syntax and semantics.

Notes on this section. The notion of sign defined here is the one that is most commonly found in linguistics. In essence it goes back to (Saussure, 1967), who takes a linguistic sign to consist of a signifier and denotatum (see also Section 5.8). De Saussure therewith diverged from Peirce, for whom a sign was a triadic relation between signifier, subject and denotatum. (See also (Lyons, 1978) for discussion.) On the other hand, it adds to de Saussure's concept the category, which is nothing but a statement of the combinatorics of that sign. This structure of a sign is most clearly employed, for example, in Montague Grammar and in the writings of Igor Mel'čuk (see for example (Mel'cuk, 1993 200)). Other theories, for example HPSG and Unification Categorial Grammar also use the tripartite distinction between what they call phonology, syntax and semantics, but signs are not triples but much more complex in structure.

The distinction between compositionality and weak compositionality trades on the question whether the generating functions should work inside the language or whether they may introduce new objects. We strongly opt for the former not only because it gives us a stronger notion. The definition in its informal rendering makes reference to parts of an expression and their meanings — and in actual practice the parts from which we compose do have meanings, and it is these meanings we employ in forming the meaning of a complex expression.

Exercise 81. Show Proposition 3.1.15.

Exercise 82. Call Δ γ–**unique** if the following holds:

1. If $\langle e, \gamma, m\rangle, \langle e', \gamma, m\rangle \in \Delta$ then $e = e'$.
2. If $\langle e, \gamma, m\rangle, \langle e, \gamma, m'\rangle \in \Delta$ then $m = m'$.

Let $C = \{\gamma\}$ and Δ be modularly decidable and γ–unique. Show that Δ is compositional.

Exercise 83. Let C be finite, $E = A^*$ and let Δ be γ–unique for every $\gamma \in C$. Finally, let Δ be modularly decidable. Show that Δ is compositional. *Hint.* Define for every γ a binary mode $\mathtt{C}_\gamma$. Let $\mathtt{C}^\varepsilon_\gamma$ be the concatenation and $\mathtt{C}^\gamma_\gamma(t, t') := t$ if $t = t' = \gamma$, and undefined otherwise. The trick is the definition of $\mathtt{C}^\mu_\gamma$.

Exercise 84. Show that English satisfies the conditions of Theorem 3.1.16. Hence English is compositional! *Hint.* It is sufficient to find a subset Δ with $\Delta = \Delta_\alpha$, which is compositional. That English is recursively enumerable shall be assumed without proof.

Exercise 85. Construct a Δ which satisfies the Conditions 2. – 4. of modular decidability but not Condition 1. Construct likewise a Δ which satisfies 1. but not 2. or 3. or 4.

Exercise 86. Show that the functions postulated in the proof of Theorem 3.1.16, z_γ and m_γ, do exist if Σ is recursively enumerable.

Exercise 87. Say that $\Sigma \subseteq E \times C \times M$ is **extra weakly compositional** if there exists a finite signature Ω and Ω–algebras $\mathfrak{E}'$, $\mathfrak{C}'$ and $\mathfrak{M}'$ over sets $E' \supseteq E$, $C' \supseteq C$ and $M' \supseteq M$, respectively, such that Σ is the set of 0–ary signs in $\mathfrak{E}' \times \mathfrak{C}' \times \mathfrak{M}'$ which belong to the set $E \times C \times M$. (So, the definition is like that of weak compositionality, only that the functions are not necessarily computable.) Show that Σ is extra weakly compositional if and only if it is countable. (See also (Zadrozny, 1994).)

3.2 Propositional Logic

Before we can enter a discussion of categorial grammar and type systems, we shall have to introduce some techniques from propositional logic. We seize the opportunity to present boolean logic using our notions of the previous section. Let $A_P := \{\mathtt{p}, \mathtt{0}, \mathtt{1}, \mathtt{(}, \mathtt{)}, \bot, \rightarrow\}$. Further, let $T := \{P\}$, and $M := \{0, 1\}$. Next, we define the following modes. The nullary modes are

$$\begin{array}{lcl} \mathtt{X}_{\vec{\alpha}} & := & \langle \mathtt{p}\vec{\alpha}, P, 0\rangle \\ \mathtt{Y}_{\vec{\alpha}} & := & \langle \mathtt{p}\vec{\alpha}, P, 1\rangle \\ \mathtt{M}_{\bot} & := & \langle \bot, P, 0\rangle \end{array}$$

Here, $\vec{\alpha}$ ranges over (possibly empty) sequences of $\mathtt{0}$ and $\mathtt{1}$. Further, let $\supset$ be the following function:

$\supset$	0	1
0	1	1
1	0	1

Then the binary mode $\mathtt{M}_{\rightarrow}$ of implication formation is spelled out as follows.

$$\mathtt{M}_{\rightarrow}(\langle \vec{x}, P, \eta\rangle, \langle \vec{y}, P, \theta\rangle) \quad := \quad \langle \mathtt{(}\vec{x} \rightarrow \vec{y}\mathtt{)}, P, \eta \supset \theta\rangle$$

The system of signs generated by these modes is called **boolean logic** and is denoted by Σ_B. To see that this is indeed so, let us explain in more conventional terms what these definitions amount to. First, the string language L we have defined is a subset of A^*, which is generated as follows.

1. If $\vec{\alpha} \in \{\mathtt{0}, \mathtt{1}\}^*$, then $\mathtt{p}\vec{\alpha} \in L$. These sequences are called **propositional variables**.

2. $\bot \in L$.

3. If $\vec{x}, \vec{y} \in L$, then $(\vec{x} \to \vec{y}) \in L$.

$\vec{x}$ is also called a **well formed formula** (**wff**) if and only if it belongs to L. There are three kinds of wffs.

Definition 3.2.1 *Let $\vec{x}$ be a well formed formula. $\vec{x}$ is a **tautology** if $\langle \vec{x}, P, 0\rangle \notin \Sigma_B$. $\vec{x}$ is a **contradiction** if $\langle \vec{x}, P, 1\rangle \notin \Sigma_B$. If $\vec{x}$ is neither a tautology nor a contradiction, it is called **contingent**.*

The set of tautologies is denoted by $\mathrm{Taut}_B(\to, \bot)$, or simply by Taut_B if the language is clear from the context. It is easy to see that $\vec{x}$ is a tautology if and only if $(\vec{x} \to \bot)$ is a contradiction. Likewise, $\vec{x}$ is a contradiction if and only if $(\vec{x} \to \bot)$ is a tautology. We now agree on the following convention. Lower case Greek letters are proxy for well formed formulae, upper case Greek letters are proxy for sets of formulae. Further, we write $\Delta; \varphi$ instead of $\Delta \cup \{\varphi\}$ and $\varphi; \chi$ in place of $\{\varphi, \chi\}$.

Our first task will be to present a calculus with which we can generate all the tautologies of Σ_B. For this aim we use a so called *Hilbert–Style Calculus*. Define the following sets of formulae.

$$
\begin{array}{ll}
\text{(a0)} & (\varphi \to (\psi \to \varphi)) \\
\text{(a1)} & ((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))) \\
\text{(a2)} & (\bot \to \varphi) \\
\text{(a3)} & (((\varphi \to \bot) \to \bot) \to \varphi)
\end{array}
$$

More precisely, (a0) is a set, namely the set $\{(\varphi \to (\psi \to \varphi)) : \varphi, \psi \in L\}$. We call (a0) an **axiom schema** and elements of this set **instances of** (a0).

Definition 3.2.2 *A* ***proof of*** φ *is a finite sequence* $\Pi = \langle \delta_i : i < n \rangle$ *of formulae such that (a)* $\delta_{n-1} = \varphi$ *and (b) for all* $i < n$ *either (b1)* δ_i *is an instance of (a0) – (a3) or (b2) there are* $j, k < i$ *such that* $\delta_k = \delta_j \rightarrow \delta_i$*. The number* n *is called the* ***length of*** Π*. We write* $\vdash^B \varphi$ *if there is a proof of* φ*.*

The formulae (a0) – (a3) are called the **axioms** of this calculus. Moreover, this calculus uses a single inference rule, which is known as Modus Ponens. It is the inference from $(\varphi \rightarrow \chi)$ and φ to χ. The easiest part is to show that the calculus generates only tautologies.

Lemma 3.2.3 *If* $\vdash^B \varphi$ *then* φ *is a tautology.*

The proof is by induction on the length of the proof. The completeness part is somewhat harder. Before we can enter this proof we shall make a detour. We shall extend the notion of proof somewhat to cover proofs from assumptions.

Definition 3.2.4 *A* ***proof of*** φ ***from*** Δ *is a finite sequence* $\Pi = \langle \delta_i : i < n \rangle$ *of formulae such that (a)* $\delta_{n-1} = \varphi$ *and (b) for all* $i < n$ *either (b1)* δ_i *is an instance of (a0) – (a3) or (b2) there are* $j, k < i$ *such that* $\delta_k = (\delta_j \rightarrow \delta_i)$ *or (b3)* $\delta_i \in \Delta$*. We write* $\Delta \vdash^B \varphi$ *if there is a proof of* φ *from* Δ*.*

To understand this notion of a hypothetical proof, we shall introduce the notion of an **assignment**. It is common to define an assignment to be a function from variables to the set $\{0, 1\}$. Here, we shall give an effectively equivalent definition.

Definition 3.2.5 *An* ***assignment*** *is a maximal subset* A *of*

$$\{\mathtt{X}_{\vec{\alpha}} : \vec{\alpha} \in (\mathtt{0} \cup \mathtt{1})^*\} \cup \{\mathtt{Y}_{\vec{\alpha}} : \vec{\alpha} \in (\mathtt{0} \cup \mathtt{1})^*\}$$

such that for no $\vec{\alpha}$ *both* $\mathtt{X}_{\vec{\alpha}}, \mathtt{Y}_{\vec{\alpha}} \in A$*.*

(So, an assignment is a set of nullary modes.) Each assignment defines a closure under the modes $\mathtt{M}_\perp$ and $\mathtt{M}_\rightarrow$, which we denote by $\Sigma_B(A)$.

Lemma 3.2.6 *Let A be an assignment and φ a well formed formula. Then either $\langle\varphi, P, 0\rangle \in \Sigma_B(A)$ or $\langle\varphi, P, 1\rangle \in \Sigma_B(A)$, but not both.*

The proof is by induction on the length of $\vec{x}$. We say that an assignment A makes a formula φ **true** if $\langle\varphi, P, 1\rangle \in \Sigma_B(A)$.

Definition 3.2.7 *Let Δ be a set of formulae and φ a formula. We say that φ **follows from** (or **is a consequence of**) Δ if for all assignments A: if A makes all formulae of Δ true then it makes φ true as well. In that case we write $\Delta \vDash \varphi$.*

Our aim is to show that the Hilbert Calculus characterizes this notion of consequence:

Theorem 3.2.8 $\Delta \vdash^B \varphi$ *if and only if* $\Delta \vDash \varphi$.

Again, the proof has to be deferred until the matter is sufficiently simplified. Let us first show the following fact, known as the **Deduction Theorem** (**DT**).

Lemma 3.2.9 (Deduction Theorem) $\Delta; \varphi \vdash^B \chi$ *if and only if* $\Delta \vdash^B (\varphi \to \chi)$.

Proof. The direction from right to left is immediate and left to the reader. Now for the other direction, suppose that $\Delta; \varphi \vdash^B \chi$. Then there exists a proof $\Pi = \langle\delta_i : i < n\rangle$ of χ from $\Delta; \varphi$. We shall inductively construct a proof $\Pi' = \langle\delta'_j : j < m\rangle$ of $(\varphi \to \chi)$ from Δ. The construction is as follows. We define Π_i inductively by $\Pi_0 := \varepsilon$,

$$\Pi_{i+1} := \Pi_i^\frown \Sigma_i \ ,$$

where Σ_i, $i < n$, is defined as given below. Furthermore, we will verify inductively that Π_{i+1} is a proof of its last formula, which is $(\varphi \to \delta_i)$. Then $\Pi' := \Pi_n$ will be the desired proof, since $\delta_{n-1} = \chi$. Choose $i < n$. Then either (1) $\delta_i \in \Delta$ or (2) δ is an instance of (a0) – (a3) or (3) $\delta_i = \varphi$ or (4) there are

$j, k < i$ such that $\delta_k = (\delta_j \to \delta_i)$. In the first two cases we put $\Sigma_i := \langle \delta_i, (\delta_i \to (\varphi \to \delta_i)), (\varphi \to \delta_i)\rangle$. In Case (3) we put

$$\begin{aligned}\Sigma_i := \quad & \langle ((\varphi \to ((\varphi \to \varphi) \to \varphi)) \to ((\varphi \to (\varphi \to \varphi)) \to \\ & \qquad\qquad\qquad\qquad\qquad\qquad (\varphi \to \varphi))) \;, \\ & (\varphi \to ((\varphi \to \varphi) \to \varphi)), \\ & ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)), \\ & (\varphi \to (\varphi \to \varphi)), \\ & (\varphi \to \varphi)\rangle\end{aligned}$$

Σ_i is a proof of $(\varphi \to \varphi)$, as is readily checked. Finally, Case (4). There are $j, k < i$ such that $\delta_k = (\delta_j \to \delta_i)$. Then by induction hypothesis $(\varphi \to \delta_j)$ and $(\varphi \to \delta_k) = (\varphi \to (\delta_j \to \delta_i))$ already occur in the proof. Then put

$$\begin{aligned}\Sigma_i := \quad & \langle ((\varphi \to (\delta_j \to \delta_i)) \to ((\varphi \to \delta_j) \to (\varphi \to \delta_i))), \\ & ((\varphi \to \delta_j) \to (\varphi \to \delta_i)), \\ & (\varphi \to \delta_i)\rangle\end{aligned}$$

It is verified that Π_{i+1} is a proof of $(\varphi \to \delta_i)$. ☐

A special variant is the following.

Lemma 3.2.10 (Little Deduction Theorem) *For all Δ and φ: $\Delta \vdash^B \varphi$ if and only if $\Delta; (\varphi \to \bot) \vdash^B \bot$.*

Proof. Assume that $\Delta \vdash^B \varphi$. Then there is a proof Π of φ from Δ and $\Pi^\frown\langle(\varphi \to \bot), \bot\rangle$ is a proof of $\bot$ from $\Delta; (\varphi \to \bot)$. Conversely, assume that $\Delta; (\varphi \to \bot) \vdash^B \bot$. Applying DT we get $\Delta \vdash^B ((\varphi \to \bot) \to \bot)$. Using (a3) we get $\Delta \vdash^B \varphi$. ☐

Proposition 3.2.11 *The following holds.*

1. $\varphi \vdash^B \varphi$.
2. *If $\Delta \subseteq^B \Delta'$ and $\Delta \vdash^B \varphi$ then also $\Delta' \vdash^B \varphi$.*
3. *If $\Delta \vdash^B \varphi$ and $\Gamma; \varphi \vdash^B \chi$ then $\Gamma; \Delta \vdash^B \chi$.*

This is easily verified. Now we are ready for the proof of Theorem 3.2.8. An easy induction on the length of a proof establishes that if $\Delta \vdash^B \varphi$ then also $\Delta \vDash \varphi$. (This is the called the *correctness* of the calculus.) So, the converse implication, which is the *completeness* part needs proof. Assume that $\Delta \nvdash^B \varphi$. We shall show that also $\Delta \nvDash \varphi$. Call a set Σ **consistent** if $\Sigma \nvdash^B \bot$.

Lemma 3.2.12 *1. Let $\Delta; (\varphi \to \chi)$ be consistent. Then either $\Delta; (\varphi \to \bot)$ is consistent or $\Delta; \chi$ is consistent.*

2. Let $\Delta; ((\varphi \to \chi) \to \bot)$ be consistent. Then also $\Delta; \varphi; (\chi \to \bot)$ is consistent.

Proof. Assume that $\Delta; (\varphi \to \bot)$ and $\Delta; \chi$ are inconsistent. Then $\Delta; (\varphi \to \bot) \vdash^B \bot$ and $\Delta; \chi \vdash^B \bot$. By DT we get $\Delta \vdash^B ((\varphi \to \bot) \to \bot)$ and, using (a3), $\Delta \vdash^B \varphi$. Hence $\Delta; (\varphi \to \chi) \vdash^B \varphi$ and so $\Delta; (\varphi \to \chi) \vdash^B \chi$. Because $\Delta; \chi \vdash^B \bot$, we also have $\Delta; (\varphi \to \chi) \vdash^B \bot$, showing that $\Delta; (\varphi \to \chi)$ is inconsistent. Now for the second claim. Assume that $\Delta; \varphi; (\chi \to \bot)$ is inconsistent. Then $\Delta; \varphi; (\chi \to \bot) \vdash^B \bot$. Then $\Delta; \varphi \vdash^B ((\chi \to \bot) \to \bot)$, by applying DT. So, $\Delta; \varphi \vdash^B \chi$, using (a3). Applying DT we get $\Delta \vdash^B (\varphi \to \chi)$. Using (a3) and DT once again it is finally seen that $\Delta; ((\varphi \to \chi) \to \bot)$ is inconsistent. □

Finally, let us return to our proof of the completeness theorem. We assume that $\Delta \nvdash^B \varphi$. We have to find an assignment A such that A makes Δ true but not φ. We may also apply the Little DT and assume that $\Delta; (\varphi \to \bot)$ is consistent and find an assignment that makes this set true. The way to find such an assignment is by applying the so–called downward closure of the set.

Definition 3.2.13 *A set Δ is **downward closed** iff (1) for all $(\varphi \to \chi) \in \Delta$ either $(\varphi \to \bot) \in \Delta$ or $\chi \in \Delta$ and (2) for all $((\varphi \to \chi) \to \bot) \in \Delta$ also $\varphi, (\chi \to \bot) \in \Delta$.*

Now, by Lemma 3.2.12 every consistent set has a consistent closure Δ^*. (It is an exercise for the diligent reader to show this. In fact, for infinite sets a little work is needed here, but we really need this only for finite sets.) Define the following assignment.

$$\begin{array}{rcl} A & := & \{\langle \mathtt{p}\vec{\alpha}, P, 1\rangle : (\mathtt{p}\vec{\alpha} \to \bot) \text{ does not occur in } \Delta^*\} \\ & & \cup\{\langle \mathtt{p}\vec{\alpha}, P, 0\rangle : (\mathtt{p}\vec{\alpha} \to \bot) \text{ does occur in } \Delta^*\} \end{array}$$

It is shown by induction on the formulae of Δ^* that the so defined assignment makes every formula of Δ^* true. Using the correspondence between syntactic derivability and semantic consequence we immediately derive the following.

Theorem 3.2.14 (Compactness Theorem) *Let φ be a formula and Δ a set of formulae such that $\Delta \vDash \varphi$. Then there exists a finite set $\Delta' \subseteq \Delta$ such that $\Delta' \vDash \varphi$.*

Proof. Suppose that $\Delta \vDash \varphi$. Then $\Delta \vdash^B \varphi$. Hence there exists a proof of φ from Δ. Let Δ' be the set of those formulae in Δ that occur in that proof. Δ' is finite. Clearly, this proof is a proof of φ from Δ', showing $\Delta' \vdash^B \varphi$. Hence $\Delta' \vDash \varphi$. ☐

Usually, one has more connectives than just $\bot$ and $\to$. Now, two effectively equivalent strategies suggest themselves, and they are used whenever convenient. The first is to introduce a new connective as an abbreviation. So, we might define (for well formed formulae)

$$\begin{array}{rcl} \neg\varphi & := & \varphi \to \bot \\ \varphi \vee \chi & := & (\varphi \to \bot) \to \chi \\ \varphi \wedge \chi & := & (\varphi \to (\chi \to \bot) \to \bot \end{array}$$

After the introduction of these abbreviations, everything is the same as before, because we have not changed the language, only our way of referring to its strings. However, we may also change the language by expanding the alphabet. In the cases at hand we will add the following unary and binary modes (depending on which symbol is to be added):

$$\begin{array}{rcl} \mathtt{M}_{\neg}(\langle \vec{x}, P, \eta\rangle) & := & \langle (\neg\vec{x}), P, -\eta\rangle \\ \mathtt{M}_{\vee}(\langle \vec{x}, P, \eta\rangle, \langle \vec{y}, P, \theta\rangle) & := & \langle (\vec{x} \vee \vec{y}), P, \eta \cup \theta\rangle \\ \mathtt{M}_{\wedge}(\langle \vec{x}, P, \eta\rangle, \langle \vec{y}, P, \theta\rangle) & := & \langle (\vec{x} \wedge \vec{y}), P, \eta \cap \theta\rangle \end{array}$$

$\cup$	0	1
0	0	1
1	1	1

$\cap$	0	1
0	0	0
1	0	1

	$-$
0	1
1	0

We then need to add axioms for the new connectives. These are for $\wedge$:

$$(\varphi \to (\psi \to (\varphi \wedge \psi))), \quad (\varphi \to (\psi \to (\psi \wedge \varphi))),$$
$$((\varphi \wedge \psi) \to \varphi), \quad ((\varphi \wedge \psi) \to \psi)$$

For $\vee$ we need these postulates:

$$(\varphi \to (\varphi \vee \psi)), \quad (\psi \to (\varphi \vee \psi)),$$
$$(((\varphi \vee \psi) \to \chi) \to (\varphi \to \chi)), \quad (((\varphi \vee \psi) \to \chi) \to (\psi \to \chi))$$

And finally for $\neg$ the following are needed:

$$((\varphi \to \psi) \to ((\neg\psi) \to (\neg\varphi))), \quad (\varphi \to (\neg(\neg\varphi)))$$

Notice that in defining the axioms we have made use of $\to$ alone. If we also add this formula as an axiom

$$((\neg(\neg\varphi)) \to \varphi)$$

we get boolean logic, not intuitionistic logic. (Hence it is *not* added.)

If we eliminate the connective $\perp$ and define $\Delta \vdash \varphi$ as before (eliminating the axioms (a2) and (a3), however) we get what is known as **intuitionistic logic**. The semantics of intuitionistic logic is too complicated to be explained here, so we just use the Hilbert Calculus to introduce it. So, what we claim in fact is that with only (a0) and (a1) it is not possible to prove all formulae of Taut_B that use only $\to$. A case in point is the formula

$$(((\varphi \to \chi) \to \varphi) \to \varphi)$$

which is known as *Peirce's Formula.* Together with Peirce's Formula, (a0) and (a1) axiomatize the full set of tautologies of boolean logic in $\to$. The calculus based on (a0) and (a1) is called **H** and we write $\Delta \vdash^H \chi$ to say that there is a proof in the Hilbert Calculus of χ from Δ using (a0) and (a1).

Now we take a look again at the matter. Rather than axiomatizing the set of tautologies we can also axiomatize the deducibility relation itself. This idea goes back to Gerhard Gentzen, who used it among other to show the consistency of arithmetic (which we shall not do here). For simplicity, we stay with the language with only the arrow. We shall axiomatize the derivability of intuitionistic logic. The statements that we are deriving now have the form '$\Delta \vdash \varphi$' and are called **sequents**. Δ is called the **antecedent** and φ the **succedent** of that sequent. Some of them are axioms. These are

$$(\mathrm{ax}) \quad \varphi \vdash \varphi$$

Then there are the following rules of introduction of connectives:

$$(\mathbf{I}\to) \quad \frac{\Delta; \varphi \vdash \chi}{\Delta \vdash (\varphi \to \chi)} \qquad (\to\mathbf{I}) \quad \frac{\Delta \vdash \varphi \qquad \Delta; \psi \vdash \chi}{\Delta; (\varphi \to \psi) \vdash \chi}$$

Notice that these rules introduce occurrences of the arrow. The rule ($\mathbf{I}\to$) introduces an occurrence on the right hand side of $\vdash$, while ($\to\mathbf{I}$) an occurrence on the left hand side. (The names of the rules are chosen accordingly.) Further, there are the following so called rules of inference:

$$(\mathrm{cut}) \quad \frac{\Delta \vdash \varphi \qquad \Theta; \varphi \vdash \chi}{\Delta; \Theta \vdash \chi} \qquad (\mathrm{mon}) \quad \frac{\Delta \vdash \varphi}{\Delta; \Theta \vdash \varphi}$$

Sequents above the lines are called **premisses**, formulae below the lines are called the **conclusions**. Further, the formulae that are introduced by the rules ($\to\mathbf{I}$) and ($\mathbf{I}\to$) are called **main formulae**, and the formula φ in (cut) the **cut–formula**. Let us call this the **Gentzen Calculus**. It is denoted by $\mathcal{G}$.

Definition 3.2.15 *Let $\Delta \vdash \varphi$ be a sequent. A **(sequent) proof of length** n of $\Delta \vdash \varphi$ in $\mathcal{G}$ is a sequence $\Pi = \langle \Sigma_i \vdash \chi_i : i < n+1 \rangle$ such that (a) $\Sigma_n = \Delta$, $\chi_n = \varphi$, (b) for all $i < n+1$ either (ba) $\Sigma_i \vdash \chi_i$ is an axiom or (bb) $\Sigma_i \vdash \chi_i$ follows from some earlier sequents by application of a rule of $\mathcal{G}$.*

It remains to say what it means that a sequent follows from some other sequents by application of a rule. This, however, is straightforward. For example, $\Delta \vdash (\varphi \to \chi)$ follows from the earlier sequents by application of the rule $(\mathbf{I} \to)$ if among the earlier sequents we find the sequent $\Delta; \varphi \vdash \chi$. We shall define also a different notion of proof, which is based on trees rather than sequences. In doing so, we shall also formulate a somewhat more abstract notion of a calculus.

Definition 3.2.16 *A **finitary rule** is a pair $\rho = \langle M, \mathfrak{S}\rangle$, where M is a finite set of sequents and $\mathfrak{S}$ a single sequent. (These rules are written down using lower case Greek letters as schematic variables for formulae and upper case Greek letters as schematic variables for sets of formulae.) A **sequent calculus** $\mathcal{S}$ is a set of finitary rules. A $\mathcal{S}$–**proof tree** is a triple $\mathbb{T} = \langle T, \succ, \ell\rangle$ such that $\langle T, \prec\rangle$ is a tree and for all x: if $\{y_i : i < n\}$ are the daughters of T, $\langle\{\ell(y_i) : i < n\}, \ell(x)\rangle$ is an instance of a rule of $\mathcal{S}$. If r is the root of $\mathbb{T}$, we say that $\mathbb{T}$ **proves** $\ell(r)$ **in** $\mathcal{S}$. We write*

$$\overset{\mathcal{S}}{\rightsquigarrow} \Delta \vdash \varphi$$

to say that the sequent $\Delta \vdash \varphi$ has a proof in $\mathcal{S}$.

We start with the only rule for $\bot$, which actually is an axiom.

$$(\bot\mathbf{I}) \quad \bot \vdash \varphi$$

For negation we have these rules.

$$(\neg\mathbf{I}) \quad \frac{\Delta \vdash \varphi}{\Delta; (\neg\varphi) \vdash \bot} \qquad (\mathbf{I}\neg) \quad \frac{\Delta; \varphi \vdash \bot}{\Delta \vdash (\neg\varphi)}$$

The following are the rules for conjunction.

$$(\wedge\mathbf{I}) \quad \frac{\Delta; \varphi; \psi \vdash \chi}{\Delta; (\varphi \wedge \psi) \vdash \chi} \qquad (\mathbf{I}\wedge) \quad \frac{\Delta \vdash \varphi \qquad \Delta \vdash \psi}{\Delta \vdash (\varphi \wedge \psi)}$$

Finally, these are the rules for $\vee$.

$$(\vee\mathbf{I}) \quad \frac{\Delta; \varphi \vdash \chi \qquad \Delta; \psi \vdash \chi}{\Delta; (\varphi \wedge \psi) \vdash \chi}$$

$$(\mathbf{I}_1\vee) \quad \frac{\Delta \vdash \varphi}{\Delta \vdash (\varphi \vee \psi)} \qquad (\mathbf{I}_2\vee) \quad \frac{\Delta \vdash \psi}{\Delta \vdash (\varphi \vee \psi)}$$

Let us return to the calculus $\mathcal{G}$. We shall first of all show that we can weaken the rule system without changing the set of derivable sequents. Notice that the following is a proof tree.

$$\cfrac{\cfrac{\varphi \vdash \varphi \qquad \psi \vdash \psi}{(\varphi \to \psi); \varphi \vdash \psi}}{(\varphi \to \psi) \vdash (\varphi \to \psi)}$$

This shows us that in place of the rule (ax) we may actually use a restricted rule, where we have only $p \vdash p$, p a variable. Call such an instance of (ax) **primitive**. This fact may be used for the following theorem.

Lemma 3.2.17 $\stackrel{\mathcal{G}}{\leadsto} \Delta \vdash (\varphi \to \chi)$ *if and only if* $\stackrel{\mathcal{G}}{\leadsto} \Delta; \varphi \vdash \chi$.

Proof. From right to left follows using the rule ($\mathbf{I}\!\to$). Let us prove the other direction. We know that there exists a proof tree of $\Delta \vdash (\varphi \to \chi)$ from primitive axioms. Now we trace backwards the occurrence of $(\varphi \to \chi)$ in the tree from the root upwards. Obviously, since the formula has not been introduced by (ax), it must have been introduced by the rule ($\mathbf{I}\!\to$). Let x be the node where the formula is introduced. Then we remove x from the tree, thereby also removing that instance of ($\mathbf{I}\!\to$). Going down from x, we have to repair our proof as follows. Suppose that at $y < x$ we have an instance of (mon). Then instead of the proof part to the left we use the one to the right.

$$\frac{\Sigma \vdash (\varphi \to \chi)}{\Sigma; \Theta \vdash (\varphi \to \chi)} \qquad \frac{\Sigma; \varphi \vdash \chi}{\Sigma; \Theta; \varphi \vdash \chi}$$

Suppose that we have an instance of (cut). Then first of all the formula on which the cut is performed may be any formula, but our specified occurrence of $(\varphi \to \chi)$ is the one that is on the right of the target sequent. Then in place of the proof part on the left we use the one on the right.

$$\dfrac{\Delta \vdash \psi \qquad \Theta; \psi \vdash (\varphi \to \chi)}{\Delta; \Theta \vdash (\varphi \to \chi)} \qquad\qquad \dfrac{\Delta \vdash \psi \qquad \Theta; \varphi; \psi \vdash \chi}{\Delta; \Theta; \varphi \vdash \chi}$$

Now suppose that we have an instance of ($\to$**I**). Then this instance must be as shown to the left. We replace it by the one on the right.

$$\dfrac{\Delta \vdash \tau \qquad \Delta; \psi \vdash (\varphi \to \chi)}{\Delta; (\tau \to \psi) \vdash (\varphi \to \chi)} \qquad\qquad \dfrac{\Delta \vdash \tau \qquad \Delta; \varphi; \psi \vdash \chi}{\Delta; (\tau \to \psi); \varphi \vdash \chi}$$

The rule ($\to$**I**) does not occur below x, as is easily seen. This concludes the replacement. It is verified that after performing these replacements, we obtain a proof tree for $\Delta; \varphi \vdash \chi$. ☐

Theorem 3.2.18 $\Delta \vdash^H \varphi$ *if and only if* $\overset{\mathcal{G}}{\leadsto} \Delta \vdash \varphi$.

Proof. Suppose that $\Delta \vdash^H \varphi$. By induction on the length of the proof we shall show that $\overset{\mathcal{G}}{\leadsto} \Delta \vdash \varphi$. Using DT we may restrict ourselves to $\Delta = \varnothing$. First, we shall show that (a0) and (a1) can be derived. (a0) is derived as follows.

$$\dfrac{\dfrac{\dfrac{\varphi \vdash \varphi}{\varphi; \psi \vdash \varphi}}{\varphi \vdash (\psi \to \varphi)}}{\vdash (\varphi \to (\psi \to \varphi))}$$

For (a1) we need a little more work.

$$\dfrac{\varphi \vdash \varphi \quad \dfrac{\varphi \vdash \varphi \quad \dfrac{\psi \vdash \psi \qquad \chi \vdash \chi}{\psi; (\psi \to \chi) \vdash \chi}}{\varphi; (\varphi \to \psi); (\psi \to \chi) \vdash \chi}}{(\varphi \to (\psi \to \chi)); (\varphi \to \psi); \varphi \vdash \chi}$$

If we apply (**I**$\to$) three times we get (a1). Next we have to show that if $\overset{\mathcal{G}}{\leadsto} \varnothing \vdash \varphi$ and $\overset{\mathcal{G}}{\leadsto} \varnothing \vdash (\chi \to \varphi)$ then $\overset{\mathcal{G}}{\leadsto} \varnothing \vdash \chi$. By DT, we also have $\overset{\mathcal{G}}{\leadsto} \varphi \vdash \chi$ and then a single application of (cut) yields the desired conclusion. This proves that $\overset{\mathcal{G}}{\leadsto} \varnothing \vdash \varphi$. Now, conversely, we have to show that $\overset{\mathcal{G}}{\leadsto} \Delta \vdash \varphi$ implies that $\Delta \vdash^H \varphi$. This is shown by induction on the height of the nodes in the proof tree. If it is 1, we have an axiom: however, $\varphi \vdash^H \varphi$ clearly holds. Now suppose the claim is true for all nodes of depth $< i$ and let x be of depth i. Then x is the result of applying one of the four rules. ($\to$**I**). By induction hypothesis, $\Delta \vdash^H \varphi$ and $\Delta; \psi \vdash^H \chi$. We need to show that $\Delta; (\varphi \to \psi) \vdash \chi$. Simply let Π_1 be a proof of φ from Δ, Π_2 a proof of χ from $\Delta; \psi$. Then Π_3 is a proof of χ from $\Delta; (\varphi \to \psi)$.

$$\Pi_3 := \Pi_1^\frown\langle(\varphi \to \psi), \psi\rangle^\frown\Pi_2$$

(**I**$\to$). This is straightfoward from DT. (cut). Suppose that Π_1 is a proof of φ from Δ and Π_2 a proof of χ from $\Theta; \varphi$. Then $\Pi_1^\frown\Pi_2$ is a proof of χ from $\Delta; \varphi$, as is easily seen. (mon). This follows from Proposition 3.2.11. ☐

Call a rule ρ **admissible** for a calculus $\mathcal{S}$ if any sequent $\Delta \vdash \varphi$ that is derivable in $\mathcal{S} + \rho$ is also derivable in $\mathcal{S}$. Conversely, if ρ is admissible in $\mathcal{S}$, we say that ρ is **eliminable from** $\mathcal{S} + \rho$. We shall show that (cut) is eliminable from $\mathcal{G}$, so that it can be omitted without losing derivable sequents. As cut–elimination will play a big role in the sequel, the reader is asked to watch the procedure carefully.

Theorem 3.2.19 *(cut) is eliminable from* $\mathcal{G}$.

Proof. Recall that (cut) is the following rule.

$$\text{(cut)} \quad \frac{\Delta \vdash \varphi \qquad \Theta; \varphi \vdash \chi}{\Delta; \Theta \vdash \chi}$$

Two measures are introduced. The **degree** of an instance of cut is the total length of the conclusion plus the length of φ. The **weight** of a cut is 2^d, where d is the degree of the cut. The **cut–weight** of a proof tree $\mathbb{T}$ is the sum over all degrees of occurrences of cuts (= instances of (cut)) in it. Obviously, the cut–weight of a proof tree is zero if and only if there are no cuts in it. We shall now present a procedure that operates on proof trees in such a way that it reduces the cut–weight of every given tree. This procedure is as follows. Let $\mathbb{T}$ be given, and let x be a node carrying the conclusion of an instance of (cut). We shall assume that above x no instances of (cut) exist. (Obviously, x exists if there are cuts in $\mathbb{T}$.) x has two mothers, y_1 and y_2. Case (1) Suppose that y_1 is a leaf. Then we have $\ell(y_1) = \varphi \vdash \varphi$, $\ell(y_2) = \Theta; \varphi \vdash \chi$ and $\ell(x) = \Theta; \varphi \vdash \chi$. In this case, we may simply skip the application of cut by dropping the nodes x and y_1. This reduces the degree of the cut by $2 \cdot |\varphi|$, since this application of (cut) has been eliminated without trace. Case (2). Suppose that y_2 is a leaf. Then $\ell(y_2) = \chi \vdash \chi$, $\ell(y_1) = \Delta \vdash \varphi$, whence $\varphi = \chi$ and $\ell(x) = \Delta \vdash \varphi = \ell(y_1)$. Eliminate x and y_2. This reduces the degree of the cut by $2 \cdot |\varphi|$. Case (3). Suppose that y_1 has been obtained by application of (mon). Then the proof is as shown on the left.

$$\dfrac{\dfrac{\Delta \vdash \varphi}{\Delta; \Delta' \vdash \varphi} \quad \Theta; \varphi \vdash \chi}{\Delta; \Delta'; \Theta \vdash \chi} \qquad\qquad \dfrac{\dfrac{\Delta \vdash \varphi \qquad \Theta; \varphi \vdash \chi}{\Delta; \Theta \vdash \chi}}{\Delta; \Delta'; \Theta \vdash \chi}$$

We replace the local tree by the one on the right. The cut weight is reduced by $|\Delta'|$. (Clearly, if $\Delta' = \varnothing$, we have not properly reduced the cut weight. However, we may either exclude such instances of (mon) (without loss of provable sequents) or add to the degree of a cut also the depth of the node in the proof tree where it occurs. Then that new number is decreased by the above operation.) Case (4). $\ell(y_2)$ has been obtained by application of (mon). This is similar to the previous case. Case (5). $\ell(y_1)$ has been obtained by ($\to$**I**). Then the main formula is not the cut formula:

$$\dfrac{\dfrac{\Delta \vdash \rho \quad \Delta; \tau \vdash \varphi}{\Delta; (\rho \to \tau) \vdash \varphi} \qquad \Theta; \varphi \vdash \chi}{\Delta; \Theta; (\rho \to \tau) \vdash \chi}$$

and the cut can be rearranged as follows.

$$\dfrac{\dfrac{\Delta \vdash \rho}{\Delta; \Theta \vdash \rho} \qquad \dfrac{\Delta; \tau \vdash \varphi \quad \Theta; \varphi \vdash \chi}{\Delta; \Theta; \tau \vdash \chi}}{\Delta; \Theta; (\rho \to \tau) \vdash \chi}$$

Here, the degree of the cut is reduced by $|(\rho \to \tau)| - |\tau| > 0$. Case (6). $\ell(y_2)$ has been obtained by ($\mathbf{I}{\to}$). Then $\chi = (\rho \to \tau)$ for some ρ and τ. We replace the left hand proof part by the right hand part, and the degree is reduced by $|(\rho \to \tau)| - |\tau| > 0$.

$$\dfrac{\Delta \vdash \varphi \qquad \dfrac{\Theta; \varphi; \rho \vdash \tau}{\Theta; \varphi \vdash (\rho \to \tau)}}{\Delta; \Theta \vdash (\rho \to \tau)} \qquad\qquad \dfrac{\dfrac{\Delta \vdash \varphi \qquad \Theta; \rho; \varphi \vdash \tau}{\Delta; \Theta; \rho \vdash \tau}}{\Delta; \Theta \vdash (\rho \to \tau)}$$

Case (7). $\ell(y_1)$ has been introduced by ($\mathbf{I}{\to}$). This case is the most tricky one, since the main formula is the cut formula. Here, we cannot simply permute the cut unless $\ell(y_2)$ is the result of applying (${\to}\mathbf{I}$). In this case we proceed as follows. $\varphi = (\rho \to \tau)$ for some ρ and τ. The local proof is as follows.

$$\dfrac{\dfrac{\Delta; \rho \vdash \tau}{\Delta \vdash (\rho \to \tau)} \qquad \dfrac{\Theta \vdash \rho \quad \Theta; \tau \vdash \chi}{\Theta; (\rho \to \tau) \vdash \chi}}{\Delta; \Theta \vdash \chi}$$

This is rearranged in the following way.

$$\dfrac{\dfrac{\Delta; \rho \vdash \tau \qquad \Theta \vdash \rho}{\Delta; \Theta \vdash \tau} \qquad \dfrac{\Theta; \tau \vdash \chi}{\Delta; \Theta; \tau \vdash \chi}}{\Delta; \Theta \vdash \chi}$$

This operation eliminates the cut in favour of two cuts. The overall degree of these cuts may be increased, but the weight has been decreased. Let $d := |\Delta; \Theta|$, $p := |(\rho \to \tau)|$. Then the first cut has weight $2^{d+p+|\chi|}$. The two other cuts have weight

$$2^{d+|\rho|+|\tau|} + 2^{d+|\tau|+|\chi|} \leq 2^{d+|\rho|+|\tau|+|\chi|} < 2^{d+p+|\chi|}$$

since $p > |\rho| + |\tau| > 0$. (Notice that $2^{a+b} + 2^{a+c} = 2^a \cdot (2^c + 2^d) \leq 2^a \cdot 2^{c+d} = 2^{a+c+d}$ if $c, d > 0$.) Now, what if Case (7) does not obtain? Then we are in one of the previous cases, which we have successfully dealt with. So, in fact we may assume that $\ell(y_2)$ has been obtained by this rule. Moreover, in each case we managed to decrease the cut–weight. This concludes the proof. □

Before we conclude this section we shall mention another deductive calculus, called **Natural Deduction**. It uses proof trees, but is based on the Deduction Theorem. First of all notice that we can write Hilbert–Style proofs also in tree format. Then the leaves of the proof tree are axioms, or assumptions, and the only rule we are allowed to use is **Modus Ponens**.

$$\text{(MP)} \quad \frac{(\varphi \to \psi) \qquad \varphi}{\psi}$$

This, however, is a mere reformulation of the previous calculus. The idea behind natural deduction is that we view Modus Ponens as a rule to *eliminate* the arrow, while we add another rule that allows to introduce it. It is as follows.

$$(\mathbf{I}{\to}) \quad \frac{\psi}{(\varphi \to \psi)}$$

However, when this rule is used, the formula φ may be eliminated from the assumptions. Let us see how this goes. Let x be a node. Let us call the set $A(x) := \{\langle y, \ell(y)\rangle : y > x, y \text{ leaf}\}$ the set of **assumptions of** x. If $(\mathbf{I}{\to})$ is used to introduce $(\varphi \to \psi)$, any number of assumptions of x that have the form $\langle y, \varphi\rangle$ may be retracted. In order to know what assumption has been effectively retracted, we checkmark the assumption by a superscript $(\varphi^{\surd})$. Here are the standard rules for the other connectives. The fact that the assumption φ is or may be removed is annotated as follows:

$$(\mathbf{I}{\to}) \quad \frac{\begin{matrix}[\varphi] \\ \vdots \\ \psi\end{matrix}}{(\varphi \to \psi)}$$

Here, $[\varphi]$ means that any number of assumptions of the form φ above the node carrying φ may be checkmarked when using the rule. (So, it does *not* mean that it requires these formulae to be assumptions.) First, conjunction.

$$(\mathbf{I}\wedge)\quad \frac{\varphi \qquad \psi}{(\varphi \wedge \psi)} \qquad (\mathbf{E}_1\wedge)\quad \frac{(\varphi \wedge \psi)}{\varphi} \qquad (\mathbf{E}_2\wedge)\quad \frac{(\varphi \wedge \psi)}{\psi}$$

The next is $\bot$:

$$(\mathbf{E}\bot)\quad \frac{\bot}{\varphi}$$

For negation we need some administration of the checkmark.

$$(\mathbf{I}\neg)\quad \frac{\begin{array}{c}[\varphi]\\ \vdots\\ \bot\end{array}}{(\neg\varphi)} \qquad (\mathbf{E}\neg)\quad \frac{\varphi \qquad (\neg\varphi)}{\bot}$$

So, using the rule $(\mathbf{I}\neg)$ any number of assumptions of the form φ may be checkmarked. Disjunction is even more complex.

$$(\mathbf{I}_1\vee)\quad \frac{\varphi}{(\varphi \vee \psi)} \qquad (\mathbf{I}_2\vee)\quad \frac{\psi}{(\varphi \vee \psi)}$$

$$(\mathbf{E}\vee)\quad \frac{(\varphi \vee \psi) \quad \begin{array}{c}[\varphi]\\ \vdots\\ \chi\end{array} \quad \begin{array}{c}[\psi]\\ \vdots\\ \chi\end{array}}{\chi}$$

In the last rule, we have three assumptions. As we have indicated, whenever it is used, we may checkmark any number of assumptions of the form φ in the second subtree and any number of assumptions of the form ψ in the third.

We shall give a characterization of natural deduction trees. A **finitary rule** is a pair $\rho = \langle\{\chi_i[A_i] : i < n\}, \varphi\rangle$, where for $i < n$, χ_i is a formula, A_i a finite set of formulae and φ a single formula.

A **natural deduction calculus** $\mathfrak{N}$ is a set of finitary rules. A **proof tree for** $\mathfrak{N}$ is a quadruple $\mathbb{T} = \langle T, \succ, \ell, \mathcal{C}\rangle$ such that $\langle T, \prec\rangle$ is a tree, $\mathcal{C} \subseteq T$ a set of leaves and $\mathbb{T}$ is derived in the following way. (Think of $\mathcal{C}$ as the set of leaves carrying discharged assumptions.)

* $\mathbb{T} = \langle \{x\}, \varnothing, \ell, \varnothing\rangle$, where $\ell : x \mapsto \varphi$.
* There is a rule $\langle\{\chi_i[A_i] : i < n\}, \gamma\rangle$, and $\mathbb{T}$ is formed from trees $\mathbb{S}_i$, $i < n$, with roots s_i, by adding a new root node r, such that $\ell_{\mathbb{S}_i}(y_i) = \chi_i$, $i < n$, $\ell_{\mathbb{T}}(x) = \gamma$. Further, $\mathcal{C}_{\mathbb{T}} = \bigcup_{i<n} \mathcal{C}_{\mathbb{S}_i} \cup \bigcup_{i<n} N_i$, where N_i is a set of leaves of $\mathbb{S}_i$ such that for all $i < n$ and all $x \in N_i$: $\ell_{\mathbb{S}_i}(x) \in A_i$.

(Notice that the second case includes $n = 0$, in which case we $\mathbb{T} = \langle\{x\}, \varnothing, \ell, \{x\}\rangle$ where $\ell(x)$ is simply an axiom.) We say that $\mathbb{T}$ **proves** $\ell(r)$ **in** $\mathfrak{N}$ from $\{\ell(x) : x \text{ leaf}, x \notin \mathcal{C}\}$. Here now is a proof tree ending in (a0).

$$\dfrac{\dfrac{\varphi^{\checkmark}}{(\psi \to \varphi)}}{(\varphi \to (\psi \to \varphi))}$$

Further, here is a proof tree ending in (a1).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\varphi \to (\psi \to \chi))^{\checkmark} \qquad \varphi^{\checkmark}}{(\psi \to \chi)} \quad \dfrac{(\varphi \to \psi)^{\checkmark} \qquad \varphi^{\checkmark}}{\psi}}{\chi}}{(\varphi \to \chi)}}{((\varphi \to \psi) \to (\varphi \to \chi))}}{((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)))}$$

A formula depends on all its assumptions that have not been retracted in the following sense.

Lemma 3.2.20 *Let $\mathbb{T}$ be a natural deduction tree with root x. Let Δ be the set of all formulae ψ such that $\langle y, \psi\rangle$ is a non retracted assumption of x and let $\varphi := \ell(x)$. Then $\Delta \vdash^H \varphi$.*

Proof. By induction on the derivation of the proof tree. □

The converse also holds. If $\Delta \vdash^H \varphi$ then there is a natural deduction proof for φ with Δ the set of unretracted assumptions.

Exercise 88. Show (a) $(\varphi \to (\psi \to \chi)) \vdash^B (\psi \to (\varphi \to \chi))$ and (b) $(\varphi \wedge \psi) \vdash^{H'} (\psi \wedge \varphi)$, where H' is H with the axioms for $\wedge$ added.

Exercise 89. Show that a set Σ is inconsistent if and only if for every φ: $\Sigma \vdash^B \varphi$.

Exercise 90. Show that a Hilbert–style calculus satisfies DT for $\to$ if and only if the formulae (a0) and (a1) are derivable in it. (So, if we add, for example, the connectives $\neg$, $\vee$ and $\wedge$ together with the corresponding axioms, DT remains valid.)

Exercise 91. Define $\varphi \approx \psi$ by $\varphi \vdash^H \psi$ and $\psi \vdash^H \varphi$. Show that if $\varphi \approx \psi$ then (a) for all Δ and χ: $\Delta; \varphi \vdash^H \chi$ if and only if $\Delta; \psi \vdash^H \chi$, and (b) for all Δ: $\Delta \vdash^H \varphi$ if and only if $\Delta \vdash^H \psi$.

Exercise 92. Let us call I the Hilbert–Calculus for $\to$, $\perp$, $\neg$, $\vee$ and $\wedge$. Further, call the Gentzen–Calculus for these connectives $\mathfrak{I}$. Show that $\Delta \vdash^I \varphi$ if and only if $\overset{\mathfrak{I}}{\leadsto} \Delta \vdash \varphi$.

Exercise 93. Show the following claim: *If $\Delta \vdash^H \varphi$ then there is a natural deduction proof for φ with Δ the set of unretracted assumptions.*

Exercise 94. Show that the rule of *Modus Tollens* is admissible in the natural deduction calculus defined above (with added negation).

$$\text{Modus Tollens:} \quad \frac{(\varphi \to \psi) \qquad (\neg\psi)}{(\neg\varphi)}$$

3.3 Basics of λ–Calculus and Combinatory Logic

There is a fundamental difference between a term and a function. The *term* $x^2 + 2xy$ is something that has a concrete value if x and y have a concrete value. For example, if x has value 5 and y has value 2 then $x^2 + 2xy = 25 + 20 = 45$. However, the function $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} : \langle x, y\rangle \mapsto x^2 + 2xy$ does not need any values for x and y. It only needs a pair of *numbers* and yields a value. That we have used variables to define f is of no concern here. We would have obtained the same function had we written $f : \langle x, u\rangle \mapsto x^2+2xu$. However, the term x^2+2xu is different from the term $x^2 + 2xy$. For if u has value 3 while x has value 5 and y value 2, then $x^2 + 2xu = 25 + 30 = 55$, while $x^2 + 2xy = 45$. To accommodate for this difference λ–calculus has been developed. The λ–calculus allows to form functions from terms. In the case above we may write f as

$$f := \lambda xy.x^2 + 2xy$$

This expression names a function f and says at the same time what it does. The prefix 'λxy' means that we are dealing with a function from pairs $\langle m, n\rangle$ and that the function assigns every such pair the value $m^2 + 2mn$. This is the same as what we have expressed with $\langle x, y\rangle \mapsto x^2 + 2xy$. Now we can also define the following functions.

$$\lambda x.\lambda y.x^2 + 2xy, \qquad \lambda y.\lambda x.x^2 + 2xy$$

The first is a function which assigns to every number m the function $\lambda y.m^2 + 2my$; the latter yields the value $m^2 + 2mn$ for every n. The second is a function which gives for every m the function $\lambda x.x^2 + 2xm$; this in turn yields $n^2 + 2nm$ for every n. Since in general $m^2 + 2mn \neq n^2 + 2nm$, these two functions are different.

In λ–calculus one usually does not make use of the simultaneous abstraction of several variables, so one only allows prefixes of the form 'λx', not those of the form 'λxy'. This we shall also do here. We shall give a general definition of λ–terms. Anyhow, by introducing pairing and projection simultaneous abstraction can be defined. The alphabet consists of the set F of function symbols, λ, the variables $V := \{\mathtt{x}_i : i \in \omega\}$ the brackets $\mathtt{(}$, $\mathtt{)}$ and the period '$\mathtt{.}$'.

Definition 3.3.1 *The set of λ–**terms over the signature** Ω, the set of λ–Ω–**terms** for short, is the smallest set $\mathrm{Tm}_{\lambda\Omega}$ for which the following holds:*

1. *Every Ω–term is in $\mathrm{Tm}_{\lambda\Omega}$.*
2. *If $M, N \in \mathrm{Tm}_{\lambda\Omega}$ then also $\mathtt{(}MN\mathtt{)} \in \mathrm{Tm}_{\lambda\Omega}$.*
3. *If $M \in \mathrm{Tm}_{\lambda\Omega}$ and x is a variable then $\mathtt{(}\lambda x\mathtt{.}M\mathtt{)} \in \mathrm{Tm}_{\lambda\Omega}$.*

*If the signature is empty or clear from the context we shall simply speak of λ–**terms**.*

Since we do not write the operator, there is no Polish Notation. Therefore we use the brackets $\mathtt{(}$ and $\mathtt{)}$. We agree now that x, y and z and so on are metavariables for elements of V. Furthermore, upper case Roman letters like M, N, X, Y, are metavariables for λ–terms. In the books one usually takes $F := \varnothing$. The interesting thing about the λ–calculus is not the basic functions but the mechanics of abstraction. If $F = \varnothing$, we speak of **pure λ–terms**. It is customary to omit the brackets if the term is bracketed to the left. Hence $MNOP$ is short for $\mathtt{(((}MN\mathtt{)}O\mathtt{)}P\mathtt{)}$ and $\lambda x.MN$ short for $\mathtt{((}\lambda x\mathtt{.(}MN\mathtt{))}$ (and distinct from $\mathtt{(}\lambda x.M\mathtt{)}N\mathtt{)}$). However, this abbreviation has to be used with care since the brackets are symbols of the language. Hence $\mathtt{x}_0\mathtt{x}_0\mathtt{x}_0$ is not a string of the language but only a shorthand for $\mathtt{((x}_0\mathtt{x}_0\mathtt{)x}_0\mathtt{)}$, a difference that we shall ignore after a while. Likewise, outer brackets are often omitted and brackets are not stacked when several λ–prefixes appear. Notice that $\mathtt{(x}_0\mathtt{x}_0\mathtt{)}$ is a term. It denotes the application of $\mathtt{x}_0$ to itself. *Occurrences* of a string $\vec{x}$ in a string $\vec{y}$ we have defined as contexts

$\langle\vec{u},\vec{v}\rangle$ where $\vec{u}\vec{x}\vec{v}=\vec{y}$. Ω–terms are thought to be written down in Polish Notation.

Definition 3.3.2 *Let x be a variable. We define the set of* ***occurrences of*** *x in a λ–term inductively as follows.*

1. *If M is an Ω–term then the set of occurrences of x in M is the set of occurrences of the variable x in the term M.*

2. *The set of occurrences of x in (MN) is the union of the set of pairs $\langle(\vec{u},\vec{v}N)\rangle$, where $\langle\vec{u},\vec{v}\rangle$ is an occurrence of x in M and the set of pairs $\langle(M\vec{u},\vec{v})\rangle$, where $\langle\vec{u},\vec{v}\rangle$ is an occurrence of x in N.*

3. *The set of occurrences of x in $(\lambda x.M)$ is the set of all $\langle(\lambda x.\vec{u},\vec{v})\rangle$, where $\langle\vec{u},\vec{v}\rangle$ is an occurrence of x in M.*

So notice that — technically speaking — the occurrence of x in the λ–prefix of $(\lambda x.M)$ is not counted as an occurrence of x. Hence $\mathtt{x}_0$ does not occur in $(\lambda\mathtt{x}_0.\mathtt{x}_1)$ as a λ–term although it does occur in it as a string!

Definition 3.3.3 *Let M be a λ–term and x a variable. x occurs* ***free in*** *M in C if C is not inside a term of the form $(\lambda x.Y)$ for some N. If x does not occur free in C it is said to occur* ***bound in*** *M* ***at*** *C. A λ–term is called* ***closed*** *if no variable occurs free in it. The set of all freely occurring variables of M is denoted by $fr(M)$.*

A few examples shall illustrate this. In $M = (\lambda\mathtt{x}_0.(\mathtt{x}_0\mathtt{x}_1))$ the variable $\mathtt{x}_0$ occurs only bound, since it only occurs in a subterm of the form $(\lambda\mathtt{x}_0.N)$ (for example $N := (\mathtt{x}_0\mathtt{x}_1)$). However, $\mathtt{x}_1$ occurs free. A variable may occur free as well as bound in a term. An example is the variable $\mathtt{x}_0$ in $(\mathtt{x}_0(\lambda\mathtt{x}_0.\mathtt{x}_0))$.

Bound and free variable occurrences behave differently under replacement. If M is a λ–term and x a variable then denote by $[N/x]M$ the result of replacing x by N. In this replacement we do not simply replace all occurrences of x by N; the definition of replacement requires some care. Therefore $[N/x]M$ is different from $M[x \mapsto N]$, which denotes the result of replacing *every* occurrence of x in M by N.

(a)	$[N/x]t$	$:=$	$t[x \mapsto N]$	t an Ω–term
(b)	$[N/x](MM')$	$:=$	$(([N/x]M)([N/x]M'))$	
(c)	$[N/x](\lambda x.M)$	$:=$	$(\lambda x.M)$	
(d)	$[N/x](\lambda y.M)$	$:=$	$(\lambda y.[N/x]M)$ if $y \neq x, y \notin fr(N)$ or $x \notin fr(M)$	
(e)	$[N/x](\lambda y.M)$	$:=$	$(\lambda z.[N/x][z/y]M)$ if $y \neq x, y \in fr(N)$ and $x \in fr(M)$	

In (e) we have to choose z in such a way that it does not occur free in N or M. In order for substitution to be uniquely defined we assume that the set of variables is well ordered and we choose z to be the smallest variable that satisfies the above conditions. The precaution in (e) of an additional substitution is necessary. For let $y = \mathtt{x}_1$ and $M = \mathtt{x}_0$. Then without this (that is, with (d) alone),

$$[\mathtt{x}_1/\mathtt{x}_0](\lambda\mathtt{x}_0.\mathtt{x}_1) = (\lambda\mathtt{x}_1.[\mathtt{x}_1/\mathtt{x}_0]\mathtt{x}_0) = (\lambda\mathtt{x}_1.\mathtt{x}_1) .$$

This contradicts our intuition. For $(\lambda\mathtt{x}_1.\mathtt{x}_0)$ is the function which for given $\mathtt{x}_1$ returns $\mathtt{x}_0$. However, $(\lambda\mathtt{x}_1.\mathtt{x}_1)$ is the identity function and so different from that function. Now the substitution of a variable by another variable shall not change the course of values of a function.

We shall present the theory of λ–terms which we shall use in the sequel. It consists in a set of equations $M = N$, where M and N are terms. These are subject to the laws displayed in Table 3.1. The theory axiomatized by (a) – (g) and (i) is called $\boldsymbol{\lambda}$, the theory axiomatized by (a) – (i) $\boldsymbol{\lambda\eta}$. Notice that (a) – (e) simply say that $\doteq$ is a congruence. A different rule is the following so called **extensionality rule**.

$$\text{(ext)} \qquad Mx = Nx \Rightarrow M = N$$

Table 3.1: Axioms and Rules of the λ–Calculus

(a)	$M = M$	
(b)	$M = N \Rightarrow N = M$	
(c)	$M = N, N = L \Rightarrow M = L$	
(d)	$M = N \Rightarrow (ML) = (NL)$	
(e)	$M = N \Rightarrow (LM) = (LN)$	
(f)	$(\lambda x.M) = (\lambda x.[y/x]M)$	(α–conversion)
(g)	$(\lambda x.M)N = [N/x]M$	(β–conversion)
(h)	$(\lambda x.M) = M, x \notin fr(M)$,	(η–conversion)
(i)	$M = N \Rightarrow (\lambda x.M) = (\lambda x.N)$	(ξ–rule)

It can be shown that $\boldsymbol{\lambda}$ + (ext) = $\boldsymbol{\lambda\eta}$. The model theory of λ–calculus is somewhat tricky. Basically, all that is assumed is that we have a domain D and an operation $\bullet$ that interprets function application. However, in order to define abstraction, we must do some more work. Basically, define a valuation β as a function from V to D. Now define $[M]^\beta$ as follows.

$$\begin{array}{lcl} [\mathtt{x}_i]^\beta & := & \beta(\mathtt{x}_i) \\ [(MN)]^\beta & := & [M]^\beta([N]^\beta) \\ [(\lambda x.M)]^\beta \bullet a & := & [M]^{\beta[x:=a]} \end{array}$$

The last clause does not fix the interpretation of $(\lambda x.M)$ uniquely given M. If it does, the structure is called extensional. We shall return to the question below. Before we can do so, we shall develop some more syntactic techniques to deal with λ–terms.

Definition 3.3.4 *Let M and N be λ–terms. We say, N* ***is obtained from*** *M* ***by replacement of bound variables*** *or by α–**conversion** and write $M \rightsquigarrow_\alpha N$ if there is a subterm $\lambda y.L$ of M and a variable z which does not occur in L such that N is the result of replacing an occurrence of $(\lambda y.L)$ by $(\lambda v.[v/y]L)$. The relation $\triangleright_\alpha$ is the transitive closure of $\rightsquigarrow_\alpha$. N is* ***congruent to*** *M, in symbols $M \equiv_\alpha N$, if both $M \triangleright_\alpha N$ and $N \triangleright_\alpha M$.*

Similarly, the definition of β–conversion.

Definition 3.3.5 *Let M be a λ–term. We write $M \rightsquigarrow_\beta N$ and say that M* ***contracts to*** *N if N is the result of a single replacement of an occurrence of $(\lambda x\,.\,L)\,P$ in M by $[P/x]L$. Further, we write $M \rhd_\beta N$ if N results from M by a series of contractions and $M \equiv_\beta N$ if $M \rhd_\beta N$ and $N \rhd_\beta M$.*

A term of the form $((\lambda x.M)\,N)$ is called a **redex** and $[N/x]M$ its **contractum**. The step from the redex to the contractum represents the evaluation of a function to its argument. A λ–term is *evaluated* or *in normal form* if it contains no redex.

Definition 3.3.6 *Similarly, for the notation $\rightsquigarrow_{\alpha\beta}$, $\rhd_{\alpha\beta}$ and $\equiv_{\alpha\beta}$. Call M and N $\alpha\beta$–***equivalent*** *($\alpha\beta\eta$–***equivalent***) if $\langle M, N\rangle$ is in the least equivalence relation containing $\rhd_{\alpha\beta}$ ($\rhd_{\alpha\beta\eta}$.*

Proposition 3.3.7 *$\boldsymbol{\lambda} \vdash M \doteq N$ if and only if M and N are $\alpha\beta$–equivalent. $\boldsymbol{\lambda\eta} \vdash M \doteq N$ if and only if M and N are $\alpha\beta\eta$–equivalent.*

A λ–term is said to be in ***normal form*** *if it does not contain a redex. If $M \rhd_{\alpha\beta} N$ and N is in normal form then N is called a* ***normal form of*** *M.*

Without proof we state the following theorem.

Theorem 3.3.8 (Church, Rosser) *Let L, M, N be λ–terms such that $L \rhd_{\alpha\beta} M, N$. Then there exists a P such that $M \rhd_{\alpha\beta} P$ and $N \rhd_{\alpha\beta} P$.*

The proof can be found in all books on the λ–calculus. This theorem also holds for $\rhd_{\alpha\beta\eta}$.

Corollary 3.3.9 *Let N and N' be normal forms of M. Then $N \equiv_\alpha N'$.*

The proof is simple. For by the theorem there exists a P such that $N \rhd_{\alpha\beta} P$ and $N' \rhd_{\alpha\beta} P$. But since N as well as N' do not contain any redex and α–conversion does not introduce any redexes then P results from N and N' by α–conversion. Hence P is α–congruent with N and N' and hence N and N' are α–congruent.

Not every λ–term has a normal form. For example

$$(\texttt{(}\lambda\texttt{x}_0\texttt{.(x}_0\texttt{x}_0\texttt{))(}\lambda\texttt{x}_0\texttt{.(x}_0\texttt{x}_0\texttt{)))} \triangleright_\beta \texttt{((}\lambda\texttt{x}_0\texttt{.(x}_0\texttt{x}_0\texttt{))(}\lambda\texttt{x}_0\texttt{.(x}_0\texttt{x}_0\texttt{)))}$$

Or

$$\begin{aligned}
& \texttt{((}\lambda\texttt{x}_0\texttt{.((x}_0\texttt{x}_0\texttt{)x}_1\texttt{))(}\lambda\texttt{x}_0\texttt{.((x}_0\texttt{x}_0\texttt{)x}_1\texttt{)))} \\
\triangleright_\beta\ & \texttt{(((}\lambda\texttt{x}_0\texttt{.((x}_0\texttt{x}_0\texttt{)x}_1\texttt{))(}\lambda\texttt{x}_0\texttt{.((x}_0\texttt{x}_0\texttt{)x}_1\texttt{)))x}_1\texttt{)} \\
\triangleright_\beta\ & \texttt{((((}\lambda\texttt{x}_0\texttt{.((x}_0\texttt{x}_0\texttt{)x}_1\texttt{))(}\lambda\texttt{x}_0\texttt{.((x}_0\texttt{x}_0\texttt{)x}_1\texttt{)))x}_1\texttt{)x}_1\texttt{)}
\end{aligned}$$

The typed λ–calculus differs from the calculus which has just been presented by an important restriction, namely that every term must have a type.

Definition 3.3.10 *Let B be a set. The set of* ***types over*** *B, $\mathrm{Typ}_{\rightarrow}(B)$, is the smallest set M for which the following holds.*

1. *$B \subseteq M$.*
2. *If $\alpha \in M$ and $\beta \in M$ then $\alpha \rightarrow \beta \in M$.*

In other words: types are nothing but terms in the signature $\{\rightarrow\}$ with $\Omega(\rightarrow) = 2$ over a set of basic types. Each term is associated with a type and the structure of terms is restricted by the type assignment. Further, all Ω–terms are admitted. Their type is already fixed. The following rules are valid.

1. If $\texttt{(}MN\texttt{)}$ is a term of type γ then there is a type α such that M has the type $\alpha \rightarrow \gamma$ and N the type γ.
2. If M has the type γ and x_α is a variable of type α then $\texttt{(}\lambda x_\alpha\texttt{.}M\texttt{)}$ is of type $\alpha \rightarrow \gamma$.

Notice that for every type α there are variables of type α and countably infinitely many of them. More exactly, we have $V^\alpha := \{\texttt{x}^i_\alpha : i \in \omega\}$. Instead of these we often use the variables x_α, y_α

and so on. If $\alpha \neq \beta$ then also $x_\alpha \neq x_\beta$ (they are different variables). With these conditions the formation of λ–terms is severely restricted. For example $(\lambda\mathtt{x}_0.(\mathtt{x}_0\mathtt{x}_0))$ is cannot be a typed term no matter which type $\mathtt{x}_0$ has. One can show that a typed term always has a normal form. This is in fact an easy matter. Notice by the way that if the term $(\mathtt{x}_0+\mathtt{x}_1)$ has the type α and $\mathtt{x}_0$ and $\mathtt{x}_1$ the type α, the function $(\lambda\mathtt{x}_0.(\lambda\mathtt{x}_1.(\mathtt{x}_0+\mathtt{x}_1)))$ has the type $\alpha \to (\alpha \to \alpha)$. The type of an Ω–term is the type of its value, in this case α.

We shall present a model of the λ–calculus which we already need in this chapter. The easiest is if we study the purely applicative structures and then return to the abstraction after the introduction of combinators. In the untyped case application is a function that is everywhere defined. The model structures are therefore so called *applicative structures.*

Definition 3.3.11 *An **applicative structure** is a pair $\mathfrak{A} = \langle A, \bullet\rangle$ where $\bullet$ is a binary operation on A. If $\bullet$ is only a partial operation, $\langle A, \bullet\rangle$ is called a **partial applicative structure**. $\mathfrak{A}$ is called **extensional** if for all $a, b \in A$:*

$$a = b \text{ if and only if for all } c \in A : a \bullet c = b \bullet c\,.$$

Definition 3.3.12 *Let B be a set of basic types. A **typed applicative structure over** B is a structure $\langle\{A_\alpha : \alpha \in \mathrm{Typ}_\to(B)\}, \bullet\rangle$ such that (a) A_α is a set for every α and (b) $a \bullet b$ is defined if and only if there are types $\alpha \to \beta$ and α such that $a \in A_{\alpha\to\beta}$ and $b \in A_\alpha$, and then $a \bullet b \in A_\beta$.*

A typed applicative structure defines a partial applicative structure. Namely, put $A := \bigcup_\alpha A_\alpha$; then $\bullet$ is nothing but a partial binary operation on A. The typing is then left implicit. (Recovering the types of elements is not a trivial affair, see the exercises.) Not every partial applicative structure can be typed, though.

We are interested in models in which A consists of sets and $\bullet$ is the usual functional application as defined in sets. More precisely, we want that A_α is a set of sets for every α. So if the type is associated with the set S then a variable may assume as value any member of S. So it follows that if β is associated with the set T and M has the type β then the interpretation of $(\lambda \mathtt{x}_\alpha . X)$ is a function from S to T. We set the realization of $\alpha \to \beta$ to be the set of all functions from S to T. This is an arbitrary choice, a different choice (for example a suitable subset) would do as well.

Let M and N be sets. Then a function from M to N is a subset F of the cartesian product $M \times N$ which satisfies certain conditions (see Section 1.1). Namely, for every $x \in M$ there must be a $y \in N$ such that $\langle x, y\rangle \in F$ and if $\langle x, y\rangle \in F$ and $\langle x, y'\rangle \in F$ then $y = y'$. (For partial functions the first condition is omitted. Everything else remains. For simplicity we shall deal only with totally defined functions.) Normally one thinks of a function as something that gives values for certain arguments. This is not so in this case. F is not a function in this sense, it is just the **graph** of a function. In set theory one does not distinguish between a function and its graph. We shall return to this later. How do we have to picture F as a set? Recall that we have defined

$$M \times N = \{\langle x, y\rangle : x \in M, y \in N\}$$

This is a set. Notice that $M \times (N \times O) \neq (M \times N) \times O$. For $M \times (N \times O)$ consists of all pairs of the form $\langle x, \langle y, z\rangle\rangle$, where $x \in M$, $y \in N$ and $z \in O$, while $(M \times N) \times O$ consists of all pairs of the form $\langle\langle x, y\rangle, z\rangle$. And if $\langle x, \langle y, z\rangle\rangle = \langle\langle x', y'\rangle, z'\rangle$, then we must have $x = \langle x', y'\rangle$ and $\langle y, z\rangle = z'$. From this follows $x' \in x$. M possesses an element which is minimal with respect to $\in$, x^* (foundation). If we put this element in place of x we get a contradiction. This shows $M \times (N \times O) \neq (M \times N) \times O$. However the mapping

$$\rtimes \; : \langle x, \langle y, z\rangle\rangle \mapsto \langle\langle x, y\rangle, z\rangle : M \times (N \times O) \to (M \times N) \times O$$

is a bijection. Its inverse is the mapping

$$\ltimes \; : \langle\langle x, y\rangle, z\rangle \mapsto \langle x, \langle y, z\rangle\rangle : (M \times N) \times O \to M \times (N \times O)$$

Finally we put

$$M \to N := \{F \subseteq M \times N : F \text{ a function }\} .$$

This is a different notation as we have used so far but in this situation it is better suited. Now functions are also sets and their argument are sets, too. Hence we need a map which applies a function to an argument. Since it must be defined for all cases of functions and arguments, it must by necessity be a partial function. If x is a function and y an element, we define $\mathsf{app}(x, y)$ as follows.

$$\mathsf{app}(x, y) := \begin{cases} z & \text{if } \langle y, z\rangle \in x, \\ \star & \text{if no } z \text{ exists such that } \langle y, z\rangle \in x. \end{cases}$$

app is a partial function. Its graph in the universe of sets is a proper class, however. It is the class of pairs $\langle\langle F, x\rangle, y\rangle$, where F is a function and $\langle x, y\rangle \in F$.

If now $F \in M \to (N \to O)$ then $F \subseteq M \times (N \to O) \subseteq M \times (N \times O)$. Then $\rtimes[F] \subseteq (M \times N) \times O$ and one easily calculates that $\rtimes[F] \subseteq (M \times N) \to O$. In this way a unary function with values in $N \to O$ becomes a unary function from $M \times N$ to O (or a binary function from M, N to O). Conversely, one can see that if $F \in (M \times N) \to O$ then $\ltimes[F] \in M \to (N \to O)$.

Theorem 3.3.13 *Let V_ω be the set of finite sets. Then $\langle V_\omega, \mathsf{app}\rangle$ is a partial applicative structure.*

In place of V_ω one can take any V_κ where κ is a limit ordinal (that is, an ordinal without predecessor). For our use the just described structure is sufficient. A more general result is the following for the typed calculus.

Theorem 3.3.14 *Let B be the set of basic types and M_b, $b \in B$, arbitrary sets. Let M_α be inductively defined by $M_{\alpha\to\beta} := (M_\beta)^{M_\alpha}$. Then $\langle\{M_\alpha : \alpha \in \mathrm{Typ}_\to(B)\}, \mathsf{app}\rangle$ is a typed applicative structure. Moreover, it is extensional.*

For a proof of this theorem one simply has to check the conditions.

In categorial grammar, with which we shall deal in this chapter, we shall use λ–terms to name meanings for symbols and strings. It is important however that the λ–term is only a formal entity (namely a certain string), and it is not the meaning in the proper sense of the word. For example, $(\lambda\mathtt{x}_0.(\lambda\mathtt{x}_1.\mathtt{x}_0+\mathtt{x}_1))$ is a string which names a function. In the set universe, this function is a subset of $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$. For this reason one has to distinguish between equality $=$ and the symbol $\equiv$. $M = N$ means that we are dealing with the same strings (hence literally the same λ–terms) while $M \equiv N$ means that M and N name the same function. In this sense $(\lambda\mathtt{x}_0.(\lambda\mathtt{x}_1.\mathtt{x}_0+\mathtt{x}_1))(\mathtt{x}_0)(\mathtt{x}_2) \neq \mathtt{x}_0+\mathtt{x}_2$, but they also denote the same value. Nevertheless, in what is to follow we shall not always distinguish between a λ–term and its interpretation, in order not to make the notation too opaque.

The λ–calculus has a very big disadvantage, namely that it requires some caution in dealing with variables, which can be quite serious. They can be avoided if one abandons the use of variables altogether. This is achieved through the use of combinators. Given a set V of variables and the zeroary constants $\mathbf{S}$, $\mathbf{K}$, $\mathbf{I}$, combinators are terms over the signature that has only one more binary symbol, $\bullet$. This symbol is generally omitted, and terms are formed using infix notation with brackets. Call this signature Γ.

Definition 3.3.15 *An element of* $\mathrm{Tm}_\Gamma(V)$ *is called a* ***combinatorial term****. A* ***combinator*** *is an element member of* $\mathrm{Tm}_\Gamma(\varnothing)$*.*

Further, the redex relation $\triangleright$ is axiomatically defined as follows. These are the axioms.

$$\begin{array}{lcl}
\mathbf{I}X & \triangleright & X \\
\mathbf{K}XY & \triangleright & X \\
\mathbf{S}XYZ & \triangleright & XY(XZ) \\
X & \triangleright & X
\end{array}$$

Combinatory logic (CL) is (a) – (e) above plus the equations $\mathbf{I}X = X$, $\mathbf{K}XY = X$ and $\mathbf{S}XYZ = XZ(YZ)$. It is an equational theory. As regards reduction, we have the following rules.

1. If $X \triangleright Y$ and $Y \triangleright Z$ then $X \triangleright Z$.

2. If $X \triangleright Y$ then $XZ \triangleright YZ$.

3. If $X \triangleright Y$ then $ZX \triangleright ZY$.

We note that there is a combinator $\mathbf{C}$ using only $\mathbf{K}$ and $\mathbf{S}$ such that $\mathbf{C} \triangleright \mathbf{I}$ (see Exercise 3.3). This explains why $\mathbf{I}$ is sometimes omitted.

We shall now show that combinators can be defined by λ–terms and vice versa. First, define

$$\begin{array}{lcl} \mathsf{I} & := & (\lambda\mathtt{x}_0\,.\,\mathtt{x}_0) \\ \mathsf{K} & := & (\lambda\mathtt{x}_0\,.\,(\lambda\mathtt{x}_1\,.\,\mathtt{x}_0)) \\ \mathsf{S} & := & (\lambda\mathtt{x}_0\,.\,(\lambda\mathtt{x}_1\,.\,(\lambda\mathtt{x}_2\,.\,\mathtt{x}_0\mathtt{x}_2(\mathtt{x}_1\mathtt{x}_2)))) \end{array}$$

Define a translation ${}^\lambda$ by $X^\lambda := X$ for $X \in V$, $\mathbf{S}^\lambda := \mathsf{S}$, $\mathbf{K}^\lambda := \mathsf{K}$, $\mathbf{I}^\lambda := \mathsf{I}$. Then the following is proved by induction on the length of the proof.

Theorem 3.3.16 *Let C and D be combinators. If $\mathsf{CL} \vdash C \doteq D$ then also $\boldsymbol{\lambda} \vdash C^\lambda \doteq D^\lambda$. Further, if $\mathsf{CL} \vdash C \triangleright D$ then also $\boldsymbol{\lambda} \vdash C^\lambda \triangleright_\beta D^\lambda$.*

The converse translation is more difficult. Put

1. $[x]x := \mathbf{I}$.

2. $[x]M := \mathbf{K}M$ if x does not occur in M.

3. $[x]Mx := M$ if x does not occur free in M.

4. $[x]MN := \mathbf{S}([x]M)([x]N)$, otherwise.

For example $[\mathtt{x}_1]\mathtt{x}_1\mathtt{x}_0 = \mathbf{S}([\mathtt{x}_1]\mathtt{x}_1)([\mathtt{x}_1]\mathtt{x}_0) = \mathbf{SI}(\mathbf{K}\mathtt{x}_0)$. Indeed, if one applies this to $\mathtt{x}_0$, then one gets

$$\mathbf{SI}(\mathbf{K}\mathtt{x}_0)\mathtt{x}_1 \triangleright \mathbf{I}\mathtt{x}_1(\mathbf{K}\mathtt{x}_0\mathtt{x}_1) \triangleright \mathtt{x}_1(\mathbf{K}\mathtt{x}_0\mathtt{x}_1) \triangleright \mathtt{x}_1\mathtt{x}_0$$

Further, one has

$$\mathbf{U} := [\mathtt{x}_1](\,[\mathtt{x}_0]\mathtt{x}_1\mathtt{x}_0) = [\mathtt{x}_1]\mathbf{SI}(\mathbf{K}\mathtt{x}_0) = \mathbf{S}(\mathbf{K}(\mathbf{SI}))\mathbf{K}$$

The reader may verify that $\mathbf{U}\mathtt{x}_0\mathtt{x}_1 \triangleright \mathtt{x}_1\mathtt{x}_0$. Now define ${}^{\kappa}$ by $x^{\kappa} := x$, $x \in V$, $(MN)^{\kappa} := (M^{\kappa}N^{\kappa})$ and $(\lambda x.N)^{\kappa} := [x]N^{\kappa}$.

Theorem 3.3.17 *Let C be a closed λ–term. Then $\boldsymbol{\lambda} \vdash C \doteq C^{\kappa}$.*

Now we have defined translations from λ–terms to combinators and back. It can be shown, however, that the theory λ is stronger than CL under translation. Curry found a list $\mathbf{A}_{\beta}$ of five equations such that $\boldsymbol{\lambda}$ is as strong as $\mathsf{CL}+\mathbf{A}_{\beta}$ in the sense of Theorem 3.3.18 below. Also, there is a list $\mathbf{A}_{\beta\eta}$ such that $\mathsf{CL}+\mathbf{A}_{\beta\eta}$ is equivalent to $\boldsymbol{\lambda\eta} = \boldsymbol{\lambda} +$ (ext). $\mathbf{A}_{\beta\eta}$ also is equivalent to the first–order postulate (ext): $(\forall xy)((\forall z)(x \bullet z \doteq y \bullet z) \to x \doteq y)$.

Theorem 3.3.18 (Curry) *Let M and N be λ–terms.*

1. *If $\boldsymbol{\lambda} \vdash M \doteq N$, $\mathsf{CL} + \mathbf{A}_{\beta} \vdash M^{\kappa} \doteq N^{\kappa}$.*
2. *If $\boldsymbol{\lambda\eta} \vdash M \doteq N$, $\mathsf{CL} + \mathbf{A}_{\beta\eta} \vdash M^{\kappa} \doteq N^{\kappa}$.*

There is also a typing version of combinatorial logic. There are two basic approaches. The first is to define typed combinators. The basic combinators now split into infinitely many typed versions as follows.

Combinator	Type
$\mathbf{I}_{\alpha}$	$\alpha \to \alpha$
$\mathbf{K}_{\alpha,\beta}$	$\alpha \to (\beta \to \alpha)$
$\mathbf{S}_{\alpha,\beta,\gamma}$	$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha\gamma)$

Together with $\bullet$ they form the typed signature Γ^{τ}. For each type there are countably infinitely many variables of that type in V. **Typed combinatorial terms** are elements of $\mathrm{Tm}_{\Gamma^{\tau}}(V)$, and **typed combinators** are elements of $\mathrm{Tm}_{\Gamma^{\tau}}$. Further, if M is a combinator of type $\alpha \to \beta$ and N a combinator of type α then

(MN) is a combinator of type β. In this way, every typed combinatorial term has a unique type.

The second approach is to keep the symbols $\mathbf{I}$, $\mathbf{K}$ and $\mathbf{S}$ and to let them stand for any of the above typed combinators. In terms of functions, $\mathbf{I}$ takes an argument N of any type α and returns N (of type α). Likewise, $\mathbf{K}$ is defined on any M, N of type α and β, respectively, and $\mathbf{K}MN = M$ of type α. Also, $\mathbf{K}M$ is defined and of type $\beta \to \alpha$. Basically, the language is the same as in the untyped case. A combinatorial term (combinator) is **stratified** if for each variable and each occurrence of $\mathbf{I}$, $\mathbf{K}$, $\mathbf{S}$ there exists a type such that if that (occurrence of the) symbol is assigned that type, the resulting string is a typed combinatorial term. (So, while each occurrence of $\mathbf{I}$, $\mathbf{K}$ and S, respectively, may be given a different type, each occurrence of the same variable must have the same type.) For example, $\mathbf{B} := \mathbf{S}(\mathbf{KS})\mathbf{K}$ is stratified, $\mathbf{SII}$ is not.

Suppose that there are types α, β, γ, δ, ϵ such that

$$\frac{((\mathbf{S}_{\alpha,\beta,\gamma} \qquad\qquad\qquad\qquad \mathbf{I}_\delta) \quad \mathbf{I}_\epsilon)}{(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)) \quad \delta \to \delta \quad \epsilon \to \epsilon}$$

is a typed combinator. Then, since $\mathbf{S}_{\alpha,\beta,\gamma}$ is applied to $\mathbf{I}_\delta$ we must have $\delta \to \delta = \alpha \to (\beta \to \gamma)$, whence $\alpha = (\beta \to \gamma)$. Now, the result is applied to $\mathbf{I}_\epsilon$, and so we have $(\alpha \to \beta) = (\beta \to \gamma) \to \beta = \epsilon \to \epsilon$, whence $\beta \to \gamma = \epsilon = \beta$, which is impossible. So, $\mathbf{SII}$ is not stratified. On the other hand, $\mathbf{B}$ is stratified. Assume types such that $\mathbf{S}_{\zeta,\eta,\theta}(\mathbf{K}_{\alpha,\beta}\mathbf{S}_{\gamma,\delta,\epsilon})\mathbf{K}_{\iota,\kappa}$ is a typed combinator. First $\mathbf{K}_{\alpha,\beta}$ is applied to $\mathbf{S}_{\gamma,\delta,\epsilon}$. This means that

$$\alpha = (\gamma \to (\delta \to \epsilon)) \to ((\gamma \to \delta) \to (\gamma \to \epsilon))$$

The result has type

$$\beta \to ((\gamma \to (\delta \to \epsilon)) \to ((\gamma \to \delta) \to (\gamma \to \epsilon)))$$

This is the argument of $\mathbf{S}_{\zeta,\eta,\theta}$. Hence we must have

$$\zeta \to (\eta \to \theta) = \beta \to ((\gamma \to (\delta \to \epsilon)) \to ((\gamma \to \delta) \to (\gamma \to \epsilon)))$$

So, $\zeta = \beta$, $\eta = \gamma \to (\delta \to \epsilon)$, $\theta = (\gamma \to \delta) \to (\gamma \to \epsilon)$. The resulting type is $(\zeta \to \eta) \to (\zeta \to \theta)$. This is applied to $\mathbf{K}_{\iota,\kappa}$ of type $\iota \to (\kappa \to \iota)$. For this to be well defined, $\iota \to (\kappa \to \iota) = \zeta \to \eta$, or $\iota = \zeta = \beta$ and $\kappa \to \iota = \eta = \gamma \to (\delta \to \epsilon)$. Finally, this results in $\kappa = \gamma$, $\iota = \beta = \delta \to \epsilon$. So, α, γ, δ and ϵ may be freely chosen, and the other types are immediately defined.

It is the second approach that will be the most useful for us later on. We call combinators **implicitly typed** if they are thought of as typed in this way. (In fact, they simply are untyped terms.) The same can be done with λ–terms, giving rise to the notion of a stratified λ–term. In the sequel we shall not distinguish between combinators and their representing λ–terms.

Finally, let us return to the models of the λ–calculus. We still owe a definition of abstraction. As before we begin with the untyped case. First of all $\rho : V \to A$ is a so called **valuation**. Then $(N)_\rho$ is defined inductively as follows.

$$\begin{array}{lcl} (a)_\rho & := & a \\ (\mathtt{x}_i)_\rho & := & \rho(\mathtt{x}_i) \\ (\mathtt{(}MN\mathtt{)})_\rho & := & (M)_\rho \bullet (N)_\rho \end{array}$$

This defines a value for every applicative term under a valuation. An applicative structure is called *combinatorially complete* if this term denotes a function.

Definition 3.3.19 *An applicative structure $\mathfrak{A}$ is called **combinatorially complete** if for every term t in the language with free variables from $\{\mathtt{x}_i : i < n\}$ there exists a y such that for all $b_i \in A$, $i < n$:*

$$(\cdots((y \bullet b_0) \bullet b_1) \bullet \cdots \bullet b_{n-1}) = t(b_0, \ldots, b_{n-1})$$

This means that for every term t there exists an element which represents this term. This is, as is easily seen, the element which we have denoted by

$$(\lambda\mathtt{x}_0 . (\lambda\mathtt{x}_1 . \cdots . (\lambda\mathtt{x}_{n-1} . t(\mathtt{x}_0, \cdots, \mathtt{x}_{n-1})) \cdots)) \; .$$

Evidently, one only needs to introduce countably many abstraction operations, one for every number $i < \omega$, but we shall ignore all details (but see Section 4.4). One can however also make a detour via combinatory logic and define the abstraction via combinators. In view of the results obtained above we get the following result.

Theorem 3.3.20 (Schönfinkel) *$\mathfrak{A}$ is combinatorially complete if there are elements k and s such that*

$$((k \bullet a) \bullet b) = a, \qquad (((s \bullet a) \bullet b) \bullet c) = (a \bullet c) \bullet (b \bullet c)$$

Definition 3.3.21 *A structure $\langle A, \bullet, \mathsf{k}, \mathsf{s}\rangle$ is called a **combinatory algebra** if $\mathfrak{A} \vDash \mathsf{k} \bullet x \bullet x \doteq x, \mathsf{s} \bullet x \bullet y \bullet z \doteq x \bullet z \bullet (y \bullet z)$. It is a λ–**algebra (extensional)** if it satisfies $\mathbf{A}_\beta$ ($\mathbf{A}_{\beta\eta}$) in addition.*

So, the class of combinatory algebras is an equationally definable class. (This is why we have not required $|A| > 1$ as is often done.) Again, the partial case is interesting. Hence, we can use the theorems by Section 1.1 to create structures. Two models are of particular significance. One is based on the algebra of combinatorial terms over V modulo derivable identity, the other is the algebra of combinators modulo derivable identity. Indirectly, this also shows how to create models for the λ–calculus. We shall explain a different method below in Section 4.4.

Call a structure $\langle A, \bullet, \mathsf{k}, \mathsf{s}\rangle$ a **partial combinatory algebra** if (i) $\mathsf{s} \bullet x \bullet y$ is always defined and (ii) the defining equations hold in the intermediate sense, that is, if one side is defined so is the other and they are equal. Consider once again the universe V_ω. Define

$$\begin{aligned} \mathfrak{k} &:= \{\langle x, \langle y, x\rangle\rangle : x, y \in V_\omega\} \\ \mathfrak{s} &:= \{\langle x, \langle y, \langle z, \mathsf{app}(\mathsf{app}(x, z), \mathsf{app}(y, z))\rangle\rangle\rangle : x, y, z \in V_\omega\} \end{aligned}$$

Then $\langle V_\omega, \mathsf{app}, \mathfrak{k}, \mathfrak{s}\rangle$ is not a partial combinatory algebra. The problem is that $\mathsf{app}(\mathsf{app}(\mathfrak{s}, x), y)$ is not always defined. In fact, the defining equations hold only in the weak sense: if both sides are defined, then they are equal.

In the typed case we need a variety of combinators. More exactly: for all types α, β and γ we need elements $\mathbf{k}_\delta \in A_\delta$, $\delta = \alpha \to (\beta \to \alpha)$ and $\mathbf{s}_\eta \in A_\eta$, $\eta = (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$ such that for all $a \in A_\alpha$ and $b \in A_\beta$ we have

$$(\mathbf{k}_\delta \bullet a) \bullet b = a$$

and for every $a \in A_{\alpha\to(\beta\to\gamma)}$, $b \in A_{\alpha\to\beta}$ and $c \in A_\alpha$ we have

$$((\mathbf{s}_\eta \bullet a) \bullet b) \bullet c = (a \bullet c) \bullet (b \bullet c)\ .$$

We now turn to an interesting connection between intuitionistic logic and type theory, known as the *Curry–Howard–Isomorphism.* Write $M : \varphi$ if M is a λ–term of type φ. Notice that while each term has exactly one type, there are infinitely many terms having the same type. The following is a Gentzen–calculus for statements of the form $M : \varphi$. Here, Γ, Δ, Θ denote arbitrary sets of such statements, x, y individual variables (of appropriate type), and M, N terms.

$$\text{(axiom)}\quad x : \varphi \vdash x : \varphi \qquad \text{(M)}\quad \frac{\Gamma \vdash M : \varphi}{\Gamma, x : \chi \vdash M : \varphi}$$

$$\text{(cut)}\quad \frac{\Gamma \vdash M : \varphi \quad \Delta, x : \varphi, \Theta \vdash N : \chi}{\Delta, \Gamma, \Theta \vdash [M/x]N : B}$$

$$(\mathbf{E}\!\to)\quad \frac{\Gamma \vdash M : \varphi \to \chi \quad \Delta \vdash N : \varphi}{\Gamma, \Delta \vdash (MN) : \chi}$$

$$(\mathbf{I}\!\to)\quad \frac{\Gamma, x : \varphi \vdash M : \chi}{\Gamma \vdash (\lambda x . M) : \varphi \to \chi}$$

First of all notice that if we strip off the labelling by λ–terms we get a natural deduction calculus for intuitionistic logic (in the only connective $\to$). Hence if a sequent $\{M_i : \varphi_i : i < n\} \vdash N : \chi$ is derivable then $\overset{\mathfrak{G}}{\leadsto} \{\varphi_i : i < n\} \vdash \chi$, whence $\{\varphi_i : i < n\} \vdash^H \chi$. Conversely, given a natural deduction proof of $\overset{\mathfrak{G}}{\leadsto} \{\varphi_i : i < n\} \vdash \chi$, we can decorate the proof with λ–terms by assigning any variables at the leaves of the tree for the axioms and then descending it until we hit the root. Then we get a proof of the sequent $\{M_i : \varphi_i : i < n\} \vdash N : \chi$ in the above calculus.

Now we interpret the intuitionistic formulae in this proof calculus as types. For a set Γ of λ–terms over the set B of basic types we put

$$|\Gamma| := \{\varphi \in \mathrm{Typ}_{\to}(B) : \text{ there is } M \in \Gamma \text{ of type } \varphi\}$$

Definition 3.3.22 *For a set Γ of types and a single type φ over a set B of basic types we put $\Gamma \vdash^{\lambda} \varphi$ if there is a term M of type φ such that every type of a variable occurring free in M is in Γ.*

Returning to our calculus above we notice that if

$$\{M_i : \varphi_i : i < n\} \vdash N : \chi$$

is derivable, we also have $\{\varphi_i : i < n\} \vdash^{\lambda} \chi$. This is established by induction on the proof. Moreover, the converse also holds (by induction on the derivation). Hence we have the following result.

Theorem 3.3.23 (Curry) $\Gamma \vdash^{\lambda} \varphi$ *if and only if* $\Gamma \vdash^{H} \varphi$.

The correspondence between intuitionistic formulae and types has also been used to obtain a rather nice characterization of shortest proofs. Basically, it turns out that a proof of $\Gamma \vdash N : \varphi$ can be shortened if N contains a redex. Suppose, namely, that N contains the redex $(\lambda x \,.\, M)U$. Then, as is easily seen, the proof contains a proof of $\Delta \vdash (\lambda x \,.\, M)U : \chi$. This proof part can be shortened. To simplify the argument here we assume that no use of (cut) and (M) has been made. Observe that we can assume that this very sequent has been introduced by the rule (**I**$\to$) and its left premiss by the rule (**E**$\to$).

$$\dfrac{\dfrac{\Delta', x : \psi \vdash M : \chi}{\Delta' \vdash (\lambda x \,.\, M) : \psi \to \chi} \qquad \Delta'' \vdash U : \psi}{\Delta', \Delta'' \vdash (\lambda x \,.\, M)U : \chi}$$

Here, $\Delta = \Delta' \cup \Delta''$. Then a single application of (cut) gives the following result:

$$\frac{\Delta'' \vdash U : \psi \qquad \Delta', x : \psi \vdash M : \chi}{\Delta', \Delta'' \vdash [M/x]U : \chi}$$

While the types and the antecedent have remained constant, the conclusion now has a term associated to it that is derived from contracting the redex. The same can be shown if we take intervening applications of (cut) and (M), but the proof is more involved. Essentially, we need to perform more complex proof transformations. There is another simplification that can be made, namely when the derived term is explicitly α–converted. Then we have a sequent of the form $\Gamma \vdash (\lambda x . Mx) : \varphi \to \chi$. Then, again putting aside intervening occurrences of (cut) and (M), the proof is as follows.

$$\frac{\dfrac{\Gamma \vdash (\lambda x . Mx) : \varphi \to \chi \qquad y : \varphi \vdash y : \varphi}{\Gamma, y : \varphi \vdash (My) : \chi}}{\Gamma \vdash (\lambda y . My) : \varphi \to \chi}$$

This proof part can be eliminated completely, leaving only the proof of the left hand premiss. An immediate corollary of this fact is that if $\{x_i : \varphi_i : i < n\} \vdash N : \chi$ is provable for some N, then there is an N' obtained from N by a series of α–/β– and η–normalization steps such that $\{x_i : \varphi_i : i < n\} \vdash N' : \chi$ also is derivable. The proof of the latter formula is shorter than the first on condition that N contains a subterm that can be β– or η–reduced.

Notes on this Section. λ–abstraction already appeared in (Frege, 1962). Frege wrote $)\varepsilon.f(\varepsilon)$. The first to study abstraction systematically was Alonzo Church (see (Church, 1933)). Combinatory logic on the other hand has appeared first in the work of Moses Schönfinkel (1924) and Haskell Curry (1930). Also, in the work of Husserl one finds the idea of semantic categories. Wilfrid Hodges has picked up these ideas (see (Hodges, 2001)). More on that in Chapter 4. Suffice it to say that two elements are of the same semantic category if and only if they have the same **intent**, that is, if they can meaningfully occur in the same terms. There are exercises below on applicative structures that demonstrate that Husserl's conception characterizes exactly the types.

Exercise 95. Find combinators **G** and **C** such that $\mathbf{G}\mathtt{xyz} = \mathtt{x(zyz)}$ and $\mathbf{C}\mathtt{xyz} = \mathtt{xzy}$.

Exercise 96. Determine all types of **B** and **C** of the previous exercise.

Exercise 97. We have seen in Section 3.2 that $\varphi \to \varphi$ can be derived from (a0) and (a1). Use this proof to give a definition of **I** in terms of **K** and **S**.

Exercise 98. Show that any combinatorially complete applicative structure with more than element is infinite.

Exercise 99. Show that $\bullet$, $\mathfrak{k}$ and $\mathfrak{s}$ defined on V_ω are proper classes in V_ω. *Hint.* It suffices to show that they are infinite. However, the construction could in principle applied to any universe V_κ, so here is a more general method. Say that $C \subseteq V_\kappa$ is *rich* if for every $x \in V_\kappa$, $x \in^+ C$. Show that no set is rich. Next show that $\bullet$, $\mathfrak{k}$ and $\mathfrak{s}$ are rich.

Exercise 100. Let $\langle\{A_\alpha : \alpha \in \mathrm{Typ}_\to(C)\}, \bullet\rangle$ be a typed applicative structure. Now define the partial algebra $\langle A, \bullet\rangle$ where $A := \bigcup_\alpha A_\alpha$. Show that if the applicative structure is combinatorially complete, the type assignment is unique up to permutation of the elements of C. Show also that if the applicative structure is not combinatorially complete, uniqueness fails. *Hint.* First, establish the elements of basic type, and then the elements of type $b \to c$, where $b, c \in C$ are basic. Now, given an element of type $b \to c$ it can be applied to all and only the elements of type c. This allows to define which elements have the same basic type. Now do induction.

Exercise 101. Let $V := \{\mathtt{p}\vec{\alpha} : \vec{\alpha} \in \{\mathtt{0},\mathtt{1}\}^*\}$. Denote the set of all types of combinators that can be formed over the set V by C.

Show that C is exactly the set of intuitionistically valid formulae, that is, the set of formulae derivable in $\vdash^H$.

3.4 The Syntactic Calculus of Categories

Categorial Grammar in contrast to phrase structure grammars specifies no special set of rules, but instead associates with each lexical element a finite set of context schemata. These context schemata can either be defined over strings or over structure trees. The second approach is older and leads to the so called Ajdukiewicz–Bar Hillel–Calculus (**AB**–Calculus), the first to the Lambek–Calculus (**L**–Calculus). We present first the **AB**–Calculus.

We assume that all trees are strictly binary branching with exception of the preterminal nodes. Hence, every node whose daughter is not a leaf has exactly two daughters. The phrase structure rule $X \to YZ$ of a grammar codes a local tree as a prescription which allows us to expand X by YZ. Categorial grammar interprets this in such a way that Y and Z represent trees which are composed to a tree with root X. The approach is therefore from bottom to top. The fact that a tree of the named kind may be composed is coded by the so called **category assignment**. To this end we first have to define *categories*. This is done with the help of operation symbols over an alphabet C of so called **basic categories**. In our case these are $/$ and $\backslash$. (Later also $\bullet$ will be added.) If C is given then $\mathrm{Cat}_M(C)$ is the set of terms over C and the operation symbols from M. Thus, for now we have $M = \{\backslash, /\}$. These terms are written in infix notation. So, we write a/b in place of $/ab$. Categories are usually denoted by lower case Greek letters basic categories by lower case Latin letter. If $C = \{a, b, c\}$ then $(a/b)\backslash c$, c/a are categories. We agree on left associative bracketing as with λ–terms. Hence $a/b/c/b/a$ is short for

$$(((a/b)/c)/b)/a$$

The interpretation of categories in terms of trees is as follows. A *tree* is an exhaustively ordered strictly binary branching tree with labels in $\mathrm{Cat}_{\backslash,/}(C) := \mathrm{Tm}_{\backslash,/}(C)$, which results from a constituent analysis. This means that nonterminal nodes branch exactly when they are not preterminal. Otherwise they have a single daughter, whose label is an element of the alphabet. The labelling function t must be correct in the sense of the following definition.

Call a tree **2–standard** if a node is at most binary branching, and if it is branching if and only if it is preterminal.

Definition 3.4.1 *Let A be an alphabet and $\zeta : A_\varepsilon \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ a function for which $\zeta(a)$ is always finite. Then ζ is called a* ***category assignment****. Let $\mathfrak{T} = \langle T, <, \sqsubset, t\rangle$ be a 2–standard tree with labels in $\mathrm{Cat}_{\backslash,/}(C)$. $\mathfrak{B}$ is* ***correctly*** *ζ–**labelled** if (1.) for every non branching x with daughter y $t(x) \in \zeta(t(y))$, and (2.) for every branching x which immediately dominates y_0, y_1 and $y_0 \sqsubset y_1$ we have: $t(y_0) = t(x)/t(y_1)$ or $t(y_1) = t(y_0)\backslash t(x)$.*

Definition 3.4.2 *An* ***AB–Categorial Grammar*** *is a quadruple $K = \langle S, C, A, \zeta\rangle$ where A and C are finite sets, the* ***alphabet*** *and the set of* ***basic categories****, $S \in C$, and $\zeta : A \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ a category assignment. The set of labelled trees that is accepted by K is denoted by $L_B(K)$. It is the set of 2–standard correctly ζ–labelled trees with labelling $t : T \to \mathrm{Cat}_{\backslash,/}(C)$ such that the root carries the label S.*

We emphasize that for technical reasons also the empty string must be assigned a category. Otherwise no language which contains the empty string is a language accepted by a categorial grammar. We shall ignore this case in the sequel, but in the exercises will shed more light on it.

AB–Categorial Grammar only allows us to define the mapping ζ. For given ζ, the set of trees that are correctly ζ–labelled can be enumerated. To this end we need to simply enumerate all possible constituents. Then for each preterminal x we choose an appropriate label $\alpha \in \zeta(t(y))$, where $y \prec x$. The labelling function therefore is fixed on all other nodes. For in any local tree either the left hand daughter carries the label α/β and then the right hand daughter has the label β and the mother α or the right hand carries the label $\alpha\backslash\beta$ the left hand daughter the label α and the mother β. Hence there exists at most one labelling function that extends the given one. The algorithm for finding analysis trees is not very effective. However, despite this we can show that already a context free grammar generates all trees, which allows us to import the results on context free grammars.

Theorem 3.4.3 *Let $K = \langle S, C, A, \zeta\rangle$ be an **AB**–Categorial Grammar. Then there exists a context free grammar G such that $L_B(K) = L_B(G)$.*

Proof. Let N be the set of all subterms of terms in $\zeta(a)$, $a \in A$. It can be seen without problem that every correctly labelled tree only carries labels from N. The start symbol is that of K. The rules have the form

$$\begin{array}{lll} \alpha & \rightarrow & \alpha/\beta \quad \beta \\ \alpha & \rightarrow & \beta \quad \beta\backslash\alpha \\ \alpha & \rightarrow & a \qquad\qquad (\alpha \in \zeta(a)) \end{array}$$

where α, β run through all symbols of N and a all symbols from A. This defines $G := \langle S, N, A, R\rangle$. If $\mathfrak{B} \in L_B(G)$ then the labelling is correct, as is easily seen. Conversely, if $\mathfrak{B} \in L_B(K)$ then every local tree is an instance of a rule from G, the root carries the symbol S, and all leaves carry a terminal symbol. Hence $\mathfrak{B} \in L_B(G)$. $\Box$

Conversely every context free grammar can be converted into an **AB**–grammar; however, these grammars are not strongly equivalent, only weakly. There exists a grammar G in Greibach Form such that $L(G) = L$. We distinguish two cases. Case 1. $\varepsilon \in L$. We assume that $\mathtt{S}$ is never on the right hand side of a production. (This can be installed keeping to Greibach Form; see the exercises.) Then we choose a category assignment as in Case 2 and add $\zeta(\varepsilon) := \{\mathtt{S}\}$. Case 2. $\varepsilon \notin L$. Now define

$$\zeta_G(a) := \{X/Y_{n-1} \ldots /Y_1/Y_0 : X \to a \cdot \prod_{i<n} Y_i \in R\}$$

Put $K := \langle S, N_G, A, \zeta_G \rangle$. We claim that $L(K) = L(G)$. To this end we shall transform G by replacing the rules $\rho : X \to a \cdot \prod_{i<n} Y_i$ by the rules

$$Z_0^\rho \to aY_0, \quad Z_1^\rho \to Z_0 Y_1, \quad \ldots, \quad Z_{n-1}^\rho \to Y_{n-2} Y_{n-1} \; .$$

This defines the grammar H. We have $L(H) = L(G)$. Hence it suffices to show that $L(K) = L(H)$. In place of K we can also take a context free grammar F; the nonterminals are N_F. We show now that that F and H generate the same trees modulo the R–simulation $\sim \subseteq N_H \times N_F$, which is defined as follows. (a) For $X \in N_G$ we have $X \sim Y$ if and only if $X = Y$. (b) $Z_i^\rho \sim W$ if and only if $W = X/Y_{n-1}/ \ldots /Y_{i+1}$ and $\rho = X \to Y_0 \cdot Y_1 \cdot \ldots \cdot Y_{n-1}$ for certain Y_j, $i < j < n$. To this end it suffices to show that the rules of F correspond via $\sim$ to the rules of H. This is directly calculated.

Theorem 3.4.4 (Bar–Hillel & Gaifman & Shamir) *Let L be a language over an alphabet A. $L = L_B(K)$ for some **AB**–Categorial Grammar K if and only if L is context free.* □

Notice that we have made use of only one category defining operation, namely $/$. The reader may show that $\backslash$ would have sufficed instead.

Now we look at Categorial Grammar from the standpoint of the sign grammars. We introduce a binary operation $\bullet$ on the set of categories which satisfies the following equations.

$$\alpha/\beta \cdot \beta = \alpha, \qquad \beta \cdot \beta\backslash\alpha = \alpha$$

Hence $\beta \cdot \gamma$ is defined only when $\gamma = \beta\backslash\alpha$ or $\beta = \alpha/\gamma$ for some α. Now let us look at the construction of a sign algebra for context free grammars of Section 3.1. Because of the results of this section we can assume that the set T' is a subset of $\mathrm{Cat}_{\backslash,/}(C)$ which is closed under $\cdot$. Then for our rule modes we may define as follows. If a is of category α then there exists a context free rule $\rho : \alpha \to a$ and we introduce a 0–ary mode $\mathtt{R}_\rho := \langle a, \alpha, a\rangle$. The other rules can be condensed into a single mode

$$\mathtt{A}(\langle\vec{x}, \alpha, \vec{x}\rangle, \langle\vec{y}, \beta, \vec{y}\rangle) := \langle\vec{x}\,\vec{y}, \alpha \cdot \beta, \vec{x}\,\vec{y}\rangle\ .$$

However, this still does not generate the intended meanings. We still have to introduce $\mathtt{S}^{\heartsuit}$ as in Section 3.1. We do not want to do this, however. Instead we shall deal with the question whether one can generate the meanings in a more systematic fashion. In general this is not possible, for we have only assumed that f is computable. However, in practice it appears that the syntactic categories are in close connection to the meanings. This is the philosophy behind Montague Semantics.

Let an arbitrary set C of basic categories be given. Further, let a set B of basic types be given. From B we can form types in the sense of the typed λ–calculus and from C categories in the sense of categorial grammar. These are connected to each other by a homomorphism. To be precise, for each basic category $c \in C$ we choose a type α_c. Now we put

1. $\sigma(c) := \alpha_c$.
2. $\sigma(\alpha/\beta) := \sigma(\beta) \to \sigma(\alpha)$.
3. $\sigma(\beta\backslash\alpha) := \sigma(\beta) \to \sigma(\alpha)$.

Let now $\langle M, \Lambda, \mathsf{app}\rangle$ be a set–model of the λ–calculus. A **realization of** B is a function $\ulcorner - \urcorner$ which assigns to each $b \in B$ a set. The realization can be extended in a canonical way to the semantic types and by that indirectly to the syntactical categories. So we define

$$\begin{aligned} \ulcorner \alpha/\beta \urcorner &:= \ulcorner \beta \urcorner \to \ulcorner \alpha \urcorner, \\ \ulcorner \beta\backslash\alpha \urcorner &:= \ulcorner \beta \urcorner \to \ulcorner \alpha \urcorner. \end{aligned}$$

We perform this for our arithmetical terms. There is a basic category $\mathtt{Z}$, and we realize by the set of numbers from 0 to 9. Further there is the category $\mathtt{T}$ which we realize by the rational numbers $\mathbb{Q}$ — for example.

$$\begin{aligned} \ulcorner \mathtt{Z} \urcorner &:= \{0, 1, \ldots, 9\}\ , \\ \ulcorner \mathtt{T} \urcorner &:= \mathbb{Q}\ . \end{aligned}$$

Now $+ : \mathbb{Q} \times \mathbb{Q} \to \mathbb{Q}$ is a binary function. We can redefine it as shown in Section 3.3 to a function in $\mathbb{Q} \to (\mathbb{Q} \to \mathbb{Q})$. For simplicity's sake we write $+$ also for this function. The syntactic category which we assign to $+$ has to match this. We choose $(\mathtt{T}\backslash\mathtt{T})/\mathtt{T}$. Indeed we now have

$$\ulcorner (\mathtt{T}\backslash\mathtt{T})/\mathtt{T} \urcorner = \mathbb{Q} \to (\mathbb{Q} \to \mathbb{Q})\ ,$$

as desired. Now we have to see to it that the meaning of the string `5+7` is indeed 12. To this end we require that if `+` is combined with `7` to the constituent `+7` the meaning of `+` (which is a function) is applied to the number 7. If we do like this we get the function $x \mapsto x + 7$ on $\mathbb{Q}$. If we finally group `+7` and `5` together to a constituent then we get a constituent of category $\mathtt{T}$ whose meaning is 12.

If things are arranged in this way we can uniformly define two modes for Categorial Grammar, $\mathtt{A}_>$ and $\mathtt{A}_<$.

$$\begin{aligned} \mathtt{A}_>(\langle \vec{x}, \alpha, X\rangle, \langle \vec{y}, \beta, Y\rangle) &:= \langle \vec{x}\,\vec{y}, \alpha \cdot \beta, XY\rangle, \\ \mathtt{A}_<(\langle \vec{x}, \alpha, X\rangle, \langle \vec{y}, \beta, Y\rangle) &:= \langle \vec{x}\,\vec{y}, \alpha \cdot \beta, YX\rangle. \end{aligned}$$

Further we assume that if $a \in A$ and its category if α then there are only finitely many $X \in \ulcorner \alpha \urcorner$ which are meanings of a of category

α. For each such meaning X we assume a 0–ary mode $\langle a, \alpha, X\rangle$. Therewith Categorial Grammar is completely standardized. In the respective algebras $\mathfrak{Z}$, $\mathfrak{T}$ and $\mathfrak{M}$ a binary product, which corresponds to the concatenation of two strings. The variability is not to be found in the modes of arity > 1, only in the 0–ary modes. Therefore one speaks of Categorial Grammar as a 'lexical' theory because all information over the language is in the lexicon.

Definition 3.4.5 *A sign grammar* $\langle \mathfrak{A}, \zeta, \gamma, \mu\rangle$ *is called an* ***AB–sign grammar*** *if the signature consists of the two modes* $\mathtt{A}_>$ *and* $\mathtt{A}_<$ *and finitely many 0–ary modes* $\mathtt{M}_i$*,* $i \in n$*, additionally (not counting the 0–ary modes)* $\mathfrak{Z} = \langle A^*, \cdot\rangle$*, and such that* $\mathfrak{T} = \langle U, \cdot\rangle$ *is a subalgebra of the algebra of category over the basic set* C*, and* $\mathfrak{M} = \langle M, \mathsf{app}\rangle$ *an applicative structure. Further, if* $\mathtt{M} = \langle \vec{x}, \alpha, X\rangle$ *is a 0–ary mode then* $\sigma(\alpha)$ *is the type of* X*.*

Normally, one takes $\mathfrak{M}$ to be a full theory of sets and function application, even though it contains more functions than necessary. Notice that the algebra of meanings is partial and has as its unique operation function application. (This is not defined if the categories do not match.) To be exact we should have to introduce the 0–ary functions (the $\mathtt{M}_i^\zeta$, $\mathtt{M}_i^\gamma$, $\mathtt{M}_i^\mu$). However, in this form the definition is more compact. As we shall see, the conception of a Categorial Grammar is somewhat restrictive with respect to the language generated (it has to be context free) and with respect to the categorial symbols, but it is not restrictive with respect to meanings. We leave it to the reader to show this.

We shall demonstrate this by means of an example. We look at our alphabet of ten digits. To every nonempty string over this alphabet corresponds a unique number, which we name by this very sequence. For example, the sequence `721` denotes the number 721 which we write in binary `101101001` or `LOLLOLOOL`. We want to write a Categorial Grammar which couples a string of digits with its number. This is not as easy as it appears at first sight. In order not to let the example appear trivial we shall write a grammar for binary numbers, with `L` in place of 1 and `O` in place of 0. To start, we need a category $\mathtt{Z}$ as in the example above. This category is realized by the set of natural numbers. Every digit has the category $\mathtt{Z}$. Then we have the following 0–ary modes.

$$\begin{aligned} \mathtt{Z}_0 &:= \langle \mathtt{0}, Z, 0\rangle, \\ \mathtt{Z}_1 &:= \langle \mathtt{L}, Z, 1\rangle. \end{aligned}$$

Now we additionally agree that digits have the category $\mathtt{Z}\backslash\mathtt{Z}$. With this the number LOL is analyzed in this way.

$$\dfrac{\dfrac{\overset{\mathtt{L}}{Z} \quad \overset{\mathtt{0}}{Z\backslash Z}}{Z} \quad \overset{\mathtt{L}}{Z\backslash Z}}{Z}$$

This means that digits are semantically speaking also functions from ω to ω. As one easily finds out these are the functions $\lambda\mathtt{x}_0.\mathtt{(2x_0+}k\mathtt{)}$, $k \in \{0,1\}$. Here k must be the value of the digit. So, additionally we need the following unary modes.

$$\begin{aligned} \mathtt{M}_0 &:= \langle \mathtt{0}, Z\backslash Z, \mathtt{(\lambda x_0.2x_0)}\rangle \\ \mathtt{M}_1 &:= \langle \mathtt{0}, Z\backslash Z, \mathtt{(\lambda x_0.2x_0+1)}\rangle. \end{aligned}$$

However, the grammar does not have the ideal form. For every digit has two different meanings which do not need to have anything to do with each other. For example, we could have introduced the following mode.

$$\mathtt{M}_2 := \langle \mathtt{0}, Z\backslash Z, \mathtt{(\lambda x_0.(2x_0+1))}\rangle \ .$$

We can avoid this by introducing a second category symbol, $\mathtt{T}$, which stands for a sequence of digits, while $\mathtt{Z}$ only stands for digits. In place of $\mathtt{M}_0$ we now define a unary mode $\mathtt{N}_0$ and a binary mode $\mathtt{N}_1$:

$$\begin{aligned} \mathtt{N}_0 &:= \langle \varepsilon, \mathtt{T/Z}, \mathtt{(\lambda x_0.x_0)}\rangle \\ \mathtt{N}_1 &:= \langle \varepsilon, \mathtt{T/Z/T}, \mathtt{(\lambda x_1.(\lambda x_0.(2x_1+x_0)))}\rangle \end{aligned}$$

For example, we get LOL as the exponent of the term

$$\mathtt{N_1N_1N_0Z_1Z_0Z_1} \ .$$

The value of this term is calculated as follows.

$$\begin{aligned}
(\mathtt{N}_1\mathtt{N}_1\mathtt{N}_0\mathtt{Z}_1\mathtt{Z}_0\mathtt{Z}_1)^\mu &= \mathtt{N}_1^\mu(\mathtt{N}_1^\mu(\mathtt{N}_0^\mu(\mathtt{Z}_1^\mu))(\mathtt{Z}_0^\mu))(\mathtt{Z}_1^\mu) \\
&= \mathtt{N}_1^\mu(\mathtt{N}_1^\mu((\mathtt{N}_0^\mu(1))(0))(1) \\
&= \mathtt{N}_1^\mu(\mathtt{N}_1^\mu((\lambda\mathtt{x}_0\,.\,\mathtt{x}_0)(1))(0))(1) \\
&= \mathtt{N}_1^\mu(\mathtt{N}_1^\mu(1)(0))(1) \\
&= \mathtt{N}_1^\mu(((\lambda\mathtt{x}_1\,.\,(\lambda\mathtt{x}_0\,.\,(2\mathtt{x}_1\mathtt{+}\mathtt{x}_0)))(1)(0))(1) \\
&= \mathtt{N}_1^\mu(2)(1) \\
&= (\lambda\mathtt{x}_1\,.\,\lambda\mathtt{x}_0\,.\,(2\mathtt{x}_1\mathtt{+}\mathtt{x}_0)))(2)(1) \\
&= 5
\end{aligned}$$

This solution is far more elegant than the first. Despite of this the result is not satisfactory. We had to postulate additional modes which one cannot see on the string. The easiest would be if we could define a binary mode whose meaning is the function g. In Categorial Grammar this is not allowed. This is a drawback, which we shall remove in the next chapter. A further problem is the restricted functionality in the realm of strings. With the example of the grammar T of the previous section we shall exemplify this. We have agreed that every term is enclosed by brackets, which merely are devices to help the eye. These brackets are now symbols of the alphabet, but void of real meaning. To place the brackets correctly, some effort must be made. We propose the following grammar.

$$\begin{aligned}
\mathtt{O}_1 &:= \langle +, (T\backslash U)/T, (\lambda\mathtt{x}_1\,.\,(\lambda\mathtt{x}_0\,.\,(\mathtt{x}_0\mathtt{+}\mathtt{x}_1)))\rangle \\
\mathtt{O}_2 &:= \langle -, (T\backslash U)/T, (\lambda\mathtt{x}_1\,.\,(\lambda\mathtt{x}_0\,.\,(\mathtt{x}_0\mathtt{-}\mathtt{x}_1)))\rangle \\
\mathtt{O}_3 &:= \langle \div, (T\backslash U)/T, (\lambda\mathtt{x}_1\,.\,(\lambda\mathtt{x}_0\,.\,(\mathtt{x}_0\mathtt{/}\mathtt{x}_1)))\rangle \\
\mathtt{O}_4 &:= \langle \times, (T\backslash U)/T, (\lambda\mathtt{x}_1\,.\,(\lambda\mathtt{x}_0\,.\,(\mathtt{x}_0\mathtt{x}_1)))\rangle \\
\mathtt{O}_5 &:= \langle -, U/T, (\lambda\mathtt{x}_0\,.\,(\mathtt{-}\mathtt{x}_0))\rangle \\
\mathtt{O}_6 &:= \langle \mathtt{(}, L/U, (\lambda\mathtt{x}_0\,.\,\mathtt{x}_0)\rangle \\
\mathtt{O}_7 &:= \langle \mathtt{)}, L\backslash T, (\lambda\mathtt{x}_0\,.\,\mathtt{x}_0)\rangle \\
\mathtt{Z}_0 &:= \langle \mathtt{L}, T, 0\rangle \\
\mathtt{Z}_1 &:= \langle \mathtt{0}, T, 1\rangle
\end{aligned}$$

The conception is that an operation symbol generates an unbracketed term which needs a left and a right bracket to become a 'real' term. A semantics that fits with this analysis will assign the meanings to all these. We simply take $\mathbb{Q}$ for all basic categories. The brackets are interpreted by the identity function. If we add a bracket, nothing happens to the value of the term. This is a viable solution. However, it amplifies the set of basic categories without any increase in semantic types as well.

The application of a function to an argument is by far not the only possible rule of composition. In particular Peter Geach has proposed in (Geach, 1972) to admit further rules of combination. This idea has been realized on the one hand in the Lambek–Calculus, which we will study later, and also in **Combinatory Categorial Grammar**. The idea to the latter is as follows. Each mode in the Categorial Grammar is interpreted by a semantical typed combinator. For example, $\mathtt{A}_<$ acts on the semantics like the combinator **U** (defined in Section 3.3) and $\mathtt{A}_>$ is interpreted by the combinator **I**. This choice of combinators is — seen from the standpoint of combinatory logic — only one of many possible choices. Let us look at other possibilities. We could add to the ones we have also the following closed λ–term.

$$\mathsf{B} := (\lambda\mathtt{x}_0 . (\lambda\mathtt{x}_1 . (\lambda\mathtt{x}_2 . (\mathtt{x}_0(\mathtt{x}_1\mathtt{x}_2))))) .$$

$\mathsf{B}MN$ is nothing but function composition of the functions M and N. For evidently, if $\mathtt{x}_2$ has type γ then $\mathtt{x}_1$ must have the type $\beta \to \gamma$ for some β and $\mathtt{x}_0$ the type $\alpha \to \beta$ for some α. Then $\mathsf{B}\mathtt{x}_0\mathtt{x}_1 \triangleright (\lambda\mathtt{x}_2 . (\mathtt{x}_0(\mathtt{x}_1\mathtt{x}_2)))$ is of type $\alpha \to \gamma$. Notice that for each α, β and γ we have a typed λ–term $\mathsf{B}_{\alpha,\beta,\gamma}$.

$$\mathsf{B}_{\alpha,\beta,\gamma} := (\lambda\mathtt{x}^0_{\beta\to\gamma} . (\lambda\mathtt{x}^1_{\alpha\to\beta} . (\lambda\mathtt{x}^2_{\alpha} . (\mathtt{x}^0_{\beta\to\gamma}(\mathtt{x}^1_{\alpha\to\beta}\mathtt{x}^2_{\alpha}))))) .$$

However, as we have explained earlier, we shall not use the explicitly typed terms, but rather resort to the implicitly typed terms (or combinators). We define a new category product $\circ$ by

$$\begin{array}{cccc}
\gamma/\beta & \circ & \beta/\alpha & := & \gamma/\alpha \\
\beta/\alpha & \circ & \beta\backslash\gamma & := & \gamma/\alpha \\
\gamma/\beta & \circ & \alpha\backslash\beta & := & \alpha\backslash\gamma \\
\alpha\backslash\beta & \circ & \beta\backslash\gamma & := & \alpha\backslash\gamma
\end{array}$$

Further we define two new modes, $\mathtt{B}_>$ and $\mathtt{B}_<$, as follows:

$$\begin{array}{rcl} \mathtt{B}_>(\langle\vec{x},\alpha,M\rangle,\langle\vec{y},\beta,N\rangle) & := & \langle\vec{x}\cdot\vec{y},\alpha\circ\beta,\mathsf{B}MN\rangle \\ \mathtt{B}_<(\langle\vec{x},\alpha,M\rangle,\langle\vec{y},\beta,N\rangle) & := & \langle\vec{x}\cdot\vec{y},\alpha\circ\beta,\mathsf{B}NM\rangle \end{array}$$

Here, it is not required that the type of M matches α in any way, or the type of N the category β. In place of $\mathsf{B}NM$ we could have used $\mathsf{V}MN$, where

$$\mathsf{V} := (\lambda \mathtt{x}^0 . (\lambda \mathtt{x}^1 . (\lambda \mathtt{x}^2 . (\mathtt{x}^1(\mathtt{x}^0\mathtt{x}^2)))))$$

We denote by CCG(B) the extension of the **AB**–Calculus by the implicitly typed combinator B. This grammar not only allows for the modes $\mathtt{A}_>$ and $\mathtt{A}_<$ but also the modes $\mathtt{B}_>$ and $\mathtt{B}_<$. The resulting tree sets are however of a new kind. For now, if x is branching with daughters y_0 and y_1, x can have the category α/γ if y_0 has the category α/β and y_1 the category β/γ. In the definition of the product $\circ$ there is a certain arbitrariness. What we must expect from the semantic typing regime is that the type of $\sigma(\alpha\circ\beta)$ equals $\eta \to \theta$ if $\sigma(\alpha) = \zeta \to \theta$ and $\sigma(\beta) = \eta \to \zeta$ for some η, ζ and θ. Everywhere else the syntactic product should be undefined. However, in fact the syntactic product has been symmetrified, and the directions specified. This goes as follows. By applying a rule a category (here ζ) is cancelled. In the category η/θ the directionality (here: right) is viewed as a property of the argument, hence of θ. If θ is not cancelled, we must find θ being selected to the right again. If, however, it is cancelled from η/θ, then the latter must be to the left of its argument, which contains some occurrence of θ (as a result, not as an argument). This yields the rules as given. We leave it to the reader to show that the tree sets that can be generated from an initial category assignment ζ are again all context free. Hence, not much seems to have been gained. We shall now study another extension, CCG(P). Here

$$\mathsf{P} := (\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . (\lambda \mathtt{x}_2 . (\lambda \mathtt{x}_3 . (\mathtt{x}_0(\mathtt{x}_1\mathtt{x}_2)\mathtt{x}_3))))) \ .$$

In order for this to be properly typed we may freely choose the type of $\mathtt{x}_2$ and $\mathtt{x}_3$, say β and γ. Then $\mathtt{x}_1$ is of type $\gamma \to (\beta \to \delta)$

Table 3.2: The Product $\bullet$

$$\begin{array}{llllll}
\alpha/\delta & \bullet & (\delta/\beta)/\gamma & := & (\alpha/\beta)/\gamma \\
(\delta/\beta)/\gamma & \bullet & \delta\backslash\alpha & := & (\alpha/\beta)/\gamma \\
\alpha/\delta & \bullet & (\beta\backslash\delta)/\gamma & := & (\beta\backslash\alpha)/\gamma \\
(\beta\backslash\delta)/\gamma & \bullet & \delta\backslash\alpha & := & (\beta\backslash\alpha)/\gamma \\
\alpha/\delta & \bullet & \gamma\backslash(\delta/\beta) & := & \gamma\backslash(\alpha/\beta) \\
\gamma\backslash(\delta/\beta) & \bullet & \delta\backslash\alpha & := & \gamma\backslash(\alpha/\beta) \\
\alpha/\delta & \bullet & \gamma\backslash(\beta\backslash\delta) & := & \gamma\backslash(\beta\backslash\alpha) \\
\gamma\backslash(\beta\backslash\delta) & \bullet & \delta\backslash\alpha & := & \gamma\backslash(\beta\backslash\alpha)
\end{array}$$

for some δ and $\mathtt{x}_0$ of type $\delta \to \alpha$ for some α. $\mathtt{x}_1$ stands for an at least binary function, $\mathtt{x}_0$ for a function that needs at least one argument. If the combinator is defined, the mode is fixed if we additionally fix the syntactic combinatorics. To this end we define the product $\bullet$ as in Table 3.2. Now we define the following new modes:

$$\begin{array}{lll}
\mathtt{P}_>(\langle\vec{x}, \alpha, M\rangle, \langle\vec{y}, \beta, N\rangle) & := & \langle\vec{x}\cdot\vec{y}, \alpha\bullet\beta, \mathsf{P}MN\rangle \\
\mathtt{P}_<(\langle\vec{x}, \alpha, M\rangle, \langle\vec{y}, \beta, N\rangle) & := & \langle\vec{x}\cdot\vec{y}, \alpha\bullet\beta, \mathsf{P}NM\rangle
\end{array}$$

We shall study this type of grammar somewhat closer. We take the following modes.

$$\begin{array}{lll}
\mathtt{M}_0 & = & \langle \mathtt{A}, (c/a)/c, \mathtt{(\lambda x_0\,.\,(\lambda x_1\,.\,(x_0{+}x_1)))}\rangle \\
\mathtt{M}_1 & = & \langle \mathtt{B}, (c/b)/c, \mathtt{(\lambda x_0\,.\,(\lambda x_1\,.\,(x_0 x_1)))}\rangle \\
\mathtt{M}_2 & = & \langle \mathtt{a}, a, 1\rangle \\
\mathtt{M}_3 & = & \langle \mathtt{b}, b, 2\rangle \\
\mathtt{M}_4 & = & \langle \mathtt{C}, c/a, \mathtt{(\lambda x_0\,.\,x_0)}\rangle
\end{array}$$

Take the string `ABACaaba`. It has the following analysis.

$$
\begin{array}{cccccccc}
\mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{C} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} \\
(c/a)/c & (c/b)/c & (c/a)/c & \underline{c/a \quad a} & & a & b & a \\
\vdots & \vdots & \underline{(c/a)/c \qquad c} & & & \vdots & \vdots & \vdots \\
\vdots & \vdots & & \underline{c/a \qquad\qquad a} & & & \vdots & \vdots \\
\vdots & \underline{(c/b)/c \qquad\qquad\qquad c} & & & & & \vdots & \vdots \\
\vdots & & \underline{c/b \qquad\qquad\qquad\qquad b} & & & & & \vdots \\
\underline{(c/a)/c \qquad\qquad\qquad\qquad c} & & & & & & & \vdots \\
& & \underline{c/a \qquad\qquad\qquad\qquad\qquad\qquad a} & & & & & \\
& & & & c & & &
\end{array}
$$

The meaning is again 5. In this analysis only the mode $\mathtt{A}_>$ has been used. Now the following analysis exists as well.

$$
\begin{array}{cccccccc}
\mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{C} & \mathtt{a} & \mathtt{a} & \mathtt{b} & \mathtt{a} \\
(c/a)/c & \underline{(c/b)/c \quad (c/a)/c} & & c/a & a & a & b & a \\
\vdots & \underline{((c/b)/a)/c} & & \underline{c/a \quad a} & & \vdots & \vdots & \vdots \\
\underline{(((c/a)/b)/a)/c \qquad\qquad c} & & & & & \vdots & \vdots & \vdots \\
& \underline{((c/a)/b)/a \qquad\qquad\qquad a} & & & & & \vdots & \vdots \\
& & \underline{(c/a)/b \qquad\qquad\qquad b} & & & & & \vdots \\
& & & \underline{c/a \qquad\qquad\qquad\qquad a} & & & & \\
& & & & & c & &
\end{array}
$$

Here as well we get the meaning 5, as it should be. This analysis uses the mode $\mathtt{P}_>$. Notice that in the course of the derivation the categories get larger and larger (and therefore also the types).

Theorem 3.4.6 *There exist CCG(**P**)–grammars which generate non context free tree sets.*

We shall show that the grammar just defined is of this kind. To this end we shall make a few more considerations.

Lemma 3.4.7 *Let* $\alpha = \eta_1/\eta_2/\eta_3$, $\beta = \eta_3/\eta_4/\eta_5$ *and* $\gamma = \eta_5/\eta_6/\eta_7$. *Then*

$$\alpha \bullet (\beta \bullet \gamma) = (\alpha \bullet \beta) \bullet \gamma.$$

Proof. Proof by direct computation. For example, $\alpha \bullet \beta = \eta_1/\eta_2/\eta_3/\eta_4/\eta_5$. □

In particular, $\bullet$ is associative if defined (in contrast to $\cdot$). Now, let us look at a string of the form $\vec{x}\mathtt{Ca}\vec{y}$, where $\vec{x} \in (\mathtt{A} \cup \mathtt{B})^*$, $\vec{y} \in (\mathtt{a} \cup \mathtt{b})^*$ and $h(\vec{x}) = \vec{y}^T$, where $h : \mathtt{A} \mapsto \mathtt{a}, \mathtt{B} \mapsto \mathtt{b}$. An example is the string AABACabaaa. Then we have: with the exception of $\vec{x}\mathtt{C}$ all prefixes are constituents. For prefixes of $\vec{x}$ are constituents, as one can easily see. If this is so, then it follows easily that the tree sets are not context free. For if $\vec{x} \neq \vec{y}$ then $\vec{x}\mathtt{Ca}h(\vec{y}^T)$ is not derivable. However, $\vec{x}\mathtt{Ca}h(\vec{x}^T)$ is derivable. If the tree set was context free, there cannot be infinitely many such $\vec{x}$, a contradiction.

So, we have already surpassed the border of context freeness. However, we can push this up still further. Let $\mathfrak{N}$ be the following grammar.

$$\begin{array}{lcl}
\mathtt{N}_0 & = & \langle \mathtt{A}, c\backslash(c/a), (\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . (\mathtt{x}_0 + \mathtt{x}_1))) \rangle \\
\mathtt{N}_1 & = & \langle \mathtt{B}, c\backslash(c/b), (\lambda \mathtt{x}_0 . (\lambda \mathtt{x}_1 . (\mathtt{x}_0 \cdot \mathtt{x}_1))) \rangle \\
\mathtt{N}_2 & = & \langle \mathtt{a}, a, 1 \rangle \\
\mathtt{N}_3 & = & \langle \mathtt{b}, b, 2 \rangle \\
\mathtt{N}_4 & = & \langle \mathtt{C}, c, (\lambda \mathtt{x}_0 . \mathtt{x}_0) \rangle
\end{array}$$

Theorem 3.4.8 $\mathfrak{N}$ *generates a non context free language.*

Proof. Let L be the language generated by $\mathfrak{N}$. Put $M := \mathtt{C}(\mathtt{A} \cup \mathtt{B})^*(\mathtt{a} \cup \mathtt{b})^*$. If L is context free, so is $L \cap M$ (by Theorem 2.1.14). Define h by $h(\mathtt{A}) := h(\mathtt{a}) := \mathtt{a}$, $h(\mathtt{B}) := h(\mathtt{b}) := \mathtt{b}$ as well as $h(\mathtt{C}) := \varepsilon$. We show:

$(*)$ $\quad \vec{x} \in L \cap M$ if and only if $\vec{x} \in L$ and $h(\vec{x}) = \vec{y}\vec{y}$ for some $\vec{y} \in (\mathtt{a} \cup \mathtt{b})^*$.

Hence $h[L \cap M] = \{\vec{y}\,\vec{y} : \vec{y} \in (\mathtt{a} \cup \mathtt{b})^*\}$. The latter is not context free. From this follows by Theorem 1.5.11 that $L \cap M$ is not context free, hence L is not context free either. Now for the proof of $(*)$. If $\Delta = \langle \delta_i : i < n \rangle$ then let c/Δ denote the category $c/\delta_0/\delta_1/\ldots/\delta_{n-1}$. Then we have:

$$c\backslash(c/\Delta_1) \bullet c\backslash(c/\Delta_2) = c\backslash(c/\Delta_2; \Delta_1)$$

Now let $\mathtt{C}\vec{x}\vec{y}$ be such that $\vec{x} \in (\mathtt{A}\cup\mathtt{B})^*$ and $\vec{y} \in (\mathtt{a}\cup\mathtt{b})^*$. It is not hard to see that then $\mathtt{C}\vec{x}$ is a constituent. Now let $\vec{x} := x_0 x_1 \ldots x_{n-1}$. Further, let $d_i := a$ if $x_i = \mathtt{A}$ and $d_i := b$ if $x_i = \mathtt{B}$, $i < n$. Then the category of $\vec{x}$ equals $c\backslash(c/\Delta)$ with $\Delta = \langle d_i : i < n \rangle$. Hence $\mathtt{C}\vec{x}$ is a constituent of category c/Δ. This means, however, that y_0 has the category d_0, y_1 the category d_1 and so on. But if y_i has the category d_i then $h(x_i) = y_i$, as is easily checked. This yields that $h(\vec{x}) = \vec{y}$. If on the other hand this is the case, the string is derivable. □

Hence we now have a grammar which generates non context free languages. Combinatory Categorial Grammars are therefore stronger than **AB**–Categorial Grammars.

There is a still different way to introduce CCGs. There we do not enlarge the set of combinatorial rules but instead introduce empty modes.

$$\begin{array}{lcl}
\mathsf{B}_0 & := & \langle \varepsilon, \gamma/\alpha/(\gamma/\beta)/(\beta/\alpha), \mathsf{B} \rangle \\
\mathsf{B}_1 & := & \langle \varepsilon, (\alpha\backslash\gamma)/(\gamma/\beta)/(\alpha\backslash\beta), \mathsf{B} \rangle \\
\mathsf{B}_2 & := & \langle \varepsilon, \gamma/\alpha/(\beta\backslash\gamma)/(\beta/\alpha), \mathsf{V} \rangle \\
\mathsf{B}_3 & := & \langle \varepsilon, (\alpha\backslash\gamma)/(\beta\backslash\gamma)/(\alpha\backslash\gamma), \mathsf{V} \rangle
\end{array}$$

Here we do not have four but infinitely many modes, one for each choice of α, β and γ. Only in this way it is possible to generate non context free languages. Lexical elements that have a parametric set of categories (together with parametric meanings) are called **polymorphic**. Particularly interesting cases of polymorphic elements are the logical connectors, `and` and `not`. Syntactically, they have the category $(\alpha\backslash\alpha)/\alpha$ and α/α, respectively, where α can assume any (non parametric) category. This means that two constituents of identical category can be conjoined by `and` to another constituent of the same category, and every constituent can be turned by `not` to a constituent of identical category.

Notes on this section. Although we have said that the meanings shall be functions in an applicative structure, we have effectively put strings in their place which denote these functions. This is not an entirely harmless affair. For example, the string $(\lambda\mathtt{x}_0.\mathtt{x}_0\mathtt{+1})$ and the string $(\lambda\mathtt{x}_1.\mathtt{x}_1\mathtt{+1})$ denote the same function. In fact, for reduced terms terms uniqueness holds only up to renaming of bound variables. It is standard practice in λ–calculus to consider λ–terms 'up to renaming of bound variables' (see (Pigozzi and Salibra, 1995) for a discussion). A possible remedy might be to use combinators. But here the same problem arises. Different strings may denote the same function. This is why normalisation becomes important. On the other hand, strings as meanings have the advantage to be finite, and thus may function as objects that can be stored (like codes of a Turing machine, see the discussion of Section 4.1).

Exercise 102. Let $\zeta : A_\varepsilon \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ a category assignment. Show that the correctly labelled trees form a context free tree set.

Exercise 103. Show that for every context free grammar there exists a weakly equivalent grammar in Greibach form, where the start symbol $\mathtt{S}$ does not occur on the right hand side of a production.

Exercise 104. Let $\zeta : A_\varepsilon \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ be a category assignment. Further, let $\mathtt{S}$ be the distinguished category. ζ' is called **normal** if $\zeta(\varepsilon) = \mathtt{S}$ and no $\zeta(a)$ contains an α of the form $\gamma/\beta_0/\dots/\beta_{n-1}$ with $\beta_i = \mathtt{S}$ for some $i < n$. Show that for any ζ there is a normal ζ' such that ζ' and ζ have the same language.

Exercise 105. Let $S \subseteq A^*$ be context free and $f : A^* \to M$. Write an **AB**–sign grammar whose interpreted language is the set of all $\langle \vec{x}, f(\vec{x})\rangle$.

Exercise 106. Let $\langle \mathfrak{A}, \zeta, \gamma, \mu\rangle$ be an **AB**–sign grammar. Show for all signs $\langle \vec{x}, \alpha, X\rangle$ generated by that grammar: X has the type $\sigma(\alpha)$. *Hint.* Induction on the length of the structure term.

Exercise 107. Show that the CCG($\mathbf{B}$) grammars only generate context free string languages, even context free tree sets. *Hint.* Show the following: if A is an arbitrary finite set of categories, then with $\mathbf{B}$ one can generate at most $|A|^n$ many categories.

3.5 The AB–Calculus

We shall now present a calculus to derive all the valid derivability statements for **AB**–Categorial Grammar. Notice that the only variable element is the elementary category assignment. We choose an alphabet A and an elementary category assignment ζ. We write $[\alpha]_\zeta$ for the set of all unlabelled binary constituent structures over A that have root category α under some correct ζ–labelling. Since **AB**–Categorial Grammars are invertible, for any given constituent structure there exists at most one labelling function (with the exception of the terminal labels). Now we introduce a binary symbol $\circ$, which takes as arguments correctly ζ–labelled constituent structures. Let $\langle X, \mathfrak{X}\rangle$ and $\langle Y, \mathfrak{Y}\rangle$ such constituent structures and $X \cap Y = \varnothing$. Then let

$$\langle X, \mathfrak{X}\rangle \circ \langle Y, \mathfrak{Y}\rangle := \langle X \cup Y, \mathfrak{X} \cup \mathfrak{Y} \cup \{X \cup Y\}\rangle$$

(In principle, $\circ$ is well defined also if the constituent structures are not binary branching.) In case where $X \cap Y \neq \varnothing$ one has to proceed to the disjoint sum. We shall not spell out the details. With the help of $\circ$ we shall form terms over A, that is, we form the algebra freely generated by A by means of $\circ$. To every term we inductively associate a constituent structure in the following way.

$$\begin{array}{rcl} a^k & := & \langle \{a\}, \{\{a\}\}\rangle \\ (s \circ t)^k & := & s^k \circ t^k \end{array}$$

Notice that $\circ$ has been used with two meanings. Finally, we take a look at $[\alpha]_\zeta$. It denotes classes of binary branching constituent structures over A. The following holds.

$$\begin{array}{rcl} [\alpha/\beta]_\zeta \circ [\beta]_\zeta & \subseteq & [\alpha]_\zeta \\ [\beta]_\zeta \circ [\beta\backslash\alpha]_\zeta & \subseteq & [\alpha]_\zeta \end{array}$$

We abstract now from A and ζ. In place of interpreting $\circ$ as a constructor for constituent structures over A we now interpret it as a constructor to form constituent structures over $\mathrm{Cat}_{\backslash,/}(C)$ for some given C. We call a term τ which has been built from categories with the help of $\circ$ a **structure term**. Categories are denoted by lower case Greek letters, structures by upper case Greek letters. Inductively, we extend the interpretation $[-]_\zeta$ to structures as follows.

$$[\Gamma \circ \Delta]_\zeta := [\Gamma]_\zeta \circ [\Delta]_\zeta \ .$$

Next we introduce yet another symbol, $\vdash$. This is a relation between structures and categories. If Γ is a structure and α a category then $\Gamma \vdash \alpha$ denotes the fact that for every interpretation in some alphabet A with category assignment ζ $[\Gamma]_\zeta \subseteq [\alpha]_\zeta$. We call the object $\Gamma \vdash \alpha$ a **sequent**. The interpretation that we get in this way we call the **cancellation interpretation**. Here categories are inserted as concrete labels which are assigned to nodes and which are subject to the cancellation interpretation.

We shall now introduce two different calculi, one of which will be shown to be adequate for the cancellation interpretation. In formulating the rules we use the following convention. $\Gamma[\alpha]$ means in this connection that Γ is a structure in which we have fixed a single occurrence of α. When we write, for example, $\Gamma[\Delta]$ below the line, then this denotes the result of replacing that occurrence of α by Δ.

$$\begin{array}{llll}
(\mathrm{ax}) & \alpha \vdash \alpha & (\mathrm{cut}) & \dfrac{\Gamma \vdash \alpha \qquad \Delta[\alpha] \vdash \beta}{\Delta[\Gamma] \vdash \beta} \\[2ex]
(\mathbf{I}\text{–}/) & \dfrac{\Gamma \circ \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} & (/\text{–}\mathbf{I}) & \dfrac{\Gamma \vdash \alpha \qquad \Delta[\beta] \vdash \gamma}{\Delta[\beta/\alpha \circ \Gamma] \vdash \gamma} \\[2ex]
(\mathbf{I}\text{–}\backslash) & \dfrac{\alpha \circ \Gamma \vdash \beta}{\Gamma \vdash \alpha\backslash\beta} & (\backslash\text{–}\mathbf{I}) & \dfrac{\Gamma \vdash \alpha \qquad \Delta[\beta] \vdash \gamma}{\Delta[\Gamma \circ \alpha\backslash\beta] \vdash \gamma}
\end{array}$$

We denote the above calculus by **AB** + (cut), and by **AB** the calculus without (cut). Further, the calculus consisting of (ax) and the rules (\–**I**) and (/–**I**) also is called $\mathbf{AB}^-$.

Definition 3.5.1 *A **categorial sequent grammar** is a quintuple $G = \langle S, C, \zeta, A, \mathbb{S}\rangle$, where C is a finite set, the set of **basic categories**, $S \in C$ the so called **distinguished category**, A a finite set, the alphabet, $\zeta : A \to \wp(\mathrm{Cat}_{\backslash,/}(C))$ a category assignment, and $\mathbb{S}$ a sequent calculus. We write $\vdash_G \vec{x}$ if for some structure term Γ whose associated string via ζ is $\vec{x}$ we have $\overset{\mathbb{S}}{\leadsto} \Gamma \vdash S$.*

We stress here that the sequent calculi are calculi to derive sequents. A sequents corresponds to grammatical a rule, or, more precisely, the sequent $\Gamma \vdash \alpha$ expresses the fact that a structure of type *Gamma* is a constituent that has the category α by the rules of the grammar. The rules of the sequent calculus can then be seen as *metarules*, which allow to pass from one valid statement concerning the grammar to another.

Proposition 3.5.2 (Correctness) *If $\Gamma \vdash \alpha$ in $\boldsymbol{AB}^-$ then $[\Gamma]_\zeta \subseteq [\alpha]_\zeta$ for every category assignment ζ.*

AB is strictly stronger than $\mathbf{AB}^-$. Notice namely that the following sequent is derivable in **AB**:

$$\alpha \vdash (\beta/\alpha)\backslash\beta$$

In natural deduction style calculi this corresponds to the following unary rule:

$$\frac{\alpha}{(\beta/\alpha)\backslash\beta}$$

This rule is known as **type raising**, since it allows to proceed from the category α to the "raised" category $(\beta/\alpha)\backslash\beta$. Perhaps we'd better call it *category raising*, but this name is now standard. To see that it is not derivable in $\mathbf{AB}^-$ we simply note that it is not correct for the cancellation interpretation. We shall return to the question of interpretation of the calculus **AB** in the next section.

An important property of these calculi is their decidability. Given Γ and α we can decide in finite time whether or not $\Gamma \vdash \alpha$.

Theorem 3.5.3 (Cut Elimination) *There exists an algorithm to construct from a proof of a sequent $\Gamma \vdash \alpha$ in* ***AB + (cut)*** *a proof of $\Gamma \vdash \alpha$ in* ***AB****. Hence (cut) is admissible for* ***AB****.*

Corollary 3.5.4 ***AB + (cut)*** *is decidable.* □

Lemma 3.5.5 *For every proof of $\Gamma \vdash \alpha$ using exactly n cuts of degree g and p instances of cuts of degree $< g$ we can construct a proof of $\Gamma \vdash \alpha$ using at most $n - 1$ instances of (cut) of degree g and and at most $p + 2$ instances of (cut) of degree $< g$.*

Proof. Let R be a rule. In an application of a rule the structure variables of the rule are instantiated to structure terms. We call the categories occurring in them **side formulae**, all other categories **main formulae**. Let now an instance of (cut) be given. If it is the highest instance, the premisses are axioms. In this case the instance has the form.

$$\frac{\alpha \vdash \alpha \qquad \alpha \vdash \alpha}{\alpha \vdash \alpha}$$

and can be replaced by $\alpha \vdash \alpha$. Similarly in the case that only one premiss is an axiom. Now we may assume that both premisses of the cut are themselves obtained by some rules. We show what happens if the cut is the main formulae of the left premiss, and we leave it to the reader to show what happens if the cut is on a side formula of the left premiss (this case is much easier). We therefore turn immediately to the case where the cut is on a main formula of the left premiss. The first case is that the formula is introduced by (**I**–/).

$$\frac{\dfrac{\Gamma \circ \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} \qquad \Delta[\beta/\alpha] \vdash \gamma}{\Delta[\Gamma] \vdash \gamma}$$

Now look at the rule instance that is immediately above $\Delta[\beta/\alpha] \vdash \gamma$. There are several cases. Case (0). The premiss is an axiom. Then $\gamma = \beta/\alpha$, and the cut is superfluous. Case (1). β/α is a main formula of the right hand premiss. Then $\Delta[\beta/\alpha] = \Theta[\beta/\alpha \circ \Xi]$ for some Θ and Ξ, and the instance of the rule was as follows.

$$\frac{\Xi \vdash \alpha \qquad \Theta[\beta] \vdash \gamma}{\Theta[\beta/\alpha \circ \Xi] \vdash \gamma}$$

Now we can restructure the derivation as follows.

$$\frac{\dfrac{\Gamma \circ \alpha \vdash \beta \qquad \Theta[\beta] \vdash \gamma}{\Theta[\Gamma \circ \alpha] \vdash \gamma} \qquad \Xi \vdash \alpha}{\Theta[\Gamma \circ \Xi] \vdash \gamma}$$

We have $d(\Delta) = d(\Xi) + d(\Theta)$. The degree of the first cut is now

$$\begin{array}{rl} & d(\Gamma \circ \alpha) + d(\beta) + d(\Theta) + d(\gamma) \\ = & d(\Gamma) + d(\beta) + d(\alpha) + d(\Theta) + d(\gamma) \\ < & d(\Gamma) + d(\beta/\alpha) + d(\Delta) + d(\gamma) \ . \end{array}$$

The degree of the second cut is

$$\begin{array}{rl} & d(\Theta) + d(\Gamma) + d(\alpha) + d(\Xi) + d(\gamma) \\ \leq & d(\Delta) + d(\Gamma) + d(\alpha) + d(\gamma) \\ < & d(\Delta) + d(\Gamma) + d(\beta/\alpha) + d(\gamma) \ . \end{array}$$

So the cut has been replaced by two cuts of lesser degree. The cut–weight is therefore decreased. Now we assume that the formula is not a main formula of the right hand premiss. Case (2). $\gamma = \zeta/\varepsilon$ and the premiss is obtained by application of (**I**–/).

$$\frac{\Delta[\beta/\alpha] \circ \varepsilon \vdash \zeta}{\Delta[\beta/\alpha] \vdash \zeta/\varepsilon}$$

Then we replace this proof by

$$\frac{\dfrac{\dfrac{\Gamma \circ \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} \qquad \Delta[\beta/\alpha] \circ \varepsilon \vdash \zeta}{\Delta[\Gamma] \circ \varepsilon \vdash \zeta}}{\Delta[\Gamma] \vdash \varepsilon/\zeta}$$

The degree of the cut is now

$$\begin{array}{rl} & d(\Gamma) + d(\beta/\alpha) + d(\Delta) + d(\varepsilon) + d(\zeta) \\ < & d(\Gamma) + d(\beta/\alpha) + d(\Delta) + d(\gamma). \end{array}$$

Case (3). $\gamma = \varepsilon\backslash\zeta$ and has been obtained by applying the rule **(I–$\backslash$)**. Then proceed as in Case (2). Case (4). The application of the rule introduces a formula which occurs in Δ. This case is left to the reader.

Now if the left hand premiss has been obtained by **($\backslash$–I)**, then one proceeds quite analogously. So, we assume that the left hand premiss is created by an application of ($\backslash$–**I**).

$$\dfrac{\dfrac{\Gamma \vdash \alpha \quad \Delta[\beta] \vdash \gamma}{\Delta[\beta/\alpha \circ \Gamma] \vdash \gamma} \quad \Theta[\gamma] \vdash \delta}{\Theta[\Delta[\beta/\alpha \circ \Gamma]] \vdash \delta}$$

Then we can restructure the proof as follows.

$$\dfrac{\Gamma \vdash \alpha \quad \dfrac{\Delta[\beta] \vdash \gamma \quad \Theta[\gamma] \vdash \delta}{\Theta[\Delta[\beta]] \vdash \delta}}{\Theta[\Delta[\beta/\alpha \circ \Gamma]] \vdash \delta}$$

Also here one calculates that the degree of the new cut is less than the degree of the old cut. The case where the left hand premiss is created by **($\backslash$–I)** is analogously. All cases have now been looked at. ☐

AB gives a method to test structure terms for their syntactic category. Now we expect that the meanings of the terms are likewise systematically connected with a term and that we can determine the meaning of a certain string once we have found a derivation for it. So we associate semantic types to categories with the map σ as in Section 3.4. We now look at the rules of **AB**. We begin with the most simple rules. We write here $\vec{x} : \alpha : X$ to say that $\vec{x}$ is a string of category α and meaning X and call this as before a 'sign'. If $\vec{x}$ is irrelevant in the context it is omitted. To begin, we shall indeed omit the strings and concentrate on the meanings. The strings introduce problems of their own, which are independent of those that the meanings pose. For the meanings we use λ–terms, which are however only proxy for the 'real' meanings (see the discussion at the end of the preceding section). Therefore we now write $(\lambda x.xy)$ in place of $(\lambda\mathtt{x}_0.(\mathtt{x}_0\mathtt{x}_1))$. A **structure** is a term made of signs with the help of $\circ$. Sequents are pairs $\Gamma \vdash \zeta$ where Γ is a structure and ζ a sign. Then we have

$$\alpha : x_\eta \vdash \alpha : x_\eta$$

Here x_η is an arbitrary variable of type η. No relationship between η and α is required. The reason why this is to is that in order for the grammar to be compositional, we may not use syntactic information on the constituents in a direct way to derive information about their meaning.

Now first we deal with cut.

$$\text{(cut)} \qquad \frac{\Gamma \vdash \alpha : Y \qquad \Delta[\alpha : x_\eta] \vdash \beta : X}{\Delta[\Gamma] \vdash \beta : [Y/x_\eta]X}$$

This is to be interpreted as follows. If $\Delta[\alpha : x_\eta]$ is a structure with marked occurrence of α then let X be the function which is the meaning of Δ and in which there is one occurrence of x_η. This occurrence of the variable is replaced by Y. So, semantically speaking cut is substitution. The other rules are more complex.

$$(/\text{–}\mathbf{I}) \qquad \frac{\Gamma \vdash \alpha : X \qquad \Delta[\beta : x_\zeta] \vdash \gamma : Y}{\Delta[\beta/\alpha : x_{\eta\to\zeta} \circ \Gamma] \vdash \gamma : [(x_{\eta\to\zeta}X)/x_\zeta]Y}$$

This corresponds to the replacement of a primitive constituent by a complex constituent or the replacement of a value $X(x)$ by the pair $\langle X, x\rangle$. Here the variable $x_{\eta\to\zeta}$ is introduced, which stands for a function from objects of type η to objects of type ζ. The variable x_ζ has, however, disappeared. This is a serious deficit of the calculus (which has other advantages, however). We shall below develop a different calculus. Analogously for the rule $(\backslash\text{–}\mathbf{I})$.

The rules (**I**–/) and (**I**–\) can be interpreted as follows. Assume that Γ is a constituent of category α/β. Then we simply have:

$$(\mathbf{I}\text{–}/) \qquad \frac{\Gamma \circ \alpha : x_\eta \vdash \beta : X}{\Gamma \vdash \beta/\alpha : (\lambda x_\eta . X)}$$

Here the meaning of Γ is of the form X and the meaning of α is Y. Notice that **AB** forces us in this way to view the meaning of a word of category α/β to be a function from η–objects to ζ–objects. For it is formally required that Γ has to have the meaning of a function. We call the rules (**I**–/) and (**I**–\) also **abstraction rules**.

If we remain with our interpretation within the universe of sets of the λ–calculus this rule must be restricted. We give an example. In the rule (\–**I**) we can put $\Delta := \beta$ and $\gamma := \beta$.

$$\frac{\alpha : x_\eta \vdash \alpha : x_\eta \qquad \beta : x_\zeta \vdash \beta : x_\zeta}{\alpha/\beta : x_{\zeta\to\eta} \circ \beta : x_\zeta \vdash \eta : (x_{\zeta\to\eta} x_\zeta)}$$

Using (**I**–/), we get

$$\frac{\alpha/\beta : x_{\zeta\to\eta} \circ \beta : x_\zeta \vdash \alpha : (x_{\zeta\to\eta} x_\zeta)}{\alpha/\beta : x_{\zeta\to\eta} \vdash \alpha/\beta : (\lambda x_\zeta . (x_{\zeta\to\eta} x_\beta))}$$

Now $\lambda x_\zeta . x_{\zeta\to\eta} x_\zeta$ is a redex of $x_{\zeta\to\eta}$ and hence the two must be the same function. On the other hand, by applying (**I**–\) we get

$$\frac{\alpha/\beta : x_{\zeta\to\eta} \circ \beta : x_\zeta \vdash \alpha : (x_{\zeta\to\eta} x_\zeta)}{\beta : x_\zeta \vdash (\alpha/\beta)\backslash\alpha : (\lambda x_{\zeta\to\eta} . (x_{\zeta\to\eta} x_\zeta))}$$

This is the type raising rule which we have discussed above. If x_ζ is of type ζ we may also regard it as a function, which for given function f taking arguments of type ζ returns the value $f(x_\zeta)$. However, x_ζ is not the same function as $(\lambda x_{\zeta\to\eta} . (x_{\zeta\to\eta} x_\zeta))$. (The latter has the type $(\beta \to \alpha) \to \alpha$.) Therefore the application of the rule is incorrect in this case. Moreover, in the typed λ–calculus the equation $x_\zeta \doteq (\lambda x_{\zeta\to\eta} . (x_{\zeta\to\eta} x_\zeta))$ is invalid.

To remedy the situation we must require that the variable which we have abstracted over appears on the left hand side of $\vdash$ in the premiss as an argument variable and not as a variable of a function that is being applied to something. So, we assume the following rule.

$$(\mathbf{I}\text{–}/) \qquad \frac{\Gamma \circ \alpha : x_\eta \vdash \beta : X}{\Gamma \vdash \beta/\alpha : (\lambda x_\eta.X)}, \qquad \text{if } x_\eta \text{ is an argument variable}$$

How can one detect whether x_η is an argument variable? To this end we require that in $\mathbf{AB}^-$ the sequent $\Gamma \vdash \beta/\alpha$ is derivable. This seems paradoxical at first sight. For with this restriction the calculus is as weak as $\mathbf{AB}^-$. Why should one make use of the rule $(\mathbf{I}\text{–}/)$ if the sequent is anyway derivable? To understand this one should take note of the difference between the uninterpreted and the interpreted calculus. We allow the use of the interpreted rule $(\mathbf{I}\text{–}/)$ if $\Gamma \vdash \beta/\alpha$ is derivable in the uninterpreted calculus; or, to be more prosaic, if Γ has the category β/α and hence the type $\alpha \to \beta$. That this indeed strengthens the calculus can be seen as follows. In the interpreted $\mathbf{AB}^-$ the following sequent is not derivable.

$$\alpha/\beta : x_{\zeta\to\eta} \vdash \alpha/\beta : \lambda x_\zeta.x_{\zeta\to\eta}x_\zeta$$

The proof of this claim is left as an exercise. However, we have just derived it in **AB**.

We shall first see why this removes the problem. To this end we return to the interpretation of sequents. We emphasize that in a derivation a start sequent may occur only once. So, a λ–operator always binds exactly one occurrence of a variable. Now we define a mapping from structure terms to λ–terms in the following way.

$$\begin{array}{lcl} h(\alpha : f) & := & f \\ h(\Gamma \circ \Delta) & := & \begin{cases} h(\Gamma)h(\Delta) & \text{if this is well defined,} \\ h(\Delta)h(\Gamma) & \text{if this is well defined,} \\ \text{undefined} & \text{otherwise.} \end{cases} \end{array}$$

A **proof** is a tree labelled with sequents such that (a) the leaves are labelled by start sequents, (b) for every nonleaf x the sequent at x follows by an application of a rule from the sequents at the daughters of x, (c) no start sequent occurs twice.

Theorem 3.5.6 *Let $\Gamma \vdash \alpha : X$ be derivable in $\boldsymbol{AB}$. Then $h(\Gamma)$ is defined and has the type $\sigma(\alpha)$, and $h(\Gamma) \equiv X$. Furthermore, every variable occurring free in Γ also occurs free in X given that no start sequent occurs twice.*

The proof is by induction on the length of the sequent. We essentially make use of the fact that a variable only occurs once free. There will be an example further below.

The calculus that has just been defined is not very attractive. It forces us to introduce new variables all the time in a proof. The next calculus, **N**, obviates the need for that.

$$
\begin{array}{llll}
\text{(ax)} & \alpha : x_\alpha \vdash \alpha : x_\alpha & & \\
(\mathbf{I}\text{–}/) & \dfrac{\Gamma \circ \alpha : x_\eta \vdash \beta : X}{\Gamma \vdash \beta/\alpha : \lambda x_\eta . X} & (\mathbf{E}\text{–}/) & \dfrac{\Gamma \vdash \alpha : X \quad \Delta \vdash \beta/\alpha : Y}{\Delta \circ \Gamma \vdash \beta : XY} \\
(\mathbf{I}\text{–}\backslash) & \dfrac{\alpha : x_\eta \circ \Gamma \vdash \beta : X}{\Gamma \vdash \alpha\backslash\beta : \lambda x_\eta . X} & (\mathbf{E}\text{–}\backslash) & \dfrac{\Gamma \vdash \alpha : X \quad \Delta \vdash \alpha\backslash\beta : Y}{\Gamma \circ \Delta \vdash \beta : XY}
\end{array}
$$

This calculus differs from the first in that formulae are not necessarily built up in going down. They can also be dismantled (as in the rules (**E**–/) and (**E**–\)). In this calculus, cut is admissible as well. The proof is left as an exercise. Also here we must assume that no two start sequents share a variable. Otherwise, the calculus yields incorrect results. For example, the following sequent would otherwise be derivable.

$$\beta/\alpha/\alpha : X \circ \alpha : x_\eta \vdash \beta/\alpha : \lambda x_\eta.((X x_\eta) x_\eta)$$

The type of the λ–term on the right hand side is ζ if X is of type $\eta \to (\eta \to \zeta)$. However, its category is β/α, and so the type should be $\eta \to \zeta \neq \zeta$. Roughly speaking, this corresponds to the situation where a syntactic argument is semantically absent because a different binder has already bound the corresponding argument variable.

Now let us turn to strings. Now we omit the interpretation, since it has been dealt with. Our objects are now written as $\vec{x} : \alpha$ where $\vec{x}$ is a string and α a category. The reader is reminded of the fact that $\vec{y}/\vec{x}$ denotes that string which results from $\vec{y}$ by removing the postfix $\vec{x}$. This is clearly defined only if $\vec{y} = \vec{u} \cdot \vec{x}$ for some $\vec{u}$, and then we have $\vec{y}/\vec{x} = \vec{u}$. Analogously for $\vec{x}\backslash\vec{y}$.

$$
\begin{array}{llll}
\text{(ax)} & \vec{x} : \alpha \vdash \vec{x} : \alpha & & \\
(\mathbf{I}\text{–}/) & \dfrac{\Gamma \circ \vec{x} : \alpha \vdash \vec{y} : \beta}{\Gamma \vdash \vec{y}/\vec{x} : \beta/\alpha} & (\mathbf{E}\text{–}/) & \dfrac{\Gamma \vdash \vec{x} : \alpha \quad \Delta \vdash \vec{y} : \beta/\alpha}{\Delta \circ \Gamma \vdash \vec{y} \cdot \vec{x} : \beta} \\
(\mathbf{I}\text{–}\backslash) & \dfrac{\vec{x} : \alpha \circ \Gamma \vdash \vec{y} : \beta}{\Gamma \vdash \vec{x}\backslash\vec{y} : \alpha\backslash\beta} & (\mathbf{E}\text{–}\backslash) & \dfrac{\Gamma \vdash \vec{x} : \alpha \quad \Delta \vdash \vec{y} : \alpha\backslash\beta}{\Gamma \circ \Delta \vdash \vec{x} \cdot \vec{y} : \beta}
\end{array}
$$

The cut rule is however no more a rule of the calculus. There is no formulation of it at all. Suppose namely we were to try to formulate a cut rule. Then it would go as follows.

$$
\text{(cut)} \qquad \frac{\Gamma \vdash \vec{x} : \alpha \qquad \Delta[\vec{y} : \alpha] \vdash \vec{z} : \beta}{\Delta[\Gamma] \vdash [\vec{y}/\vec{x}]\vec{z} : \beta}
$$

Here $[\vec{y}/\vec{x}]\vec{z}$ denotes the result of replacing $\vec{x}$ by $\vec{y}$ in $\vec{z}$. However, only one such occurrence may be replaced, so if it occurs several times, the result of this operation is not uniquely determined. The string does not reveal which of the occurrences is the right one. To remedy this, one would have to introduce variables for strings if one would insist on having the rule. In this way however it cannot be formulated. On the other hand, from a semantic point of view it is admissible, so we can do without it anyway.

We call the full calculus **NZ**. It has the rules ($\mathbf{I}$–/), ($\mathbf{I}$–\) with the restriction that only argument variables may be abstracted over, and the rules ($\mathbf{E}$–/) and ($\mathbf{E}$–\). There is no cut rule. Hence, the calculi cannot easily be compared with each other. Given an **AB**–sign grammar $\mathfrak{A}$ we can devise an **NZ**–calculus as follows. In place of the start sequents of **NZ** we only admit the following.

(ax$_{\mathfrak{A}}$) $\langle \vec{x}, \alpha, X\rangle \vdash \langle \vec{x}, \alpha, X\rangle$, where $\langle \vec{x}, \alpha, X\rangle$ is a 0–ary mode of $\mathfrak{A}$.

Normally, an **AB**–grammar does not possess any modes of the form $\langle \varepsilon, \alpha, x_\alpha \rangle$ where x_α is a variable. This is anyway the case in natural languages. It is easily seen that the rules (**I**–/) and (**I**–\) will never come to be used since x_α needs to be a variable, and these have not been admitted to $\mathfrak{A}$. It is easy to show that the so modified **NZ**–calculus and the grammar $\mathfrak{A}$ are equivalent. In the case of the **L**–Calculus, which we shall discuss in the next section, matters are different.

Exercise 108. Prove the correctness theorem, Proposition 3.5.2.

Exercise 109. Define p as follows.

$$\begin{array}{lcl} p(\alpha) & := & \alpha, \\ p(\Gamma \circ \Delta) & := & p(\Gamma) \cdot p(\Delta). \end{array}$$

Show that $\Gamma \vdash \alpha$ is derivable in $\mathbf{AB}^-$ if and only if $p(\Gamma) = \alpha$. Show that this also holds for $\mathbf{AB}^-$ + (cut). Conclude from this that the cut rule is admissible for $\mathbf{AB}^-$. (This could in principle be extracted from the proof for **AB**, but this proof here is quite simple.)

Exercise 110. Prove the completeness of **AB** with respect to the cancellation interpretation. This means that if for all A and all ζ $[\Gamma]_\zeta \subseteq [\alpha]_\zeta$ then $\Gamma \vdash \alpha$ is derivable in **AB**.

Exercise 111. Show that every context free language can be generated by an **AB**–Categorial Grammar over the basic set of categories $C = \{s, t\}$.

Exercise 112. Show the following claim: in the interpreted $\mathbf{AB}^-$–calculus no sequents are derivable which contain bound variables.

Exercise 113. Show that (cut) is admissible for **N**.

3.6 The Lambek–Calculus

The Lambek–Calculus, **L**, is in many respects an extension of **AB**. It has been introduced in (Lambek, 1958). In contrast to **AB** categories are not interpreted as sets of labelled trees but as sets of strings. This means that the calculus has different laws. Furthermore, **L** possesses a new category constructor, the *pair formation*; this constructor is written $\bullet$ and has a counterpart on the level of categories, also denoted by that symbol. The constructors of the classical Lambek–calculus for categories therefore are $\backslash$, $/$ and $\bullet$. Now let an alphabet A be given, as well as an elementary category assignment ζ. We denote by $\{\alpha\}_\zeta$ the set of all strings over A which are of category α with respect to ζ. Then the following holds.

$$\begin{array}{rcl} \{\alpha \bullet \beta\}_\zeta & := & \{\alpha\}_\zeta \cdot \{\beta\}_\zeta, \\ \{\Gamma \circ \Delta\}_\zeta & := & \{\Gamma\}_\zeta \cdot \{\Delta\}_\zeta. \end{array}$$

Since we have the constructor $\bullet$ at our disposal, we can in principle dispense with the symbol $\circ$. However, we shall not do so. We shall formulate the calculus as before using $\circ$, which makes it directly comparable to the ones we have defined above. Hence we before we distinguish *terms* from *structures*. We write $\Gamma \vdash \beta$ if $\{\Gamma\}_\zeta \subseteq \{\alpha\}_\zeta$. We shall axiomatize the sequents of $\vdash$. In order to do so we add the following rules to the calculus **AB** (without (cut)).

$$\begin{array}{llll} (\mathbf{ass1}) & \dfrac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash \alpha}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash \alpha} & (\mathbf{ass2}) & \dfrac{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash \alpha}{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash \alpha} \\ (\bullet\text{–}\mathbf{I}) & \dfrac{\Gamma[\alpha \circ \beta] \vdash \gamma}{\Gamma[\alpha \bullet \beta] \vdash \gamma} & (\mathbf{I}\text{–}\bullet) & \dfrac{\Gamma \vdash \alpha \qquad \Delta \vdash \beta}{\Gamma \circ \Delta \vdash \alpha \bullet \beta} \end{array}$$

Furthermore, $\alpha \vdash \alpha$ is the only axiom. This calculus is called the **Lambek–Calculus**, or simply **L**. The **Nonassociative Lambek–Calculus**, called also **NL**, is the calculus which in addition to the rules of **AB** only has the rules (**I**–$\bullet$) and ($\bullet$–**I**). The calculus for the cancellation interpretation is $\mathbf{L}^- = \mathbf{AB}^- + (\bullet\text{–}\mathbf{I}) + (\mathbf{I}\text{–}\bullet)$.

Theorem 3.6.1 (Lambek) *(cut) is admissible for* ***L****.*

Corollary 3.6.2 (Lambek) ***L*** *with or without (cut) is decidable.*

For a proof we only have to look at applications of (cut) in which one of the premisses of (cut) has been generated by one of the new rules. The degree of a cut is defined analogously as the number of occurrences of $/$, $\backslash$ and $\bullet$. Assume that the left hand premiss has been obtained by an application of (ass1).

$$\dfrac{\dfrac{\Gamma[\Theta_1 \circ (\Theta_2 \circ \Theta_2)] \vdash \alpha}{\Gamma[(\Theta_1 \circ \Theta_2) \circ \Theta_2] \vdash \alpha} \qquad \Delta[\alpha] \vdash \beta}{\Delta[\Gamma[(\Theta_1 \circ \Theta_2) \circ \Theta_3]] \vdash \beta}$$

This proof we reformulate into the following one.

$$\dfrac{\dfrac{\Gamma[\Theta_1 \circ (\Theta_2 \circ \Theta_3)] \vdash \alpha \qquad \Delta[\alpha] \vdash \beta}{\Delta[\Gamma[\Theta_1 \circ (\Theta_2 \circ \Theta_3)]] \vdash \beta}}{\Delta[\Gamma[(\Theta_1 \circ \Theta_2) \circ \Theta_3]] \vdash \beta}$$

Analogously if the left hand premiss has been obtained by using (ass2). We leave it to the reader to treat the case where the right hand premiss has been obtained by using (ass1) or (ass2). We have to remark here that by reformulation we do not diminish the degree of the cut. So the original proof is not easily transported into the new setting. However, the *depth* of the application has been diminished. Here, depth means (intuitively) the length of a longest path through the proof tree from the top up to the rule occurrence. If we assume that $\Gamma[\Theta_1 \circ (\Theta_2 \circ \Theta_3)] \vdash \alpha$ has depth i and $\Delta[\alpha] \vdash \beta$ depth j then in the first tree the application of (cut) has depth $max\{i, j\} + 1$, in the second however it has depth $max\{i, j\}$.

Let us look at the cases of introduction of $\bullet$. The case of ($\bullet$–**I**) on the left hand premiss is easy.

$$\dfrac{\dfrac{\Gamma[\theta_1 \circ \theta_2] \vdash \alpha}{\Gamma[\theta_1 \bullet \theta_2] \vdash \alpha} \qquad \Delta[\alpha] \vdash \gamma}{\Delta[\Gamma[\theta_1 \bullet \theta_2]] \vdash \gamma} \quad \leadsto \quad \dfrac{\dfrac{\Gamma[\theta_1 \circ \theta_2] \vdash \alpha \quad \Delta[\alpha] \vdash \gamma}{\Delta[\Gamma[\theta_1 \circ \theta_2]] \vdash \gamma}}{\Delta[\Gamma[\theta_1 \bullet \theta_2]] \vdash \gamma}$$

Now for the case of (**I**–$\bullet$) on the right hand premiss.

$$\dfrac{\Gamma \vdash \alpha \qquad \dfrac{\Theta_1 \vdash \theta_1 \qquad \Theta_2 \vdash \theta_2}{\Delta[\alpha] \vdash \gamma}}{\Delta[\Gamma] \vdash \gamma}$$

In this case $\gamma = \theta_1 \bullet \theta_2$. Furthermore, $\Delta = \Theta_1 \circ \Theta_2$ and the marked occurrence of α either is in Θ_1 or in Θ_2. Without loss of generality we assume that it is in Θ_1. Then we can replace the proof by

$$\dfrac{\dfrac{\Gamma \vdash \alpha \qquad \Theta_1[\alpha] \vdash \theta_1}{\Theta_1[\Gamma] \vdash \alpha} \qquad \Theta_2 \vdash \theta_2}{\Theta_1[\Gamma] \circ \Theta_2 \vdash \theta_1 \bullet \theta_2}$$

We have $\Theta_1[\Gamma] \circ \Theta_2 = \Delta[\Gamma]$ by hypothesis on the occurrence of α. Now we look at the case where the left hand premiss of cut has been introduced by (**I**–$\bullet$). We may assume that the right hand premiss has been obtained through application of ($\bullet$–**I**). The case where α is a side formula is once again easy. So let α be main formula. We get the following local tree.

$$\dfrac{\dfrac{\Theta_1 \vdash \theta_1 \qquad \Theta_2 \vdash \theta_2}{\Theta_1 \circ \Theta_2 \vdash \theta_1 \bullet \theta_2} \qquad \dfrac{\Delta[\theta_1 \circ \theta_2] \vdash \gamma}{\Delta[\theta_1 \bullet \theta_2] \vdash \gamma}}{\Delta[\Theta_1 \circ \Theta_2] \vdash \gamma} \qquad \leadsto$$

$$\dfrac{\Theta_2 \vdash \theta_2 \qquad \dfrac{\Theta_1 \vdash \theta_1 \qquad \Delta[\theta_1 \circ \theta_2] \vdash \gamma}{\Delta[\Theta_1 \circ \theta_2] \vdash \gamma}}{\Delta[\Theta_1 \circ \Theta_2] \vdash \gamma}$$

In all cases the degree of the cut has been reduced. The admissibility of cut is proved thus. Each of the given local constructions either reduces the degree of the cut or it leaves the degree invariant and instead reduces the depth of that cut. The method therefore is finite in all cases and ends when all cuts have degree 0 and also depth 0. This means that they have been successfully eliminated. (See the discussion in the previous section).

We shall also present a different formulation of **L** using natural deduction over ordered dags. It is perhaps best to first give the rules:

$$(\mathbf{I}\text{–}\bullet)\quad \frac{\alpha \quad \beta}{\alpha \bullet \beta} \qquad\qquad (\mathbf{E}\text{–}\bullet)\quad \frac{\alpha \bullet \beta}{\alpha \quad \beta}$$

$$(\mathbf{I}\text{–}/)\quad \begin{array}{c} [\alpha] \\ \vdots \\ \hline \beta \\ \hline \alpha/\beta \end{array} \qquad\qquad (\mathbf{E}\text{–}/)\quad \frac{\alpha \quad \alpha\backslash\beta}{\beta}$$

$$(\mathbf{I}\text{–}\backslash)\quad \begin{array}{c} [\alpha] \\ \vdots \\ \hline \beta \\ \hline \alpha/\beta \end{array} \qquad\qquad (\mathbf{E}\text{–}\backslash)\quad \frac{\beta/\alpha \quad \alpha}{\beta}$$

These are rules very much like the natural deduction rules for intuitionistic logic. However, two differences must be noted. Disregard for the moment the rules for $\bullet$. (This would incidentally give exactly the natural deduction calculus corresponding to **AB**.) Then we find that the rules create not trees but ordered trees. Otherwise, the difference between then rules for / and the rules for \ is obliterated. Second, the elimination rule for $\bullet$ creates two linearly ordered daughters for a node, thus we not only create ordered trees, we in fact create ordered dags. We shall not spell out exactly how the rules are interpreted in terms of ordered dags, but point out a few noteworthy things. First, this style of presentation is very much linguistically oriented. We may in fact proceed in the same way as for **AB** and define algorithms that decorate strings with certain categorial labels and proceed downward using the rules shown above. Yet, it must be clear that the so created structures cannot be captured by constituency rules (let alone rules of a context free grammar) for the simple reason that we do not have trees. The following derivation is illustrative of this.

$$\begin{array}{ccc}
\multicolumn{3}{c}{(\alpha \bullet (\alpha\backslash\gamma)/\beta) \bullet \beta} \\ \hline
\multicolumn{2}{c}{\alpha \bullet (\alpha\backslash\gamma)/\beta} & \beta \\ \hline
\alpha & (\alpha\backslash\gamma)/\beta & \beta \\ \cline{2-3}
\vdots & \multicolumn{2}{c}{\alpha\backslash\gamma} \\ \hline
\multicolumn{3}{c}{\gamma}
\end{array}$$

Notice that if a rule has two premisses, these must be adjacent and follow each other in the order specified in the rule. No more is required. This allows among other to derive associativity, that is, $(\alpha \bullet \beta) \bullet \gamma \dashv\vdash \alpha \bullet (\beta \bullet \gamma)$. However, notice the role of the so–called assumptions and their discharge. Once an assumption is discharged, it is effectively removed, so that the items to its left and its right are now adjacent. This plays a crucial role in the derivation of the rule of function composition.

$$\begin{array}{ccc}
\alpha/\beta & \beta/\gamma & \gamma^{\surd} \\ \cline{2-3}
\vdots & \multicolumn{2}{c}{\beta} \\ \hline
 & \alpha & \\ \cline{2-2}
 & \alpha/\gamma &
\end{array}$$

As soon as the assumption γ is removed, the top sequence reads $\alpha/\beta, \beta/\gamma$.

The relationship with **L** is as follows. Let Γ be a sequence of categories. We interpret this as a labelled dag, which is linearly ordered. Now we successively apply the rules above. It is verified that each rule application preserves the property that the leaves of the dag are linearly ordered. Define a category corresponding to a sequence as follows.

$$\begin{array}{lcl}
\alpha^{\bullet} & := & \alpha \\
(\alpha, \Delta)^{\bullet} & := & \alpha \bullet \Delta^{\bullet}
\end{array}$$

First of all we say that for two sequences Δ and Δ', Δ' is **derivable** in the natural deduction style calculus if there is a tree constructed according to the rules above, whose topmost sequence is Δ and whose lowermost sequence is Δ'. (Notice that assumptions get discharged, so that we cannot simply say that Δ is the sequence we started off with.) The following is then shown by induction.

Theorem 3.6.3 *Let Δ and Θ be two sequences of categories. Θ is derivable from Δ if and only if $\Delta \vdash \Theta^\bullet$ is derivable in* **L**.

This shows that the natural deduction style calculus is effectively equivalent to **L**.

L allows for a result akin to the Curry–Howard–Isomorphism. This is an extension of the latter result in two respects. First, we have an additional type constructor, $\bullet$, which we have to match by some category constructor, and second, there are different structural rules. First, the new type constructor is actually the pair–formation.

Definition 3.6.4 *Every λ–term is a $\lambda^\bullet$–term. Given two $\lambda^\bullet$–terms, $\langle M, N\rangle$, $p_1(M)$ and $p_2(M)$ also are λ–terms. Further, the following equations hold.*

$$p_1(\langle M, N\rangle) = M, \qquad p_2(\langle M, N\rangle) = N$$

$p_1(U)$ and $p_2(U)$ is not defined if U is not of the form $\langle M, N\rangle$ for some M and N. The functions p_1 and p_2 are called the ***projections***.

Notice that antecedents of sequents consist no longer of sets of sequences. Hence, Γ, Δ, Θ now denote sequences rather than sets.

$$\text{(cut)}\quad \frac{\Gamma \vdash M : \varphi \quad \Delta, x : \varphi, \Theta \vdash N : \chi}{\Delta, \Gamma, \Theta \vdash [M/x]N : B} \qquad \text{(ax)}\quad x : \varphi \vdash x : \varphi$$

$$(\mathbf{E}\text{–}/)\quad \frac{\Gamma \vdash M : \alpha/\beta \quad \Delta \vdash N : \beta}{\Gamma, \Delta \vdash MN : \alpha} \qquad (\mathbf{I}\text{–}/)\quad \frac{\Gamma, x : \beta \vdash M : \alpha}{\Gamma \vdash \lambda x.M : \alpha/\beta}$$

$$(\mathbf{E}\text{–}\backslash)\quad \frac{\Gamma \vdash M : \beta\backslash\alpha \quad \Delta \vdash N : \beta}{\Delta, \Gamma \vdash MN : \alpha} \qquad (\mathbf{I}\text{–}\backslash)\quad \frac{x : \beta, \Gamma \vdash M : \alpha}{\Gamma \vdash \lambda x.M : \beta\backslash\alpha}$$

$$(\mathbf{E}\text{–}\bullet)\quad \frac{\Gamma \vdash M : \alpha \bullet \chi \quad \Delta, x : \alpha, y : \chi, \Theta \vdash U : \psi}{\Delta, \Gamma, \Theta \vdash [p_1(M)/x][p_2(M)/y]U : \psi}$$

$$(\mathbf{I}\text{–}\bullet)\quad \frac{\Gamma \vdash M : \alpha \quad \Delta \vdash N : \chi}{\Gamma, \Delta \vdash \langle M, N\rangle : \alpha \bullet \chi}$$

We have also put a general constraint on the proofs that variables may not be used twice. To implement this constraint, we define the notion of a linear term:

Definition 3.6.5 *$\mathfrak{L}_K$ is the smallest set of $\lambda^\bullet$–terms, called* ***linear terms****, such that the following holds.*

1. *Every variable is a linear term.*
2. *If M and N are linear terms of type $\alpha \to \beta$ and α, respectively, and if M and N are disjoint in free variables then (MN) is a linear term.*
3. *If M is a linear term and x free in M then $(\lambda x . M)$ is a linear term.*
4. *If N and N are linear terms and disjoint in their free variables then $\langle M, N\rangle$ is a linear term.*
5. *If M and N are linear terms, N of type $\alpha \bullet \beta$, if M and N are disjoint in variables, if x is a variable of type α, y a variable of type β then $[p_1(N)/x][p_2(N)/y]M$ is a linear term.*

In short, a term is linear if in every subterm $(\lambda x . M)$, x occurs exactly once free in M. It is established that the calculus above yields only linear terms if we start with variables and require in the rules (**I**–$\bullet$), (**E**–/), (**E**–\\) we require that the sets of free variables be disjoint, and that in (**I**–/) and (**I**–\\) we require that the variable occurs free in M. This way we can ensure that for every sequent derivable in **L** there actually exists a labelling such that the labelled sequent is derivable in the labelled calculus. This new calculus establishes a close correspondence between linear $\lambda^\bullet$–terms and the so–called multiplicative fragment of linear logic, which naturally arises from the above calculus by stripping off the terms and leaving only the formulae. A variant of proof normalization can be shown, and all this yields that **L** has quite well–behaved properties.

In presence of the rules (ass1) and (ass2) $\circ$ behaves exactly like concatenation, that is, it is a fully associative operation. Therefore we shall change the notation in what is to follow. In place of structures consisting of categories we shall consider finite sequences of categories, that is, strings over $\mathrm{Cat}_{\backslash,\bullet,/}(C)$. We denote concatenation by comma, as is common in proof theory.

Now we return to the theory of meaning. In the previous section we have seen how to extend **AB** by a component for meanings, which computes the meaning in tandem with the category. We shall do the same here. To this end we shall have to first clarify what we mean by a realization of $\alpha \bullet \beta$. We shall agree on the following.

$$\ulcorner \alpha \bullet \beta \urcorner := \ulcorner \alpha \urcorner \times \ulcorner \beta \urcorner$$

The rules are tailored to fit this interpretation. The rules are as follows.

$$\frac{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash \alpha : X}{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash \alpha : X}$$

This means that the restructuring of the term is without influence on its meaning. Likewise we have

$$\frac{\Gamma[(\Delta_1 \circ \Delta_2) \circ \Delta_3] \vdash \alpha : X}{\Gamma[\Delta_1 \circ (\Delta_2 \circ \Delta_3)] \vdash \alpha : X}$$

$$\frac{\Gamma[\alpha : x_\alpha \circ \beta : x_\beta] \vdash \gamma : X}{\Gamma[\alpha \bullet \beta] \vdash \gamma : [p_1(z_{\alpha\bullet\beta})/x_\alpha, p_2(z_{\alpha\bullet\beta})/x_\beta]X}$$

This rule says that in place of a function of two arguments α and β we can form a function of a single argument of type $\alpha \bullet \beta$. The two arguments we can recover by application of the projection functions. The fourth rule finally tells us how the type/category $\alpha \bullet \beta$ is interpreted.

$$\frac{\Gamma \vdash \alpha : X \qquad \Delta \vdash \beta : Y}{\Gamma \circ \Delta \vdash \alpha \bullet \beta : \langle X, Y \rangle}$$

Here we have the same problem as before with **AB**. The meaning assignments that are being computed are not in full accord with the interpretation. The term $(\mathtt{x}_0(\mathtt{x}_1\mathtt{x}_2))$ does not denote the same function as them $((\mathtt{x}_0\mathtt{x}_1)\mathtt{x}_2)$. (Usually, one of them is not even well defined.) So this raises the question whether it is at all legitimate to proceed in this way. We shall avoid the question by introducing a totally different calculus which builds on the calculus **N** of the previous section. The rules (**ass1**) and (**ass2**) are dropped. Furthermore, (**•–I**) is restricted to $\Gamma = \varnothing$. These restrictions are taken over from **N** for the abstraction rules.

$$
\begin{array}{lll}
(\mathrm{ax}) & \vec{x} : \alpha : x_\alpha \vdash \vec{x} : \alpha : x_\alpha & \\
(\mathbf{I}\text{–}/) & \dfrac{\Gamma \circ \vec{x} : \alpha : x_\alpha \vdash \vec{y} : \beta : Y}{\Gamma \vdash \vec{y}/\vec{x} : \beta/\alpha : \lambda x_\alpha.Y} & x_\alpha \text{ an argument variable} \\
(\mathbf{I}\text{–}\backslash) & \dfrac{\vec{x} : \alpha : x_\alpha \circ \Gamma \vdash \vec{y} : \beta : Y}{\Gamma \vdash \vec{x}\backslash\vec{y} : \alpha\backslash\beta : \lambda x_\alpha.Y} & x_\alpha \text{ an argument variable} \\
(\mathbf{E}\text{–}/) & \dfrac{\Gamma \vdash \vec{x} : \alpha : X \qquad \Delta \vdash \vec{y} : \beta/\alpha : Y}{\Delta \circ \Gamma \vdash \vec{y} \cdot \vec{x} : \beta : XY} & \\
(\mathbf{E}\text{–}\backslash) & \dfrac{\Gamma \vdash \vec{x} : \alpha : X \qquad \Delta \vdash \vec{y} : \alpha\backslash\beta : Y}{\Gamma \circ \Delta \vdash \vec{x} \cdot \vec{y} : \beta : XY} & \\
(\bullet\text{–}\mathbf{I}) & \dfrac{\vec{x} : \alpha : x_\alpha \circ \vec{y} : \beta : y_\beta \vdash \vec{z} : \gamma : X}{\vec{x} \cdot \vec{y} : \alpha \bullet \beta : z_{\alpha\bullet\beta} \vdash \vec{z} : \gamma : [p_1(z_{\alpha\bullet\beta})/x_\alpha, p_2(z_{\alpha\bullet\beta})/y_\beta]X} & \\
(\mathbf{I}\text{–}\bullet) & \dfrac{\Gamma \vdash \vec{x} : \alpha : X \qquad \Delta \vdash \vec{y} : \beta : Y}{\Gamma \circ \Delta \vdash \vec{x} \cdot \vec{y} : \alpha \bullet \beta : \langle X, Y \rangle} &
\end{array}
$$

This calculus is not as elegant as **L**. However, it is semantically correct. If one desperately wants to have associativity, one has to introduce combinators at the right hand side. So, a use of the associativity rule is accompanied in the semantics by a use of **C** with $\mathbf{C}XYZ = X(YZ)$. We shall not spell out the details here.

Exercise 114. In place of sequents of the form $\alpha \vdash \alpha$ for arbitrary α only sequents $c \vdash c$, $c \in C$, are axioms. Show that with the rules of **L** $\alpha \vdash \alpha$ can be derived for every α.

Exercise 115. Prove Theorem 3.7.1.

Exercise 116. Let $G = \langle C, S, A, \zeta, \mathbf{L}^- \rangle$ be an $\mathbf{L}^-$–grammar. Show that the language $\{\vec{x} : \vdash_G \vec{x}\}$ is context free.

Exercise 117. Show that the sequent $\alpha/\beta \circ \beta/\gamma \vdash \alpha/\gamma$ is derivable in **L**, however not in **AB**. What semantics does the structure $\alpha/\beta \circ \beta/\gamma$ have?

Exercise 118. A **loop** is a structure $\langle L, \cdot, \backslash, / \rangle$ where $\Omega(\cdot) = \Omega(\backslash) = \Omega(/) = 2$ and the following equations hold for all $x, y \in L$.

$$x \cdot (x\backslash y) = y, \quad (y/x) \cdot x = y$$

The categories do not form a loop with respect to $\backslash$, $/$ and $\cdot$ (!), for the reason that $\cdot$ is only partially defined. Here is a possible remedy. Define $\approx \subseteq \mathrm{Cat}_{\backslash,\bullet,/}(C)^2$ to be the least congruence such that

$$(\alpha \bullet \beta)/\beta \approx \alpha, \qquad \beta\backslash(\beta \bullet \alpha) \approx \alpha$$

Show that the free algebra of categories over C factored by $\approx$ is a loop. What is $\alpha \cdot \beta$ in the factored algebra?

Exercise 119. Show that the following rules are admissible in **L**.

$$(\bullet\text{–}\mathbf{E})\ \frac{\Gamma[\theta_1 \bullet \theta_2] \vdash \alpha}{\Gamma[\theta_1 \circ \theta_2] \vdash \alpha} \quad (\mathbf{E}\text{–}/)\ \frac{\Gamma \vdash \alpha/\beta}{\Gamma \circ \beta \vdash \alpha} \quad (\mathbf{E}\text{–}\backslash)\ \frac{\Gamma \vdash \beta\backslash\alpha}{\beta \circ \Gamma \vdash \alpha}$$

3.7 Pentus' Theorem

It was conjectured by Noam Chomsky that the languages generated by **L** are context free, which means that **L** is in effect not stronger than **AB**. This was first shown by Mati Pentus (see (Pentus, 1997)). It starts with an observation by Dirk Roorda. Let $\mathfrak{G} := \langle G, \cdot, ^{-1}, 1 \rangle$ be a group and $\gamma : C \to G$. We extend γ to all types and structures as follows.

$$\begin{array}{lcl}
\gamma(\alpha \bullet \beta) & := & \gamma(\alpha) \cdot \gamma(\beta), \\
\gamma(\alpha/\beta) & := & \gamma(\alpha) \cdot \gamma(\beta)^{-1}, \\
\gamma(\beta\backslash\alpha) & := & \gamma(\beta)^{-1} \cdot \gamma(\alpha), \\
\gamma(\Gamma \circ \Delta) & := & \gamma(\Gamma) \cdot \gamma(\Delta).
\end{array}$$

We call γ a **group valued interpretation**.

Theorem 3.7.1 (Roorda) *If $\Gamma \vdash \alpha$ then for all group valued interpretations γ $\gamma(\Gamma) = \gamma(\alpha)$.*

The proof is performed by induction over the length of the derivation and is left as an exercise. Let C be given and $b \in C$. For a category α over C we define

$$\begin{array}{lcl} \sigma_b(c) & := & \left\{ \begin{array}{ll} 1 & \text{if } b = c, \\ 0 & \text{otherwise.} \end{array} \right. \\ \sigma_b(\alpha \bullet \beta) & := & \sigma_b(\alpha) + \sigma_b(\beta), \\ \sigma_b(\alpha/\beta) & := & \sigma_b(\alpha) + \sigma_b(\beta), \\ \sigma_b(\beta\backslash\alpha) & := & \sigma_b(\alpha) + \sigma_b(\beta). \end{array}$$

Likewise we define

$$\begin{array}{lcl} |\alpha| & := & \sum_{b \in B} \sigma_b(\alpha) \\ \pi(\alpha) & := & \{b \in B : \sigma_b(\alpha) > 0\} \end{array}$$

These definitions are extended in the canonical way to structures. Let Δ be a nonempty structures (so $\Delta \neq \varepsilon$) and $\Gamma[-]$ a structure containing a marked occurrence of a substructure. An **interpolant** for a sequent $\Gamma[\Delta] \vdash \alpha$ in a calculus **K** with respect to Δ is a category θ such that

1. $\sigma_b(\theta) \leq min\{\sigma_b(\Gamma) + \sigma_b(\alpha), \sigma_b(\Delta)\}$, for all $b \in C$,
2. $\Delta \vdash \theta$ is derivable in **K**,
3. $\Gamma[\theta] \vdash \alpha$ is derivable in **K**.

If θ satisfies these conditions then in particular $\pi(\theta) \subseteq \pi(\Gamma \circ \alpha) \cap \pi(\Delta)$. We say **K** has **interpolation**, if for every derivable sequent $\Gamma[\Delta] \vdash \alpha$ there exists an interpolant with respect to Δ.

We are interested in the calculi **AB** and **L**. In the case of **L** we have to remark that in presence of full associativity the interpolation property can be formulated as follows. We deal with sequents of the form $\Gamma \vdash \alpha$ where Γ is a sequence of categories. If $\Gamma = \Theta_1, \Delta, \Theta_2$ with $\Delta \neq \varepsilon$ then there is an interpolant with respect to Δ. For let Δ° be a structure in $\circ$ which corresponds to Δ (after omitting all occurrences of $\circ$). Then there exists a sequent $\Gamma^\circ \vdash \alpha$ which is derivable and in which Δ° occurs as a substructure.

The interpolation property is typically shown by induction on the derivation. In the case of an axiom there is nothing to show. For there we have a sequent $\alpha \vdash \alpha$ and the marked structure Δ has to be α. In this case α is an interpolant. Now let us assume that the rule (**I**–$/$) has been applied to yield the final sequent. Further, assume that the interpolation property has been shown for the premisses. Then we have the following constellation.

$$\frac{\Gamma[\Delta] \circ \alpha \vdash \beta}{\Gamma[\Delta] \vdash \beta/\alpha}$$

We have to find an interpolant with respect to Δ. Because of the induction hypothesis we find a formula so that $\Gamma[\theta] \circ \alpha \vdash \beta$ and $\Delta \vdash \theta$ are both derivable and $\sigma_b(\theta) \leq min\{\sigma_b(\Gamma \circ \alpha \circ \beta), \sigma_b(\Delta)\}$ for all $b \in C$. Then however $\Gamma[\theta] \vdash \beta/\alpha$ as well as $\Delta \vdash \theta$ are derivable and we have $\sigma_b(\theta) \leq min\{\sigma_b(\Gamma \circ \beta/\alpha), \sigma_b(\Delta)\}$. Hence θ also is an interpolant with respect to Δ in $\Gamma[\Delta] \vdash \beta/\alpha$. The case of (**I**–$\backslash$) is fully analogous.

Now we look at the case that the last rule that has been applied is ($/$–**I**).

$$\frac{\Gamma \vdash \beta \qquad \Delta[\alpha] \vdash \gamma}{\Delta[\alpha/\beta \circ \Gamma] \vdash \gamma}$$

Now we choose a substructure Z from $\Delta[\alpha/\beta \circ \Gamma]$. Several cases have to be distinguished. (1) Z is a substructure of Γ, that is, $\Gamma = \Gamma'[Z]$. Then there exists an interpolant θ for $\Gamma'[Z] \vdash \beta$ with respect to Z. It is easily computed that θ also is an interpolant for $\Delta[\alpha/\beta \circ \Gamma'[Z]] \vdash \gamma$ with respect to Z. (2) Z is disjoint with the marked occurrence in $\alpha/\beta \circ \Gamma$. Then $\Delta[\alpha] = \Delta'[Z, \alpha]$ (now we have two marked occurrences of structures) and there is an interpolant θ with respect to Z for $\Delta'[Z, \alpha] \vdash \gamma$. Also in this case one calculates that θ is the desired interpolant. (3) $Z = \alpha/\beta$.

By induction hypothesis there is an interpolant θ_ℓ for $\Gamma \vdash \beta$ with respect to Γ, as well as an interpolant θ_r for $\Delta[\alpha] \vdash \gamma$ with respect to α. Then $\theta := \theta_r/\theta_\ell$ is the interpolant. For we have

$$\begin{array}{rcl}\sigma_b(\theta) & = & \sigma_b(\theta_r) + \sigma_b(\theta_\ell) \\ & \leq & min\{\sigma_b(\Delta \circ \gamma), \sigma_b(\alpha)\} + min\{\sigma_b(\beta), \sigma_b(\Gamma)\} \\ & \leq & min\{\sigma_b(\Delta \circ \Gamma \circ \gamma), \sigma_b(\alpha/\beta)\}\end{array}$$

Furthermore,

$$\frac{\Gamma \vdash \theta_\ell \qquad \Delta[\theta_r] \vdash \gamma}{\Delta[\theta_r/\theta_\ell \circ \Gamma] \vdash \gamma} \qquad\qquad \frac{\dfrac{\theta_\ell \vdash \beta \qquad \alpha \vdash \theta_r}{\alpha/\beta \circ \theta_\ell \vdash \theta_r}}{\alpha/\beta \vdash \theta_r/\theta_\ell}$$

So we have an interpolant for some Θ. (4) $Z = \Theta[\alpha/\beta \circ \Gamma]$. Then $\Delta[\alpha/\beta \circ \Gamma] = \Delta'[\Theta[\alpha/\beta \circ \Gamma]]$ for some Δ'. Then by hypothesis there is an interpolant for $\Delta'[\Theta[\alpha]] \vdash \gamma$ with respect to $\Theta[\alpha]$. We show that θ is the desired interpolant.

$$\frac{\Gamma \vdash \beta \qquad \Theta[\alpha] \vdash \theta}{\Theta[\alpha/\beta \circ \Gamma] \vdash \theta} \qquad\qquad \Delta'[\theta] \vdash \gamma$$

In addition

$$\begin{array}{rcl}\sigma_b(\theta) & \leq & min\{\sigma_b(\Delta' \circ \gamma), \sigma_b(\Theta[\alpha])\} \\ & \leq & min\{\sigma_b(\Delta' \circ \gamma), \sigma_b(\Theta[\alpha/\beta \circ \Gamma])\}\end{array}$$

This ends the proof for the case (/–**I**). The case (\–**I**) again is fully analogous.

Theorem 3.7.2 ***AB*** *has interpolation.* □

Now we move on to **L**. We may use the previous proof to cover the old rules, but now we have to discuss some new cases. Let us first consider the case where we add $\bullet$ together with its introduction rules. Thereafter we go on to **L**. Let now the last rule be ($\bullet$–**I**).

$$\frac{\Gamma[\alpha \circ \beta] \vdash \gamma}{\Gamma[\alpha \bullet \beta] \vdash \gamma}$$

(1) Z does not contain the marked occurrence of $\alpha \bullet \beta$. Then $\Gamma[\alpha \bullet \beta] = \Gamma'[Z, \alpha \bullet \beta]$, and by induction hypothesis we get an interpolant θ for $\Gamma'[Z, \alpha \circ \beta] \vdash \gamma$ with respect to Z. It is easily checked that θ also is an interpolant for $\Gamma'[Z, \alpha \bullet \beta] \vdash \gamma$ with respect to Z. (2) Let $Z = \Theta[\alpha \bullet \beta]$. Then $\Gamma[\alpha \bullet \beta] = \Gamma'[\Theta[\alpha \bullet \beta]]$. By induction hypothesis there is an interpolant θ for $\Gamma'[\Theta[\alpha \circ \beta]] \vdash \gamma$ with respect to $\Theta[\alpha \circ \beta]$, and it also is an interpolant for $\Gamma'[\Theta[\alpha \bullet \beta]] \vdash \gamma$ with respect to $\Theta[\alpha \bullet \beta]$. In both cases we have found an interpolant.

Now we turn to the case (**I**–$\bullet$).

$$\frac{\Gamma \vdash \alpha \qquad \Delta \vdash \beta}{\Gamma \circ \Delta \vdash \alpha \bullet \beta}$$

There are now three cases for Z. (1) $\Gamma = \Gamma'[Z]$. By induction hypothesis there is an interpolant θ_ℓ for $\Gamma'[Z] \vdash \alpha$ with respect to Z. This is the desired interpolant. (2) $\Delta = \Delta'[Z]$. Analogous to (1). (3) $Z = \Gamma \circ \Delta$. By hypothesis there is an interpolant θ_ℓ for $\Gamma \vdash \alpha$ with respect to Γ and an interpolant θ_r for $\Delta \vdash \beta$ with respect to Δ. Put $\theta := \theta_\ell \bullet \theta_r$. This is the desired interpolant. For

$$\frac{\Gamma \vdash \theta_\ell \qquad \Delta \vdash \theta_r}{\Gamma \circ \Delta \vdash \theta_\ell \bullet \theta_r} \qquad\qquad \cfrac{\cfrac{\theta_\ell \vdash \alpha \qquad \theta_r \vdash \beta}{\theta_\ell \circ \theta_r \vdash \alpha \bullet \beta}}{\theta_\ell \bullet \theta_r \vdash \alpha \bullet \beta}$$

In addition it is calculated that $\sigma_b(\theta) \leq min\{\sigma_b(\alpha \bullet \beta), \sigma_b(\Gamma \circ \Delta)\}$.

Finally we must study **L**. The rules (**ass1**), (**ass2**) pose a technical problem since we cannot proceed by induction on the derivation, since the applications of these rules change the structure. Hence we chance to another system of sequents and turn — as discussed above — to sequents of the form $\Gamma \vdash \alpha$ where Γ is a sequence of categories. In this case the rules (**ass1**) and (**ass2**) must be eliminated. However, in the proof we must make more distinctions in cases. The rules (**I**–/) and (**I**–\) are still unproblematic. So we look at a more complicated case, namely an application of the rule (/–**I**).

$$\frac{\Gamma \vdash \beta \qquad \Delta[\alpha] \vdash \gamma}{\Delta[\alpha/\beta, \Gamma] \vdash \gamma}$$

We can segment the structure $\Delta[\alpha/\beta, \Gamma]$ into $\Delta', \alpha/\beta, \Gamma, \Delta''$. Let a subsequence Z be distinguished in $\Delta', \alpha/\beta, \Gamma, \Delta''$. The case where Z is fully contained in Δ' is relatively easy likewise that case where Z is fully contained in Δ''. The following cases remain. (1) $Z = \Delta_1, \alpha/\beta, \Gamma_1$, where $\Delta' = \Delta_0, \Delta_1$ for some Δ_0, and $\Gamma = \Gamma_1, \Gamma_2$ for some Γ_2. Even if Z is not empty Δ_1 as well as Γ_1 may be empty. Assume $\Gamma_1 \neq \varepsilon$. In this case θ_ℓ an interpolant for $\Gamma \vdash \beta$ with respect to Γ_2 and θ_r an interpolant of $\Delta[\alpha] \vdash \gamma$ with respect to Δ_1, α. (Here it becomes clear why we need not assume $\Delta_1 \neq \varepsilon$.) The following sequents are therefore derivable.

$$\begin{array}{ll} \Gamma_2 \vdash \theta_\ell & \quad \Gamma_1, \theta_\ell \vdash \beta \\ \Delta_1, \alpha \vdash \theta_r & \quad \Delta_0, \theta_r, \Delta'' \vdash \gamma \end{array}$$

Now put $\theta := \theta_r/\theta_\ell$. Then we have

$$\frac{\dfrac{\Delta_1, \alpha \vdash \theta_r \qquad \Gamma_1, \theta_\ell \vdash \beta}{\Delta_1, \alpha/\beta, \Gamma_1, \theta_\ell \vdash \theta_r}}{\Delta_1, \alpha/\beta, \Gamma_1 \vdash \theta_r/\theta_\ell} \qquad\qquad \frac{\Gamma_2 \vdash \theta_\ell \qquad \Delta_0, \theta_r, \Delta'' \vdash \gamma}{\Delta_0, \theta_r/\theta_\ell, \Gamma_2, \Delta'' \vdash \gamma}$$

The conditions on the numbers of occurrences of symbols are easy to check. (2) As in Case (1), but with Γ_1 empty. Let then θ_ℓ be an interpolant for $\Gamma \vdash \beta$ with respect to Γ and θ_r an interpolant for $\Delta_0, \Delta_1, \alpha, \Delta'' \vdash \gamma$ with respect to Δ_1, α. Then put $\theta := \theta_r/\theta_\ell$. θ is an interpolant for the end sequent with respect to Z.

$$\frac{\dfrac{\theta_\ell \vdash \beta \qquad \Delta_1, \alpha \vdash \theta_r}{\Delta_1, \alpha/\beta, \theta_\ell \vdash \theta_r}}{\Delta_1, \alpha/\beta \vdash \theta_r/\theta_\ell} \qquad\qquad \frac{\Gamma \vdash \theta_\ell \qquad \Delta_0, \theta_r, \Delta'' \vdash \gamma}{\Delta_0, \theta_r/\theta_\ell, \Gamma, \Delta'' \vdash \gamma}$$

(3) Z does not contain the marked occurrence of α/β. In this case $Z = \Gamma_2, \Delta_1$ for some final part Γ_2 of Γ and an initial part Δ_1 of Δ''. Γ_2 as well as Δ_1 may be assumed to be nonempty, since otherwise we have a case that has already been discussed. The situation is therefore as follows, for $Z = \Gamma_2, \Delta_1$.

$$\frac{\Gamma_1, \Gamma_2 \vdash \beta \qquad \Delta', \alpha, \Delta_1, \Delta_2 \vdash \gamma}{\Delta', \alpha/\beta, \Gamma_1, \Gamma_2, \Delta_1, \Delta_2 \vdash \gamma}$$

Let θ_ℓ be an interpolant for $\Gamma_1, \Gamma_2 \vdash \beta$ with respect to Γ_2 and θ_r an interpolant for $\Delta', \alpha, \Delta_1, \Delta_2 \vdash \gamma$ with respect to Δ_1. Then the following are derivable

$$\begin{array}{ll} \Gamma_2 \vdash \theta_\ell & \Gamma_1, \theta_\ell \vdash \beta \\ \Delta_1 \vdash \theta_r & \Delta', \alpha, \theta_r, \Delta_2 \vdash \gamma \end{array}$$

Now we choose $\theta := \theta_\ell \bullet \theta_r$. Then we have

$$\frac{\Gamma_2 \vdash \theta_\ell \qquad \Delta_1 \vdash \theta_r}{\Gamma_2, \Delta_1 \vdash \theta_\ell \bullet \theta_r} \qquad \frac{\dfrac{\Gamma_1, \theta_\ell \vdash \beta \qquad \Delta', \alpha, \theta_r, \Delta_2 \vdash \gamma}{\Delta', \alpha/\beta, \Gamma_1, \theta_\ell, \theta_r, \Delta_2 \vdash \gamma}}{\Delta', \alpha/\beta, \Gamma_1, \theta_\ell \bullet \theta_r, \Delta_2 \vdash \gamma}$$

In this case as well the conditions on numbers of occurrences are easily checked. This exhausts all cases. Notice that we have used $\bullet$ to construct the interpolant. In the case of the rules (**I**–$\bullet$) and ($\bullet$–**I**) there are no surprises with respect to **AB**.

Theorem 3.7.3 (Roorda) ***L*** *has interpolation.* □

Now we shall move on to show that **L** is context free. To this end we introduce a series of weak calculi of which we shall show that together they are not weaker than **L**. These calculi are called $\mathbf{L}_m$, $m < \omega$. The axioms of $\mathbf{L}_m$ are sequents $\Gamma \vdash \alpha$ such that the following holds.

1. $\Gamma = \beta_1, \beta_2$ or $\Gamma = \beta_1$ for certain categories β_1, β_2.
2. $\Gamma \vdash \alpha$ is derivable in **L**.
3. $|\alpha|, |\beta_1|, |\beta_2| < m$.

(cut) is the only rule of inference. The main work is in the proof of the following theorem.

Theorem 3.7.4 (Pentus) *Let* $\Gamma = \beta_0, \beta_1, \ldots, \beta_{n-1}$. $\Gamma \vdash \alpha$ *is derivable in* $\boldsymbol{L}_m$ *if and only if*

1. $|\beta_i| < m$ *for all* $i < m$,

2. $|\alpha| < m$ *and*

3. $\Gamma \vdash \alpha$ *is derivable in* $\boldsymbol{L}$.

We shall show first how to get from this fact that **L**–grammars are context free. We weaken the calculi still further. The calculus $\mathbf{L}_m^{\boxminus}$ has the axioms of $\mathbf{L}_m$ but (cut) may be applied only if the left hand premiss is an axiom.

Lemma 3.7.5 *For all sequents* $\Gamma \vdash \alpha$ *the following holds:* $\Gamma \vdash \alpha$ *is derivable in* $\boldsymbol{L}_m^{\boxminus}$ *if and only if* $\Gamma \vdash \alpha$ *is derivable in* $\boldsymbol{L}_m$.

The proof is relatively easy and left as an exercise.

Theorem 3.7.6 *The languages accepted by L–Grammars are context free.*

Proof. Let $\mathbb{L} = \langle B, A, \zeta, S\rangle$ be given. Let m be larger than the maximum of all $|\alpha|$, $\alpha \in \zeta(a)$, $a \in A$. Since A as well as $\zeta(a)$ are finite, m exists. For simplicity we shall assume that $C = \bigcup\langle \pi(\alpha) : \alpha \in \zeta(a), a \in A\rangle$. Now we put $N := \{\alpha : |\alpha| < m\}$. $G := \langle N, A, S, R\rangle$, where

$$\begin{array}{rll} R := & & \{\alpha \to a : a \in \zeta(a)\} \\ & \cup & \{\alpha \to \beta : \alpha, \beta \in N, \beta \vdash \alpha \text{ is L–derivable}\} \\ & \cup & \{\alpha \to \beta_0\beta_1 : \alpha, \beta_0, \beta_1 \in N, \beta_0, \beta_1 \vdash \alpha \text{ is L–derivable}\} \end{array}$$

Now let $\mathbb{L} \vdash \vec{x}$, $\vec{x} = x_0 \cdot x_1 \cdot \ldots x_{n-1}$. Then there exist $\alpha_i \in \zeta(x_i)$ for all $i < n$ such that $\Gamma \vdash S$ is derivable in $\mathbf{L}$, where $\Gamma := \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$. By Theorem 3.7.4 and Lemma 3.7.5 $\Gamma \vdash S$ is derivable also in $\mathbf{L}_m^{\boxminus}$. Induction over the length of the derivation yields that $\vdash_G \alpha_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_{n-1}$ and hence also $\vdash_G \vec{x}$. Now let conversely $\vdash_G \vec{x}$. We extend the category assignment ζ to $\zeta^+ : A \cup N \to Cat_{\backslash,\bullet,/}(C)$ by putting $\zeta^+(\alpha) := \{\alpha\}$ while $\zeta^+ \upharpoonright A = \zeta$. Inductively over the length of the derivation of $\vec{\alpha}$ one shows that from $\vdash_G \vec{\alpha}$ we get $\mathbb{L} \vdash \vec{\alpha}$. □

Now on to the proof of Theorem 3.7.4.

Definition 3.7.7 *A category α is called **thin** if $\sigma_b(\alpha) \leq 1$ for all $b \in C$. A sequent $\Gamma \vdash \alpha$ is called **thin** if the following holds.*

1. *$\Gamma \vdash \alpha$ is derivable in $\boldsymbol{L}$.*

2. *All categories occurring in Γ as well as α are thin.*

3. *$\sigma_b(\Gamma, \alpha) \leq 2$ for all $b \in C$.*

For a thin category α we always have $|\alpha| = |\pi(\alpha)|$. We remark that for a thin sequent only $\sigma_b(\Gamma, \alpha) = 0$ or $= 2$ can occur since $\sigma_b(\Gamma, \alpha)$ always is an even number in a derivable sequent. Let us look at a thin sequent $\Gamma[\Delta] \vdash \alpha$ and an interpolant θ of it with respect to Δ. Then $\sigma_b(\theta) \leq \sigma_b(\Delta) \leq 1$. For either $b \notin \pi(\Delta)$, and then $b \notin \pi(\theta)$, whence $\sigma_b(\theta) = 0$. Or $b \in \pi(\Delta)$; but then $b \in \pi(\Gamma, \alpha)$, and so by assumption $\sigma_b(\Delta) = 1$.

$$\sigma_b(\Delta, \theta) \leq \sigma_b(\Delta) + \sigma_b(\theta) \leq \sigma_b(\Gamma[\Delta], \alpha) + \sigma_b(\theta) \leq 2 + 1.$$

Now $\sigma_b(\Delta) + \sigma_b(\theta)$ is an even number hence either 0 or 2. Hence $\Delta \vdash \theta$ also is thin. Likewise it is shown that $\Gamma[\theta] \vdash \alpha$ is thin.

Lemma 3.7.8 *Let $\Gamma, \Theta, \Delta \vdash \alpha$ be a sequent and $b, c \in B$ two distinct elementary categories. Further, let $b \in \pi(\Gamma) \cap \pi(\Delta)$ as well as $c \in \pi(\Theta) \cap \pi(\alpha)$. Then $\Gamma, \Theta, \Delta \vdash \alpha$ is not thin.*

Proof. Let $\mathfrak{F}_G(B)$ be the free group generated by the elementary categories. The elements of this group are finite products of the form $b_0^{s_0} \cdot b_2^{s_2} \cdot \ldots \cdot b_{n-1}^{s_{n-1}}$, where $b_i \neq b_{i+1}$ for $i < n-1$ and $s_i \in \mathbb{Z}$. (If $n = 0$ then the empty product denotes the group unit, 1.) If namely $b_0 = b_1$ the term $b_0^{s_0} \cdot b_1^{s_1}$ can be shortened to $b_0^{s_0+s_1}$. Look at the following group valued interpretation $\gamma : b \mapsto b$. If the sequent was thin we would have $\gamma(\Gamma) \cdot \gamma(\Theta) \cdot \gamma(\Delta) = \gamma(\alpha)$. By hypothesis the left hand side is of the form $w \cdot b^{\pm 1} \cdot x \cdot c^{\pm 1} \cdot y \cdot b^{\pm 1} \cdot z$ for certain products w, x, y, z. The right hand side equals $t \cdot c^{\pm 1} \cdot u$ for certain t, u. Furthermore, we know that terms which stand for w, x, y, z as well as t and u cannot contain b or c if maximally reduced. But then equality cannot hold. ☐

Lemma 3.7.9 *Let $n > 0$ and assume furthermore that the sequent $\alpha_0, \alpha_1, \ldots, \alpha_n \vdash \alpha_{n+1}$ is thin. Then there exists a k with $0 < k < n+1$ such that $\pi(\alpha_k) \subseteq \pi(\alpha_{k-1}) \cup \pi(\alpha_{k+1})$.*

Proof. The proof is by induction on n. We start with $n = 1$. Here the sequent has the form $\alpha_0, \alpha_1 \vdash \alpha_2$. Let $b \in \pi(\alpha_1)$. Then $\sigma_b(\alpha_1) = 1$ since the sequent is thin. And since $\sigma_b(\alpha_0, \alpha_1, \alpha_2) = 2$, we have $\sigma_b(\alpha_0, \alpha_2) = 1$, whence $b \in \pi(\alpha_0) \cup \pi(\alpha_2)$. This finishes the case $n = 1$. Now let $n > 1$ and the claim proved for all $m < n$. We consider two cases. **Case a.** $\pi(\alpha_0, \alpha_1, \ldots, \alpha_{n-2}) \cap \pi(\alpha_n) = \varnothing$. Then we choose $k := n$. For if $b \in \pi(\alpha_n)$ then $\sigma_b(\alpha_0, \ldots, \alpha_{n-2}) = 0$, and so $\sigma_b(\alpha_{n-1}) + \sigma_b(\alpha_{n+1}) = 1$. Hence $b \in \pi(\alpha_{n-1})$ or $b \in \pi(\alpha_{n+1})$. **Case b.** $\pi(\alpha_0, \alpha_1, \ldots, \alpha_{n-2}) \cap \pi(\alpha_n) \neq \varnothing$. Then there exists an elementary category b with $b \in \pi(\alpha_0, \ldots, \alpha_{n-2})$ and $b \in \pi(\alpha_n)$. Put $\Gamma := \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, $\Delta := \alpha_n$. Let θ be an interpolant for $\Gamma, \Delta \vdash \alpha_{n+1}$ with respect to Γ. Then $\Gamma \vdash \theta$ as well as $\theta, \alpha_n \vdash \alpha_{n+1}$ are thin. By induction hypothesis there exists a k such that $\pi(\alpha_k) \subseteq \pi(\alpha_{k-1}) \cup \pi(\alpha_{k+1})$, if $k < n-1$, or $\pi(\alpha_k) \subseteq \pi(\alpha_{k-1}) \cup \pi(\theta)$ in case $k = n-1$. If $k < n-1$ then k is the desired number for the main sequent. Let now $k = n-1$. Then $\pi(\alpha_{n-1}) \subseteq \pi(\alpha_{n-2}) \cup \pi(\theta) \subseteq \pi(\alpha_{n-2}) \cup \pi(\alpha_n) \cup \pi(\alpha_{n+1})$. We show that k in this case too is the desired number for the main sequent. Let $\pi(\alpha_{n-1}) \cap \pi(\alpha_{n+1}) \neq \varnothing$, say $c \in \pi(\alpha_{n-1}) \cap \pi(\alpha_{n+1})$. Then surely $c \notin \pi(\alpha_n)$, hence $c \neq b$. So the sequent is not thin, by Lemma 3.7.8. Hence we have $\pi(\alpha_{n-1}) \cap \pi(\alpha_{n+1}) = \varnothing$, and so $\pi(\alpha_{n-1}) \subseteq \pi(\alpha_{n-2}) \cup \pi(\alpha_n)$. □

Lemma 3.7.10 *Let $\Gamma \vdash \gamma$ be a thin sequent in which all categories have length $< m$. Then $\Gamma \vdash \gamma$ is already derivable in $\boldsymbol{L}_m$.*

Proof. Let $\Gamma = \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$; put $\alpha_n := \gamma$. If $n \leq 2$ then $\Gamma \vdash \gamma$ already is an axiom of $\mathbf{L}_m$. So, let $n > 2$. By the previous lemma there is a k with $\pi(\alpha_k) \subseteq \pi(\alpha_{k-1}) \cup \pi(\alpha_{k+1})$. **Case 1.** $k < n$. **Case 1a.** $|\pi(\alpha_{k-1}) \cap \pi(\alpha_k)| \geq |\pi(\alpha_{k+1}) \cap \pi(\alpha_k)|$. Put $\Xi := \alpha_0, \alpha_1, \ldots, \alpha_{k-2}$, $\Delta := \alpha_{k-1}, \alpha_k$, $\Theta := \alpha_{k+1}, \ldots, \alpha_{n-1}$. Let θ be an interpolant for $\Xi, \Delta, \Theta \vdash \alpha_n$ with respect to Δ. Then the sequent

$$\alpha_0, \ldots, \alpha_{k-2}, \theta, \alpha_{k+1}, \ldots, \alpha_{n-1} \vdash \alpha_n$$

is thin. Furthermore

$$\begin{array}{rcl} \pi(\theta) & \subseteq & (\pi(\alpha_{k-1}) \cup \pi(\alpha_k)) \cap \pi(\Xi, \Theta, \alpha_n) \\ & = & (\pi(\alpha_{k-1}) \cap \pi(\Xi, \Theta, \alpha_n)) \cup (\pi(\alpha_k) \cap \pi(\Xi, \Theta, \alpha_n)). \end{array}$$

Let $b \in \pi(\alpha_{k-1})$. Then $\sigma_b(\alpha_{k-1}) = 1$, $\sigma_b(\Xi, \alpha_{k-1}, \alpha_k, \Theta, \alpha_n) = 2$, whence we have $\sigma_b(\Xi, \alpha_k, \Theta, \alpha_n) = 1$. Hence either $\sigma_b(\alpha_k) = 1$ or $\sigma_b(\Xi, \Theta, \alpha_n) = 1$. Since b was arbitrary we have $\pi(\alpha_k) \cap \pi(\Xi, \Theta, \alpha_n) = \pi(\alpha_{k-1}) - (\pi(\alpha_{k-1}) \cap \pi(\alpha_k))$. By choice of k we have $\pi(\alpha_k) \cap \pi(\Xi, \Theta, \alpha_n) = \pi(\alpha_k) \cap \pi(\alpha_{k+1})$. Hence it holds that

$$\begin{array}{rcl} \pi(\theta) & \subseteq & (\pi(\alpha_{k-1}) \cap \pi(\Xi, \Theta, \alpha_n)) \cup (\pi(\alpha_k) \cap \pi(\Xi, \Theta, \alpha_n)) \\ & \subseteq & (\pi(\alpha_k) - (\pi(\alpha_{k-1}) \cap \pi(\alpha_k)) \cup (\pi(\alpha_k) \cap \pi(\alpha_{k+1})). \end{array}$$

So

$$\begin{array}{rcl} |\pi(\theta)| & = & |\pi(\alpha_{k-1})| + |\pi(\alpha_{k-1}) \cap \pi(\alpha_k)| + |\pi(\alpha_k) \cap \pi(\alpha_{k+1})| \\ & \leq & |\pi(\alpha_{k-1})| \\ & < & m. \end{array}$$

(Note that $|\pi(\alpha_{k-1})| = |\alpha_k|$.) Therefore also $|\theta| < m$ and so $\alpha_{k-1}, \alpha_k \vdash \theta$ is an axiom of $\mathbf{L}_m$. By induction hypothesis $\Xi, \theta, \Theta \vdash \alpha_n$ is derivable in $\mathbf{L}_m$. A single application from both sequents yields the main sequent. It is therefore derivable in $\mathbf{L}_m$. **Case 1b.** $|\pi(\alpha_{k-1}) \cap \pi(\alpha_k)| < |\pi(\alpha_k) \cap \pi(\alpha_{l+1})|$. Here one puts $\Xi = \alpha_0, \ldots, \alpha_{k-1}$, $\Delta := \alpha_k, \alpha_{k+1}$, $\Theta := \alpha_{k+1}, \ldots, \alpha_{n-1}$ and proceeds as in Case 1a. **Case 2.** $k = n - 1$. This means that $\pi(\alpha_{n-1}) \subseteq \pi(\alpha_{n-2}) \cup \pi(\gamma)$. Also here we distinguish to cases. **Case 2a.** $|\pi(\alpha_{n-2}) \cap \pi(\alpha_{n-1})| \geq |\pi(\alpha_{n-1}) \cap \pi(\alpha_n)|$. This case is similar to Case 1a. **Case 2b.** $|\pi(\alpha_{n-2}) \cap \pi(\alpha_{n-1})| < |\pi(\alpha_{n-1}) \cap \pi(\alpha_n)|$. Here put $\Delta := \alpha_0, \ldots, \alpha_{n-2}$, $\Theta := \alpha_{n-1}$. Let θ be an interpolant for $\Delta, \Theta \vdash \alpha_n$ with respect to Δ. Then $\Delta \vdash \theta$ as well as $\theta, \alpha_{n-1} \vdash \alpha_n$ are thin. Further we have

$$\begin{array}{rcl} \pi(\theta) & \subseteq & \pi(\Delta) \cap (\pi(\alpha_{n-1}) \cup \pi(\alpha_n)) \\ & = & (\pi(\Delta) \cap \pi(\alpha_{n-1})) \cup (\pi(\Delta) \cap \pi(\alpha_n)) \\ & = & (\pi(\alpha_{n-2}) \cap \pi(\alpha_{n-1})) \cup (\pi(\alpha_n) - (\pi(\alpha_{n-1}) \cap \pi(\alpha_n))). \end{array}$$

Like in Case 1a we conclude that

$$\begin{aligned}|\pi(\theta)| &= |\pi(\alpha_{n-2}) \cap \pi(\alpha_{n-1})| + |\pi(\alpha_n)| - |\pi(\alpha_{n-1}) \cap \pi(\alpha_n)| \\ &< |\pi(\alpha_n)| \\ &< m.\end{aligned}$$

Hence $\theta, \alpha_{n-1} \vdash \alpha_n$ is an axiom of $\mathbf{L}_m$. By induction hypothesis $\Delta \vdash \theta$ is derivable in $\mathbf{L}_m$. Single application of (cut) yields the main sequent, which is therefore derivable in $\mathbf{L}_m$. ☐

Finally we proceed to the proof of Theorem 3.7.4. Let $|\beta_i| < m$ for all $i < n$, and $|\alpha| < m$. Finally, let $\beta_0, \beta_1, \ldots, \beta_{m-1} \vdash \alpha$ be derivable in $\mathbf{L}$. We choose a derivation of this sequent. We may assume here that the axioms are only sequents of the form $b \vdash b$. For every occurrence of an axiom $b \vdash b$ we choose a new elementary category $\widehat{b}$ and replace this occurrence of $b \vdash b$ by $\widehat{b} \vdash \widehat{b}$. We extend this to the entire derivation and so we get a new derivation of a sequent $\widehat{\beta}_0, \widehat{\beta}_1, \ldots, \widehat{\beta}_{n-1} \vdash \widehat{\alpha}$. We get $\sigma_b(\widehat{\alpha}) + \sum_{i<n} \sigma_b(\widehat{\beta}_i) = 2$, if b occurs at all in the sequent. Nevertheless, the sequent need not be thin, since it may contain categories which are not thin. However, if $\sigma_b(\delta) = 2$ for some δ and some b, then b is not contained in any other category. We exploit this as follows. By successively applying interpolation we get the following sequents, which are all derivable in $\mathbf{L}$.

$$\begin{array}{rl} \widehat{\beta}_0 \vdash \theta_0 & \theta_0, \widehat{\beta}_1, \widehat{\beta}_2, \ldots, \widehat{\beta}_{n-1} \vdash \widehat{\alpha} \\ \widehat{\beta}_1 \vdash \theta_1 & \theta_0, \theta_1, \widehat{\beta}_2, \ldots, \widehat{\beta}_{n-1} \vdash \widehat{\alpha} \\ \vdots & \quad\vdots \\ \widehat{\beta}_{n-1} \vdash \theta_{n-1} & \theta_0, \theta_1, \ldots, \theta_{n-1} \vdash \widehat{\alpha} \\ \theta_0, \theta_1, \ldots, \theta_{n-1} \vdash \gamma & \gamma \vdash \widehat{\alpha} \end{array}$$

It is not hard to show that $\sigma_b(\theta_i) \leq 1$ for all b and all $i < n$. So the sequent $\theta_0, \theta_1, \ldots, \theta_{n-1} \vdash \gamma$ is thin. Certainly $|\gamma| \leq |\widehat{\alpha}| = |\alpha| < m$ as well as $|\theta_i| \leq |\widehat{\alpha}_i| = |\alpha_i| < m$ for all $i < n$. By Lemma 3.7.10 the sequent $\theta_0, \theta_1, \ldots, \theta_{n-1} \vdash \gamma$ is derivable in $\mathbf{L}_m$. The sequents $\widehat{\beta}_i \vdash \theta_i$, $i < n$, as well as $\gamma \vdash \widehat{\alpha}_n$ are axioms of $\mathbf{L}_m$. Hence $\widehat{\beta}_0, \widehat{\beta}_0, \ldots, \widehat{\beta}_{n-1} \vdash \widehat{\alpha}$ is derivable in $\mathbf{L}_m$. We undo the replacement in the derivation. This can in fact be done by applying a homomorphism (substitution) t which replaces $\widehat{b}$ by b. By this we get a derivation $\beta_0, \beta_1, \ldots, \beta_{n-1} \vdash \beta_n$ in $\mathbf{L}_m$. This concludes the proof of Theorem 3.7.4.

We remark that Pentus as shown in (Pentus, 1995) that $\mathbf{L}$ is complete with respect to the cancellation interpretation on semigroups.

Exercise 120. Prove Theorem 3.7.1.

Exercise 121. Let $\Gamma \vdash \alpha$ be derivable in $\mathbf{L}$, $b \in C$. Then $\sigma_b(\Gamma) + \sigma_b(\alpha)$ is an even number.

Exercise 122. Prove Lemma 3.7.5.

3.8 Montague Semantics I

Until the beginning of the 70s semantics of natural languages was considered a hopeless affair. Natural language was thought of (and often still is thought of) as being completely illogical so that no formal theory of semantics for natural languages could ever be given. By contrast, Montague believed that natural languages can be analysed in the same way as formal languages. Even if this was too optimistic (and it is quite certain that Montague did deliberately overstate his case) there is enough evidence that natural languages are quite well behaved. As a proof of his thesis, Montague gave a very small fragment of English for which he produced a formal semantics. In this section we shall give a glimpse of the theory shaped by Montague. Before we can start, we have to talk about predicate logics and its models. For Montague has actually built his semantics somewhat differently than we have done so far. In place of defining the interpretation in a model directly, he defined a translation into λ–calculus over predicate logic, whose interpretation on the other hand is fixed by some general conventions.

A language of first–order predicate logic with identity has the following symbols:

1. a set R of relation symbols, a disjoint set F of function symbols,
2. a countably infinite set $V := \{\mathtt{x}_i : i \in \omega\}$ of variables,
3. the equality symbol $\doteq$,
4. the booleans $\neg$, $\wedge$, $\vee$, $\rightarrow$,
5. the quantifiers $\exists$, $\forall$.

As outlined in Section 1.1, we choose a signature $\langle \Omega, \Xi \rangle$. Then r is a $\Xi(r)$–ary relation symbol and f a $\Omega(f)$–ary function symbol. Equality is a binary relation symbol. We define the set of terms:

* x is a term for $x \in V$.
* If t_i, $i < \Omega(f_j)$, are terms, then $f_j(t_0, \ldots, t_{\Omega(f_j)-1})$ is a term as well.

Next we define formulae (see also Section 2.6; we use 'shy' brackets, since the material constitution of the formulae is of no concern):

* If t_i, $i < \Xi(r_j)$, are terms then $r_j(t_0, \ldots, t_{\Xi(r_j)-1})$ is a formula.
* If t_0 and t_1 are terms then $t_0 \doteq t_1$ is a formula.
* If φ and ψ are formulae, so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$ and $\varphi \rightarrow \psi$.
* If φ is a formula and $x \in V$, then $(\forall x)\varphi$ and $(\exists x)\varphi$ are formulae.

A $\langle \Omega, \Xi \rangle$–**structure** is a triple $\langle M, \{f^{\mathfrak{M}} : f \in F\}, \{r^{\mathfrak{M}} : r \in R\}\rangle$ such that $f^{\mathfrak{M}} : M^{\Omega(f)} \rightarrow M$ for every $f \in F$ and $r^{\mathfrak{M}} \subseteq M^{\Xi(r)}$ for every $r \in R$. Now let $\beta : V \rightarrow M$. Then we define $\langle \mathfrak{M}, \beta \rangle \vDash \varphi$ for a formula by induction. To begin, we associate with every t its value $[t]^\beta$ under β.

$$\begin{array}{rcl} [x]^\beta & := & \beta(x) \\ [f(t_0, \ldots, t_{\Omega(f)-1})]^\beta & := & f^{\mathfrak{M}}([t_0]^\beta, \ldots, [t_{\Omega(f)-1}]^\beta) \end{array}$$

Now we move on to formulae. (In this definition, $\gamma \sim_x \beta$, $x \in V$, if $\beta(y) \neq \gamma(y)$ only if $y = x$.)

$$
\begin{array}{lll}
\langle \mathfrak{M}, \beta \rangle \vDash s_0 \doteq s_1 & :\Leftrightarrow & [s_0]^\beta = [s_1]^\beta \\
\langle \mathfrak{M}, \beta \rangle \vDash r(t_0, \ldots, t_{\Xi(r)-1}) & :\Leftrightarrow & \langle [t_i] : i < \Xi(r) \rangle \in r^{\mathfrak{M}} \\
\langle \mathfrak{M}, \beta \rangle \vDash \neg \varphi & :\Leftrightarrow & \langle \mathfrak{M}, \beta \rangle \nvDash \varphi \\
\langle \mathfrak{M}, \beta \rangle \vDash \varphi \wedge \psi & :\Leftrightarrow & \langle \mathfrak{M}, \beta \rangle \vDash \varphi \text{ and } \langle \mathfrak{M}, \beta \rangle \vDash \psi \\
\langle \mathfrak{M}, \beta \rangle \vDash \varphi \vee \psi & :\Leftrightarrow & \langle \mathfrak{M}, \beta \rangle \vDash \varphi \text{ or } \langle \mathfrak{M}, \beta \rangle \vDash \psi \\
\langle \mathfrak{M}, \beta \rangle \vDash \varphi \rightarrow \psi & :\Leftrightarrow & \text{if } \langle \mathfrak{M}, \beta \rangle \vDash \varphi \text{ then } \langle \mathfrak{M}, \beta \rangle \vDash \psi \\
\langle \mathfrak{M}, \beta \rangle \vDash (\exists x)\varphi & :\Leftrightarrow & \text{there is } \beta' \sim_x \beta : \langle \mathfrak{M}, \beta' \rangle \vDash \varphi \\
\langle \mathfrak{M}, \beta \rangle \vDash (\forall x)\varphi & :\Leftrightarrow & \text{for all } \beta' \sim_x \beta : \langle \mathfrak{M}, \beta' \rangle \vDash \varphi
\end{array}
$$

In this way formulae are interpreted in models.

Definition 3.8.1 *Let Δ be a set of formulae, and φ a formula. Then $\Delta \vDash \varphi$ if for all models $\langle \mathfrak{M}, \beta \rangle$: if $\langle \mathfrak{M}, \beta \rangle \vDash \delta$ for every $\delta \in \Delta$, then also $\langle \mathfrak{M}, \beta \rangle \vDash \varphi$.*

For example, the arithmetical terms in $+$, 0 and $\times$ with the relation $<$ can be interpreted in the structure $\mathbb{N}$ where $+^{\mathbb{N}}$ and $\times^{\mathbb{N}}$ are the usual operations $+$ and $\times$, $0^{\mathbb{N}} = 0$ and $<^{\mathbb{N}} = <$. Then for the valuation β with $\beta(z) = 7$ we have:

$$(*) \qquad \langle \mathbb{N}, \beta \rangle \vDash (\forall x)(\forall y)(x \times y \doteq z \rightarrow (x \doteq 1 \vee y \doteq 1))$$

This formula says nothing but that $\beta(z)$ is a prime number. For a number w is is a prime number if for all numbers u and v: if $u \times^{\mathbb{N}} v = w$ then $u = 1$ or $v = 1$. We compare this with $(*)$. $(*)$ holds if for all $\beta' \sim_x \beta$

$$\langle \mathbb{N}, \beta' \rangle \vDash (\forall y)(x \times y \doteq z \rightarrow (x \doteq 1 \vee y \doteq 1)) \ .$$

This in turn is the case if for all $\beta'' \sim_y \beta'$

$$\langle \mathbb{N}, \beta'' \rangle \vDash x \times y \doteq z \rightarrow (x \doteq 1 \vee y \doteq 1) \ .$$

This means: if $u := \beta''(x)$, $v := \beta''(y)$ and $w := \beta''(z)$ and if we have $w = u \times^{\mathbb{N}} v$, then $u = 1$ or $v = 1$. This holds for all u and v.

Since on the other hand $w = \beta(z)$ we have $(*)$ if and only if $\beta(z)$, that is to say 7, is a prime number.

The reader may convince himself that for every β

$$\langle \mathbb{N}, \beta \rangle \vDash (\forall z)(\exists u)(\forall x)(\forall y)(z > u \wedge (x \times y \doteq u \to (x \doteq 1 \vee y \doteq 1)))$$

This says that for every number there exists a prime number larger than it. For later use we introduce a type e. This is the type of terms. They have values in M. So, e is realized by M.

Before we can start to design a semantics for natural language we shall have to eliminate the relations from the predicate logic. To this end we shall introduce a new basic type, t, which is the type of truth values. It is realized by the set $\{0, 1\}$. An n–place relation r is now replaced by the characteristic function $r^{\spadesuit}$ from n–tuples of objects to truth values, which is defined as follows.

$$r^{\spadesuit}(x_0, x_1, \ldots, x_{\Xi(r)-1}) = 1 :\Leftrightarrow r(x_0, \ldots, x_{\Xi(r)-1}) \ .$$

This allows us to use the λ–calculus for handling the argument places of r. For example, from the binary relation r we can define the following functions r_1 and r_2.

$$\begin{array}{rcl} r_1 & := & \lambda x_e.\lambda y_e.r^{\spadesuit}(x_e, y_e) \\ r_2 & := & \lambda x_e.\lambda y_e.r^{\spadesuit}(y_e, x_e) \end{array}$$

So, we can define functions that either take the first argument of $r^{\spadesuit}$ first, or one which takes the first argument of $r^{\spadesuit}$ second.

Further, we shall also regard $\neg$, $\wedge$, $\vee$ and $\to$ as functions interpreted standardly by $-$, $\cap$, $\cup$ and $\supset$, respectively:

$$\begin{array}{c|c} & - \\ \hline 0 & 1 \\ 1 & 0 \end{array} \qquad \begin{array}{c|cc} \cap & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array} \qquad \begin{array}{c|cc} \cup & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 1 \end{array} \qquad \begin{array}{c|cc} \supset & 0 & 1 \\ \hline 0 & 1 & 1 \\ 1 & 0 & 1 \end{array}$$

Syntactically speaking $\neg$ has category t/t and $\wedge$, $\vee$ and $\to$ have the category $(t\backslash t)/t$. Finally, also the quantifiers will be regarded as functions. To this end we define the function symbols Π and Σ of type $((e \to t) \to t)$. Moreover, we shall require that $\Pi(X)$ is true if and only if for all x $X(x)$ is true, and $\Sigma(X)$ is true if and only if for some x $X(x)$ is true. The interpretation of $(\forall x)\varphi$ is now $\Pi(\lambda x.\varphi)$, and the interpretation of $(\exists x)\varphi$ is now $\Sigma(\lambda x.\varphi)$. So we have $(\forall x) = \lambda X.\Pi(\lambda x.X)$ and $(\exists x) = \lambda X.\Sigma(\lambda x.X)$. We shall however continue to write $\forall x.\varphi$ and $\exists x.\varphi$. This definition can in fact be used to define quantification for all functions. This is the core idea behind the language of simple type theory according to (Church, 1940). Church assumes that the set of basic categories contains at least t. The symbol $\neg$ has the type $t \to t$, while the symbols $\wedge$, $\vee$, $\to$ and $\leftrightarrow$ have the type $t \to (t \to t)$. (Church actually works only with negation and conjunction as basic symbols, but by now it should be clear that this is just a matter of convenience.) To get the power of predicate logic he assumes for each type α a symbol Π^α of type $(\alpha \to t) \to t$ and a symbol ι^α of type $\alpha \to (\alpha \to t)$.

Definition 3.8.2 *A* ***Henkin frame*** *is a structure* $\mathfrak{H} = \langle\{D_\alpha : \alpha \in \mathit{Typ}_\to(B)\}, \bullet, -, \cap, \{\mathsf{p}^\alpha : \alpha \in \mathit{Typ}_\to(B)\}\rangle$ *such that the following holds.*

* $\langle\{D_\alpha : \alpha \in \mathit{Typ}_\to(B)\}, \bullet\rangle$ *is a functionally complete applicative structure.*

* $D_t = \{0,1\}$, $- : D_t \to D_t$ *and* $\cap : D_t \to (D_t \to D_t)$ *are complement and intersection.*

* *For every* $a \in D_{\alpha\to t}$ $\mathsf{p}^\alpha \bullet a = 1$ *if and only if for every* $b \in D_\alpha$: $b \bullet a = 1$.

* *For every* $a \in D_{\alpha\to t}$, *if there is a* $b \in D_\alpha$ *such that* $a \bullet b = 1$ *then* $a \bullet (\mathsf{i}^\alpha \bullet a) = 1$.

A valuation into a Henkin frame is a function β such that for every variable x of type α $\beta(x) \in D_\alpha$. $\langle\mathfrak{H}, \beta\rangle \vDash N$ is defined for every N that has category t by $[N]^\beta = t$. Further, for a set Γ of expressions of type t and N of type N, $\Gamma \vDash N$ if for every Henkin frame and every varluation β: if $\langle\mathfrak{H}, \beta\rangle \vDash M$ for all $M \in \Gamma$ then $\langle\mathfrak{H}, \beta\rangle \vDash N$.

p^α is the interpretation of Π^α and i^α the interpretation of ι^α. So, p^α is nothing but the expression discussed above that allows to define the universal quantifier for functions of type $\alpha \to t$. i^α on the other hand is a kind of choice or 'witness' function. If a is a function from objects of type α into truth values then $\mathsf{i}^\alpha \bullet a$ is an object of type α, and, moreover, if a is at all true on some b of type α, then it is true on $\mathsf{i}^\alpha \bullet a$. In Section 4.4 we shall deliver an axiomatization of the simple type theory and show that it is complete with respect to these models. The reason for explaining about simple type theory is that every semantics or calculus that will be introduced in the sequel can easily be interpreted into simple type theory.

We begin with Montague Semantics. For simplicity's sake we choose a very small fragment. We start with

$$\{\texttt{Paul}, \texttt{Peter}, \texttt{sleeps}, \texttt{sees}\}$$

The semantic type of `Paul` and `Peter` is e, the semantic type of `sleeps` is $e \to t$, the type of `sees` $e \to (e \to t)$. This means: names are interpreted by individuals, intransitive verbs by unary relations, and transitive verbs by binary relations. The (finite) verb `sleeps` is interpreted by the relation sleeps' and `sees` by the relation see'. Because of our convention a transitive verb denotes a function (!) of type $e \to (e \to t)$. So the semantics of these verbs is

$$\begin{array}{lcl} \texttt{sleeps} & \mapsto & \lambda x_e.\mathsf{sleep}'(x_e) \\ \texttt{sees} & \mapsto & \lambda x_e.\lambda y_e.\mathsf{see}'(y_e, x_e) \end{array}$$

We already note here that the variables are unnecessary. After we have seen how the predicate logical formulae can be massaged into typed λ–expressions, we might as well forget this history and write sleep' for the function $\lambda x_e.\mathsf{sleep}'(x_e)$ and see' in place of $\lambda x_e.\lambda y_e.\mathsf{see}'(y_e, x_e)$. This has the additional advantage that we need not mention the variables at all (which is a moot point, as we have seen above). We continue in this section to use the somewhat more longwinded notation, however. We keep this convention introduced by Montague and agree further that the value of `Paul` shall be the constant paul' and the value of `Peter` the constant peter'. Finally, we agree on the syntactic categories.

$$\begin{array}{lcl} \texttt{Paul} & : & e \\ \texttt{Peter} & : & e \\ \texttt{sleeps} & : & e\backslash t \\ \texttt{sees} & : & (e\backslash t)/e \end{array}$$

Here are our 0–ary modes.

$$\begin{array}{lll} \langle \texttt{Paul}, & e, & \mathsf{paul}' \rangle \\ \langle \texttt{Peter}, & e, & \mathsf{peter}' \rangle \\ \langle \texttt{sleeps}, & e\backslash t, & \lambda x_e.\mathsf{sleep}'(x_e) \rangle \\ \langle \texttt{sees}, & (e\backslash t)/e, & \lambda x_e.\lambda y_e.\mathsf{see}'(y_e, x_e) \rangle \end{array}$$

The sentences `Peter sleeps` or `Peter sees Peter` are grammatical, and their meaning is — as is easily verified — $\mathsf{sleep}'(\mathsf{paul}')$ and $\mathsf{see}'(\mathsf{peter}', \mathsf{peter}')$.

The syntactic categories possess an equivalent in syntactic terminology. e for example is the category of proper names. The category $e\backslash t$ is the category of intransitive verbs and the category $(e\backslash t)/e$ is the category of transitive verbs.

This minilanguage can be extended for example by logical words. For example, we can introduce the word `not` by means of the following constant mode.

$$\langle \texttt{not}, (e\backslash t)\backslash(e\backslash t), \lambda e_{e\to t}.\lambda x_e.\neg x_{e\to t}(x_e) \rangle$$

The reader is asked to verify that now `sleeps not` is an intransitive verb, whose meaning is the contrary of the meaning `sleeps`. Hence `Paul sleeps not` is true if and only if `Paul sleeps` is false. This is perhaps not such a good example, since the negation in English is formed using the auxiliary `do`. However, we may introduce `and` by the following mode.

$$\langle \texttt{and}, ((e\backslash t)\backslash(e\backslash t))/(e\backslash t), \lambda x_{e\to t}.\lambda y_{e\to t}.\lambda z_e.x_{e\to t}(z_e) \wedge y_{e\to t}(z_e) \rangle$$

In this way we have a small language which can generate infinitely many grammatical sentences and which assigns them correct meanings. Of course, English (in fact any natural language) is by far more complex than this.

The real advance that Montague made was to show that one can treat quantification. Let us take a look at how this can be done. (Actually, what we are going to explain right now is not Montague's own solution, which will be explained in Chapter 4.) Nouns like `cat` and `mouse` are not proper names but semantically speaking unary predicates. For `cat` does not denote a single individual but a class of individuals. Hence, following our conventions, the semantic type of `cat` and `mouse` is $e \to t$. Syntactically speaking this would correspond to either t/e or $e\backslash t$. Here, no decision is possible, for neither is `Cat Paul` nor is `Paul cat` a grammatical sentence. Montague did not solve this problem; he introduced a new category constructor $//$, which allows to distinguish a category $t//e$ from t/e (the intransitive verb) even though they are not distinct in type. Our approach is simpler. We introduce a category n and stipulate that $\sigma(n) := e \to t$. This is an example where the basic categories are different from the basic semantic types. Now we say that the subject quantifier `every` has the sentactic category $(t/(e\backslash t))/n$. This means the following. It forms a constituent together with a noun which has the category $t/(e\backslash t)$. This in turn is a constituent that needs an intransitive verb to form a sentence. We therefore have the following constant modes.

$$\langle \texttt{every}, (t/(e\backslash t))/n, \lambda x_{e\to t}.\lambda y_{e\to t}.\forall x_e.(x_{e\to t}(x_e) \to y_{e\to t}(x_e))\rangle$$

Let us calculate this with an example.

`every cat sees Peter`

The syntactic analysis is as follows.

$$\dfrac{\dfrac{\underset{t/(e\backslash t)/n}{\texttt{every}} \quad \underset{n}{\texttt{cat}}}{t/(e\backslash t)} \qquad \dfrac{\underset{(e\backslash t)/e}{\texttt{sees}} \quad \underset{e}{\texttt{Peter}}}{e\backslash t}}{t}$$

This induces the following constituent structure.

$$((\texttt{every cat}) \quad (\texttt{sees Peter}))$$

Now we shall have to insert the meanings in place of the words and calculate. This means converting into normal form. For by convention, a constituent has the meaning that arises from applying the meaning of one immediate part to the meaning of the other. That this is now well defined is checked by the syntactic analysis. We calculate in several steps. `sees Peter` is a constituent and its meaning is

$$(\lambda x_e.\lambda y_e.\mathsf{see}'(y_e, x_e))(\mathsf{peter}') = \lambda y_e.\mathsf{see}'(y_e, \mathsf{peter}')$$

Further, `every cat` is a constituent with the following meaning

$$\begin{array}{rl} & (\lambda x_{e\to t}.\lambda y_{e\to t}.(\forall x_e.x_{e\to t}(x_e) \to y_{e\to t}(x_e)))(\lambda x_e.\mathsf{cat}'(x_e)) \\ = & \lambda y_{e\to t}.\forall x_e.((\lambda x_e.\mathsf{cat}'(x_e))(x_e) \to y_{e\to t}(x_e)) \\ = & \lambda y_{e\to t}.\forall x_e.(\mathsf{cat}'(x_e) \to y_{e\to t}(x_e)) \end{array}$$

NOw we combine these two:

$$\begin{array}{rl} & (\lambda y_{e\to t}.\forall x_e.\mathsf{cat}'(x_e) \to y_{e\to t}(x_e))(\lambda y_e.\mathsf{see}'(y_e, \mathsf{peter}')) \\ = & \forall x_e.(\mathsf{cat}'(x_e) \to (\lambda y_e.\mathsf{see}'(y_e, \mathsf{peter}'))(x_e)) \\ = & \forall x_e.(\mathsf{cat}'(x_e) \to \mathsf{see}'(x_e, \mathsf{peter}')) \end{array}$$

This is the desired result. Similarly to `every` we define `some`:

$$\langle \texttt{some}, (t/(e\backslash t))/n, \lambda x_{e\to t}.\lambda y_{e\to t}.\exists x_e.(x_{e\to t}(x_e) \wedge y_{e\to t}(x_e))\rangle$$

If we also want to have quantifiers for direct objects we have to introduce new modes.

$$\begin{array}{l} \langle \texttt{every}, (e\backslash t)/((e\backslash t)/e)/n, \\ \qquad \lambda x_{e\to t}.\lambda y_{e\to(e\to t)}.\lambda y_e.\forall x_e.(x_{e\to t}(x_e) \to y_{e\to(e\to t)}(x_e)(y_e))\rangle \\ \langle \texttt{some}, (e\backslash t)/((e\backslash t)/e)/n, \\ \qquad \lambda x_{e\to t}.\lambda y_{e\to(e\to t)}.\lambda y_e.\exists x_e.(x_{e\to t}(x_e) \to y_{e\to(e\to t)}(x_e)(y_e))\rangle \end{array}$$

For `every cat` as direct object is analyzed as a constituent, which turns a transitive verb into an intransitive verb. Hence it must have the category $(e\backslash t)/((e\backslash t)/e)$. From this follows immediately the category assignment for `every`.

Let us look at this using an example.

`some cat sees every mouse`

The constituent structure is as follows.

((`Some cat`) (`sees` (`every mouse`))))

The meaning of `every mouse` is, as is easily checked, the following:

$$\lambda y_{e\to(e\to t)}.\lambda y_e.\forall x_e(\mathsf{mouse}'(x_e) \to y_{e\to(e\to t)}(x_e)(y_e))$$

In this way we get for `sees every mouse`

$$\begin{array}{rl} & \lambda y_e.\forall x_e(\mathsf{mouse}'(x_e) \to (\lambda x_e.\lambda y_e.\mathsf{see}'(y_e, x_e))(x_e)(y_e)) \\ = & \lambda y_e.\forall x_e(\mathsf{mouse}'(x_e) \to \mathsf{see}'(y_e, x_e)) \end{array}$$

`some cat` is analogous to `every cat`:

$$\lambda y_{e\to t}.\exists x_e.(\mathsf{cat}'(x_e) \wedge y_{e\to t}(x_e))$$

We combine these two.

$$\begin{array}{rl} & \lambda y_{e\to t}.\exists x_e.(\mathsf{cat}'(x_e) \wedge y_{e\to t}(x_e)) \\ & \qquad (\lambda y_e.\forall x_e.(\mathsf{mouse}'(x_e) \to \mathsf{see}'(y_e, x_e))) \\ = & \exists x_e.(\mathsf{cat}'(x_e) \wedge (\lambda y_e.\forall x_e.(\mathsf{mouse}'(x_e) \to \mathsf{see}'(y_e, x_e)))(x_e)) \\ = & \exists x_e.(\mathsf{cat}'(x_e) \wedge \forall z_e.(\mathsf{mouse}'(z_e) \to \mathsf{see}'(x_e, z_e))) \end{array}$$

With this example one can see that the calculations require some caution. Sometimes variables may clash and this calls for the substitution of a variable. This is the case for example when we insert a term and by doing so create a bound occurrences of a variable. The λ–calculus is employed to this dirty work for us.

Montague used the cancellation interpretation for his calculus, hence its sequent formulation uses $\mathbf{AB}^-$. We have seen that this calculus can also be rendered into a sign grammar, which has two modes, forward application ($\mathtt{A}_>$) and backward application

($\mathtt{A}_<$). In syntactic theory, however, the most popular version of grammar is the Lambek–Calculus. However, the latter does not lend itself easily to a compositional interpretation. The fault lies basically in the method of hypothetical assumptions. Let us see why this is so. An adjective like `big` has as its syntactic type n/n, and its semantical type is $(e \to t) \to (e \to t)$. (This is not quite true, but good enough for illustration.) This means that it can modify simply nouns such as `car`, but not relational nouns such as `friend` or `neighbour`. Let us assume that the latter have syntactical type n/g (this means that they need a genitive argument.) Now, in Natural Deduction style Lambek–Calculus we can derive a constituent `big neighbour` by first feeding it a hypothetical argument and then abstracting over it.

$$\dfrac{\dfrac{\begin{array}{c}\texttt{big}\\ n/n : \mathsf{big}' \\ \vdots\end{array} \qquad \dfrac{\begin{array}{c}\texttt{neighbour}\\ n/g : \mathsf{neighbour}'\end{array} \qquad g : y}{n : \mathsf{neighbour}'(y)}}{n : \mathsf{big}'(\mathsf{neighbour}'(y))}}{n/g : \lambda y.\mathsf{big}'(\mathsf{neighbour}'(y))}$$

This allows us, for example, to coordinate `big neighbour` and `friend` and then compose with `of mine`. Since we have made no use of $\bullet$, this proof is also available in $\mathbf{AB}$ (but not in $\mathbf{AB}^-$!). The question arises whether we can find a sign based analogon of this. This is possible. Introduce binary modes $\mathtt{L}_>$ and $\mathtt{L}_<$:

$$\begin{array}{rcl}\mathtt{L}_>(\langle \vec{x}, \alpha, M\rangle, \langle \vec{y}, \gamma, x_\gamma\rangle) & := & \langle \vec{x}/\vec{y}, \alpha/\gamma, (\lambda x_\gamma.Mx_\gamma)\rangle \\ \mathtt{L}_<(\langle \vec{x}, \alpha, M\rangle, \langle \vec{y}, \gamma, x_\gamma\rangle) & := & \langle \vec{y}\backslash\vec{x}, \gamma\backslash\alpha, (\lambda x_\gamma.Mx_\gamma)\rangle\end{array}$$

A condition on the application of these modes is that the variable x_γ actually occurs free in the term. Now introduce a new 0–ary mode with exponent ©, which shall be a symbol not in the alphabet.

$$\mathtt{V}_{\alpha,i} := \langle \copyright, \alpha, x_{\alpha,i}\rangle$$

Consider the structure term

$$\mathtt{L}_>\mathtt{A}_>\mathtt{BgA}_>\mathtt{NbV}_{\alpha,0}\mathtt{V}_{\alpha,0}$$

On condition that it is definite, it has the following unfolding.

$$\langle \texttt{big neighbour}, n/g, \lambda x_{g,0}.\mathsf{big}'\mathsf{neighbour}'(x_{g,0})\rangle$$

These modes play the role of hypothetical arguments in Natural Deduction style derivations. However, the combined effect of these modes is not exactly the same as in the Lambek–Calculus. The reason is that abstraction can only be over a variable that is introduced right or left peripherally to the constituent. However, if we introduce two arguments in succession, we can abstract over them in any order we please, as the reader may check (see the exercises). The reason is that Ⓒ bears no indication of the name of the variable that it introduces. This can be remedied by introducing instead the following 0–ary modes.

$$\mathtt{T}_{\alpha,i} := \langle \copyright_{\alpha,i}, \alpha, x_{\alpha,i}\rangle$$

Notice that these empty elements can be seen as the categorial analogon of traces in Transformational Grammar (see Section 6.5). Now the exponent reveals the exact identity of the variable and the Lambek–Calculus is exactly mirrored by these modes. The price we pay is that there are structure terms whose exponents are not pronounceable: they contain elements that are strictly speaking not overtly visible. The strings are therefore not surface strings. Already in (Harris, 1963) the idea is defended that one must sometimes pass through 'nonexistent' strings, and TG has made much use of this. An idea that suggests itself is to use combinators. This route has been taken by Steedman in (1990; 1996). For example, the addition of the modes $\mathtt{B}_{>}$ and $\mathtt{B}_{<}$ assures us that we can derive the these constituents as well. Steedman and Jacobson emphasize in their work also that variables can be dispensed with in favour of combinators. This helps to eliminate the gap between terms and meanings to some extent.

Exercise 123. Write an **AB**–Categorial Grammar for predicate logic over a given signature and a given structure. *Hint.* You need two types of basic categories: e and t, which now stand for *terms* and *truth–values*. Although the λ–calculus uses more categories, the signs of category e and t should be exactly as the signs corresponding to the terms and formulae in predicate logic, while the others can be freely chosen.

Exercise 124. The solutions we have presented here fall short of taking certain aspects of orthography into account. In particular, sentences do not end in a period and the first word of a sentence is written using lower case letters only. Can you think of a remedy for this situation?

Exercise 125. Show that with the help of $\mathtt{L}_<$ and $\mathtt{L}_>$ and the 0–ary modes $\mathtt{V}_{\alpha,i}$ it is possible to derive the sign

$$\langle \mathtt{give}, (e\backslash t)/e/e, \lambda x.\lambda y.\lambda z.\mathsf{give}'(z)(x)(y)\rangle$$

from the sign

$$\langle \mathtt{give}, (e\backslash t)/e/e, \lambda x.\lambda y.\lambda z.\mathsf{give}'(z)(y)(x)\rangle$$

Exercise 126. We have noted earlier that `and`, `or` and `not` are polymorphic. The polymorphicity can be accommodated directly by defining polyadic operations in the λ–calculus. Here is how. Call a type α t–**final** if it has the following form: (a) $\alpha = t$, or (b) $\alpha = \beta \to \gamma$, where γ is t–final. Define $\wedge_\alpha$, $\vee_\alpha$ and $\neg_\alpha$ by induction. Similarly, define for every function X of type $\alpha \to t$ functions Σ_α and Π_α that interpret the existential and universal quantifier. For the latter you might need some functions in the set–universe.

Exercise 127. A (**unary**) **generalized quantifier** is a function from properties to truth values (so, it is an object of type $(e \to t) \to t$). Examples are `some` and `every`, but there are many more:

(3.8.1) `more than three`
(3.8.2) `an even number of`
(3.8.3) `the director's`

First, give the semantics of each of the generalized quantifiers and define a sign for them. Now try to define the semantics of `more than`. (It takes a number and forms a generalized quantifier.)

Exercise 128. In $CCG(B)$, many (but not all) substrings are constituents. We should therefore be able to coordinate them with `and`. As was noted for example by Eisenberg in (1973), such a coordination is constrained (the brackets enclose the critical constituents).

(3.8.4) *[`John said that I`] `and` [`Mary said that she`] `is the best swimmer.`

(3.8.5) [`John said that I`] `and` [`Mary said that she`] `was the best swimmer.`

The constraint is as follows. $[\vec{x}$ `and` $\vec{y}]$ $\vec{z}$ is well formed only if both $\vec{x}\ \vec{z}$ and $\vec{y}\ \vec{z}$ are. The suggestion is therefore that first the sentence $[\vec{x}\ \vec{z}]$ `and` $[\vec{y}\ \vec{z}]$ is formed and then the first occurrence of $\vec{z}$ is 'deleted'. Can you suggest a different solution? *Note.* The construction is known as **forward deletion**. The more common **backward deletion** gives $\vec{x}\ \vec{z}$ `and` $\vec{y}$, and is far less constrained.

Chapter 4

Semantics

4.1 The Nature of Semantical Representations

This chapter lays the foundation of semantics. In contrast to much of the current semantical theory we shall not use a model–theoretic approach but rather an algebraic one. As it turns out, the algebraic approach helps to circumvent much of the difficulties that beset a model–theoretic analysis, since it does not try to spell out the meanings in every detail, only in as much detail as is needed for the purpose at hand.

In this section we shall be concerned with the question of feasibility of interpretation. Much of semantical theory simply defines mappings from strings to meanings without assessing the question whether such mappings can actually be computed. While on a theoretical level this gives satisfying answers, we still need to consider the question how it is possible that a human being can actually understand a sentence. The question is quite the same for computers. Mathematicians 'solve the equation $x^2 = 2$ by writing $x = \pm\sqrt{2}$. However, this is just a piece of notation. If we want to decide whether or not $\sqrt{2} < 1.5$ we need to do some calculations. In this case the question has a straightforward answer, but but this is not typical (think of trigonometric functions or the solutions of differential equations). However, hope is not lost. There are algorithms by which the number $\sqrt{2}$ can be approximated to any degree of precision needed, using only elementary operations. Much of mathematical theory has been inspired by the need to calculate difficult functions (for example logarithms) by means of elementary ones. Evidently, even though we do not have to bother any more with them thanks to computers, the computer still has to do the job. Computer hardware actually implements sophisticated algorithms for computing nonelementary functions. Furthermore, computers do not compute with arbitrary degree of precision. Numbers are stored in fixed size units (this is not necessary, but the size is limited anyhow by the size of the memory of the computer). Thus, they are only *close* to the actual input, not necessarily equal. Calculations on the numbers propagate these errors and in bad cases it can happen that small errors in the input yield astronomic errors in the output (problems that have this property independent of any algorithm that computes the solution are called **ill conditioned**). Now, what reason do we have to say that a particular machine with a particular algorithm computes, say, $\sqrt{2}$? One answer would be: that the program will yield *exactly* $\sqrt{2}$ given exact input and enough time. Yet, for approximative methods — the ones we generally use —, the computation is never complete. However, then it computes a series of numbers a_n, $n \in \omega$, which converges to $\sqrt{2}$. That is to say, if $\varepsilon > 0$ is any real number (the error) we have to name an n_ε such that for all $n \geq n_\varepsilon$: $|a_n - \sqrt{2}| \leq \varepsilon$, given exact computation. That an algorithm computes such a series is typically shown using pure calculus over the real numbers. This computation is actually independent of the way in which the computation proceeds as long as it can be shown to compute the approximating series. For example, to compute $\sqrt{2}$ using Newton's method, all you have to do is to write a program that calculates

$$a_{n+1} := a_n - (a_n^2 - 2)/2a_n$$

For the actual computation on a machine it matters very much how this series is calculated. This is so because each operation induces an error, and the more we compute the more we depart from the correct value. Knowing the error propagation of the basic operations it is possible to compute exactly, given any algorithm, with what precision it computes. To sum up, in addition to calculus, computation on real machines needs two things:

* A theory of approximation.

* A theory of error propagation.

Likewise, semantics is in need of two things: a theory of approximation, showing us what is possible to compute and what not, and how we can compute meanings, and second a theory of error propagation, showing us how we can determine the meanings in approximation given only limited resources for computation. We shall concern ourselves with the first of these. Moreover, we shall look only at a very limited aspect, namely: what meanings can in principle be computed and which ones not.

We have earlier characterized the computable functions as those that can be computed by a Turing machine. To see that this is by no means an innocent assumption, we shall look at propositional logic. Standardly, the semantics of classical propositional logic is given as follows. (This differs only slightly from the setup of Section 3.2.) The alphabet is $\{\mathtt{(}, \mathtt{)}, \mathtt{p}, \mathtt{0}, \mathtt{1}, \neg, \wedge\}$. Let $V := \mathtt{p}(\mathtt{0} \cup \mathtt{1})^*$ be the set of variables. A function from V to 2 is called a **valuation**. We write $\beta(p)$ for the value of the variable p under β. Now we extend this to a mapping $\overline{\beta}$ from the entire language to 2.

$$\begin{array}{lcll} \overline{\beta}(p) & := & \beta(p) & (p \in V) \\ \overline{\beta}(\mathtt{(}\neg\varphi\mathtt{)}) & := & -\overline{\beta}(\varphi) & \\ \overline{\beta}(\mathtt{(}\varphi \wedge \chi\mathtt{)}) & := & \overline{\beta}(\varphi) \cap \overline{\beta}(\chi) & \end{array}$$

This is well defined if we add that $\cup : 2^2 \to 2$ is the set union and $- : 2 \to 2$ is the set complement.

To obtain from this a compositional interpretation for the language we turn matters around and define the meaning of a proposition to be a function from valuations to 2. Let *Val* be the set of functions from V to 2. Then for every proposition φ, $[\varphi]$ denotes the function from *Val* to 2 that satisfies

$$[\varphi](\beta) = \overline{\beta}(\varphi)$$

(The reader is made aware of the fact that what we have performed here is akin to type raising, turning the argument into a function over the function that applies to it.) $[\varphi]$ can also be defined inductively.

$$\begin{array}{lll} [p] & := & \{\beta : \beta(p) = 1\} \quad (p \in V) \\ [\mathtt{(}\neg\varphi\mathtt{)}] & := & \mathit{Val} - [\varphi] \\ [\mathtt{(}\varphi \wedge \chi\mathtt{)}] & := & [\varphi] \cap [\chi] \end{array}$$

Now notice that V is infinite. However, we have excluded that the set of basic modes is infinite, and so we need to readjust the syntax. Rather than working with only one type of expression, we introduce a new type, that of a **register**. Registers are elements of $G := (\mathtt{0} \cup \mathtt{1})^*$. Then $V = \mathtt{p} \cdot G$. Valuations are now functions from G to 2. The rest is as above. Here is now a sign grammar for propositional logic. The modes are $\mathtt{E}$ (0–ary), $\mathtt{P}_0$, $\mathtt{P}_1$, $\mathtt{V}$, $\mathtt{J}_\neg$ (all unary), and $\mathtt{J}_\wedge$ (binary). The exponents are strings over the alphabets, categories are either R or P, and meanings are either registers (for expressions of category R) or sets of functions from registers to 2 (for expressions of category P).

$$\begin{array}{lll} \mathtt{E} & := & \langle \varepsilon, R, \varepsilon \rangle \\ \mathtt{P}_0(\langle \vec{x}, R, \vec{y} \rangle) & := & \langle \vec{x}\mathtt{0}, R, \vec{y}\mathtt{0} \rangle \\ \mathtt{P}_1(\langle \vec{x}, R, \vec{y} \rangle) & := & \langle \vec{x}\mathtt{1}, R, \vec{y}\mathtt{1} \rangle \\ \mathtt{V}(\langle \vec{x}, R, \vec{x} \rangle) & := & \langle \mathtt{p}\vec{x}, P, [\mathtt{p}\vec{x}] \rangle \\ \mathtt{J}_\neg(\langle \vec{x}, P, M \rangle) & := & \langle \mathtt{(}\neg\vec{x}\mathtt{)}, P, \mathit{Val} - M \rangle \\ \mathtt{J}_\wedge(\langle \vec{x}, P, M \rangle, \langle \vec{y}, P, N \rangle) & := & \langle \mathtt{(}\vec{x} \wedge \vec{y}\mathtt{)}, P, M \cap N \rangle \end{array}$$

It is easily checked that this is well defined. This defines a sign grammar that meets all requirements for being compositional except for one: the functions on meanings are not computable. Notice that (a) valuations are infinite objects, and (b) there are uncountably many valuations. However, this is not sufficient as an argument because we have not actually said how we encode sets of valuations as strings and how we compute with them. Notice also that the notion of computability is defined only on strings. Therefore, meanings too must be coded as strings. We may improve the situation a little bit by assuming that valuations are functions from finite subsets of G to 2. Then at least valuations can be represented as strings (for example, by listing pairs consisting of a register and its value). However, still the set of all valuations that make a given proposition true is infinite. On the other hand, there is an algorithm that can check for any given partial function whether it assigns 1 to a given register (it simply scans the string for the pair whose first member is the given register). Notice that if the function is not defined on the register, we must still give an output. Let it be #. We may then simply take the code of the Turing machine computing that function as the meaning the variable (see Section 1.7 for a definition). Then, inductively, we can define for every proposition φ a machine T_φ that computes the value of φ under any given partial valuation that gives a value for the occurring variables, and assigns # otherwise. Then we assign as the meaning of φ the code $T_\varphi^\spadesuit$ of that Turing machine. This is a perfectly viable approach. However, it suffers from a number of deficiencies that make it less worthwhile.

First, the idea of using partial valuations is not easily generalized. To see this let us now turn to predicate logic (see Section 3.8). As in the case of propositional logic we shall have to introduce binary strings for registers, to form variables. The meaning of a formula φ is by definition a function from pairs $\langle\mathfrak{M},\beta\rangle$ to $\{0,1\}$, where $\mathfrak{M}$ is a structure and β a function from variables to the domain of $\mathfrak{M}$. Again we have the problem to name finitary or at least computable procedures. We shall give two ways of doing so that yield quite different results. The first attempt is to exclude infinite models. Then $\mathfrak{M}$, and in particular the domain M of $\mathfrak{M}$, are finite. A valuation is a partial function from V to M with a finite domain. The meaning of a term under such a valuation is a member of M or $= \star$. (For if x_α is in t, and if β is not defined on x_α then t^β is undefined.) The meaning of a formula is either a truth value or $\star$. The truth values can be inductively defined as in Section 3.8. M has to be finite, since we usually cannot compute the value of $\forall x_\alpha.\varphi(x_\alpha)$ without knowing all values of x_α.

This definition has a severe drawback: it does not give the correct results. For the logic of finite structures is stronger than the logic of all structures. For example, the following set of formulae is not satisfiable in finite structures while it has an infinite model. (Here `0` is a 0–ary function symbol, and `s` a unary function symbol.)

Proposition 4.1.1 *The theory T is consistent but has no finite model.*

$$T := \{\forall \mathtt{x}_0 \neg(\mathtt{sx}_0 \doteq \mathtt{0}), \forall \mathtt{x}_0 \forall \mathtt{x}_1 (\mathtt{sx}_0 \doteq \mathtt{sx}_1 \to \mathtt{x}_0 \doteq \mathtt{x}_1)\}$$

Proof. Let $\mathfrak{M}$ be a finite model for T. Then for some n and some $k > 0$: $s^{n+k}0 = s^n 0$. From this it follows with the second formula that $s^k 0 = 0$. Since $k > 0$, the first formula is false in $\mathfrak{M}$. There is, however, an infinite model for these formulae, namely the set of numbers together with 0 and the successor function. □

We remark here that the logic of finite structures is not recursively enumerable if we have two unary relation symbols. (This is a theorem by (Trakhténbrodt, 1950).) However, the logic of all structures is clearly recursively enumerable, showing that the sets are different. This throws us into a dilemma: we can obviously not compute the meanings of formulae in a structure directly, since quantification requires search throughout the entire structure. (This problem has once worried some logicians, see (**?**). Nowadays it is felt that these are not problems of logic proper.) So, once again we have to actually try out another *semantics*.

The first route is to let a formula denote the set of all formulae that are equivalent to it. Alternatively, we may take the set of all formulae that follow from it. (These are almost the same in boolean logic. For example, $\varphi \leftrightarrow \chi$ can be defined using $\to$ and $\wedge$; and $\varphi \to \chi$ can be defined by $\varphi \leftrightarrow (\varphi \wedge \chi)$. So these approaches are not very different. However the second one is technically speaking more elegant.) This set is again infinite. Hence, we do something different. We shall take a formula to denote any formula that follows from it. Before we start we seize the opportunity to introduce a more abstract theory. A **propositional language** is a language of formulas generated by a set V of variables and a signature. The identity of V is the same as for boolean logic above. As usual, propositions are considered here as certain strings. The language is denoted by the letter L. A **substitution** is given by a map $\sigma : V \to L$. σ defines a map from L to L by replacement of occurrences of variables by their σ–image. We denote by φ^σ the result of substituting via σ.

Definition 4.1.2 *A **consequence relation over** L is a relation $\vdash \subseteq \wp(L) \times L$ such that the following holds. (We write $\Delta \vdash \varphi$ for the more complicated $\langle \Delta, \varphi\rangle \in \vdash$.)*

1. $\varphi \vdash \varphi$.
2. *If* $\Delta \vdash \varphi$ *and* $\Delta \subseteq \Delta'$ *then* $\Delta' \vdash \varphi$.
3. *If* $\Delta \vdash \chi$ *and* $\Sigma; \chi \vdash \varphi$, *then* $\Delta; \Sigma \vdash \varphi$.

*$\vdash$ is called **structural** if from $\Delta \vdash \varphi$ follows $\Delta^\sigma \vdash \varphi^\sigma$ for every substitution. $\vdash$ is **finitary** if $\Delta \vdash \varphi$ implies that there is a finite subset Δ' of Δ such that $\Delta' \vdash \varphi$.*

In the sequel consequence relations is always assumed to be structural. A rule is an element of $\wp(L) \times L$, that is, a pair $\rho = \langle \Delta, \varphi\rangle$. ρ is **finitary** if Δ is finite; it is n–**ary** if $|\Delta| = n$. Given a set R of rules, we call $\vdash^R$ the least structural consequence relation containing R. This relation can be explicitly defined. Say that χ is a **1–step** R–**consequence** of Σ if there is a substitution σ of some rule $\langle \Delta, \varphi\rangle \in R$ such that $\Delta^\sigma \subseteq \Sigma$ and $\chi = \varphi^\sigma$. Then, an n–**step consequence** of Σ is inductively defined.

Proposition 4.1.3 *$\Delta \vdash^R \varphi$ if and only if there is a natural number n such that φ is an n–step R–consequence of Δ.*

The reader may also try to generalize the notion of a proof from a Hilbert calculus and show that they define the same relation on condition that the rules are all finitary. We shall also give an abstract semantics and show its completeness. The notion of an Ω–algebra has been defined.

Definition 4.1.4 *Let L be a propositional logic over the signature Ω. A **matrix** L and $\vdash$ is a pair $\mathfrak{M} = \langle \mathfrak{A}, D\rangle$, where $\mathfrak{A}$ is an Ω–algebra (the algebra of **truth values**) and D a subset of A, called the set of **designated truth values**. Let h be a homomorphism from $\mathfrak{Tm}_\Omega(V)$ into $\mathfrak{M}$. We write $\langle \mathfrak{M}, h\rangle \vDash \varphi$ if $h(\varphi) \in D$ and say that φ is **true under** h **in** $\mathfrak{M}$. Further, we write $\Delta \vDash_{\mathfrak{M}} \varphi$ if for all homomorphisms $h : \mathfrak{Tm}_\Omega(V) \to \mathfrak{A}$: if $h[\Delta] \subseteq D$ then $h(\varphi) \in D$.*

Proposition 4.1.5 *If $\mathfrak{M}$ is a matrix for L, $\vDash_{\mathfrak{M}}$ is a structural consequence relation.*

Notice that in boolean logic $\mathfrak{A}$ is the 2–element boolean algebra and $D = \{1\}$, but we shall encounter other cases later on. Here is a general method for obtaining matrices.

Definition 4.1.6 *Let L be a propositional language and $\vdash$ a consequence relation. Denote by $\Delta^\vdash := \{\varphi : \Delta \vdash \varphi\}$. A set of formulae Δ is **deductively closed** if $\Delta = \Delta^\vdash$. Δ is **consistent** if $\Delta^\vdash \neq L$. It is **maximally consistent** if it is consistent but not proper superset is.*

A matrix is **canonical** for $\vdash$ if it has the form $\mathfrak{S} = \langle \mathfrak{Tm}_\Omega(V), \Delta^\vdash\rangle$ for some set Δ. (Here, $\mathfrak{Tm}_\Omega(V)$ is the canonical algebra with carrier set L whose functions are just the associated string functions.) Given $\vdash$ it is straightforward to verify that $\vdash \subseteq \vDash_{\mathfrak{S}}$. Now consider some set Δ and a formula such that $\Delta \nvdash \varphi$. Then put $\mathfrak{S} := \langle \mathfrak{Tm}_\Omega(V), \Delta^\vdash\rangle$ and let h be the identity. Then $h[\Delta] = \Delta \subseteq \Delta^\vdash$, but $h(\varphi) \notin \Delta^\vdash$ by definition of $\Delta^\vdash$. So, $\Delta \nvDash_{\mathfrak{S}} \varphi$. This shows the following.

Theorem 4.1.7 (Completeness of Matrix Semantics) *Let $\vdash$ be a structural consequence relation over L. Then*

$$\vdash = \bigcap \langle \vDash_{\mathfrak{S}} : \mathfrak{S} \textit{ canonical for } \vdash \rangle$$

(The reader may verify that an arbitrary intersection of consequence relations again is a consequence relation.) This theorem establishes that for any consequence relation we can find enough matrices such that they together characterize that relation. We shall notice also the following. Given $\vdash$ and $\mathfrak{M} = \langle \mathfrak{A}, D \rangle$, then $\vDash_{\mathfrak{M}} \supseteq \vdash$ if and only if D is closed under the consequence. (This is pretty trivial: all it says is that if $\Delta \vdash \varphi$ and h is a homomorphism, then if $h[\Delta] \subseteq D$ we must have $h(\varphi) \in D$.) Such sets are called **filters**. Now, let $\mathfrak{M} = \langle \mathfrak{A}, D \rangle$ be a matrix, and Θ a congruence on $\mathfrak{A}$. Suppose that for any x: $[x]_\Theta \subseteq D$ or $[x]_\Theta \cap D = \varnothing$. Then we call Θ **admissible for** $\mathfrak{M}$ and put $\mathfrak{M}/\Theta := \langle \mathfrak{A}/\Theta, D/\Theta \rangle$, where $D/\Theta := \{[x]_\Theta : x \in D\}$. The following is easy to show.

Proposition 4.1.8 *Let $\mathfrak{M}$ be a matrix and Θ an admissible congruence on $\mathfrak{M}$. Then $\vDash_{\mathfrak{M}/\Theta} = \vDash_{\mathfrak{M}}$.*

Finally, call a matrix **reduced** if only the diagonal is an admissible congruence. Then, by Proposition 4.1.8 and Theorem 4.1.7 we immediately derive that every consequence relation is complete with respect to reduced matrices. One also calls a class of matrices $\mathcal{K}$ a (**matrix**) **semantics** and says $\mathcal{K}$ is **adequate for** a consequence relation $\vdash$ if $\vdash = \bigcap_{\mathfrak{M} \in \mathcal{K}} \vDash_{\mathfrak{M}}$.

Now, given L and $\vdash$, the system of signs for the consequence relation is this.

$$\Sigma_P := \{\langle \vec{x}, R, \vec{x} \rangle : \vec{x} \in G\} \cup \{\langle \vec{x}, P, \vec{y} \rangle : \vec{x} \vdash \vec{y}\}$$

How does this change the situation? Notice that we can axiomatize the consequences by means of rules. The following is a set of rules that fully axiomatize the consequence. The proof of that will be left to the reader (see the exercises), since it is only peripheral to our interests.

$$\begin{array}{lcl}
\rho_d & := & \langle\{\texttt{(}\neg\texttt{(}\neg\texttt{p))}\}, \texttt{p}\rangle, \\
\rho_{dn} & := & \langle\{\texttt{p}\}, \texttt{(}\neg\texttt{(}\neg\texttt{p))}\rangle, \\
\rho_u & := & \langle\{\texttt{p}, \texttt{(}\neg\texttt{p)}\}, \texttt{p0}\rangle \\
\rho_c & := & \langle\{\texttt{p}, \texttt{p0}\}, \texttt{(p} \wedge \texttt{p0)}\rangle \\
\rho_{p0} & := & \langle\{\texttt{(p} \wedge \texttt{p0)}\}, \texttt{p}\rangle \\
\rho_{p1} & := & \langle\{\texttt{(p} \wedge \texttt{p0)}\}, \texttt{p0}\rangle \\
\rho_{mp} & := & \langle\{\texttt{p}, \texttt{(}\neg\texttt{(p} \wedge \texttt{(}\neg\texttt{p0)))}\}, \texttt{p0}\rangle
\end{array}$$

There is a finite set U of rules such that $\Delta \vdash^B \chi$ if and only if $\chi \in \Delta^{\vdash^U}$. With each rule we can actually associate a mode. We only give examples, since the general scheme for defining modes is easily extractable.

$$\begin{array}{lcl}
F_d(\langle\vec{x}, P, \texttt{(}\neg\texttt{(}\neg\vec{y}\texttt{))}\rangle & := & \langle\vec{x}, P, \vec{y}\rangle \\
F_c(\langle\vec{x}, P, \vec{y}\rangle, \langle\vec{x}, P, \vec{z}\rangle) & := & \langle\vec{x}, P, \texttt{(}\vec{y} \wedge \vec{z}\texttt{)}\rangle
\end{array}$$

If we have $\rightarrow$, then the following mode corresponds to the rule ρ_{mp}, Modus Ponens.

$$\texttt{F}_{mp}(\langle\vec{x}, P, \texttt{(}\vec{y} \rightarrow \vec{z}\texttt{)}\rangle, \langle\vec{x}, P, \vec{y}\rangle) := \langle\vec{x}, P, \vec{z}\rangle$$

This is satisfactory in that it allows to derive all and only the consequences of given proposition. A drawback is that the functions on the exponents are nonincreasing. They always return $\vec{x}$. The structure term of the sign $\langle\vec{x}, P, \vec{y}\rangle$ on the other hand encodes a derivation of $\vec{y}$ from $\vec{x}$.

Now, the reader may get worried by the proliferation of different semantics. Aren't we always solving a different problem? Our answer is indirect. The problem is that we do not know exactly what meanings are. Given a natural language, what we can observe more or less directly is the exponents. Although it is not easy to write down rules that generate them, the entities are more or less concrete. A little less concrete are the syntactic categories. We have already seen in the previous chapter that the assignment of categories to strings (or other exponents, see next chapter) are also somewhat arbitrary. We shall return to this issue. Even less clearly definable, however, are the meanings. What, for example, is the meaning of (4.1.1)?

(4.1.1) `Caesar crossed the Rubicon.`

The first answer we have given was: a truth value. For this sentence is either true or false. But even though it is true, it might have been false, just in case Caesar did not cross the Rubicon. What makes us know this? The second answer (for first–order theories) was: the meaning is a set of models. Knowing what the model is and what the variables are assigned to, we know whether that sentence is true. But we simply cannot look at all models, and still it seems that we know what (4.1.1) means. So then, the next answer was: its meaning is an algorithm, which, given a model, tells us whether the sentence is true. Then, finally, we do not have to know everything in order to know whether (4.1.1) is true. Most facts are irrelevant, for example, whether Napoleon was French. On the other hand, suppose we witness how Caesars walk across the Rubicon, or suppose we know that first he was north of the Rubicon and the next day to the south of it. This will make us think that (4.1.1) is true. Thus, the algorithm that computes the truth value does not need all of a model; a small part of it actually suffices. We can introduce partial models and define algorithms on them, but all this is a variation on the same theme. A different approach is provided by our last answer: a sentence means whatever it implies.

We may cast this as follows. Start with the set L of propositions and a set (or class) $\mathcal{M}$ of models. A **primary** (or **model theoretic**) **semantics** is given in terms of a relation $\vDash \subseteq L \times \mathcal{M}$. Most approaches are variants of the primary semantics, since they more or less characterize meanings in terms of facts. However, from this semantics we may define a **secondary semantics**, which is the semantics of consequence. $\Delta \vDash \varphi$ if and only if for all $M \in \mathcal{M}$: if $M \vDash \delta$ for all $\delta \in \Delta$ then $M \vDash \varphi$. (We say in this case that Δ entails φ.) Secondary semantics is concerned only with the relationship between the objects of the language, there is no model involved. It is clear that the secondary semantics is not fully adequate. Notice namely that knowing the logical relationship between sentences does not reveal anything about the nature of the models. Second, even if we knew what the models are: we could not say whether a given sentence is true in a given model or not. It is perfectly conceivable that we know English to the extent that we know which sentences entail which other sentences, but still we are unable to say, for example, whether or not (4.1.1) was true even when we witness Caesar cross the Rubicon. An example might make this clear. Suppose that the house is on fire. Then I know that I am in danger and I run away. However, suppose that someone shouts (4.1.2) at me. Then I can infer that he thinks (4.1.2) is true. This will make me believe that (4.1.2) is true and even that (4.1.3) is true as well. But I do not know that the house is on fire, and I keep on working, since I also do not know that I am in danger.

(4.1.2) `The house is on fire.`
(4.1.3) `I am in danger.`

Therefore, knowing how sentences hang together in a deductive system has little to do with the actual world. The situation is not simply remedied by knowing some of the meanings. Suppose I additionally know that (4.1.2) means that the house is on fire. Then if I see that the house is on fire then I know that I am in danger, and I also know that (4.1.3) is the case. But I still may fail to see that (4.1.3) means that I am in danger. It may just mean something else that is being implied by (4.1.2). This is reminiscent of Searle's thesis that language is about the world: knowing what things mean is not constituted by an ability to manipulate certain symbols. We may phrase this as follows.

> *Indeterminacy of secondary semantics.* No secondary semantics can fix the truth conditions of propositions uniquely for any given language.

Searle's claims go further than that, but this much is perhaps quite uncontroversial. Despite the fact that secondary semantics is underdetermined, we shall not deal with primary semantics at all. We are not going to discuss what a word, say, **life** *really* means — we are only interested in how its meaning functions language internally. Formal semantics really cannot do more that that. In what is to follow, we shall sketch an algebraic approach to semantics. This contrasts with the far more widespread model–theoretic approach. The latter may be more explicit and intuitive, but on the other hand turns out to be very inflexible.

We begin by examining a very influential principle in semantics, the Leibniz' Principle. We quote one of its original formulation from (Leibniz, 2000)). *Eadem vel Coincidentia sunt quae sibi ubique substitui possunt salva veritate. Diversa quae non possunt.* Translated it says: *The same or coincident are those which can everywhere be substituted for each other not affecting truth. Different are those that cannot.* Clearly, substitution must be understood here in the context of sentences, and we must assume that what we substitute is constituent occurrences of the expressions. We therefore reformulate the principle as follows.

> *Leibniz' Principle.* Two expressions A and B have the same meaning if and only if in every sentence any occurrence of A can be substituted by B and any occurrence of B by A without changing the truth of that sentence.

To some people this principle seems to assume bivalence. If there are more than two truth values we may interpret Leibniz' original definition as saying that substitution does not change the truth value rather than just the truth. (See also Lyons for a discussion.) First we give some unproblematic examples. In second order logic (SO, see Chapter 6.1), the following is a theorem.

$$(\ddagger) \qquad (\forall x)(\forall y)(x \doteq y \leftrightarrow (\forall P)(P(x) \leftrightarrow P(y)))$$

Hence, Leibniz' Principle holds of second order logic with respect to terms. There is general no identity relation for predicates, but if there is, it is defined according to Leibniz' Principle: two predicates are equal if and only if they hold of the same individuals. This requires full second order logic, for what we want to have is the following for each $n \in \omega$ (with P_n and Q_n variables for n–ary relations):

$$(\dagger_n) \qquad (\forall P_n)(\forall Q_n)(P_n \doteq Q_n \leftrightarrow (\forall \vec{x})(P_n(\vec{x}) \leftrightarrow Q_n(\vec{x})))$$

(Here, $\vec{x}$ abbreviates the n–tuple $x_0, \dots, x_{n-1}$.) (‡) is actually the basis for Montague's type raising. Recall that Montague identified an individual with the set of all of its properties. In virtue of (‡) this identification does not conflate distinct individuals. To turn that around: by Leibniz' Principle, this identification is one–to–one. We shall see in the next section that boolean algebras of any kind can be embedded into powerset algebras. The background of this proof is the result that if there are two elements x, y in a boolean algebra $\mathfrak{B}$ and for all homomorphisms $h : \mathfrak{B} \to \mathbf{2}$ we have $h(x) = h(y)$, then $x = y$. (More on that in the next section. We have to use homomorphisms here since properties are functions that commute with the boolean operations, that is to say, homomorphisms.) Thus, Leibniz' Principle also holds for boolean semantics, defined in Section 4.5. Notice that the proof relies on the Axiom of Choice (in fact the somewhat weaker Prime Ideal Axiom), so it is not altogether innocent.

We use Leibniz Principle to detect whether two items have the same meaning. Notice that Leibniz' Principle has for consequence that semantics is essentially unique. If $\mu : A^* \overset{p}{\twoheadrightarrow} M$, $\mu' : A^* \overset{p}{\twoheadrightarrow} M'$ are surjective functions assigning meanings to expressions, and if both satisfy Leibniz' Principle, then there is a bijection $\pi : M \to M'$ such that $\mu' = \pi \circ \mu$ and $\mu = \pi^{-1} \circ \mu'$. Thus, as far as formal semantics is concerned, any solution is as good any other.

As we have briefly mentioned in Section 3.5, we may use the same idea to define types. This method goes back to Husserl, and is a key ingredient to the theory of compositionality by Wilfrid Hodges (see his (2001)). A type is a class of expressions that can be substituted for each other without changing meaningfulness.

Hodges just uses pairs of exponents and meanings. If we want to assimilate his setup to ours, we may add a category U, and let for every mode F, $F^\tau(U, \ldots, U) := U$. However, the idea is to do without categories. If we further substract the meanings, we get what Hodges calls a *grammar*. We prefer to call it an **H–grammar**. (The letter **H** honours Hodges here.) Thus, an H–grammar is defined by some signature and corresponding operations on the set E of exponents, which may even be partial. An **H–semantics** is a partial map μ from the structure terms (!) to a set M of meanings. Structure terms $\mathfrak{s}$ and $\mathfrak{t}$ are **synonymous** if μ is defined on both and $\mu(\mathfrak{s}) = \mu(\mathfrak{t})$. We write $\mathfrak{s} \equiv_\mu \mathfrak{t}$ to say that $\mathfrak{s}$ and $\mathfrak{t}$ are synonymous. (Notice that $\mathfrak{s} \equiv_\mu \mathfrak{s}$ if and only if μ is defined on $\mathfrak{s}$.) An H–semantics ν is **equivalent** to μ if $\equiv_\mu \; = \; \equiv_\nu$. An **H–synonymy** is an equivalence relation on a subset of the set of structure terms. We call that subset the **field** of the H–synonymy. Given an H–synonymy $\equiv$, we may define M to be the set of all equivalence classes of $\equiv$, and set $\mu^\equiv(\mathfrak{s}) := [\mathfrak{s}]_\equiv$ if and only if $\mathfrak{s}$ is in that subset, and undefined otherwise. Thus, up to equivalence, H–synonymies and H–semantics are in one–to–one correspondence. We say that $\equiv'$ **extends** $\equiv$ if the field of $\equiv'$ contains the field of $\equiv$, and the two coincide on the field of $\equiv$.

Definition 4.1.9 *Let G be an H–grammar and μ an H–semantics for it. We write $\mathfrak{s} \sim_\mu \mathfrak{s}'$ if and only if for every structure term $\mathfrak{t}(x)$ with a single variable x, $[\mathfrak{s}/x]\mathfrak{t}$ is μ–meaningful if and only if $[\mathfrak{s}'/x]\mathfrak{t}$ is μ–meaningful. The equivalence classes of $\sim_\mu$ are called the μ–**categories**.*

This is the formal rendering of the 'meaning categories' that Husserl defines.

Definition 4.1.10 *ν and its associated synonymy is called μ–**Husserlian** if for all structure terms $\mathfrak{s}$ and $\mathfrak{s}'$: if $\mathfrak{s} \equiv_\nu \mathfrak{s}'$ then $\mathfrak{s} \sim_\mu \mathfrak{s}'$. μ is called **Husserlian** if it is μ–Husserlian.*

It is worthwhile comparing this definition with the Leibniz' Principle. Leibniz' Principle defines identity in meaning via intersubstitutability in all sentences; what must remain constant is truth.

Husserl's meaning categories are also defined by intersubstitutability in all sentences; however, what must remain constant is the meaningfulness. We may connect these principles as follows.

Definition 4.1.11 *Let Σ be a set of sentential terms and $\Delta = \{\mathfrak{s} : \mu(\mathfrak{s}) = 1\} \subseteq \Sigma$. We call $\mathfrak{s}$* ***sentential*** *if $\mathfrak{s} \in \Sigma$, and* ***true*** *if $\mathfrak{s} \in \Delta$. μ is* ***Leibnizian*** *if for all structure terms $\mathfrak{u}$ and $\mathfrak{u}'$: $\mathfrak{u} \equiv_\mu \mathfrak{u}'$ if and only if for all structure terms $\mathfrak{s}$ such that $[\mathfrak{u}/x]\mathfrak{s} \in \Delta$ also $([\mathfrak{u}'/x]\mathfrak{s}) \in \Delta$ and conversely.*

Under mild assumptions on μ it holds that Leibnizian implies Husserlian. The following is from (Hodges, 2001).

Theorem 4.1.12 (Hodges) *Let μ be an H–semantics for the H–grammar G. Suppose further that subterm of a μ–meaningful structure term is again μ–meaningful. Then the following are equivalent.*

1. *For each mode F there is an $\Omega(F)$–ary function $F^\mu : M^{\Omega(F)} \to M$ such that μ is a homomorphism of partial algebras.*
2. *If $\mathfrak{s}$ is a structure term and $\mathfrak{u}_i$, $\mathfrak{v}_i$ $(i < n)$ are structure terms such that $[\mathfrak{u}_i/x_i : i < n]\mathfrak{s}$ and $[\mathfrak{v}_i/x_i : i < n]\mathfrak{s}$ are both μ–meaningful and if for all $i < n$ $\mathfrak{u}_i \equiv_\mu \mathfrak{v}_i$ then*
$$[\mathfrak{u}_i/x_i : i < n]\mathfrak{s} \equiv_\mu [\mathfrak{v}_i/x_i : i < n]\mathfrak{s} \ .$$

Furthermore, if μ is Husserlian then the second already holds if it holds for $n = 1$.

Proof. The first implies the second. Suppose that such function can be defined. We show the claim by induction on $\mathfrak{s}$. If $\mathfrak{s}$ is a basic term, nothing is to be shown. So, assume that $\mathfrak{s} = F\mathfrak{t}_0 \ldots t_{m-1}$. By our assumption $\mathfrak{t}_i$, $i < m$, is μ–meaningful. Hence, by inductive hypothesis, for all $j < m$

$$[\mathfrak{u}_i/x_i : i < n]\mathfrak{t}_j \equiv_\mu [\mathfrak{v}_i/x_i : i < n]\mathfrak{t}_j \ .$$

This means in effect that both have the same H–meaning for each $j < m$. It follows that

$$[\mathfrak{u}_i/x_i : i < n]\mathfrak{s} \equiv_\mu [\mathfrak{v}_i/x_i : i < n]\mathfrak{s} \ .$$

For the converse, define F^μ as follows.

$$F^\mu(\mu(\mathfrak{t}_0), \ldots, \mu(\mathfrak{t}_{m-1})) := \mu(F\mathfrak{t}_0 \ldots \mathfrak{t}_{m-1}) \ .$$

By our assumptions on μ this is well–defined.

Finally, we need to see why the property of being Husserlian allows to deduce (2) from its special case $n = 1$. We define $\mathfrak{s}^j$ to be $[\mathfrak{u}_i/x_i : i < j][\mathfrak{v}_i/x_i : j \leq i < n]\mathfrak{s}$. Let $j < m$. Suppose that $\mathfrak{s}^j$ is μ–meaningful. Then, from the fact that μ is Husserlian we deduce that $\mathfrak{s}^{j+1}$ is also μ–meaningful and $\mathfrak{s}^j \equiv_\mu \mathfrak{s}^{j+1}$. Now, $\mathfrak{s}^0$ is μ–meaningful by assumption. Then so is $\mathfrak{s}^j$ for every $j < m$, and we get $\mathfrak{s}^j \equiv \mathfrak{s}^k$ for all $j, k < m$. □

If $\bullet$ is the only operation, we can actually use this method to defined the types (see Section 3.5). In the following sections we shall develop an algebraic account of semantics, starting first with boolean algebras and then going over to intensionality, and finally carrying out the full algebraization.

Notes on this section. We shall briefly mention a different issue that is quite important for semantics although it rarely receives attention. The algorithms that we define typically allow to say what things an expression denotes, and give more or less direct translation algorithms. However, language users do not only understand sentences, they can also produce them. Sentence production, however, starts with an intention to communicate a certain content, and proceeds to the utterance of a sentence that expresses this content. This requires the ability to express ideas by means of words. This, however, is a fairly difficult matter unless we assume that our ideas (or mental representations) are more or less structured in the same way as the sentences that express them. So, the translation of mental representations into sentences can proceed in an algorithmic way. Unfortunately, there is little of substance that we are going to say about that in the sequel.

Exercise 129. Prove Proposition 4.1.8.

Exercise 130. Let $\rho = \langle \Delta, \varphi \rangle$ be a rule. Devise a mode $\mathtt{M}_\rho$ that captures the effect of this rule in the way discussed above. Translate the rules given above into modes. What happens with 0–ary rules (that is, rules with $\Delta = \varnothing$)?

Exercise 131. There is a threefold characterization of a consequence: as a consequence relation, as a closure operator, and as a set of theories. Let $\vdash$ be a consequence relation. Show that $\Delta \mapsto \Delta^{\vdash}$ is a closure operator. The closed sets are the theories. If $\vdash$ is structural the set of theories of $\vdash$ are inversely closed under substitution. That is to say, if T is a theory and σ a substitution, then $\sigma^{-1}[T]$ is a theory as well. Conversely, show that every closure operator on $\wp(\mathfrak{Tm}_\Omega(V))$ gives rise to a consequence relation and that the consequence relation is structural if the set of theories is inversely closed under substitutions.

Exercise 132. Show that the rules given above are complete for boolean logic in $\wedge$ and $\neg$.

Exercise 133. Show that for any given finite signature the set of predicate logical formulae valid in all finite structures for that signature is co–recursively enumerable. (The latter only means that its complement is recursively enumerable.)

Exercise 134. Let L be a first–order language which contains at least the symbol for equality $\doteq$. Show that a first–order theory T in L satisfies Leibniz' Principle if the following holds for any relation symbol R

$$T; \{x_i \doteq y_i : i < \Xi(R)\} \vdash R(\vec{x}) \leftrightarrow R(\vec{y})$$

and the following for every function symbol f:

$$T; \{x_i \doteq y_i : i < \Omega(f)\} \vdash f(\vec{x}) \doteq f(\vec{y})$$

Use this to show that the first–order set theory ZFC satisfies Leibniz' Principle. Further, show that every equational theory satisfies

Leibniz' Principle.

4.2 Boolean Semantics

Boolean semantics is actually the title of a book by Keenan and Faltz (1985). However, it is much less of a particular semantical theory than an approach to semantics in general and uses techniques that are needed in all areas of semantics. Therefore we have decided to provide a section devoted entirely to boolean semantics. A few definitions for a start.

Definition 4.2.1 *A* ***boolean algebra*** *is an algebra* $\langle B, 0, 1, -, \cap, \cup\rangle$*, where* $0, 1 \in B$*,* $- : B \to B$ *and* $\cap, \cup : B^2 \to B$*, which satisfies the following equations for all* $x, y, z \in B$*.*

(as∩)	$x \cap (y \cap z) = (x \cap y) \cap z$	(as∪)	$x \cup (y \cup z) = (x \cup y) \cup z$
(co∩)	$x \cap y = y \cap x$	(co∪)	$x \cup y = y \cup x$
(id∩)	$x \cap x = x$	(id∪)	$x \cup x = x$
(ab∩)	$x \cap (y \cup x) = x$	(ab∪)	$x \cup (y \cap x) = x$
(di∩)	$x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$	(di∪)	$x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$
(li−)	$x \cap (-x) = 0$	(ui∪)	$x \cup (-x) = 1$
(ne∩)	$x \cap 1 = x$	(ne0)	$x \cup 0 = x$
(dm∩)	$-(x \cap y) = (-x) \cup (-y)$	(dm∪)	$-(x \cup y) = (-x) \cap (-y)$
(dn−)	$-(-x) = x$		

The operation $\cap$ is generally referred to as the **meet** (**operation**) and $\cup$ as the **join** (**operation**). $-x$ is called the **complement** of x and 0 the **zero** and 1 the **one**.

The laws (as∩) and (as∪) are called **associativity laws**, the laws (co∩) and (co∪) **commutativity laws**, (id∩) and (id∪) the **laws of idempotence** and (ab∩) and (ab∪) the **laws of absorption**. A structure $\langle L, \cap, \cup\rangle$ satisfying these laws is called a **lattice**. If only one operation is present and the corresponding laws hold we speak of a **semilattice**. (So, a semilattice is a semigroup that satisfies commutativity and idempotence.) Since $\cap$ and $\cup$ are associative and commutative, we follow the general practice and omit brackets whenever possible. So, rather than $(x \cap (y \cap z))$ we simply write $x \cap y \cap z$. Also, $(x \cap (y \cap x))$ is simplified to $x \cap y$. Furthermore, given a finite set $S \subseteq L$ the notation $\bigcup\langle x : x \in S\rangle$ or simply $\bigcup S$ is used for the iterated join of the elements of S. This is uniquely defined, since the join is independent of the order and multiplicity in which the elements appear.

Definition 4.2.2 *Let $\mathfrak{L}$ be a lattice. We write $x \leq y$ if $x \cup y = y$.*

Notice that $x \leq y$ if and only if $x \cap y = x$. This can be shown using the equations above. We leave this as an exercise to the reader. Notice also the following.

Lemma 4.2.3 *1. $\leq$ is a partial ordering.*

2. $x \cup y \leq z$ if and only if $x \leq z$ and $y \leq z$.

3. $z \leq x \cap y$ if and only if $z \leq x$ and $z \leq y$.

Proof. (1.) (a) $x \cup x = x$, whence $x \leq x$. (b) Suppose that $x \leq y$ and $y \leq x$. Then we get $x \cup y = x$ and $y \cup x = y$, whence $y = x \cup y = x$. (c) Suppose that $x \leq y$ and $y \leq z$. Then $x \cup y = y$ and $y \cup z = z$ and so $x \cup z = x \cup (y \cup z) = (x \cup y) \cup z = y \cup z = z$. (2.) Let $x \cup y \leq z$. Then, since $x \leq x \cup y$, we have $x \leq z$ by (1c) and by the same reason also $y \leq z$. Now assume $x \leq z$ and $y \leq z$. Then $x \cup z = y \cup z = z$ and so $z = z \cup z = (x \cup z) \cup (y \cup z) = (x \cup y) \cup z$, whence $x \cup y \leq z$. (3.) Similarly. □

In fact, it is customary to define a lattice by means of $\leq$. This is done as follows.

Definition 4.2.4 *Let $\leq$ be a partial order on L. Let $X \subseteq L$ be an arbitrary set. The* ***greatest lower bound (glb)*** *of X, also denoted $\bigcap X$, is that element u such that for all z: if $x \geq z$ for all $x \in X$ then also $u \geq z$. Analogously, the* ***least upper bound (lub)*** *of X, denoted by $\bigcup X$, is that element v such that for all z: if $x \leq z$ for all $x \in X$ then also $v \leq z$.*

By the facts established above, the join of two elements x and y is simply the lub of $\{x, y\}$, and the meet is the glb of $\{x, y\}$. It is left to the reader to verify that these operations satisfy all laws of lattices. So, a partial order $\leq$ is the order determined by a lattice structure if and only if all finite sets have a least upper bound and a greatest lower bound.

The laws (di$\cap$) and (di$\cup$) are the **distributivity laws**. A lattice is called **distributive** if they hold in it. A nice example of a distributive lattice is the following. Take a natural number, say 28, and list all divisors of it: 1, 2, 4, 7 , 14, 28. Write $x \leq y$ if x is a divisor of y. (So, $2 \leq 14$, $2 \leq 4$, but not $4 \leq 7$.) Then $\cap$ turns out to be the greatest common divisor and $\cup$ the least common multiple. Another example is the linear lattice defined by the numbers $< n$ with $\leq$ the usual ordering. $\cap$ is then the minimum and $\cup$ the maximum.

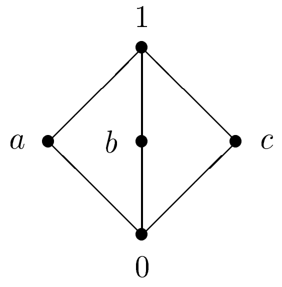
A **bounded lattice** is a structure $\langle L, 0, 1, \cap, \cup \rangle$ which is a lattice with respect to $\cap$ and $\cup$, and in which satisfies (ne$\cap$) and (ne$\cup$). From the definition of $\leq$, (ne$\cap$) means that $x \leq 1$ for all x and (ne$\cup$) that $0 \leq x$ for all x. Every finite lattice has a least and a largest element and can thus be extended to a bounded lattice. This extension is usually done without further notice.

Definition 4.2.5 *Let $\mathfrak{L} = \langle L, \cap, \cup \rangle$ be a lattice. An element x is **join irreducible in $\mathfrak{L}$** if for all y and z such that $x = y \cup z$ either $x = y$ or $x = z$. x is **meet irreducible** if for all y and z such that $x = y \cap z$ either $x = y$ or $x = z$.*

It turns out that in a distributive lattice irreducible elements have a stronger property. Call x **meet prime** if for all y and z: from $x \geq y \cap z$ follows $x \geq y$ or $x \geq z$. Obviously, if x is meet prime it is also meet irreducible. The converse is generally false. Look at M_3 shown in Figure 4.1. Then $c \geq a \cap b (= 0)$, but neither $c \geq a$ nor $c \geq b$ holds.

Lemma 4.2.6 *Let $\mathfrak{L}$ be a distributive lattice. Then x is meet (join) prime if and only if x is meet (join) irreducible.*

Figure 4.1: The Lattice M_3



Let us now move on to the complement. (li$\cap$) and (ui$\cup$) have no special name. They basically ensure that $-x$ is the unique element y such that $x \cap y = 0$ and $x \cup y = 1$. The laws (dm$\cap$) and (dm$\cup$) are called **de Morgan laws**. Finally, (dn$-$) is the law of **double negation**.

Lemma 4.2.7 *The following holds in a boolean algebra.*

1. *$x \leq y$ if and only if $-y \leq -x$.*

2. *$x \leq y$ if and only if $x \cap (-y) = 0$ if and only if $(-x) \cup y = 1$.*

Proof. (1.) $x \leq y$ means $x \cup y = y$, and so $-y = -(x \cup y) = (-x) \cap (-y)$, whence $-y \leq -x$. Conversely, from $-y \leq -x$ we get by the same argument $x = --x \leq --y = y$. (2.) If $x \leq y$ then $x \cap y = x$, and so $x \cap (-y) = (x \cap y) \cap (-y) = x \cap 0 = 0$. Conversely, suppose that $x \cap (-y) = 0$. Then $x \cap y = (x \cap y) \cup (x \cap (-y)) = x \cap (y \cup (-y)) = x \cap 1 = x$. So, $x \leq y$. It is easily seen that $x \cap (-y) = 0$ if and only if $(-x) \cup y = 1$. $\Box$

We can apply the general terminology of universal algebra (see Section 1.1). So, the notions of homomorphisms and subalgebras, congruences, of these structures should be clear. We now give some examples of boolean algebras. The first example is the powerset of a given set. Let X be a set. Then $\wp(X)$ is a boolean algebra with $\varnothing$ in place of 0, X in place of 1, $-A = X - A$, $\cap$ and $\cup$

the intersection and union. We write $\mathfrak{P}(X)$ for this algebra. A subalgebra of this algebra is called a **field of sets**. Also, a subset of $\wp(X)$ closed under the boolean operations is called a field of sets. The smallest examples are the algebra $\mathbf{1} := \mathfrak{P}(\varnothing)$, consisting just of one element ($\varnothing$), and $\mathbf{2} := \mathfrak{P}(\{\varnothing\})$, the algebra of subsets of $1 = \{\varnothing\}$. Now, let X be a set and $\mathfrak{B} = \langle B, 0, 1, \cap, \cup, -\rangle$ be a boolean algebra. Then for two functions $f, g : X \to B$ we may define $-f$, $f \cap g$ and $f \cap g$ as follows.

$$\begin{array}{lcl}(-f)(x) & := & -f(x) \\ (f \cup g)(x) & := & f(x) \cup g(x) \\ (f \cap g)(x) & := & f(x) \cap g(x)\end{array}$$

Further, let $\underline{0} : X \to B : x \mapsto 0$ and $\underline{1} : X \to B : x \mapsto 1$. It is easily verified that the set of all functions from X to B form a boolean algebra: $\langle B^X, \underline{0}, \underline{1}, \cap, \cup, -\rangle$. We denote this algebra by $\mathfrak{B}^X$. The notation has been chosen on purpose: this algebra is nothing but the direct product of $\mathfrak{B}$ indexed over X. A particular case is $\mathfrak{B} = \mathbf{2}$. Here, we may actually think of $f : X \to 2$ as the characteristic function χ_M of a set, namely the set $f^{-1}(1)$. It is then again verified that $\chi_{-M} = -\chi_M$, $\chi_{M\cap N} = \chi_M \cap \chi_N$, $\chi_{M\cup N} = \chi_M \cup \chi_N$. So we find the following.

Theorem 4.2.8 *$\mathbf{2}^X$ is isomorphic to $\mathfrak{P}(X)$.*

We provide some applications of these results. The intransitive verbs of English have the category $e\backslash t$. Their semantic type is therefore $e \to t$. This in turn means that they are interpreted as functions from objects to truth values. We assume that the truth values are just 0 and 1 and that they form a boolean algebra with respect to the operations $\cap$, $\cup$ and $-$. Then we can turn the interpretation of intransitive verbs into a boolean algebra in the way given above. Suppose that the interpretation of `and`, `or` and `not` is also canonically extended in the given way. That is: suppose that they can now also be used for intransitive verbs and have the meaning given above. Then we can account for a number of inferences, such as the inference from (4.2.1) to (4.2.2) and (4.2.3), and from (4.2.2) and (4.2.3) together to (4.2.1). Or we can infer that (4.2.1) implies that (4.2.4) is false; and so on.

(4.2.1) `Claver walks and talks.`
(4.2.2) `Claver walks.`
(4.2.3) `Claver talks.`
(4.2.4) `Claver does not walk.`

With the help of that we can now also assign a boolean structure to the transitive verb denotations. For their category is $(e\backslash t)/e$, which corresponds to the type $e \to (e \to t)$. Now that the set functions from objects to truth values carries a boolean structure, we may apply the construction again. This allows us then to deduce (4.2.6) from (4.2.5).

(4.2.5) `Claver sees or hears Patrick.`
(4.2.6) `Claver sees Patrick or Claver hears Patrick.`

Obviously, any category that finally ends in t has a space of denotations associated to it that can be endowed with the structure of a boolean algebra. (See also Exercise 3.8.) These are, however, not all categories. However, for the remaining ones we can use a trick used already by Montague. Montague was concerned with the fact that names such as `Peter` and `Susan` denote objects, which means that their type is e. Yet, they fill a subject NP position, and subject NP positions can also be filled by (nominative) quantified NPs such as `some philosopher`, which are of type $(e \to t) \to t$. In order to have homogeneous type assignment, Montague lifted the denotation of `Peter` and `Susan` to $(e \to t) \to t$. In terms of syntactic categories we lift from e to $t/(e\backslash t)$. We have met this earlier in Section 3.4 as raising. Cast in terms of boolean algebras this is the following construction. From an arbitrary set X we first form the boolean algebra $\mathfrak{P}(X)$ and then the algebra $\mathbf{2}^{\mathfrak{P}(X)}$.

Proposition 4.2.9 *The map $x \mapsto x^\dagger$ given by $x^\dagger(f) := f(x)$ is an embedding of X into $\mathbf{2}^{\mathfrak{P}(X)}$.*

Proof. Suppose that $x \neq y$. Then $x^\dagger(\chi_{\{x\}}) = \chi_{\{x\}}(x) = 1$, while $y^\dagger(\chi_{\{x\}}) = \chi_{\{x\}}(y) = 0$. Thus $x^\dagger \neq y^\dagger$. $\Box$

To see that this does the trick, consider the following sentence.

(4.2.7) `Peter and Susan walk.`

We interpret `Peter` now by $\mathsf{peter}'^{\dagger}$, where peter' is the individual Peter. Similarly, $\mathsf{susan}'^{\dagger}$ interprets `Susan`. Then (4.2.7) means

$$\begin{aligned}(\mathsf{peter}'^{\dagger} \cap \mathsf{susan}'^{\dagger})(\mathsf{walk}') &= (\mathsf{peter}'^{\dagger}(\mathsf{walk}')) \cap (\mathsf{susan}'^{\dagger}(\mathsf{walk}')) \\ &= \mathsf{walk}'(\mathsf{peter}') \cap \mathsf{walk}'(\mathsf{susan}')\end{aligned}$$

So, this licenses the inference from (4.2.7) to (4.2.8) and (4.2.9), as required. (We have tacitly adjusted the morphology here.)

(4.2.8) `Peter walks.`
(4.2.9) `Susan walks.`

It follows that we can make the denotations of any linguistic category a boolean algebra.

The next theorem we shall prove is that boolean algebras are (up to isomorphism) the same as fields of sets. Before we prove the full theorem, we shall prove a special case, which is very important in many applications. An **atom** is an element $x \neq 0$ such that for all $y \leq x$: either $y = 0$ or $y = x$. $\mathbf{At}(\mathfrak{B})$ denotes the set of all atoms of $\mathfrak{B}$.

Lemma 4.2.10 *In a boolean algebra, an element is an atom if and only if it is join irreducible.*

This is easy to see. An atom is clearly join irreducible. Conversely, suppose that x is join irreducible. Suppose that $0 \leq y \leq x$. Then $x \cap (-y) = 0$. Hence $x = x \cap 1 = x \cap (y \cup (-y)) = (x \cap y) \cup ((-y) \cap x) = y \cup (x \cap (-y)) = y$. So, x is an atom. Denote by $\widehat{x}$ the set of all atoms below x. We notice that in a finite boolean algebra this set is nonempty if and only if $x \neq 0$. Just take a minimal element $\leq x$ which is different from 0.

Lemma 4.2.11 *Let $\mathfrak{B}$ be a finite boolean algebra. Then for all x: $x = \bigcup\langle y : y \in \widehat{x}\rangle$.*

Proof. Let x' be the join of all atoms below x. Clearly, $x' \leq x$. Now suppose that $x' < x$. Then $(-x') \cap x \neq 0$. Hence there is an atom $u \leq (-x') \cap x$, whence $u \leq x$. But $u \nleq x'$, a contradiction. $\Box$

Theorem 4.2.12 *Let $\mathfrak{B}$ be a finite boolean algebra. The map $x \mapsto \widehat{x}$ is an isomorphism from $\mathfrak{B}$ onto $\mathfrak{P}(\mathbf{At}(\mathfrak{B}))$.*

Proof. The first to be shown is that the map is actually a homomorphism. To that effect, it must be shown that $\widehat{-x} = -\widehat{x}$, $\widehat{x \cap y} = \widehat{x} \cap \widehat{y}$ and $\widehat{x \cup y} = \widehat{y} \cup \widehat{y}$. This is left to the reader. Now we have to show that the map is an isomorphism. Consider two elements x, y such that $\widehat{x} = \widehat{y}$. Then $x = \bigcup \widehat{x} = \bigcup \widehat{y} = y$, by the previous lemma. So the map is injective. It is also surjective; for given any set U of atoms we can form the element $c := \bigcup U$. Since atoms are join irreducible, for any atom $x \leq c$ we must already have $x \leq u$ for some $u \in U$. But u also is an atom; hence $x = u$, showing that $\widehat{c} = U$. So, the map is surjective. $\Box$

Now we proceed to the general case. First, notice that this theorem is false in general. A subset N of M is called **cofinite** if its complement, $M - N$, is finite. Let Ω be the set of all subsets of ω which are either finite or cofinite. Now, as is easily checked, Ω contains $\varnothing$, ω and is closed under complement, union and intersection. The singletons $\{x\}$ are the atoms. However, not every set of atoms corresponds to an element of the algebra. A case in point is $\{\{2k\} : k \in \omega\}$. Its union in ω is the set of even numbers, which is neither finite nor cofinite. Moreover, there exists infinite boolean algebras that have no atoms (see the exercises). Hence, we must take a different route.

Definition 4.2.13 *Let $\mathfrak{B}$ be a boolean algebra. A* ***point*** *is a homomorphism $h : \mathfrak{B} \to \mathbf{2}$. The set of points of $\mathfrak{B}$ is denoted by $pt(\mathfrak{B})$.*

Notice that points are necessarily surjective. For we must have $h(0^{\mathfrak{B}}) = 0$ and $h(1^{\mathfrak{B}}) = 1$. (As a warning to the reader: we will usually not distinguish $1^{\mathfrak{B}}$ and 1.)

Definition 4.2.14 *A* ***filter*** *of $\mathfrak{B}$ is a subset that satisfies the following.*

* *$1 \in F$.*
* *If $x, y \in F$ then $x \cap y \in F$.*
* *If $x \in F$ and $x \leq y$ then $y \in F$.*

*A filter F is called an **ultrafilter** if $F \neq B$ and there is no filter G such that $F \subsetneq G \subsetneq B$.*

A filter F is an ultrafilter if and only if for all x: either $x \in F$ or $-x \in F$. For suppose neither is the case. Then let F^x be the set of elements y such that there is an $u \in F$ with $y \geq u \cap x$. This is a filter, as is easily checked. It is a proper filter: it does not contain $-x$. For suppose otherwise. Then $-x \geq u \cap x$ for some $u \in F$. By Lemma 4.2.7 this means that $0 = u \cap x$, from which we get $u \leq -x$. So, $-x \in F$, since $u \in F$. Contradiction.

Proposition 4.2.15 *Let $h : \mathfrak{B} \to \mathfrak{A}$ be a homomorphism of boolean algebras. Then $F_h := h^{-1}(1^{\mathfrak{A}})$ is a filter of $\mathfrak{B}$. Moreover, for any filter F of $\mathfrak{B}$, Θ_F defined by $x \; \Theta_F \; y$ if and only if $x \leftrightarrow y \in F$ is a congruence. The factor algebra $\mathfrak{B}/\Theta_F$ is also denoted by $\mathfrak{B}/F$ and the map $x \mapsto [x]_{\Theta_F}$ by h_F.*

It follows that if $h : \mathfrak{B} \twoheadrightarrow \mathfrak{A}$ then $\mathfrak{A} \cong \mathfrak{B}/F_h$. Now we specialize $\mathfrak{A}$ to $\mathbf{2}$. Then if $h : \mathfrak{B} \to \mathbf{2}$, we have a filter $h^{-1}(1)$. It is clear that this must be an ultrafilter. Conversely, given an ultrafilter U, $\mathfrak{B}/U \cong \mathbf{2}$. We state without proof the following theorem. A set $X \subseteq B$ has the **finite intersection property** if for every finite subset $S \bigcap S \neq 0$.

Theorem 4.2.16 *For every subset of B with the finite intersection property there exists an ultrafilter containing it.*

Now put $\widehat{x} := \{h \in pt(\mathfrak{B}) : h(x) = 1\}$. It is verified that

$$\begin{array}{rcl} \widehat{-x} & = & -\widehat{x} \\ \widehat{x \cap y} & = & \widehat{x} \cap \widehat{y} \\ \widehat{x \cup y} & = & \widehat{x} \cup \widehat{y} \end{array}$$

To see the first, assume $h \in \widehat{-x}$. Then $h(-x) = 1$, from which $h(x) = 0$, and so $h \notin \widehat{x}$, that is to say $h \in -\widehat{x}$. Conversely, if $h \in -\widehat{x}$ then $h(x) \neq 1$, whence $h(-x) = 1$, showing $h \in \widehat{-x}$. Second, $h \in \widehat{x \cap y}$ implies $h(x \cap y) = 1$, so $h(x) = 1$ and $h(y) = 1$, giving $h \in \widehat{x}$ as well as $h \in \widehat{y}$. Conversely, if the latter holds then $h(x \cap y) = 1$ and so $h \in \widehat{x \cap y}$. Similarly with $\cup$.

Theorem 4.2.17 *The map $x \mapsto \widehat{x}$ is an injective homomorphism from $\mathfrak{B}$ into the algebra $\mathfrak{P}(pt(\mathfrak{B}))$. Consequently, every boolean algebra is isomorphic to a field of sets.*

Proof. It remains to see that the map is injective. To that end, let x and y be two different elements. We claim that there is an $h : \mathfrak{B} \to \mathbf{2}$ such that $h(x) \neq h(y)$. For we either have $x \not\leq y$, in which case $x \cap (-y) > 0$; or we have $y \not\leq x$, in which case $y \cap -x > 0$. Assume (without loss of generality) the first. By Theorem 4.2.16 there is an ultrafilter U containing the set $\{x \cap (-y)\}$. Obviously, $x \in U$ but $y \notin U$. Then h_U is the desired point. ☐

The original representation theorem for finite boolean algebras can be extended in the following way (this is the route that Keenan and Faltz take). A boolean algebra $\mathfrak{B}$ is called **complete** if any set has a least upper bound and a greatest lower bound.

Theorem 4.2.18 *A complete atomic boolean algebra $\mathfrak{B}$ is isomorphic to $\mathfrak{P}(\mathbf{At}(\mathfrak{B}))$.*

It should be borne in mind that within boolean semantics (say, in the spirit of Keenan and Faltz) the meaning of a particular linguistic item is a member of a boolean algebra, but it may at the same time be a function from some boolean algebra to another. For example, adjectives denotations form a boolean algebra, but they may also be seen as functions from the algebra of common noun denotations (type $e \to t$) to itself. These maps are, however, in general not homomorphisms. The meaning of a particular adjective, say `tall`, can in principle by any such function. However, some adjectives behave better than others. A host of properties of such functions can be considered.

Definition 4.2.19 *Let $\mathfrak{B}$ be a boolean algebra and $f : B \to B$. f is called* ***monotone*** *if and only if for all $x, y \in B$: if $x \leq y$ then $f(x) \leq f(y)$. f is called* ***antitone*** *if for all $x, y \in B$: if $x \leq y$ then $f(x) \geq f(y)$. f is called* ***restricting*** *if and only if for each $x \in B$ $f(x) \leq x$. f is called* ***intersecting*** *if and only for each $x \in B$: $f(x) = x \cap f(1)$.*

Adjectives that denote intersecting functions are often also called **intersective**. An example is `white`. A white car is something that is both white and a car. Hence we find that white' is intersecting. Intersecting functions are restricting but not necessarily conversely. The adjective `tall` denotes a restricting function (and is therefore also called **restricting**). A tall student is certainly a student. Yet, a tall student is not necessarily also tall. The problem is that tallness varies with the property that is in question. (We may analyze it, say, as: belongs to the 10 % of the longest students. Then it becomes clear that it has this property.) Suppose that students of sports are particularly tall. Then a tall student of sports will automatically qualify as a tall student, but a tall student may not be a tall student of sports. On the other hand, if students of sports were particularly short, then a tall student will be a tall student of sports, but the converse need not hold. The adjective `fake` is an example of an monotonic adjective that is not restricting. There are also adjectives that have none of these properties (for example, `supposed` or `alleged`). We will return to sentential modifiers in the next section.

We conclude the section with a few remarks on the connection with theories and filters. Let Ω be the signature of boolean logic: the 0–ary symbols $\top$, $\bot$, the unary $\neg$ and the binary $\vee$ and $\wedge$. Then we can define boolean algebras by means of equations, as we have done with Definition 4.2.1. For reference, we call the set of equations BEq. Or we may actually define a consequence relation, for example by means of a Hilbert–calculus. Table **??** gives

Table 4.1: The Axioms of Propositional Logic

(a0)	$p_0 \to (p_1 \to p_0)$
(a1)	$(p_0 \to (p_1 \to p_2)) \to ((p_0 \to p_1) \to (p_0 \to p_2))$
(a2)	$((p_0 \to p_1) \to p_0) \to p_0$
(a3)	$\bot \to p_0$
(a4)	$\neg p_0 \to (p_0 \to \bot)$
(a5)	$(p_0 \to \bot) \to \neg p_0$
(a6)	$\top$
(a7)	$p_0 \to (p_1 \to (p_0 \wedge p_1))$
(a8)	$(p_0 \wedge p_1) \to p_0$
(a9)	$(p_0 \wedge p_1) \to p_1$
(a10)	$p_0 \to (p_0 \vee p_1)$
(a11)	$p_1 \to (p_0 \vee p_1)$
(a12)	$((p_0 \vee p_1) \to p_2) \to ((p_0 \to p_2) \wedge (p_1 \to p_2))$

a complete set of axioms, which together with the rule MP axiomatize boolean logic. Call this calculus PC. We have to bring the equational calculus and the deductive calculus into correspondence. We have a calculus of equations (see Section 1.1), which tells us what equations follow from what other equations. Write $\varphi \leftrightarrow \chi$ in place of $(\varphi \to \chi) \wedge (\chi \to \varphi)$.

Lemma 4.2.20 *The following are equivalent.*

1. $\varphi \leftrightarrow \chi$ *is a theorem of* PC.
2. *For every boolean algebra* $\mathfrak{A}$: $\mathfrak{A} \vDash \varphi \doteq \chi$.
3. $\mathsf{BEq} \vDash \varphi \doteq \chi$.

The proof is lengthy, but routine. (2) and (3) are equivalent by the fact that an algebra is a boolean algebra if and only if it satisfies BEq. So, (1) $\Leftrightarrow$ (3) needs proof. The interested reader is referred to (Kracht, 1999) for a proof of this equivalence. We shall now show that the so defined logic is indeed the logic of the two element matrix with designated element 1.

By the deduction theorem (which holds in PC), $\varphi \leftrightarrow \chi$ if and only if $\varphi \vdash \chi$ and $\chi \vdash \varphi$. As a consequence we get that $\Delta := \{\langle\varphi, \chi\rangle : \vdash^{\mathsf{PC}} \varphi \leftrightarrow \chi\}$ is a congruence on the term algebra. What is more, it is admissible for every deductively closed set. For if Σ is deductively closed and $\varphi \in \Sigma$, then also $\chi \in \Sigma$ for every $\chi \,\Delta\, \varphi$, by Modus Ponens.

Lemma 4.2.21 *$\mathfrak{Tm}_\Omega(V)/\Delta$ is a boolean algebra. Moreover, if Σ is a deductively closed set in $\mathfrak{Tm}_\Omega(V)$ then Σ/Δ is a filter on $\mathfrak{Tm}_\Omega(V)/\Delta$. If Σ is maximally consistent, Σ/Δ is an ultrafilter. Conversely, if F is a filter on $\mathfrak{Tm}_\Omega(V)/\Delta$, the full preimage under the canonical homomorphism is a deductively closed set. If F is an ultrafilter, this set is a maximally consistent set of formulae.*

Thus, $\vdash^{\mathsf{PC}}$ is the intersection of all $\vDash_{\langle\mathfrak{A},F\rangle}$, where $\mathfrak{A}$ is a boolean algebra and F a filter. Now, instead of deductively closed sets we can also take maximal (consistent) deductively closed sets. Their image under the canonical map is an ultrafilter. However, the equivalence $x \leftrightarrow y \in U$ is a congruence, and it is admissible for U. Thus, we can once again factor it out and obtain the following completeness theorem.

Theorem 4.2.22 $\vdash^{\mathsf{PC}} = \vdash_{\langle\mathbf{2},\{1\}\rangle}$.

This says that we have indeed axiomatized the logic of the 2–valued algebra. What is more, equations can be seen as statements of equivalence and conversely. We can draw from this characterization a useful consequence. Call a propositional logic **inconsistent**, if every formula is a theorem.

Corollary 4.2.23 PC *is maximally complete. That is to say, if an axiom or rule ρ is not derivable in* PC, $\mathsf{PC} + \rho$ *is inconsistent.*

Proof. Let $\rho = \langle\Delta, \varphi\rangle$ be a rule that is not derivable in PC. Then by Theorem 4.2.22 there is a valuation β which makes every formula of Δ true but φ false. Define the following substitution: $\sigma(p) := \top$ if $\beta(p) = 1$, and $\sigma(p) := \bot$ otherwise. Then for every $\chi \in \Delta$, $\sigma(\chi) \leftrightarrow \top$, while $\sigma(\varphi) \leftrightarrow \bot$. Hence, as PC derives $\sigma(\chi)$ for every $\chi \in \Delta$, it also derives $\sigma(\varphi)$, and so $\bot$. On the other hand, in PC, every proposition follows from $\bot$. Thus, $\mathsf{PC} + \rho$ is inconsistent. ☐

Notes on this section. The earliest sources of propositional logic are the writing of the Stoa. Stoic logic was couched in terms of inference rules. The first to introduce equations and a calculus of equations was Leibniz. The characterization of $\leq$ in terms of union (or intersection) is explicitly mentioned by him. Leibniz only left incomplete notes. Later, de Morgan, Boole and Frege have completed the axiomatization of what is now known as Boolean logic.

Exercise 135. Show that $x \leq y$ if and only if $x \cap y = x$.

Exercise 136. The dual of a lattice term t^d is defined as follows.

$$\begin{array}{lll} x^d & := & x, \qquad x \text{ a variable}, \\ (t \cup t')^d & := & t^d \cap t'^d, \\ (t \cap t')^d & := & t^d \cup t'^d. \end{array}$$

Show that $t^{dd} = t$. Moreover, show that $s \doteq t$ holds in every lattice if and only if $s^d \doteq t^d$ holds in every lattice. *Hint.* For a lattice $\mathfrak{L} = \langle L, \cap, \cup \rangle$ define $\mathfrak{L}^d := \langle L, \cup, \cap \rangle$. This is a lattice. (Can you show this?) It is called the dual lattice of $\mathfrak{L}$. Then show that $s \doteq t$ holds in $\mathfrak{L}$ if and only if $s^d \doteq t^d$ holds in $\mathfrak{L}^d$.

Exercise 137. (Continuing the previous exercise.) For a boolean term define additionally $0^d := 1$, $1^d := 0$, $(-t)^d := -t^d$ and $\mathfrak{B}^d := \langle B, 1, 0, \cup, \cap, - \rangle$ for $\mathfrak{B} = \langle B, 0, 1, \cap, \cup, - \rangle$. Show that $\mathfrak{B}^d \cong \mathfrak{B}$. This implies that $s = t$ holds in $\mathfrak{B}$ if and only if $s^d = t^d$ holds in $\mathfrak{B}$.

Exercise 138. Prove Lemma 4.2.6.

Exercise 139. Let $\leq$ be a partial ordering on L. Define $x \cup y$ to be the element u such that whenever $x \leq z$ and $y \leq z$ then also

$u \leq z$. Define $x \cap y$ to be the element u such that whenever $z \leq x$ and $z \leq y$ then also $z \leq u$. Suppose that $x \cup y$ and $x \cap y$ so defined always exist. Show that $\langle L, \cap, \cup \rangle$ is a lattice.

Exercise 140. Let $\mathbb{Z}$ be the set of entire numbers. For $i, j \in \omega$ and $j < 2^i$ let $R_{i,j} := \{m \cdot 2^i + j : m \in \mathbb{Z}\}$. Let H be the set of subsets of $\mathbb{Z}$ generated by all *finite* unions of sets of the form $R_{i,j}$. Show that H forms a field of sets (hence a boolean algebra). Show that it has no atoms.

4.3 Intensionality

Leibniz' Principle has given rise to a number of problems in formal semantics. One such problem is its apparent failure with respect to intensional contexts. This is what we shall discuss now. The following context does not admit any substitution of A by B different from A without changing the truth value of the entire sentence.

(4.3.1) `The expression '`A`' is the same as the expression '`B`'.`

Obviously, if such sentences were used to decide about synonymy, no expression is synonymous with any other. However, the feeling with these types of sentences is that the expressions do not enter with their proper meaning here; one says, the expressions A and B are not **used** in (4.3.1) they are only **mentioned**. This need not cause problems for our sign based approach. We might for example say that the occurrences of A where it is used are occurrences with a different category than those where it is mentioned. If we do not assume this we must exclude those sentences in which the occurrences of A or B are only mentioned, not used. However, in that case we need a criterion for deciding when an expression is used and when it is mentioned. The picture is as follows. Let $S(x)$ be shorthand for a sentence S missing a constituent x. We call them **contexts**. Then Leibniz' Principle says that A and B have identical meaning, in symbols $A \equiv B$, if and only if $S(A) \leftrightarrow S(B)$

is true for all $S(x)$. Now, let Σ be the set of all contexts, and Π the set of all contexts where the missing expression is used, not mentioned. Then we end up with two kinds of identity:

$$\begin{array}{rcl} A \equiv_\Sigma B & :\Leftrightarrow & (\forall S(x) \in \Sigma)(S(A) \leftrightarrow S(B)) \\ A \equiv_\Pi B & :\Leftrightarrow & (\forall S(x) \in \Pi)(S(A) \leftrightarrow S(B)) \end{array}$$

Obviously, $\equiv_\Sigma \subseteq \equiv_\Pi$. Generalizing this, we get a Galois correspondence here between certain sets of contexts and equivalence relations on expressions. Contexts outside of Π are called **hyperintensional**. In our view, (4.3.1) does not contain occurrences of the language signs for A and B but only occurrences of strings. Strings denote themselves. So, what we have inserted are not the same signs as the signs of the language, and this means that Leibniz' Principle is without force in example (4.3.1) with respect to the signs. However, if put into the context `the meaning of '__'`, we get the actual meaning of A that the language gives to it. Thus, the following is once again transparent for the meanings of A and B:

(4.3.2) `The expression '`A`' has the same meaning as the expression '`B`'.`

Other hyperintensional context is

(4.3.3) `John thinks that palimpsests are kinds of leaflets.`

What John thinks here is that the expression `palimpsest` denotes a special kind of leaflet, where in fact it denotes a kind of manuscript. Although this is a less direct case of mentioning an expression, it still is the case that the sign with exponent `palimpsest` is not an occurrence of the genuine English language sign, because it is used with a different meaning. The meaning of that sign is once again the exponent (string) itself.

There are other problematic instance of Leibniz' Principle, namely so called *intensional* contexts. Consider the following sentences.

(4.3.4) `The morning star is the evening star.`
(4.3.5) `John believes that the morning star is the morning star.`
(4.3.6) `John believes that the morning star is the evening star.`
(4.3.7) `The square root of 2 is less than 3/2.`
(4.3.8) `John believes that the square root of 2 is less than 3/2.`

It is known that (4.3.4) is true. However, it is quite conceivable that (4.3.5) may be true and (4.3.6) false. By Leibniz' Principle, we must assume that `the morning star` and `the evening star` have different meaning. However, as Frege points out, in *this* world they refer to the same thing (the planet Venus), so they are not different. Frege therefore distinguishes **reference** (*Bedeutung*) from **sense** (*Sinn*). In (4.3.4) the expressions enter with their reference, and this is why the sentence is true. In (4.3.5), however, they do not enter with their reference, otherwise John holds an inconsistent belief. Rather, they enter with their senses, and the senses are different. Thus, we have seen that expressions that are used (not mentioned) in a sentence may either enter with their reference or with their sense. The question is however the same as before: how do we know when an expression enters with its sense rather than its reference? The general feeling is that one need not be worried by that question. Once the sense of an expression is given, we know what its referent is. We may think of the sense as an algorithm that gives us the reference on need. (This analogy has actually been pushed by Yannis Moschovakis, who thinks that sense actually *is* an algorithm (see (Moschovakis, 1994)). However, this requires great care in defining the notion of an algorithm, otherwise it is too fine grained to be useful. Notice that Moschovakis shows that equality of meaning is decidable, even when equality of denotation isn't.) Contexts that do not vary with the sense only with the reference of their subexpression are called **extensional**. Nonextensional contexts are **intensional**. Just how fine grained intensional contexts are is a difficult matter. For example, it is not inconceivable that (4.3.7) is true but (4.3.8) is false. Since $\sqrt{2} < 1.5$ we expect that it cannot be otherwise, and that one cannot even believe otherwise. This holds, for example, under the modal analysis of belief by Hintikka (1962). Essentially, this is what we shall assume here, too. The problem of intensionality with respect to Leibniz' Principle disappears once we realize that it speaks of identity in meaning, not just identity of denotation. These are totally different things, as Frege rightly observed. Of course, we still have to show how meaning and denotation work together, but there is no problem with Leibniz' Principle.

Intensionality has been a very important area of research in formal semantics, partly because Montague already formulated an intensional system. The influence of Carnap is clearly visible here. It will turn out that equating intensionality with normal modal operators is not always helpful. Nevertheless, the study of intensionality has helped enormously in understanding the process of algebraization.

Let $A := \{\mathtt{(}, \mathtt{)}, \mathtt{p}, \mathtt{0}, \mathtt{1}, \wedge, \neg, \Box\}$, where the boolean symbols are used as before and $\Box$ is a unary symbol, which is written before its argument. We form expressions in the usual way, using brackets. The language we obtain shall be called L_M. The abbreviations $\varphi \to \chi$ and $\varphi \leftrightarrow \chi$ as well as typical shorthands (omission of brackets) are used without warning. Notice that we have a propositional language, so that the notions of substitution, consequence relation and so on can be taken over straightforwardly from Section 4.1.

Definition 4.3.1 *A* ***modal logic*** *is a subset Θ of L_M which contains all boolean tautologies and which is closed under substitution and Modus Ponens. Θ is called* ***classical*** *if from $\varphi \leftrightarrow \chi \in \Theta$ follows that $\Box\varphi \leftrightarrow \Box\chi \in \Theta$,* ***monotone*** *if from $\varphi \to \chi \in \Theta$ follows $\Box\varphi \to \Box\chi \in \Theta$. Θ is* ***normal*** *if for all $\varphi, \chi \in L_M$ (a) $\Box(\varphi \to \chi) \to (\Box\varphi \to \Box\chi) \in \Theta$, (b) if $\varphi \in \Theta$ then $\Box\varphi \in \Theta$.*

The smallest normal modal logic is denoted by K, after Saul Kripke. A **quasi–normal** modal logic is a modal logic that contains K.

One typically defines

$$\Diamond\varphi := \neg(\Box(\neg\varphi))$$

and calls this the **dual operator**. $\Box$ is usually called a **necessity operator**, $\Diamond$ a **possibility operator**. If φ is an axiom and Θ a (normal) modal logic, then $\Theta + \varphi$ ($\Theta \oplus \varphi$) is the smallest (normal) logic containing $\Theta \cup \{\varphi\}$. Analogously the notation $\Theta + \Gamma$, $\Theta \oplus \Gamma$ for a set Γ is defined.

Definition 4.3.2 *Let Θ be a modal logic. Then $\vdash_\Theta$ is the following consequence relation. $\Delta \vdash_\Theta \varphi$ if and only if φ can be deduced from $\Delta \cup \Theta$ using (mp) only. $\Vdash_\Theta$ is the consequence relation generated by the axioms of Θ, the rule (mp) and the rule (mn): $\langle\{\mathtt{p}\}, (\Box\mathtt{p})\rangle$. $\vdash_\Theta$ is called the* ***local consequence relation****, $\Vdash_\Theta$ the* ***global consequence relation*** *associated with Θ.*

It is left to the reader to verify that this indeed defines a consequence relation. We remark here that for $\vdash_{\mathsf{K}}$ the rule (mn) is by definition admissible. However, it is not derivable (see the exercises); in $\Vdash_{\mathsf{K}}$ on the other hand it is (by definition) derivable. Before we develop the algebraic approach further, we shall restrict our attention to normal logics. For these logics, a geometric (or 'model theoretic') semantics has been given.

Definition 4.3.3 *A* ***Kripke–frame*** *is a pair $\langle F, \lhd\rangle$ where F is a set, the* ***set of worlds****, and $\lhd \subseteq F^2$, the* ***accessibility relation****. A* ***generalized Kripke–frame*** *is a triple $\langle F, \lhd, \mathbb{F}\rangle$ where $\langle F, \lhd\rangle$ is a Kripke–frame and $\mathbb{F} \subseteq \wp(F)$ a field of sets closed under the operation $\blacksquare$ on $\wp(F)$ defined as follows:*

$$\blacksquare A := \{x : \textit{ for all } y : \textit{ if } x \lhd y \textit{ then } y \in A\}\ .$$

Call a **valuation** into a general Kripke–frame $\mathfrak{F} = \langle F, \lhd, \mathbb{F}\rangle$ a function $\beta : V \to \mathbb{F}$. Then we define as follows.

$$\begin{array}{lcl}
\langle\mathfrak{F}, \beta, x\rangle \vDash p & \Leftrightarrow & x \in \beta(p) \qquad (p \in V) \\
\langle\mathfrak{F}, \beta, x\rangle \vDash (\neg\varphi) & \Leftrightarrow & \langle\mathfrak{F}, \beta, x\rangle \nvDash \varphi \\
\langle\mathfrak{F}, \beta, x\rangle \vDash (\varphi \wedge \chi) & \Leftrightarrow & \langle\mathfrak{F}, \beta, x\rangle \vDash \varphi; \chi \\
\langle\mathfrak{F}, \beta, x\rangle \vDash (\Box\varphi) & \Leftrightarrow & \text{for all } y : \text{ if } x \lhd y \text{ then } \langle\mathfrak{F}, \beta, y\rangle \vDash \varphi
\end{array}$$

Furthermore, the **local frame consequence** is defined as follows. $\Delta \vDash_{\mathfrak{F}} \varphi$ if for every β and $x \in F$: if $\langle \mathfrak{F}, \beta, x\rangle \vDash \delta$ for every $\delta \in \Delta$ then $\langle \mathfrak{F}, \beta, x\rangle \vDash \varphi$. This is a consequence relation. Moreover, the axioms and rules of PC are valid. Furthermore, $\vDash_{\mathfrak{F}} \Box(\varphi \to \chi) \to (\Box\varphi \to \Box\chi)$. For if $\langle \mathfrak{F}, \beta, x\rangle \vDash \Box(\varphi \to \chi); \Box\varphi$ and if $x \lhd y$, then $\langle \mathfrak{F}, \beta, y\rangle \vDash \varphi \to \chi; \varphi$, from which $\langle \mathfrak{F}, \beta, y\rangle \vDash \chi$. As y was arbitrary, $\langle \mathfrak{F}, \beta, x\rangle \vDash \Box\chi$. Finally, suppose that $\mathfrak{F} \vDash \varphi$. Then $\mathfrak{F} \vDash \Box\varphi$. For choose x and β. Then for all y such that $x \lhd y$: $\langle \mathfrak{F}, \beta, y\rangle \vDash \varphi$. Hence $\langle \mathfrak{F}, \beta, x\rangle \vDash \Box\varphi$. Since x and β were arbitrarily chosen, the conclusion follows. Define $\Delta \vDash^g_{\mathfrak{F}} \varphi$ if for all β: if $\overline{\beta}(\delta) = F$ for all $\delta \in \Delta$, then also $\overline{\beta}(\varphi) = F$. This is the **global frame consequence** determined by $\mathfrak{F}$. For a class of frames $\mathcal{K}$ we put

$$\vDash_{\mathcal{K}} := \bigcap \langle \vDash_{\mathfrak{F}} : \mathfrak{F} \in \mathcal{K}\rangle$$

Analogously, $\vDash^g_{\mathcal{K}}$ is the intersection of all $\vDash^g_{\mathfrak{F}}$, $\mathfrak{F} \in \mathcal{K}$.

Theorem 4.3.4 *For every class $\mathfrak{K}$ of frames there is a modal logic Θ such that $\vDash_{\mathcal{K}} = \vdash_{\Theta}$. Moreover, $\vDash^g_{\mathcal{K}} = \Vdash_{\Theta}$.*

Proof. We put $\Theta := \{\varphi : \varnothing \vDash_{\mathcal{K}} \varphi\}$. We noticed that this is a normal modal logic if $\mathcal{K}$ is one membered. It is easy to see that this therefore holds for all classes of frames. Clearly, since both $\vdash_{\Theta}$ and $\vDash_{\mathcal{K}}$ have a deduction theorem, they are equal if they have the same tautologies. This we have just shown. For the global consequence relation, notice first that $\Theta = \{\varphi : \varnothing \vDash^g_{\mathcal{K}} \varphi\}$. Moreover, $\Vdash_{\Theta}$ is the smallest global consequence relation containing $\vdash_{\Theta}$, and similarly $\vDash^g_{\mathcal{K}}$ the smallest global consequence relation containing $\vDash_{\mathcal{K}}$. $\Box$

We shall give some applications of modal logic to the semantics of natural language. The first is that of (meta)physical necessity. In uttering (4.3.9) we suggest that (4.3.10) obtains whatever the circumstances. Likewise, in uttering (4.3.11) we suggest that there are circumstances under which (4.3.12) is true.

(4.3.9) `2+3 is necessarily greater than 4.`
(4.3.10) `2+3 is greater than 4.`
(4.3.11) `Caesar might not have defeated Vercingetorix.`
(4.3.12) `Caesar has not defeated Vercingetorix.`

The analysis is as follows. We consider `necessarily` as an operator on sentences. Although it appears here in postverbal position, it may be rephrased by `it is necessary that`, which can be iterated any number of times. The same can be done with `might`, which can be rephrased as `it is possible that` and turns out to be the dual of the first. We disregard questions of form here and represent sentential operators simply as $\Box$ and $\Diamond$, prefixed to the sentence in question. $\Box$ is a modal operator, and it is normal. For example, if A and B are both necessary, then so is $A \wedge B$, and conversely. Second, if A is logically true, then A is necessary. Necessity has been modelled according to Carnap by frames of the form $\langle W, W \times W \rangle$. Metaphysically possible worlds should be possible no matter what is the case (that is, which world we are in). It turns out that the interpretation above yields a particular logic, called $\mathsf{S5}$.

$$\mathsf{S5} := \mathsf{K} \oplus \{p \to \Diamond p, \Diamond p \to \Diamond\Diamond p, p \to \Box\Diamond p\}$$

We defer a proof of the fact that this characterizes $\mathsf{S5}$.

Hintikka (1962) has axiomatized the logic of knowledge and belief. Write $[K_J]\varphi$ to represent the proposition 'John knows that φ' and $[B_J]\varphi$ to represent the proposition 'John believes that φ'. Then, according to Hintikka, both turn out to be normal modal operators. In particular, we have the following axioms.

$[B_J](\varphi \to \chi) \to ([B_J]\varphi \to [B_J]\chi)$	logical 'omniscience' for believe
$[B_J]\varphi \to [B_J][B_J]\varphi$	positive introspection
$[K_J](\varphi \to \chi) \to ([K_J]\varphi \to [K_J]\chi)$	logical omniscience
$[K_J]\varphi \to \varphi$	factuality of knowledge
$[K_J]\varphi \to [K_J][K_J]\varphi$	positive introspection
$\neg[K_J]\varphi \to [K_J]\neg[K_J]\varphi$	negative introspection

Further, if φ is a theorem, so is $[B_J]\varphi$ and $[K_J]\varphi$. Now, we may either study both operators in isolation, or put them together in one language, which now has two modal operators. We trust that the reader can make the necessary amendments to the above definitions to take care of any number of operators. We can can then also formulate properties of the operators in combination. It turns out, namely, that the following holds.

$$[K_J]\varphi \to [B_J]\varphi$$

The logic of $[B_J]$ is known as $\mathsf{K4} := \mathsf{K} \oplus \Diamond\Diamond p \to \Diamond p$ and it is the logic of all transitive Kripke–frames; $[K_J]$ is once again $\mathsf{S5}$.

A different interpretation of modal logic is in the area of *time*. Here there is no consensus on how the correct model structures look like. If one believes in determinism, one may for example think of time points as lying on the real line $\langle \mathbb{R}, < \rangle$. Introduce an operator $\boxright$ by

$$\langle \mathbb{R}, <, \beta, t\rangle \vDash \boxright\chi :\Leftrightarrow \text{ for all } t' > t : \langle \mathbb{R}, <, \beta, t'\rangle \vDash \chi$$

One may read $\boxright\chi$ as *it will always be the case that* χ. Likewise, $\diamondright\chi$ may be read as *it will at least once be the case that* χ. The logic of $\boxright$ is

$$\mathsf{Th}\,\langle \mathbb{R}, <\rangle := \{\chi : \text{ for all } \beta, x : \langle \mathbb{R}, <, \beta, x\rangle \vDash \chi\}$$

Alternatively, we may define an operator $\boxleft$ by

$$\langle \mathbb{R}, <, \beta, t\rangle \vDash \boxleft\chi :\Leftrightarrow \text{ for all } t' < t : \langle \mathbb{R}, <, \beta, t'\rangle \vDash \chi$$

to read as *it has always been the case that* χ. Finally, $\diamondleft\chi$ reads *it has been the case that* χ. On $\langle \mathbb{R}, <\rangle$, $\boxleft$ has the same logic as $\boxright$. We may also study both operators in combination. What we get is a bimodal logic (which is simply a logic over a language with two operators, each defining a modal logic in its own fragment). Furthermore, $\langle \mathbb{R}, <\rangle \vDash p \to \boxright\diamondleft p; p \to \boxleft\diamondright p$. The details need not be of much concern here. Suffice it to say that the modelling of time with the help of modal logic has received great attention in philosophy and linguistics. Obviously, to be able to give a model theory of tenses is an important task. Already Montague integrated into his theory a treatment of time in combination with necessity (as discussed above).

We shall use the theory of matrices to define a semantics for these logics. We have seen earlier that one can always choose matrices of the form $\langle \mathfrak{Tm}_\Omega(V), \Delta\rangle$, Δ deductively closed. Now assume that Θ is classical. Then put $\varphi \; \Theta \; \chi$ if and only if $\varphi \leftrightarrow \chi \in \Theta$. This is a congruence relation, and we can form the factor algebra along that congruence. (Actually, classicality is exactly the condition that $\leftrightarrow$ induces a congruence relation.) It turns out that this algebra is a boolean algebra and that $\Box$ is interpreted by a function $\blacksquare$ on that boolean algebra (and $\Diamond$ by a function $\blacklozenge$).

Definition 4.3.5 *A* ***boolean algebra with (unary) operators (BAO)*** *is an algebra $\langle A, 0, 1, \cap, \cup, -, \langle \blacksquare_i : i < \kappa\rangle\rangle$ such that $\blacksquare_i : A \to A$ for all $i < \kappa$.*

If furthermore Θ is a normal modal logic, $\blacksquare$ turns out to be a so called hemimorphism.

Definition 4.3.6 *Let $\mathfrak{B}$ be a boolean algebra and $h : B \to B$ a map. h is called a* ***hemimorphism*** *if (i) $\blacksquare 1 = 1$ and (2) for all $x, y \in B$: $\blacksquare(x \cap y) = \blacksquare(x) \cap \blacksquare(y)$. A* ***multimodal algebra*** *is an algebra $\mathfrak{M} = \langle M, 0, 1, \cap, \cup, -, \langle \blacksquare_i : i < \kappa\rangle\rangle$, where $\langle M, 0, 1, \cap, \cup, -\rangle$ is a boolean algebra and $\blacksquare_i$, $i < \kappa$, a hemimorphism on it.*

We shall remain with the case $\kappa = 1$ for reasons of simplicity. A hemimorphism is thus not a homomorphism (since it does not commute with $\cup$). The modal algebras form the semantics of modal propositional logic. We also have to look at the deductively closed sets. First, if $\varphi \; \Theta \; \chi$ then $\varphi \in \Delta$ if and only if $\chi \in \Delta$. So, we can factor Δ by Θ. It turns out that Δ, being closed under (mp), becomes a filter of the boolean quotient algebra. Thus, normal modal logics are semantically complete with respect to matrices $\langle \mathfrak{M}, F\rangle$, where $\mathfrak{M}$ is a modal algebra and F a filter. We can refine this still further to F being an ultrafilter. This is so since if $\Delta \nvdash_\Theta \varphi$ there actually is a maximally consistent set of formulae that contains Δ but not φ, and reduced by Θ this turns into an ultrafilter. Say that $\mathfrak{M} \vDash \chi$ if $\langle \mathfrak{M}, U\rangle \vDash \chi$ for all ultrafilters U

on $\mathfrak{M}$. Since x is in all ultrafilters if and only if $x = 1$, we have $\mathfrak{M} \vDash \chi$ exactly if for all homomorphisms h into $\mathfrak{M}$, $h(\chi) = 1$. (Equivalently, $\mathfrak{M} \vDash \chi$ if and only if $\langle \mathfrak{M}, \{1\} \rangle \vDash \chi$.) Notice that $\mathfrak{M} \vDash \varphi \to \chi$ if for all h: $h(\varphi) \leq h(\chi)$.

Now we shall apply the representation theory of the previous section. A boolean algebra can be represented by a field of sets, where the base set is the set of all ultrafilters (alias points) over the boolean algebra. Now take a modal algebra $\mathfrak{M}$. Underlying it we find a boolean algebra, which we can represent by a field of sets. The set of ultrafilters is denoted by $U(\mathfrak{M})$. Now, for two ultrafilters U, V put $U \lhd V$ if and only if for all $\blacksquare x \in U$ we have $x \in V$. Equivalently, $U \lhd V$ if and only if $x \in V$ implies $\blacklozenge x \in U$. We end up with a structure $\langle U(\mathfrak{M}), \lhd, \mathbb{S} \rangle$, where $\lhd$ is a binary relation over $U(\mathfrak{M})$ and $\mathbb{S} \subseteq \wp(U(\mathfrak{M}))$ a field of sets closed under the operation $A \mapsto \blacksquare A$.

A modal algebra $\mathfrak{M}$ is an $\mathsf{S5}$–algebra if it satisfies the axioms given above. Let $\mathfrak{M}$ be an $\mathsf{S5}$–algebra and U, V, W ultrafilters. Then (a) $U \lhd U$. For let $x \in U$. Then $\blacklozenge x \in U$ since $\mathfrak{M} \vDash p \to \Diamond p$. Hence, $U \lhd U$. (b) Assume $U \lhd V$ and $V \lhd W$. We show that $U \lhd W$. Pick $x \in W$. Then $\blacklozenge x \in V$ and so $\blacklozenge\blacklozenge x \in U$. Since $\mathfrak{M} \vDash \Diamond\Diamond p \to \Diamond p$, we have $\blacklozenge x \in U$. (c) Assume $U \lhd V$. We show $V \lhd U$. To this end, pick $x \in U$. Then $\blacksquare\blacklozenge x \in U$. Hence $\blacklozenge x \in V$, by definition. Hence we find that $\lhd$ is an equivalence relation on $U(\mathfrak{M})$. More exactly, we have shown the following.

Proposition 4.3.7 *Let $\mathfrak{M}$ be a modal algebra, and $\lhd \subseteq U(\mathfrak{M})^2$ be defined as above.*

1. *$\mathfrak{M} \vDash p \to \Diamond p$ if and only if $\lhd$ is reflexive.*
2. *$\mathfrak{M} \vDash \Diamond\Diamond p \to \Diamond p$ if and only if $\lhd$ is transitive.*
3. *$\mathfrak{M} \vDash p \to \Box\Diamond p$ if and only if $\lhd$ is symmetric.*

The same equivalences hold for Kripke–frames. For example, $\langle F, \lhd \rangle \vDash p \to \Diamond p$ if and only if $\lhd$ is reflexive. Therefore, $\langle U(\mathfrak{M}), \lhd \rangle$ already satisfies all the axioms of $\mathsf{S5}$. Finally, let $\langle F, \lhd \rangle$ be a Kripke–frame,

$G \subseteq F$ be a set such that $x \in G$ and $x \lhd y$ implies $y \in G$. (Such sets are called **generated**.) Then the induced frame $\langle G, \lhd \cap G^2 \rangle$ is called a **generated subframe**. A special case of a generated subset is the set $F \uparrow x$ consisting of all points that can be reached in finitely many steps from x. Write $\mathfrak{F} \uparrow x$ for the generated subframe induced by $F \uparrow x$. Then a valuation β on $\mathfrak{F}$ induces a valuation on $\mathfrak{F} \uparrow x$, which we denote also by β.

Lemma 4.3.8 *$\langle \mathfrak{F}, \beta, x \rangle \vDash \varphi$ if and only if $\langle \mathfrak{F} \uparrow x, \beta, x \rangle \vDash \varphi$. It follows that if $\mathfrak{F} \vDash \Theta$ and $\mathfrak{G}$ is a generated subframe of $\mathfrak{F}$ then also $\mathfrak{G} \vDash \Theta$.*

A special consequence is the following. Let $\mathfrak{F}_0 := \langle F_0, \lhd_0 \rangle$ and $\mathfrak{F}_1 := \langle F_1, \lhd_1 \rangle$ be Kripke–frames. Assume that F_0 and F_1 are disjoint. Then $\mathfrak{F}_0 \oplus \mathfrak{F}_1 := \langle F_0 \cup F_1, \lhd_0 \cup \lhd_1 \rangle$. Moreover, $\mathfrak{F}_0 \oplus \mathfrak{F}_1 \vDash \varphi$ if and only if $\mathfrak{F}_0 \vDash \varphi$ and $\mathfrak{F}_1 \vDash \varphi$. (More exactly, if $x \in F_0$ then $\langle \mathfrak{F}_0 \oplus \mathfrak{F}_1, \beta, x \rangle \vDash \varphi$ if and only if $\langle \mathfrak{F}_0, \beta, x \rangle \vDash \varphi$, and analogously for $x \in F_1$.) It follows that a modal logic which is determined by some Kripke–frames is already determined by some connected Kripke–frames. This shows the following.

Theorem 4.3.9 $\mathsf{S5}$ *is the logic of all Kripke–frames of the form $\langle M, M \times M \rangle$.*

Now that we have looked at intensionality we shall look at the question of individuation of meanings. In algebraic logic a considerable amount of work has been done concerning the semantics of propositional languages. Notably in (Blok and Pigozzi, 1990) Leibniz' Principle was made the starting point of a definition for algebraizability of logics. We shall exploit this work for our purposes here. We start with a propositional language of signature Ω. Recall the definition of logics, consequence relation and matrix from 4.1. The first distinction we shall look at is between a *theory*, *(world) knowledge* and a *meaning postulate*.

Definition 4.3.10 *Let $\vdash$ be a consequence relation. A $\vdash$–**theory** is a set Δ such that $\Delta^{\vdash} = \Delta$. If T is a set such that $T^{\vdash} = \Delta$, T is called an **axiomatization of** Δ.*

Theories are therefore sets of formulae, and may contain variables. For example, $\{$`p0`, `(p1` $\to$ `(`$\neg$`p01))`$\}$ is a theory. However, in virtue of the fact that variables are placeholders, it is not appropriate to say that knowledge is essentially a theory. Rather, for a theory to be knowledge it must be closed under substitution. Sets of this form shall be called *logics*.

Definition 4.3.11 *Let $\vdash$ be a structural consequence relation. A $\vdash$–**logic** is a $\vdash$–theory closed under substitution.*

Finally, meaning postulates. Here, it is appropriate not to use sets of formulae, but rather equations.

Definition 4.3.12 *Let L be a propositional language. A **meaning postulate** for L is an equation. Given a set M of meaning postulates, an equation $s \doteq t$ follows from M if $s \doteq t$ holds in all algebras satisfying M.*

Thus, the meaning postulates effectively axiomatize the variety of meaning algebras, and the consequences of a set of equations can be derived using the calculus of equations of Section 1.1. In particular, if f is an n–ary operation and $s_i \doteq t_i$ holds in the variety of meaning algebras, so does $f(\vec{s}) \doteq f(\vec{t})$, and likewise, if $s \doteq t$ holds, then $\sigma(t) \doteq \sigma(t)$ holds for any substitution σ. These are natural consequences if we assume that meaning postulates characterize identity of meaning. We shall give an instructive example.

(4.3.13) `Caesar crossed the Rubicon.`
(4.3.14) `John does not believe that Caesar crossed the Rubicon.`
(4.3.15) `Bachelors are unmarried men.`
(4.3.16) `John does not believe that bachelors are unmarried men.`

It is not part of the meanings of the words that Caesar crossed the Rubicon, so John may safely believe or disbelieve it. However, it is part of the language that bachelors are unmarried men, so not believing it means associating different meanings to the words. Thus, if (4.3.15) is true and moreover a meaning postulate, (4.3.16) cannot be false.

It is unfortunate having to distinguish postulates that take the form of a formula from those that take the form of an equation. Therefore, one has sought to reduce the equational calculus to the logical calculus and conversely. The notion of equivalential logic has been studied among other by Janusz Czelakowski and Roman Suszko. The following definition is due to Prucnal and Wroński (1974). (For a set Φ of formulae, we write $\Delta \vdash \Phi$ to say that $\Delta \vdash \varphi$ for all $\varphi \in \Phi$.)

Definition 4.3.13 *Let $\vdash$ be a consequence relation. A set $\Delta(p, q) := \{\delta_i(p, q) : i \in I\}$ is called a* ***set of equivalential terms for*** $\vdash$ *if the following holds*

(eq1) $\vdash \Delta(p, p)$
(eq2) $\Delta(p, q) \vdash \Delta(q, p)$
(eq3) $\Delta(p, q); \Delta(q, r) \vdash \Delta(p, r)$
(eq4) $\bigcup_{i<\Omega(f)} \Delta(p_i, q_i) \vdash \Delta(f(\vec{p}), f(\vec{q}))$
(eq5) $p; \Delta(p, q) \vdash q$

$\vdash$ *is called* ***equivalential*** *if it has a set of equivalential terms, and* ***finitely equivalential*** *if it has a finite set of equivalential terms. If $\Delta(p, q) = \{\delta(p, q)\}$ is a set of equivalential terms for $\vdash$ then $\delta(p, q)$ is called an* ***equivalential term*** *for $\vdash$.*

As the reader may check, $p \leftrightarrow q$ is an equivalential term for $\vdash^{\mathsf{PC}}$. If there is no equivalential term then synonymy is not definable language internally. Since (Zimmermann, 1999) requires among other that meaning postulates should be expressible in the language itself, we shall introduce a 0–ary symbol $\mathsf{1}$ and a binary symbol $\vartriangle$ such that $\vartriangle$ is an equivalential term for $\vdash$ in the expanded language. To secure that $\vartriangle$ and $\mathsf{1}$ do the job as intended, we shall stipulate that the logical and the equational calculus are intertranslatable in the following way.

$$\begin{array}{lcl} \{s_i \doteq t_i : i < n\} \vDash u \doteq v & \Leftrightarrow & \{s_i \vartriangle t_i : i < n\} \vdash u \vartriangle v \\ \{\delta_i : i < \kappa\} \vdash \varphi & \Leftrightarrow & \{\delta_i \doteq \mathsf{1} : i < \kappa\} \vDash \varphi \doteq \mathsf{1} \end{array}$$

Here, $\vDash$ denotes model theoretic consequence, or, alternatively, derivability in the equational calculus (see Section 1.1). In this way, equations are translated into sets of formulae, and the postulates above secure that this translation is faithful. However, we are still not done. In order to secure Leibniz' Principle, we need the following additional rule, called **G–rule**. (See (Pigozzi, 1991).)

$$x \vdash x \vartriangle \mathsf{1} \qquad \text{(G–rule)}$$

An equivalent condition is $x; y \vdash x \vartriangle y$. Second, we require the following.

$$x \vartriangle y \doteq \mathsf{1} \vDash x \doteq y \qquad \text{(R–rule)}$$

Then one can show that on any algebra $\mathfrak{A}$ and any two congruences Θ, Θ' on $\mathfrak{A}$, $\Theta = \Theta'$ if and only if $[\mathsf{1}]\Theta = [\mathsf{1}]\Theta'$, so every congruence is induced by a theory. Classical modal logics admit the addition of $\vartriangle$. $\mathsf{1}$ is simply $\top$. The postulates can more or less directly be verified. Notice however that for a modal logic Θ there are two choices for $\vdash$ in Definition 4.3.13: if we choose $\Vdash_\Theta$ then $p \leftrightarrow q$ is an equivalential term; if, however, we choose $\vdash_\Theta$ then $\{\Box^n(p \leftrightarrow q) : n \in \omega\}$ is a set of equivalential terms. In general no finite set can be named in the local case.

In fact, this holds for any logic which is an extension of boolean logic by any number of congruential operators. There we may conflate meaning postulates with logics. However, call a logic **Fregean** if it satisfies $(p \leftrightarrow q) \to (p \vartriangle q)$. A modal logic is Fregean if and only if it contains $p \to \Box p$. There are exactly four Fregean modal logics the least of which is $\mathsf{K} \oplus p \to \Box p$. The other three are $\mathsf{K} \oplus p \leftrightarrow \Box p$, $\mathsf{K} \oplus \Box\bot$ and $\mathsf{K} \oplus \bot$, the inconsistent logic. This follows from the following theorem.

Proposition 4.3.14 $\mathsf{K} \oplus p \to \Box p = \mathsf{K} \oplus (\Box p \leftrightarrow (p \vee \Box\bot))$.

Proof. Clearly, $\Box p \leftrightarrow (p \vee \Box\bot) \vdash_{\mathsf{K}} p \to \Box p$, so have to show that in a Fregean logic $\Box p \leftrightarrow (p \vee \Box\bot)$ is a tautology. (1) $\Box p \to (\Diamond p \vee \Box\bot)$ is a tautology of K. Furthermore, $\Diamond p \to p$ holds in a Fregean logic. Hence $\Box p \to (p \vee \Box\bot)$. (2) $\Box\bot \to \Box p$ is a theorem of K, $p \to \Box p$ holds by assumption. This shows the claim. ∎

Now, in a Fregean logic, any proposition φ is equivalent either to a nonmodal proposition or to a proposition $(\chi\wedge\Diamond\top)\vee(\chi'\wedge\Box\bot)$, where χ and χ' are nonmodal. It follows from this that the least Fregean logic has only constant extensions: by the axiom $\Box\bot$, by its negation $\Diamond\top$, or both (which yields the inconsistent logic).

Now let us return to Leibniz' Principle. Fix a theory T. Together with the algebra of formulae it forms a matrix. This matrix tells us what is true and what isn't. Notice that the members of the algebra are called truth–values in the language of matrices. In the present matrix, a sentence is true only if it is a member of T. Otherwise it is false. Thus, although we can have as many truth values as we like, sentences are simply true or false. Now apply Leibniz' Principle. It says: two propositions φ and φ' have identical meaning if and only if for every proposition χ and every variable p: $[\varphi/p]\psi \in T$ if and only if $[\varphi'/p]\chi \in T$. This leads to the following definition.

Definition 4.3.15 *Let Ω be a signature and $\mathfrak{A}$ an Ω–algebra. Then for any subset F of A put*

$$\Delta_{\mathfrak{A}}F := \{\langle a,b\rangle : \text{ for all } t \in \mathbf{Pol}_1(\mathfrak{A}) : t(a) \in F \Leftrightarrow t(b) \in F\}$$

*This is called the **Leibniz equivalence** and the map $\Delta_{\mathfrak{A}}$ the **Leibniz operator**.*

Notice that the definition uses unary polynomials, not just terms. This means in effect that we have enough constants to name all expressible meanings, not an unreasonable assumption.

Lemma 4.3.16 *$\Delta_{\mathfrak{A}}F$ is an admissible congruence on $\langle\mathfrak{A}, F\rangle$.*

Proof. It is easy to see that this is an equivalence relation. We shall show therefore (i) that it is a congruence, and (ii) that is compatible with F. Put $\Theta := \Delta_{\mathfrak{A}}F$. Suppose that $x_i \;\Theta\; y_i$ for all

$i < \Omega(f)$. Then we have

$$\begin{array}{rcl} f(x_0, x_1, \ldots, x_{\Omega(f)-1}) & \Theta & f(y_0, x_1, \ldots, x_{\Omega(f)-1}) \\ & \Theta & f(y_0, y_1, x_2, \ldots, x_{\Omega(f)-1}) \\ & & \ldots \\ & \Theta & f(y_0, y_1, \ldots, y_{\Omega(f)-2}, x_{\Omega(f)-1}) \\ & \Theta & f(y_0, y_1, \ldots, y_{\Omega(f)-2}, y_{\Omega(f)-1}) \end{array}$$

So, (i) is proved. Next, let $x \in F$ and $x \;\Theta\; y$. Then, taking $t := p$ we get $[x/p]t = x$, $[y/p]p = y$. Since $x \in F$ we have $y \in F$ as well. $\Box$

We know that the consequence relation of $\langle \mathfrak{A}, F\rangle$ is the same as the congruence relation of $\langle \mathfrak{A}/\Delta_{\mathfrak{A}}F, F/\Delta_{\mathfrak{A}}F\rangle$. So, from a semantical point of view we may simply factor out the congruence $\Delta_{\mathfrak{A}}F$. Moreover, this matrix satisfies Leibniz' Principle! So, in aiming to define meanings from the language and its logic, we must first choose a theory and then factor out the induced Leibniz equivalence. We may then take as the meaning of a proposition simply its equivalence class with respect to that relation. Yet, the equivalence depends on the theory chosen and so do therefore the meanings. If the 0–ary constant c_0 is a particular sentence (say, that Caesar crossed the Rubicon) then depending on whether this sentence is true or not we get different meanings for our language objects. However, we shall certainly not make the assumption that meaning depends on accidental truth. Therefore, we shall say the following.

Definition 4.3.17 *Let $\vdash$ be a structural consequence relation over a language of signature Ω. Then the* ***canonical Leibniz congruence*** *is defined to be*

$$\nabla_{\vdash} := \Delta_{\mathfrak{Fm}_{\Omega}(\varnothing)}\mathrm{Taut}(\vdash \cap \mathrm{Tm}_{\Omega}(\varnothing)) \ .$$

For a proposition φ free of variables, the object $[\varphi]_{\nabla_{\vdash}}$ is called the ***canonical Leibniz meaning*** *of φ.*

The reader is asked to check that $\mathbf{Pol}_1(\mathfrak{Tm}_{\Omega}(\varnothing)) = \mathbf{Clo}_1(\mathfrak{Tm}_{\Omega}(\varnothing))$, so that nothing hinged on the assumption made earlier to admit all polynomials for the definition of $\Delta_{\mathfrak{A}}$. We shall briefly comment on the fact that we deal only with constant propositions. From the standpoint of language, propositional variables have no meaning except as placeholders. To ask for the meaning of $(\mathtt{p0} \vee (\neg \mathtt{p11}))$ in the context of language as a means of communicating concrete meanings makes no sense. Moreover, all that a variable stands for is concrete meanings. Thus we end up with a single algebra of meanings, one that even satisfies Leibniz' Principle.

In certain cases the Leibniz operator actually induces an isomorphism between the lattice of deductively closed sets and the lattice of congruences on $\mathfrak{A}$. This means that different theories will actually generate different equalities and different equalities will generate different theories. For example, in boolean logic, a theory corresponds to a deductively closed set in the free algebra of propositions. Moreover, $\langle a, b\rangle \in \Delta_{\mathfrak{B}} T$ if and only if $\varphi \leftrightarrow \chi \in T$. On the left hand side we find the Leibniz congruence generated by T, on the right hand side we find T applied to a complex expression formed from φ and χ. It means in words the following (taking T to be the theory generated by $\varnothing$). Two propositions, φ and χ, have the same meaning if and only if the proposition $\varphi \leftrightarrow \chi$ is a tautology. This does not hold for modal logic; for here a theory induces a nontrivial consequence only if it is closed under the necessitation rule. (The exactness of the correspondence is guaranteed for boolean logic by the fact that it is Fregean.)

We shall now briefly address the general case of a language as a system of signs. We assume for a start a grammar, with certain modes. The grammar supplies a designated category S of sentences. We may define notions of logic, theory and so on on the level of definite structure terms, since these are unique by construction. A **theory** is now a set of definite structure terms closed under consequence. Given a thory T we define the following relation on definite structure terms: $\mathfrak{t} \mathrel{\Delta} \mathfrak{t}'$ if and only if the two are intersubstitutable in any structure term preserving definiteness, and for a structure term $\mathfrak{s}$: $[\mathfrak{t}/x]\mathfrak{s} \in T$ if and only if $[\mathfrak{t}'/x]\mathfrak{s} \in T$. Again, this proves to be a congruence on the partial algebra of definite structure terms, and the congruence relation can be factored. What we get is the algebra of natural meanings.

Notes on this section. The analysis of propositional attitudes using modal operators is absolutely general unless one assumes particular postulates. In general, most attitudes will not give rise to a normal logic, though classicality must be assumed if the name propositional attitude is to make sense at all. The reason is the general consensus that a proposition is an expression modulo the laws of PC. However, notice that this means only that if two expressions are interderivable in PC, we must have the same attitude towards them. It does not say, for example, that if χ follows from φ then if I believe φ I also believe that χ. Classical logics need not be monotone (see the exercises below). For the general theory of modal logic see (Kracht, 1999).

Exercise 141. Set up a Galois correspondence between contexts and equivalence classes of expressions. You may do this for any category α. Can you characterize those context sets that generate the same equivalence class?

Exercise 142. Show that $\vdash_\Theta$ as defined above is a consequence relation. Show that (mn) is not derivable in $\vDash_{\mathsf{K}}$. *Hint.* You have to find a formula φ such that $\varphi \nvDash_{\mathsf{K}} \Box\varphi$.

Exercise 143. Define $\Delta \vDash^g_{\mathfrak{F}} \varphi$ as follows. For every β: if $\overline{\beta}(\delta) = F$ for all $\delta \in \Delta$, then $\overline{\beta}(\varphi) = F$ as well. Likewise for a class of frames $\vDash^g_{\mathcal{K}}$ is defined. Show that $\vDash^g_{\mathcal{K}}$ is the global consequence relation associated with $\mathsf{Th}\,\mathcal{K}$.

Exercise 144. Show that the logic of knowledge axiomatized above is $\mathsf{S5}$.

Exercise 145. Let β be a valuation into the generalized Kripke–frame $\langle F, \lhd, \mathbb{F}\rangle$. Put $\mathfrak{M} := \langle \mathbb{F}, \varnothing, F, \cap, \cup, \blacksquare\rangle$. Then β has an obvious homomorphic extension $\overline{\beta} : \mathfrak{Tm}_\Omega(V) \to \mathfrak{M}$. Show that $\langle \mathfrak{F}, \beta, x\rangle \vDash \varphi$ if and only if $x \in \overline{\beta}(\varphi)$.

Exercise 146. Show that there are classical modal logics which are not monotone. *Hint.* There is a counterexample based on a two–element algebra.

Exercise 147. Prove Lemma 4.3.8.

4.4 Binding and Quantification

Quantification and binding are one of the most intricate phenomena of formal semantics. Examples of quantifiers we have seen already: the English phrases `every` and `some` are of this kind, and in predicate logic it is $\forall$ and $\exists$. Examples of binding without quantification can be found easily in mathematics. The integral

$$h(\vec{y}) := \int_0^1 f(x, \vec{y})dx$$

is a case in point. The integration operator takes a function (which may have parameters) and returns the integral with respect to its values for x over the interval $[0, 1]$. What this has in common with quantification is that the function $h(\vec{y})$ does not depend on x. Likewise, the limit $\lim_{n\to\infty} a_n$ of a convergent series $a : \omega \to \mathbb{R}$, is independent of n. (The fact that these operations are not everywhere defined shall not concern us here.) So, as with quantifiers, integration and limits take entities that depend on a variable x and return an entity that is independent of it. The easiest way to analyze this phenomenon is as follows. Given a function f that depends on x, $\lambda x.f$ is a function which is independent of x. Moreover, everything that lies encoded in f is also encoded in $\lambda x.f$. So, unlike quantification and integration, λ–abstraction does not give rise to a loss of information. This is ensured by the identity $(\lambda x.f)x = f$. Moreover, extensionality ensures that abstraction also does not add any information: the abstracted function is essentially nothing more than the graph of the function. λ–abstraction therefore is *the* mechanism of binding. Quantifiers, integrals, limits and so on just take the λ–abstract and return a value. This is exactly how we have introduced the quantifiers:

$\exists x.\varphi$ was just an abbreviation of $\Sigma(\lambda x.\varphi)$. Likewise, the integral can be decomposed into two steps: first, abstraction of a variable and then the actual integration. Notice, namely, that the choice of variable matters:

$$y/3 = \int_0^1 x^2 y dx \neq x/2 = \int_0^1 x^2 y dy$$

The notation dx actually does the same as λx: it shows us over which variable we integrate. We may define integration as follows. First, we define an operation $I : \mathbb{R}^{\mathbb{R}} \to \mathbb{R}$, which performs the integration over the interval $[0,1]$ of $f \in \mathbb{R}^{\mathbb{R}}$. Then we define

$$\int_0^1 f(x)dx := I(\lambda x.f)$$

This definition decouples the definition of the actual integration from the binding process that is involved. In general, any operator $O\langle x_i : i < n\rangle.M$ which binds the variables x_i, $i < n$, and returns a value, can be defined as

$$(\dagger) \qquad O\langle x_i : i < n\rangle.M := \widehat{O}(\lambda x_0.\lambda x_1.\ldots.\lambda x_{n-1}.M)$$

for a suitable $\widehat{O}$. In fact, since $O\vec{x}.M$ does not depend on $\vec{x}$, we can use (†) to define $\widehat{O}$. What this shows is that λ–calculus can be used as a general tool for binding. It also shows that we can to some extent get rid of explicit variables, something that is quite useful for semantics. The elimination of variables namely removes a point of arbitrariness in the representation that makes meanings nonunique. In this section, we shall introduce two different algebraic calculi. The first is the algebraic approach to predicate logic using so called *cylindric algebras*, the other an equational theory of λ–calculus, which embraces the (marginally popular) variable free approach to semantics for first order logic.

We have introduced already the syntax and semantics of first–order predicate logic. Now we are going to present an axiomatization. To this end we expand the set of axioms for propositional logic by the following axioms.

$$\begin{array}{lll}
\text{(a13)} & (\forall x)(\varphi \to \chi) \to (\forall x)\varphi \to (\forall x)\chi & \\
\text{(a14)} & (\forall x)\varphi \to [t/x]\varphi & \\
\text{(a15)} & \varphi \to (\forall x)\varphi & (x \notin \mathrm{fr}(\varphi)) \\
\text{(a16)} & (\forall x)\varphi \to \neg(\exists x)\neg\varphi & \\
\text{(a17)} & \neg(\exists x)\neg\varphi \to (\forall x)\varphi &
\end{array}$$

In virtue of (a16) und (a17) we get that $(\forall x)\varphi \leftrightarrow \neg(\exists x)\neg\varphi$ as well as $(\exists x)\varphi \leftrightarrow \neg(\forall x)\neg\varphi$ which means that one of the quantifiers can be defined from the other. If equation is a symbol of the language, we add the following axioms.

$$\begin{array}{ll}
\text{(e1)} & (\forall x)(x \doteq x) \\
\text{(e2)} & (\forall xy)(x \doteq y \to y \doteq x) \\
\text{(e3)} & (\forall xyz)(x \doteq y \wedge y \doteq z \to x \doteq z) \\
\text{(e4)} & (\forall x_0 \ldots x_{\Xi(R)-1} y)(R(x_0, \ldots, x_{\Xi(R)-1}) \leftrightarrow [y/x_i]R(x_0, \ldots, x_{\Xi(R)-1})
\end{array}$$

In (e4), we assume $i < \Xi(R)$. If the signature is finite, this is a finite set of axioms.

It was Kurt Gödel who proved that this axiom system is actually complete. We shall not present this proof, though, since it can be derived from a more powerful result shown by Leon Henkin, that the simply type theory presented by Alonzo Church is complete with respect to a semantics that generalizes first–order models in a specific way.

Theorem 4.4.1 (Completeness) *φ is a tautology of first–order predicate logic if and only if it can be derived from (a0) – (a17) using the rules (mp) and (gen).*

$$\text{(mp)} \quad \frac{\varphi, \quad \varphi \to \chi}{\chi} \qquad \text{(gen)} \quad \frac{\varphi}{(\forall x)\varphi}$$

The status of (gen) is the same as that of (mn) is modal logic. (gen) is admissible with respect to the model theoretic consequence $\vDash$ defined in Section3.8, but it is not derivable in it. To see the first, suppose that φ is a theorem and let x be a variable. Then

$(\forall x)\varphi$ is a theorem, too. However, $\varphi \vDash (\forall x)\varphi$ does not follow. Simply take a unary predicate letter P and a structure consisting of two elements, 0, 1, such that P is true of 0 but not of 1. Then with $\beta(x) := 0$ we have $\langle \mathfrak{M}, \beta \rangle \vDash P(x)$ but $\langle \mathfrak{M}, \beta \rangle \nvDash (\forall x)P(x)$. Now let $\underline{P}$ be the set of all formulae that can be obtained from (a0) – (a17) by applying (gen). Then the following holds.

Theorem 4.4.2 *φ is derivable from (a0) – (a17) using (mp) and (gen) if and only if it is derivable from $\underline{P}$ using only (mp).*

The proof proceeds by showing that applications of (gen) can be moved to the beginning of the proof. The next theorem asserts that this axiomatization is complete.

Definition 4.4.3 *Let Δ be a set of formulae of predicate logic over a signature, φ a formula over that same signature. Then $\Delta \vdash \varphi$ if and only if φ can be proved from $\Delta \cup \underline{P}$ using only (mp).*

Theorem 4.4.4 *$\Delta \vdash \varphi$ if and only if $\Delta \vDash \varphi$.*

It was once again Tarski who first noticed the analogy between modal operators and quantifiers. Consider a language L of first order logic without quantifiers. We may interpret the atomic formulae of this language as propositional atoms, and formulae made from them using the boolean connectives. Then we have a somewhat more articulate version of our propositional boolean language. We can now introduce a quantifier Q simply as a unary operator. For example, $\forall\mathtt{x0}$ is a unary operator on formulae. Given a formula φ, $(\forall\mathtt{x0})\varphi$ is again a formula. (Notice that the way we write the formulae is somewhat different, but this can easily be accounted for.) In this way we get an extended language: a language of formulae extended by a single quantifier. Moreover, the laws of $\forall\mathtt{x0}$ turn the logic exactly into a normal modal logic. The quantifier $\exists\mathtt{x0}$ then corresponds to $\Diamond$, the dual of $\Box$. Clearly, in order to reach full expressive power of predicate logic we need to add infinitely many such operators, one for each variable. The resulting algebras are called **cylindric algebras**. The principal reference is to (Henkin *et al.*, 1971).

We start with the intended models of cylindric algebras. A formula may be seen as a function from models, that is, pairs $\langle \mathfrak{M}, \beta \rangle$, to 2, where $\mathfrak{M}$ is a structure and β an assignment of values to the variables. First of all, we shall remove the dependency on the structure, which allows us to focus on the assignments. There is a general first order model for any complete (= maximal consistent) theory, in which exactly those sentences are valid that belong to the theory. Moreover, this model is countable. If a theory is not complete, however, it has completions Δ_i, $i \in I$. Let $\mathfrak{M}_i$ be the canonical structure associated with Δ_i. If $\mathfrak{A}_i$ is the cylindrical algebra associated with $\mathfrak{M}_i$ (to be defined below), the algebra associated with Δ will $\prod_{i \in I} \mathfrak{A}_i$. In this way, we may reduce the study to that of a cylindric algebra of a single structure.

Take a first order structure $\langle M, \mathfrak{I} \rangle$, where M is the universe and $\mathfrak{I}$ the interpretation function. For simplicity, we assume that there are no functions. (The reader shall see in the exercises that there is no loss of expressivity in renouncing functions.) Let $V := \{x_i : i \in \omega\}$ be the set of variables. Let $\mathfrak{V}(V; M)$ be the boolean algebra of sets of functions into M. Then for every formula φ we associate the following set of assignments:

$$[\varphi] := \{\beta : \langle \mathfrak{M}, \beta \rangle \vDash \varphi\}$$

Now, for each number i we assume an operation A_i, which is defined as follows:

$$\mathsf{A}_i(S) := \{\beta : \text{ for all } \gamma \sim_{x_i} \beta : \gamma \in S\}$$

Then $\mathsf{E}_i(S) := -\mathsf{A}_i(-S)$. (The standard notation for E_i is c_i. The letter c here is suggestive for 'cylindrification'. We have decided to stay with a more logical notation.) Furthermore, for every pair of numbers $i, j \in \omega$ we assume an element $\mathsf{d}_{i,j}$, which is defined as follows.

$$\mathsf{d}_{i,j} := \{\beta : \beta(x_i) = \beta(x_j)\}$$

It is interesting to note that with the help of these elements substitution can be defined. Namely, put

$$\mathsf{s}^i_j(x) := \begin{cases} x & \text{if } i = j, \\ \mathsf{E}_i(\mathsf{d}_{i,j} \cap x) & \text{otherwise.} \end{cases}$$

Lemma 4.4.5 *Let y be a variable distinct from x. Then $[y/x]\varphi$ is equivalent with* $(\exists x)\,.\,((y \doteq x) \wedge \varphi)$.

Thus, equality and quantification alone can define substitution. The relevance of this observation for semantics has been nicely explained in (Driesner, 2001). For example, in applications it becomes necessary to introduce constants for the relational symbols. Suppose, namely that `taller` is a binary relation symbol. Its interpretation is a binary relation on the domain. If we want to replace the structure by its associated cylindric algebra, the relation is replaced by an element of that algebra, namely

$$[\mathsf{taller}'(x_0, x_1)] := \{\beta : \langle \beta(x_0), \beta(x_1)\rangle \in \mathfrak{I}(\mathtt{taller})\}$$

However, this allows us prima facie only to assess the meaning of 'x_0 is taller than x_1'. We do not know, for example, what happens to 'x_2 is taller than x_7'. For that we need the substitution functions. Now that we have the unary substitution functions, any finitary substitution becomes definable. In this particular case,

$$[\mathsf{taller}'(x_2, x_7)] = \mathsf{s}^7_1\mathsf{s}^2_0[\mathsf{taller}'(x_0, x_1)]\ .$$

Thus, given the definability of substitutions, we need only name for an n–ary relation symbol the element $[R(x_0, x_1, \dots, x_{\Xi(R)})]$.

The advantage in using this formulation of predicate logic is that it can be axiomatized using equations. It is directly verified that the equations listed in the next definition are valid in the intended structures.

Definition 4.4.6 *A **cylindric algebra** of dimension κ, κ a cardinal number, is a structure*

$$\mathfrak{A} = \langle A, 0, 1, -, \cap, \cup, \langle \mathsf{E}_\lambda : \lambda < \kappa\rangle, \langle \mathsf{d}_{\lambda,\mu} : \lambda, \mu < \kappa\rangle\rangle$$

such that the following holds for all $x, y \in A$ and $\lambda, \mu, \nu < \kappa$:

(1) $\langle A, 0, 1, -, \cap, \cup \rangle$ *is a boolean algebra.*
(2) $\mathsf{E}_\lambda 0 = 0$.
(3) $x \cup \mathsf{E}_\lambda x = \mathsf{E}_\lambda x$.
(4) $\mathsf{E}_\lambda(x \cap \mathsf{E}_\lambda y) = \mathsf{E}_\lambda x \cap \mathsf{E}_\lambda y$.
(5) $\mathsf{E}_\lambda \mathsf{E}_\mu x = \mathsf{E}_\mu \mathsf{E}_\lambda x$.
(6) $\mathsf{d}_{\lambda,\mu} = 1$.
(7) *If* $\lambda \neq \mu, \nu$ *then* $\mathsf{d}_{\mu,\nu} = \mathsf{E}_\lambda(\mathsf{d}_{\mu,\lambda} \cap \mathsf{d}_{\lambda,\nu})$.
(8) *If* $\lambda \neq \mu$ *then* $\mathsf{E}_\lambda(\mathsf{d}_{\lambda,\mu} \cap x) \cap \mathsf{E}_\lambda(\mathsf{d}_{\lambda,\mu} \cap (-x)) = 0$.

We shall see that this definition allows to capture the effect of the axioms above, with the exception of (a15). Notice first the following. $\leftrightarrow$ is a congruence in predicate logic as well. For if $\varphi \leftrightarrow \chi$ is a tautology then so is $(\exists x_i)\varphi \leftrightarrow (\exists x_i)\chi$. Hence, we can encode the axioms of predicate logic as equations of the form $\varphi \leftrightarrow \top$. However, notice that with the exception of (a15) axioms can be directly encoded. We shall not go into the details, but notice that if for example $\varphi = (\forall x)\chi$ x occurs trivially bound. It remains to treat the rule (gen). It corresponds to the rule (mn) of modal logic. In equational logic, it is implicit anyway. For if $x \doteq y$ then $O(x) \doteq O(y)$ for any unary operator O.

Definition 4.4.7 *Let $\mathfrak{A}$ be a cylindric algebra of dimension κ, and $a \in A$. Then*

$$\Delta a := \{i : i < \kappa, \mathsf{c}_i a \neq a\}$$

is called the ***dimension of*** *a. $\mathfrak{A}$ is said to be* ***locally finite dimensional*** *if $|\Delta a| < \aleph_0$ for all $a \in A$.*

A particular example of a cylindric algebra is $\mathfrak{L}_\kappa / \equiv$, $\mathfrak{L}_\kappa$ the formulae of pure equality based on the variables $\mathtt{x}_i$, $i < \kappa$. (If additional function or relation symbols are needed, they can be added with little change to the theory.) This algebra is locally finite dimensional and is freely κ–generated.

The second approach we are going to elaborate is one which takes substitutions as basic functions. For predicate logic this has been proposed by Halmos (1956), but most people credit Quine (1960) for this idea. For an exposition see (Pigozzi and Salibra, 1995). Basically, Halmos takes substitution as primitive. This has certain advantages that will become apparent soon. Let us agree that the index set is κ, again called the **dimension**. Halmos defines operations $\mathsf{S}(\tau)$ for every function $\tau : \kappa \to \kappa$ such that there are only finitely many i such that $\tau(i) \neq i$. The theory of such functions is axiomatized independently of quantification. Now, for every finite set $I \subset \kappa$ Halmos admits an operator $\mathsf{E}(I)$, which represents quantification over each of variables x_i where $i \in I$. If $I = \varnothing$, $\mathsf{E}(I)$ is the identity, otherwise $\mathsf{E}(I)(\mathsf{E}(K)x) = \mathsf{E}(I \cup K)x$. Thus, it is immediately clear that the ordinary quantifiers $\mathsf{E}(\{i\})$ suffice to generate all the others. However, the axiomatization is somewhat easier with the polyadic quantifiers. Another problem, noted in (Sain and Thompson, 1991), is the fact that the axioms for polyadic algebras cannot be schematized using letters for elements of the index set. However, they also note that the addition of transpositions is actually enough to generate the same functions. To see this, here are some definitions.

Definition 4.4.8 *Let I be a set and $\pi : I \to I$. The **support** of π, supp(π), is the set $\{i : \pi(i) \neq i\}$. A function of finite support is called a **transformation**. π is called a **permutation of** I if it is bijective. If the support contains exactly two elements, π is called a **transposition**.*

The functions whose support has at most two elements are of special interest. Notice first the case when *supp*(π) has exactly one element. In that case, π is called an **elementary substitution**. Then there are $i, j \in I$ such that $\pi(i) = j$ and $\pi(k) = k$ if $k \neq i$. If i and j are in I, then denote by (i, j) the permutation that sends i to j and j to i. Denote by $[i, j]$ the elementary substitution that sends i to j.

Proposition 4.4.9 *Let I be a set. The set $\Phi(I)$ of functions $\pi : I \to I$ of finite support are closed under concatenation. More-*

over, $\Phi(I)$ is generated by the elementary substitutions and the transpositions.

Proof. (Sketch) Look at the set $supp(\pi)$. We distinguish two sets. $i \in W$ if there is no $k > 1$ such that $\pi^k(i) = i$, and C is there is such a k. The elements of C are called **cyclic**. Clearly, $supp(\pi) = C \cup W$. We can decompose $\pi = \sigma \circ \zeta$ such that ζ is a permutation and σ has no cyclic elements. So, we may solve the problem for permutations separately. Assume that $S = \{i_j : j < p\}$ is a set such that $\pi(i_j) = i_{j+1}$, $i < p-1$, $\pi(i_{p-1}) = i_0$. Then $S \subseteq C$. Such a set is called a π–**cycle**. C is the disjoint sum of its π–cycles. Moreover, π is the decomposition of the restrictions of π to its cycles. Look at the composition $(i_1\ i_2) \circ (i_2\ i_3) \ldots \circ (i_{p-1}\ i_0)$. Its support is S and its action on S is the same as π. This closes the case of a permutation of finite nonzero support. Now we come to σ. Call a σ–**chain** a maximal set of the kind $\{i_j : j < p\}$ such that $\sigma(i_j) = i_{j+1}$, $i < p-1$, and $\sigma(i_{p-1}) = i_{p-1}$. σ is a decomposition of the maps on its chains, so we may assume now that σ has a single chain. Consider the map $[i_0\ i_1] \circ [i_1\ i_2] \circ \ldots \circ [i_{p-2}\ i_{p-1}]$. This is exactly σ, as is easily checked. $\Box$

This shows that it is enough if we take only elements corresponding to $[i, j]$ and (i, j). The elements of the first kind are already known: these are s_i^j. However, for the functions of the second kind, introduce a new symbol, $\mathsf{p}_{i,j}$. Sain and Thompson effectively axiomatize cylindric algebras that have these additional operations. They call them **finitary polyadic algebras**. Notice also the following useful fact, which we leave as an exercise.

Proposition 4.4.10 *Let $\pi : I \to I$ be an arbitrary function, and $M \subseteq I$ finite. Then there is a product γ of elementary substitutions such that $\gamma \restriction M = \pi \restriction M$.*

This theorem is both stronger and weaker than the previous one. It is stronger because it does not assume π to have finite support. On the other hand, γ only approximates π on a given finite set. (The reader may take notice of the fact that there is no sequence of elementary substitutions that equals the transformation (0 1) on ω. However, we can approximate it on any finite subset.)

Rather than developing this in detail for predicate logic we shall do it for λ–calculus, as the latter is more rich and allows to encode arbitrarily complex abstraction. Recall that $V := \{\mathsf{x}_i : i \in \kappa\}$ is the set of variables. Now, for each elementary substitution $[i, j]$ we assume a unary symbol s^i_j, and for every transposition (i, j) we assume a symbol $\mathsf{p}_{i,j}$. Evidently, the signature becomes infinite as soon as the index set is. For every $i \in \kappa$ we take a unary symbol L_i. Further, there is a binary symbol $\bullet$. Finally, we assume an infinite set of variables x_i, $i \in \omega$, over terms. If we need special function symbols (or special constants, for example quantifiers or numerical operations) these can be added. We use f as a metavariable for such a symbol. The basic calculus has the following equations. (The equatiosn are from (Pigozzi and Salibra, 1995) using the reduced signature of (Sain and Thompson, 1991)).

$$\begin{array}{lcll}
\mathsf{p}_{i,i}x & \doteq & x & \\
\mathsf{p}_{i,j}x & \doteq & \mathsf{p}_{j,i}x & \\
\mathsf{p}_{i,j}\mathsf{p}_{i,k}x & \doteq & \mathsf{p}_{j,k}\mathsf{p}_{i,j}x & \text{if } |\{i, j, k\}| = 3 \\
\mathsf{p}_{i,j}\mathsf{s}^j_i x & \doteq & \mathsf{s}^i_j x & \\
\mathsf{s}^i_j f(x_0, \dots, x_{\Omega(f)-1}) & \doteq & f(\mathsf{s}^i_j x_0, \dots, \mathsf{s}^i_j x_{\Omega(f)-1}) & \\
\mathsf{p}_{i,j} f(x_0, \dots, x_{\Omega(f)-1}) & \doteq & f(\mathsf{p}_{i,j} x_0, \dots, \mathsf{p}_{i,j} x_{\Omega(f)-1}) & \\
\mathsf{s}^i_j \mathsf{L}_k x & \doteq & \mathsf{L}_k \mathsf{s}^i_j x & \text{if } |\{i, j, k\}| = 3 \\
\mathsf{s}^j_i \mathsf{L}_i x & \doteq & \mathsf{L}_i x & \\
\mathsf{p}_{i,j} \mathsf{L}_k x & \doteq & \mathsf{L}_k \mathsf{p}_{i,j} x & \text{if } |\{i, j, k\}| = 3 \\
\mathsf{p}^j_i \mathsf{L}_j x & \doteq & \mathsf{L}_i \mathsf{p}^j_i x &
\end{array}$$

These equations characterize the pure binding by the unary operators L_i. The set of equations is invariant under permutation of the indices. Moreover, we can derive the invariance under replacement of bound variables, for example. Thus, effectively, once the interpretation of L_0 is known, the interpretation of all L_i, $i \in \kappa$, is known as well. What the equations do not do, however, is fix the interpretation of the L_i to function abstraction. To do that, we must reintroduce the variables again. Thus, we adjoin a set

$\{\mathsf{x}_i : i \in \kappa\}$ of 0–ary function symbols. Now, the following equations are added.

$$\mathsf{s}^i_j \mathsf{x}_k \doteq \begin{cases} \mathsf{x}_i & \text{if } k \in \{i, j\}, \\ \mathsf{x}_k & \text{otherwise.} \end{cases}$$

$$\mathsf{p}^i_j \mathsf{x}_k \doteq \begin{cases} \mathsf{x}_i & \text{if } k = j, \\ \mathsf{x}_j & \text{if } k = i, \\ \mathsf{x}_k & \text{otherwise.} \end{cases}$$

$$\begin{array}{lcl}
\mathsf{L}_i(y \bullet \mathsf{x}_i) & \doteq & y \\
\mathsf{L}_i(f(x_0, \ldots, x_{\Omega(f)-1})) \bullet y & \doteq & f((\mathsf{L}_i x_0) \bullet y, \ldots, (\mathsf{L}_i x_{\Omega(f)-1}) \bullet y) \\
(\mathsf{L}_i(x \bullet y)) \bullet z & \doteq & ((\mathsf{L}_i x) \bullet z) \bullet ((\mathsf{L}_i y) \bullet z) \\
(\mathsf{L}_i \mathsf{L}_i x) \bullet y & \doteq & \mathsf{L}_i x \\
(\mathsf{L}_i \mathsf{x}_i) \bullet y & \doteq & y \\
(\mathsf{L}_i \mathsf{x}_j) \bullet y & \doteq & \mathsf{x}_j, \qquad i \neq j.
\end{array}$$

We shall provide a general model construction. Start with a signature $\langle F, \Omega \rangle$. Let Eq_Ω be the equational theory of the functions from the signature alone. Now let $\mathfrak{A} = \langle A, \bullet, \{f^{\mathfrak{A}} : f \in F\}\rangle$ be a model of $\mathsf{Eq}_\Omega + \mathsf{CL} + \mathbf{A}_{\beta\eta}$. We write

$$V := A^\kappa$$

This is the set of *valuations*, members of which are as usual denoted by β, γ. Notice that each transformation τ naturally induces a map $\widehat{\tau} : V \to V : \beta \mapsto \beta \circ \tau$. Further,

$$\begin{array}{rcl}
\widehat{\tau \circ \sigma}(\beta) & = & (\tau \circ \sigma) \circ \beta \\
& = & \tau(\sigma(\beta)) \\
& = & \widehat{\tau}(\widehat{\sigma}(\beta)) \\
& = & \widehat{\tau} \circ \widehat{\sigma}(\vec{x})
\end{array}$$

The model is constructed on the set A^V. Members of this set are denoted by Gothic letters: $\mathfrak{x}$, $\mathfrak{y}$. The model will be

$$\begin{array}{l}
\langle A^V, \bullet, \langle [f] : f \in F\rangle, \langle [\mathsf{x}_i] : i \in \kappa\rangle, \langle [\mathsf{L}_i] : I \in \kappa\rangle, \\
\qquad \langle [\mathsf{s}^i_j] : i, j \in \kappa\rangle, \langle [\mathsf{p}_{i,j}] : i, j \in \kappa\rangle\rangle
\end{array}$$

We shall define these elements and functions one by one. First, $\bullet$ is interpreted as follows.

$$\bullet : A^V \times A^V \to A^V : \langle \mathfrak{x}, \mathfrak{y} \rangle \mapsto \{\langle \beta, \mathfrak{x}(\beta) \bullet \mathfrak{y}(\beta)\rangle : \beta \in V\}$$

$[\mathsf{x}_i] : V \to A : \beta \mapsto \beta(i)$ is simply the projection. For a function symbol f,

$$[f] : \left(\prod_{i<\Omega(f)} A^V\right) \to A^V : \langle \mathfrak{x}_i : i < \Omega(f)\rangle \mapsto f^{\mathfrak{A}} \circ \langle \mathfrak{x}_i : i < \Omega(f)\rangle$$

By induction it is shown that each term built from the x_i and the function symbols is interpreted by a function from V to A. For the substitution maps we do the following.

$$\begin{aligned} [\mathsf{s}^i_j] &: A^V \to A^V : f \mapsto f \circ \widehat{[j, i]} \\ [\mathsf{p}_{i,j}] &: A^V \to A^V : f \mapsto f \circ \widehat{(i, j)} \end{aligned}$$

Finally, we define abstraction.

$$[\mathsf{L}_i] : A^V \to A^V : \mathfrak{x} \mapsto \{\langle \beta, \{\langle y, \mathfrak{x}([y/\beta(i)]\beta)\rangle : y \in A\}\rangle : \beta \in V\}$$

It takes some time to digest this definition. Basically, $g(\beta) := \{\langle y, f([y/\beta(i)]\beta)\rangle : y \in A\}$ is a function from A to A with parameter $\beta \in V$. Hence it is a member of A, by functional completeness of the latter. It assigns to y the value of $\mathfrak{x}$ on β, where now $\beta(i)$ is replaced by y. This is the abstraction from y. Finally, for each $\mathfrak{x} \in A^V$, $[\mathsf{L}_i]\mathfrak{x}$ assigns to $\beta \in V$ the value $g(\beta)$. (Notice that the role of abstraction is now taken over by the set formation operator $\{x : \quad \}$.)

Notes on this section. The theory of cylindric algebras has given rise to a number of difficult problems. First of all, the axioms shown above to not fully characterize the cylindric algebras that are representable, that is to say, have as their domain U^κ, κ the dimension, and where relation variables range over n–ary relations over U. Thus, although this kind of cylindric algebra was the motivating example, the equations do not fully characterize it. As Donald Monk (1969) has shown, there is no finite set of schemes (equations using variables for members of set of variable indices) axiomatizing the class of representable cylindric algebras of dimension κ if $\kappa \geq \aleph_0$; moreover, for finite κ, the class of representable algebras is not finitely axiomatizable. J. S. Johnson has shown in (Johnson, 1969) an analogue of the second result for polyadic algebras, Sain and Thompson an analogue of the first.

The model construction for the model of the λ–calculus is called a syntactical model in (Barendregt, 1985) and is said to be due to Hindley and Longo from (Hindley and Longo, 1980). A different construction of so called *functional environment models* is applied in (Meyer, 1982)

Exercise 148. For f an n–ary function symbol let R_f be an $n+1$–ary relation symbol. Define a translation from terms to formulae as follows. First, let for a term t x_t be a variable such that $x_t \neq x_s$ whenever $s \neq t$.

$$\begin{array}{lcl} (x_i)^\dagger & := & x_i \doteq x_i \\ f(t_0, \dots, t_{n-1})^\dagger & := & R(x_{t_0}, \dots, x_{t_{n-1}}), x_{f(\vec{t})} \wedge \bigwedge_{i<n} t_i^\dagger \end{array}$$

Finally, extend this to formulae as follows.

$$\begin{array}{lcl} R(t_0, \dots, t_{n-1})^\dagger & := & R(x_{t_0}, \dots, x_{t_{n-1}}) \wedge \bigwedge_{i<n} t_i^\dagger \\ (\neg\varphi)^\dagger & := & (\neg\varphi^\dagger) \\ (\varphi \wedge \chi)^\dagger & := & (\varphi^\dagger \wedge \chi^\dagger) \\ ((\exists x)\varphi)^\dagger & := & (\exists x)\varphi^\dagger \end{array}$$

Now, let $\mathfrak{M} = \langle M, \Pi, \mathfrak{I}\rangle$ be a signature. We replace the function symbols by relation symbols, and let $\mathfrak{I}^+$ be the extension of $\mathfrak{I}$ such that

$$\mathfrak{I}^+(R_f) = \{\langle \vec{x}, y\rangle \in M^{\Xi(f)+1} : \Pi(f)(\vec{x}) = y\} .$$

Then put $\mathfrak{M}^\dagger := \langle M, \mathfrak{I}^+\rangle$. Show that $\langle \mathfrak{M}, \beta\rangle \vDash \varphi$ in and only if $\langle \mathfrak{M}^+, \beta\rangle \vDash \varphi^\dagger$.

Exercise 149. Show that if $\mathfrak{A}$ is a cylindric algebra of dimension

κ, every E_λ, $\lambda < \kappa$, satisfies the axioms of $\mathsf{S5}$. Moreover, show that if $\mathfrak{A} \vDash \varphi$ then $\mathfrak{A} \vDash \neg\mathsf{E}_\lambda\neg\varphi$.

Exercise 150. Prove Lemma 4.4.5.

Exercise 151. Show Proposition 4.4.10.

Exercise 152. Show that $\mathfrak{L}_\kappa/\equiv$ is a cylindric algebra of dimension κ and that it is locally finite dimensional.

4.5 Montague Semantics II

In this section we shall investigate the formal power of Montague Semantics and the **AB**–calculus. Let us return back to the definition of compositionality. Recall that we have defined a progression from languages as sets of strings to interpreted languages to systems of signs. Now we shall define for each of them what it means that it is context free. In Section 3.1 we have defined an *interpreted language* as a subset $\mathfrak{I}$ of $A^* \times M$ where M is the set of (possible) (sentence) meanings. The corresponding string language is $S(\mathfrak{I})$. A linear system of signs is a subset of $A^* \times T \times M$, where T is a set of categories.

Definition 4.5.1 *A linear sign grammar is* ***context free*** *if (a) the set of categories is finite, (b) if F is a mode of of arity > 0 then $F^\varepsilon := \prod_{i<n} x_i$, (c) for F of arity n, $F^\mu(M_0, \dots, M_{n-1})$ is defined if there exist derivable signs $\sigma_i = \langle E_i, T_i, M_i\rangle$, $i < n$, such that $F^\tau(T_0, \dots, T_{n-1})$ is defined and (d) if $F \neq G$ then $F^\tau \neq G^\tau$. A linear system of signs is* ***context free*** *if it is generated by a context free linear sign grammar.*

This definition is somewhat involved. (a) says that if F is an n–ary mode, F^τ can be represented by a list of n–ary immediate dominance rules. It conjunction with (b) we get that we have a finite list of context free rules. Condition (c) says that the semantics does not add any complexity to this by introducing partiality. Finally, (d) ensures that the rules of the context free grammar uniquely define the modes. (For we could in principle have two modes which reduce to the same phrase structure rule.) The reader may verify the following simple fact.

Proposition 4.5.2 *Suppose that Σ is a context free linear system of signs. Then the string language of Σ is context free.*

Definition 4.5.3 *An interpreted language $\mathfrak{I}$ is **strongly context free** if there is a context free linear system of signs Σ and a category $\mathtt{S}$ such that $\mathtt{S}(\Sigma) = \mathfrak{I}$.*

For example, let L be the set of declarative sentences of English. M is arbitrary. Declarative sentences have meanings, for example functions from contexts into truth values, or simply elements of $\{0, 1\}$. Next, we shall also specify what it means for a system of signs to be context free.

Obviously, a linear context free system of signs defines a strongly context free interpreted language. The converse does not hold, however. A counterexample is provided by the following grammar, which generates simple equality statements.

$$\begin{array}{lcl} E & \to & C\,\mathtt{=}\,C \\ C & \to & D \mid D\,\mathtt{+}\,D \\ D & \to & \mathtt{1} \mid \mathtt{2} \end{array}$$

Expressions of category E are called equations, and they have as their meaning either T or F. Now, assign the following meanings to the strings. `1` has as its D– and C–meaning the number 1, `2` the number 2, and `1+1` as its C–meaning the number 2. Now, the E–meanings are as follows.

$$\begin{array}{lcllcl} [\mathtt{2=2}]^E & = & \{T\}, & [\mathtt{1+1=2}]^E & = & \{F\}, \\ [\mathtt{1=2}]^E & = & \{F\}, & [\mathtt{1=1+1}]^E & = & \{T\}, \\ [\mathtt{2=1}]^E & = & \{F\}, & [\mathtt{1+1=1}]^E & = & \{T\}, \\ [\mathtt{1=1}]^E & = & \{T\}, & [\mathtt{2=1+1}]^E & = & \{F\}, \\ & & & [\mathtt{1+1=1+1}]^E & = & \{T\}. \end{array}$$

This grammar is unambiguous; and every string of category X has exactly one X–meaning for $X \in \{C, D, E\}$. Yet, there is no context free grammar of signs for this language. This is so, since the string `1+1` has the same T–meaning as `2`, while substituting one for the other in an equation changes the truth value.

Theorem 4.5.4 below guarantees that under the condition that the sentence meanings are unique the terms 'weakly context free' and 'strongly context free' coincide for an interpreted language. To motivate the proof, we begin with an example. Let $A := \{\mathtt{Paul}, \mathtt{Marcus}, \mathtt{sees}\}$. Let

$$\begin{aligned} L := \ & \{\mathtt{Paul\ sees\ Paul}, \mathtt{Paul\ sees\ Marcus}, \\ & \mathtt{Marcus\ sees\ Paul}, \mathtt{Marcus\ sees\ Marcus}\} \end{aligned}$$

We associate the following truth values to the sentences.

$$\begin{aligned} \mathfrak{I} = \ & \{\langle \mathtt{Paul\ sees\ Paul}, 0\rangle \\ & \langle \mathtt{Paul\ sees\ Marcus}, 1\rangle, \\ & \langle \mathtt{Marcus\ sees\ Paul}, 0\rangle, \\ & \langle \mathtt{Marcus\ sees\ Marcus}, 1\rangle\} \end{aligned}$$

Furthermore, we fix a context free grammar that generates L:

$$\begin{array}{llllllll} 0 & \mathtt{S} & \to & \mathtt{NP\ VP}, & 1 & \mathtt{VP} & \to & \mathtt{V\ NP}, \\ 2 & \mathtt{NP} & \to & \mathtt{Paul}, & 3 & \mathtt{V} & \to & \mathtt{sees}, \\ 4 & \mathtt{NP} & \to & \mathtt{Marcus}. & & & & \end{array}$$

We construct a context free system of signs Σ with $\mathtt{S}(\Sigma) = \mathfrak{I}$. To this end we associate to the terminal elements some constants.

$$\begin{aligned} \mathtt{N}_2 &:= \langle \mathtt{Paul}, \mathtt{NP}, t_P\rangle, \\ \mathtt{N}_4 &:= \langle \mathtt{Marcus}, \mathtt{NP}, t_M\rangle, \\ \mathtt{N}_3 &:= \langle \mathtt{sees}, \mathtt{V}, t_S\rangle. \end{aligned}$$

For every rule ρ of arity $n > 0$ we add a function symbol t_ρ of arity n. t_0 and t_1 are both binary. One finds out easily that there is a biunique correspondence between parse trees and terms. Now we define the following semantic functions.

$$\mathtt{F}_0' \quad : \quad \langle s, s' \rangle \mapsto t_0(s, s') \qquad \mathtt{F}_1 \quad : \quad \langle s, s' \rangle \mapsto t_1(s, s')$$

To this end we define signs, one for every rule. In this way we associate for every structure term a term as its meaning.

$$\begin{aligned} M' := \quad & \{t_M, t_P, t_S, t_1(t_S, t_M), t_1(t_S, t_M), \\ & t_0(t_M, t_1(t_S, t_M)), t_0(t_M, t_1(t_S, t_P)), \\ & t_0(t_P, t_1(t_S, t_M)), t_0(t_P, t_1(t_S, t_P))\} \end{aligned}$$

Finally, we define F_0. The meaning of $\mathtt{S}$ is fixed and so we have to satisfy the following equations.

$$\begin{aligned} t_0(t_M, t_1(t_S, t_M)) &= 1, \\ t_0(t_M, t_1(t_S, t_P)) &= 0, \\ t_0(t_P, t_1(t_S, t_M)) &= 1, \\ t_0(t_P, t_1(t_S, t_P)) &= 0. \end{aligned}$$

Hence we substitute the right hand sides for the corresponding left hand sides in M' and F_0' and we get a ready made context free system of signs.

$$M := \{t_M, t_P, t_S, t_1(t_S, t_M), t_1(t_S, t_M), 0, 1\}$$

$$\mathtt{F}_0 : \left\{ \begin{array}{l} \langle t_M, t_1(t_S, t_M) \rangle \mapsto 1, \\ \langle t_M, t_1(t_S, t_P) \rangle \mapsto 0, \\ \langle t_P, t_1(t_S, t_M) \rangle \mapsto 1, \\ \langle t_P, t_1(t_S, t_P) \rangle \mapsto 0. \end{array} \right\}$$

For every rule $\rho : B \to A_0 \ldots A_{n-1}$ with $n > 0$ we put

$$T_\rho(\gamma_0, \ldots, \gamma_n) := \begin{cases} B & \text{if for all } i < n : \gamma_i = A_i, \\ \star & \text{otherwise.} \end{cases}$$

The functions on the exponents are finally concatenation (with a blank in between). If we want to do our job perfectly, we shall define C_0 to be the concatenation with a blank in between and a final period after the second argument.

$$\begin{aligned} \mathtt{Z}_1 &:= \langle C_1, T_1, F_1 \rangle, \\ \mathtt{Z}_0 &:= \langle C_0, T_0, F_0 \rangle. \end{aligned}$$

Together with the nullary signs $\mathtt{N}_2$, $\mathtt{N}_3$ and $\mathtt{N}_4$ this generates the language.

Theorem 4.5.4 *Let $\mathfrak{I}$ be a countable interpreted language.*

1. *If the string language of $\mathfrak{I}$ is context free then $\mathfrak{I}$ is strongly context free.*

2. *For every context free grammar G for $S(\mathfrak{I})$ there exists a context free system of signs Σ and a category $\mathtt{S}$ such that*

   (a) $\mathtt{S}(\Sigma) = \mathfrak{I}$,

   (b) for every nonterminal symbol A $\{\vec{x} :$ for some $m \in M : \langle \vec{x}, A, m\rangle \in \Sigma\} = \{\vec{x} : A \vdash_G \vec{x}\}$.

Proof. Evidently, it is enough to show the second claim. Let $G := \langle \mathtt{S}, N, A, R\rangle$ be given. First, we shall change G slightly so that any sentence has countably many parse trees. Namely, suppose that $S \neq \mathtt{S}$ is the start symbol of G. Then we add the rules $\mathtt{S} \to S$ and $S \to S$. Put $T' := N \cup \{\mathtt{S}\}$, $A' := A$. We start by defining a signature. For every $n \in \omega$ and every n–ary rule ρ we choose an n–ary term symbol t_ρ. This defines the signature $\langle F, \Omega\rangle$. Now let $M' := \mathrm{Tm}_\Omega \cup M$. For any given term we assign a string by induction. If the rule is 0–ary, say $\rho : a \to .$, with $a \in A$, then let $Z(t_\rho) := a$. For other ρ:

$$Z(t_\rho(t_0, \ldots, t_{n-1})) := Z(t_0) \cdot \ldots \cdot Z(t_{n-1}) \ .$$

Next we define the main symbol, $H(t)$, of a term as follows. If $t = t_\rho(\vec{s})$ where $\rho : B \to A_0 \ldots A_{n-1}$ then we put $H(t) := B$. Let

$$\begin{aligned} \Sigma := & \quad \{\langle \vec{x}, \mathtt{S}, m\rangle : \langle \vec{x}, m\rangle \in \mathfrak{I}\} \\ & \cup \ \ \{\langle \vec{x}, A, t\rangle : H(t) = A, A \vdash_G \vec{x}\}. \end{aligned}$$

If $\rho : B \to A_0 \dots A_{n-1}$ is an n–ary B–production, $B \neq \mathtt{S}$, then let

$$\mathtt{F}_\rho(t_0, \dots, t_{n-1}) := \begin{cases} t_\rho(t_0, \dots, t_{n-1}) & \text{if for all } i < n\text{: } H(t_i) = A_i, \\ \star & \text{otherwise.} \end{cases}$$

Finally, let $\vec{x}$ be a string and $M_{\vec{x}} := \{m : \langle \vec{x}, m\rangle \in \mathfrak{I}\}$. This set is countable. Likewise, the set $T_{\vec{x}} := \{t : Z(t) = \vec{x}\}$. Choose a surjection $\sigma_{\vec{x}} : T_{\vec{x}} \to M_{\vec{x}}$. Let $\sigma := \bigcup\langle \sigma_{\vec{x}} : \vec{x} \in S(\mathfrak{I})\rangle$. Then let

$$F_{\mathsf{S}}(t_0) := \sigma(m) .$$

This is uniquely determined by assumption. □

Notice that we have made use of unary rules. This is important. For if the number of meanings for a string grows more than exponentially with the length of the string, this proof is impossible without them. For a string can then only have an exponential number of structures. We shall actually show that this is enough. To see that we need an auxiliary result.

Lemma 4.5.5 *Let L be a context free language and $d > 0$. Then there is a context free grammar G and such that for all $\vec{x} \in L$ the set of nonisomorphic G–trees for $\vec{x}$ has at least $d^{|\vec{x}|}$ members.*

Proof. Notice that it is sufficient that the result be proved for almost all $\vec{x}$. For finitely many words we can provide as many analyses as we wish. First of all, there is a grammar in Chomsky normal form that generates L. Take two rules that can be used in succession.

$$A \quad \to \quad BC, \qquad C \quad \to \quad DE\ .$$

Add the rules

$$A \quad \to \quad XE, \qquad X \quad \to \quad AD\ .$$

Then the string ABC has two analyses: $[A\ [B\ C]]$ and $[[A\ B]\ C]$. Likewise, if we have a pair of rules

$$A \quad \to \quad XE, \qquad X \quad \to \quad AB\ .$$

This grammar assigns exponentially many parses to a given string. To see this notice that any given string $\vec{x}$ of length n contains n distinct constituents. For $n \leq 3$, we use the 'almost all' clause. Now, let $n > 3$. Then $\vec{x}$ has a decomposition $\vec{x} = \vec{y}_0\vec{y}_1\vec{y}_2$ into constituents. By inductive hypothesis, for $\vec{y}_i$ we have $d^{|\vec{y}_i|}$ many analyses. Thus $\vec{x}$ has at least $2d^{|\vec{y}_0|}d^{|\vec{y}_1|}d^{|\vec{y}_2|} = 2d^{|\vec{x}|}$ analyses. ☐

There is an even simpler proof of this theorem. For each symbol $a \in A$ and terminal rule $X \to a$ we assume d copies $X_k \to a$, $k < d$, and replace X in all other rules by X_k, $k < d$.

Theorem 4.5.6 *Let $\mathfrak{I}$ be a countable interpreted language. Then $\mathfrak{I}$ is strongly context free if and only if it is weakly context free and there is some constant $c > 0$ such that for almost all strings of A^*: the number of meanings of $\vec{x}$ is bounded by $c^{|\vec{x}|}$.*

Proof. From left to right is obvious. Suppose now that $\mathfrak{I}$ is weakly context free and there exists c. By Lemma 4.5.5 we can see to it that a given $\vec{x} \in L$ has at least $c^{|\vec{x}|}$ different terms associated with it. Each term we can assign one of the meanings of $\vec{x}$, and we can see to it that this mapping is surjective. ☐

$\mathfrak{I}$ is called **unambiguous** if from $\langle \vec{x}, m \rangle \in \mathfrak{I}$ and $\langle \vec{x}, m' \rangle \in \mathfrak{I}$ follows $m = m'$. Obviously, Theorem 4.5.6 applies to unambiguous languages. Let Σ be a system of signs, $\vec{x} \in A^*$ and $A \in T$. We write $[\vec{x}]^A_\Sigma := \{m : \langle \vec{x}, A, m \rangle \in \Sigma\}$ and call this set the set of A–**meanings of $\vec{x}$ in** Σ. If Σ is given by the context, we omit it. If a system of signs is context free then it satisfies the following equations for every A and every $\vec{x}$.

$$(\dagger) \qquad \bigcup_{Z \in F} Z^\mu[[\vec{x}_0]^{B_0} \times \ldots \times [\vec{x}_{\Omega(Z)-1}]^{B_{\Omega(Z)-1}}] = [\vec{x}_0 \cdot \ldots \cdot \vec{x}_{\Omega(Z)-1}]^A ,$$

where $Z^\tau(B_0, \ldots, B_{n-1}) = A$. This means simply put that the A–meanings of $\vec{x}$ can be computed directly from the meanings of the immediate subconstituents.

Now, suppose we are given such a system of equations. Does it mean that Σ is context free? There is a trivial answer to the problem. Namely, this system of equations always has a solution. As shown above, we can simply assume any meanings for the words (= terminal strings). Subsequently, we insert as the set of meanings the freely generated term algebra and define in this way an abstract semantics. Now we consider the problem under some 'boundary conditions'. The first kind of boundary condition we have already met: we fix the $[\vec{x}]^{\mathsf{S}}$ for all $\vec{x} \in A^*$. In general we can require that for a given $\vec{x}$ and A a certain meaning is in $[\vec{x}]^A$ (bounding from below of $[\vec{x}]^A$), and we may require which meanings are definitely not in $[\vec{x}]^A$ (bounding from above). It turns out that bounding from below is unproblematic, as long as these sets are finite. We simply consider for every A the interpreted language $\mathfrak{I}^A := \bigcup_{\vec{x} \in A^*} \langle \vec{x}, [\vec{x}]^A \rangle$ and apply Theorem 4.5.4. For every A we get a set of terms and certain functions. We unite these systems of signs as follows. For every A we possess a free term algebra with universes U^A and functions $(\mathtt{F}^A_\rho)^\mu$. The set M shall be the free term algebra which is generated by the union of the of the universes U^A, $A \in T$. The functions $\mathtt{F}^\mu_\rho$ are defined in the term algebra as usual.

Now we bound the sets $[\vec{x}]^A$ from above. Here first complications arise. For from (†) we can derive the following estimate.

$$(\ddagger) \qquad \sum_{Z \in F} \left(\prod_{i < \Omega(Z)} |[\vec{x}_i]^{B_i}| \right) \geq |[\vec{x}_0 \cdot \ldots \cdot \vec{x}_{\Omega(Z)-1}]^A|$$

This means roughly speaking that a string cannot have more meanings than it has structural readings (= structure terms). We call the condition (‡) the **count condition**. The count condition can be rephrased as follows.

Lemma 4.5.7 *The count condition is satisfied if and only if there is a constant c such that for almost all $\vec{x}$ and all A:*

$$|[\vec{x}]^A| \leq c^{|\vec{x}|} \ .$$

The proof is left as an exercise. If the count condition is satisfied it is, however, not always possible to construct a context free system of signs.

Now take for every $\vec{x}$ and every A an arbitrary surjective function

$$h(\vec{x}, A) : \bigcup_{Z \in F} \mathtt{F}^{\mu}[[\vec{x}_0]^{B_0}, \ldots, [\vec{x}_{\Omega(Z)-1}]^{B_{\Omega(Z)-1}}] \twoheadrightarrow [\vec{x}_0 \cdot \ldots \cdot \vec{x}_{\Omega(Z)-1}]^A$$

Let $\rho = A \to B_0 \ldots B_{n-1}$ be a rule. Then F_ρ simply is the union of all

$$\langle m_0, m_1, \ldots, m_{n-1}, h(\vec{x}, A)(m_0, \ldots, m_{n-1}) \rangle ,$$

where $\vec{x}$ is an A–string and $h(\vec{x}, A)$ is defined on the n–tuple $\langle m_i : i < n \rangle$. This, however, does not always have to be well defined. For the functions $h(\vec{x}, A)$ need not have disjoint domains for given A. Consider an example. Let $\vec{x}_0$, $\vec{y}_0$, $\vec{x}_1$ and $\vec{y}_1$ be strings. If $[\vec{x}_0]^{B_0} = [\vec{y}_0]^{B_0} = \{m_0\}$ as well as $[\vec{x}_1]^{B_1} = [\vec{y}_1]^{B_1} = \{m_1\}$ then we must have $[\vec{x}_0\vec{x}]^A = [\vec{y}_0\vec{y}_1]^A$. Hence we must have $h(\vec{x}_0\vec{x}_1, A)(\vec{x}_0, \vec{x}_1) = h(\vec{y}_0\vec{y}_1, A)(\vec{y}_0, \vec{y}_1)$. This in turn is a combinatorial problem, which has a solution always if the language is unambigous, but otherwise matters might be less clear.

Let $\rho = A \to B_0 \ldots B_{n-1}$. Notice that the function $\mathtt{F}_\rho$ makes A–meanings from certain B_i–meanings of the constituents of the string. However, often linguists use their intuition to say what an A–string means under a certain analysis, that is to say, structure term. This — as is easy to see — is tantamount to knowing the functions themselves, not only their domains and ranges. For let us assume we have a function which assigns to every structure term $\mathtt{t}$ of an A–string $\vec{x}$ an A–meaning. Then, so we claim the functions $\mathtt{F}_\rho$ are uniquely determined. For let a derivation of $\vec{x}$ be given. This derivation determines derivations of its immediate constituents, which are now unique by assumption. For the tuple of meanings of the subconstituents we know what the function does. Hence, it is clear that for any given tuple of meanings we can say what the function does. (Well, not quite. We do not know what it does on meanings that are not expressed by a B_i–string,

$i < \Omega(f)$. However, on any count we have as much knowledge as we need.)

Let us return to Montague Grammar. Let $\Sigma \subseteq A^* \times T \times M$ be strongly context free, with F the set of modes. We want to show that there is an **AB**–categorial grammar which generates Σ. We have to precisify in what sense we want to understand this. We cannot expect that Σ is any context free system, since **AB**–grammars are always binary branching. This, however, means that we have to postulate other constituents than those of Σ. Therefore we shall only aim to have the same sentence meanings. In what ways we can get more, we shall see afterwards. To start, there is a trivial solution of our problem. If $\rho = A \to B_0 B_1$ is a rule we add a 0–ary mode

$$\mathtt{N}_\rho = \langle \varepsilon, A/B_0/B_1, \lambda x_{B_0}.\lambda x_{B_1}.F_\rho(x_{B_0}, x_{B_1})\rangle \ .$$

This allows us to keep our constituents. However, postulating empty elements — while being quite legitimate in linguistics — does have its drawbacks. It increases the costs of parsing, for example. We shall therefore ask whether one can do without empty categories. This is possible. For, as we have seen, with the help of combinators one can liberate oneself from the straightjacket of syntactic structure. Recall the transformation of a context free grammar into Greibach normal form, as described in Section 2.2. This uses essentially the tool of skipping a rule and of eliminating left recursion. We leave it to the reader to formulate (and prove) an analogon of the skipping of rules for context free sign grammars. This allows us to concentrate on the elimination of left recursion. We look again at the construction of Lemma 2.2.19. Choose a nonterminal X. Assume that we have the following X–productions, where $\vec{\alpha}_j$, $j < m$, and $\vec{\beta}_i$, $i < n$, do not contain X.

$$\rho_j : X \to X \cdot \vec{\alpha}_i, \quad i < m, \qquad \sigma_i : X \to \beta_i, \quad i < n.$$

Further let $\mathtt{F}^\mu_{\rho_j}$, $j < m$, and $\mathtt{F}^\mu_{\sigma_i}$, $i < n$, be given. To keep the proof legible we assume that $\vec{\beta}_j = Y_j$, $\vec{\alpha}_i = U_i$, are nonterminal symbols. (Evidently, this can be achieved by introducing some more nonterminal symbols.) We we have now these rules.

$$\rho_j : X \to X \cdot U_i, \quad i < m, \qquad \sigma_i : X \to Y_i, \quad i < n.$$

So, we generate the following structures.

$$[Y' \, [U_{i_0} \, [U_{i_1} \ldots [U_{i_{n-2}} \, U_{i_{n-1}}] \ldots]]]$$

We want to replace them by these structures instead:

$$[[\ldots [[Y' \, U_{i_0}] \, U_{i_1}] \ldots U_{i_{n-2}}] \, U_{i_{n-1}}]$$

Proceed as in the proof of Lemma 2.2.19. Choose a new symbol Z and replace the rules by the following ones.

$$\begin{array}{llll} \lambda_j : X \to U_j, & j < m, & \nu_i : Z \to Y_i, & i < n, \\ \mu_j : X \to Y_j \cdot Z, & j < m, & \xi \ : Z \to Y_i \cdot Z, & i < n. \end{array}$$

Now define the following functions.

$$\begin{array}{lcl} \mathtt{G}^{\mu}_{\lambda_i} & := & \mathtt{F}^{\mu}_{\sigma_i}, \\ \mathtt{G}^{\mu}_{\mu_i}(x_0, x_1) & := & x_1(x_0), \\ \mathtt{G}^{\mu}_{\nu_i}(x_0, x_1) & := & \lambda x_2.\mathtt{F}^{\mu}_{\rho_i}(x_1(x_2), x_0), \\ \mathtt{G}^{\mu}_{\xi_i}(x_0) & := & \lambda x_0.\mathtt{F}^{\mu}_{\rho_i}(x_0, x_1). \end{array}$$

(The reader is made aware of the fact that these λ–terms serve here as definitional devices. They tell us what the new functions are in terms of the old ones.) Now we have eliminated all left recursion on X. We only have to show that we have not changed the set of X–meanings for any string. To this end, let $\vec{x}$ be an X–string, say $\vec{x} = \vec{y} \cdot \prod_{i<k} \vec{z}_i$, where $\vec{y}$ is a Y_j–string and $\vec{z}_i$ a U_{j_i}–string. Then in the transformed grammar we have the Z–strings

$$\vec{u}_p := \prod_{p \leq i < k} \vec{z}_i$$

and $\vec{x}$ is an X–string. Now we still have to determine the meanings. Let $\mathfrak{m}$ be a meaning of $\vec{y}$ as a Y_i–string and $\mathfrak{n}_i$, $i < k$, a meaning of $\vec{z}_i$ as a U_{j_i}–string. The meaning of $\vec{x}$ as an X–string under this analysis is then

$$\mathtt{F}^{\mu}_{\rho_{j_{n-1}}}(\mathtt{F}^{\mu}_{\rho_{j_{n-2}}}(\ldots(\mathtt{F}^{\mu}_{\rho_{j_0}}(\mathfrak{m},\mathfrak{n}_0),\mathfrak{n}_1),\ldots,\mathfrak{n}_{n-2}),\mathfrak{n}_{n-1}) \ .$$

As a Z–string $\vec{u}_{n-1}$ has the meaning

$$\mathfrak{u}_{n-1} := \mathtt{G}^{\mu}_{\nu_{i_{n-1}}}(\mathfrak{n}_{n-1}) = \lambda x_0.\mathtt{F}^{\mu}_{\rho_i}(x_0,\mathfrak{n}_{n-1}) \ .$$

Then $\vec{u}_{n-2}$ has the meaning

$$\begin{aligned} \mathfrak{u}_{n-2} \ &:= \ \mathtt{G}^{\mu}_{\nu_{i_{n-2}}}(\mathfrak{n}_{n-2},\mathfrak{u}_{n-1}) \\ &= \ \lambda x_2.\mathtt{F}^{\mu}_{\rho_{n-2}}(\mathfrak{u}_{n-1}(x_2),\mathfrak{n}_{n-2}) \\ &= \ \lambda x_2.\mathtt{F}^{\mu}_{\rho_{n-2}}(\mathtt{F}^{\mu}_{\rho_i}(x_2,\mathfrak{n}_{n-1}),\mathfrak{n}_{n-2}). \end{aligned}$$

Inductively we get

$$\mathfrak{u}_{n-j} = \lambda x_0.\mathtt{F}^{\mu}_{\rho_{n-1}}(\mathtt{F}^{\mu}_{\rho_{n-2}}(\ldots\mathtt{F}^{\mu}_{\rho_{n-j}}(x_0,\mathfrak{n}_{n-j}),\ldots,\mathfrak{n}_{n-2}),\mathfrak{n}_{n-1}).$$

If we put $j = n$, and if we apply at last the function $\mathtt{F}^{\mu}_{\mu_j}$ on $\mathfrak{m}$ and the result we finally get that $\vec{x}$ has the same X–meaning under this analysis. The converse shows likewise that every X–analysis of $\vec{x}$ in the transformed grammar can be transformed back into an X–analysis in the old grammar, and the X–meanings of the two are the same.

The reader may actually notice the analogy with the semantics of the Geach rule. There we needed to get new constituent structures by bracketing $[A[B\ C]]$ into $[[A\ B]\ C]$. Supposing that A and B are heads, the semantics of the rule forming $[A\ B]$ must be function composition. This is what the definitions achieve here. Notice, however, that we have no categorial grammar to start with, so the proof given here is not fully analogous. Part of the semantics of the construction is still in the modes themselves, while categorial grammar requires that it be in the meaning of the lexical items.

After some more steps, consisting in more recursion elimination and skipping of rules we are finally done. Then the grammar is in Greibach normal form. The latter can be transformed into an **AB**–categorial grammar, as we have already seen.

Theorem 4.5.8 *Let Σ be a context free linear system of signs. Then there exists an **AB**–categorial grammar that generates Σ.*

The moral to be drawn is that Montague grammmar is actually quite powerful from the point of view of semantics. If the string languages are already context free, then if any context free analysis succeeds, so does an analysis in terms of Montague grammar (supposing here that nothing except linear concatenation is allowed in the exponents). We shall extend this result later to **PTIME**–languages.

Exercise 153. Let G be a linear sign grammar satisfying (a) – (c) but not (d). Construct a grammar H satisfying (a) – (d), which generates the same interpreted language.

Exercise 154. Let G be determined by the following rules.

$$\mathtt{S} \to \mathtt{SS}, \qquad \mathtt{S} \to \mathtt{a}.$$

Show that the set of constituent structures of G cannot be generated by an **AB**–categorial grammar. *Hint.* Let $d(\alpha)$ be the number of occurrences of type constructors in α (type constructors are $\backslash$ or $/$). If $\alpha \to \beta\ \gamma$ then either $d(\beta) > d(\alpha)$ or $d(\gamma) > d(\alpha)$.

Exercise 155. Prove Lemma 4.5.7.

Exercise 156. Let Σ be strongly context free with respect to a binary branching grammar G, which has the following property: there exists a $k \in \omega$ such that for every G–tree $\langle T, <, \sqsubset, \ell\rangle$ and every node $x \in T$ there is a terminal node $y \in T$ with $[y, x] \leq k$. Then there exists an **AB**–categorial grammar for Σ which generates the same constituent structures as G.

Exercise 157. As we have seen above, left recursion can be eliminated from a context free grammar G. Show that there exists a $CCG(\mathbf{B})$ grammar, which generates for every nonterminal X the same set of X–strings. Derive from this that we can write an **AB**–categorial grammar which for every X generates the same X–strings as G. Why does it not follow that $L_B(G)$ can be generated by some **AB**–categorial grammar? *Hint.* For the first part of the exercise consider Exercise 3.4.

Exercise 158. Let $\langle B, A, \zeta, S\rangle$ be an **AB**–categorial grammar. Put $\mathcal{C}^0 := \bigcup_{a\in A} \zeta(a)$. These are the 0th projections. Inductively we put for $\beta^i = \alpha/\gamma \in \mathcal{C}^i$, $\gamma \neq \alpha$, $\beta^{i+1} := \alpha$. In this way we define the projections of the symbols from $\mathcal{C}^0$. Show that by these definitions we get a grammar which satisfies the principles of $\overline{X}$–syntax. *Hint.* The maximal projections are not uniformly in $\mathcal{C}^2$.

4.6 Partiality and Discourse Dynamics

After having outlined the basic setup of Montague Semantics, we shall deal with an issue that has been tacitly shoved aside, namely *partiality*. This will lead us directly into a discussion of Discourse Representation Theory and Dynamic Predicate Logic. The name 'partial logic' covers a wide variety of logics that deal with radically different problems. We shall deal with two of them. The first is that of partiality as undefinedness. The second is that of partiality as ignorance. We start with partiality as undefinedness.

Consider the assignment $y := (x+1)/(u^2-9)$ to y in a program. This clause is potentially dangerous, since u may equal 3, in which case no value can be assigned to y. Similarly, for a sequence $\mathfrak{a} = (a_n)_{n\in\omega}$, $\lim \mathfrak{a} := \lim_{n\to\infty} a_n$ is defined only if the series is convergent. If not, no value can be given. Or in type theory, a function f may only be applied to x if f has type $\alpha \to \beta$ for certain α and β x has type α. In linguistical and philosophical literature, this phenomenon is known as **presupposition**. It is defined as a relation between propositions (see (van der Sandt, 1988)).

Definition 4.6.1 *A proposition φ* ***presupposes*** *χ if both $\varphi \vdash \chi$ and $\neg\varphi \vdash \chi$. We write $\varphi \gg_{\vdash} \chi$ (or simply $\varphi \gg \chi$) to say that φ presupposes χ.*

The definition needs only the notion of a negation in order to be well defined. Clearly, in boolean logic this definition gives pretty uninteresting results. φ presupposes χ in PC if and only if χ is a tautology. However, if we have more than two truth values, interesting results appear. First, notice that we have earlier remedied partiality by assuming a 'dummy' element $\star$ that a function assumes as soon as it is not defined on its regular input. Here, we shall remedy the situation by giving the expression itself the truth value $\star$. That is to say, rather than making functions themselves total, we make the assignment of truth values a total function. This has different consequences, as will be seen. Suppose that we totalize the operator $\lim_{n\to\omega}$ so that it can be applied to all sequences. Then if $(a_n)_{n\in\omega}$ is not a convergent series, say $a_n = (-1)^n$, $3 = \lim_{n\to\infty} a_n$ is not true, since $\lim_{n\to\omega} a_n = \star$ and $3 \neq \star$. The negation of the statement will then be true. This is effectively what Russel (1905) and Kempson (1975) claim. Now suppose we say that $3 = \lim_{n\to\infty} a_n$ has no truth value; then $3 \neq \lim_{n\to\infty} a_n$ also has no truth value. To nevertheless be able to deal with such sentences rather than simply excluding them from discourse, we introduce a third truth value, $\star$. The question is now: how do we define the 3–valued counterparts of $-$, $\cap$ and $\cup$? In order to keep confusion at a minimum, we agree on the following conventions. $\vdash_3$ denotes the 3–valued consequence relation determined by a matrix $\mathfrak{M} = \langle\{0, 1, \star\}, \Pi, \{1\}\rangle$, where Π is an interpretation of the connectives. We shall assume that F consists of a subset of the set of the 9 unary and 27 binary symbols, which represent the unary and binary functions on the three element set. This defines Ω. Then $\vdash_3$ is uniquely fixed by F, and the logical connectives will receive a distinct name every time we choose a different function on $\{0, 1, \star\}$. What remains to be solved, then, is not what logical language to use but rather by what connective to translate the ordinary language connectors `not`, `and`, `or`, and `if...then`. Here, we assume that whatever interprets them is a function on $\{0, 1, \star\}$ ($\{0, 1, \star\}^2$), whose restriction to $\{0, 1\}$ is its boolean counterpart, which is already given. For those functions, the 2–valued consequence is also defined and denoted by $\vdash_2$.

Now, if $\star$ is the truth value reserved for the otherwise truth valueless statements, we get the following three valued logic, which is due to Bochvar (1938). Its characteristics is the fact that undefinedness is strictly hereditary.

	$-$
0	1
1	0
$\star$	$\star$

$\cap$	0	1	$\star$
0	0	0	$\star$
1	0	1	$\star$
$\star$	$\star$	$\star$	$\star$

$\cup$	0	1	$\star$
0	0	1	$\star$
1	1	1	$\star$
$\star$	$\star$	$\star$	$\star$

The basic connectives are $\neg$, $\wedge$ and $\vee$, which are interpreted by $-$, $\cap$ and $\cup$. Here is a characterization of presupposition in Bochvar's Logic. Call an n–ary connective J a **Bochvar–connective** if $\Pi(J)(\vec{x}) = \star$ if and only if $x_i = \star$ for some $i < n$.

Proposition 4.6.2 *Let Δ and χ be composed using Bochvar–connectives. $\Delta \vdash_3 \chi$ if and only if (i) Δ is not classically satisfiable or (ii) $\Delta \vdash_2 \chi$ and $var(\chi) \subseteq var[\Delta]$.*

Proof. Suppose that $\Delta \vdash_3 \chi$ and that Δ is satisfiable. Let β be a valuation such that $\overline{\beta}(\delta) = 1$ for all $\delta \in \Delta$. Put $\beta^+(p) := \beta(p)$ for all $p \in var[\Delta]$ and $\beta^+(p) := \star$ otherwise. Suppose that $var(\chi) - var[\Delta] \neq \varnothing$. Then $\overline{\beta^+}(\chi) = \star$, contradicting our assumption. Hence, $var(\chi) \subseteq var[\Delta]$. It follows that every valuation that satisfies Δ also satisfies χ, since the valuation does not assume $\star$ on its variables (and can therefore be assumed to be a classical valuation). Now suppose that $\Delta \nvdash_3 \chi$. Then clearly Δ must be satisfiable. Furthermore, by the argument above either $var(\chi) - var[\Delta] \neq \varnothing$ or else $\Delta \vdash_2 \chi$. $\square$

This characterization can be used to derive the following corollary.

Corollary 4.6.3 *Let φ and χ be composed by Bochvar–connectives. Then $\varphi \gg \chi$ if and only if $var(\chi) \subseteq var(\varphi)$ and $\vdash_2 \chi$.*

Hence, although Bochvar's Logic makes room for undefinedness, the notion of presupposition is again trivial. Bochvar's Logic seems nevertheless adequate as a treatment of the ι–operator. It is formally defined as follows.

Definition 4.6.4 *ι is a partial function from predicates to objects such that $\iota x.\chi(x)$ is defined if and only if there is exactly one a such that $\chi(a)$, and in that case $\iota x.\chi(x) := a$.*

Most mathematical statements which involve presuppositions are instances of a (hidden) use the ι–operator. Examples are the integral and the limit. In ordinary language, ι corresponds to the definite determiner `the`. Using the ι–operator, we can bring out the difference between the bivalent interpretation and the three valued one. Define the predicate cauchy' on infinite sequences as follows:

$$\mathsf{cauchy}'(\mathfrak{a}) := (\forall \varepsilon > 0)(\exists n)(\forall m \geq n)|\mathfrak{a}(m) - \mathfrak{a}(n)| < \varepsilon$$

This is in formal terms the definition of a Cauchy sequence. Further, define a predicate cum' as follows.

$$\mathsf{cum}'(\mathfrak{a})(x) := (\forall \varepsilon > 0)(\exists n)(\forall m \geq n)|\mathfrak{a}(m) - x| < \varepsilon$$

This predicate says that x is a cumulation point of $\mathfrak{a}$. Now, we may set $\lim \mathfrak{a} := \iota x.\mathsf{cum}'(\mathfrak{a})(x)$. Notice that $\mathsf{cauchy}'(\mathfrak{a})$ is equivalent to

$$(\exists x)(\mathsf{cum}'(\mathfrak{a})(x) \wedge (\forall y)(\mathsf{cum}'(\mathfrak{a})(y) \rightarrow y \doteq x))$$

. This is exactly what must be true for $\lim \mathfrak{a}$ to be defined.

(4.6.1) `The limit of` $\mathfrak{a}$ `equals three.`
(4.6.2) $\iota x.\mathsf{cum}'(\mathfrak{a})(x) = 3$
(4.6.3) $(\exists x)(\mathsf{cum}'(\mathfrak{a})(x) \wedge (\forall y)(\mathsf{cum}'(\mathfrak{a})(y) \rightarrow y \doteq x) \wedge x \doteq 3)$.

Under the analysis (4.6.2) the sentence (4.6.1) presupposes that $\mathfrak{a}$ is a Cauchy–sequence. (4.6.3) does not presuppose that. However, the dilemma for the translation (4.6.3) is that the negation of (4.6.1) is also false (at least in ordinary judgement). What this means is that the translation of (4.6.4) is not (4.6.6) but (4.6.7) in the bivalent analysis, while in the three valued analysis, the negation of (4.6.2) is appropriate.

(4.6.4) `The limit of` $\mathfrak{a}$ `does not equal three.`

(4.6.5) $\neg(\iota x.\mathsf{cum}'(\mathfrak{a})(x) \doteq 3)$

(4.6.6) $\neg(\exists x)(\mathsf{cum}'(\mathfrak{a})(x) \wedge (\forall y)(\mathsf{cum}'(\mathfrak{a})(y) \to y \doteq x) \wedge x \doteq 3)$

(4.6.7) $(\exists x)(\mathsf{cum}'(\mathfrak{a})(x) \wedge (\forall y)(\mathsf{cum}'(\mathfrak{a})(y) \to y \doteq x) \wedge \neg(x \doteq 3))$.

Proponents of bivalence will have to explain how the translation (4.6.7) is arrived at. Furthermore, the strategy of the bivalent translation is successful only if it thereby eliminates the imminent lack of truth value. However, it is not clear that this can be achieved. The best example is that of the truth predicate. Suppose we define the semantics of T on the set of natural numbers as follows.

$$(\ddagger) \qquad \mathsf{T}(\ulcorner\varphi\urcorner) \leftrightarrow \varphi$$

Here, $\ulcorner\varphi\urcorner$ is, say, the Gödel code of φ. Then, as one can show that there is a χ such that $\mathsf{T}(\ulcorner\chi\urcorner) \leftrightarrow \neg\chi$ is provable in first–order number theory, we derive an inconsistency of the latter if we add the schema ($\ddagger$). The sentence χ corresponds to the following liar paradox.

(4.6.8) `This sentence is false.`

Thus, as Tarski has observed, a truth predicate that is consistent with the facts in a sufficiently rich theory must be partial. As sentence (4.6.8) shows, natural languages are sufficiently rich to produce the same effect. Since we do not want to give up the correctness of the truth predicate (or the falsity predicate), the only alternative is to assume that it is partial. If so, however, there is no escape from the use of three valued logic, since bivalence must fail.

Let us assume therefore that we three truth values. What Bochvar's Logic gives us is called the logic of hereditary undefinedness. For many reasons it is problematic, however. Consider the following two examples.

(4.6.9) `If` $\mathfrak{a}$ `and` $\mathfrak{b}$ `are convergent sequences,` $\lim(\mathfrak{a}+\mathfrak{b}) = \lim\mathfrak{a} + \lim\mathfrak{b}$`.`

(4.6.10) $\mathsf{if}\ u \neq 3\ \mathsf{then}\ y := (x+1)/(u^2-9)\ \mathsf{else}\ y := 0\ \mathsf{fi}$

By Bochvar's Logic, (4.6.9) presupposes that $\mathfrak{a}$, $\mathfrak{b}$ and $\mathfrak{a}+\mathfrak{b}$ are convergent series. (4.6.10) presupposes that $u \neq 3$. However, it is clear that none of the two sentences have nontrivial presuppositions. Let us illustrate this with (4.6.9). Intuitively, the if–clause preceding the equality statement excludes all sequences from consideration where $\mathfrak{a}$ and $\mathfrak{b}$ are nonconvergent sequences. One can show that $\mathfrak{a}+\mathfrak{b}$, the pointwise sum of $\mathfrak{a}$ and $\mathfrak{b}$, is then also convergent. Hence, the if–clause covers all cases of partiality. The statement $\lim(\mathfrak{a}+\mathfrak{b}) = \lim \mathfrak{a} + \lim \mathfrak{b}$ never fails. Similarly, `and` has the possibility to eliminate presuppositions.

(4.6.11) $\mathfrak{a}$ `and` $\mathfrak{b}$ `are convergent series and` $\lim(\mathfrak{a}+\mathfrak{b})$
$= \lim \mathfrak{a} + \lim \mathfrak{b}$ `.`

(4.6.12) $u := 4; y := (x+1)/(u^2-9).$

As it turns out, there is an easy fix for that. Simply associate the following connectives with `if...then` and `and`.

$\to'$	0	1	$\star$
0	1	1	1
1	0	1	$\star$
$\star$	$\star$	$\star$	$\star$

$\wedge'$	0	1	$\star$
0	0	0	1
1	0	1	$\star$
$\star$	$\star$	$\star$	$\star$

$\vee'$	0	1	$\star$
0	0	1	$\star$
1	1	1	1
$\star$	$\star$	$\star$	$\star$

The reader may take notice of the fact that while $\wedge'$ and $\to'$ are reasonable candidates for `and` and `if`...`then`, $\vee'$ is not as good for `or`.

In the linguistic literature, various attempts have been made to explain these facts. First, we distinguish in a sentence its *presupposition* from its *assertion*. The definition of these terms is somewhat cumbersome. The general idea is that the presupposition of a sentence is a characterization of those circumstances under which it is either true or false, and the assertion is what the sentence says when it is either true of false (that is to say, the assertion tells us when the sentence is true given it is either true or false). Let us attempt to define this. Let φ be a proposition. Call χ a **generic presupposition of** φ if the following holds. (a) $\varphi \gg \chi$, (b) if $\varphi \gg \psi$ then $\chi \vdash_3 \psi$. If χ is a generic presupposition of φ, $\chi \to \varphi$ is called an **assertion** of φ. First, notice that presuppositions are only defined up to interderivability. This is not a congruence. We may have $\varphi \dashv\vdash_3 \chi$ without φ and χ receiving the same truth value under all assignments. Namely, $\varphi \dashv\vdash_3 \chi$ if and only if φ and χ are truth equivalent, that is, $\overline{\beta}(\varphi) = 1$ exactly when $\overline{\beta}(\chi) = 1$. In order to have full equivalence, we must also require $\neg\varphi \dashv\vdash_3 \neg\chi$. In fact, the whole problem has a trivial solution: $\varphi \vee \neg\varphi$ is actually a generic presupposition of φ, as is easily checked. However, $\varphi \vee \neg\varphi$ presupposes itself. Ideally, however, one wishes the presupposition not to presuppose itself, rather, it should be **bivalent**, where φ is **bivalent** if for every valuation β: $\overline{\beta}(\varphi) \in \{0, 1\}$. Define the following connective.

$\downarrow$	0	1	$\star$
0	$\star$	0	$\star$
1	$\star$	1	$\star$
$\star$	$\star$	$\star$	$\star$

Definition 4.6.5 *Let φ be a proposition. χ is a* ***generic presupposition of φ with respect to*** $\vdash$ *if (a) $\varphi \gg \chi$, (b) χ is bivalent and (c) if $\varphi \gg \psi$ then $\chi \vdash \psi$. Write $P(\varphi)$ for the generic presupposition (if it exists), and $A(\varphi)$ for the assertion.*

It is not a priori clear that a proposition has a generic presupposition. A case in point is the truth predicate. To define the range of sentences to which it can be applied seems to be a really demanding task and has inspired a lot of research. However, the following is true.

Proposition 4.6.6 *Let φ have a generic presupposition. Then $A(\varphi)$ is bivalent. Moreover, $\varphi \equiv A(\varphi) \downarrow P(\varphi)$.*

The **projection algorithm** is a procedure that assigns generic presuppositions to complex propositions by induction over their structure. Here is a projection algorithm for the connectives defined so far.

$$\begin{array}{llll}
A(\neg\varphi) & = \neg A(\varphi) & P(\neg\varphi) & = P(\varphi) \\
A(\varphi \wedge \chi) & = A(\varphi) \wedge A(\chi) & P(\varphi \wedge \chi) & = P(\varphi) \wedge P(\chi) \\
A(\varphi \vee \chi) & = A(\varphi) \vee A(\chi) & P(\varphi \vee \chi) & = P(\varphi) \wedge P(\chi) \\
A(\varphi \to \chi) & = A(\varphi) \to A(\chi) & P(\varphi \to \chi) & = P(\varphi) \wedge P(\chi) \\
A(\varphi \wedge' \chi) & = A(\varphi) \wedge A(\chi) & P(\varphi \wedge' \chi) & = P(\varphi) \\
 & & & \quad \wedge(A(\varphi) \to P(\chi)) \\
A(\varphi \vee' \chi) & = A(\varphi) \vee A(\chi) & P(\varphi \vee' \chi) & = P(\varphi) \\
 & & & \quad \wedge(\neg A(\varphi) \to P(\chi)) \\
A(\varphi \to' \chi) & = (A(\varphi) \wedge P(\varphi)) & P(\varphi \to' \chi) & = P(\varphi) \\
 & \quad \to A(\chi) & & \quad \wedge(A(\varphi) \to P(\chi))
\end{array}$$

It is an easy matter to define projection algorithms for all connectives. The prevailing intuition is that the three valued character of `and`, `or` and `if...then` is best explained in terms of **context change**. A **text** is a sequence of propositions, say $\Delta = \langle \delta_i : i < n \rangle$. A text is **coherent** if for every $i < n$: $\langle \delta_j : j < i \rangle \vdash_3 \delta_i \vee \neg\delta_i$. In other words, every member is either true or false given that the previous propositions are considered true. (Notice that the ordering is important now.) In order to extend this to parts of the δ_j we define the **local context** as follows.

Definition 4.6.7 *Let $\Delta = \langle \delta_i : i < n \rangle$. The **local context** of δ_j is $\langle \delta_i : i < j \rangle$. For a subformula occurrence of δ_j, the **local context** of that occurrence is defined as follows.*

1. *If Σ is the local context of $\varphi \wedge' \chi$ then (a) Σ is the local context of φ and (b) $\Sigma; \varphi$ is the local context of χ.*
2. *If Σ is the local context of $\varphi \to' \chi$ then (a) Σ is the local context of φ and (b) $\Sigma; \varphi$ is the local context of χ.*
3. *If Σ is the local context of $\varphi \vee' \chi$ then (a) Σ is the local context of φ and (b) $\Sigma; \neg\varphi$ is the local context of χ.*

*δ_j is **bivalent in its local context** if for all valuations that make all formulae in the local context true, δ_j is true or false.*

The presupposition of φ is the formula χ such that φ is bivalent in the context χ, and which implies all other formulae that make φ bivalent. It so turns out that the context dynamics define a three valued extension of a 2–valued connective, and conversely. The above rules are an exact match. Such formulations have been given by Irene Heim (1983), Lauri Karttunen (1974) and also Jan van Eijck (1994). Indeed, the reason why computer programs do not fail even if some statement carry presuppositions is that whenever the clause carrying the presupposition is evaluated, that presupposition is satisfied. In other words, the local context of that clause satisfies the presuppositions. What the local context is, is defined by the evaluation procedure for the connectives. It is strictly speaking not necessary to evaluate α`and` β by first evaluating α and then β only if α is true. We could just as well do the converse: evaluate α whenever β is true. In fact, in (Kracht, 1994) it is shown that the local context clauses are redundant. All that needs to be specified is the directionality of evaluation. Otherwise one gets connectives that extend the boolean connectives but not in a proper way (see below on that notion).

Presuppositions also occur inside quantifiers and propositional attitudes. Here the behaviour is less clear. We shall give only some sketch.

(4.6.13) `Every bachelor of the region got a letter from that marriage agency.`

(4.6.14) `Every person in that region is a bachelor.`

(4.6.15) `John believes that his neighbour is a bachelor.`

Previously, we have translated `every` using the unary quantifier $\forall$. In view of the fact that (4.6.13) is true even if not everybody in the region is a bachelor, rather, it is true if and only if there is no non–bachelor, the following should now be done. $(\forall x)\varphi$ is true if and only if there is no x for which $\varphi(x)$ is false. $(\forall x)\varphi$ is false if there is no x for which $\varphi(x)$ is false. Thus, the presupposition effectively restricts the range of the quantifier. $(\forall x)\varphi$ is bivalent. This predicts that (4.6.14) has no presuppositions. On (4.6.15) the intuitions vary. One might say that it does not have any presuppositions, or else that it presupposes that the neighbour is a man (or perhaps: that John believes that his neighbour is a man). This is deep water (see (Geurts, 1998)).

Now we come to the second interpretation of partiality, namely *ignorance*. Let $\star$ now stand for the fact that the truth value is not known. Also here the resulting logic is not unique. Let us take the example of a valuation β that is only defined on some of the variables. Now let us be given a formula φ. Strictly speaking $\overline{\beta}(\varphi)$ is not defined on φ if the latter contains a variable that is not in the domain of β. On the other hand, there are clear examples of propositions that receive a definite truth–value no matter how we extend β to a total function. For example, even if β is not defined on p, every extension of it must make $p \vee \neg p$ true. Hence, we might say that β also makes $p \vee \neg p$ true. This is the idea of *supervaluations* by Bas van Fraassen. Say that φ is **sv–true** (**sv–false**) under β if φ is true under every total $\gamma \supseteq \beta$. If φ is neither sv–true nor sv—false, call it **sv–indeterminate**. Unfortunately, there is no logic to go with this approach. Look at the interpretation of `or`. Clearly, if either φ or χ is sv–true, so is their disjunction. However, what if both φ and χ are sv–indeterminate?

$\cup$	0	1	$\star$
0	0	1	$\star$
1	1	1	1
$\star$	$\star$	1	?

The formula $p \vee q$ is sv–indeterminate under the empty valuation. It has no definite truth value, because both p and q could turn out to be either true or false. On the other hand, $p \vee \neg p$ is sv–true under the empty valuation, even though both p and $\neg p$ are sv–indeterminate. So, the supervaluation approach is not so well–suited. Stephen Kleene actually had the idea of doing a worst case interpretation: if you can't always say what the value is, fill in $\star$. This gives the so called **Strong Kleene Connectives** (the weak ones are like Bochvar's).

	$-$
0	1
1	0
$\star$	$\star$

$\cap^\diamond$	0	1	$\star$
0	0	0	0
1	0	1	$\star$
$\star$	0	$\star$	$\star$

$\cup^\diamond$	0	1	$\star$
0	0	1	$\star$
1	1	1	1
$\star$	$\star$	1	$\star$

$\supset^\diamond$	0	1	$\star$
0	1	1	1
1	0	1	$\star$
$\star$	$\star$	1	$\star$

These connectives can be defined in the following way. Put $1^\diamond := \{1\}$, $0^\diamond := \{0\}$ and $\star := \{0,1\}$. Now put

$$f^\diamond(x_0^\diamond, x_1^\diamond) := f[x_0^\diamond \times x_1^\diamond]$$

For example, $\cup^\diamond(\langle 0^\diamond, \star^\diamond\rangle) = \cup[\{0\} \times \{0,1\}] = \{0 \cup 0, 0 \cup 1\} = \{0,1\} = \star^\diamond$. So, we simply take sets of truth values and calculate with them.

A more radical account of ignorance is presented by intuitionism. A constructivist denies that the truth or falsity of a statement can always be assessed directly. In particular, an existential statement is true only if we produce an instance that satisfies it. A universal statement can be considered true only if we possess a proof that any given element satisfies it. For example, Goldbach's conjecture is that every even number greater than 2 is the sum of two primes. According to a constructivist, at present it is neither true nor false. For on the one hand we have no proof that it holds, on the other hand we know of no even number greater than 2 of which we can show it is not the sum of two primes. Both constructivists and intuitionists unanimously reject axiom (a3). They also reject $p \vee \neg p$, the so called **Law of the Exluded Middle**. The difference between a constructivist and an intuitionist is the treatment of negative evidence. While a constructivist accepts basic negative evidence, for example, that this lemon is not green, for an intuitionist there is no such thing as direct evidence to the contrary. We only witness the absence of the fact that the lemon is green. Both, however, are reformist in the sense that they argue that the mathematical connectives `and`, `or`, `not`, and `if`...`then` have a different meaning. However, one can actually give a reconstruction of both inside classical mathematics. We shall deal first with intuitionism. Here is a new set of connectives.

$$\begin{array}{lcl} \neg^i \varphi & := & \Box(\neg\varphi) \\ \varphi \vee^i \chi & := & \varphi \vee \chi \\ \varphi \wedge^i \chi & := & \varphi \wedge \chi \\ \varphi \rightarrow^i \chi & := & \Box(\varphi \rightarrow \chi) \end{array}$$

Call an I–proposition a proposition formed from variables and $\bot$ using only the connectives just defined.

Definition 4.6.8 *An* ***I–model*** *is a pair $\langle P, \leq, \beta\rangle$, where $\langle P, \leq\rangle$ is a partially ordered set and $\beta(p) =\uparrow \beta(p)$ for all variables p.*

Intuitively, the nodes of P represent stages in the development of knowledge. Knowledge develops in time along $\leq$. We say that x **accepts** φ if $\langle P, \leq, x, \beta\rangle \vDash \varphi$, and the x **knows** φ if $\langle P, \leq, x, \beta\rangle \vDash \Box\varphi$. By definition of β, once a proposition p is accepted, it is accepted for good and therefore considered known. Actually, the Gödel translation does the following: it assigns to each variable p the formula $\Box p$. Thus, intuitionistically the statement that p may therefore be understood as 'p is known' rather than 'p is accepted'. The systematic conflation of knowledge and simple temporary acceptance as true is the main feature of intuitionistic logic.

Proposition 4.6.9 *Let $\langle P, \leq, \beta\rangle$ be an I–model and φ an I–proposition. Then $\overline{\beta}(\varphi) = \uparrow \overline{\beta}(\varphi)$ for all φ.*

Constructivism in the definition by Nelson adds to intuitionism a second valuation for those variables that are definitely rejected, and allows for the possibility that neither is the case. (However, a variables is never both accepted and rejected.) This is reformulated as follows.

Definition 4.6.10 *A* ***C–model*** *is a pair $\langle P, \leq, \beta\rangle$, where $\langle P, \leq\rangle$ is a partially ordered set and $\beta : V \times P \to \{0, 1, \star\}$ such that if $\beta(p, v) = 1$ and $v \leq w$ then also $\beta(p, w) = 1$, and if $\beta(p, v) = 0$ and $v \leq w$ then $\beta(p, w) = 0$. We write $\langle P, \leq, x, \beta\rangle \vDash^+ p$ if $\beta(p, x) = 1$ and $\langle P, \leq, x, \beta\rangle \vDash^- p$ if $\beta(p, x) = 0$.*

We can interpret any propositional formula over 3–valued logic that we have defined so far. We have to interpret $\Box$ and $\Diamond$, however.

$$\begin{array}{lll}
\langle P, \leq, x, \beta\rangle \vDash^{+} \Box\varphi & :\Leftrightarrow & \text{for no } y \geq x : \langle P, \leq, y, \beta\rangle \vDash^{-} \varphi \\
\langle P, \leq, x, \beta\rangle \vDash^{-} \Box\varphi & :\Leftrightarrow & \text{there is } y \geq x : \langle P, \leq, y, \beta\rangle \vDash^{-} \varphi \\
\langle P, \leq, x, \beta\rangle \vDash^{+} \Diamond\varphi & :\Leftrightarrow & \text{there is } y \geq x : \langle P, \leq, y, \beta\rangle \vDash^{+} \varphi \\
\langle P, \leq, x, \beta\rangle \vDash^{-} \Box\varphi & :\Leftrightarrow & \text{for no } y \geq x : \langle P, \leq, y, \beta\rangle \vDash^{+} \varphi
\end{array}$$

Now define the following new connectives.

$$\begin{array}{lll}
\neg^{c}\varphi & := & \neg\varphi \\
\varphi \vee^{c} \chi & := & \varphi \vee^{\diamond} \chi \\
\varphi \wedge^{c} \chi & := & \varphi \wedge^{\diamond} \chi \\
\varphi \to^{c} \chi & := & \Box(\varphi \to^{\diamond} \chi)
\end{array}$$

In his data semantics (see (Veltman, 1985)), Frank Veltman uses constructive logic and proposes to interpret `must` and `may` as $\Box$ and $\Diamond$, respectively. What is interesting is that the set of points accepting $\Diamond\varphi$ is lower closed but *not* necessarily upper closed, while the set of points rejecting it is upper but not necessarily lower closed. The converse holds with respect to $\Box$. This is natural, since if our knowledge grows there are less things that *may* be true but more that *must* be.

The interpretation of the arrow carries the germ of the relational interpretation discussed here. A different strand of thought is the theory of conditionals (see again (Veltman, 1985) and also (Gärdenfors, 1988)). The conditional $\varphi > \chi$ is accepted as true under Ramsey's interpretation if, on taking φ as a hypothetical assumption (doing as if φ is the case) and performing the standard reasoning, we find that χ is true as well. Notice that after this routine of hypothetical reasoning we retract the assumption that φ. In Gärdenfors models, there are no assignments in the ordinary sense. A proposition φ is mapped directly onto a function, the update function U_{φ}. The states in the Gärdenfors model carry no structure.

Definition 4.6.11 *A **Gärdenfors model** is a pair $\langle G, U\rangle$, where G is a set, and $U : Fml \to G^G$ subject to the following constraints.*

1. *For all χ: $U_\chi \circ U_\chi = U_\chi$.*
2. *For all φ and χ: $U_\varphi \circ U_\chi = U_\chi \circ U_\varphi$.*

*We say that $x \in G$ **accepts** φ if $U_\varphi(x) = x$.*

We may introduce the following relation: $x \leq y$ if and only there is a finite set $\{\chi_i : i < n\}$ such that $y = U_{\chi_0} \circ U_{\chi_1} \circ \ldots \circ U_{\chi_{n-1}}(x)$. The reader may verify that this relation is reflexive and transitive. If we require that $x = y$ if and only if x and y accept the same propositions, then this ordering is also a partial ordering. We can define as follows. If $U_\chi = U_\varphi \circ U_\psi$ then we write $\chi = \varphi \wedge \psi$. Hence, if our language actually has a conjunction, the latter is a condition on the interpretation of it. To define $\to$, Gärdenfors does the following. Suppose that for all φ and χ there exists a δ such that $U_\varphi \circ U_\delta = U_\chi \circ U_\delta$. Then we simply put $U_{\varphi\leftrightarrow\psi} := U_\delta$. Finally, since $\varphi \to \psi$ is equivalent to $\varphi \leftrightarrow \varphi \wedge \psi$, once we have $\wedge$ and $\leftrightarrow$, we can also define $\to$. For negation we need to assume the existence of an inconsistent state. The details need not concern us here. Obviously, Gärdenfors models are still more general than data semantics. In fact, any kind of logic can be modelled by a Gärdenfors model (see exercises).

Notes on this section. It is an often discussed problem whether or not a statement of the form $(\forall x)\varphi$ is true if there are no x at all. Equivalently, in three valued logic, it might be said that $(\forall x)\varphi$ is undefined if there is no x such that $\varphi(x)$ is defined.

Exercise 159. A three valued binary connective $\circ$ satisfies the Deduction Theorem if for all Δ, φ and χ: $\Delta; \varphi \vdash_3 \chi$ if and only if $\Delta \vdash_3 \varphi \circ \chi$. Establish a truth table for a connective that satisfies the Deduction Theorem. Does any of the implications defined above have this property?

Exercise 160. Let $\mathfrak{L}$ be a language and $\vdash$ a structural consequence relation over $\mathfrak{L}$. Let $G_\vdash$ be the set of theories of $\vdash$. For

$\varphi \in \mathfrak{L}$, let $U_\varphi : T \mapsto (T \cup \{\varphi\})^\vdash$. Show that this is a Gärdenfors model. Show that the set of formulae accepted by all $T \in G_\vdash$ is exactly the set of tautologies of $\vdash$.

4.7 Discourse Representation Theory and Referent Systems

One of the problems of Montague Semantics was the treatment of so called extrasentential anaphora. These are the anaphorical expressions whose antecedent is not within the same sentence and also not within a superordinate sentence. This is a problem that GB theory also has, since lack of c–command generally inhibits the possibility of binding (see Section 6.5).

(4.7.1) `Paul sees a red car. It pleases him much.`
(4.7.2) `If Paul sees a red car then it pleases him.`

We assume that `it` refers to whatever `a red car` refers to and `him` refers to whatever `Paul` refers to, namely Paul. Therefore, `Paul` is the antecedent of `him`. We denote this by giving the antecedent a superscript and the pronoun a subscript. So we use the following annotated sentences.

(4.7.3) $[\texttt{Paul}]^0$ `sees` $[\texttt{a red car}]^1$. $\texttt{It}_1$ `pleases` $\texttt{him}_0$ `much.`
(4.7.4) `If` $[\texttt{Paul}]^0$ `sees` $[\texttt{a red car}]^1$ `then` $\texttt{it}_1$ `pleases` $\texttt{him}_1$.

To deal with these sentences, Hans Kamp has developed in (Kamp, 1981) the **Discourse Representation Theory** (**DRT**, see also (Kamp and Reyle, 1993) and the survey article (van Eijck and Kamp, 1997)). DRT wants to answer the following two questions.

1. If `a(n)` actually is to be interpreted as an existential quantifier, why is it effectively a universal quantifier in (4.7.4)?

2. Why are certain occurrences of pronouns good and others not?

Why we choose this interpretation rather than another is not what DRT can tell us. It assumes that we have chosen antecedents for pronouns in the form of the indexing. To see that there is a problem with the quantifiers, look at (4.7.4). On the surface it has the form $((\exists x)\varphi(x)) \to \chi(x)$, with x occurring free in χ. Intuitively speaking this says, however, that every red car which Paul sees pleases him; so the meaning is $(\forall x)(\varphi(x) \to \chi(x))$. The emergence of the universal quantifier is mysterious. For notice that in predicate logic $((\exists x)\varphi) \to \chi \equiv (\forall x)(\varphi \to \chi)$ only if x does not occur in χ. In (4.7.4) this is not the case.

Now for the second question. We observe the following.

(4.7.5) `If Paul sees [a red car]`^{1} `it`$_1$ `pleases him much.` *`It`$_1$ `has an airbag.`

(4.7.6) `Paul sees [a red car]`^{1}`. It`$_1$ `pleases him much.` `It`$_1$ `has an airbag.`

In (4.7.5) the second occurrence of `him`$_1$ is illicit. This does not seem to have syntactic reasons. For c–command as an explanation is out of the question. The if–clause is not the main clause (this must be argued for; in German for example, the verb is not in second place, which is a good indication of the fact that we have a subordinate clause). So, neither occurrence of `it`$_1$ is c–commanded by its antecedent. If we exchange the first sentence by (4.7.1) we get (4.7.6). Here, however, the pronoun is acceptable, even though c–command still does not obtain.

We fix an arbitrary semantic representation language $\mathfrak{L}$, for example the language of first order predicate logic. This fixes the notion of a variable and that of a formula.

Definition 4.7.1 *The set of* ***discourse representation structures (DRSs) over*** $\mathfrak{L}$ *is defined as follows.*

1. *If δ is a $\mathfrak{L}$–formula, then δ is a DRS over $\mathfrak{L}$.*

2. *If $\Delta \subseteq \mathfrak{L}$ is a set of DRSs and V a set of variables then $[V : \Delta]$ is a DRS over $\mathfrak{L}$.*

3. *If δ and δ' are DRSs over $\mathfrak{L}$ then so is $\neg\delta$, $\delta \Rightarrow \delta'$ and $\delta \vee \delta'$.*

Often a DRS is represented in the form of a box and the set braces are ignored. For example $[\{x_0, x_3\} : \{P(x_0), Q(x_3, x_0); S(x_3, x_3)\}]$ is written down as follows.

x_0, x_3
$P(x_0); Q(x_3, x_0); S(x_3, x_3)$.

A DRS is therefore either a term over DRSs (created with the help of $\neg$ and $\vee$) or a pair $[V : \Delta]$, where V is a set of variables and Δ a set of DRSs. In the second case we call V the **head** of δ and Δ the **body**. This terminology is not by chance: a DRS looks quite similar to an `Algol` program. At first, variables are declared and then the program proper follows. The terminology is the same as in `Algol`. Now it is required in `Algol` to declare every variable before using it (see also the Section 2.6). Exactly such a condition also exists for DRSs.

Definition 4.7.2 *In a DRS is, **direct accessibility** between occurrences of sub–DRSs defined as follows.*

1. *If $\delta = [V : \Delta]$ and $\delta' \in \Delta$ then the occurrence of δ' in δ is **directly accessible from** δ.*

2. *If $\delta = (\neg\delta')$ then the occurrence of δ' in δ is **directly accessible from** δ.*

3. *If $\delta = (\delta' \vee \delta'')$ then the occurrences of δ' and δ'' in δ are **directly accessible from** δ.*

4. *If $\delta = (\delta' \Rightarrow \delta'')$ then the occurrences of δ' and δ'' in δ is **directly accessible from** δ. Further, the named occurrence of δ'' is **directly accessible from** the named occurrence of δ'.*

*The relation of **accessibility** is the transitive closure of relation of direct accessibility.*

Definition 4.7.3 *Let δ be a DRS and x a variable. An occurrence of x in a formula φ is called* ***bound*** *if there is a DRS $[V : \Delta]$ which contains this occurrence of φ and for which we have $x \in V$. A DRS is called* ***closed*** *if for every occurrence of a variable is bound.*

In DRT one assumes that a complete text is interpreted by a closed DRS. This does not have to be the case for the individual sentences, only for every initial segment of the text.

The notion of a structure for the language $\mathfrak{L}$ is already defined. A **valuation** is a function $\beta : V \to M$ where V is the set of $\mathfrak{L}$–variables and M the universe of $\mathfrak{M}$. If β and β' are valuations and W a set of variables then we write $\beta \sim_W \beta'$ if $\beta(x) = \beta'(x)$ for every variable $x \notin W$.

Definition 4.7.4 *Let $\mathfrak{M}$ be a structure for $\mathfrak{L}$ and β a valuation.*

1. $\langle\mathfrak{M}, \beta\rangle \vDash^d \delta$ *if* $\langle\mathfrak{M}, \beta\rangle \vDash \delta$.
2. $\langle\mathfrak{M}, \beta\rangle \vDash^d [V : \Delta]$ *if there is a* β' *with* $\beta \sim_V \beta'$ *and* $\langle\mathfrak{M}, \beta'\rangle \vDash^d \delta$ *for every* $\delta \in \Delta$.
3. $\langle\mathfrak{M}, \beta\rangle \vDash^d \neg\delta$ *if* $\langle\mathfrak{M}, \beta\rangle \nvDash^d \delta$.
4. $\langle\mathfrak{M}, \beta\rangle \vDash^d \delta' \vee \delta''$ *if* $\langle\mathfrak{M}, \beta\rangle \vDash^d \delta'$ *or* $\langle\mathfrak{M}, \beta\rangle \vDash^d \delta''$.
5. $\langle\mathfrak{M}, \beta\rangle \vDash^d [V : \Delta] \Rightarrow [W : \Delta']$ *if for all* β' *with* $\beta \sim_V \beta'$ *and* $\langle\mathfrak{M}, \beta\rangle \vDash^d \Delta$ *there is a* β'' *with* $\beta' \sim_W \beta''$ *and* $\langle\mathfrak{M}, \beta''\rangle \vDash \Delta'$.

In fact, it is easy to give a translation into plain first order logic. For $V = \{x_i : i < n\}$ write $(\exists V)$ in place of $(\exists x_0)(\exists x_1)\ldots(\exists x_{n-1})$. Given Δ write $\Delta^\dagger := \{\delta^\dagger : \delta \in \Delta\}$. The following translation procedure nust be applied outside in:

$$\begin{array}{lcl}
([V : \Delta] \Rightarrow [W : \Delta'])^\dagger & := & (\forall V)[\bigwedge \Delta^\dagger \to (\exists W) \bigwedge \Delta'^\dagger] \\
([V : \Delta])^\dagger & := & (\exists V) \bigwedge \Delta^\dagger \\
(\neg\delta)^\dagger & := & \neg\delta^\dagger \\
(\delta \vee \delta')^\dagger & := & \delta^\dagger \vee \delta'^\dagger
\end{array}$$

Proposition 4.7.5 *For every DRS δ and every model $\langle \mathfrak{M}, \beta \rangle$:*

$$\langle \mathfrak{M}, \beta \rangle \vDash^d \delta \Leftrightarrow \langle \mathfrak{M}, \beta \rangle \vDash \delta^\dagger$$

The proof is left as an exercise. Conversely, predicate logic can be embedded into DRT, which shows that they are actually equally expressive. (Here, ; is used in the same way in proof sequents.)

$$\begin{array}{lcl} R(\vec{x})_\dagger & := & \{R(\vec{x})\} \\ (\neg\varphi)_\dagger & := & \neg\varphi_\dagger \\ (\varphi \wedge \chi)_\dagger & := & \varphi_\dagger; \chi_\dagger \\ ((\exists x)\varphi)_\dagger & := & [\{x\} : \varphi_\dagger] \end{array}$$

Proposition 4.7.6 *For every first-order formula φ and every model $\langle \mathfrak{M}, \beta \rangle$:*

$$\langle \mathfrak{M}, \beta \rangle \vDash \varphi \Leftrightarrow \langle \mathfrak{M}, \beta \rangle \vDash^d \varphi^\dagger$$

Since every formula is a DRS and $\vDash^d$ coincides with the usual notion $\vDash$ we will simply not distinguish the two. We say the DRSs δ and δ' are **equivalent** if for all model $\mathfrak{M}$ and all partial valuations β: $\langle \mathfrak{M}, \beta \rangle \vDash \delta$ if and only if $\langle \mathfrak{M}, \beta \rangle \vDash \delta'$, if and only if $\delta^\dagger \leftrightarrow \delta'^\dagger$ is a theorem of predicate logic. In this case we write $\delta \equiv \delta'$.

Proposition 4.7.7 *We have $\delta \equiv [\varnothing : \delta]$. Further $[V : [V' : \Delta]] \equiv [V \cup V' : \Delta]$.*

From this theorem it immediately follows that a DRS can be made up to equivalence from DRSs of the form $[\{x\} : \varnothing]$ and $[\varnothing : \varphi]$ with the help $\bullet$, $\neg$, $\vee$ and $\Rightarrow$. If $\delta = [V : \Delta]$ and $\delta' = [V' : \Delta']$ DRSs over $\mathfrak{L}$ so is

$$\delta \bullet \delta' := [V \cup V' : \Delta \cup \Delta'] \, .$$

This operation is called the **Zeevat–Merge**. This is not the same as the conjunction. To give an example, let P be a unary predicate, $M := \{a, b\}$ and P are interpreted in $\mathfrak{M}$ by the set $\{a\}$. Then for every β:

$$\langle \mathfrak{M}, \beta \rangle \vDash [\{x\} : P(x)]; [\{x\} : \neg P(x)] \, .$$

But we do not have

$$\langle \mathfrak{M}, \beta \rangle \vDash [\{x\} : P(x); \neg P(x).] \;.$$

We shall now show how DRT treats simple sentences and texts. Let it be said in advance that DRT tries to avoid quantifiers and variable binders. This is not necessary but it can be done in virtue of the expressive strength of DRT. Therefore we do not have bound variables as in Montague Semantics. In place of that, all occurring formulae are open.

$$\begin{array}{c} \mathtt{a}_i \\ \begin{array}{|c|}\hline \{x_i\} \\ \hline \varnothing \\ \hline \end{array} \end{array} \qquad \begin{array}{c} \mathtt{car}_i \\ \begin{array}{|c|}\hline \varnothing \\ \hline \mathsf{car}'(x_i) \\ \hline \end{array} \end{array} \qquad \begin{array}{c} \mathtt{red}_i \\ \begin{array}{|c|}\hline \varnothing \\ \hline \mathsf{red}'(x_i) \\ \hline \end{array} \end{array}$$

$$\begin{array}{c} \mathtt{he}_{\langle i,k \rangle} \\ \begin{array}{|c|}\hline \{x_i\} \\ \hline x_i \doteq x_k \\ \hline \end{array} \end{array} \qquad \begin{array}{c} \mathtt{Paul}_i \\ \begin{array}{|c|}\hline \varnothing \\ \hline x_i \doteq p \\ \hline \end{array} \end{array} \qquad \begin{array}{c} \mathtt{see}_{\langle j,k \rangle} \\ \begin{array}{|c|}\hline \varnothing \\ \hline \mathsf{see}'(x_i, x_k) \\ \hline \end{array} \end{array}$$

These DRSs are to be understood as examples. Notice that we have indexed the DRSs. This is absolutely necessary to guarantee a proper translation. Consider now the sentence

(4.7.7) `Paul sees a red car.`

In order to convert this into a DRS we have to establish the syntactic structure and an indexation. Then we get the following result.

(4.7.8) $[\mathtt{Paul}_0\ [\mathtt{sees}_{\langle 0,1 \rangle}\ [\mathtt{a}_1\ [\mathtt{red}_1\ \mathtt{car}_1]]]]$

The constituent bracket is translated by $\bullet$. Hence we get as a translation of (4.7.8) now (4.7.9)

(4.7.9) $[\{x_0\} : x_0 \doteq p] \bullet ([\varnothing : \mathsf{see}'(x_0, x_1)]$
$\bullet([\{x_1\} : \varnothing] \bullet ([\varnothing : \mathsf{red}'(x_1)] \bullet [\varnothing : \mathsf{car}'(x_1)])))$

$$\begin{array}{|c|}\hline \{x_0, x_1\} \\ \hline x_0 \doteq p; \mathsf{see}'(x_0, x_1); \\ \mathsf{red}'(x_1); \mathsf{car}'(x_1). \\ \hline \end{array}$$

Under certain circumstances the Zeevat–Merge acts as a dynamic conjunction. Call the merge $\delta \bullet \delta'$ **normal** if every variable occurring bound at the main level in δ' does not occur in δ.

Proposition 4.7.8 *Suppose that $\delta \bullet \delta'$ is normal. Then*

$$\delta \bullet \delta' \equiv [\varnothing : \delta; \delta \Rightarrow \delta']]$$

Notice that classically, $\varphi \wedge (\varphi \rightarrow \chi) = \varphi \wedge \chi$, but since we do not disallow accidental capture of variables, we cannot automatically conclude that the theorem holds. The proof is left as an exercise. Now, it is also easily established that every DRS has a normal decomposition.

Proposition 4.7.9 *Up to equivalence, every DRS can be built up from formulae and $[\{x\} : \varnothing]$ ($x \in X$) using normal merge, $\Rightarrow$, $\vee$, and $\neg$. The term that represents this DRS is called a **normal decomposition**.*

Jeroen Groenendijk and Martin Stokhof pointed out in (1991) that the problems that DRT treats could be given a compositional interpretation once the format of the semantics is changed. Basically, rather than viewing propositions as conditions on states, they are viewed as relations between states. In order to motivate this, we shall introduce dynamic logic. The dynamic logics has been developed in the 1980s in order to formally write down propositions over programs and verify them. The basic innovation of dynamic logic consisted in providing a calculus of *actions* in addition to propositions or formulae. This allows to take account for the character of programs and this has also motivated the name for this logic. In between more domains of application for dynamic logic have been found, for example the dynamic semantics. An overview can be found in (Muskens *et al.*, 1997). *Propositional dynamic logic* (PDL) has the following syntax.

Definition 4.7.10 *Let $\Pi := \{\pi_i : i < \kappa\}$ be a set of so called programs and $V := \{p_i : i < \omega\}$ a set of propositional variables. Then $Prg(\Pi)$ and $Fml(\Pi)$ are defined as follows.*

1. *$\pi_i \in Prg(\Pi)$ for all $i < \kappa$.*
2. *If α, β are from $Prg(\Pi)$ then also α^*, $\alpha \cup \beta$ as well as $\alpha;\beta$.*
3. *If $\varphi \in Fml(\Pi)$ then $\varphi? \in Prg(\Pi)$.*
4. *$p_i \in Fml(\Pi)$ for all $i < \kappa$.*
5. *If φ and χ are from $Fml(\Pi)$ so are $\top$, $\varphi \wedge \chi$.*
6. *If $\alpha \in Prg(\Pi)$ and $\chi \in Fml(\Pi)$ then also $[\alpha]\chi \in Fml(\Pi)$.*

We call elements from $Prg(\Pi)$ ***programs*** *and elements from $Fml(\Pi)$* ***formulae****. $\pi \in \Pi$ is called an* ***elementary program****. Programs of the form $\chi?$ are called* ***tests****.*

We agree on a few abbreviations.

$$\begin{array}{lll} \langle\alpha\rangle\chi & := & \neg[\alpha]\neg\chi, \\ \varphi \vee \chi & := & \neg(\neg\varphi \wedge \neg\chi), \\ \varphi \to \chi & := & \neg(\varphi \wedge \neg\chi), \\ \bot & := & \neg\top. \end{array}$$

PDL is an extension of modal logic. It is interpreted in Kripke–structures, as introduced in Section 6.1. The only innovation is the fact that we now have a structure on the set of modalities. Let F be a set $R : \Pi \to \wp(F \times F)$ and $\beta : V \to \wp(F)$. We define by intercalated induction the extensions of these maps.

$$\begin{array}{lll} \overline{R}(\pi_i) & := & R(\pi), \\ \overline{R}(\alpha;\beta) & := & R(\alpha) \circ R(\beta), \\ \overline{R}(\alpha^*) & := & R(\alpha)^*, \\ \overline{R}(\alpha \cup \beta) & := & R(\alpha) \cup R(\beta), \\ \overline{R}(\chi?) & := & \{\langle v, v\rangle : v \in \overline{\beta}(\chi)\}, \\ \overline{\beta}(p_i) & := & \beta(p_i), \\ \overline{\beta}(\top) & := & \wp(F), \\ \overline{\beta}(\neg\varphi) & := & F - \overline{\beta}(\varphi), \\ \overline{\beta}(\varphi \wedge \chi) & := & \overline{\beta}(\varphi) \cap \overline{\beta}(\chi), \\ \overline{\beta}([\alpha]\chi) & := & \{v : \text{ for all } w \in F : \text{ if } v \; R(\alpha) \; w \text{ then } w \in \overline{\beta}(\chi)\}. \end{array}$$

$\langle F, R, \beta \rangle$ is called a **Kripke–model**. Finally, we write $\langle F, R, \beta, x \rangle \models \chi$ if $x \in \overline{\beta}(\chi)$. A program therefore is a relation which connects one world with another. In dynamic logic programs are more like action schemata which construct subsequent states. Then it may be that there is not a subsequent state (one then says the program *fails*) or it may happen that there are several successor states (one speaks of *nondeterministic choice*). Typically, programs are deterministic, which means that they always yield the same value for the same input. We write $v \xrightarrow{\alpha} w$ if $v\ R(\alpha)\ w$. Further we write $v \xrightarrow{\alpha} \surd$ if there is a $w \in F$ such $v \xrightarrow{\alpha} w$. We have $v \xrightarrow{\alpha;\beta} w$ if and only if there is a $u \in F$ with $v \xrightarrow{\alpha} u \xrightarrow{\beta} w$. Further $v \xrightarrow{\alpha\cup\beta} w$ if and only if $v \xrightarrow{\alpha} w$ or $v \xrightarrow{\beta} w$ and $v \xrightarrow{\alpha^*} w$ if and only if there is a sequence u_i, $i < n$, such that $u_0 = v$, $u_n = w$ and $u_i \xrightarrow{\alpha} u_{i+1}$ for all $i < n$. (If $n = 0$, we must have $u = v$.) Finally, we have $v \xrightarrow{\chi?} w$ if and only if χ is satisfied at v and $w = v$.

We remark that some junctors can be defined by others. Namely we have the following in any Kripke–model.

$$\begin{array}{lcl}
[\alpha \cup \beta]\chi & \leftrightarrow & [\alpha]\chi \wedge [\beta]\chi \\
[\alpha; \beta]\chi & \leftrightarrow & [\alpha][\beta]\chi \\
\chi & \leftrightarrow & \langle \chi? \rangle \top \\
\varphi \to \chi & \leftrightarrow & [\varphi?]\chi
\end{array}$$

Dynamic logic allows to define certain basic constructions of the usual programming languages, the so called *loops*.

$$\begin{array}{lcl}
\mathsf{while}\ \chi\ \mathsf{do}\ \alpha & := & (\chi?; \alpha)^*; (\neg\chi?) \\
\mathsf{until}\ \chi\ \mathsf{do}\ \alpha & := & (\neg\chi?; \alpha)^*; \chi?
\end{array}$$

Both constructs are programs. Let $\beta := \mathsf{while}\ \chi\ \mathsf{do}\ \alpha$. Then $v \xrightarrow{\beta} w$ if and only if w can be reached from v by iterating α as long as χ is satisfied. Since χ need not hold at w the subsequent test on $\neg\chi$ is necessary. Analogously for until.

There exists also **quantified dynamic logic** (QDL). It is different insofar as it assumes instead of the apparatus of propositional logic the standard language of predicate logic. However, there is one exception: there are no quantifiers. Instead, for every variable x_i there is a program called **random reset of** x_i, denoted by $x_i \to ?$. The structures are the structure of predicate logic. A model is a triple $\langle M, \mathfrak{I}, \beta\rangle$, where $\langle M, \mathfrak{I}\rangle$ is a (single) first order structure and β an assignment. (For the benefit of the reader we deal only with assignments, that is, total functions.) We add to the definition of $\overline{R}$ the following clause.

$$\overline{R}(x_i \leftarrow ?) \quad := \quad \{\langle \beta, \beta'\rangle : \beta \sim_x \beta'\}$$

Furthermore, we assume for atomic formulae the following.

$$\langle M, \mathfrak{I}, \beta\rangle \models R(\vec{x}) \quad :\Leftrightarrow \quad \langle \beta(x_i) : i < \Xi(R)\rangle \in \mathfrak{I}(R)$$

The clauses for $\neg$, $\vee$ and the other propositional connectives are entirely standard. Terms are evaluated in the usual manner, using the valuation: $[t]^\beta$ is defined inductively. Now observe the following.

$$\langle M, \mathfrak{I}, \beta\rangle \models \langle x_i \leftarrow ?\rangle\varphi \text{ if and only if there is } \beta' \sim_{x_i} \beta : \langle M, \mathfrak{I}, \beta'\rangle \models \varphi$$

Thus, $\langle x_i \leftarrow ?\rangle\varphi$ behaves technically like $\exists x_i.\varphi$. Both are true in the same models. In this way, standard predicate logic can be translated into dynamic predicate logic. However, let us now resume the discussion on DRT. We shall introduce a relational interpretation of the connectives in place of the usual definition as sets of assignments. PDL offers a natural way of doing this in terms of actions. Here is a translation from normal decompositions.

$$\begin{array}{lcl}
(R(\vec{x}))^\nabla & := & \varphi? \\
([\{x\} : \varnothing])^\nabla & := & x \to ? \\
(\delta \bullet \delta')^\nabla & := & \delta^\nabla ; \delta'^\nabla \\
(\delta \Rightarrow \delta')^\nabla & := & [\delta^\nabla]\langle \delta'^\nabla\rangle \top ? \\
(\delta \vee \delta')^\nabla & := & \delta^\nabla \cup \delta'^\nabla
\end{array}$$

Theorem 4.7.11 *Let δ be a DRS and $\langle \mathfrak{M}, \beta\rangle$ a model. Then*

$$\langle \mathfrak{M}, \beta\rangle \models^d \delta \quad \Leftrightarrow \quad \langle \mathfrak{M}, \beta\rangle \models \langle \delta^\nabla\rangle \top$$

We now turn to the question of the choice of variables. As said above, DRT does not choose the indices, it assumes them to be given beforehand. To remedy the situation, Kees Vermeulen (1995) introduced what he calls **Referent Systems**. Let $Var := \{0,1\}^+$. (For technical convenience, ε is excluded.)

Definition 4.7.12 *Let N be a set. A referent system over N is a triple $\mathfrak{R} = \langle I, R, E\rangle$, where $R \subseteq Var$ is called the set of **referents**, I is a partial injective function from N to R, called the **import function**, and E a partial injective function from R to N, called the **export function**. N is the set of **names**.*

If $E(r) = A$ we say that $\mathfrak{R}$ **exports** r **under the name** A, and if $I(B) = r$ we say that $\mathfrak{R}$ **imports** r **under the name** B. It is not required that $A = B$! Meaning units are pairs $\mathfrak{C} = \langle \mathfrak{R}, \Phi\rangle$ where $\mathfrak{R} = \langle I, R, E\rangle$ is a referent system over some appropriate set of names and Φ a set of formulae using only the referents from R. Let $\langle D, I\rangle$ be a model. An **assignment** is a map from a set of referents into D. $\langle \mathfrak{R}, \Phi\rangle$ is **satisfied** in a model under the assignment h if h assigns a value to each referent from $\mathfrak{R}$ and all formulae from Φ are true under h. Hence, the satisfaction clauses are pretty much those of DRT. The consequence is that the renaming of referents does not change satisfiability in a model.

Let $\mathfrak{C}_i = \langle \mathfrak{R}_i, \Phi_i\rangle$, $i < 2$, be meaning units. The merge $\mathfrak{C}_0 \circ \mathfrak{C}_1$ is defined as follows. We define the merge $\mathfrak{R}_2 := \mathfrak{R}_0 \bullet \mathfrak{R}_1$ of referent systems plus injective functions $\iota_0 : R_0 \to R_2$, $\iota_1 : R_0 \to R_2$, and then put

$$\mathfrak{C}_0 \circ \mathfrak{C}_1 := \langle \mathfrak{R}_2, \iota_0[\Phi_0] \cup \iota_1[\Phi_1]\rangle$$

We are left with a definition of the merge. We say first of all given two referents $r \in R_0$ and $s \in R_1$ that r **supervenes** s if $I_1(E_0(r)) = s$. Supervenience is a relation $\subseteq R_0 \times R_1$. Let U be the set of supervened elements of R_1. Write r^0 in place of $r \cdot 0$ and

r^1 in place of $r \cdot 1$. Then we put

$$
\begin{array}{lcl}
R_2 & := & (R_0 \times \{0\} \cup R_1 \times \{1\}) - U \times \{1\} \\
\iota_0(r) & := & r^0 \\
\iota_1(s) & := & \begin{cases} r^0 & \text{if } s \text{ is supervened by } r, \\ s^1 & \text{otherwise.} \end{cases}
\end{array}
$$

Say that r **I–preempts** s if there is an $A \in N$ such that $I_0(A) = r$ and $I_1(A) = s$; and that s **E–preempts** r if $E_1(s) = E_0(r)$. (Notice that r can both I–preempt and supervene s.) The import and export functions are now as follows.

$$
I_2(A) := \begin{cases} r^0 & \text{if } I_0(A) = r, \\ s^1 & \text{if } s \text{ is not I–preempted and } I_1(A) = s, \\ \star & \text{otherwise.} \end{cases}
$$

$$
E_2(u) := \begin{cases} E_1(s) & \text{if } u = t^0 \text{ and } t \text{ supervenes } s, \\ E_1(s) & \text{if } u = s^1 \text{ and } s \text{ is not superverned,} \\ E_0(r) & \text{if } u = r^0, E_1 \text{ is defined on } r \text{ and is not E–preempted,} \\ \star & \text{otherwise.} \end{cases}
$$

Table 4.2 shows the merge of referent systems containing just one referent. We write $[A : r : B]$ if r is a referent that is imported under A and exported under the name B. We write $[- : r : B]$ if r has no import name, and similarly $[A : r : -]$ and $[- : r : -]$. The table does not show the effect of the preemption. It can happen that two referents compete for the same import (export) name. In that case they must be from different referent systems (since both are functions and injective). Then the referent from the first system wins the import name; if they compete for the export name, the one from the second system wins.

With the referent systems the Latin examples can be accounted for; as names we choose the cases, and the verbs and prepositions are simply referent systems importing referents under certain names, while inflected nouns export referents under a given name.

Table 4.2: The Merge of Referent Systems

$$\begin{array}{lcccl}
[A : r : B] & \bullet & [B : s : C] & = & [A : r^0 : C] \\
[- : r : B] & \bullet & [B : s : C] & = & [- : r^0 : C] \\
[A : r : B] & \bullet & [B : s : -] & = & [A : r^0 : -] \\
[- : r : B] & \bullet & [B : s : -] & = & [- : r^0 : -]
\end{array}$$

For example, Latin *horto* and *magno* are translated by

$$\begin{array}{c} \texttt{/horto/} \\ \begin{array}{|l|}\hline [- : 0 : \mathrm{abl}] \\ \hline \mathsf{garden}'(0) \\ \hline \end{array} \end{array} \qquad \begin{array}{c} \texttt{/magno/} \\ \begin{array}{|l|}\hline [\mathrm{abl} : 0 : \mathrm{abl}] \\ \hline \mathsf{big}'(0) \\ \hline \end{array} \end{array}$$

Their merge is — according to the definition above —:

$$\begin{array}{c} \texttt{/horto magno/} \\ \begin{array}{|l|}\hline [- : 00 : \mathrm{abl}] \\ \hline \mathsf{garden}'(00) \\ \mathsf{big}'(00) \\ \hline \end{array} \end{array}$$

This is the translation of `horto magno`. Notice that `magno horto` would in this system not get the right translation. We need to assume in fact that lexical entries are associated with sets of referent systems, thereby allowing for different word order (and the fact that both `magno` and `horto` can also be dative). Obviously, this model is very simplistic, but it shows how one can incorporate morphological information about the syntactic structure into the semantics.

How would referent systems handle our examples (1) and (2)? We will present a solution, which is based on the following insight. Pronouns pick up their referent not by an index (such an index is simply not part of the language) but rather by the information that is resident in the semantics of the antecedent, the gender of the pronoun and more. To make matters simple, we assume that we only use gender information. Therefore the set of names consists of combinations of gender and case. However, one or both of gender and case can be absent, and this is represented by $\star$. Hence the set of names is as follows:

$$\{\text{M}, \text{F}, \text{N}, \star\} \times \{\text{N}, \text{A}, \star\}$$

(Here, N abbreviates *nominative*, A *accusative*, and F, M and N are the genders.) A particular pair is written like a vector, for example (M, A). Notice that $(\text{M}, \star)$ is a name in the technical sense, likewise $(\star, \star)$ and $(\star, \text{A})$. The cases will be used to steer the syntactic translation, and the gender is used to get at the binding. To make the whole thing work we have to play with the $\star$ to switch the assignment of gender and case on and off. For example, the pronouns `he` and `him` have the following semantics

/`he`/	/`him`/
$[(\text{M}, \star) : x : (\text{M}, \text{N})]$	$[(\text{M}, \text{A}) : x : (\text{M}, \star)]$
$\varnothing$	$\varnothing$

The difference is that `he` has no case to the left and nominative to the right, while `him` has accusative to the left and no case to the right. The semantics of `fat`, `man` (nominative) and `man` (accusative) are as follows:

/`fat-`γ/	/`man-`NOM/	/`man-`ACC/
$[(\alpha, \gamma) : 0 : (\alpha, \gamma)]$	$[(\text{M}, \star) : x : (\text{M}, \text{N})]$	$[(\text{M}, \text{A}) : 0 : (\text{M}, \star)]$
$\mathsf{fat}'(0)$	$\mathsf{man}'(0)$	$\mathsf{man}'(0)$

Here the variable is instantiated to any appropriate value (in this case, genders or $\star$ for α and case or $\star$ for γ). This is an extensions of the original proposal; what we argue is that the variables are part of the lexical representation and get instantiated after the representation has been inserted into the structure. The indefinite determiner $\mathsf{a(n)}$ wipes out the gender to the left. It has the semantics

$$\begin{array}{c} /\mathsf{a}\text{-}\gamma/ \\ \begin{array}{|l|} \hline [(\star,\gamma) : 0 : (\alpha,\gamma)] \\ \hline \varnothing \\ \hline \end{array} \end{array}$$

Finally, the transitive verb `see` has this entry

$$\begin{array}{c} /\mathsf{see}/ \\ \begin{array}{|l|} \hline [(\alpha,\textsc{n}) : 0 : (\alpha,\star)] \\ [(\beta,\star) : 1 : (\beta,\textsc{a})] \\ \hline \mathsf{see}'(0,1) \\ \hline \end{array} \end{array}$$

We now take the following sentence, with the bracketing (plus case assignment) given by the syntax.

(4.7.10) `Kim (sees (a fat man (at the metro entrance))).`
`He (is selling souvenirs).`

We leave the phrases 'at the metro entrance' and 'is selling souvenirs' unanalyzed. Putting together the object noun phrase and renaming the referents suitably gives

$$\begin{array}{|l|} \hline [(\star,\textsc{a}) : 0 : (\textsc{m},\star)] \\ \hline \mathsf{fat}(x), \mathsf{man}'(0) \\ \mathsf{at\text{–}metro\text{–}entrance}'(0) \\ \hline \end{array}$$

So the first sentence is translated as shown in Table 4.3. Here, α is instantiated to F and β to M. $\simeq$ means that the structures are equal only after some renaming of referents. The point is now that the referents are open to anaphoric reference, but are syntactically inert since they carry the case description $\star$. Subsequently they can only be picked up by a pronoun. For example, we can merge with

$$\begin{array}{|l|} \hline [(\textsc{m},\star) : 0 : (\textsc{m},\star)] \\ \hline \mathsf{sell\text{–}souvenirs}'(0) \\ \hline \end{array}$$

Table 4.3: Merge Using Referent Systems

$$
\begin{array}{|l|}\hline
[(\textsc{f},\star) : 0 : (\textsc{f},\textsc{n})] \\ \hline
0 \doteq k \\ \hline
\end{array}
\quad \circ \quad
\left(
\begin{array}{|l|}\hline
[(\alpha,\textsc{n}) : 0 : (\alpha,\star)] \\
[(\beta,\star) : 1 : (\beta,\textsc{a})] \\ \hline
\mathsf{see}'(0,1) \\ \hline
\end{array}
\right.
$$

$$
\circ \quad
\left.
\begin{array}{|l|}\hline
[(\star,\textsc{a}) : 0 : (\textsc{m},\star)] \\ \hline
\mathsf{fat}'(0); \mathsf{man}'(0); \\
\mathsf{at\text{–}metro\text{–}entrance}'(0). \\ \hline
\end{array}
\right)
$$

$$
\simeq \quad
\begin{array}{|l|}\hline
[(\textsc{f},\star) : 0 : (\textsc{f},\star)] \\
[(\textsc{m},\star) : 1 : (\textsc{m},\star)] \\ \hline
0 \doteq k; \mathsf{fat}(1); \mathsf{man}'(1); \\
\mathsf{at\text{–}metro\text{–}entrance}'(1); \\
\mathsf{see}'(0,1). \\ \hline
\end{array}
$$

This gives

$$
\begin{array}{|l|}\hline
[(\textsc{f},\star) : 0 : (\textsc{f},\star)] \\
[(\textsc{m},\star) : 1 : (\textsc{m},\star)] \\ \hline
0 \doteq k; \mathsf{fat}'(1), \mathsf{man}'(1); \\
\mathsf{at\text{–}metro\text{–}entrance}'(1); \\
\mathsf{see}'(0,1); \\
\mathsf{sell\text{–}souvenirs}'(1). \\ \hline
\end{array}
$$

In order to be able to judge the success and failure of referent systems, compare the result with a slightly different sentence.

(4.7.11) `Paul sees a fat man at the metro entrance.`
`He is selling souvenirs.`

What will happen is that the phrase 'a fat man at the metro entrance' will get a referent that is different from that for Paul. This is due to the fact that the determiner blocks the gender to the left. But the referent also E–preempts the referent for Paul since they export the same name before merge, $(\text{M}, \star)$. The pronoun 'he' can therefore not refer back to Paul. Hence referent systems do not handle the facts correctly. In this case it is because the space of names is too small to make enough distinctions. However, the fact that anaphoric reference is blocked by antecedents which are less distant is not so far off the mark. This is a topic that deserves attention.

Exercise 161. Prove Proposition 4.7.5.

Exercise 162. Show Proposition 4.7.8.

Exercise 163. Let $\bot$ be a DRS that is never satisfiable. Show that $\vee$ and $\neg$ can be defined in terms of $\Rightarrow$ and $\bot$ (modulo $\equiv$).

Exercise 164. Show that $\delta \equiv \top \Rightarrow \delta$, but that in general $\delta \bullet \delta' \not\equiv (\top \Rightarrow \delta) \bullet \delta'$. ($\top \Rightarrow \delta$ functions a bit like the 'existential closure' of δ in predicate logic though it does not use any quantification. We prefer to call the map $\delta \mapsto \top \Rightarrow \delta$ 'existential blindfolding'.)

4.8 Translating Meaning to Form

In this section we shall be concerned with the various ways in which languages express a given meaning. This will be useful in several ways. First, it illustrates how diverse human languages are and what kind of typological differentiation there is. Second, it shows us how complex the algorithms for mediating form and meaning really are in living languages. The reader will therefore get an idea of why linguistic theories are so incredibly complicated and diverse. Third, we shall see how the present analysis of languages as semiotic systems fits vis a vis other theoretical frameworks.

We shall be concerned with the translation between form and meaning. However, if by meaning we understand simply model–theoretic meaning this task is at present impossible. Say we have a language using certain basic properties of individuals P_i, $i < m$, which are interpreted as subsets of the universe M. Boolean combinations are likewise interpreted as subsets of M. Now given an arbitary subset of M, which expression will be used to refer to this set? If the set is finite there seem to be tools available that make this possible, but if the set is infinite then this in turn depends on the mode in which that set is presented. (For example, suppose that the set is described by a recursive procedure, naming the first two elements, say 1 and 2, and then saying that the next element is the sum of all the previous elements. Would you have guessed that this is the set $\{1, 2\} \cup \{3 \cdot 2^n : n \in \omega\}$?) There is all reason to believe that this problem is not the one we should attempt to solve. Rather, given that we have a language (say, some language of simple type theory) in which we express meanings, then we should describe a procedure that takes elements of that language and translates them into expressions of the language in question. Thus, it may turn out that t and u are synomous terms but their translations are nevertheless different (but again synonymous if translation is biunique). This is the problem we shall attack here, progressing from simple to complex.

First step. $F = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, `a` and `b` have the same type, e, `c` has the type $e \to t$ (intransitive verb) and `d` the type $e \to (e \to t)$ (transitive verb). The wellformed expressions of this language are `ca`, `cb`, `daa`, `dab`, `dba` and `dbb`. Call the language L. We shall now study the ways in which these expressions are rendered in natural languages. The first observation we make is that languages differ in the relative order of the elements. English, for example, puts the subject before the verb, the object after. Thus, if `a` is translated by `Marcus`, `b` by `Paul`, `c` by `sleeps` and `d` by `chases`, we get the following correspondence.

L	English
`ca`	`Marcus sleeps.`
`cb`	`Paul sleeps.`
`daa`	`Marcus chases Marcus.`
`dab`	`Marcus chases Paul.`
`dba`	`Paul chases Marcus.`
`dbb`	`Paul chases Paul.`

In this way, we can highlight the different word order patterns. For English, we may postulate the following basic signs in an **AB**–grammar.

$$\langle \texttt{Marcus}, e, \texttt{a}\rangle, \langle \texttt{Paul}, e, \texttt{b}\rangle, \langle \texttt{sleeps}, e/t, \texttt{c}\rangle, \langle \texttt{chases}, e/(e\backslash t), \texttt{d}\rangle$$

Other languages have different word order patterns, as we have described earlier. However, more is in stock. Many languages use *cases* to distinguish the arguments. Hungarian, for example, marks the object of a transitive verb by a suffix `t`.

L	Hungarian
`ca`	`Marcusz alszik. Alszik Marcusz.`
`ca`	`Pál alszik. Alsik Pál.`
`daa`	`Marcusz látja Marcuszt. Marcusz Marcuszt látja. ...`
`dab`	`Marcusz látja Pált. Marcusz Pált látja. ...`

Word order is irrelevant (in Hungarian it is constrained by other factors, so is not completely free). For the purpose of translating back from the Hungarian sentence into the term, however, word order is actually irrelevant, unlike in English. Suppose for the sake of the argument that any permutation of the sentences on the right hand side is a grammatical sentence of Hungarian whose meaning is the one given on the left. Then translation is still biunique. `Látja Pált Marcusz.` for example has meaning `dba`. All we have to do is to look for the word carrying the suffix `t`. This word is the object.

Additionally, Hungarian employs what is known as **head marking**. Depending on whether the subject is singular or plural, and depending on whether it is 1st, 2nd or 3rd person the verb form is different (see also Section 6.1).

(Subject)	English	Hungarian
1 Sg	`I sleep.`	`Alszom.`
2 Sg	`You sleep.`	`Alszod.`
3 Sg	`He/She/It sleeps.`	`Alszik.`
1 Pl	`We sleep.`	`Alszunk.`
2 Pl	`You sleep.`	`Alszatok.`
3 Pl	`They sleep.`	`Alszanak.`

We say that in Hungarian the verb agrees in number and person with its subject. English has traces of that phenomenon: if the subject is in the first person singular the verb carries an additional suffix `s`. The reason why this is called head marking is that this time it is the head (the verb) that is marked for properties of its argument. Some languages not only have agreement with the subject, they also have agreement with the object, some even with the indirect object. We remark here that this phenomenon is not restricted to verbs. A particularly widespread phenomenon is the so–called **possessor agreement**. Once again we can use Hungarian to exemplify it.

(Possessor)	English	Hungarian
1 Sg	`my car`	`az én kocsim`
2 Sg	`your car`	`a te kocsid`
3 Sg	`his/her/its car`	`az ö kocsija`
1 Pl	`our car`	`a mi kocsink`
2 Pl	`your car`	`a ti kocsitek`
3 Pl	`their car`	`az ö kocsijuk`

In German, the possessive pronouns distinguish the gender of the possessor (in 3rd person) as well as number, gender and case of the possessed item. For example, it is `sein Auto` for `his car` but `ihr Auto` for `her car`. Put into the dative this would be `seinem Auto` and `ihrem Auto`, and `seinen Autos/ihren Autos` in the plural (and dative). Also, adjectives in many languages agree with the noun they modify in case, number, person and other categories. English possesses number agreement with respect to the demonstratives `this` and `that`. We have `this city` but `these` `cities`, and we have `that cities` but `those cities`.

For the rest of this section we shall sketch the advantages and disadvantages of the various types of languages, and then turn to a formal analysis of case marking. Our discussion of the various representations of the terms concerns the question whether the representations are uniquely readable. Obviously, if they are not this is a source of potential confusion in communication. In logic, Polish Notation has been popular for a while since it is very economical: once one is used to it, it has many advantages. One is of course, that it does away with brackets. This means less work in writing or typing formulae. On the face of it it seems that any consistent way of putting the function symbol with respect to its arguments is the same as any other. This, however, is false. Suppose we write $\neg$ before its argument and $\Box$ after. Then $\neg\mathtt{p0}\Box$ is ambiguous between $\mathtt{(}\Box\mathtt{(}\neg\mathtt{p0))}$ and $\mathtt{(}\neg\mathtt{(}\Box\mathtt{p0))}$. Thus, we must be consistent in either putting every functor before its arguments or every functor following them. On the other hand, if this fixed, we can specify for each functor independently in which way the arguments have to be written down. For example, we may require that $\rightarrow$ takes its first argument second, and the second argument first. As long as do not additionally employ the standard notation this remains uniquely readable. Evidently, if $\rightarrow \varphi\chi$ is actually taken to mean what $\rightarrow \chi\varphi$ means we had better use a different symbol, say $\leftarrow$. And if so, it is clear that we might just as well declare the argument that appears first to be the first argument *tout court*. Trivial as this point may seem, it allows to see that nothing much distinguishes a VSO language from a VOS language. However, one of the most famous linguistic universals in language typology is the claim that the subject precedes the object (Greenberg's Universal No. 1), thus favouring VSO, SVO and SOV. From the figures given in Section1.4 we conclude that this is the case in 19 out of 20 languages. We shall not elabourate on why this is so. We mention only that theoretically SVO languages are at a disadvantage since they are not uniquely readable. However, other factors come into play. First, the immediate constituents of the verb typically are noun phrases, and their structures can be much different so that it is not immediate to what degree the ambiguity is real.

Chapter 5

PTIME Languages

5.1 Mildly–Context Sensitive Languages

The introduction of the Chomsky hierarchy has sparked off a lot of research into the complexity of formal and natural languages. Chomsky's own position was that language was not even of Type 1. In transformational grammar, heavy use of context sensitivity and deletion has been made. However, Chomsky insisted that these grammars were not actually models of performance, neither of sentence production nor of analysis; they were just models of competence. They were theories of language or of languages, couched in algorithmic terms. In the next chapter we shall study a different type of theory, based on axiomatic descriptions of structures. Here we shall remain with the algorithmic approach. If Chomsky is right, the complexity of the generated languages is only of peripheral interest and, moreover, cannot even be established by looking at the strings of the language. To do that, we actually need three things:

▷ A theory of the human language(s).

▷ A theory of human sentence production.

▷ A theory of human sentence analysis (and understanding).

Without going into much detail, this is explained as follows. The reason that a language may fail to show its complexity in speech or writing is that humans simply are not able to produce them, even though given further means they would be. The same goes for analysis. Certain sentences might be avoided not because they are illegitimate but because they are misleading or too difficult to understand. A useful analogy is that of a computer. A computer is thought to be able to understand every program of a given computer language if it has been endowed with an understanding of the syntactic primitives and knows how to translate them into executable routines. Yet, some programs may simply be too large for the computer to be translated let alone executed. This may be remedied by giving it a bigger memory (to store the program) or a bigger processing unit (to be able to execute it). None of the upgrading operations, however, seem to touch on the basic ability of the computer to understand the language: the translation or compilation program usually remains the same. Some people have advanced the thesis that certain monkeys possess the symbolic skills of humans but since they cannot handle recursion, their ability to use language is restricted to single clauses (consisting of not more than four words).

One should be aware of the fact that the average complexity of spoken language is linear ($= O(n)$) for humans. We understand sentences as they are uttered, and typically we seem to be able to follow the structure and message word by word. To conclude that therefore human languages must be regular is premature. For it is not clear what processing device the human brain is. Suppose that it is a finite state automaton. Then the conclusion is trivially true. However, if it is a pushdown automaton, the language can be deterministically context free. More complex devices can be imagined giving rise to even larger classes of languages that can be parsed in linear time. This is so since it is not clear that what is one step for the human brain also is one step for, say, a Turing machine. It is known that the human brain works with massive use of parallelism.

Therefore, the problem with the line of approach advocated by Chomsky is that we do not possess a reliable theory of human sentence processing let alone of sentence production (see (Levelt, 1991) for an overview of the latter). Without them, however, it is impossible to assess the correctness of any proposed theory of grammar. Many people have therefore ignored this division of labour into three faculties (however reasonable that may appear) and tried to assess the complexity of the language as we see it. Thus let us ask once more:

> How complex is human language (are human languages)?

While the Chomsky hierarchy has suggested measuring complexity in terms of properties of rules, it is not without interest to try to capture its complexity in terms of resources (time and space complexity). The first suggestion that comes to mind is this.

> Human languages are in **PTIME**.

In computer science, **PTIME** problems are also called *tractable*, since the time consumption grows slowly. On the other hand, **EXPTIME** problems are called *intractable*. Their time consumption grows too fast. In between the two lie the classes **NP** and **PSPACE**. Still today it is not known whether or not **NPTIME** is contained in (and hence equal to) **PTIME**. **NPTIME**–complete problems usually do possess algorithms that run (deterministically) in polynomial time on the average.

The term 'mildly context sensitive' has been introduced in the mid 80s by Aravind Joshi (see (Joshi, 1985)). Mildly context sensitive languages are characterised as follows.

1. Every context free language is mildly context sensitive. There are mildly context sensitive languages which are not context free.

2. Mildly context sensitive languages can be recognized in deterministic polynomial time.

3. There is only a finite number of crossed dependency types.

4. Mildly context sensitive languages have the constant growth property. (L has the **constant growth property** if $\{|\vec{x}| : \vec{x} \in L\}$ contains a linear subset.)

These conditions are not very strong except for the second. It implies that the mildly context sensitive languages form a proper subset of the context sensitive languages. The first condition needs no comment. The fourth is quite weak. All it says is that for every language L there is a number c_L such that for every $\vec{x} \in L$ there is a $\vec{y} \in L$ such that $|\vec{y}| \leq |\vec{x}| + c_L$. It has been tried to strengthen this condition, for example by saying that every intersection of the language with some subalphabet, or every intersection with a semilinear language must have this property. However, these conditions cannot be motivated from linguistics. Similarly for the third condition. What exactly is a crossed dependency type? In this chapter we shall study grammars in which the notion of structure can be defined as with context free languages. Constituents are certain subsets of disjoint (occurrences of) subwords. If this definition is accepted, the third condition can be interpreted as follows: there is a number n such that a constituent has no more than n parts. This is certainly not what Joshi had in mind when formulating his conditions, but it is certainly not easy to come up with a definition that is better than this one and as clear.

So, the conditions are problematic with the exception of the second. Notice that it automatically implies the first, and, as we shall see, also the third (if only weak equivalence counts here). The last one goes out of the window, but it seems that there are good reasons that it has to anyway. In order not to create confusion we shall call a language a **PTIME** language if it has a deterministic polynomial time recognition algorithm (see Definition 1.7.18). In general, we shall also say that a function $f : A^* \to B^*$ is in **PTIME** if there is a deterministic Turing machine which computes that function. Almost all languages that we have considered so far are **PTIME** languages. This shall emerge from the theorems that we shall prove further below.

Proposition 5.1.1 *Every context free language is in **PTIME**.*

This is a direct consequence of Theorem 2.3.24. However, we get more than this.

Proposition 5.1.2 *Let A be a finite alphabet and L_1, L_2 languages over A. If $L_1, L_2 \in$ **PTIME** then so is $A^* - L_1$, $L_1 \cap L_2$ and $L_1 \cup L_2$.*

The proof of this theorem is very simple and left as an exercise. So we get that the intersection of context free languages, for example $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$, are **PTIME** languages. Condition 1 for mildly context sensitive languages is satisfied by the class of **PTIME** languages. Further, we shall show that the full preimage of a **PTIME** language under the Parikh–map is again a **PTIME** language. To this end we shall identify $M(A)$ with the set of all strings of the form $\prod_{i<n} \mathtt{a}_i^{p_i}$. The Parikh–map is identified with the following function $\pi : A^* \to A^*$. Given the string $\vec{x}$, it returns the unique string $\vec{\mathtt{a}}_0^{p_0} \cdot \vec{\mathtt{a}}_1^{p_1} \cdot \ldots \cdot \mathtt{a}_{n-1}^{p_{n-1}}$ for which p_j is the number of occurrences of $\mathtt{a}_j$ in $\vec{x}$. Now take an arbitrary polynomial time computable function $g : A^* \to 2$. Clearly, $g \upharpoonright M(A)$ is also in **PTIME**. The preimage of 1 under this function is contained in the image of π. $g^{-1}(1) \cap M(A)$ can be thought of in a natural way as a subset of $M(A)$.

Theorem 5.1.3 *Let $L \subseteq M(A)$ be in **PTIME**. Then the full preimage of L under π, $\pi^{-1}[L]$, also is in **PTIME**. In particular all preimages of semilinear languages are in **PTIME**.*

The reader is warned that there nevertheless are semilinear languages which are not in **PTIME**. For **PTIME** is countable, but there are uncountably many semilinear languages (see Exercise 2.5).

Theorem 5.1.3 is the direct consequence of the following theorem, which is nevertheless easy to show.

Theorem 5.1.4 *Let $f : B^* \to A^*$ be in **PTIME** and $L \subseteq A^*$ in **PTIME**. Then $M := f^{-1}[S]$ also is in **PTIME**.*

Proof. By definition $\chi_L \in$ **PTIME**. Then $\chi_M = \chi_L \circ f \in$ **PTIME**. This is the characteristic function of L. $\square$

Another requirement which mildly context sensitive languages should satisfy was the constant growth property. It means intuitively speaking that the set $\{|\vec{x}| : \vec{x} \in L\} \subseteq \omega$ does not have arbitrarily large gaps. We leave it to the reader to show that every semilinear language has the constant growth property but that there are languages which have the constant growth property without being semilinear.

We have introduced the Polish Notation for terms in Section 1.2. Here we shall introduce a somewhat exotic method for writing down terms, which has been motivated by the study of certain Australian languages (see (Ebert and Kracht, 2000)). Let $\langle F, \Omega\rangle$ be a finite signature. Further let $\Omega := \max\{\Omega(f) : f \in \Omega\}$. Inductively, we assign to every term t a set $M(t) \subseteq (F \cup \{\mathtt{0}, \mathtt{1}, \dots, \Omega - 1\})^*$:

1. If $t = f$ with $\Omega(f) = 0$ then put $M(f) := \{f\}$.
2. If $t = f(s_0, \dots, s_{\Omega(f)-1})$ then put

$$M(t) := \{t\} \cup \bigcup_{i<\Omega(f)} \{\vec{x} \cdot i : \vec{x} \in M(s_i)\}$$

An element of $M(t)$ is a product $f \cdot \vec{y}$, where $f \in F$ and $\vec{y} \in \Omega^*$. We call f the **main symbol** and $\vec{y}$ its **key**. Now we say that $\vec{y}$ is an **A–form** of t if $\vec{y}$ is the product of the elements of $M(t)$ in an arbitrarily chosen (nonrepeating) sequence, with a boundary marker interspersed. Let `#` be neither in F nor identical to any of the numbers `0`, `1`, ..., $\Omega - 1$. Let $t :=$ `(((x + a) - y) + (z - c))`. Then

$$M(t) = \{\mathtt{+}, \mathtt{-0}, \mathtt{-1}, \mathtt{+00}, \mathtt{x000}, \mathtt{a001}, \mathtt{y01}, \mathtt{z10}, \mathtt{c11}\}.$$

Hence the following string is an A–form of t:

`c11#z10#+00#-0#-1#y01#x000#a001#+`

Theorem 5.1.5 *Let $\langle F, \Omega\rangle$ be a finite signature and L the language of A–forms of terms of this signature. Then L is in* ***PTIME***.

Proof. There is a method of calculating the set M from an A–form $\vec{x}$. One simply has to segment $\vec{x}$ into correctly formed parts. These parts are maximal sequences consisting of a main symbol and a key, which we shall now simply call **parts**. The segmentation into parts is always possible and unique regardless of $\vec{x}$. Now we begin the construction of t. t will be given in Polish Notation. We list on a tape the strings in Ω^* in lexicographic order. They enumerate our keys. We begin with ε. We search in M for an element with key ε and write it on the output tape. Let it be f. We delete the string $f = f \cdot \varepsilon$ on the input. If $\Omega(f) = 0$, the input tape must now be empty. If not, $\vec{x}$ is not an A–form. If $\Omega(f) > 0$, we move on to the key $\mathtt{0}$ and look for the part with key $\mathtt{0}$. If it is not unique, the string is not an A–form. If it is, let it be $g\mathtt{0}$. We delete that substring on the input tape and append g on the output. If $\Omega(g) = 0$, there can be no part with key $\mathtt{00}$. Otherwise we do not have an A–form. We continue in this fashion. Each time a part $f\vec{\alpha}$ has been identified and has a key that we were looking for, then this string is deleted and f added onto the output. We have to monitor for the uniqueness of keys and for the fact that all keys will eventually be deleted. In fact, we can do all this in one go. We simply delete the first occurrence of a key that we are looking for, and keep on generating new keys (in depth–first manner). Once we need no more keys, the output is a term. It is the term for the string $\vec{x}$ if the input tape is empty. If we need keys but the input tape is empty, the output is not a term. We have to verify that this algorithm runs in polynomial time. However, this is not hard to show and is left to the reader. □

Now we shall start the proof of an important theorem on the characterization of **PTIME** languages. An important step is a theorem by Chandra, Kozen und Stockmeyer (1981), which characterizes the class **PTIME** by a space requirement. Here however we need to introduce some special machines, which look like Turing machines but have a special way of handling parallelism. Before we can do that, we introduce yet another class of functions. We say that a function $f : A^* \to B^*$ is in **LOGSPACE** if it can be computed by a so called deterministic logarithmically bounded Turing machine. Here, a Turing machine is called **logarithmically space bounded** if it has $k+2$ tapes (where k may be 0) such that the length of the tapes number 1 through k is bounded by $\log_2 |\vec{x}|$. Tape 0 serves as the input tape, Tape $k+1$ as the output tape. Tape 0 is read only, Tape k is write only. At the end of the computation, T has to have written $f(\vec{x})$ onto that tape. (Actually, a moment's reflection shows that we may assume that the length of the intermediate tapes is bounded by $c \log_2 |\vec{x}|$, where c is a constant.) This means that if $\vec{x}$ has length 12 the tapes 2 to $k+1$ have length 3 since $3 < \log_2 12 < 4$. It need not concern us further why this restriction makes sense. We shall see in Section 5.2 that it is well motivated. We emphasize that $f(\vec{x})$ can be arbitrarily large. It is not restricted at all in its length, although we shall later see that the machine cannot compute outputs that are too long anyway. The reader may reflect on the fact that we may require the machine to use the last tape only in this way: it moves strictly to the right without ever looking at the previous cells again. Further, we can see to it that the intermediate tapes only contain single binary numbers.

Definition 5.1.6 *Let A^* be a finite alphabet and $L \subseteq A^*$. We say that L is in **LOGSPACE** if χ_S is deterministically **LOGSPACE**–computable.*

Theorem 5.1.7 *Let $f : A^* \to B^*$ be **LOGSPACE**–computable. Then f is in **PTIME**.*

Proof. We look at the configurations of the machine. A configuration is defined with the exception of the output tape. It consists of the positions of the read head of the first tape and the content of the intermediate tapes plus the position of the read/write heads of the intermediate tapes. Thus the configurations are k–tuples of binary numbers of length $\leq c \log_2 |\vec{x}|$. A position on a string likewise corresponds to a binary number. So we have $k+1$ binary numbers and there are at most

$$2^{(k+1)\cdot c \log_2 |\vec{x}|} = |\vec{x}|^{c(k+1)}$$

of them. So the machine can calculate at most $|\vec{x}|^{c(k+1)}$ steps. For if there are more the machine is caught in a loop, and the computation does not terminate. Since this was excluded, there can be at most polynomially many steps. □

Since f is polynomially computable we immediately get that $|f(\vec{x})|$ is likewise polynomially bounded. This shows that a space bound implies a time bound. (In general, if $f(n)$ is the space bound then $2^{f(n)}$ is the corresponding time bound.)

We have found a subclass of **PTIME** which is defined by its space consumption. Unfortunately, these classes cannot be shown to be equal. We have to do much more work. For now, however, we remark the following.

Theorem 5.1.8 *Let $f : A^* \to B^*$ and $g : B^* \to C^*$ be **LOGSPACE** computable. Then $g \circ f$ also is **LOGSPACE** computable.*

Proof. By assumption there exists a logarithmically space bounded deterministic $k+2$–tape machine T which computes f and a logarithmically space bounded deterministic $\ell+2$–tape machine U which computes g. We cascade these machines in the following way. We use $k+\ell+3$ tapes, of which the first $k+2$ are the tapes of T and the last $\ell+2$ the tapes of U. We use the tape number $k+1$ both as the output tape of T and as the input tape of U. Call it τ. The resulting machine is deterministic but not necessarily logarithmically space bounded. The problem is Tape τ. However, we shall now demonstrate that this tape is not needed at all. For notice that T cannot but move forward on this tape and write on it, while U on the other hand can only progress to read the input. Now rather than reading the symbols from Tape τ, it would be enough for U if it can by some means access the symbol number

i on the output tape of T on request. Since we do not care about the amount of time it takes to compute that symbol, it is clear that if U can always get this information on request, it does not need that tape. Hence we do the following. We replace Tape τ by two logarithmically space bounded tapes, which we call τ_T and τ_U. τ_T contains the binary code of the position of the write head of τ. So, we keep T running, but instead of writing on its output tape, we only note the binary code of the position of the write head. (Additionally, the symbol on that cell is coded in the internal state of the machine.) τ_U on the other hand codes the position of the read head of U on Tape τ_U (its input tape) in binary. We have $|f(\vec{x})| \leq p(|\vec{x}|)$ for some polynomial p for the length of the output computed by T, so we have $\log_2 |f(\vec{x})| \leq \lambda \log_2 |\vec{x}|$ for some natural number λ. So the two tapes are logarithmically bounded. The new machine works as follows. In place of the pointers on the input tape, U now uses the binary code on tape τ_U. This number is diminished or increased by one, if U makes a move. (These numerical operations may require additional tapes, which we have not mentioned.) Whenever U needs the symbol number i, it calls the machine T and gives it the number i in binary (copying a number onto that tape). T starts the computation of $f(\vec{x})$ and decreases the number on the tape τ_T until it is zero. Then, the machine has successfully computed the symbol (which can be encoded into the state the machine is in). □

This proof is the key to all following proofs. We shall now show that there is a certain class of problems which are, as one says, *complete* with respect to the class **PTIME** modulo **LOGSPACE**–reductions. These shall be taken as a measuring rod for the characterization of **PTIME** problems.

An n–ary **boolean function** is an arbitrary function $f : 2^n \to 2$. Every such function is contained in the polynomial clone of functions generated by the functions $\cup$, $\cap$ and $-$. These are easily seen to be the following functions.

$\cap$	0	1
0	0	0
1	0	1

$\cup$	0	1
0	0	1
1	1	1

	$-$
0	1
1	0

We shall now assume that f is somehow composed from the functions $\wedge$, $\vee$, $\neg$. For example, let $f(x_0, x_1, x_2) := -(-x_2 \cap (x_1 \cup x_0))$. Now for the variables x_0, x_1 and x_2 we insert concrete values (so either 0 or 1). Which value does f have? This problem can be solved in **PTIME**. This is relatively easy to see. However, the formulation of the problem is a delicate affair. Namely, we want to write down f not simply as a string in the usual way but as a sequence of so called *cells*. A **cell** is a string of the form $(\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\vec{\eta}\texttt{,}\vee\texttt{)}$, $(\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\vec{\eta}\texttt{,}\wedge\texttt{)}$, or of the form $(\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\neg\texttt{)}$ where $\vec{\alpha}$ is a binary sequence — written in the alphabet $\{\texttt{a}, \texttt{b}\}$ —, and $\vec{\varepsilon}, \vec{\eta}$ either is a binary string (written down using the letters `a` and `b` in place of `0` and `1`) or a single symbol of the form `0` or `1`. $\vec{\alpha}$ is called the **number** of the cell and $\vec{\varepsilon}$ and $\vec{\eta}$ the **argument key**, if it is not of the form `0` or `1`. Further, we assume that the number represented by $\vec{\varepsilon}$ and $\vec{\eta}$ is smaller than the number represented by $\vec{\alpha}$. (This makes sure that there are no cycles.) A sequence of cells is called a **network**, if (a) there are no two cells with identical number, (b) the numbers of cells are the numbers from 1 to a certain number ν, and (c) for every cell with argument key $\vec{\varepsilon}$ (or $\vec{\eta}$) there is a cell with number $\vec{\varepsilon}$ ($\vec{\eta}$). The cell with the highest number is called the **goal** of the network. Intuitively, a network defines a boolean function into which some constant values are inserted for the variables. This function shall be evaluated. With the cell number $\vec{\alpha}$ we associate a value $w_{\vec{x}}(\vec{\alpha})$ as follows.

$$w_{\vec{x}}(\vec{\alpha}) := \begin{cases} w_{\vec{x}}(\vec{\varepsilon}) \cup w_{\vec{x}}(\vec{\eta}), & \text{if } \vec{x} \text{ contains the cell } (\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\vec{\eta}\texttt{,}\vee\texttt{)}, \\ w_{\vec{x}}(\vec{\varepsilon}) \cap w_{\vec{x}}(\vec{\eta}), & \text{if } \vec{x} \text{ contains the cell } (\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\vec{\eta}\texttt{,}\wedge\texttt{)}, \\ -w_{\vec{x}}(\vec{\varepsilon}), & \text{if } \vec{x} \text{ contains the cell } (\vec{\alpha}\texttt{,}\vec{\varepsilon}\texttt{,}\neg\texttt{)}. \end{cases}$$

We write $w(\vec{\alpha})$ in place of $w_{\vec{x}}(\vec{\alpha})$. The value of the network is the value of its goal (incidentally the cell with number ν). Let

$v : W^* \to \{0, 1, \star\}$ be the following function. $v(\vec{x}) := \star$ if $\vec{x}$ is not a network. Otherwise, $v(\vec{x})$ is the value of $\vec{x}$. We wish to define a machine calculating v. Networks are strings over the alphabet $W := \{\texttt{(}, \texttt{)}, \texttt{,}, \texttt{0}, \texttt{1}, \texttt{a}, \texttt{b}, \wedge, \vee, \neg\}$. We give an example. We want to evaluate our function $f(x_0, x_1, x_2) = -(-x_2 \cap (x_1 \cup x_0))$ for $x_0 := 0$, $x_1 := 1$ und $x_2 := 0$. Then we write down the following network:

$$\texttt{(a,0,}\neg\texttt{)(b,1,0,}\vee\texttt{)(ba,a,b,}\wedge\texttt{)(bb,ba,}\neg\texttt{)}$$

Now $w(\texttt{a}) = -0 = 1$, $w(\texttt{b}) = 1 \cup 0 = 1$, $w(\texttt{ba}) = w(\texttt{a}) \cap w(\texttt{b}) = 1 \cap 1 = 1$, and $v(\vec{x}) = w(\texttt{bb}) = -w(\texttt{ba}) = -1 = 0$.

Lemma 5.1.9 *The set of all networks is in **LOGSPACE**.*

Proof. The verification is a somewhat longwinded matter but not difficult to do. To this end we shall have to run over the string several times in order to check the different criteria. The first condition is checked as follows. One begins with cell number 1 and memorizes its position. Then one runs to the next cell and memorizes its position and compares the two. If they are equal, output 0 and stop. Otherwise, continue with the next cell. Compare the numbers. If they are equal, output 0 and stop. Otherwise continue to the next cell. And so on. To memorize a position requires logarithmic space. To compare two strings at different positions requires to memorize only one symbol at a time, running back and forth between the strings. □

Theorem 5.1.10 *v is in **PTIME**.*

Proof. Let $\vec{x}$ be given. First we compute whether $\vec{x}$ is a network. This is in **PTIME**. If $\vec{x}$ is not a network, output $\star$. If it is, we do the following. Moving up with the number k we compute the value of the cell number k. For each cell we have to memorize its value on a separate tape, storing pairs $(\vec{\alpha}, w(\vec{\alpha}))$ consisting of the name and the value of that cell. This can be done in polynomial time. Once we have reached the cell with the highest number we are done. □

It is not known whether can can actually calculate the value of a network in **LOGSPACE**. The problem is that we may not be able to bound the number of intermediate values. Now the following holds.

Theorem 5.1.11 *Let $f : A^* \to \{\mathtt{0}, \mathtt{1}\}$ be in **PTIME**. Then there exists a function $N : A^* \to W^*$ in **LOGSPACE** such that for every $\vec{x} \in A^*$ $N(\vec{x})$ is a network and $f(\vec{x}) = (v \circ N)(\vec{x})$.*

Proof. First we construct a network and then show that it is in **LOGSPACE**. By assumption there exists a k and a c such that $f(\vec{x})$ is computable in $\rho := c \cdot |\vec{x}|^k$ time using a deterministic Turing machine T. We define a construction algorithm that defines for given $\vec{x}$ a matrix $\mathcal{C} := (C(i,j))_{i,j}$, where $0 \leq i, j \leq c \cdot |\vec{x}|^k$. (Actually, the word 'matrix' is somewhat inappropriate here.) It is a sequence of strings $C(i,j)$ ordered in the following way:

$$\begin{array}{l} C(0,0), C(0,1), C(0,2), \ldots, \\ C(1,0), C(1,1), C(1,2), \ldots, \\ C(2,0), C(2,1), C(2,2), \ldots \end{array}$$

$C(i,j)$ contains the following information: (a) the content of the jth cell of the tape of T at time point i, (b) information, whether the read head is on that cell at time point i, (c) if the read head is on this cell at i also the state of the automaton. This information needs bounded length. Call the bound λ. We denote by $C(i,j,k)$ the kth binary digit of $C(i,j)$. The entry $C(i+1,j)$ depends in a predictable way on the entries $C(i,j-1)$, $C(i,j)$ and $C(i,j+1)$. (T is deterministic.) (A) $C(i+1,j) = C(i,j)$ if either (A1) at i the head is not at $j-1$ or else did not move right, (A2) at i the head is not at $j+1$ or else did not move left; (B) $C(i+1,j)$ can be computed from (B1) $C(i,j-1)$ if the head was at i positioned at j and moved right, (B2) $C(i,j)$ if the head was at i positioned at j and did not move, (B3) $C(i,j+1)$ if the head at i was positioned at $j+1$ and moved left. Hence, for every $k < \lambda$ there exist boolean functions f_L^k, f_M^k and f_R^k such that $f_L^k, f_R^k : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}$,

$f^k_M : \{0,1\}^{3\lambda} \to \{0,1\}^\lambda$, and

$$\begin{array}{rcl} C(i+1,0,k) & = & f_L(C(i,0),C(i,1)) \\ C(i+1,j,k) & = & f_M(C(i,j-1),C(i,j),C(i,j+1)) \\ C(i+1,\rho,k) & = & f_R(C(i,\rho-1),C(i,\rho)) \end{array}$$

These functions can be computed from T and take an amount of time independent of $\vec{x}$. Moreover, we can compute sequences of cells that represent these functions. Basically, the network we have to construct results in replacing for every $i > 0$ and appropriate j the cell $C(i,j)$ by a sequence of cells calling on appropriate other cells to give the value $C(i,j)$. This sequence is obtained by adding a fixed number to each argument key of the cells of the sequence computing the boolean functions.

Now let $\vec{x}$ be given. We compute a sequence of cells $\gamma(i,j)$ corresponding to $C(i,j)$. The row $\gamma(0,j)$ is empty. Then ascending in i, the rows $\gamma(i,j)$ are computed and written on the output tape. If row i is computed, the following numbers are computed and remembered: the length of the ith row, the position of the read head at $i+1$ and the number of the first cell of $\gamma(i,j')$, where j' is the position of the read head at i. Now the machine writes down $C(i+1,j)$ with ascending j. This is done as follows. If j is not the position of the read head at $i+1$, the sequence is a sequence of cells that repeat the value of the cells of $\gamma(i,j)$. (So, $\gamma(j+1,i,k) = (\vec{\alpha},\vec{\varepsilon},\mathtt{1},\wedge)$ for $\vec{\alpha}$ the number of the actual cell and $\vec{\varepsilon}$ is $\vec{\alpha}$ minus some appropriate number, which is computed from the length of the ith row the length of the sequence $\gamma(i,j')$ and the length of the sequence $\gamma(i+1,j)$. If j is the position of the read head, we have to insert more material, but basically it is a sequence shifted as discussed above. The number by which we shift can be computed in **LOGSPACE** from the numbers which we have remembered. Obviously, it can be decided on the basis of this computation when the machine T terminates on $\vec{x}$ and therefore when to stop the sequence. The last entry is the goal of the network. □

One also says that the problem of calculating the value of a network is complete with respect to the class **PTIME**. A network is **monotone** if it does not contain the symbol $\neg$.

Theorem 5.1.12 *There exists a **LOGSPACE**–computable function M which transforms an arbitrary network into a monotone network with identical value.*

The proof is longwinded but rather straightforward, so we shall only sketch it. We assume that our network computes the function $f(\vec{x})$ for some argument vector. Now we change $\vec{x}$ in such a way that only $\wedge$ and $\vee$ are used. If $\vec{x} = \prod_{i<n} x_i$ define $y_i := \mathtt{01}$ if $x_i = \mathtt{0}$, and $y_i := \mathtt{10}$ else. Finally put $\vec{y} := \prod_{i<n} y_i$. This means: if we have $\mathtt{1}$ at position $2i$ then we have $\mathtt{0}$ at position $2i+1$, and if we have $\mathtt{0}$ at position $2i$ then we have $\mathtt{1}$ at position $2i+1$. If for example $\vec{x} = \mathtt{010}$ then $\vec{y} = \mathtt{011001}$. We claim: there exists a monotone function g such that $g(\vec{y}) = f(\vec{x})$. This is easy to show and is left as an exercise. Now, the input is doubled. This is transferred to the entire network. Every cell of the form $(\vec{\alpha}, \vec{\varepsilon}, \vec{\eta}, \wedge)$ is doubled by cells of the form $(\vec{\alpha}_i, \vec{\varepsilon}_i, \vec{\eta}_i, \vee)$, $i < 2$, and dually for the cell of the form $(\vec{\alpha}, \vec{\varepsilon}, \vec{\eta}, \vee)$ by cells of the form $(\vec{\alpha}_i, \vec{\varepsilon}_i, \vec{\eta}_i, \wedge)$. The basis for this are the so called de Morgan Laws $-(x \cap y) = (-x) \cup (-y)$ and $-(x \cap y) = (-x) \cap (-y)$. With each cell Z with value x there exists a cell Z_1 with value $\neg x$. We shall see to it that the numbers of Z and Z_1 differ only by one. Now we can eliminate $\neg$ completely. For if the cell Z uses the value of the cell U (in the old network) with the number k now negated then we can now find the value of U at number $k'+1$ or $k'-1$, depending on whether k' is even or odd. An the end one has to do some cosmetics: we eliminate the last cell if it is not needed.

We now have the following situation: if g is an arbitrary **PTIME**–computable function from A^* to $\{0, 1\}$, the exists a **LOGSPACE**–computable function N^+, which for given $\vec{x}$ constructs a monotone network with identical value.

Now we shall turn to the promised new type of machines.

Definition 5.1.13 *An **alternating Turing machine** is a sex-*

tuple

$$\langle A, \mathtt{L}, Q, q_0, f, \gamma \rangle \ ,$$

where $\langle A, \mathtt{L}, Q, q_0, f \rangle$ *is a Turing machine and* $\gamma : Q \to \{\wedge, \vee\}$ *an arbitrary function. A state* q *is called* ***universal*** *if* $\gamma(q) = \wedge$*, and otherwise* ***existential****.*

We tacitly generalize the concepts of Turing machines to the alternating Turing machines. So we assume that the reader is able to perform these generalizations, for example an alternating k–tape Turing machine, and a logarithimically space bounded alternative Turing machine. To this end one has to add the function γ in the definitions. Now we have to define when a Turing machine accepts an input $\vec{x}$. This is done via configurations. A configuration is said to be **accepted** by T if one of the following is the case:

- ▷ T is in an existential state and one of the immediately subsequent configurations is accepted by T.
- ▷ T is in a universal state and all immediately subsequent configurations are accepted by T.

Notice that the machine can accept a configuration that has no immediately subsequent configurations. In this case the computation terminates, and it is in a state that is universal. The difference between universal and existential states is effective if the machine is not deterministic. Then there can be several subsequent configurations. The typical definition of a Turing machine defines the acceptance in the same way as for an existential state if there is a successor state, otherwise however it defines it like a universal state. Universal states are of a different kind. In this case the machine must split itself into several copies that compute the various subsequent alternatives. Now we define $f : A^* \to B^*$ to be in **ALOGSPACE** if there is a logarithmically space bounded alternating k–tape Turing machine which computes f.

Definition 5.1.14 *A language* $L \subseteq A^*$ *is in* ***ALOGSPACE*** *if its characteristic function is* ***ALOGSPACE****–computable.*

Theorem 5.1.15 (Chandra & Kozen & Stockmeyer)

$$\boldsymbol{ALOGSPACE} = \boldsymbol{PTIME}.$$

There is little work left to prove this theorem. First of all, the following is clear.

Lemma 5.1.16 $\boldsymbol{LOGSPACE} \subseteq \boldsymbol{ALOGSPACE}$.

For every deterministic logarithmically space bounded Turing machine also is an alternating machine by simply letting every state be universal. Likewise the following claim is easy to show, if we remind ourselves of the facts concerning **LOGSPACE**–computable functions.

Lemma 5.1.17 *Let $f : A^* \to B^*$ and $g : B^* \to C^*$ be functions. If f and g are in* $\boldsymbol{ALOGSPACE}$*, so is $g \circ f$.*

Lemma 5.1.18 $\boldsymbol{ALOGSPACE} \subseteq \boldsymbol{PTIME}$.

Also, this proof is not hard. We already know that there are at most polynomially many configurations. The dependency between these configurations can also be checked in polynomial time. (Every configuration has a bounded number of successors. The bound does only depend on T.) This yields a computation tree which can be determined in polynomial time. Now we must determine in the last step whether the machine accepts the initial configuration. To this end we must determine by induction on the depth in a computation tree whether the respective configuration is accepted. This can be done as well in polynomial time. This completes the proof.

Now the converse inclusion remains to be shown. For this we use the following idea. Let f be in **PTIME**. We can write f as $v \circ N^+$ where N^+ is a monotone network computing f. As remarked above we can construct N^+ in **LOGSPACE** and in particular because of Lemma 5.1.16 in **ALOGSPACE**. If now we can show that v is in **ALOGSPACE** and so Lemma 5.1.17 gives us that $f = v \circ N^+ \in$ **ALOGSPACE**.

Lemma 5.1.19 $v \in \boldsymbol{ALOGSPACE}$.

Proof. We construct a logarithmically space bounded alternating machine which for an arbitrary given monotone network $\vec{x}$ calculates its value $w(\vec{x})$. Let an arbitrary network be given. First move to the goal. Descending from it compute as follows.

1. If the cell contains $\wedge$ go into the universal state q_1.
2. If the cell contains $\vee$ go into the existential state q_2.
3. Choose and argument key $\vec{\alpha}$ of the current cell and go to the cell number $\vec{\alpha}$.
4. If $\vec{\alpha}$ is not an argument key go into state q_f if $\vec{\alpha} = \mathtt{1}$ and into q_g if $\vec{\alpha} = \mathtt{0}$. Here q_f is universal and q_g existential and there are no transitions defined from q_f and q_g.

All other states are universal, however the machine works nondeterministically only in one case, namely if it gets the values of the arguments. Then it makes a nondeterministic choice. If the cell is an $\vee$–cell then it will accept that configuration if one argument has value 1, since the state is existential. If the cell is a $\wedge$–cell then it shall accept the configuration if both arguments have value 1 for now the state is universal. The last condition is the termination condition. If the string is not an argument key then it is either `0` or `1` and its value can be computed without recourse to other cells. If it is `1` the automaton changes into a final state which is universal and so the configuration is accepted. If the value is `0` the automaton changes into a final state which is existential and the configuration is rejected. $\Box$

Notes on this Section. In computer science, the gap between **PTIME** and **NPTIME** is considered to be a very big one, though it is even open whether the two are distinct. Polynomial time computable problems are called 'tractable', others 'intractable'. Thus, the fact that virtually all languages are in **PTIME** is good news, telling us that natural languages are tractable, at least syntactically. Concerning the tractability of languages as sign systems not very much is known, however.

Exercise 165. Show Proposition 5.1.2: *With* L_1 *and* L_2 *also* $L_1 \cup L_2$, $L_1 \cap L_2$ *as well as* $A^* - L_1$ *are in* ***PTIME***.

Exercise 166. Show the following. *If* L_1 *and* L_2 *are in* ***PTIME*** *then so is* $L_1 \cdot L_2$.

Exercise 167. Show that L has the constant growth property if L is semilinear. Give an example of a language which has the constant growth property but is not semilinear.

Exercise 168. Let $f : 2^n \to 2$ a boolean function. Show that it can be obtained from the constants and the functions $-$, $\cup$, and $\cap$. *Hint.* Start with the functions $g_{\vec{x}} : 2^n \to 2$ such that $g_{\vec{x}}(\vec{y}) := 1$ if and only if $\vec{x} = \vec{y}$. Show that they can be generated from $-$ and $\cap$. Proceed to show that every boolean function is either the constant 0 or can be obtained from functions of type $g_{\vec{x}}$ using $\cup$.

Exercise 169. Show that for every n–ary boolean function f there is a monotone boolean $2n$–ary function g such that

$$f(x_0, \ldots, x_{n-1}) = g(x_0, -x_0, x_1, -x_1, \ldots, x_{n-1}, -x_{n-1}).$$

Exercise 170. Call a language $L \subseteq A^*$ **weakly semilinear** if every intersection with a semilinear language $\subseteq A^*$ has the constant growth property. Show that every semilinear language is also weakly semilinear. Let $M := \{\mathtt{a}^m\mathtt{b}^n : n \geq 2^m\}$. Show that $M \subseteq \{\mathtt{a}, \mathtt{b}\}^*$ is weakly semilinear but not semilinear.

Exercise 171. Show that every function $f : A^* \to B^*$ which is computable using an alternating Turing machine can also be computed using a Turing machine. *Hint.* It is not a priori clear that the class of alternating Turing machines is not more powerful than the class of Turing machines. This has to be shown.

5.2 Literal Movement Grammars

The concept of a literal movement grammar — **LMG** for short — has been introduced by Annius Groenink (1997). With the help of these grammars one can characterize the **PTIME** languages by means of a generating device. The idea to this characterization goes back to a result by Bill Rounds (1988). Many grammar types are special LMGs. The central feature of LMGs is that the rules contain a context free skeleton which describes the abstract structure of the string and in addition to this a description of the way in which the constituent is formed from the basic parts. The notation is as follows. In place of a nonterminal Q we now write $Q(\vec{x})$, where $\vec{x}$ is a string, and say that the string $\vec{x}$ has the property Q or that $\vec{x}$ **is a** Q–**string**. The properties play the role of the nonterminals in the context free grammars, but technically speaking they are handled differently. For a LMG does not treat them as symbols and therefore does not write Q. Rather, it treats them as properties of strings and therefore writes $Q(\vec{x})$. Since $\vec{x}$ is metavariable for strings, we now need another set of (official) variables for strings in the fomulation of the LMGs. To this end we use the plain symbols x, y, z and so on (with subscripts) for these variables. In addition to these variables there are also constants $\mathtt{a}$, $\mathtt{b}$, for the symbols of our alphabet A. We look at a simple example of an LMG. It has two rules.

$$S(x \cdot x) \leftarrow S(x).; \qquad S(\mathtt{a}) \leftarrow .$$

These rules are written in Horn—clause format, as in `Prolog`. Indeed, this corresponds exactly to the way they are being used. The first rule says: if x is a string of category S so is $x \cdot x$. The second rule says: $\mathtt{a}$ is a string of category S. We agree that S is — as usual — the distinguished symbol. The language which is generated by this grammar is the smallest set of strings that satisfies these rules, and it is denoted by $L(G)$. We may formalize this in so called monadic second order predicate logic (see Section 6.1

for a definition) as follows.

$$\varphi(S) := (\forall x)(S(x) \to S(x \cdot x)) \wedge S(\mathsf{a}).$$

$$L(G) = \{x : (\forall S)(\varphi(S) \to S(x))\}$$

So, $L(G)$ can be described by means of a sentence in monadic second order predicate logic if one assumes variables for strings, and as logical symbols the equality, a constant for ε the letters of the alphabet and the concatenation operation.

Proposition 5.2.1 *$L(G) = \{\mathsf{a}^{2^n} : 0 \leq n\}$.*

Proof. First we have: $\mathsf{a} \in L(G)$. This finishes the case $n = 0$. By induction one shows $\mathsf{a}^{2^n} \in L(G)$ for every $n > 0$. For if a^{2^n} is a string of category S so is $\mathsf{a}^{2^{n+1}} = \mathsf{a}^{2^n} \cdot \mathsf{a}^{2^n}$. This shows that $L(G) \supseteq \{\mathsf{a}^{2^n} : n \geq 0\}$. On the other hand this set satisfies the formula φ. For we have $\mathsf{a} \in L(G)$ and with $\vec{x} \in L(G)$ we also have $\vec{x} \cdot \vec{x} \in L(G)$. For if $\vec{x} = \mathsf{a}^{2^n}$ for a certain $n \geq 0$ then $\vec{x} \cdot \vec{x} = \mathsf{a}^{2^n \cdot 2} = \mathsf{a}^{2^{n+1}} \in L(G)$. ☐

Alternatively for the definition of $L(G)$ there is a direct definition of $L(G)$ by means of generation. We say $G \vdash S(\vec{x})$ (vector arrow!), if either $S(\vec{x}) \leftarrow .$ is a rule or $\vec{x} = \vec{y} \cdot \vec{y}$ and $G \vdash S(\vec{y})$. Both definitions define the same set of strings. Now we define the so called 1–LMGs.

Definition 5.2.2 *Let A be an alphabet. A **1–LMG** over A is a quadruple $\langle S, N, A, R\rangle$, where N is a finite set disjoint to A, called the set of **predicates**, $S \in N$, and R a finite set of **rules**. An (**n–ary**) rule has the form*

$$T(t) \leftarrow U_0(s_0), U_1(s_1), \ldots, U_{n-1}(s_{n-1}).$$

*where t, s_i $(i < n)$ are polynomials which are built from the terminal symbols, variables and ε with the help of $\cdot$ (concatenation). The maximum of all n such that G has an n–ary rule is called the **branching number of** G.*

In the rule

$$S(x \cdot x) \leftarrow S(x).$$

we have $n = 1$ and $T = U_0 = S$, $t = x \cdot x$ and $s_0 = x$, $n = 0$ and $T = S$, $t = \mathsf{a}$, respectively. In the rule

$$S(\mathsf{a}) \leftarrow .$$

we have $n = 0$, $T = S$ and $t = \mathsf{a}$. This definition is not the final one, even though it already is quite involved. We now define what it means that a 1–LMG generates or accepts a string. Before we do so, we make the following agreement. A function α which associates a string in A^* to each string variable is called a **valuation**. Given α we define s^α for a polynomial by homomorphic extension. For example, if $s = \mathsf{a} \cdot x^2 \cdot \mathsf{b} \cdot y$ and $\alpha(x) = \mathsf{ac}$, $\alpha(y) = \mathsf{bba}$ then $s^\alpha = \mathsf{aacacbbba}$, as is easily computed.

Definition 5.2.3 *Let $G = \langle S, N, A, R\rangle$ be a 1–LMG over A. We define $G \vdash^0 Q(\vec{x})$, if there is a polynomial s and a valuation α such that (a) $\vec{x} = s^\alpha$ and (b) $Q(s) \leftarrow . \in R$. Further, $G \vdash^{n+1} Q(\vec{x})$ holds if $G \vdash^n Q(\vec{x})$ or if there is an $n \in \omega$, polynomials t, s_i, $i < n$, predicates R_i, $i < n$, strings $\vec{y}_i$, $i < n$, and a valuation α such that the following holds.*

- $\vec{x} = t^\alpha$,
- $\vec{y}_i = s_i^\alpha$ *for all* $i < n$,
- $G \vdash^n R_i(\vec{y}_i)$ *for all* $i < n$, *and*
- $Q(t) \leftarrow R_0(s_0), \dots, R_{n-1}(s_{n-1}). \in R.$

Finally, $G \vdash Q(\vec{x})$ if and only if there is an $n \in \omega$ such that $G \vdash^n Q(\vec{x})$.

$$L(G) := \{\vec{x} : G \vdash S(\vec{x})\}$$

is the ***language generated by*** *G.*

We shall give an example to illustrate these definitions. Let K be the following grammar.

$$S(v \cdot x \cdot y \cdot z) \leftarrow S(v \cdot y \cdot x \cdot z).; \qquad S(\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot x) \leftarrow S(x).; \quad S(\varepsilon) \leftarrow .$$

Then $L(K)$ is that language which contains all strings that contain an identical number of $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$. To this end one first shows that $(\mathtt{abc})^* \in L(K)$ and in virtue of the first rule $L(K)$ is closed under permutations. Here $\vec{y}$ is a **permutation of** $\vec{x}$ if $\vec{y}$ and $\vec{x}$ have identical image under the Parikh map. Here is an example (the general case is left to the reader as an exercise). We can derive $S(\mathtt{abc})$ in one step from $S(\varepsilon)$ using the second rule, and $S(\mathtt{abcabc})$ in two steps, using again the second rule. In a third step we can derive $S(\mathtt{aabbcc})$ from this, using the first rule this time. Put $\alpha(v) := \mathtt{a}$, $\alpha(x) := \mathtt{ab}$, $\alpha(y) := \mathtt{bc}$ and $\alpha(z) := \mathtt{c}$. Then

$$(v \cdot x \cdot y \cdot z)^\alpha = \mathtt{aabbcc}, \quad (v \cdot y \cdot x \cdot z)^\alpha = \mathtt{abcabc}$$

There is another way to conceptualize these definitions. We say that

$$T(\vec{x}) \leftarrow U_0(\vec{y}_0), U_1(\vec{y}_1), \ldots, U_{n-1}(\vec{y}_{n-1}).$$

is an **instance** of the rule

$$T(t) \leftarrow U_0(s_0), U_1(s_1), \ldots, U_{n-1}(s_{n-1}).$$

if there is a valuation β such that $\vec{x} = t^\beta$ and $\vec{y}_i = s_i^\beta$ for all $i < n$. Now we can say: $G \vdash^0 Q(\vec{x})$ if $Q(\vec{x}) \leftarrow .$ is an instance of a rule of G and $G \vdash^{n+1} Q(\vec{x})$ if $G \vdash^n Q(\vec{x})$ or there is a number n, predicates R_i and strings $\vec{y}_i$, $i < n$ such that

- ▷ $G \vdash^n R_i(\vec{y}_i)$, and
- ▷ $Q(\vec{x}) \leftarrow R_0(\vec{y}_0), \ldots, R_{n-1}(\vec{y}_{n-1})$ is an instance of a rule of G.

This makes the notion of generation by an LMG somewhat simpler. We have to regard the rules as schemata into which we insert concrete strings for variables.

Let H be a context free grammar. We define a 1–LMG $H^\spadesuit$ as follows. (For the presentation we shall assume that H is already in Chomsky normal form.) For every nonterminal $\mathtt{A}$ we introduce a predicate $\underline{\mathtt{A}}$. The start symbol is $\underline{S}$. If $\mathtt{A} \to \mathtt{B}\ \mathtt{C}$ is a rule from H then $H^\spadesuit$ contains the rule

$$\underline{\mathtt{A}}(x \cdot y) \leftarrow \underline{\mathtt{B}}(x), \underline{\mathtt{C}}(y).$$

If $\mathtt{A} \to \mathtt{a}$ is a terminal rule then we introduce the following rule into $H^\spadesuit$:

$$\underline{\mathtt{A}}(\mathtt{a}) \leftarrow .$$

One can show relatively easily that $L(H) = L(H^\spadesuit)$.

The LMGs can therefore generate all context free languages. Additionally, they can generate languages without constant growth, as we have already seen. Let us note the following facts.

Theorem 5.2.4 *Let L_1 and L_2 be languages over A which can be generated by 1–LMGs. Then there exist 1–LMGs generating the languages $L_1 \cap L_2$ and $L_1 \cup L_2$.*

Proof. Let G_1 and G_2 1–LMGs which generate L_1 and L_2, respectively. We assume that the set of nonterminals of G_1 and G_2 are disjoint. Let S_i be the start predicate of G_i, $i \in \{1, 2\}$. Let $H_\cup$ be constructed as follows. We form the union of the nonterminals and rules of G_1 and G_2. Further, let $\underline{S}^\heartsuit$ be a new predicate, which will be the start predicate of $H_\cup$. At the end we add the following rules: $S^\heartsuit(x) \leftarrow S_1(x).$; $S^\heartsuit(x) \leftarrow S_2(x)$. This defines $H_\cup$. $H_\cap$ is defined similarly, only that in place of the last two rules we have a single rule, $S^\heartsuit(x) \leftarrow S_1(x), S_2(x)$. It is easily checked that $L(H_\cup) = L_1 \cup L_2$ and $H_\cap = L_1 \cap L_2$. We show this for $H_\cap$. We have $\vec{x} \in L(H_\cap)$ if there is an n with $H_\cap \vdash^n S^\heartsuit(\vec{x})$. This in turn is the case exactly if $n > 0$ and $H_\cap \vdash^{n-1} S_1(\vec{x})$ as well as $H_\cap \vdash^{n-1} S_2(\vec{x})$. This is nothing but $G_1 \vdash^{n-1} S_1(\vec{x})$ and $G_2 \vdash^{n-1} S_1(\vec{x})$. Since n was arbitrary, we have $\vec{x} \in H_\cap$ if and only if $\vec{x} \in L(G_1) = L_1$ and $\vec{x} \in L(G_2) = L_2$, as promised. $\Box$

The 1–LMGs are quite powerful, as the following theorem shows.

Theorem 5.2.5 *Let A be a finite alphabet and $L \subseteq A^*$. $L = L(G)$ for a 1–LMG if and only if L is recursively enumerable.*

The proof is left to the reader as an exercise. Since the set of recursively enumerable languages is closed under union and intersection, Theorem 5.2.4 already follows from Theorem 5.2.5. It also follows that the complement of a language that can be generated by a 1–LMG does not have to be such a language again. For the complement of a recursively enumerable language does not have to be recursively enumerable again. (Otherwise every recursively enumerable set is also decidable, which is not the case.)

In order to arrive at interesting classes of languages we shall restrict the format of the rules. Some definitions are in order. Let the following rule be given.

$$\rho := T(t) \leftarrow U_0(s_0), U_1(s_1), \ldots, U_{n-1}(s_{n-1}).$$

- ▷ ρ is called **upward nondeleting** if every variable which occurs in one of the s_i, $i < n$, also occurs in t.
- ▷ ρ is called **upward linear** if no variable occurs more than once in t.
- ▷ ρ is called **downward nondeleting** if every variable which occurs in t also occurs in one of the s_i.
- ▷ ρ is called **downward linear** if none of the variables occurs twice in the s_i. (This means: the s_i are pairwise disjoint in their variables and no variable occurs twice in any of the s_i.)
- ▷ ρ is called **noncombinatorial** if the s_i are variables.
- ▷ ρ is called **simple** if it is noncombinatorial, upward nondeleting and upward linear.

G has the property $\mathcal{P}$ if all rules of G possess $\mathcal{P}$. In particular the type of simple grammars shall be of concern for us. The definitions are not always what one would intuitively expect. For example, the following rule is called upward nondeleting even though applying this rule means deleting a symbol: $\mathtt{U}(x) \leftarrow \mathtt{U}(x \cdot \mathtt{a})$. This is so since the definition focusses on the variables and ignores the constants. Further, downward linear could alternatively be formulated as follows. One requires any symbol to occur in t as often as it occurs in the s_i taken together. This, however, is too strong. One would like to allow for a variable to occur twice to the right even though on the left it occurs only once.

Lemma 5.2.6 *Let ρ be simple. Further, let*

$$Q(\vec{y}) \leftarrow R_0(\vec{x}_0), R_1(\vec{x}_1), \ldots, R_{n-1}(\vec{x}_{n-1}).$$

be an instance of ρ. Then $|\vec{y}| \geq \sum_{i<n} |\vec{x}_i| \geq \max\{|\vec{x}_i : i < n\}|$. Further, $\vec{x}_i$ is a subword of $\vec{y}$ for every $i < n$.

Theorem 5.2.7 *Let $L \subseteq A^*$ be generated by some simple 1–LMG. Then L is in **PTIME**.*

Proof. Let $\vec{x}$ be an arbitrary string and $n := \sharp N \cdot |\vec{x}|$. Because of Lemma 5.2.6 for every predicate Q: $G \vdash Q(\vec{x})$ if and only if $G \vdash^n Q(\vec{x})$. From this follows that every derivation of $S(\vec{x})$ has length at most n. Further, in a derivation there are only predicates of the form $Q(\vec{y})$ where $\vec{y}$ is a subword of $\vec{x}$. The following chart–algorithm (which is a modification of the standard chart–algorithm) only takes polynomial time:

- ▷ For $i = 0, \ldots, n$: For every substring $\vec{y}$ of length i and every predicate Q check if there are subwords $\vec{z}_j$, $j < p$, of length $< i$ and predicates R_j, $j < p$, such that $Q(\vec{y}) \leftarrow R_0(\vec{z}_0), R_1(\vec{z}_1), \ldots, R_{p-1}(\vec{z}_{p-1})$ is an instance of a rule of G.

The number of subwords of length i is proportional to n. A subword can decomposed in $O(n^{p-1})$ ways as product of p (sub)strings.

Thus for every i, $O(n^p)$ many steps are required, in total $O(n^{p+1})$ on a deterministic multitape Turing machine. □

The converse of Theorem 5.2.7 is in all likelihood false. Now we shall present the concept of an LMG in full generality. Basically, in order to get the full definition, all we need to do is to vectorize the polynomials. We allow that the predicates are not only unary predicates of strings but predicates of any finite arity. The arity can be chosen uniformly for all predicates. However, in practice it is better to keep the arity flexible.

Definition 5.2.8 *Let A be an alphabet. A **literal movement grammar (LMG** for short) **over** A is a quintuple $G = \langle S, N, \Omega, A, R\rangle$, where N is a finite set disjoint to A, called the set of **predicates**, $\Omega : N \to \omega$ a function, assigning an arity to each predicate, $S \in N$ with $\Omega(S) = 1$, and R a finite set of **rules**. An (**n–ary**) rule has the form*

$$T(t^0, t^1, \dots, t^{p-1}) \leftarrow \; U_0(s_0^0, s_0^1, \dots, s_0^{q_0-1}), U_1(s_1^0, s_1^1, \dots, s_1^{q_1-1}), \dots, \\ \dots, U_{n-1}(s_{n-1}^0, s_{n-1}^1, \dots, s_{n-1}^{q_{n-}-1}).$$

*where $p = \Omega(T)$, $q_i = \Omega(U_i)$, and the t^j and the s_i^j $(i < n, j < q_i)$ are polynomials which are built from the terminal symbols, the string variables and ε using only $\cdot$ (concatenation). If k is at least as large as the maximum of all $\Omega(U)$, $U \in N$, we say that G is a k–**LMG**.*

For future purposes we shall agree on the following terminology.

Definition 5.2.9 *Let P be an n–ary predicate and t_i, $i < n$, terms. Then $L := P(\langle t_i : i < n\rangle)$ is called a **literal**. If σ is a substitution of terms for variables then $L^\sigma := P(\langle t_i^\sigma : i < n\rangle)$ is called an **instance of** L. If $\sigma(x) \in A^*$ for all x we call σ a **ground substitution** and L^σ a **ground instance of** L.*

The notions of instance and generation are the same as before. Likewise the other notions are carried over to the general case. We have kept the arity of the predicates flexible. There sometimes occurs the situation that one wishes to have uniform arity for all predicates. This can be arranged as follows. For an i–ary predicate A (where $i < k$) we introduce a k–ary predicate A^* which satisfies

$$A^*(x_0, \ldots, x_{k-1}) \leftrightarrow A(x_0, \ldots, x_{i-1}) \wedge \bigwedge_{j=i}^{k-1} x_j \doteq \varepsilon.$$

There is a small difficulty in that the start predicate is required to be unary. So we also lift this restriction and allow the start predicate to have any arity. Let $k := \Omega(S)$. Then we put

$$L(G) := \{\prod_{i<k} \vec{x}_i : G \vdash S(x_0, \ldots, x_{k-1})\} .$$

This does not change the generative power. An important class, which we shall study in the sequel, is the class of simple LMGs. Notice that in virtue of the definition of a simple rule a variable is allowed to occur on the right hand side several times, while on the left it may not occur more than once. This restriction however turns out to be not effective. Consider the following grammar.

$$\begin{array}{lcll} E(\varepsilon, \varepsilon) & \leftarrow & . & \\ E(a, a) & \leftarrow & . & (a \in A), \\ E(xa, ya) & \leftarrow & E(x, y). & (a \in A). \end{array}$$

This grammar generates all pairs $\langle \vec{x}, \vec{x} \rangle$ such that $\vec{x} \in A^*$. It is simple. Now take a rule in which a variable occurs several times on the left hand side.

$$S(x \cdot x) \leftarrow S(x).$$

We replace this rule by the following one.

$$S(x \cdot y) \leftarrow S(x), \quad E(x, y).$$

This latter rule is simple. Together with the rules for E we get a grammar which is simple and generates the same strings. Furthermore, we can see to it that no variable occurs more than three times on the right hand side, and that $s^j_i \neq s^j_k$ for $i \neq k$. Namely, replace s^j_i by distinct variables, say x^j_i, and add the clauses $E(x^j_i, x^{j'}_{i'})$, if $s^j_i = s^{j'}_{i'}$. We do need to introduce all of these clauses, for each variable there we only need two. (If we want to have $A_i = A_j$ for all $i < n$ we simply have to require $A_i = A_j$ for all $j \equiv i + 1 \pmod{n}$.)

With some effort we can generalize Theorem 5.2.7.

Theorem 5.2.10 *Let $L \subseteq A^*$ be generated by a simple LMG. Then L is in **PTIME**.*

The main theorem of this section will be to show that the converse also holds. We shall make some preparatory remarks. We have already seen that the class **PTIME** equals the class **ALOGSPACE**. Now we shall improve this result. Let T be a Turing machine. We call T **read only** if no head can write. If a machine has several tapes, such a machine will get the input on all of its tapes. (A read only tape is otherwise useless.) Alternatively, we may think of the machine as having only one tape but several read heads that can be independently operated.

Definition 5.2.11 *Let $L \subseteq A^*$. We say that L is in **ARO** if there is an alternating read only Turing machine which accepts L.*

Theorem 5.2.12 ***ARO** = **ALOGSPACE**.*

Proof. Let $L \in$ **ARO**. Then there exists an alternating read only Turing machine T which accepts L. We have to construct a logarithmically space bounded alternating Turing machine that recognizes L. The input and output tape remain, the other tapes are replaced by read and write tapes, which are initially empty. Indeed, let τ be a read only tape. The actions that can be performed on it are: moving the read head to the left or to the right (and reading the symbol). We code the position of the head using binary coding. Evidently, this coding needs only $\log_2 |\vec{x}| + 1$ space. Calculating the successor and predecessor (if it exists) of a binary number is **LOGSPACE** computable (given some extra tapes). Accessing the ith symbol of the input, where i is given in binary code, is as well. This shows that we can replace the read only tapes by logarithmically space bounded tapes. Hence $L \in$ **ALOGSPACE**. Suppose now that $L \in$ **ALOGSPACE**. Then $L = L(U)$ for an alternating, logarithmically space bounded Turing machine U. We shall construct a read only alternating Turing machine which accepts the same language. To this end we shall replace every intermediate Tape τ by several read only tapes. Thus, all we need to show is that the following operations are computable on read only tapes (using enough auxiliary tapes). (For simplicity, we may assume that the alphabet on the intermediate tapes is just `0` and `1`.) (a) Moving the head to the right, (b) moving the head to the left, (c) writing `0` onto the tape, (d) writing `1` onto the tape. Now, we must use at least two read only tapes; one, call it τ_a, contains the content of Tape τ, τ_b contains the position of the head of τ. The position, being bounded by $\log_2 |\vec{x}|$ can be coded in unary. Call i_a the position of the head of τ_a, i_b the position of the head of τ_b. Arithmetically, these steps correspond to the following functions: (a) $i_b \mapsto i_b + 1$, (b) $i_b + 1 \mapsto i_b$, (c) replacing the i_bth symbol in the binary code of i_a by `0`, (d) replacing the i_bth symbol in the binary code of i_a by `1`. We must show that we can compute (c) and (d). (It is easy to see that if we can compute this number, we can reset the head of τ_b onto the position corresponding to that number.) (A) The i_bth symbol in the binary code of i_a is accessed as follows. We successively divide i_a by 2, exactly i_b times, throwing away the remainder. If the number is even, the result is `0`, otherwise it is `1`. (B) 2^{i_b} is computed by doubling 1 i_b times. So, (c) is performed as follows. First establish check the i_bth digit in the representation. If it is `0`, leave i_a unchanged. Otherwise, substract 2^{i_b}. Similarly for (d). This shows that we can find an alternating read only Turing machine that recognizes L. □

Now for the announced proof. We assume that L is in **PTIME**. Then we know that there is an alternating read only Turing machine which accepts L. This machine works with k tapes. For simplicity we shall assume that the machine can move only one head in a single step. We shall construct a $2k+2$–LMG G such that $L(G) = L$. If $\vec{w}$ is the input we can code the position of a read head by a pair $\langle \vec{x}, \vec{y} \rangle$ for which $\vec{x} \cdot \vec{y} = \vec{w}$. A configuration is simply determined by naming the state of the machine and k pairs $\langle \vec{x}_i, \vec{y}_i \rangle$ with $\vec{x}_i \cdot \vec{y}_i = \vec{w}$. Our grammar will monitor the actions of the machine step by step. To every state q we associate a predicate $q^\star$. If q is existential the predicate is $2k+2$–ary and we assume the following rules with q. If q changes to r when reading the letter a and if the machine moves to the left on Tape j then the following rule is added to G.

$$\begin{aligned} & q^\star(w, x_j \cdot y_j, x_0, y_0, \dots, x_{j-1}, y_{j-1}, x'_j, y'_j, x_{j+1}, y_{j+1}, \dots, x_{k-1}, y_{k-1}) \\ & \qquad \leftarrow r^\star(w, w, x_0, y_0, \dots, x_{k-1}, y_{k-1}), \quad y'_j = a \cdot y_j, \quad x_j = x'_j \cdot a. \end{aligned}$$

If the machine moves the head to the right we add instead the following rule.

$$\begin{aligned} & q^\star(w, x_j \cdot y_j, x_0, y_0, \dots, x_{j-1}, y_{j-1}, x'_j, y'_j, x_{j+1}, y_{j+1}, \dots, x_{k-1}, y_{k-1}) \\ & \qquad \leftarrow r^\star(w, w, x_0, y_0, \dots, x_{k-1}, y_{k-1}), \quad x'_j = x_j \cdot a, \quad y_j = a \cdot y'_j. \end{aligned}$$

If the machine does not move the head, then the following rule is added.

$$\begin{aligned} & q^\star(w, x_j \cdot y_j, x_0, y_0, \dots, x_{j-1}, y_{j-1}, x'_j, y'_j, x_{j+1}, y_{j+1}, \dots, x_{k-1}, y_{k-1}) \\ & \qquad \leftarrow r^\star(w, w, x_0, y_0, \dots, x_{k-1}, y_{k-1}), \quad E(x'_j, x_j), \quad E(y'_j, y_j). \end{aligned}$$

These rules contain predicates of the form $x'_j = a \cdot x_j$. They are not in the right form. However, we can replace them for each $a \in A$ by a binary predicate $L^a(x'_j, x_j)$ with the following rules added to the grammar.

$$L^a(\varepsilon, a) \leftarrow . \qquad L^a(xc, yc) \leftarrow L^a(x, y).$$

Then $G \vdash L^a(\vec{x}, \vec{y})$ if and only if $\vec{y} = a \cdot \vec{x}$. Likewise we define $R^a(x, y)$ by

$$L^a(\varepsilon, a) \leftarrow . \qquad L^a(cx, cy) \leftarrow L^a(x, y).$$

Here, $G \vdash R^a(\vec{x}, \vec{y})$ if and only if $\vec{y} = \vec{x} \cdot a$. These rules are simple (in the technical sense).

Notice that the first argument places of the predicate is used to get rid of 'superfluous' variables. If the state q is q universal and if there are exactly p transitions with successor states r_i, $i < p$, (which do not have to be different) then $q^\star$ becomes $2k+2$–ary and we introduce p more symbols $q(i)^\star$ which are $2k+2$–ary. Now, first the following rule is introduced.

$$\begin{aligned}
&q^\star(w, w', x_0, y_0, \dots, x_{k-1}, y_{k-1}) \leftarrow \\
&\qquad q(0)^\star(w, w', x_0, y_0, \dots, x_{k-1}, y_{k-1}), \\
&\qquad q(1)^\star(w, w', x_0, y_0, \dots, x_{k-1}, y_{k-1}), \\
&\qquad \dots, \\
&\qquad q(p-1)^\star(w, w', x_0, y_0, \dots, x_{k-1}, y_{k-1}).
\end{aligned}$$

Second, we perform the same actions as above. If the transition i consists in the state q changing to r_i when reading the symbol a and if the machine moves to the left on Tape j G gets the following rule.

$$\begin{aligned}
&q(i)^\star(w, x_j \cdot y_j, x_0, y_0, \dots, x_{j-1}, y_{j-1}, x'_j, y'_j, x_{j+1}, y_{j+1}, \dots \\
&\dots, x_{k-1}, y_{k-1}) \leftarrow \\
&\qquad r_i^\star(w, w, x_0, y_0, \dots, x_{k-1}, y_{k-1}), \quad L^a(y'_j, y_j), \quad R^a(x_j, x'_j).
\end{aligned}$$

If the machine moves to the right, this rule is added instead:

$$\begin{aligned}
&q(i)^\star(w, x_j \cdot y_j, x_0, y_0, \dots, x_{j-1}, y_{j-1}, x'_j, y'_j, x_{j+1}, y_{j+1}, \dots \\
&\dots, x_{k-1}, y_{k-1}) \leftarrow \\
&\qquad r^\star(w, w, x_0, y_0, \dots, x_{k-1}, y_{k-1}), \quad R^a(x'_j, x_j), \quad L^a(y_j, y'_j).
\end{aligned}$$

If the machine does not move the head, then this rule is added.

$$\begin{aligned}
&q^\star(w, x_j \cdot y_j, x_0, y_0, \dots, x_{j-1}, y_{j-1}, x'_j, y'_j, x_{j+1}, y_{j+1}, \dots, x_{k-1}, y_{k-1}) \leftarrow \\
&\qquad r^\star(w, w, x_0, y_0, \dots \\
&\dots, x_{k-1}, y_{k-1}), \quad E(x'_j, x_j), \quad E(y'_j, y_j).
\end{aligned}$$

Again these rules are simple. If q is an accepting state, then we also take the following rule on board.

$$q^\star(w, w', x_0, y_0, \dots, x_{k-1}, y_{k-1}) \leftarrow .$$

The last rule we need is

$$S(w) \leftarrow q_0^\star(w, w, \varepsilon, w, \varepsilon, w, \ldots, \varepsilon, w).$$

This is a simple rule. For the variable w occurs to the left only once. With this definition made we have to show that $L(G) = L$. Since $L = L(T)$ it suffices to show that $L(G) = L(T)$. We have $\vec{w} \in L(T)$ if there is an $n \in \omega$ such that T moves into an accepting state from the initial configuration for $\vec{w}$. Here the initial configuration is as follows. On all tapes we have $\vec{w}$ and the read heads are to the left of the input. An end configuration is a configuration from which no further moves are possible. It is accepted if the machine is in a universal state.

We define the following mapping: we associate the configuration ζ^K to the tuple $\zeta := q^\star(\vec{w}, \vec{w}, \vec{x}_0, \vec{y}_0, \ldots, \vec{x}_{k-1}, \vec{y}_{k-1})$ where T is in state q and the read head of the ith tape on the symbol immediately following $\vec{x}_i$. Now we have:

1. $G \vdash^0 \zeta$ if and only if ζ^K is an accepting end configuration.

2. If q is existential then $\zeta \leftarrow \eta$ is an instance of a rule of G if and only if T computes η^K from ζ^K in one step.

3. If q is universal then ζ is derivable from η_i, $i < p$, in two rule steps if and only if T computes the transitions $\zeta^K \to \eta_i^K$ $(i < p)$.

Let $\vec{w} \in L(G)$. This means that $G \vdash^n S(\vec{w})$ and so

$$G \vdash^{n-1} \zeta := q_0^\star(\vec{w}, \vec{w}, \varepsilon, \vec{w}, \varepsilon, \vec{w}, \ldots, \varepsilon, \vec{w}).$$

This corresponds to the initial configuration of T for the input $\vec{w}$. We conclude from what we have said above that if $G \vdash^{n-1} \zeta$ there exists a $k \leq n$ such that T accepts ζ^K in k steps. Furthermore: if T accepts ζ^K in k steps, then $G \vdash^{2k} \zeta$. Hence we have $L(G) = L(T)$.

Theorem 5.2.13 (Groenink) *L is accepted by a simple **LMG** if and only if $L \in$ **PTIME**.*

Notes on this section. There is an alternative characterization of **PTIME**–languages. Let $\mathsf{FO}(\mathsf{LFP})$ be the expansion of first–order predicate logic (with constants for each letter and a single binary symbol $<$ in addition to equality) by the least–fixed point operator. Then teh **PTIME**–languages are exactly those that can be defined in $\mathsf{FO}(\mathsf{LFP})$. A proof may be found in (Ebbinghaus and Flum, 1995).

Exercise 172. Prove Theorem 5.2.5. *Hint.* You have to simulate the actions of a Turing machine by the grammar. Here we code the configuration by means of the string, the states by means of the predicates.

Exercise 173. Prove Theorem 5.2.10.

Exercise 174. Show that $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \in \omega\}$ is accepted by a simple 1–LMG.

Exercise 175. Let $G = \langle S, N, A, R\rangle$ be an LMG which generates L. Furthermore, let U be the language of all $\vec{x}$ whose Parikh image is that of some $\vec{y} \in L$. (In other words: U is the permutation closure of L.) Let $G^p := S^\heartsuit, N \cup \{S^\heartsuit\}, A, R^p\rangle$ where $S^\heartsuit \notin A \cup N$ and let

$$R^p := R \cup \{S^\heartsuit(x) \leftarrow S(x).; S^\heartsuit(v \cdot y \cdot x \cdot z) \leftarrow S^\heartsuit(v \cdot x \cdot y \cdot z).\}$$

Show that $L(G^p) = U$.

Exercise 176. Let L be the set of all theorems of intuitionistic logic. Write a 1–LMG that generates this set. *Hint.* You may use the Hilbert–style calculus here.

5.3 Interpreted Literal Movement Grammars

In this section we shall concern ourselves with interpreted LMGs. The basic idea behind interpreted LMGs is quite simple. Every rule is connected with a function which tells us how the meanings of the elements on the right hand side are used to construct the meaning of the item on the left. We shall give an example. The following grammar generates — as we have shown above — the language $\{\mathtt{a}^{2^n} : n \geq 0\}$.

$$S(x \cdot x) \leftarrow S(x). \qquad S(\mathtt{a}) \leftarrow .$$

We write a grammar which generates all pairs $\langle \mathtt{a}^{2^n}, n\rangle$. So, we take the number n to be the meaning of the string $\mathtt{a}^{2^n}$. For the first rule we choose the function $\lambda n.n+1$ as the meaning function and for the second the constant 0. We shall adapt the notation to the one used previously and write as follows.

$$(*) \qquad \mathtt{aaaa} : S : 2 \qquad \text{or} \qquad \langle \mathtt{aaaa}, S, 2\rangle$$

Both notations will be used concurrently. $(*)$ names a sign with exponent $\mathtt{aaaa}$ with category (or predicate) S and with meaning 2. The rules of the above grammar are written as follows:

$$\langle x \cdot x, S, n+1\rangle \leftarrow \langle x, S, n\rangle. \qquad \langle \mathtt{a}, S, 0\rangle \leftarrow .$$

This grammar is easily transformed into a sign grammar. We define a 0–ary mode $\mathtt{A}_0$ and a unary mode $\mathtt{A}_1$.

$$\begin{array}{lcl} \mathtt{A}_0 & := & \langle \mathtt{a}, S, 0\rangle, \\ \mathtt{A}_1(\langle x, S, n\rangle) & := & \langle x \cdot x, S, n+1\rangle. \end{array}$$

The structure term $\mathtt{A}_1\mathtt{A}_1\mathtt{A}_1\mathtt{A}_0$ for example defines the sign $\langle \mathtt{a}^8, S, 3\rangle$.

It seems that one can always define a sign grammar from a literal movement grammars in this way. However, this is not so. Consider the following rule, which we shall add to our present grammar.

$$\langle x\mathtt{ab}y, S, 3n\rangle \leftarrow \langle x\mathtt{aa}y, S, n\rangle$$

The problem with this rule is that the left hand side is not uniquely determined by the right hand side. For example, from $\langle \mathtt{aaaa}, S, 2\rangle$ we can derive in one step $\langle \mathtt{abaa}, S, 6\rangle$ as well as $\langle \mathtt{aaba}, S, 6\rangle$ and $\langle \mathtt{aaab}, S, 6\rangle$. We shall therefore agree on the following.

Definition 5.3.1 *Let*

$$\rho : T(t^0, t^1, \ldots, t^{p-1}) \leftarrow \qquad U_0(s_0^0, s_0^1, \ldots, s_0^{q_0-1})$$
$$U_1(s_1^0, s_1^1, \ldots, s_1^{q_1-1}) \ \ldots U_{n-1}(s_{n-1}^0, s_{n-1}^1, \ldots, s_{n-1}^{q_n-1})$$

be a rule. ρ is called ***definite*** *if for all instances of the rule the following holds: For all α, if the $(s_i^j)^\alpha$ are given, the $(t^j)^\alpha$ are uniquely determined. A literal movement grammar is called t* ***definite*** *if each of its rules is definite.*

Evidently we need for the transformation into a sign grammar that the LMG is definite. However, this is still a very general concept. Hence we shall restrict our attention to simple LMGs. They are definite, as is easily seen. These grammars have the advantage that the s_i^j are variables over strings and the t^j polynomials. We can therefore write them in λ–notation. Our grammar can therefore be specified as follows.

$$\begin{array}{lcl} \mathtt{A}_0 & := & \langle \mathtt{a}, S, 0\rangle, \\ \mathtt{A}_1 & := & \langle \lambda x.x \cdot x, S, \lambda n.n+1\rangle. \end{array}$$

In certain cases the situation is not so simple. For this specification only works if a variable of the right hand side only occurs there once. If it occurs several times, we cannot regard the t^j as polynomials using concatenation. Namely, they are partial, as is easily seen. An easy example is provided by the following rule.

$$C(x) \leftarrow A(x), B(x).$$

Intuitively, one would choose $\lambda x.x$ for the string function; however, how does one ensure that the two strings on the right hand side are equal? For suppose we were to introduce a binary mode $\mathtt{C}$.

$$\mathtt{C}(\langle \vec{x}, \alpha, X\rangle, \langle \vec{y}, \beta, Y\rangle) := \langle \mathtt{C}^\varepsilon(\vec{x}, \vec{y}), \mathtt{C}^\tau(\alpha, \beta), \mathtt{C}^\mu(X, Y)\rangle.$$

Then we must ensure that $\mathtt{C}^\varepsilon(\vec{x}, \vec{y})$ is only defined if $\vec{x} = \vec{y}$. So we agree that in addition to concatenation on A^* we also have a binary operation ι, which is defined as follows.

$$\iota(\vec{x}, \vec{y}) := \begin{cases} \vec{x}, & \text{if } \vec{x} = \vec{y}, \\ \star, & \text{otherwise.} \end{cases}$$

With the help of this operation we can transform the rule into a binary mode. Then we simply put $\mathtt{C}^{\varepsilon} := \lambda x.\lambda y.\iota(x, y)$.

We shall try out our concepts by giving a few examples. Let $\vec{x} \in \{\mathtt{L}, \mathtt{O}\}^*$ be a binary sequence. This is the binary code $n^\flat$ of a natural number n. This binary sequence we shall take as the meaning of the same number in Turing code. This is for the number n the sequence $n^\sharp := \mathtt{a}^{n+1}$. Here is a grammar for the language $\{\langle n^\sharp, S, n^\flat\rangle : n \in \omega\}$.

$$\begin{array}{lcl}
\langle x \cdot \mathtt{a}, S, n\rangle & \leftarrow & \langle x, T, n\rangle. \\
\langle x \cdot x \cdot \mathtt{a}, T, n \cdot \mathtt{L}\rangle & \leftarrow & \langle x, T, n\rangle. \\
\langle x \cdot x, T, n \cdot \mathtt{O}\rangle & \leftarrow & \langle x, T, n\rangle. \\
\langle \varepsilon, T, \varepsilon\rangle & \leftarrow & .
\end{array}$$

Notice that the meanings are likewise computed using concatenation. In place of $\lambda n.2n$ or $\lambda n.2n+1$ we therefore have $\lambda x.x \cdot \mathtt{O}$ and $\lambda x.x \cdot \mathtt{L}$.

We can also write a grammar which transforms binary codes into Turing codes. To this end we exchange exponent and meaning. So, we turn the triples of the grammar around.

A somewhat more complex example is a grammar which derives triples $\langle \vec{x} \otimes \vec{y}, S, \vec{z}\rangle$ of binary numbers where $\vec{z} = \vec{x} + \vec{y}$. (The symbol $\otimes$ will be explained later. At the moment it serves to separate $\vec{x}$

from $\vec{y}$.)

$$\begin{array}{lcl}
\langle \mathtt{0}x \otimes y, S, z\rangle & \leftarrow & \langle x \otimes y, S, z\rangle \mid \langle x \otimes y, A, z\rangle \\
\langle x \otimes \mathtt{0}y, S, z\rangle & \leftarrow & \langle x \otimes y, S, z\rangle \mid \langle x \otimes y, A, \rangle \\
\langle x \otimes y, S, \mathtt{L}z\rangle & \leftarrow & \langle x \otimes y, U, z\rangle \\
\langle \mathtt{0}x \otimes y, S, \mathtt{L}z\rangle & \leftarrow & \langle x \otimes y, U, z\rangle \\
\langle x \otimes \mathtt{0}y, S, \mathtt{L}z\rangle & \leftarrow & \langle x \otimes y, U, z\rangle \\
\langle \mathtt{0}x \otimes \mathtt{0}y, S, \mathtt{L}z\rangle & \leftarrow & \langle x \otimes y, U, z\rangle \\
\\
\langle \mathtt{0}x \otimes \mathtt{0}y, A, \mathtt{L}z\rangle & \leftarrow & \langle x \otimes y, U, z\rangle \mid \langle x \otimes y, A, z\rangle \\
\\
\langle \mathtt{0}x \otimes \mathtt{L}y, U, \mathtt{0}z\rangle & \leftarrow & \langle x \otimes y, U, z\rangle \\
\langle \mathtt{0}x \otimes \mathtt{L}y, A, \mathtt{L}z\rangle & \leftarrow & \langle x \otimes y, A, z\rangle \\
\\
\langle \mathtt{L}x \otimes \mathtt{0}y, U, \mathtt{0}z\rangle & \leftarrow & \langle x \otimes y, U, z\rangle \\
\langle \mathtt{L}x \otimes \mathtt{0}y, A, \mathtt{L}z\rangle & \leftarrow & \langle x \otimes y, A, z\rangle \\
\\
\langle \mathtt{L}x \otimes \mathtt{L}y, U, \mathtt{L}z\rangle & \leftarrow & \langle x \otimes y, U, z\rangle \mid \langle x \otimes y, A, z\rangle \\
\\
\langle \varepsilon \otimes \varepsilon, A, \varepsilon\rangle & \leftarrow & .
\end{array}$$

Now let us return to the specification of interpreted LMGs. First of all we shall ask how LMGs can be interpreted to become sign grammars. To this end we have to reconsider our notion of an exponent. Up to now we have assumed that exponents are strings. Now we have to assume that they are sequences of strings (we say rather vectors of strings, since strings are themselves sequences). This motivates the following definition.

Definition 5.3.2 *Let A be a finite set. We denote by $V(A) := \bigcup_{k<\omega}(A^*)^k$ the set of vectors of strings over A. Furthermore, let $F^V := \{\varepsilon, 0, \cdot, \otimes, \rhd, \lhd, \zeta, \iota\}$, $\Omega(\cdot) = \Omega(\otimes) = \Omega(\iota) = 2$, $\Omega(\rhd) = \Omega(\lhd) = \Omega(\zeta) = 1$; $\Omega(\varepsilon) = \Omega(0) = 0$. Here, the following is assumed to hold. (Variables for strings are denoted by vector arrows, while $\mathfrak{x}$, $\mathfrak{y}$ and $\mathfrak{z}$ range over $V(A)$.)*

$\rhd$ $\mathfrak{z} \otimes (\mathfrak{y} \otimes \mathfrak{z}) = (\mathfrak{z} \otimes \mathfrak{y}) \otimes \mathfrak{z}$

- $\triangleright$ $0 = \langle\,\rangle$ *is the empty sequence.*
- $\triangleright$ $\cdot$ *is the usual concatenation of strings, so it is not defined on vectors of length* $\neq 1$.
- $\triangleright$ ε *is the empty string.*
- $\triangleright$ $\triangleright \bigotimes_{i<m} \vec{x} = \bigotimes_{i<m-1} \vec{x}_i$, $\triangleleft \bigotimes_{i<m} \vec{x}_i = \bigotimes_{0<i<m} \vec{x}_i$.
- $\triangleright$ $\zeta(\mathfrak{x}) = \star$ *if* $\mathfrak{x}$ *is not a string;* $\zeta(\vec{x}) = \vec{x}$ *otherwise.*
- $\triangleright$ $\iota(\mathfrak{x}, \mathfrak{y}) = \mathfrak{x}$ *if* $\mathfrak{x} = \mathfrak{y}$ *and* $\iota(\mathfrak{x}, \mathfrak{y}) = \star$ *otherwise.*

The resulting (partial) algebra is called the ***vector algebra over*** A *and is denoted by* $\mathfrak{V}(A)$.

In this algebra the following laws hold among other.

$$\begin{array}{lcl}
\mathfrak{x} \otimes 0 & = & \mathfrak{x} \\
0 \otimes \mathfrak{x} & = & \mathfrak{x} \\
\mathfrak{x} \otimes (\mathfrak{y} \otimes \mathfrak{z}) & = & (\mathfrak{x} \otimes \mathfrak{y}) \otimes \mathfrak{z} \\
\triangleright(\mathfrak{x} \otimes \zeta(\mathfrak{y})) & = & \mathfrak{x} \\
\triangleleft(\zeta(\mathfrak{y}) \otimes \mathfrak{x}) & = & \mathfrak{x}
\end{array}$$

The fourth and fifth equation hold under the condition that $\zeta(\mathfrak{y})$ is defined. A vector $\mathfrak{x}$ has length m if $\zeta(\triangleright^{m-1}\mathfrak{x})$ is defined. In this case $\zeta(\triangleright^{m-(i+1)} \triangleleft^i \mathfrak{x})$ is defined for all $i < m$ and they are the projection functions. Now we have:

$$\mathfrak{x} = \triangleright^{m-1}\mathfrak{x} \otimes \triangleright^{m-2} \triangleleft \mathfrak{x} \otimes \ldots \otimes \triangleright \triangleleft^{m-2} \mathfrak{x} \otimes \triangleleft^{m-1}\mathfrak{x}.$$

All polynomial functions that appear in the sequel can be defined in this algebra. The basis is the following theorem.

Theorem 5.3.3 *Let* $p : (A^*)^m \to A^*$ *be a function which is a polynomial in* $\cdot$ *and* ι. *Then there exists a vector polynomial* $\mathfrak{q} : V(A) \to V(A)$ *such that*

1. $\mathfrak{q}(\mathfrak{x})$ *is defined only if* $\mathfrak{x} \in (A^*)^m$.

2. If $\mathfrak{x} \in (A^*)^m$ *and* $\mathfrak{x} = \langle \vec{x}_i : i < m\rangle$ *then* $\mathfrak{q}(\mathfrak{x}) = p(\vec{x}_0, \ldots, \vec{x}_{m-1})$.

Proof. Let p be given. We assume that one of the variables appears at least once. (Otherwise $p = \varepsilon$ and then we put $q := \varepsilon$.) Let q arise from p by replacement of x_i by $z(\rhd^{m-(i+1)} \lhd^i \mathfrak{x})$, for all $i < m$. This defines q. (It is well defined, for the symbols ε, $\cdot$, ι are in the signature F^V.) Let now $\mathfrak{x}$ be given. As remarked above, q is defined on $\mathfrak{x}$ only if $\mathfrak{x}$ has length m. In this case $\mathfrak{x} = \langle \vec{x}_i : i < n\rangle$ for certain $\vec{x}_i$, and we have $\vec{x}_i = z(\rhd^{m-(i+1)} \lhd^i \mathfrak{x})$. Since the symbols ε, $\cdot$ and ι coincide on the strings in both algebras (that of the strings and that of the vectors) we have $q(\mathfrak{x}) = q(\vec{x}_0, \ldots, \vec{x}_{m-1})$. ☐

That $p : (A^*)^m \to (A^*)^n$ is a polynomial function means that there exist polynomials p_i, $i < n$, such that

$$p(\vec{x}_0, \ldots, \vec{x}_{m-1}) = \langle p_i(\vec{x}_0, \ldots, \vec{x}_{m-1}) : i < n\rangle.$$

We can therefore replace the polynomials on strings by polynomials over vectors of strings. Thise simplifies the presentation of LMGs considerably. We can now write down a rule as follows.

$$\langle q(\mathfrak{x}_0, \ldots, \mathfrak{x}_{m-1}), A, f(X_0, \ldots, X_{m-1})\rangle \leftarrow \langle \mathfrak{x}_0, B_0, X_0\rangle \quad \ldots \quad \langle \mathfrak{x}_{m-1}, B_{m-1}, X_{m-1}\rangle.$$

We shall make a further step and consider LMGs as categorial grammars. To this end we shall first go over to Chomsky normal form. This actually brings up a surprise. For there are k–LMGs for which no k–LMG in Chomsky normal form can be produced (see the exercises). However, there exists a k'–LMG in Chomsky normal form for a certain effectively determinable $k' \leq \pi k$, where π is the maximal productivity of a rule. Namely, look at a rule. We introduce new symbols Z_i, $i < m - 2$, and replace this rule by the following rules.

$$\begin{array}{lcl}
\langle \mathfrak{x}_0 \otimes \mathfrak{x}_1, Z_0, X_0 \times X_1\rangle & \leftarrow & \langle \mathfrak{x}_0, B_0, X_0\rangle, \quad \langle \mathfrak{x}_1, B_1, X_1\rangle. \\
\langle \mathfrak{y}_0 \otimes \mathfrak{x}_2, Z_1, Y_0 \times X_2\rangle & \leftarrow & \langle \mathfrak{y}_0, Z_0, Y_0\rangle, \quad \langle \mathfrak{x}_2, B_2, X_2\rangle. \\
 & \ldots & \\
\langle \mathfrak{y}_{m-4} \otimes \mathfrak{x}_{m-2}, Z_{m-3}, Y_{m-4} \times X_{m-2}\rangle & \leftarrow & \langle \mathfrak{y}_{m-4}, Z_{m-4}, Y_{m-4}\rangle, \\
 & & \quad \langle \mathfrak{x}_{m-2}, B_{m-2}, X_{m-2}\rangle. \\
\langle q^*(\mathfrak{y}_{m-3} \otimes \mathfrak{x}_{m-1}), A, f^*(Y_{m-3} \times X_{m-1})\rangle & \leftarrow & \langle \mathfrak{y}_{m-3}, Z_{m-3}, Y_{m-3}\rangle, \\
 & & \quad \langle \mathfrak{x}_{m-1}, B_{m-1}, X_{m-1}\rangle.
\end{array}$$

Here q^* and f^* are chosen in such a way that

$$\begin{array}{rcl} q^*(\mathfrak{x}_0 \otimes \ldots \otimes \mathfrak{x}_{m-1}) & = & q(\mathfrak{x}_0, \ldots, \mathfrak{x}_{m-1}), \\ f^*(X_0 \times \ldots \times X_{m-1}) & = & f(X_0, \ldots, X_{m-1}). \end{array}$$

It is not hard to see how to define the functions by polynomials. Hence, in the sequel we may assume that we have at most binary branching rules. 0–ary rules are the terminal rules. A unary rule has the following form

$$\langle q(\mathfrak{x}), C, f(X) \rangle \leftarrow \langle \mathfrak{x}, A, X \rangle \ .$$

We keep the sign $\langle \mathfrak{x}, A, X \rangle$ and introduce a new sign $\mathtt{Z}_\rho$ which has the following form.

$$\mathtt{Z}_\rho := \langle \lambda \mathfrak{x}.q(\mathfrak{x}), C/A, \lambda x.f(x) \rangle \ .$$

We agree that there is only one binary mode $\mathtt{C}$ which is defined thus:

$$\mathtt{C}(\langle p, A, X \rangle, \langle q, B, Y \rangle) := \langle p(q), A \cdot B, (XY) \rangle \ .$$

This is exactly the scheme of application in categorial grammar. One difference remains. The polynomial p is not necessarily the concatenation. Furthermore, we do not have to distinguish between two modes, since we have the possibility of putting p to be $\lambda x.x \cdot y$ or $\lambda x.y \cdot x$. Application has in this way become independent of the accidental order. Many more operations can be put here, for example reduplication. The grammar that we have mentioned at the beginning of the section is defined by the following two modes.

$$\begin{array}{rcl} \mathtt{D}_0 & := & \langle \mathtt{a}, S, 0 \rangle \\ \mathtt{D}_1 & := & \langle \lambda x.x \cdot x, S/S, \lambda n.n + 1 \rangle \end{array}$$

To the previous structure term $\mathtt{A}_1\mathtt{A}_1\mathtt{A}_1\mathtt{A}_0$ now corresponds the structure term

$$\mathtt{C}\mathtt{D}_1\mathtt{C}\mathtt{D}_1\mathtt{C}\mathtt{D}_1\mathtt{D}_0$$

In this way the grammar has become an **AB**–grammar, with one exception: the treatment of strings must be explicitly defined.

The binary rules remain. A binary rule has the following form.

$$\langle q(\mathfrak{x}, \mathfrak{y}), C, f(X, Y)\rangle \leftarrow \langle \mathfrak{x}, A, X\rangle, \quad \langle \mathfrak{y}, B, Y\rangle$$

We keep the sign on the right hand side and introduce a new sign.

$$\mathtt{Z}_\rho := \langle \lambda\mathfrak{y}.\lambda\mathfrak{x}.q(\mathfrak{x}, \mathfrak{y}), (C/A)/B, \lambda y.\lambda x.f(x, y)\rangle$$

Exercise 177. Show that for any $k > 1$ there are simple k–LMGs G with branching number 3 such that for no simple k–LMG H with branching number 2, $L(G) = L(H)$.

Exercise 178. Here are some facts from Arabic. In Arabic a root typically consists of three consonants. A popular example is `ktb`, which means roughly *to write*. A root need not be a verb, `klb` means *dog*. There are also roots with four letters, such as `drhm` (from Greek *Drachme*), which names a numismatic unit. From this root one forms so–called *Binyanim*, roughly translated as word classes, by inserting vowels or changing the consonantism of the root. Here are some examples of verbs derived from the root `ktb`.

(5.3.1)	I	`katab`	to write
	II	`kattab`	to make write
	III	`kaatab`	to correspond
	VI	`takaatab`	to write to each other
	VIII	`ktatab`	to write, to be inscribed

Of these forms we can in turn form verbal forms in different tenses and voices.

(5.3.2)		Perf. Act.	Perf. Pass.	Imp. Act.	Imperf. Pass
	I	`katab`	`kutib`	`aktub`	`uktab`
	II	`kattab`	`kuttib`	`ukattib`	`ukattab`
	III	`kaatab`	`kuutib`	`ukaatib`	`ukaatab`
	VI	`takaatab`	`tukuutib`	`atakattab`	`utakattab`
	VIII	`ktatab`	`ktutib`	`aktatib`	`uktatab`

We have only shown the transparent cases, there are other classes whose forms are not so regular. Write an interpreted LMG that generates these forms. For the meanings, simply assume unary operators, for example caus' for II, pass' for passive, and so on.

Exercise 179. In Chinese a yes–no–question is formed as follows. A simple assertive sentence has the form (5.3.3) and the corresponding negative sentence the form (5.3.4). Chinese is an SVO–language, and so the verb phrase follows the subject. In between these two the word `bu` inserts itself. (We do not write tones.)

(5.3.3)	`Ta`	`zai`	`jia.`	
	He/She/It	at	home	
(5.3.4)	`Ta`	`bu`	`zai`	`jia.`
	He/She/It	not	at	home

The yes–no–question is formed by concatenation the subject phrase with the positive verb phrase and directly after that the negative verb phrase.

(5.3.5) `Ta zai jia bu zai jia?`
Is he/she/it at home?

As Radzinski (1990) argues, the verb phrases have to be completely identical. For example, (5.3.6) is grammatical, (5.3.7) is ungrammatical. However, (5.3.8) is again grammatical and means roughly what (5.3.6) means.

(5.3.6)	`Ni`	`xihuan`	`ta-de`	`chenshan`	`bu`	`xihuan`
	You	like	his	shirt	not	like
	`ta-de`	`chenshan?`				
	his	shirt?				
	Do you like his shirt?					
(5.3.7)	*`Ni`	`xihuan`	`ta-de`	`bu`	`xihuan`	`ta-de`
	You	like	his	not	like	his
	`chenshan?`					
	shirt?					
(5.3.8)	`Ni`	`xihuan`	`bu`	`xihuan`	`ta-de`	`chenshan?`
	You	like	not	like	his	shirt?

Write an interpreted LMG generating these examples. Use $? : \varphi$ to denote the question, whether or not φ is the case.

5.4 Discontinuity

In this section we shall study a very important type of grammars, the so called **Linear Context–Free Rewrite Systems** — **LCFRS** for short. These have been studied in detail in (Seki *et al.*, 1991). We call them simply linear LMGs (which is a difference only in name).

Definition 5.4.1 *A k–LMG is called* ***linear*** *if it is a simple k–LMG and every rule which is not 0–ary is downward nondeleting and downward linear, while 0–ary rules have the form $X(a) \leftarrow .$ with $a \in A$.*

In other words, if we have a rule of this form

$$A(t_0, \ldots, t_{k-1}) \leftarrow B_0(s_0^0, \ldots, s_{k-1}^0), \ldots, B_{n-1}(s_0^{n-1}, \ldots, s_0^{n-1}).$$

then for every $i < k$ and $j < n$ $s_i^j = x_i^j$, and $x_j^i = x_{j'}^{i'}$ implies $i = i'$ and $j = j'$. Finally, $\prod_{i<k} t_i$ is a product of these variables (so it contains no occurrences of constants) and every variable occurs exactly once. In case $k = 1$ we get exactly the context free

grammars, though in somewhat disguised form. For now a rule is of the form

$$A(\prod_{i<n} x_{\pi(n)}) \leftarrow B_0(x_0), B_1(x_1), \dots, B(x_{n-1})$$

where π is a permutation of the numbers $< n$. If $\rho := \pi^{-1}$ is the permutation inverse to π we can write the rule in this way.

$$A(\prod_{i<n} x_i) \leftarrow B_{\rho(0)}(x_0), B_{\rho(1)}(x_1), \dots, B_{\rho(0)}(x_{n-1}).$$

(To this end we replace the variable x_i by the variable $x_{\rho(i)}$ for every i. After that we permute the B_i. The order of the conjuncts is anyway insignificant in an LMG.) This is as one can easily see exactly the form of a context free rule. For we have

$$\prod_{i<n} x_i = x_0 x_1 \dots x_{n-1} .$$

This rule says therefore that if we have constituents $\vec{u}_i$ of type $B_{\rho(i)}$ for $i < n$, then $\prod_{i<n} \vec{u}_i$ is a constituent of type A.

The next case is $k = 2$. This defines a class of grammars which have been introduced before, using a somewhat different notation and which have been shown to be powerful enough to generate non context free languages such as Swiss German. In linear 2–LMGs we may have rules of this kind.

$$\begin{array}{rcl} A(x_1x_2, y_1y_2) & \leftarrow & B(x_1, y_1), C(x_2, y_2). \\ A(x_2x_1, y_1y_2) & \leftarrow & B(x_1, y_1), C(x_2, y_2). \\ A(y_1x_1y_2, x_2) & \leftarrow & B(x_1, y_1), C(x_2, y_2). \\ A(x_1, y_1x_2y_2) & \leftarrow & B(x_1, y_1), C(x_2, y_2). \end{array}$$

The following rules however are excluded.

$$\begin{array}{lcl} A(x_1, y_1y_2) & \leftarrow & B(x_1, y_1), C(x_2, y_2). \\ A(x_2x_2x_1, y_1y_2) & \leftarrow & B(x_1, y_1), C(x_2, y_2). \end{array}$$

The first is upward deleting, the second not linear. The language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n \in \omega\}$ can be generated by a linear 2–LMG, the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n\mathtt{e}^n : n \in \omega\}$ however cannot. The second fact follows from a generalized form of the Pumping Lemma which shall be shown further below. For the first claim we give the following grammar.

$$\begin{array}{lcl}
\mathtt{S}(y_0x_0y_1, z_0x_1z_1) & \leftarrow & \mathtt{S}(x_0, x_1), \mathtt{A}(y_0, y_1), \mathtt{B}(z_0, z_1). \\
\mathtt{S}(\varepsilon, \varepsilon) & \leftarrow & . \\
\mathtt{A}(\mathtt{a}, \mathtt{b}) & \leftarrow & . \\
\mathtt{B}(\mathtt{c}, \mathtt{d}) & \leftarrow & .
\end{array}$$

This shows that 2–linear LMGs are strictly stronger than context free grammars. As a further example we shall dig out our fragment of Swiss German from Section 2.6. We define the following grammar.

$$\begin{array}{lcl}
\mathtt{NPa}(\mathtt{d'chind}) & \leftarrow & . \\
\mathtt{NPd}(\mathtt{em\ Hans}) & \leftarrow & . \\
\mathtt{NPs}(\mathtt{Jan}) & \leftarrow & . \\
\mathtt{NPa}(\mathtt{es\ huus}) & \leftarrow & . \\
\mathtt{Vdr}(\mathtt{hälfe}) & \leftarrow & . \\
\mathtt{Vf}(\mathtt{lönd}) & \leftarrow & . \\
\mathtt{Var}(\mathtt{laa}) & \leftarrow & . \\
\mathtt{Van}(\mathtt{aastriche}) & \leftarrow & . \\
\mathtt{C}(\mathtt{das}) & \leftarrow & . \\
\mathtt{NPs}(\mathtt{mer}) & \leftarrow & . \\
\mathtt{Vc}(\mathtt{säit}) & \leftarrow & . \\
\mathtt{S}(xy) & \leftarrow & \mathtt{NPs}(x), \mathtt{VP}(y). \\
\mathtt{VP}(xy) & \leftarrow & \mathtt{Vc}(x), \mathtt{CP}(y). \\
\mathtt{CP}(xyz_0z_1u) & \leftarrow & \mathtt{C}(x), \mathtt{NPs}(y), \mathtt{VI}(z_0, z_1), \mathtt{Vf}(u). \\
\mathtt{VI}(xz_0, yz_1) & \leftarrow & \mathtt{NPa}(x), \mathtt{Var}(y), \mathtt{VI}(z_0, z_1). \\
\mathtt{VI}(xz_0, yz_1) & \leftarrow & \mathtt{NPd}(x), \mathtt{Vdr}(y), \mathtt{VI}(z_0, z_1). \\
\mathtt{VI}(x, y) & \leftarrow & \mathtt{NPa}(x), \mathtt{Van}(y).
\end{array}$$

This grammar is pretty realistic also with respect to the constituent structure, of which more below. For simplicity we have varied the arities of the predicates. Notice in particular the last two rules. They are the real motor of the Swiss German infinitive constructions. For we can derive the following.

$$\begin{array}{l}
\mathtt{VI(d'chind}\ z_0, \mathtt{laa}\ z_1) \leftarrow \mathtt{VI}(z_0, z_1). \\
\mathtt{VI(em\ Hans}\ z_0, \mathtt{h\ddot{a}lfe}\ z_1) \leftarrow \mathtt{VI}(z_0, z_1). \\
\mathtt{VI(d'chind, aastriche)} \leftarrow . \\
\mathtt{VI(es\ huus, aastriche)} \leftarrow .
\end{array}$$

However, we do not have

$$\mathtt{VI(em\ Hans, laa)} \leftarrow .$$

From this deduce inductively that

$$\begin{array}{l}
\mathtt{VI(d'chind\ em\ Hans}\ z_0, \mathtt{laa\ h\ddot{a}lfe}\ z_1) \leftarrow \mathtt{VI}(z_0, z_1). \\
\mathtt{VI(em\ Hans\ es\ huus}\ z_0, \mathtt{h\ddot{a}lfe\ laa}\ z_1) \leftarrow \mathtt{VI}(z_0, z_1).
\end{array}$$

The sentences of Swiss German as reported in Section 2.6 are derivable and some further sentences, which are all grammatical.

Linear LMGs can also be characterized by the vector polynomials which occur in the rules. We shall illustrate this by way of example with linear 2–LMGs and here only for the at most binary rules. We shall begin with the unary rules. They can make use of these vector polynomials.

$$\begin{array}{lcl}
\mathfrak{i}(x_0, x_1) & := & \langle x_0, x_1 \rangle, \\
\mathfrak{p}_X(x_0, x_1) & := & \langle x_1, x_0 \rangle, \\
\mathfrak{p}_F(x_0, x_1) & := & \langle x_0 x_1, \varepsilon \rangle, \\
\mathfrak{p}_G(x_0, x_1) & := & \langle x_1 x_0, \varepsilon \rangle, \\
\mathfrak{p}_H(x_0, x_1) & := & \langle \varepsilon, x_0 x_1 \rangle, \\
\mathfrak{p}_K(x_0, x_1) & := & \langle \varepsilon, x_1 x_0 \rangle.
\end{array}$$

Then the following holds.

$$\begin{array}{lcl}
\mathfrak{p}_X(\mathfrak{p}_X(x_0, x_1)) & = & \mathfrak{i}(x_0, x_1), \\
\mathfrak{p}_G(x_0, x_1) & = & \mathfrak{p}_F(\mathfrak{p}_X(x_0, x_1)), \\
\mathfrak{p}_K(x_0, x_1) & = & \mathfrak{p}_H(\mathfrak{p}_X(x_0, x_1)).
\end{array}$$

This means that one has $\mathfrak{p}_G$ at one's disposal if one also has $\mathfrak{p}_X$ and $\mathfrak{p}_F$ and that one has p_F if one also has $\mathfrak{p}_X$ and $\mathfrak{p}_G$ and so on. With binary rules already the situation gets quite complicated. Therefore we shall assume that we have all unary polynomials. A binary vector polynomial is of the form $\langle p_0(x_0, x_1, y_0, y_1), p_1(x_0, x_1, y_0, y_1)\rangle$ such that $q := p_0 \cdot p_1$ is linear. Then we can assume that in $q(x_0, x_1, y_0, y_1)$ x_0 always appears to the left of x_1 and y_0 to the left of y_1. Further, one may also assume that x_0 is to the left of y_0 (otherwise exchange the x_i with the y_i). p_0 and p_1 result by cutting q in two halves. So we only need to list q; for the vector polynomials there exist in total 5 (!) possibilities. After simplification this gives the following polynomials.

$$\begin{array}{rcl} q_C(x_0, x_1, y_0, y_1) & := & x_0x_1y_0y_1 \\ q_W(x_0, x_1, y_0, y_1) & := & x_0y_0x_1y_1 \\ q_Z(x_0, x_1, y_0, y_1) & := & x_0y_0y_1x_1 \end{array}$$

Let us take a look at q_W. From this polynomial we get the followng vector polynomials.

$$\begin{array}{rcl} \mathfrak{q}_{W0}(x_0, x_1, y_0y_1) & := & \langle \varepsilon, x_0y_0x_1y_1\rangle \\ \mathfrak{q}_{W1}(x_0, x_1, y_0y_1) & := & \langle x_0, y_0x_1y_1\rangle \\ \mathfrak{q}_{W2}(x_0, x_1, y_0y_1) & := & \langle x_0y_0, x_1y_1\rangle \\ \mathfrak{q}_{W3}(x_0, x_1, y_0y_1) & := & \langle x_0y_0x_1, y_1\rangle \\ \mathfrak{q}_{W4}(x_0, x_1, y_0y_1) & := & \langle x_0y_0x_1y_1, \varepsilon\rangle \end{array}$$

We shall let the reader show in the exercises how the polynomials in the grammars behave. We say that a linear LMG has polynomial basis Q if in the rules of this grammar only vector polynomials from Q have been used. It is easy to see that if $\mathfrak{q}$ is a polynomial that can be presented by means of polynomials from Q, then one may add $\mathfrak{q}$ to Q without changing the generative capacity. Notice also that it does not matter if the polynomial contains constants. If we have, for example,

$$\mathtt{X}(\mathtt{a}x\mathtt{bc}y) \leftarrow \mathtt{X}(x), \mathtt{Z}(y).$$

we can replace this by the following rules.

$$\begin{array}{lcl}
\mathtt{X}(uxvwy) & \leftarrow & \mathtt{A}(u), \mathtt{X}(x), \mathtt{B}(v), \mathtt{C}(w), \mathtt{Z}(y). \\
\mathtt{A}(\mathtt{a}) & \leftarrow & . \\
\mathtt{B}(\mathtt{b}) & \leftarrow & . \\
\mathtt{C}(\mathtt{c}) & \leftarrow & .
\end{array}$$

This is advantageous in proofs. We bring to the attention of the reader some properties of languages that can be generated by linear LMGs.

Proposition 5.4.2 (Vijay–Shanker & Weir & Joshi) *Let G be a linear k–LMG. Then $L(G)$ is semilinear.*

A special type of linear LMGs are the so called head grammars. These grammars have been introduced by Carl Pollard in (Pollard, 1984). The strings that are manipulated are of the form $\vec{x}a\vec{y}$ where $\vec{x}$ and $\vec{y}$ are strings and $a \in A$. One speaks in this connection of a in the string as the distinguished **head**. This head is marked by underlining it. Strings containing an underlined occurrence of a letter are called **marked**. The following rules for manipulating marked strings are now admissible.

$$\begin{array}{lcl}
h_{C1}(\vec{v}\underline{a}\vec{w}, \vec{y}\underline{b}\vec{z}) & := & \vec{v}\underline{a}\vec{w}\vec{y}b\vec{z} \\
h_{C2}(\vec{v}\underline{a}\vec{w}, \vec{y}\underline{b}\vec{z}) & := & \vec{v}a\vec{w}\vec{y}\underline{b}\vec{z} \\
h_{L1}(\vec{v}\underline{a}\vec{w}, \vec{y}\underline{b}\vec{z}) & := & \vec{v}\underline{a}\vec{y}b\vec{z}\vec{w} \\
h_{L2}(\vec{v}\underline{a}\vec{w}, \vec{y}\underline{b}\vec{z}) & := & \vec{v}a\vec{y}\underline{b}\vec{z}\vec{w} \\
h_{R1}(\vec{v}\underline{a}\vec{w}, \vec{y}\underline{b}\vec{z}) & := & \vec{v}\vec{y}b\vec{z}\underline{a}\vec{w} \\
h_{R2}(\vec{v}\underline{a}\vec{w}, \vec{y}\underline{b}\vec{z}) & := & \vec{v}\vec{y}\underline{b}\vec{z}a\vec{w}
\end{array}$$

Since one wanted to admit the head being empty the format has been changed slightly. In place of marked strings one takes 2–vectors of strings. The marked head is the comma. This leads to the following definition.

Definition 5.4.3 *A **head grammar** is a linear 2–LMG with the following polynomial basis.*

$$\begin{array}{rcl}
\mathfrak{p}_{C1}(\langle x_0, x_1\rangle, \langle y_0, y_1\rangle) & := & \langle x_0, x_1y_0y_1\rangle \\
\mathfrak{p}_{C2}(\langle x_0, x_1\rangle, \langle y_0, y_1\rangle) & := & \langle x_0x_1y_0, y_1\rangle \\
\mathfrak{p}_{L1}(\langle x_0, x_1\rangle, \langle y_0, y_1\rangle) & := & \langle x_0, y_0y_1x_1\rangle \\
\mathfrak{p}_{L2}(\langle x_0, x_1\rangle, \langle y_0, y_1\rangle) & := & \langle x_0y_0y_1, x_1\rangle \\
\mathfrak{p}_{R1}(\langle x_0, x_1\rangle, \langle y_0, y_1\rangle) & := & \langle x_0y_0y_1, x_1\rangle \\
\mathfrak{p}_{R2}(\langle x_0, x_1\rangle, \langle y_0, y_1\rangle) & := & \langle x_0y_0, y_1x_1\rangle
\end{array}$$

Notice that in this case there are *no* extra unary polynomials. However, some of them can be produced by feeding empty material. These are exactly the polynomials ι, $\mathfrak{p}_F$ and $\mathfrak{p}_H$. The others cannot be produced, since the order of the component strings must always be respected. For example, one has

$$\mathfrak{p}_{C2}(\langle x_0, x_1\rangle, \langle y_0, y_1\rangle) = \langle x_0x_1, \varepsilon\rangle = \mathfrak{p}_F(x_0, x_1)$$

Now, to begin with the analysis of discontinuity, we shall concern ourselves with the description of the structures that are implicit in LCFRSs. To this end, let G be a k–linear grammar. If $G \vdash A(\vec{x}_0, \dots, \vec{x}_{k-1})$ then this means that the k–tuple $\langle \vec{x}_i : i < k\rangle$ can be a constituent of category A. To make this idea precise we shall introduce some more terminology. A **derivation** in an LMG is a sequence

$$\Gamma = \langle (n_i, \rho_i, H_i) : i < p\rangle$$

where every H_i, $i < p$, is a finite sequence of literals $n_i < |H_i|$, ρ_i the name of a rule, and H_{i+1} results from H_i by applying ρ_{i+1} to the n_ith literal. So, if $H_i = \langle L_j : j < r\rangle$ then $H_{i+1} = \langle L'_j : j < r + p(\rho_{i+1})\rangle$. (Recall the definition of the productivity of a rule.) Here, we must have that

$$L_{n_{i+1}} \leftarrow L'_{n_{i+1}} \quad L'_{n_{i+1}+1} \dots L'_{n_{i+1}+p(\rho_{i+1})}$$

is an instance of a rule (not necessarily a ground instance!) and

$$L_j = L'_j \quad (j < n_{i+1}), \qquad L_j = L'_{j+p(\rho_{i+1}} \quad (j > n_{i+1}) .$$

In this connection L_n is called the **main literal** of H_i. If L_{p-1} is a sequence of instances of 0–ary rules, then Γ is a **complete derivation**. An example shall illustrate this. Look at the grammar displayed above, which generates the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n \in \omega\}$. We reformulate this grammar slightly. The following grammar is called $G^{\heartsuit}$.

$$\begin{array}{lcllll}
\rho_0 & : & \mathtt{S}(y_0x_0y_1, z_0x_1z_1) & \leftarrow & \mathtt{S}(x_0, x_1) \quad \mathtt{X}(y_0, y_1), \mathtt{Y}(z_0, z_1). \\
\rho_1 & : & \mathtt{S}(\varepsilon, \varepsilon) & \leftarrow & . \\
\rho_2 & : & \mathtt{X}(x, y) & \leftarrow & \mathtt{A}(x), \mathtt{B}(y). \\
\rho_3 & : & \mathtt{A}(\mathtt{a}) & \leftarrow & . \\
\rho_4 & : & \mathtt{B}(\mathtt{b}) & \leftarrow & . \\
\rho_5 & : & \mathtt{Y}(x, y) & \leftarrow & \mathtt{C}(x), \mathtt{D}(y). \\
\rho_6 & : & \mathtt{C}(\mathtt{c}) & \leftarrow & . \\
\rho_7 & : & \mathtt{D}(\mathtt{d}) & \leftarrow & .
\end{array}$$

The following is a complete derivation.

$$\begin{array}{l}
\langle(0, \rho_0, \langle\mathtt{S}(\mathtt{aabb}, \mathtt{ccdd})\rangle), \\
\;(0, \rho_0, \langle\mathtt{S}(\mathtt{ab}, \mathtt{cd}), \mathtt{X}(\mathtt{a}, \mathtt{b}), \mathtt{Y}(\mathtt{c}, \mathtt{d})\rangle), \\
\;(1, \rho_2, \langle\mathtt{S}(\mathtt{ab}, \mathtt{cd}), \mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{Y}(\mathtt{c}, \mathtt{d})\rangle), \\
\;(3, \rho_5, \langle\mathtt{S}(\mathtt{ab}, \mathtt{cd}), \mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{C}(\mathtt{c}), \mathtt{D}(\mathtt{d})\rangle), \\
\;(0, \rho_0, \langle\mathtt{S}(\varepsilon, \varepsilon), \mathtt{X}(\mathtt{a}, \mathtt{b}), \mathtt{Y}(\mathtt{c}, \mathtt{d}), \mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{C}(\mathtt{c}), \mathtt{D}(\mathtt{d})\rangle), \\
\;(1, \rho_2, \langle\mathtt{S}(\varepsilon, \varepsilon), \mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{Y}(\mathtt{c}, \mathtt{d}), \mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{C}(\mathtt{c}), \mathtt{D}(\mathtt{d})\rangle), \\
\;(3, \rho_5, \langle\mathtt{S}(\varepsilon, \varepsilon), \mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{C}(\mathtt{c}), \mathtt{D}(\mathtt{d}), \mathtt{A}(\mathtt{a}), \mathtt{B}(\mathtt{b}), \mathtt{C}(\mathtt{c}), \mathtt{D}(\mathtt{d})\rangle)\rangle
\end{array}$$

Also the following is a derivation.

$$\langle(0, \rho_0, \langle\mathtt{S}(xy, zu)\rangle, (0, \rho_0, \langle\mathtt{S}(\varepsilon, \varepsilon), \mathtt{X}(x, z), \mathtt{Y}(y, u)\rangle)\rangle.$$

(This simply is an instance of a rule applied once. Notice that 0 and ρ_0 in the first member of the sequence are without meaning. They are added only to keep the definition uniform.) If L_0 consists of a single literal we say that Γ is a **derivation of** L_0. Now let G be k–linear; further, let us assume that 0–ary rules are of the form $X(a)$, $a \in A$. Let Γ be given and Γ a derivation of a ground instance of a literal. Then wherever in Γ we find a $X(a)$ we replace a variable in place of a; we do this for every such occurrence, each time taking a different variable. Then we replace from right to left the occurrences of the letters by the corresponding variables so that we obtain a derivation Γ'.

$$\begin{aligned}
&\langle(0, \rho_0, \langle \mathtt{S}(x_4x_0x_1x_5, x_6x_2x_3x_7)\rangle),\\
&\ (0, \rho_0, \langle \mathtt{S}(x_0x_1, x_2x_3), \mathtt{X}(x_4, x_5), \mathtt{Y}(x_6, x_7)\rangle),\\
&\ (1, \rho_2, \langle \mathtt{S}(x_0x_1, x_2x_3), \mathtt{A}(x_4), \mathtt{B}(x_5), \mathtt{Y}(x_6, x_7)\rangle),\\
&\ (3, \rho_5, \langle \mathtt{S}(x_0x_1, x_2x_3), \mathtt{A}(x_4), \mathtt{B}(x_5), \mathtt{C}(x_6), \mathtt{D}(x_7)\rangle),\\
&\ (0, \rho_1, \langle \mathtt{S}(\varepsilon, \varepsilon), \mathtt{X}(x_0, x_1), \mathtt{Y}(x_2, x_3), \mathtt{A}(x_4), \mathtt{B}(x_5), \mathtt{C}(x_6), \mathtt{D}(x_7)\rangle),\\
&\ (1, \rho_2, \langle \mathtt{S}(\varepsilon, \varepsilon), \mathtt{A}(x_0), \mathtt{B}(x_1), \mathtt{Y}(x_2, x_3), \mathtt{A}(x_4), \mathtt{B}(x_5), \mathtt{C}(x_6), \mathtt{D}(x_7)\rangle),\\
&\ (3, \rho_5, \langle \mathtt{S}(\varepsilon, \varepsilon), \mathtt{A}(x_0), \mathtt{B}(x_1), \mathtt{C}(x_2), \mathtt{D}(x_3), \mathtt{A}(x_4), \mathtt{B}(x_5), \mathtt{C}(x_6), \mathtt{D}(x_7)\rangle)\rangle
\end{aligned}$$

We leave it to the reader to show that Γ' is uniquely defined from Γ up to renaming of variables. At the beginning of Γ' we have a literal $X_0(p_0, p_1, \ldots, p_{k-1})$ where the p_i are such polynomials in the variables x_i that in $q := \prod_{i<k} p_i$ every variable occurs at most once. We call q (which is unique up to renaming of variables) the **characteristic polynomial** of Γ and the derivation Γ' the **skeleton of** Γ. A polynomial is called **linear** if it contains a given variable at most once. All polynomials of Γ' are linear. Further, if $X_i(\langle q^i_j : j < k\rangle)$ is in Γ' then also the product $\prod_{j<k} q^i_j$ is linear. We write $[p]$ for the set of all x_i which occur in p. The characteristic polynomial for our derivation is

$$p(x_0, x_1, \ldots, x_7) := x_4x_0x_1x_5x_6x_2x_3x_7 \;.$$

Now let Γ as before be a derivation of $X_0(\langle \vec{x}_i : i < k\rangle)$. We get a bijective correspondence between occurrences of letters in $\vec{u} := \prod_{i<k} \vec{x}_i$ and variables in the characteristic polynomial p. For the sake of simplicity we let the variable x_i correspond to the ith letter. Then $\vec{u}$ results from p by replacing the variable x_i by the ith letter. Our polynomial is therefore not the one given above but now it is

$$p(x_4, x_0, x_1, x_5, x_6, x_2, x_3, x_7) = x_0x_1x_2x_3x_4x_5x_6x_7 \;.$$

Now let $M := \{\gamma_i : i < |\Gamma|\}$ (where the γ_i are arbitary symbols) and $P := \{x_i : i < |\vec{u}|\}$. We call $S \subseteq P$ a **constituent** and S' a **segment of** S if for the skeleton Γ', for some $i < p$ and the main literal $X_i(\langle q^i_j : j < k\rangle)$ of L_i we have $[q^i_j] = S'$ and $S = \bigcup_{j<k}[q^j_i]$. We also write $C(i)$ for S. We put $x_i \sqsubset x_j$ if and only if $i < j$, $x_i < \gamma_j$ if and only if $x_i \in C(j)$, $\gamma_i < \gamma_j$, if and only if $j < i$ (!) and $C(i) \subseteq C(j)$. The second condition says that constituent parts must consist of subwords. Further, $\gamma_i \sqsubset \gamma_j$ if for all $x_{i'} < \gamma_i$ and alle $x_{j'} < \gamma_j$ we have $x_{i'} \sqsubset x_{j'}$ (so $i' < j'$). Notice by the way that for every $i < |\vec{u}|$, $\{x_i\}$ is a constituent. Finally, let $\ell(\gamma_i) := X_i$ and $\ell(x_i) := u_i$ where u_i is the ith letter of $\vec{u}$. Put $\mathfrak{B}(\Gamma) = \langle M \cup P, <, \sqsubset, \ell\rangle$. This is a tree.

Theorem 5.4.4 *For every complete derivation* Γ, $\mathfrak{B}(\Gamma)$ *is an ordered labelled tree, the so called* ***structure tree of*** Γ.

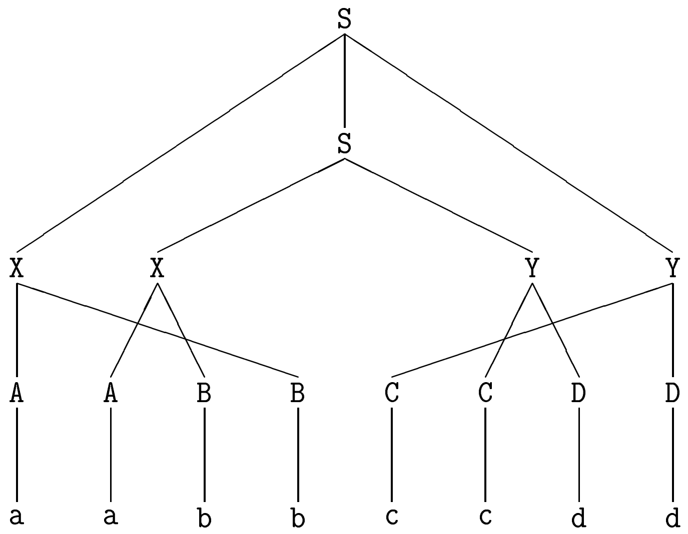
This tree says something about the constituent which the derivation Γ defines over $\vec{u}$. We give an example. The following is the derivation above with the variables replaced accordingly.

$$\begin{array}{l}
\langle(0, \rho_0, \langle \mathtt{S}(x_0x_1x_2x_3, x_4x_5x_6x_7)\rangle),\\
\ (0, \rho_0, \langle \mathtt{S}(x_1x_2, x_5x_6), \mathtt{X}(x_0, x_3), \mathtt{Y}(x_4, x_7)\rangle),\\
\ (1, \rho_2, \langle \mathtt{S}(x_1x_2, x_5x_6), \mathtt{A}(x_0), \mathtt{B}(x_3), \mathtt{Y}(x_4, x_7)\rangle),\\
\ (3, \rho_5, \langle \mathtt{S}(x_1x_2, x_5x_6), \mathtt{A}(x_0), \mathtt{B}(x_3), \mathtt{C}(x_4), \mathtt{D}(x_7)\rangle),\\
\ (0, \rho_1, \langle \mathtt{S}(\varepsilon, \varepsilon), \mathtt{X}(x_1, x_2), \mathtt{Y}(x_5, x_6), \mathtt{A}(x_0), \mathtt{B}(x_3), \mathtt{C}(x_4), \mathtt{D}(x_7)\rangle),\\
\ (1, \rho_2, \langle \mathtt{S}(\varepsilon, \varepsilon), \mathtt{A}(x_1), \mathtt{B}(x_2), \mathtt{Y}(x_5, x_6), \mathtt{A}(x_0), \mathtt{B}(x_3), \mathtt{C}(x_4), \mathtt{D}(x_7)\rangle),\\
\ (3, \rho_5, \langle \mathtt{S}(\varepsilon, \varepsilon), \mathtt{A}(x_1), \mathtt{B}(x_2), \mathtt{C}(x_5), \mathtt{D}(x_6), \mathtt{A}(x_0), \mathtt{B}(x_3), \mathtt{C}(x_4), \mathtt{D}(x_7)\rangle)\rangle
\end{array}$$

We have the following main literals and constituents.

line	main literal	constituent
6	$\mathtt{Y}(x_5, x_6)$	$\{x_5, x_6\}$
5	$\mathtt{X}(x_1, x_2)$	$\{x_1, x_2\}$
4	$\mathtt{S}(x_1x_2, x_5x_6)$	$\{x_1, x_2, x_5, x_6\}$
3	$\mathtt{Y}(x_4, x_7)$	$\{x_4, x_7\}$
2	$\mathtt{X}(x_0, x_3)$	$\{x_0, x_3\}$
1	$\mathtt{S}(x_0x_1x_2x_3, x_4, x_5, x_6, x_7)$	$\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$

The structure tree is, as one easily establishes, the following.



This tree is not exhaustively ordered. This is the main difference with context free grammars. Notice that the tree does not reflect the position of the empty constituent. Its segments are found between the second and the third as well the between the sixth and the seventh letter. One can define the structure tree also in this way that it explicitly contains the empty strings. To this end one has to replace also occurrences of ε by variables. The rest is then analogous.

The structures that result in this way can also be generated by context free graph grammars. We shall show how. To simplify the problem we shall assume that all rules are monotone. This means that they do not change the order of the segments within a literal.

Definition 5.4.5 *Let $\rho = L \leftarrow M_0 \ldots M_{n-1}$ be a linear rule and $L = B(\langle t^j : j < k \rangle)$ as well as $M_i = A_i(\langle x_i^j : j < k_i \rangle)$. ρ is called* ***monotone*** *of for every $i < n$ and every pair $j < j' < k_i$ the following holds: if x_j occurs in the qth segment and $x_{j'}$ in the q'th segment of L then either $q < q'$ or x_j occurs before $x_{j'}$ in the polynomial t_q.*

The following rule is monotone.

$$A(x_0y_0x_1, y_1) \leftarrow B(x_0, x_1), C(y_0, y_1).$$

For we have that x_0 is to the left of x_1 and y_0 in the segment that occurs before the segment of y_1. The following rules however are not monotone.

$$A(x_1, x_0y_0y_1) \leftarrow B(x_0, x_1), C(y_0, y_1).; A(x_1x_0, y_0y_1) \leftarrow B(x_0, x_1), C(y_0, y_1).$$

We shall at first restrict ourselves to monotone grammars. After that we shall show how this restriction can be lifted. Let k be given. For the sake of simplicity we assume that all predicates are k–ary and terminal rules are of the form $Y(a, \varepsilon, \dots, \varepsilon)$, $a \in A$. We call a $k \times k$–matrix $M = (m_{ij})_{ij}$ with entries from $\{0, 1\}$ a k–**scheme** if for all j, i and i' it holds that: if $i > i'$ then $m_{ij} \leq m_{i'j}$. We interpret this scheme as follows. If $\vec{x}_i$, $i < k$, and $\vec{y}_j$, $j < k$, are sequences of subwords of $\vec{u}$, $\vec{u}$ itself a subword then we define the scheme of the $\vec{x}_i$ with respect to the $\vec{y}_i$ by $m_{ij} = 1$ if $\vec{x}_i$ is to the left of $\vec{y}_j$. M is the scheme that belongs to two sequences. In what is to follow, $\langle \vec{x}_i : i < k\rangle$ as well as $\langle \vec{y}_j : j < k\rangle$ are constituents. The $\vec{x}_i$ and the $\vec{y}_j$ are therefore always pairwise disjoint; and further, $\vec{x}_i$ is to the left of $\vec{x}_j$ and $\vec{y}_i$ to the left of $\vec{y}_j$ if $i > j$ (this is a consequence of the monotonicity). With this being given the linear order between the $2k$ segments is fixed by naming a schema. We illustrate this with an example. Here are all admissible 2–schemes together with the orders which define them. (We omit the vector arrows; the ordering is defined by 'is to the left of in the string'.)

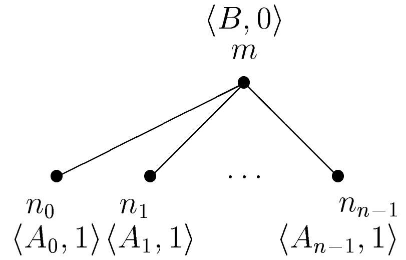
$$\begin{array}{ccc} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \\ x_0x_1y_0y_1 & x_0y_0x_1y_1 & x_0y_0y_1x_1 \end{array}$$

$$\begin{array}{ccc} \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ y_0x_0x_1y_1 & y_0x_0y_1x_1 & y_0y_1x_0x_1 \end{array}$$

For every k–scheme M let $\lambda(M)$ be a relation. Now we define our graph–grammar. The set of vertex colours $F_V := N \times \{0,1\} \cup A$, the set of terminal vertex colours is $F_V^T := N \times \{0\} \cup A$, the set of edge colours is $\{<\} \cup \{\lambda(M) : M \text{ a } k\text{–scheme}\}$. (As we will see immediately, $\sqsubset$ is the relation which fits to the relation $\lambda(\mathbb{I})$ where $\mathbb{I} = (1)_{ij}$ is the matrix which consists only of 1s.) The start graph is the one–element graph $\mathfrak{S}$ which has no (!) edges and whose vertex has colour S. For every rule ρ we add a graph replacement rule. Let

$$\rho = B(\langle t^j : j < k\rangle) \leftarrow A_0(\langle x_0^j : j < k\rangle) \ldots A_{n-1}(\langle x_{n-1}^j : j < k\rangle)$$

be given. Let $p := \prod_{i<n} t^i$ be the characteristic polynomial of the rule. Then the graph replacement ρ^γ replaces the node $\langle B, 1\rangle$ by the following graph.



Furthermore, between the nodes n_i and n_j the following relations hold (which are not visible in the picture). Put $m_{i'j'} := 1$ if $x_i^{i'}$ is to the left of $x_j^{j'}$ in p. Then put $H_{ij} := (m_{i'j'})_{i'j'}$. The relation between n_i and n_j is therefore equal to H_{ij}. (Notice that by definition always either $x_i^{i'}$ is to the left of $x_j^{j'}$ or to the right of it. Hence the relation between n_j and n_i is $1 - H_{ij}$.) This defines the graph. Now we have to determine the colour functional.

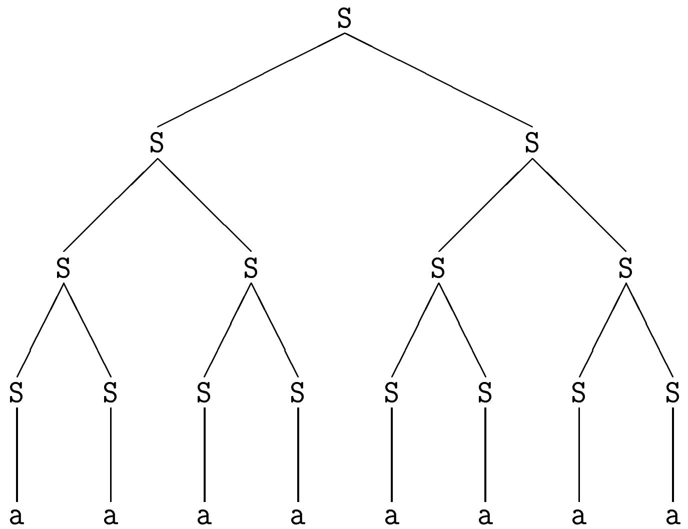
There is a possibility of defining structure in some restricted cases, namely always when the right hand sides do not contain a variable twice. This differs from linear grammars in that variables are still allowed to occur several times on the left, but only once on the right. An example is the grammar

$$\mathtt{S}(xx) \leftarrow \mathtt{S}(x).; \quad \mathtt{S}(\mathtt{a}).$$

In this case the notion of structure that has been defined above can be transferred to grammar. We simply do as if the first rule was of this form

$$\mathtt{S}(xy) \leftarrow \mathtt{S}(x), \mathtt{S}(y).$$

where it is clear that x and y always represent the same string. In this ways we get this structure tree for aaaaaaaa.



Notes on this Section. In his paper (1997), Edward Stabler describes a formalisation of minimalist grammars akin to Noam Chomsky's Minimalist Program (outlined in (Chomsky, 1993)). Subsequently, in (Michaelis, 2001) and (Harkema, 2001) it is shown that the languages generated by this formalism are exactly those that can be generated by a simple LMG, or, for that matter, by an LCFRSs. **Exercise 180.** Show that the derivation Γ' is determined by Γ up to renaming of variables.

Exercise 181. Prove Proposition 5.4.2.

Exercise 182. Determine the graph grammar $\gamma G^{\heartsuit}$.

Exercise 183. Show the following. *Let* $N = \{x_i : i < k\} \cup \{y_i : i < k\}$ *and* $<$ *be linear orderings on* N *with* $x_i < x_j$ *as well as* $y_i < y_j$ *for all* $i < j < k$*. Then if* $m_{ij} = 1$ *if and only if* $x_i < y_j$ *then* $M = (m_{ij})_{ij}$ *is a* k*–scheme. Conversely: let* M *be a* k*–scheme and* $<$ *defined by (1)* $x_i < x_j$ *if and only if* $i < j$*, (2)* $y_i < y_j$ *if and only if* $i < j$*, (3)* $x_i < y_j$ *if and only if* $m_{ij} = 1$*. Then* $<$ *is a linear ordering. The correspondence between orderings and schemes is biunique.*

Exercise 184. Show the following: *If in a linear* k*–LMG all structure trees are exhaustively ordered, the generated tree set is context free.*

5.5 Adjunction Grammars

In this and the next section we shall concern ourselves with some alternative types of grammars which are all (more or less) equivalent to head grammars. These are the tree adjunction grammars, the CCGs (which are some refined version of the adjunction grammars of Section 1.4 and the grammars $CCG(\mathbf{Q})$ of Section 3.4, respectively) and the so called linear index grammars.

Let us return to the concept of tree adjunction grammars. These are pairs $G = \langle \mathsf{C}, N, A, \mathsf{A}\rangle$, where C is a set of so called center trees and A a set of so called adjunction trees. In an adjunction tree a node is called **central** if it is above the distinguished leaf or identical with it. We define a **derivation** as follows. A derivation is a tree which is generated by the following grammar. The nonterminals are pairs $(\mathfrak{B}, i)$ where $\mathfrak{B}$ is a tree and i a node of $\mathfrak{B}$. For the sake of simplicity we assume that the set of nodes of $\mathfrak{B}$ is the set $j(\mathfrak{B}) = \{0, 1, \ldots, j(\mathfrak{B}) - 1\}$. Terminal symbols are symbols of the form i where i is a node in a tree. The start symbol is S. The rules are of this form.

$$\begin{array}{llll} \text{(s)} & S & \to & (0, \mathfrak{C}) \\ \text{(a)} & (j, \mathfrak{A}) & \to & X_0 \quad X_1 \ldots X_{j(\mathfrak{A})-1} \end{array}$$

where (a) is to be seen as rule a scheme: for every $\mathfrak{A}$ and every admissible j we have 2^k many rules, where X_i always is either i or $(i, \mathfrak{B}_i)$ for some tree $\mathfrak{B}_i$ which can be adjoined at i in $\mathfrak{A}$. In (s) we must have $\mathfrak{C} \in \mathtt{C}$. This grammar we denote by $D(G)$ and call it the **derivation grammar**. This is a very useful concept.

If $\mathfrak{T}$ is a tree which is derived by means of G the exists an associated derivation tree $D(\mathfrak{T})$ defined in the following way. (a) If no tree has been adjoined to a tree to give $\mathfrak{T}$, $\mathfrak{T}$ is a center tree. The associated tree is determined by the derivation

$$S \to (0, \mathfrak{T}) \to 0 \quad 1 \ \ldots \ j(\mathfrak{T}) - 1$$

Now let $\mathfrak{A}$ be a simple adjunction tree. We associate with it the following derivation.

$$D^j(\mathfrak{T}) := (j, \mathfrak{T}) \to 0 \quad 1 \ \ldots \ j(\mathfrak{T}) - 1$$

Here j is arbitrary. Let $\mathfrak{T}$ result by adjunction of the trees $\mathfrak{U}_{k_j}$ to the nodes k_j, $j < p$, of the tree $\mathfrak{A}$. Then we associate the following tree with $\mathfrak{T}$:

$$(j, \mathfrak{A}) \to X_0 \quad X_1 \ \ldots \ X_{j(\mathfrak{A})-1}$$

where $X_i = i$ if i is no k_j otherwise let $X_{k_j} := D^{k_j}(\mathfrak{U}_{k_j})$. (Notice that we define trees and not derivations. Otherwise this definition makes no sense.) Conversely, if $\mathfrak{D}$ is a tree which is generated by $D(G)$ we can find by induction a tree $A(\mathfrak{D})$ such that $D(A(\mathfrak{D})) \cong \mathfrak{D}$. We shall perform this construction further below. Insofar the promised correspondence exists.

If a derivation tree $\mathfrak{D} = \langle D, <, \sqsubset, \ell\rangle$ is given, the terminal nodes of $A(\mathfrak{D})$ are in bijective correspondence with the leaves of the derived tree. Look at a leaf of x of the derivation. In analogy to the branch expression (see Exercise 2.2) define an **address** as follows. In the definition let $y \succ x$.

$$\begin{array}{lcll}
\alpha(x) & := & \mathfrak{C} & \text{if } t(x) = 1, \ell(x) = (0, \mathfrak{C}) \\
\alpha(x) & := & \alpha(y) \cdot i & \text{if } \ell(x) = i \\
\alpha(x) & := & \alpha(y) \cdot i \cdot GA & \text{if } \ell(x) = (i, \mathfrak{A})
\end{array}$$

For x a leaf this reveals the adjunction history of the tree associated node in the tree. Now let

$$A(\mathfrak{D}) := \langle A, <^A, \sqsubset^A, \ell^A \rangle ,$$

where

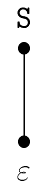
$$A := \{\alpha(x) : x \text{ leaf of } \mathfrak{D}\} .$$

Further, if $\alpha(x) = \vec{\imath}\cdot\mathfrak{A}\cdot j$, let $\ell^A(x) := X$ for the unique X which is the label of the node j in $\mathfrak{A}$. Second, $\alpha(x) \sqsubset \alpha(y)$ if $\alpha(x) = \vec{\imath}\cdot\mathfrak{A}\cdot j\cdot\vec{\jmath}$ and $\alpha(y) = \vec{\imath}\cdot\mathfrak{A}\cdot j';\vec{\jmath}'$ for certain $\vec{\imath}$, $\vec{\jmath}$, $\mathfrak{A}$ and $j \neq j'$ such that $j \sqsubset j'$. Third, $\alpha(x) < \alpha(y)$ if $\alpha(x) = \vec{\imath} \cdot \mathfrak{A} \cdot j \cdot \vec{\jmath}$ and $\alpha(y) = \vec{\imath} \cdot \mathfrak{A} \cdot j' \cdot \vec{\jmath}'$ for certain $\vec{\imath}$, $\vec{\jmath}$, $\vec{\jmath}'$, $\mathfrak{A}$ and $j \neq j'$, such that (a) $j < j'$, (b) j' and every node of $\vec{\jmath}'$ is central in its corresponding adjunction tree.

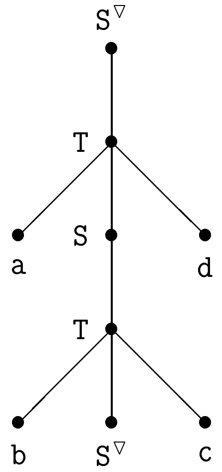
Tree adjunction grammars (TAGs) differ from the unregulated tree adjunction grammars in that it is allowed to specify

1. whether adjunction at a certain node is licit,
2. which trees may be adjoined at which node,
3. whether adjunction is obligatory at certain nodes.

We shall show that the first restriction indeed give greater expressivity for the grammar, and that the other two are not really effective. To establish control over derivations, we shall have to change our definitions a little bit. We begin with the first point, the prohibition of adjunction. To implement it, we assume that the category symbols are now of the form a, $a \in A$, or X and $X^\triangledown$ respectively, where $X \in N$. Center- and adjunction trees are defined as before. Adjunction trees are defined as follows. There is a leaf i which has the same label as the root (all other leaves carry terminal labels). However, no adjunction is licit at nodes with label $X^\triangledown$. We now allow (even if this is not necessary) that $\mathfrak{T}$ is an adjunction tree also if a leaf carries the label X or $X^\triangledown$ and the root the label $X^\triangledown$ or X. So, the distinguished leaf and the root do not really have to carry exactly the same nonterminal label, only the letter X must be the same. Using such grammars one can generate the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n \in \omega\}$. For example, choose the following center tree.



There is only one adjunction tree.



It is not hard to show that one can reduce such grammars to those where both the root and the leaf carry labels of the form $X^\triangledown$. To this end we remark the following. If a node in the interior of adjunction tree carries a node to which one may not adjoin, then we replace the label by a label $\notin N$. Then no tree can be adjoined there. Further, if the root does not carry an adjunction prohibition then add a new root which does, and carries the same base label. The same with the distinguished leaf.

Definition 5.5.1 *A **standard tree adjunction grammar (TAG)** is an adjunction grammar in which the adjunction trees carry an adjuncton prohibition at the root and the distinguished leaf.*

Now let us return to the other points. It is possible to specify whether adjunction is obligatory and which trees may be adjoined. So, we also have a function f, which maps all nodes with nonterminal labels to set of adjunction trees. (If $f(i) = \varnothing$ the node has an adjunction prohibition.) We can simulate this as follows. Let A be the set of adjunction trees. We think of the nonterminals as labels of the form $\langle X, \mathfrak{T}\rangle$ and $\langle X, \mathfrak{T}\rangle^{\triangledown}$, respectively, where $X \in N$ and $\mathfrak{T} \in \mathsf{A}$. A tree $\mathfrak{T}$ is replaced by all trees $\mathfrak{T}'$ on the same set of nodes, where i carries the label $\langle X, \mathfrak{U}\rangle$ if i had label X in $\mathfrak{T}$ if $\mathfrak{U} \in f(i)$, and $\langle X, \mathfrak{U}\rangle^{\triangledown}$ if i has the label $X^{\triangledown}$ in $\mathfrak{T}$. The second says nothing but which tree is going to be adjoined next. This eliminates the second point from the list, as we can reduce the grammars by keeping the tree structure.

Now let us turn to the last point, the obligation for adjunction. We can implement this by introducing labels of the form $X^{\bullet}$. (Since obligation and prohibition to adjoin are exclusive, $\bullet$ occurs only when $\triangledown$ does not.) A tree is complete only if there are no nodes with label $X^{\bullet}$ for any X. Now we shall show that for every adjunction grammar of this kind there exists a grammar generating the same set of trees where there is no obligation for adjunction. For every center tree we adjoin as often as necessary to eliminate the obligation. The same we do for adjunction trees. The resulting trees shall be our new center and adjunction trees. Obviously, such trees exist (otherwise we may choose the set of center trees to be empty). Now we have to show that there exists a finite set of minimal trees. Look at a tree without adjunction obligation and take a node. This node has a history. It has been obtained by successive adjunction. If this sequence contains an adjunction tree twice, we may cut the cycle. (The details of this operation are left to the reader.) This grammar still generates the same trees. So, we may remain with the standard form of TAGs.

Now we shall first prove that adjunction grammars cannot generate more languages as linear 2–LMGs. From this it immediately follows that they can be parsed in polynomial time.

Theorem 5.5.2 (Weir) *For every TAG G there exists a head* *grammar K such that $L(K) = L(G)$.*

Proof. Let G be given. We assume that the trees have pairwise disjoint sets of nodes. We may also assume that the trees are at most binary branching. (We need only show weak equivalence.) Furthermore, we can assume that the nodes are strictly branching if not preterminal. The set of all nodes is denoted by M. The alphabet of nonterminals is $N' := \{i^a : i \in M\} \cup \{i^n : i \in M\}$. The start symbol is the set of all i^a and i^n where i is the root of a center tree. By massaging the grammar somewhat one can achieve that the grammar contains only one start symbol. Now we shall define the rules. For a local tree we put

$$(t) \qquad i(a, \varepsilon) \leftarrow . \, ,$$

where j is a leaf with terminal symbol a. If i is a distinguished leaf of an adjunction tree we also take the rule

$$(e) \qquad i^n(\varepsilon, \varepsilon) \leftarrow . $$

Now let $i \to j \quad k$ be a branching local tree. Then we add the following rules.

$$(p) \qquad i^a(x_0x_1, y_0y_1) \leftarrow j^n(x_0, x_1), k^n(y_0, y_1).$$

Further we have the following rules. If i is a node to which a tree with root j can be adjoined, then also this is a rule.

$$(f) \qquad i^n(x_0y_0, y_1x_1) \leftarrow j^n(x_0, x_1) \quad i^a(y_0, y_1) \, .$$

If adjunction is not necessary or prohibited at i, then finally the following rule is added.

$$(n) \qquad i^n(x_0, x_1) \leftarrow i^a(x_0, x_1) \, .$$

This ends the definition of K. In view of the rules (p) it is not entirely clear that we are dealing with a head grammar. So replace the rules (p) by the following rules:

$$\begin{array}{lcl} i^a(x_0, x_1y_0y_1) & \leftarrow & j^{n\bullet}(x_0, x_1), k^{n\bullet}(y_0, y_1). \\ j^{n\bullet}(x_0x_1y_0, y_1) & \leftarrow & j^n(x_0, x_1), \mathtt{L}(y_0, y_1). \\ k^{n\bullet}(x_0, x_1y_0y_1) & \leftarrow & \mathtt{L}(x_0, x_1), k^n(y_0, y_1). \\ \mathtt{L}(\varepsilon, \varepsilon) & \leftarrow & . \end{array}$$

These are rules of a head grammar; (p) can be derived from them. For this reason we remain with the rules (p).

It remains to show that $L(K) = L(G)$. First the inclusion $L(G) \subseteq L(K)$. We show the following. Let $\mathfrak{T}$ be a local tree which contains exactly one distinguished leaf and nonterminal leaves x_i, $i < n$, with labels k_i. Let therefore $j < i$ be distinguished. We associate with $\mathfrak{T}$ a vector polynomial $\mathfrak{p}(\mathfrak{T})$ which returns

$$\langle \prod_{i<j} \vec{y}_i\vec{z}_i, \prod_{j<i<n} \vec{y}_i\vec{z}_i\rangle$$

for given pairs of strings $\langle \vec{y}_i, \vec{y}_i'\rangle$. It is possible to show by induction over $\mathfrak{T}$ that there is a K–derivation

$$i^n(\mathfrak{p}(\mathfrak{T})(\langle\langle \vec{y}_i, \vec{z}_i\rangle : i < n\rangle)) \leftarrow^* k_0^n(\langle \vec{y}_0, \vec{z}_0\rangle), \ldots, k_{n-1}^n(\langle \vec{y}_{n-1}, \vec{z}_{n-1}\rangle) \ .$$

If no leaf is distinguished in $\mathfrak{T}$ the value of $p(\mathfrak{T})$ is exactly

$$\langle \vec{y}_0\vec{z}_0, \prod_{0<i<n} \vec{y}_i\vec{z}_i\rangle \ .$$

This claim can be proved inductively over the derivation of $\mathfrak{T}$ in G. From this it follows immediately that $\vec{x} \in L(K)$ if $\vec{x} \in L(G)$. For the converse inclusion one has to choose a different proof. Let $\vec{x} \in L(K)$. We choose a K–derivation of $\vec{x}$. Assume that no rule of type (f) has been used. Then $\vec{x}$ is the string of a center tree as is easily seen. Now we assume that the claim has been shown for derivations with fewer than n applications of (f) and that the proof has exactly n applications. We look at the last application. This is followed only by applications of (p), (t) and (e). These commute, if they belong to different subtrees. We can therefore rearrange the order such that our application of (f) is followed exactly by those applications of (p), (t) and (e) which belong to that subtree. They derive

$$i^a(\vec{x}_0, \vec{x}_1).$$

where i is the left hand side of the application of (f), and $\langle \vec{x}_0, \vec{x}_1 \rangle$ is the pair of the adjunction tree whose root is i. ($\vec{x}_0$ is to the left of the distinguished leaf, $\vec{x}_1$ to the right.) Before that we have the application of our rule (f):

$$j^a(\vec{x}_0\vec{y}_0, \vec{y}_1\vec{x}_1) \leftarrow i^a(\vec{x}_0, \vec{x}_1), j^n(\vec{y}_0, \vec{y}_1) \ .$$

Now we scrap this part of the derivation. This means that in place of $j^a(\vec{x}_0\vec{y}_0, \vec{y}_1\vec{x}_1)$ we only have $j^n(\vec{y}_0, \vec{y}_1)$. This however is derivable (we already have the derivation). But on the side of the adjunction this corresponds exactly to the disembedding of the corresponding adjunction tree. □

The question arises whether the converse holds. This not the case. For example look at the following grammar G.

$$\begin{array}{lcl}
\mathtt{S}(y_0x_0y_1, x_1) & \leftarrow & \mathtt{T}(x_0, x_1), \mathtt{H}(y_0, y_1). \\
\mathtt{T}(x_0, \mathtt{c}x_1\mathtt{d}) & \leftarrow & \mathtt{U}(x_0, y_1). \\
\mathtt{U}(\mathtt{a}x_0\mathtt{b}, x_1) & \leftarrow & \mathtt{S}(x_0, x_1). \\
\mathtt{S}(\mathtt{ab}, \mathtt{cd}) & \leftarrow & . \\
\mathtt{H}(\mathtt{t}x_0\mathtt{u}, x_1) & \leftarrow & \mathtt{K}(x_0, x_1). \\
\mathtt{K}(x_0, \mathtt{v}x_1\mathtt{w}) & \leftarrow & \mathtt{H}(x_0, x_1). \\
\mathtt{H}(\varepsilon, \varepsilon) & \leftarrow & .
\end{array}$$

To begin, this is indeed a head grammar. To analyze the generated language we fix the following facts.

Lemma 5.5.3 $\mathtt{H}(\vec{x}, \vec{y})$ *if and only if* $\langle \vec{x}, \vec{y} \rangle = \langle \mathtt{t}^n\mathtt{u}^n, \mathtt{v}^n\mathtt{w}^n \rangle$ *for a certain* $n \in \omega$.

As a proof one may reflect that first of all $\vdash_G \mathtt{H}(\varepsilon, \varepsilon)$ and secondly

$$\vdash_G \mathtt{H}(\mathtt{t}x_0\mathtt{u}, \mathtt{v}x_1\mathtt{w}) \text{ if and only if } \vdash_G \mathtt{H}(x_0, x_1)$$

From this the following characterization can be derived.

Lemma 5.5.4 *Let* $\vec{x}_n := \mathtt{t}^n\mathtt{u}^n$ *and* $\vec{y}_n := \mathtt{v}^n\mathtt{w}^n$. *Then*

$$L(G) = \{\mathtt{a}\vec{x}_{n_0}\mathtt{a}\vec{x}_{n_1}\mathtt{a}\ldots\vec{x}_{n_{k-1}}\mathtt{ab}\vec{y}_{n_{k-1}}\mathtt{b}\ldots\mathtt{b}\vec{y}_{n_1}\mathtt{b}\vec{y}_{n_0}\mathtt{bc}^k\mathtt{d}^k : k \in \omega, n_i \in \omega\}$$

In particular, for every $\vec{x} \in L(G)$

$$\mu(\vec{x}) = m(\mathtt{a} + \mathtt{b} + \mathtt{c} + \mathtt{d}) + n(\mathtt{t} + \mathtt{u} + \mathtt{v} + \mathtt{w})$$

for certain natural numbers m *and* n.

For example

`aabbccdd`, `atuabvwbccdd`, `attuuatuabbvwbvvwwbcccddd`, ...

are in $L(G)$ but not

`atuabbcd`, `attuuatuabvwbvvwwbccdd`

Now for the promised proof that there is no TAG which can generate this language. Assume the contrary. Let H be a TAG with $L(H) = L(G)$. Then

Lemma 5.5.5 *Let* H *be a TAG with* $L(H) = L(G)$ *and* $\mathfrak{B}$ *a center or adjunction tree. Then*

$$\mu(\mathfrak{B}) = m_{\mathfrak{B}}(\mathtt{a} + \mathtt{b} + \mathtt{c} + \mathtt{d}) + n_{\mathfrak{B}}(\mathtt{t} + \mathtt{u} + \mathtt{v} + \mathtt{w})$$

for certain natural numbers $m_{\mathfrak{B}}$ *and* $n_{\mathfrak{B}}$.

We put $\rho_{\mathfrak{B}} := n_{\mathfrak{B}}/m_{\mathfrak{B}}$. (This is ∞, if $m = 0$.) Certainly, there exists the minimum of all $\rho_{\mathfrak{B}}$ for all adjunction trees. It is easy to show that it must be $= 0$. So there exists an adjunction tree which consists only of `t`, `u`, `v` and `w`, in equal number. Further there exists an adjunction tree which contains `a`.

We now look at a string $\vec{x}$ from $L(G)$ with the following property. We have

$$\mu(\vec{x}) = m(\mathtt{a} + \mathtt{b} + \mathtt{c} + \mathtt{d}) + n(\mathtt{t} + \mathtt{u} + \mathtt{v} + \mathtt{w})$$

for certain natural numbers m and n such that (a) m is larger than any $m_{\mathfrak{B}}$, and (b) n/m is smaller than any $\rho_{\mathfrak{B}}$ that is not equal to 0. It is to be noticed that such a $\vec{x}$ exists. If m and n are chosen, the following string does the job.

$$\mathtt{at}^n\mathtt{u}^n\mathtt{a}^{m-1}\mathtt{b}^{n-1}\mathtt{v}^n\mathtt{w}^n\mathtt{bc}^m\mathtt{d}^m$$

This string results from a center tree by adjoining (a′) an $\mathfrak{A}$ in which a occurs, by adjoining (b′) a $\mathfrak{B}$ in which a does not occur. Now we look at points in which $\mathfrak{B}$ has been inserted. These can only be as follows.

$$\mathtt{at}^n \bullet \mathtt{u}^n\mathtt{a}^{m-1}\mathtt{b}^{n-1}\mathtt{v}^n \bullet \mathtt{w}^n\mathtt{bc}^m\mathtt{d}^m$$

However, let us look where the adjunction tree $\mathfrak{A}$ has been inserted.

$$\mathtt{at}^n\mathtt{u}^n\mathtt{a}^{m-1} \circ \mathtt{b}^{n-1}\mathtt{v}^n\mathtt{w}^n\mathtt{bc}^m \circ \mathtt{d}^m$$

If we put this on top of each other, we get this,

$$\mathtt{at}^n \bullet \mathtt{u}^n\mathtt{a}^{m-1} \circ \mathtt{b}^{n-1}\mathtt{v}^n\mathtt{w}^n \bullet \mathtt{bc}^m \circ \mathtt{d}^m$$

Now we have a contradiction. The points of adjunction may not cross! For the subword between the two $\bullet$ must be a constituent, likewise the part between the two $\circ$. However, these constituents are not contained in each other. (In order for this to become a real proof one has to reflect over the fact that the constituent structure is not changed by adjunction. This is an exercise.)

Now we have a head grammar which generates a language that cannot be generated by a TAG. Now we shall show that in turn head grammars are weaker than 2–branching 2–LMGs and these weaker than full linear 2–LMGs. Some parts of the argumentation shall be transferred to the exercises, since they are not of central concern.

Definition 5.5.6 *A linear LMG is called n–**branching** if the polynomial base consists of at most k–ary vector polynomials.*

The reason for this definition is the following fact.

Proposition 5.5.7 *Let $L = L(G)$ for some n–branching, k–linear LMG. Then there exists a k–linear LMG H with $L(H) = L$ in which every rule is at most n–branching.*

To this end one has to see that a rule with more than n daughters can be replaced by a canonical sequence of rules with at most n daughters, if the corresponding vector polynomial is generated by at most n–ary polynomials. On the other hand it is not guaranteed that there is no n–branching grammar if higher polynomials have been used. However, it is possible to construct languages such that essentially $n+1$–ary polynomials have been used and they cannot be reduced to at most n–ary polynomials. Let for example this language be given.

$$\vec{x}_n := \mathtt{t}^n\mathtt{u}^n, \qquad \vec{y}_n := \mathtt{v}^n\mathtt{w}^n \ .$$

The following polynomials is not generable using polynomials that are at most ternary.

$$\mathtt{q}(\langle w_0, w_1\rangle, \langle x_0, x_1\rangle, \langle y_0, y_1\rangle, \langle z_0 z_1\rangle) := \langle w_0 x_0 y_0 z_0, y_1 w_1 z_1 x_1\rangle$$

From this we can produce a proof that the following language cannot be generated by a 2–branching LMG.

$$L = \{\vec{x}_{n_0}\vec{x}_{n_1}\vec{x}_{n_2}\vec{x}_{n_3}\vec{y}_{n_2}\vec{y}_{n_0}\vec{y}_{n_3}\vec{y}_{n_1} : n_0, n_1, n_2, n_3 \in \omega\}$$

We shall not dwell on this, but turn instead to the question of head grammars. We shall show that there is a language which is generated by a 2–branching LMG, but cannot be generated by a head grammar.

$$M := \{\vec{x}_{n_0}\vec{x}_{n_1}\vec{y}_{n_0}\vec{y}_{n_1} : n_0, n_1 \in \omega\}$$

To begin we shall give the grammar that generates it.

$$\begin{array}{lcl} \mathtt{S}(x_0 x_1, y_0 y_1) & \leftarrow & \mathtt{H}(x_0, x_1), \mathtt{H}(y_0, y_1). \\ \mathtt{H}(\varepsilon, \varepsilon) & \leftarrow & . \\ \mathtt{H}(\mathtt{t}x_0\mathtt{u}, \mathtt{v}x_0\mathtt{w}) & \leftarrow & \mathtt{H}(x_0, x_1). \end{array}$$

Evidently, this is a 2–branching LMG. We can assume that this grammar is monotone. Now we shall show that there is no head grammar generating the language. First a definition. Let $\vec{x}$ be a string and $C = \langle \vec{y}_0, \vec{y}_1 \rangle$ as well as $D = \langle \vec{z}_0, \vec{z}_1 \rangle$ disjoint constituents. We say C is **to the left of** (**to the right of**) D if all segments of C are to the left of (to the right of) all segments of D; C is **central** in D if $\vec{z}_0 \sqsubset \vec{y}_0$ and $\vec{y}_1 \sqsubset \vec{z}_1$. Finally, we say C and D **cross** if $\vec{y}_0 \sqsubseteq \vec{z}_0 \sqsubset \vec{y}_1 \sqsubset \vec{z}_1$ or $\vec{z}_0 \sqsubset \vec{y}_0 \sqsubset \vec{z}_1 \sqsubset \vec{y}_1$.

Lemma 5.5.8 *Let G be a head grammar and $\mathfrak{T}$ a structure tree generated by γG. Then there exist in $\mathfrak{T}$ no crossing constituents.*

Now we go over to the language M. We shall show that every grammar generating it must contain two crossing constituents. Let $\vec{x}_0\vec{x}_1\vec{y}_0\vec{y}_1$ be given. If n_0 and n_1 are large enough there exists a constituent of type A which contains another constituent of type A. Then it is easily seen that A is contained either in $\vec{x}_0\vec{y}_0$ or in $\vec{x}_1\vec{y}_1$. (For if A is being pumped, then we must pump an equal number of $\mathtt{t}$, $\mathtt{u}$, $\mathtt{v}$ and $\mathtt{w}$. One automatically gets four subwords of the form $\mathtt{t}^p$, $\mathtt{u}^p$, $\mathtt{v}^p$ und $\mathtt{w}^p$, which are simultaneously being pumped. In order for this to be possible, they have to be subwords of $\vec{x}_0\vec{y}_0$ or of $\vec{x}_1\vec{y}_1$.) Now let n_0 and n_1 be sufficiently large. Then $\vec{x}_0$ as well as $\vec{x}_1$ contain two constituents of equal type (properly contained in each other). This means that we may now assume that $\vec{x}_0\vec{y}_0$ contains two different constituents C_0 and C_1 of type A and $\vec{x}_1\vec{y}_1$ two different constituents D_0 and D_1 of type B. Now it is easily seen that C_0 and D_0 are crossing constituents. For neither C_0 nor D_0 are empty. Further, C_0 contains at least one $\mathtt{t}$, one $\mathtt{u}$, one $\mathtt{v}$ and one $\mathtt{v}$; likewise for D_0. Hence $C_0 = \langle \mathtt{t}^{p_0}\mathtt{u}^{p_1}, \mathtt{v}^{p_2}\mathtt{w}^{p_3} \rangle$ as well as $D_0 = \langle \mathtt{t}^{q_0}\mathtt{u}^{q_1}, \mathtt{v}^{q_2}\mathtt{w}^{q_3} \rangle$. The left hand segment of C_0 is contained in $\vec{x}_0$, the right hand segment in $\vec{y}_0$; the left hand segment of D_0 is contained in $\vec{x}_1$, the right hand segment in $\vec{y}_1$.

Exercise 185. Let $\mathfrak{B}$ be a tree and $\mathfrak{A}$ an adjunction tree. Let $\mathfrak{C}$ be the result of adjoining $\mathfrak{A}$ to x in $\mathfrak{B}$. We view $\mathfrak{B}$ in a natural way as a subtree of $\mathfrak{C}$ with x the lower node of $\mathfrak{A}$ in $\mathfrak{C}$. Show the following: the constituents of $\mathfrak{B}$ are exactly the intersection of constituents of $\mathfrak{C}$ with the set of nodes of $\mathfrak{B}$.

Exercise 186. Show that the language $L := \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n : n \in \omega\}$ cannot be generated by an unregulated TAG. *Hint.* Proceed as in the proof above. Take a string which is large enough so that a tree has been adjoined and analyze the places where it has been adjoined.

Exercise 187. Show that in the example above $min\{\rho_{\mathfrak{B}} : \mathfrak{B} \in \mathsf{A}\} = 0$. *Hint.* Compare the discussion in Section 2.6.

Exercise 188. Show the following: *For every TAG G there is a TAG $G^{\diamondsuit}$ in standard form such that $G^{\diamondsuit}$ and G have the same constituent structures.* What can you say about the labelling function?

Exercise 189. Prove Proposition 5.5.7.

Exercise 190. Prove Lemma 5.5.8.

5.6 Index Grammars

Index grammars broaden the concept of context free grammar in a very special way. They allow to use in addition of the nonterminals a sequence of indices. We may may however consider these grammars also as grammars that contain rule schemata for the manipulation of entire sequences of nonterminals. To this end, let A be our alphabet, N the set of nonterminals (disjoint with A) and I a set of **indices**, disjoint to both A and N. Furthermore, $\sharp$ shall be a symbol that does not occur in $A \cup N \cup I$. An **index scheme** σ has the form

$$A \cdot \vec{\alpha} \to B_0 \cdot \vec{\beta}_0 \quad \ldots \quad B_{n-1} \cdot \vec{\beta}_{n-1}$$

or alternatively the form

$$A \cdot \vec{\alpha} \to a$$

where $\vec{\alpha}, \vec{\beta}_i \in I^* \cup \{\sharp\}$ for $i < n$, and $a \in A$. The schemata of the second kind are called **terminal schemata**. An **instantiation**

of σ is a rule

$$A \cdot \vec{x}\vec{\alpha} \to B_0 \cdot \vec{y}_0\vec{\beta}_0 \quad \ldots \quad B_{n-1} \cdot \vec{y}_{n-1}\vec{\beta}_{n-1}$$

where the following holds.

1. If $\vec{\alpha} = \sharp$ then $\vec{x} = \varepsilon$ and $\vec{y}_i = \varepsilon$ for all $i < n$.

2. If $\vec{\alpha} \neq \sharp$ then for all $i < n$: $\vec{y}_i = \varepsilon$ or $\vec{y}_i = \vec{x}$.

3. For all $i < n$: if $\vec{\beta}_i = \sharp$ then $\vec{y}_i = \varepsilon$.

For a terminal scheme the following condition holds: if $\vec{\alpha} = \sharp$ then $\vec{x} = \varepsilon$. An index scheme simply codes the set of all of its instantiations. So we may also call it a **rule scheme**. If in a rule scheme σ we have $\vec{\alpha} = \sharp$ as well as $\vec{\beta}_i = \sharp$ for all $i < n$ then we have the classical case of a context free rule. We therefore call an index scheme **context free** if it has this form. We call it **linear** if $\vec{\beta}_i \neq \sharp$ for at most one $i < n$. Context free schemata are therefore also linear but the converse need not hold. One uses the following suggestive notation. $A[\,]$ denotes an A with an arbitrary stack; on the other hand, A is short for $A\sharp$. Notice for example the following rule.

$$\mathtt{A}[\mathtt{i}] \to \mathtt{B}[\,] \quad \mathtt{A} \quad \mathtt{C}[\mathtt{ij}]$$

This is another form for the scheme

$$\mathtt{Ai} \to \mathtt{B} \quad \mathtt{A}\sharp \quad \mathtt{Cij}\ ,$$

which in turn comprises all rules of the following form

$$\mathtt{A}\vec{x}\mathtt{i} \to \mathtt{B}\vec{x} \quad \mathtt{A}\sharp \quad \mathtt{C}\vec{x}\mathtt{ij}\ .$$

Definition 5.6.1 *An* ***index grammar*** *is a sextuple* $G = \langle S, A, N, I, \sharp, R\rangle$ *where* A*,* N*, and* I *are pairwise disjoint finite sets not containing* $\sharp$*,* $S \in N$ *the start symbol and* R *a finite set of index schemata over* A*,* N*,* I *and* $\sharp$*.* G *is called* ***linear*** *if all its index schemata are linear.*

The notion of a derivation can be formulated over strings as well as trees. (To this end one needs A, N and I to be disjoint. Otherwise the category symbols cannot be uniquely reconstructed from the strings.) The easiest is to picture an index grammar as a grammar $\langle S, N, A, R\rangle$, where in contrast to a context free rule set we have put an infinite set of rules which is specified by means of schemata, which may allow infinitely many instantiations. This allows us to transfer many notions to the new type of grammars. For example, it is easily seen that for an index grammar there is a 2–standard form which generates the same language.

The following is an example of an index grammar. Let $A = \{\mathtt{a}\}$, $N = \{\mathtt{S}, \mathtt{T}, \mathtt{U}\}$, $I = \{\mathtt{i}, \mathtt{j}\}$, and

$$\begin{array}{lll}
\mathtt{S}[\,] & \to & \mathtt{T}[\mathtt{j}] \\
\mathtt{T}[\,] & \to & \mathtt{T}[\mathtt{i}] \\
\mathtt{T}[\mathtt{i}] & \to & \mathtt{U}[\,] \\
\mathtt{U}[\mathtt{i}] & \to & \mathtt{U}[\,] \quad \mathtt{U}[\,] \\
\mathtt{U}[\mathtt{j}] & \to & \mathtt{a}
\end{array}$$

This defines the grammar G. We have $L(G) = \{\mathtt{a}^{2^n} : n \in \omega\}$. As an example, look at the following derivation.

$$\begin{array}{lllll}
\mathtt{S} & \to & \mathtt{Tj} & \to & \mathtt{Tji} \\
 & \to & \mathtt{Tjii} & \to & \mathtt{Tjiii} \\
 & \to & \mathtt{Ujii} & \to & \mathtt{UjiUji} \\
 & \to & \mathtt{UjiUjUj} & \to & \mathtt{UjUjUjUj} \\
 & \to & \mathtt{aUjUjUj} & \to & \mathtt{aaUjUj} \\
 & \to & \mathtt{aaaUj} & \to & \mathtt{aaaa}
\end{array}$$

Index grammars are therefore quite strong. Nevertheless, one can show that they too can only generate **PTIME**–languages. (For index grammars one can define a variant of the chart–algorithm This variant also needs only polyomial time.) Of particular interest are the linear index grammars.

Now we turn to the equality between LIGs and TAGs. Let G be an LIG; we shall construct a TAG which generates the same constituent structures. We shall aim for roughly the same proof as with context free grammars. The idea is again to look for nodes x and y with equal label $X\vec{x}$. This however can fail. For on the one hand we can expect to find two nodes with identical label from N, but they may have different index stack. It may indeed happen that no such pair of nodes exists. Therefore we shall introduce the first simplification. We only allow rules of the following form.

$$\begin{array}{lcl} X[i] & \to & Y_0 \dots Y_{j-1}\; Y_j[\,]\; Y_{j+1} \dots Y_{n-1} \\ X[\,] & \to & Y_0 \dots Y_{j-1}\; Y_j[i]\; Y_{j+1} \dots Y_{n-1} \\ X & \to & Y_0 \dots Y_{n-1} \\ X & \to & a \end{array}$$

In other words, we only admit rules that stack or unstack a single letter, or which are context free. Such a grammar we shall call **simple**. It is clear that we can turn G into simple form while keeping the same constituent structures. Then we always have the following property. If x is a node with label $X\vec{x}$ and if x immediately dominates the node x' with label $Y\vec{x}i$ then there exists a node $y' \leq x'$ with label $V\vec{x}i$ which immediately dominates a node with label $W\vec{x}$. At least the stacks are now identical, but we need not have $Y = V$. To get this we must do a second step. We extend the set of nonterminals to the set of all pairs $\langle A, B\rangle$ where $A, B \in N$. Further, every pair gets a superscript from $\{o, a, e\}$. The superscripts keep score of the fact whether at this point we stack an index (a), we unstack a letter (e) or we do nothing (o). The index alphabet is now $N^2 \times I$. The rules above are now reformed as follows. (For the sake of perspicuity we assume that $n = 3$ and $j = 1$.) For a rule of the second type we put all rules of the form

$$\langle X, X'\rangle^a \to \langle Y_0, Y_0'\rangle^{a/o} \quad \langle Y_1, Y_1'\rangle^{a/o}[\langle X, X', i\rangle] \quad \langle Y_2, Y_2'\rangle^{a/o} .$$

So we stack in addition to the index i also the information about the label with which we have started. The superscript a is obligatory for $\langle X, X'\rangle$! From the rules of the first kind we make rules of the following form.

$$\langle X, X'\rangle^{a/o}[\langle W, Y_1', i\rangle] \to \langle Y_0, Y_0'\rangle^{a/o} \quad \langle W, Y_1'\rangle^{e} \quad \langle Y_2, Y_2'\rangle^{a/o} .$$

However, we shall also add these rules:

$$(\S) \qquad \langle Y_1, Y_1'\rangle^{e} \to \langle Y_1', Z\rangle^{a/o},$$

for all $Y_1, Y_1', Z \in N$. The rules of the third kind are replaced thus.

$$\langle X, X'\rangle^{o} \to \langle Y_0, Y_0'\rangle^{a/o} \quad \langle Y_1, Y_1'\rangle^{a/o} \quad \langle Y_2, Y_2'\rangle^{a/o}$$

Finally, the rules of the fourth kind are replaced by these rules.

$$\langle X, X'\rangle^{o} \to a .$$

We call this grammar $G^{\clubsuit}$. We shall at first see why G and $G^{\clubsuit}$ generate the same constituent structures. To this end, let us be given a $G^{\clubsuit}$–derivation. We then get a G–derivation as follows. Every symbol of the form $\langle X, X'\rangle^{a/e/o}$ is replaced by X, every stack symbol $\langle X, X', i\rangle$ by i. Subsequently, the rules of type (§) are skipped. This yields a G–derivation, as is easily checked. It gives the same constituent structure. Conversely, let a G–derivation be given with associated ordered labelled tree $\mathfrak{B}$. Then going from bottom to top we do the following. Suppose a rule of the second type has been applied to a node x and that i has been stacked. Then look for the highest node $y < x$ where the index i has been unstacked. Let y have the label B, x the label A. Then replace A by $\langle A, B\rangle^{a}$ and the index i on all nodes up to y by $\langle A, B, i\rangle$. In between x and y we insert a node y^* with label $\langle A, B\rangle^{e}$. y^* has y as its only daughter. y keeps at first the label B. If however no symbol has been stacked at x then exchange the label A by $\langle A, A'\rangle^{o}$, where A' is arbitrary. If one is at the bottom of the tree, one has a $G^{\clubsuit}$–tree. Again the constituent structures have been kept, since only unary rules have been inserted.

Now the following holds. If at x the index $\langle A, B, i\rangle$ has been stacked then x has the label $\langle A, B\rangle^{a}$ and there is a node y below

x at which this index is again removed. It has the label $\langle A, B\rangle^e$. We say that y is **associated to** x. Now define as in the case of context free languages center trees as trees whose associated string is a terminal string and in which no pair of associated nodes exist. In is easy to see that in such trees no symbol is ever put on the stack. No node carries a stack symbol and therefore there are only finitely many such trees. Now we define the adjunction trees. These are trees in which the root has label $\langle A, B\rangle^a$ exactly one leaf has a nonterminal label and this is $\langle A, B\rangle^e$. Further, in the interior of the tree no pair of associated nodes shall exist. Again it is clear that there are only finitely many such trees. They form the basic set of our adjunction trees. However, we do the following. The labels $\langle X, X'\rangle^o$ we replace by $\langle X, X'\rangle$, the labels $\langle X, X'\rangle^a$ and $\langle X, X'\rangle^e$ by $\langle X, X'\rangle^\triangledown$. (Root and associated node get an adjunction prohibition.) Now the proof is as in the context free case.

Now let conversely a TAG $G = \langle \mathsf{C}, N, A, \mathsf{A}\rangle$ be given. We shall construct a LIG which generates the same constituent structures. To this end we shall assume that all trees from C and A are based on pairwise disjoint sets of nodes. Let K be the union all sets of nodes. This is our set of nonterminals. The set A is our set of indices. Now we formulate the rules. Let $i \to j_0 \quad j_1 \cdots j_{n-1}$ be a local subtree of a tree. (A) i is not central. Then let

$$i \to j_0 \quad j_1 \cdots j_{n-1}$$

be a rule. (B) Let i be root of $\mathfrak{T}$ and j_k central (and therefore not a distinguished leaf). Then let

$$i[\,] \to j_0 \quad j_{k-1} \quad j_k[\mathfrak{T}] \quad j_{k+1} \cdots j_{n-1}$$

be a rule. (C) Let j_k be a distinguished leaf $\mathfrak{T}$. Then let

$$i[\mathfrak{T}] \to j_0 \quad j_{k-1} \quad j_k[\,] \quad j_{k+1} \cdots j_{n-1}$$

be a rule. (D) Let i be central in $\mathfrak{T}$, but not a root and j_k central but not a distinguished leaf. Then let

$$i[\,] \to j_0 \cdots j_{k-1} \quad j_k[\,] \quad j_{k+1} \cdots j_{n-1}$$

be a rule. Nothing else shall be a rule. This defines the grammar G^I. (This grammar may have start trees over distinct start symbols. This can be remedied.) Now we claim that this grammar generates the same constituent structures over A. This is done by induction over the length of the derivation. Let $\mathfrak{T}$ be a center tree, say $\mathfrak{T} = \langle B, <, \sqsubset, \ell\rangle$. Then let $\mathfrak{T}^I := \langle B, <, \sqsubset, \ell^I\rangle$, where $\ell^I(i) := i$ if i is nonterminal and $\ell^I(i) := \ell(i)$ otherwise. One establishes easily that this tree is derivable. Now let $\mathfrak{T} = \langle B, <, \sqsubset, \ell\rangle$ and $\mathfrak{T}^I = \langle B, <, \sqsubset, \ell^I\rangle$ already be constructed; let $\mathfrak{U} = \langle C, <', \sqsubset', \ell'\rangle$ result from $\mathfrak{T}$ by adjoining a tree $\mathfrak{B}$ to a node x. By making x into the root of an adjoined tree we get $B \subseteq C$, $<' \cap B^2 = <$, $\sqsubset' \cap B^2 = \sqsubset$ and $\ell' \restriction B = \ell$. Now $\mathfrak{U}^I = \langle C, <', \sqsubset', \ell'^I\rangle$. Further, there is an isomorphism between the adjunction tree $\mathfrak{B}$ and the local subtree induced on $C \cup \{x\}$. Let $\pi : C \cup \{x\} \to B$ be this isomorphism. Put $\ell'^I(y) := \ell^I(y)$ if $y \in B - C$. Put $\ell'^I(y) := \pi(y)$ if $\pi(y)$ is not central; and put $\ell'^I(y) := \ell'^I(x) := \ell^I(x)$ if y is distinguished leaf. Finally, assume $\ell^I(x) = X\vec{x}$, where X is a nonterminal symbol and $\vec{x} \in I^*$. If y is central but not root or leaf then put

$$\ell'^I(y) := \pi(y)\vec{x}\mathfrak{B} \,.$$

Now it is easily checked that the so defined tree is indeed derivable in G^I. We have to show likewise that if $\mathfrak{U}$ is derivable in G^I there exists a tree $\mathfrak{U}^A$ with $(\mathfrak{U}^A)^I \cong \mathfrak{U}$ which is derivable in G. To this end we use the method of disembedding. One looks for nodes x and y such that they have the same stack, $x > y$, there is no element in between the two that has the same stack. Further, there shall be no such pair in $\downarrow x - (\downarrow y \cup \{x\})$. If one has such a tree, then it is easily seen that it is isomorphic to an adjunction tree. One disembeds this tree and gets a tree which is strictly smaller. (Of course, the existence of such a tree must still be shown. This is done as in the context free case. Choose x of minimal height such that such there exists a $y < x$ with identical stack. Subsequently, choose y maximal with this property. In $\downarrow x - (\downarrow y \cup \{x\})$ there can then be no pair x', y' of nodes with identical stack such that $y' < x'$. Otherwise, x would not be minimal.) We summarize.

Theorem 5.6.2 *A set of constituent structures is generated by a linear index grammar if and only if it is generated by a TAG.*

We also say that these types of grammars are equivalent in constituent analysis.

A rule is called **right linear** if the index is only passed on to the right hand daughter. So, the right hand rule is right linear, the left hand rule is not:

$$\mathtt{A}[\,] \to \mathtt{B} \quad \mathtt{C}[\mathtt{i}] \quad \mathtt{B}, \qquad \mathtt{A}[\,] \to \mathtt{B} \quad \mathtt{C} \quad \mathtt{B}[\mathtt{i}].$$

An index grammar is called **right linear** if all of its rules are right linear. Hence it is automatically linear.

Theorem 5.6.3 (Michaelis & Wartena) *A language is generated by a right linear index grammar if and only if it is context free.*

Proof. Let G be right linear, $X \in N$. Define H_X as follows. The alphabet of nonterminals has the form $T := \{X^\diamond : X \in N\}$. The alphabet of terminals is the one of G, likewise the alphabet of indices. The start symbol is X. Now for every rule

$$A[\,] \to B_0 \ldots B_{n-1} \quad B[i]_n$$

we add the rule

$$A^\diamond[\,] \to A \quad B^\diamond[i]\ .$$

This grammar is right regular and generates a context free language (see the exercises). So there exists a context free grammar $L_X := \langle S^L_X, N^L_X, N, R^L_X \rangle$ which generates $L(H_X)$. (Here N is the alphabet of nonterminals of G but the terminal alphabet of L_X.) We assume that N^X_L is disjoint to our previous alphabets. We put $N' := \bigcup N^L_X \cup N$ as well as $R' := \bigcup R^L_X \cup R \cup R^-$ where R is the set of context free rules of G and R^- the set of rules $A[\,] \to B_0 \ldots B_{n-1}$ such that $A[\,] \to B_0 \ldots B_{n-1} \quad B_n[i] \in R$. Finally, let $G' := \langle S_L, N', A, R' \rangle$. G' is certainly context free. It remains to show that $L(G') = L(G)$. To this end let $\vec{x} \in L(G)$.

There exists a tree $\mathfrak{B}$ with associated string $\vec{x}$ which is derived from G. By induction over the height of this tree one shows that $\vec{x} \in L(G')$. The inductive claim is more exactly this one: *For every G–tree $\mathfrak{B}$ with associated string $\vec{x}$ there exists a G'–tree $\mathfrak{B}'$ with associated string $\vec{x}$; and if the root of $\mathfrak{B}$ carries the label $X\vec{x}$ then the root of $\mathfrak{B}'$ carries the label X.* If $\mathfrak{B}$ contains no stack symbols, this claim is certainly true. Simply take $\mathfrak{B}' := \mathfrak{B}$. Further, the claim is easy to see if the root has been expanded with a context free rule. Now let this not be the case; let the tree have a root with label U. Let P be the set of right hand nodes of $\mathfrak{B}$. For every $x \in P$ let $B(x)$ be that tree which contains all nodes which are below x but not below any $y \in P$ with $y < x$. It is easy to show that these sets form a partition of $\mathfrak{B}$. Let $u \prec x$, $u \notin P$. By induction hypothesis, the tree dominated by u can be restructured into a tree $\mathfrak{T}_u$ which has the same associated string and the same root label and which is generated by G'. The local tree of x in $B(x)$ is therefore an instance of a rule of R^-. We denote the tree obtained from x in such a way by $\mathfrak{B}'_x$. $\mathfrak{B}'_x$ is a G'–tree. Furthermore: if $y < x$, $y \in P$, and if $u < x$ then $u \sqsubset y$. Therefore we have that $P = \{x_i : i < n\}$ is an enumeration with $x_i > x_{i+1}$ for all $i < n - 1$. Let A_i be the root label of x_i in $\mathfrak{B}'_{x_i}$. The string $\prod_{i<n} A_i$ is a string of H_U. Therefore it is generated by L_U. Hence it is also generated by G'. So, there exists a tree $\mathfrak{C}$ associated to this string. Let the leaves of this tree be exactly the x_i and let x_i have the label A_i. Then we insert $\mathfrak{B}'_{x_i}$ at the place of x_i for all $i < n$. This defines $\mathfrak{D}$. $\mathfrak{D}$ is a G'–tree with associated string $\vec{x}$. The converse inclusion is left to the reader. ☐

We have already introduced Combinatorial Categorial Grammars (CCGs) in Section 3.4. The concept of these grammars was very general. In the literature, the term CCG is usually fixed — following Mark Steedman — to a particular variant where only those combinators may be added that perform function application and generalized function composition. In order to harmonize the notation, we define as follows.

$$\begin{array}{lcl} \alpha \mid^{+} \beta & := & \alpha/\beta, \\ \alpha \mid^{-} \beta & := & \beta\backslash\alpha. \end{array}$$

We take p_i as a variable for symbols from $\{+, -\}$. A **category** is a well formed string over $\{B, \texttt{(}, \texttt{)}, \mid^{+}, \mid^{-}\}$. As usual we agree on left associative bracketing so that in this particular case brackets are not needed. However, to be certain we shall assume that the brackets that we do not write actually are not present in the string. Hence $\texttt{a} \mid^{+} \texttt{b} \mid^{-} \texttt{c}$ is a category as is $\texttt{a} \mid^{+} \texttt{(b} \mid^{-} \texttt{c)}$, however $\texttt{((a} \mid^{+} \texttt{b)} \mid^{-} \texttt{c)}$ and $\texttt{(a} \mid^{+} \texttt{(b} \mid^{-} \texttt{c))}$ are not. A **block** is a sequence of the form $\mid^{+} \beta$ or of the form $\mid^{-} \beta$ where β is a category symbol. A **p–category** is a sequence of blocks, seen as a string. With this a category is simply a string of the form $\alpha \cdot \Delta$ where Δ is a p–category. If Δ and Δ' are p–categories, so is $\Delta \cdot \Delta'$. For a category α we define by induction the head, α, $K(\alpha)$, as follows.

1. $K(b) := b$.
2. $K(\alpha \mid^{p} \beta) := K(\alpha)$.

Lemma 5.6.4 *Every category α can be segmented as $\alpha = K(\alpha) \cdot \Delta$ where Δ is a p–category.*

If we regard the sequence simply as a string we can use $\cdot$ as the concatenation symbol of blocks as well as of sequences of blocks. We admit the following operations.

$$\begin{array}{llllll} \alpha \mid^{+} \beta & \circ_1 & \beta & := & \alpha, \\ \beta & \circ_2 & \alpha \mid^{-} \beta & := & \alpha, \\ \alpha \mid^{+} \beta & \circ_3^{n} & \beta \cdot \Delta^{n} & := & \alpha \cdot \Delta^{n}, \\ \beta \cdot \Delta^{n} & \circ_4^{n} & \alpha \mid^{-} \beta. & := & \alpha \cdot \Delta^{n} \end{array}$$

Here Δ^n is a variable for p–categories consisting of n blocks. In addition it is possible to restrict the choice of heads for α and β. This means that we define operations $\circ_i^{F,A,n}$ in such a way that

$$\alpha \circ_i^{L,R,n} \beta := \begin{cases} \alpha \circ^{n} \beta, & \text{if } K(\alpha) \in L, K(\beta) \in R, \\ \star, & \text{otherwise.} \end{cases}$$

This means that we have to step back from our ideal to let the categories be solely determined by the combinators.

Definition 5.6.5 *A* ***Combinatorial Categorial Grammar*** *is a categorial grammar which uses finitely many operations from* $\{\circ_1^{L,R}, \circ_2^{L,R} : L, R \subseteq B\} \cup \{\circ_3^{L,R,n}, \circ_4^{L,R,n} : n \in \omega, L, R \subseteq B\}$.

Notice by the way that $\circ_1 = \circ_3^0$ and $\circ_2 = \circ_4^n$. This simplifies the calculations.

Lemma 5.6.6 *Let G be a CCG over A and M the set of categories which are subcategories of some $\alpha \in \zeta(a)$, $a \in A$. Then the following holds. If $\vec{x}$ is a string of category α in G then $\alpha = \beta \cdot \Delta$ where $\alpha \in M$ and Δ is a p–category over M.*

The proof is by induction over the length of $\vec{x}$ and is left as an exercise.

Theorem 5.6.7 *For every CCG G there exists a linear index grammar H which generates the same trees.*

Proof. Let G be given. In particular, G associates with every letter $a \in A$ a finite set $\zeta(a)$ of categories. We consider the set M of subterms of categories from $\bigcup\langle\zeta(a) : a \in A\rangle$. This is a finite set. We put $N := M$ and $I := M$. By Lemma 5.6.6, categories can be written as pairs $\alpha[\Delta]$ where $\alpha \in N$ and Δ is a p–category over I. Further, there exist finitely many operations which we write as rules. Let for example $\circ_1^{L,R}$ be an operation. This means that we have rules of the form

$$\alpha \to \alpha|^+\beta \quad \beta$$

where $K(\beta) \in L$ and $K(\alpha) \in R$. We write this into linear index rules. Notice that in any case $\beta \in M$ because of Lemma 5.6.6. Furthermore, we must have $\alpha \in M^+$. So we write down all the rules of the form

$$(\dagger) \qquad \alpha \to \delta[\Delta] \quad \beta$$

where $\delta[\Delta] = \alpha|^{+}\beta$ for certain $\alpha, \Delta \in M^*$ and $\delta, \beta \in M$. We can group these into finitely many rule schemata. Simply fix β where $K(\beta) \in R$. Let $\mathbb{B}$ be the set of all sequences $\langle \gamma_i : i < p\rangle \in M^*$ whose concatenation is $|^{+}\beta$. $\mathbb{B}$ is finite. Now put for (†) all rules of the form

$$(\ddagger) \qquad \alpha'[] \to \alpha'[\Delta] \quad \beta$$

where $\alpha' \in M$ is arbitrary with $K(\alpha) \in L$ and $\Delta \in \mathbb{B}$. Now one can see easily that every instance of (†) is an instance of (‡) and conversely.

Analogously for the rules of the following form.

$$\beta \to \alpha|^{+}\beta \quad \beta$$

In a similar way we obtain from the operations $\circ_3^{L,R,n}$ rules of the form

$$(\star) \qquad \alpha \cdot \Delta^n \to \alpha|^{+}\beta \quad \beta \cdot \Delta^n$$

where $K(\alpha) \in L$ and $K(\beta) \in R$. Now it turns out that, because of Lemma 5.6.6 $\Delta^n \in M^n$ and $\beta \in M$. Only α may again be arbitrarily large. Nevertheless we have $\alpha \in M^+$, because of Lemma 5.6.6. Therefore $(\star)$ only corresponds to finitely many index schemata. $\Box$

The converse does not hold: for the trees which are generated by an LIG need not be 3–branching. However, the two grammar types are weakly equivalent.

Notes on this Section. There is descriptive characterization of indexed languages akin to the results of Chapter 6 in (Langholm, 2001). The idea there is to replace the index by a so called contingency function, which is a function on the nodes of the constituent structure that codes the adjunction history.

Exercise 191. Show the following claim. *For every index grammar G there is an index grammar H in 2–standard form such that $L(G) = L(H)$. If G is linear (context free) H can be chosen linear (context free) as well.*

Exercise 192. Prove the Lemma 5.6.6.

Exercise 193. Write an index grammar that generates the sentences of predicate logic. (See Section 2.6 for a definition.)

Exercise 194. Let *NB* be the set of formulae of predicate logic with equality and $\wedge$ in which every quantifier binds at least one occurrence of a variable. Show that there is no index grammar that generates *NB*. *Hint.* It is useful to concentrate on formulae of the form QM, where Q is a sequence of quantifiers and M a formula without quantifiers (but containing any number of conjuncts). Show that in order to generate these formulae from *NB*, a branching rule is needed. Essentially, looking top down, the index stack has to memorize which variables have been abstracted over, and the moment that there is a branching rule, the stack is passed on to both daughters. However, it is not required that the same variables occur in the left branch as in the right branch.

5.7 Compositionality and Constituent Structure

In this section we shall pick up the discussion of compositionality which we started in Chapter 3. Our concern is how constituent structure and compositionality constrain each other. Ultimately, this investigation will reveal something about the notion of constituent structure that is independent of syntactic arguments, and instead uses semantic criteria.

Recall once again Leibniz' Principle. It recurs in its formulation on the idea of substitution. However, exactly this notion is highly problematic in itself. Intuitively, if we substitute, say, `fast` by `good` in (5.7.1), we get (5.7.3) and not (5.7.2).

(5.7.1) `Simon is faster than Paul.`
(5.7.2) `*Simon is gooder than Paul.`
(5.7.3) `Simon is better than Paul.`

This is reminiscent of substitution in λ–calculus and predicate logic, which also is not exactly string substitution but something more complex. What obscures the facts even more is the fact that there seems to be no uniform algorithm to perform such substitution. Another problem, related to substitution, is that of determining occurrences. For example, does `cater` occur as a subexpression in `caterpillar`, `berry` as a subexpression of `cranberry`? Does `kick` enter with its meaning in `kick the bucket`? Worse still, does `tack` occur as a subexpression of `stack`, `rye` as a subexpression of `rice`? Obviously, no one would say that `tack` occurs in `stack`, and that its meaning is distinct from `needle` since you do not have `sneedle`. Likewise, in the formula $(\mathtt{p0} \wedge \mathtt{p01})$, the variable `p` does not occur, even though its string is a substring of the formula as a string.

To be able to make progress on these questions we have to resort to the distinction between language and grammar. As the reader will see in the exercises, there is a tight connection between the choice of constituents and the meanings these constituents can have. If we fix the possible constituents and their meanings this eliminates some but not all choices. However it does settle the question of identity in meaning and can then via Leibniz' Principle lead to a detection of subconstituents. For if there is no question whether two expressions have the same meaning, we conclude that they can be substituted for each other without change in the truth value in any given sentence on condition that they also have the same category. (Just an aside: sometimes substitution can be blocked by the exponents so that substitution is impossible even when the meanings and the category are the same. These facts are however generally ignored.) So, Leibniz' Principle does tell us something about which substitutions are legitimate, and which occurrences of substrings are actually occurrences of a given expression. If `sneedle` is a nonce word, we can safely conclude that `tack` has no proper occurrence in `stack` if substitution is simply string substitution. Moreover, Leibniz' Principle also says that if two expressions are intersubstitutable everywhere without changing the truth value, then they have the same meaning. (This allows for languages to not satisfy it.)

Definition 5.7.1 *Let* $\mathfrak{s}$ *be a structure term for the sign* σ*. Let* $\mathfrak{u}$ *be a subterm of* $\mathfrak{s}$ *which unfolds to the sign* τ*. Then we say that the sign* τ ***occurs in*** σ ***under the analysis*** $\mathfrak{s}$*. Suppose now* $\mathfrak{u}'$ *is a term unfolding to* τ'*, and substituting* $\mathfrak{u}'$ *for* $\mathfrak{u}$ *in* $\mathfrak{t}$ *is definite and unfolds to* σ'*. Then we say that* σ' ***results from*** σ ***by replacing*** τ ***with*** τ' ***under the analysis*** $\mathfrak{t}$*.*

This definition is complicated since a given sign may have different structure terms, and before we can define the substitution operation we must fix a structure term. This is particularly apparent when we want to define simultaneous substitution. Now in ordinary parlance one usually dispenses with mentioning the structure term. Second, substitution is typically defined not on signs but on exponents (which are called **expressions**). This, however, is dangerous and the reason for much confusion. For example, we have proved in Section 3.1 that recursively enumerable sign systems have compositional grammars. The proof used rather tricky functions on the exponents. As a result, there is no guarantee that if $\vec{y}$ is the exponent of τ and $\vec{x}$ the exponent of σ there is anything in $\vec{x}$ that resembles $\vec{y}$. Contrast this with context free grammars where a subexpression is actually also a substring. To see the dangers of this we discuss the theory of compositionality by Hodges. In (2001), Hodges discusses in passim the following principle, which he attributes to Tarski (from (Tarski, 1983)).

> *Tarski's Principle.* If there is a μ–meaningful structure term $[\mathfrak{s}/x]\mathfrak{u}$ unequal to $\mathfrak{u}$ and $[\mathfrak{s}'/x]\mathfrak{u}$ also is a μ–meaningful structure term with $[\mathfrak{s}'/x]\mathfrak{u} \sim_\mu [\mathfrak{s}/x]\mathfrak{u}$ then $\mathfrak{s} \sim_\mu \mathfrak{s}'$.

Notice that the typed λ–calculus satisfies this condition. Hodges dismisses Tarski's Principle on the following grounds.

(5.7.9) `The beast ate the meat.`
(5.7.10) `The beast devoured the meat.`
(5.7.11) `The beast ate.`
(5.7.12) `*The beast devoured.`

Substituting `The beast ate` by `The beast devoured` in (5.7.9) yields a meaningful sentence, but it does not with (5.7.11). Thus, substitutability in one construction does not imply substitutability in another, so the argument goes. The problem with this argument is that it holds only if we assume that substituting `The beast devoured` for `The beast ate` is what happens if we replace a structure term $\mathfrak{u}$ by $\mathfrak{u}'$ in a structure term for (5.7.9) and moreover that if we perform the same substitution in a structure term for (5.7.11) we get (5.7.12). Unfortunately, there is no guarantee for this. (The pathological examples of Section 3.1 should suffice to destroy this illusion.) What we need is a strengthening of the conditions on what can be possible operations on exponents. In the example sentence, the substituted strings are actually nonconstituents, so even under standard assumptions they do not constitute counterexamples. We can try a different substitution (`the meat` and ε), which is a constituent substitution under the standard analysis, but this is in no way necessary. It is not clear that Tarski's Principle is a good principle. Bit the alleged facts are not conclusive against it.

Obviously, what is needed is a restriction on the syntactic operations. We shall present two such restrictions, one based on polynomials, the other based on relevant λ–terms, as used in de Saussure–grammars. The idea is to require that the functions do not destroy any material. In this way the notion of composition does justice to the original meaning of the word. (Compositionality derives from Latin compositiō, *the putting together*.) Thus, every derived string is the result of applying some polynomial applied to certain vectors, and this polynomial determines the structure as well as — indirectly — the meaning and the category. In order not to get into trouble with syncategorematic occurrences, we shall simply exclude them.

Suppose $\mathsf{q} : (A^*)^{mn} \to (A^*)^n$ is a vector polynomial. Then there exist polynomials p_i, $i < n$, such that

$$\mathsf{q}(\langle \vec{x}_i^j : i < n, j < m \rangle) = \langle p_i(\langle \vec{x}_i^j : j < m \rangle) : i < n \rangle$$

We call $\prod_{i<n} p_i(\langle x_i^j : j < n\rangle)$ the **polynomial associated to** $\mathfrak{q}$. (Notice: no vector arrows!) This is an ordinary string polynomial. Notice that a polynomial not using constants for the letters of the alphabet is called a *term*.

Definition 5.7.2 *An n–ary term $p(\vec{x})$ is called **proper** if it depends of every x_i, $i < n$. A vector term $\mathfrak{q}$ representing the function $(A^*)^{mn} \rightarrow (A^*)^n$ is called a **proper** if the associated string term is proper.*

We shall deal with λ–terms below. Now that we have restricted the exponents and the functions on them we shall return to Leibniz' Principle and also turn to the choice of categories. We now assume that the sign grammars only use proper terms. This allows us to talk of constituents, as was shown in Section 5.4. For let $\vec{x}$ be given. If $\vec{x}$ is derivable and if the derivation inserts a symbol σ then $\vec{x}$ is the result of applying a proper term function to the exponent of σ. By doing this for all subconstituents we get the constituent structure. Now we define the following.

Definition 5.7.3 *A sequence $C = \langle \vec{w}_i : i < n+1\rangle$ of strings is called an n–**context**. A sequence $\mathfrak{v} = \langle \vec{v}_i : i < n\rangle$ **occurs in** $\vec{x}$ **in the context** C if*

$$\vec{x} = (\prod_{i<n} \vec{w}_i \vec{v}_i) \cdot \vec{w}_n \ .$$

We write $C(\mathfrak{v})$ in place of $\vec{x}$.

Notice that an n–sequence of strings can alternatively be regarded as an $n-1$–context. Let now Σ be a system of signs and $\mathtt{S}$ the distinguished category. Further, let

$$S(\Sigma) := \{\prod_{i<n} \vec{x}_i : \textit{there is } \mu : \langle\langle \vec{x}_i : i < n\rangle, \mathtt{S}, \mu\rangle \in \Sigma\} \ .$$

Now let $\vec{x}$ be a word from $S(\Sigma)$ which is decomposable as follows.

$$\vec{x} = (\prod_{i<n} \vec{w}_i \vec{v}_i) \cdot \vec{w}_n \ .$$

Further let us be given a sign grammar G over n–vectors of strings which generate Σ such that the occurrence of $\mathfrak{v} := \langle \vec{v}_i : i < n\rangle$ in $\vec{x}$ is a constituent. Now we define

$$\mathbb{C}_\Sigma(\mathfrak{v}) := \{C : C(\mathfrak{v}) \in S(\Sigma)\} .$$

This is the **context set** of $\mathfrak{v}$. If we have no categories — or if the algebra of categories is the trivial algebra **1** — then the context sets are simply the Husserlian categories. Suppose that we want to define what syntactic categories are. Then we need a criterion to decide which items belong to the same category. Obviously, we can use Husserl's criterion here. However, there is an intuition that certain sentences are semantically well formed but not syntactically. If this distinction can be made, the close connection between syntactic categories and context sets is broken. Nevertheless, structural linguistics, following Zellig Harris and others, tries to define categories in this way, using context sets. We shall only assume here that categories may not distinguish items finer than the context sets. (Harris held more particular ideas on the connection between categories and context sets, but we do not want to be more restrictive than absolutely necessary.)

Definition 5.7.4 *Let Σ be a system of signs and G a grammar which generates it. G is **natural** if the following holds. If $\mathbb{C}_\Sigma(\mathfrak{v}) = \mathbb{C}_\Sigma(\mathfrak{w})$ then $\mathfrak{v}$ and $\mathfrak{w}$ have the same category. This means: if $\langle \mathfrak{v}, \tau, \mu\rangle \in \Sigma$ then there exists a μ' with $\langle \mathfrak{w}, \tau, \mu'\rangle \in \Sigma$ and if $\langle \mathfrak{w}, \tau, \mu\rangle \in \Sigma$ then there exists a μ' with $\langle \mathfrak{v}, \tau, \mu'\rangle \in \Sigma$.*

Definition 5.7.5 *A **vectorial system of signs** is a set $\Sigma \subseteq (A^*)^n \times T \times M$ for some $n \in \omega$. Σ is **compositional** if there is a finite signature Ω and partial Ω–algebras $\mathfrak{Z} := \langle (A^*)^n, \{f^{\mathfrak{Z}} : f \in F\}\rangle$, $\mathfrak{T} := \langle T, \{f^{\mathfrak{T}} : f \in F\}\rangle$ and $\mathfrak{M} := \langle M, \{f^{\mathfrak{M}} : f \in M\}\rangle$ such that all functions are computable and Σ is exactly the set of 0–ary signs from $\mathfrak{Z} \times \mathfrak{T} \times \mathfrak{M}$. Σ is **strictly compositional** if there is a natural sign grammar for Σ in which for every $f \in F$ the function $f^{\mathfrak{Z}}$ is a proper vector term.*

The definition of compositionality is approximately the one that is used in the literature (modulo adaptation to systems of signs) while the notion of strict compositionality is the one which we think is the genuine notion. We say that a vector term is **incremental** if it is not the 0–ary constant ε and not the identity.

Theorem 5.7.6 *Let Σ be a strictly compositional vectorial system of signs. Further let for every mode M the functions M^{ε}, M^{τ} and M^{μ} be in **EXPTIME** and M^{ε} incremental. Then there exists an exponential algorithm which computes for given vector $\mathfrak{v}$ a meaning m and a category t such that $\langle \mathfrak{v}, t, m\rangle \in \Sigma$ if such m and t exist, and returns $\star$ otherwise. In particular, the string language defined by Σ is in **EXPTIME**.*

For a proof we note the following fact.

Lemma 5.7.7 *Let Σ be a strictly compositional system of signs with a natural grammar G in which all terms are proper and incremental. Let $\mathfrak{v}$ be a vector whose associated string has length n and t a structure term whose associated vector is $\mathfrak{v}$. Then $|t| \leq n$.*

The proof of Theorem 5.7.6 is left as an exercise. One has to list all structure terms of length $\leq n$ and then to compute the associated string vectors for these terms. For every term one needs $O(2^{cn})$ steps and there are $O(2^{dn})$ terms (c and d are positive real numbers). Therefore, there is a total need of $O(2^{(c+d)n})$ steps. If one requires that the functions M^{ε}, M^{τ} and M^{μ} are in **PTIME** the time requirement is not necessarily reduced to **PTIME**. A chart–algorithm is only polynomial since it does not need to distinguish between different meanings of vectors of the same category. However, since the meaning functions are obviously sensitive to the meanings, it may be necessary to store an exponential amount of different signs. This can lead to an exponential load of computation. As we shall see further below, the requirement that the terms be incremental is crucial for the success of the proof.

A particularly well known case where these conditions are violated is the analysis of quantification by Montague.

(5.7.13) `Nobody has seen Paul.`
(5.7.14) `No man has seen Paul.`

In the traditional, pre–Fregean understanding the subject of this sentence is `nobody` and the remainder of the sentence is the predicate. Further it is being said that the predicate is predicated of the subject. Hence, it is said of nobody that he has seen Paul. Now, who is this 'nobody'? Montague, in agreement with many others has claimed that the subject of this sentence is from a semantical point of view contrary to all expectations *not* the argument of its predicate. The subject denotes a so called *generalized quantifier*. Type theoretically the generalized quantifier of a subject has the type $e \to (e \to t)$. This is a set of properties, in this case the set of properties that are disjoint to the set of all humans. Since `has seen Paul` denotes a property (5.7.14) is true if and only if this property is in the set denoted by `no human` that is to say if it is disjoint with the set of all humans.

The development initiated by Montague has given rise to a rich literature. Generalised quantifiers have been a big issue for semantics for quite some time. Similarly for the treatment of intensionality that he proposed. He systematically assigned intensional types as meanings, which allowed to treat world or situation dependencies. The general ideas were laid out in the semiotic program and left room for numerous alternatives. This is what we shall discuss here. However, first we scrutinize Montague's analysis of quantifiers. The problem that he had to deal with was the ambiguity of sentences that were unambiguous with respect to the type assignment. Instructive examples are the following.

(5.7.15) `Some man loves every woman.`
(5.7.16) `Jan is looking for a unicorn.`

Both sentences are ambiguous. (5.7.15) may say that there is a man such that he loves all women. Or it may say that for every woman there is a man who loves her. In the first reading the universal quantifier is in the scope of the existential, in the second reading the existential quantifier is in the scope of the universal quantifier. Likewise, (5.7.16) may mean two things. That there is a real unicorn and Jan is looking for it, or Jan is looking for something that is in his opinion a unicorn. Here we are dealing with scopes between an existential quantifier and a modal operator. Since we are not dealing with modality in semantics, we shall concentrate on example (5.7.15). The problem with this sentence is that the universal quantifier `every woman` may not take scope over `some man loves`. This is so since the latter does not form a constituent. (Of course, we may allow it to be a constituent, but then this option creates problems of its own. In particular, this does not fit with the tight connections between the categories and the typing regime.) So, if we insist on our analysis there is only one reading: the universal quantifier is in the scope of the existential. Montague solved the problem in a rather particular way. He assumed an infinite set of pronouns called $\mathtt{he}_n$. These pronouns exist in inflected forms and in other genders as well, so we also have $\mathtt{him}_n$, $\mathtt{her}_n$ and so on. The remaining reading is created as follows. We feed the verb the pronoun $\mathtt{she}_0$ and get the constituent $\mathtt{loves\ she}_0$. This is an intransitive verb. Then we feed another pronoun, say $\mathtt{he}_0$ and get $\mathtt{he}_0\ \mathtt{loves\ she}_1$. Subsequently we combine this with `every man` and then with `a woman`. These operations delete the occurrences of the pronouns and introduce the corresponding phrases in their place. How this is done we are told by the following rule. Notice that it makes quantifiers into sentence adjuncts. Assume that we have two signs,

$$\sigma = \langle \vec{x}, \mathtt{S/S}, m\rangle, \quad \sigma' = \langle \vec{y}, \mathtt{S}, m'\rangle \ .$$

Then Q_n is the following function of signs:

$$Q_n(\sigma, \sigma') := (S_n(\vec{x}, \vec{y}), \mathtt{S}, m(\lambda x_n.m'))\rangle$$

Here $S_n(\vec{x}, \vec{y})$ is defined as follows. (Let us ignore gender and inflection at this point.)

1. For some k: $\vec{x} = \mathtt{he}_k$. Then $S_n(\vec{x}, \vec{y})$ is the result of replacing all occurrences of $\mathtt{he}_n$ by $\mathtt{he}_k$.

2. For all k: $\vec{x} \neq \mathtt{he}_k$. Then $S_n(\vec{x}, \vec{y})$ is the result of replacing the first occurrence of $\mathtt{he}_n$ by $\vec{x}$ and be deleting the index n on all other occurrences $\mathtt{he}_n$.

At last we have to give the signs for the pronouns. These are

$$\mathtt{P}_n := \langle \mathtt{he}_n, \mathtt{NP}, \lambda x_{\mathcal{P}}.x_{\mathcal{P}}(x_n)\rangle \ .$$

Depending on case, $x_{\mathcal{P}}$ is a variable of type $e \to t$ (for nominative pronouns) or of type $e \to (e \to t)$ (for accusative pronouns). Starting with the sign

$$\mathtt{L} \quad := \quad \langle \mathtt{loves}, (e\backslash t)/e, \lambda x_0.\lambda x_1.\mathsf{love}'(x_1, x_0)\rangle$$

we get the sign

$$\begin{aligned} & \mathtt{C}_{>}\mathtt{P}_1\mathtt{C}_{>}\mathtt{P}_0\mathtt{L} \\ = \; & \langle \mathtt{he}_1 \ \mathtt{loves} \ \mathtt{he}_0, t, \mathsf{loves}'(x_1, x_0))\rangle \end{aligned}$$

Now we need the following additional signs.

$$\begin{array}{lcl} \mathtt{E}_n & := & \langle \lambda y.S_n(\mathtt{every}, y), (t/t)/(e\backslash t), \lambda x.\lambda y.\forall x_n.(x(x_n) \to y)\rangle \\ \mathtt{S}_n & := & \langle \lambda y.S_n(\mathtt{some}, y), (t/t)/(e\backslash t), \lambda x.\lambda y.\exists x_n.(x(x_n) \wedge y)\rangle \\ \mathtt{M} & := & \langle \mathtt{man}, e\backslash t, \lambda x.\mathsf{man}'(x)\rangle \\ \mathtt{W} & := & \langle \mathtt{W}, e\backslash t, \lambda x.\mathsf{woman}'(x)\rangle \end{array}$$

If we feed the existential quantifier first we get the reading $\forall\exists$, and if we feed the universal quantifier first we the reading $\exists\forall$. The structure terms are as follows.

$$\begin{array}{c} \mathtt{C}_{>}\mathtt{C}_{>}\mathtt{E}_0\mathtt{W}\mathtt{C}_{>}\mathtt{C}_{>}\mathtt{S}_1\mathtt{M}\mathtt{C}_{>}\mathtt{P}_0\mathtt{C}_{>}\mathtt{P}_1\mathtt{L} \\ \mathtt{C}_{>}\mathtt{C}_{>}\mathtt{S}_1\mathtt{M}\mathtt{C}_{>}\mathtt{C}_{>}\mathtt{E}_0\mathtt{W}\mathtt{C}_{>}\mathtt{P}_0\mathtt{C}_{>}\mathtt{P}_1\mathtt{L} \end{array}$$

We have not looked at the morphological realization of the phrase $\vec{x}$. Number and gender must be inserted with the substitution. So, the case is determined by the local context, the other features are not. We shall not go into this here. (Montague had nothing to say about morphology as English has very little. We can only speculate what would have been the case if Montague had spoken another language.)

Recall from Section 2.6 that the grammar of sentences is very complex. Hence, since Montague defined the meaning of sentences to be closed formulae, it is almost unavoidable that something had to be sacrificed. In fact, Montague has violated several of our basic principles. First, there are now infinitely many lexical elements. Second the syntactic structure is not respected by the translation algorithm, and this yields the wrong results. Rather than taking an example from a different language, we shall exemplify the problems with the genitive pronouns. We consider `his` as the surface realization of `him's`, where `'s` is the so called Anglo Saxon genitive. Look for example at (5.7.17), which resulted from (5.7.18). Application of the above rules gives (5.7.19), however.

(5.7.17) `With a hat on his head every man looks better.`
(5.7.18) `With a hat on his`$_0$ `head he`$_0$ `looks better.`
(5.7.19) `With a hat on every man's head he looks better.`

This is not so because the possessive pronouns are also part of the rules. Of course, they have to be part of the semantic algorithm. It is because the wrong occurrence of the pronoun is being replaced by the quantifier phrase. This is due to the fact that the algorithm is ignorant about the syntactic structure (which the string reveals only partly) and second because the algorithm is order sensitive at places where it should better not be. Here, a modicum of GB–theory would probably do good (see 6.5). This theory has concerned itself extensively with the question which NP is realized in what way (in (1994), Fiengo and May speak quite plastically of *vehicle change*, to name the phenomenon, that a variable appears sometimes as a pronoun, sometimes as pro (an empty pronoun, see 6.5), sometimes as a lexical NP and so on.) The synchronicity between surface structure and derivational history which has been required in the subsequent categorial grammar, is not found with Montague. He bases himself on a distinction proposed by Church between **tectogrammar** (the inner structure, as von Humboldt would have called it) and **phenogrammar** (the outer structure, which is simply what we see). Montague admits quite powerful phenogrammatical operations, and it seems as if only the label distinguishes him from GB theory. For in principle his maps could be interpreted as transformations. We shall deal in a subsequent section with the problem of the pronouns postulated by Montague. Up to now they share the fate of the variables of predicate logic: with them a exponents of basic signs no compositional grammar is possible. We have to find some alternative.

We shall briefly discuss the problems of blocking and other apparent failures of compositionality. In principle, we have allowed the exponent functions to be partial. They can refuse to operate on certain items. This may be used, for example, in the analysis of defective words, for example `courage`. This word exists only in the singular (though it arguably has plural meaning). There is no form `courages`. In morphology, one says that each word has a root; in this case the root may simply be `courage`. The singular is formed by adding ε, the plural by adding `s` (see next section on how is to be understood). Now, the word `courage` does not let the plural be formed, therefore it is defective. If that is so, we are in trouble with Leibniz' Principle. Suppose we have a word X that is synonymous but exists in the singular and the plural. Then by Leibniz' Principle the two roots can never have the same meaning, since it is not possible to exchange them for each other in all contexts (the context where X appears in plural is a case in point). To avoid this, we must actually assume that there is no root form of `courage`. The classical grammar calls it a **singulare tantum**, a **singular only**. This is actually more appropriate. If namely this word has no root and exists only as a singular form, there is no context for the root to be exchanged. We remark here that English has **pluralia tanta** (**plural only** words), for example `troops`. In Latin, `tenebrae` (*darkness*), `indutiae` (*cease fire*) are ases examples. Additionally, there are words which are only formwise derived from the singular counterpart (or the root, for that matter). One such example is `forces` in meaning of 'troops', in Latin `fortunae` (*assets*) whose singular `fortuna` means 'luck'. Again, if both forms are assumed to be derived from the root, we have problems with the meaning of plural. Hence, some of these forms (typically — but not always — the plural form) will have to be part of the lexicon (that is, constitute a 0–ary mode).

Leibniz' Principle has also a striking application in deciding issues of strong generative capacity. Recall the facts from (Radzinski, 1990). Mandarin has a special form of yes–no–question, which is formed by iterating the statement with the negation word in between. Although it is conceivable that Mandarin is context free as a string language, Radzinski argues that it is not strongly context free. Now, suppose we understand by strongly context free that there is a context free sign grammar. Then we shall show that under mild conditions Mandarin is not strongly context free. To simplify the dicussion, let us define the following. The string language is $M := G \cup G\Box\mathtt{bu}\Box G$, where G is a context free language. Further, there is a meaning function μ defined on G. We extend μ to M by positing that

$$\mu(\vec{x}\Box\mathtt{bu}\Box\vec{y}) := \begin{cases} \mu(\vec{x}) \vee \neg\mu(\vec{y}) & \text{if } \vec{x} \neq \vec{y} \\ \mu(\vec{x})? & \text{if } \vec{x} = \vec{y} \end{cases}$$

It is assumed that meanings of the first kind are distinct from the meanings of the second kind (indeed, since the first are disjunctive statements and the second questions). Assume that there are two distinct expressions $\vec{u}$ and $\vec{v}$ of G such that $\mu(\vec{u}) = \mu(\vec{v})$. Then, by Leibniz' Principle, we must conclude that they cannot be substituted for each other. If G has a sublanguage of the form $\{\vec{r}\,\vec{z}^i\,\vec{s} : i \in \omega\}$ such that $\mu(\vec{r}\,\vec{z}^i\,\vec{s}) = \mu(\vec{r}\,\vec{z}_j\,\vec{s})$ for all i, j, it follows that G cannot be context free. For by the Pumping Lemma, there is k such that $\vec{z}^k$ can be adjoined into some $\vec{r}\,\vec{z}^i\,\vec{s}$ any number of times. Now look at the expressions

$$\vec{r}\,\vec{z}^i\,\vec{s}\Box\mathtt{bu}\,\Box\vec{r}\,\vec{z}^j\,\vec{s}$$

Adjunction is the result of substitution. However, the meaning of these expressions is $\top$ if $i \neq j$ and a yes–no question if $i = j$. Now put $j = i + k$. If we adjoin $\vec{z}^k$ on the left side, we get a yes–no question, if we substitute it to the right, we get the same meaning, which is not a yes–no question. It follows that one and the same syntactic substitution operation defines two different semantic functions, depending on where it is performed. Contradiction. Hence, by Leibniz' Principle, Mandarin is not strongly context free.

Likewise for Bahasa Indonesia. Recall that it forms the plural by reduplication. If the lexicon is finite, we can still generate the set of plural expressions. However, we must assume a distinct syntactic category for each noun. This is clearly unsatisfactory. For every time the lexicon grows by another noun, we must add a few rules to the grammar (see (Manaster-Ramer, 1986)). However, let us grant this point. Suppose, we have two nouns, $\vec{m}$ and $\vec{n}$, which have identical meaning. By Leibniz' Principle, any constituent occurrence of the first can be substituted by the second and vice versa. Therefore, if $\vec{m}$ has two constituent occurrences in $\vec{m}$-$\vec{m}$, we must have a word $\vec{m}$-$\vec{n}$ and a word $\vec{n}$-$\vec{m}$, and both mean the same as the first. This is precisely what is not the case. Hence, no such pair of words can exist if Bahasa Indonesia is strongly context free. Notice that our previous discussion of context sets does not help here. The noun $\vec{m}$ has a different context set as the noun $\vec{n}$, since it occurs in a plural noun $\vec{m}$-$\vec{m}$, where $\vec{n}$ does not occur. However, notice that the context set of $\vec{m}$ contains occurrences of $\vec{m}$ itself. If that circularity is removed, $\vec{m}$ and $\vec{n}$ become indistinguishable.

Exercise 195. Suppose that $A = \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}\}$, with the following modes.

$$\begin{array}{lcl} \mathsf{0} & := & \langle \mathtt{0}, Z, 0\rangle \\ \mathsf{S}(\langle \vec{x}, Z, n\rangle) & := & \langle S(\vec{x}), Z, n+1\rangle \end{array}$$

Here, $S(\vec{x})$ denotes the successor of $\vec{x}$ in the decimal notation, for example, $S(\mathtt{19}) = \mathtt{20}$. Let a string be given. What does a derivation of that string look like? When does a sign σ occur in another sign τ? Describe the exponent of $[\mathfrak{s}'/\mathfrak{s}]\mathfrak{t}$, for given structure terms

$\mathfrak{s}$, $\mathfrak{s}'$, $\mathfrak{t}$.

Exercise 196. Let $A = \{\texttt{0}, \dots, \texttt{9}, \texttt{+}, \texttt{(}, \texttt{)}, \cdot\}$. We shall present two ways for generating ordinary arithmetical terms. Recall that there is a convention to drop brackets in the following circumstances. (a) When the same operation symbol is used in succession (`5 + 7 + 4` in place of `(5 + (7 + 4))`), (b) When the enclosed term is multiplicative (`3·4 + 3` in place of `(3·4) + 3`). Moreover, (c) the outermost brackets are dropped. Write a sign grammar that generates triples $\langle \vec{x}, T, n\rangle$, where $\vec{x}$ is a term and n its value, where the conventions (a), (b) and (c) are optionally used. (So that you generate $\langle \texttt{5+7+4}, T, 16\rangle$ as well as $\langle \texttt{(5+(7+4))}, T, 16\rangle$). Now, apply Leibniz' Principle to the pairs `(5+7)` and `5+7`, `5+(7+4)` and `5+7+4`. What problems arise? Can you suggest a solution?

Exercise 197. (Continuing the previous exercise.) Write a grammar that treats every accidental occurrence of a term as a constituent occurrence in some different parse. For example, the occurrence of `3+4` in `3+4·7` is in the grammar of the previous exercise a nonconstituent occurrence, now however it shall be a constituent occurrence under some parse. Apply Leibniz' Principle. Show that `5+7` is not identical to `7+5`, however `2+5+7+3` is identical to `2+7+5+3`. Can you formulate some identities and nonidentities?

Exercise 198. The Latin verbs `aio` and `inquam` ('I say') are highly defective. They exist only in the present. Apart from one or two more forms (which we shall ignore for simplicity), here is a synopsis of what forms exist of these verbs andf contrast them with `dico`:

aio	inquam	dico	I say
ais	inquis	dicis	you(sg) say
ait	inquit	dicit	he says
		dicimus	we say
		dicitis	you(pl) say
aiunt	inquiunt	dicunt	they say

The morphology of `inquam` is irregular in that form (we expect `inquo`); also the syntax of `inquit` is somewhat peculiar (it is used epenthetically). Discuss whether `inquit` and `dicit` can be identical in meaning by Leibniz' Principle or not. Further, the verb `memini` is formwise in the perfect, but it means 'I remember'; similarly `odi` 'I hate'.

5.8 de Saussure Grammars

In his famous Cours de Linguistique Générale, de Saussure speaks about linguistic signs and the nature of language as a system of signs. In his view, a sign possesses two elements: a **signifier** and a **signified**. In our terms, these are the exponent and the meaning, respectively. Moreover, de Saussure says that signifiers are *linear*, without further specifying what he means by that. To a modern linguist all this seems obviously false: there are categories, and linguistic objects are structured, they are not linear. Notably Chomsky has repeatedly offered arguments to support this view. He believed that structuralism was fundamentally mistaken. In this section we shall show that this is not quite true. We shall offer a proof that any PTIME language possesses a grammar of this kind. Before we can do so, we need to say what we mean by linear.

Let us return to the idea mentioned earlier, that of λ–terms on strings. It should be clear from the exposition in Section 3.3 what a λ–term over the algebra of strings is. We call them **string terms**. We assume here that strings are typed, and that we have strings of different type. Assume for the moment that there is only one type, that of a string. Denote this by s. Then $\lambda x.\lambda y.y \cdot x$ is the function of reverse concatenation, and it is of type $s \rightarrow (s \rightarrow s)$. Now we wish to implement restrictions on these terms that make sure we do not lose any material. Call a λ–term **relevant** if for all subterms $\lambda x.N$, x occurs at least once free in N. $\lambda x.\lambda.y \cdot x$ is relevant, $\lambda x.\lambda y.x$ is not. Clearly, relevance is a necessary restriction. However, it is not sufficient. Let $\mathcal{P}$ and $\mathcal{Q}$ be variables of type $s \to s$, x a variable of type x. Then function composition, $\lambda x.\lambda \mathcal{P}.\lambda \mathcal{Q}.\mathcal{P}(\mathcal{Q}(x))$, is a relevant λ–term. But this term is problematic. Applying it leaves no visible trace on the string, it just changes the analysis. Thus, we shall also exclude *combinators*. This means, an admissible λ–term is a relevant term that contains $\cdot$ or an occurrence of a constant at least once.

Definition 5.8.1 *A string term is* ***progressive*** *if it is relevant and not a combinator.*

Definition 5.8.2 *A* ***de Saussurean sign*** *or simply* ***dS–sign*** *is a pair $\delta = \langle E, M\rangle$, where E is a progressive string term and M a λ–term over meanings. The* ***type*** *of δ is the pair $\langle \sigma, \tau\rangle$, where σ is the type of E and τ the type of M. If $\delta' = \langle E', M'\rangle$ is another de Saussurean sign then $\delta(\delta')$ is defined if and only if EE' is defined and MM' is defined, and then*

$$\delta(\delta') := \langle EE', MM'\rangle$$

In this situation we call δ the ***functor sign*** *and δ' the* ***argument sign****. A* ***de Saussurean grammar*** *is a finite set of dS–signs.*

So, the typing regime of the strings and the typing regime of the meanings do all the work here.

Proposition 5.8.3 *Let δ and δ' be dS–signs of type $\langle \sigma, \tau\rangle$ and $\langle \sigma', \tau'\rangle$, respectively. Then $\delta(\delta')$ is defined if and only if there are μ, ν such that $\sigma = \sigma' \to \mu$, $\tau = \tau' \to \nu$, and then $\delta(\delta')$ has type $\langle \mu, \nu\rangle$.*

The rest is actually completely analogous to the notion of an **AB**–grammar. Before we shall prove any results, we shall comment on the definition itself. In Montague grammar and much of categorial grammar there is a conflation of information that belongs to the realm of meaning and information that belongs to the realm of exponents. The category B/A, for example, were A and B are basic categories, tells us that the meaning must be a function of type

$\sigma(A) \to \sigma(B)$, and that the exponent giving us the argument must be found to the right. $A\backslash B$, on the other hand, is different only in that the exponent is to be found to the left. While this seems to be reasonable at first sight, it is already apparent that the syntactic categories simply repeat the semantic types. (This is why σ is a homomorphism.) The information concerning the semantic types is however not necessary, since the merger would fail anyhow if we did not supply signs with the correct types. So, we could leave to syntax to specify only the directionality. However, syntax is not well equipped for that. There are discontinuous constituents, which are not easily accommodated in categorial grammar. Much of the research can be seen as an attempt to upgrade the string handling potential in this direction. Notice further that the original categorial apparatus created distinctions that are nowhere attested. For example, suppose that we have an adjective. It is of category n/n. If it modifies a relational noun, however, it must be lifted to the category of a relational noun. The lifting will have to specify whether the noun is looking for its complement to its right or to its left. Generally, however, modifiers and functors do not care very much about the selectional and/or directional properties of their arguments. However, the categories force us to be explicit about every detail.

In sum, the apparatus of categorial grammar leaves a lot to be desired, and de Saussure grammars provide a solution to some of them. For there is now no need to iterate the semantic typing in the category, and the string handling has more power than in standard categorial grammar. We shall discuss a few applications of de Saussure grammars. These will illustrate both the strength as well as certain deficiencies of these grammars.

A striking fact about de Saussurean grammars is that they allow for word order variation in the most direct way. Let us take a binary relation, $\mathsf{see}' = \lambda x.\lambda y.\mathsf{see}'(y)(x)$. Its first argument is the direct object and the second its subject. We assume no case marking, so that the following nouns will be either subject or object.

$$\begin{array}{lcl} \text{JOHN} & := & \langle \texttt{John}, \mathsf{john}' \rangle \\ \text{MARY} & := & \langle \texttt{Mary}, \mathsf{mary}' \rangle \end{array}$$

Now we can give to the verb one of the following six signs. of which each corresponds to a different word order pattern. Here, we set $x \odot y := x \cdot \Box \cdot y$ where $\Box$ is the blank, which is an alphabetical symbol.

$$\begin{array}{lcll} \text{SEES}_0 & := & \langle \lambda x.\lambda y.y \odot x \odot \texttt{sees}, \mathsf{see}' \rangle & \text{SOV} \\ \text{SEES}_1 & := & \langle \lambda x.\lambda y.y \odot \texttt{sees} \odot x, \mathsf{see}' \rangle & \text{SVO} \\ \text{SEES}_2 & := & \langle \lambda x.\lambda y.\texttt{sees} \odot y \odot x, \mathsf{see}' \rangle & \text{VSO} \\ \text{SEES}_3 & := & \langle \lambda x.\lambda y.x \odot y \odot \texttt{sees}, \mathsf{see}' \rangle & \text{OSV} \\ \text{SEES}_4 & := & \langle \lambda x.\lambda y.x \odot \texttt{sees} \odot y, \mathsf{see}' \rangle & \text{OVS} \\ \text{SEES}_5 & := & \langle \lambda x.\lambda y.\texttt{sees} \odot x \odot y, \mathsf{see}' \rangle & \text{VOS} \end{array}$$

The structure term for a basic sentence expressing that John sees Mary is in all cases the same. It is $\text{SEES}_i(\text{MARY})(\text{JOHN})$ $(i < 6)$. Only that the order of the words is different in each case. For example,

$$\begin{array}{ll} & \text{SEES}_0(\text{MARY})(\text{JOHN}) \\ = & \langle \lambda x.\lambda y.y \odot x \odot \texttt{sees}, \mathsf{see}' \rangle (\langle \texttt{Mary}, \mathsf{mary}' \rangle)(\langle \texttt{John}, \mathsf{john}' \rangle) \\ = & \langle \lambda y.y \odot \texttt{sees Mary}, \mathsf{see}'(\mathsf{mary}') \rangle (\langle \texttt{John}, \mathsf{john}' \rangle) \\ = & \langle \texttt{John sees Mary}, \mathsf{see}'(\mathsf{mary}')(\mathsf{john}') \rangle \\ & \text{SEES}_4(\text{MARY})(\text{JOHN}) \\ = & \langle \lambda x.\lambda y.x \odot \texttt{sees} \odot y, \mathsf{see}' \rangle (\langle \texttt{Mary}, \mathsf{mary}' \rangle)(\langle \texttt{John}, \mathsf{john}' \rangle) \\ = & \langle \lambda y.\texttt{Mary sees} \odot y, \mathsf{see}'(\mathsf{mary}') \rangle (\langle \texttt{John}, \mathsf{john}' \rangle) \\ = & \langle \texttt{Mary sees John}, \mathsf{see}'(\mathsf{mary}')(\mathsf{john}') \rangle \end{array}$$

Notice that this construction can be applied to heads in general, and of heads with any number of arguments. Thus, de Saussurean grammars are more at ease with word order variation than categorial grammars. Moreover, in the case of OSV word order we see that the dependencies are actually crossing, since the verb does not form a constituent together with its subject.

We have seen in Section 5.3 how interpreted LMGs can be transformed into **AB**–categorial grammars using vector polynomials. Evidently, if we avail ourselves of vector polynomials (for example by introducing pair formation and projections and redefining the notion of progressivity accordingly) this result can be reproduced here for de Saussure grammars. Thus, de Saussure grammars suitably generalized are as strong as interpreted LMGs. However, we shall actually not follow this path. We shall not use pair formation; instead, we shall stay with the more basic apparatus. The examples that we shall provide below will give evidence that this much power is actually sufficient for natural languages, though some modifications will have to be made.

Next we shall look at plural in Bahasa Indonesia (or Malay). The plural is formed by reduplicating the noun. For example, the plural of `orang` (*man*) is `orang-orang`, the plural of `anak` (*child*) `anak-anak`. To model this, we assume one type of strings, n.

$$\textsc{plu} := \langle \lambda x.x \cdot \texttt{-} \cdot x, \lambda \mathcal{P}.\{x : \mathcal{P}(x)\}\rangle$$

Notice that the term $\lambda x.x \cdot x$ actually is progressive. It is clear that we cannot apply plural to a plural noun for semantic reasons. Now let us turn to English. In English, the plural is formed a suffixing an `s`. However, some morphophonological processes apply, and some nouns form their plural irregularly. Here is an (incomplete) list of plural formation. Above the line we find regular plurals, below irregular plurals.

Singular	Plural	
`tree`	`trees`	plain suffix
`bush`	`bushes`	e-insertion
`ox`	`oxen`	en–suffix
`fish`	`fish`	no change
`man`	`men`	vowel change

As we have outlined in Section 1.3, these differences are explained by postulating different plural morphs, one for each noun class.

We can account for that by introducing noun class distinctions in the semantic types. For example, we may introduce a semantic type for nouns endings in a nonsibilant, another for nouns ending in a sibilant, and so on. However, apart from introducing the distinction where it obviously does not belong, this proposal has another drawback. Recall, namely, that linguists speak of a plural *morpheme*, which abstracts away from the particular realizations of plural formation. Mel'čuk defines a morpheme as a set of signs that have identical category and identical meaning. So, for him the plural morpheme is simply the set of plural morphs. However, suppose that we want the morpheme to be a (de Saussurean) sign. Then its meaning is that of any of its morphs, but the string function cannot be a simple λ–term. For it acts differently on strings of different noun class. A striking example is German `Bank`. It can mean (i) money institute, (ii) something to sit on, (iii) banks of a river. The plural of `Bank` is `Banken` if it means a money institute, and `Bänke` otherwise. Now, the function forming the plural has no access to the meaning. Therefore we must distinguish two string classes and put the word `Bank` in both, with different meanings. Inevitably, we then have two signs for `Bank`, one to mean money institute and the other to mean something to sit on or the banks of a river. This is the common practice. The classes are morphological, that is, they do not pertain to meaning, just to form.

Thus we are led to the introduction of string types. We assume that types are ordered by some partial ordering $\leq$, so that if α and β are string types and $\alpha \leq \beta$ then any string of type α is a string of type β. The basic type s is the largest basic type. Returning now to English, we shall split the type n into various subtypes. In particular, we need the types ni, nr, of irregular and regular nouns. We shall first treat the regular nouns. The rule is that if a noun ends in a sibilant, the vowel `e` is inserted, otherwise not. Since this is a completely regular phenomenon, we can only define the string function if we have a predicate *sib* that is true of a string if and only if it ends in a sibilant. Further, we need to be able to define a function by cases. In computer languages, this would be written as follows.

$$\mathsf{rplu} := \lambda x.\mathsf{if}\ sib(x)\ \mathsf{then}\ x \cdot \mathsf{es}\ \mathsf{else}\ x \cdot \mathsf{s}$$

Thus, we must allow in our language the type of booleans. Moreover, unless we guarantee the uniqueness of the result, we must also make room for nonuniqueness. The latter is not so difficult to do, but we shall leave the matter as is. Notice a further problem. The minute that we admit different types we have to be specific about the type of the resulting string. This is not an innocent matter. The operation $\cdot$ is defined on all strings. Suppose now that `tree` is a string of type nr, which type does `trees` have? Obviously, we do not want it to be just a string, and we may not want it to be of type nr again. (The difference between regular and irregular is needed only for plural formation.) Also, as we shall see below, there are operations that simply change the type of a string without changing the string itself. Hence we shall move from a system of implicit typing to one of explicit typing (see (Mitchell, 1990) for an overview). Rather than using variables for each type, we use a single set of variables. λ–abstraction is now written $\lambda x : \sigma.M$ in place of $\lambda x.M$, where x is a variable of type σ in the latter case. We shall also use the notation $M : \tau$ in the body to say that M is assigned the type τ. Thus, $\lambda x : nr.x \cdot \mathtt{s} : n$ denotes the function from nr to n which appends `s`. The reader may recall from Section 4.1 the idea that strings can be taken to mean different things, depending on what type they are paired with. Internally, the typed strings are represented as pairs $\langle \vec{x}, \sigma \rangle$, where $\vec{x}$ is the string and σ its type. The operation $M : \tau$ does the following: it evaluates M on $\vec{x}$, and gives it the type τ. Now, the function is also defined for all $\sigma' \leq \sigma$, so we finally have

$$(\lambda x : \sigma.M : \tau)(\langle \vec{x}, \sigma' \rangle) = \begin{cases} \langle [\vec{x}/x]M, \tau \rangle & \text{if } \sigma' \leq \sigma, \\ * & \text{otherwise.} \end{cases}$$

Now we turn to the irregular plural. Here we face two choices. We may simply take all singular and plural nouns as being in the lexicon; or we devise rules for all occurring subcases. The first is not a good idea since it does not allow us to say that `ox` and the plural morpheme actually occur in `oxen`. The sign is namely an unanalyzable unit. So we discard the first alternative and turn to the second. In order to implement the plural we again need a predicate of strings that tells us whether a string equals some given string. The minimum we have to do is to introduce equality $\doteq$ on strings. This allows to define the plural by cases. However, suppose we add a binary predicate $suf(x, y)$ which is true of x and y if and only if x is a suffix of y. Then the regular plural can be defined also as follows:

$$\mathsf{rplu} := \lambda x : nr.(\mathsf{if}\ (suf(\mathtt{s}, x)\ \mathsf{or}\ suf(\mathtt{sh}, x)); \mathsf{then}\ x \cdot \mathtt{es}\ \mathsf{else}\ x \cdot \mathtt{s}) : n$$

Moreover, equality is definable from *suf*.

Now take another case, causatives. Many verbs may become causative. Examples are `laugh`, `drink`, `clap`.

(5.8.1) `The audience laughed the conductor off the stage.`
(5.8.2) `The manager drank his friends under the table.`
(5.8.3) `The audience was clapping the musician back onto the stage.`

In all these cases the meaning of the causative is regularly formed, so that we may actually assume that there is a sign that performs the change. But it leaves no visible trace (see the exercises). Thus, we must also assume that we have operators that perform type conversion. In the type system we have advocated above they can be succinctly represented by

$$\lambda x : \sigma.x : \tau$$

Now, the conversion of a string of one type into another is often accompanied by morphological marking. For example, the gerund in English turns a verb into a noun (`singing`). It is formed regularly by suffixing `ing`. So, it has the following sign:

$$\textsc{ger} = \langle \lambda x : v.x \cdot \mathtt{ing} : n, \lambda x.x \rangle$$

The semantics of these nominalizations is rather complex (see (Hamm and van Lambalgen, 2001)), so have put the identity for simplicity here. Signs that consist of nothing more than a type conversion are called **conversionemes** in (Mel'cuk, 1993 200). Obviously, they are not progressive in the intuitive sense. For we can in principle change a string from σ to τ and back; and we could do this as often as we like. However, there is little harm in admitting such signs. The additional complexity can be handled in much the same way as unproductive context free rules.

Another challenge is Swiss German. Since we do not want to make use of products, it is not obvious how we can instrumentalize the λ–terms to get the word order right. Here is how it can be done. We distinguish the main (inflected) verb from its subordinate verbs, and raising from nonraising verbs.

noninflected nonraising tr	$\langle \lambda x.\lambda y.\lambda z.y \odot x \odot z \odot \mathtt{aastriche}, \mathsf{paint}' \rangle$
inflected nonraising tr	$\langle \lambda x.\lambda y.y \odot x \odot \mathtt{aastricht}, \mathsf{paint}' \rangle$
noninflected nonraising itr	$\lambda y.\lambda z.y \odot z \odot \mathtt{schwimme}, \mathsf{sim}' \rangle$
inflected nonraising itr	$\langle \lambda x.x \odot \mathtt{schwimmt}, \mathsf{swim}' \rangle$
noninflected raising	$\langle \lambda \mathcal{P}.\lambda x.\lambda y.\mathcal{P}(x)(y \odot \mathtt{laa}), \mathsf{let}' \rangle$
inflected raising	$\langle \lambda \mathcal{P}.\lambda x.\mathcal{P}(x)(\mathtt{laa}), \mathsf{let}'$

Here, x, z are variables over NP–cluster strings, y a variable over verb–cluster strings, and $\mathcal{P}$ a variable for functions from NP–cluster strings to functions from verb–cluster strings to strings. Further, we distinguish between raised subjects and nonraised subjects.

raised subject	$\langle \lambda \mathcal{P}.\lambda x.\lambda y.\mathcal{P}(x \odot \mathtt{Jan})(y), \mathsf{jan}' \rangle$
nonraised subject	$\langle \mathtt{Jan}, \mathsf{jan}' \rangle$
nonraised object	$\langle \mathtt{es\ huus}, \mathsf{the\ house}' \rangle$

Let us see how this works.

$$\begin{aligned}
& \textsc{hälfe}(\textsc{aastriche}(\textsc{es huus})) \\
= \; & \textsc{hälfe}(\langle \lambda y.\lambda z.y \odot \texttt{es huus} \odot z \odot \texttt{aastriche}, \mathsf{paint}'(\mathsf{the\ house}')\rangle) \\
= \; & \langle (\lambda \mathcal{P}.\lambda x.\lambda y.\mathcal{P}(x)(y \odot \mathsf{hälfe}))(\lambda y.\lambda z.y \odot \texttt{es huus} \odot z \odot \texttt{aastriche}), \\
& \quad \mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}'))\rangle \\
= \; & \langle \lambda x.\lambda y.x \odot \texttt{es huus} \odot y \odot \texttt{hälfe aastriche}, \\
& \quad \mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}'))\rangle
\end{aligned}$$

This is, as is easily checked, of the same type as the entry of an noninflected nonraising intransitive verb. Indeed, doing another round produces a similar string term.

$$\begin{aligned}
& \textsc{laa}(\textsc{d'chind}(\textsc{hälfe}(\textsc{aastriche}(\textsc{es huus})))) \\
= \; & \textsc{laa}(\textsc{d'chind}(\langle \lambda x.\lambda y.x \odot \texttt{es huus} \odot y \odot \texttt{hälfe aastriche}, \\
& \quad \mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}'))\rangle \\
= \; & \textsc{lönd}(\langle \lambda \mathcal{P}.\lambda x.\lambda y.\mathcal{P}(x \odot \texttt{d'chind})(y)(\lambda x.\lambda y.x \cdot \texttt{es huus} \odot y \odot \texttt{hälfe aastriche}), \\
& \quad \mathsf{the\ children}'(\mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}'))))\rangle) \\
= \; & \textsc{laa}(\langle \lambda x.\lambda y.x \cdot \texttt{d'chind es huus} \cdot y \cdot \texttt{hälfe aastriche}, \\
& \quad \mathsf{the\ children}'(\mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}'))))\rangle) \\
= \; & \langle (\lambda \mathcal{P}.\lambda x.\lambda y.\mathcal{P}(x)(y \odot \texttt{laa}))(\lambda x.\lambda y.x \odot \texttt{d'chind es huus} \odot y \odot \texttt{hälfe aastriche}), \\
& \quad \mathsf{let}'(\mathsf{the\ children}'(\mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}')))))\rangle \\
= \; & \langle \lambda x.\lambda y.x \odot \texttt{d'chind es huus} \odot y \odot \texttt{laa hälfe aastriche}, \\
& \quad \mathsf{the\ children}'(\mathsf{let}'(\mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}')))))\rangle
\end{aligned}$$

The reader may check that if in place of the uninflected raising verb we had inserted the inflected form, the λ–term would have been closed off.

$$\begin{aligned}
& \textsc{mer}(\textsc{lönd}(\textsc{d'chind}(\textsc{hälfe}(\textsc{aastriche}(\textsc{es huus})))) \\
= \; & \textsc{mer}(\langle \lambda x.x \odot \texttt{d'chind es huus lönd hälfe aastriche}, \\
& \quad \mathsf{we}'(\mathsf{let}'(\mathsf{the\ children}'(\mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}'))))))\rangle \\
= \; & \langle \texttt{mer d'chind es huus lönd hälfe aastriche}, \\
& \quad \mathsf{we}'(\mathsf{let}'(\mathsf{the\ children}'(\mathsf{help}'(\mathsf{paint}'(\mathsf{the\ house}'))))))\rangle
\end{aligned}$$

Notice that the structure term has the form of the corresponding English structure. The λ–terms simply transliterate it into Swiss German.

Now let us take a different phenomenon, case agreement inside a noun phrase. In many languages, adjectives agree in case with the noun they modify. We take our example from Finnish. The phrase `iso juna` *a/the big train* inflects in the singular as follows. (We show only a fraction of the case system.)

nominative	`iso juna`
genitive	`ison junan`
allative	`isolle junalle`
inessive	`isossa jonassa`

In the case at hand, it is the same suffix that is suffixed to the adjective as well as the noun. Now, suppose we analyze the allative as a suffix that turns a caseless noun phrase into a case marked noun phrase. Then we want to avoid analyzing the allative `isolle junalle` as consisting of occurrences of the allative case. We want to say that it occurs once, but is spelled out twice. To achieve this, we view adjectives as modifiers of caseless nouns in the following way.

$$\text{ISO} := \langle \lambda x.\lambda y.y(\texttt{iso}) \odot y(x), \mathsf{big}' \rangle$$

Here, x has the type of caseless nouns, y the type of caseless–into–case marked nouns. This sign combines with the following sign

$$\text{TRAIN} := \langle \mathsf{juna}, \mathsf{train}' \rangle$$

to give

$$\text{ISO}(\text{JUNA}) = \langle \lambda y.y(\texttt{iso}) \odot y(\texttt{juna}), \mathsf{big}'(\mathsf{train}') \rangle$$

Finally, assume the following sign for the allative.

$$\text{ALL} := \langle \lambda x.x \cdot \texttt{lle}, \mathsf{move\text{-}to}' \rangle$$

Then the last two signs combine to

$$\text{ALL}(\text{ISO}(\text{JUNA})) = \langle \texttt{isolle junalle}, \mathsf{move\text{-}to}'(\mathsf{big}'(\mathsf{train}')) \rangle$$

This has the advantage that the composition of signs is much like the semantic structure, and that we can stack as many adjectives as we like: the case ending will automatically be distributed to all constituents. Notice that LMGs cannot achieve this effect in the same way. There is a fixed limit on the number of occurrences that can be controlled at the same time. Thus, de Saussure grammars sometimes provide a different analysis from LMGs. We remark here that the present analysis conforms to the ideas proposed in (Harris, 1963), who considers agreement simply as a multiple manifestation of a single morpheme. Case assignment can also be handled in a rather direct way. Standardly, a verb that takes a case marked noun phrase is assumed to select the noun phrase as a noun phrase of that case. Instead, however, we may assume that the sign for a case marking verb actually carries the case marker and fixes it onto the NP. The Finnish verb `tuntua` (*to resemble*) selects ablative case. Assume that it has an ablative marked argument that it takes directly to its right. Then its sign may be assumed to be like this (taking the 3rd person singular present form).

$$\textsc{tuntuu} := \langle \lambda x.\mathtt{tuntuu} \odot \lambda x.x \cdot \mathtt{lta}, \mathsf{resemble}' \rangle$$

Why should this at all be a good proposal? The reason is that if we simply insist that the noun phrase comes equipped with the correct case, then it enters with its ablative case meaning rather than with its typical NP meaning. Notice namely that the ablative has an unmotivated appearance here. Semantically it is the same as if the verb were transitive. Notice that the fact that `tuntua` selects something in the ablative is a coincidence in this setup. The ablative form is directly put onto the complement selected. This is not the best way of arranging things, and in (Kracht, 2002) a proposal has been made to remedy the situation.

There is a list of potential problems for de Saussure grammars. We mention a few of them. The plural in German is formed with some stems by umlauting them (see Section 1.3). This is (at least on the surface) an operation that is not additive. Another problem what is known as **portmanteau realization**. We exemplify this phenomenon the gradation of Latin adjectives. Recall that adjectives in many languages possess three forms: a positive (`happy`) a comparative (`happier`) and a superlative (`happiest`). This is so in Latin. In the following list the example above the line are regularly formed, the ones below not.

Positive	Comparative	Superlative
`parvus`	`parvior`	`parvissimus`
`beatus`	`beatior`	`beatissimus`
`bonus`	`melior`	`optimus`
`malus`	`peior`	`pessimus`

Interesting is however the fact that it is not the comparative or superlative suffix that is irregular: it is the root itself. The expected form `bonior` is replaced by `melior`: the root changes from `bon` to `mel`. (The English adjective `good` is also an example.)

Exercise 199. Recall from Section 3.5 the notion of a combinatory extension of categorial grammar. We may attempt the same for de Saussure grammars. Define a new mode of combination, `B`, which is defined as follows.

$$\mathtt{B}(\langle E, M\rangle)(\langle E', M'\rangle) := \langle \lambda x.E(E'(x)), \lambda y.M(M'(y))\rangle$$

Here, E is of type $\mu \to \nu$, E' of type $\lambda \to \mu$ and x of type λ, so that the string term $\lambda x.E(E'(x))$ is of type $\lambda \to \nu$. Likewise for the semantics. Show that this extension does not generate different signs, it just increases the set of structure terms. Contrast this with the analogous extension of **AB**–grammars. Look especially at mixed composition rules.

Exercise 200. Review the facts from Exercise 5.3 on Arabic. Write a de Saussurean grammar that correctly accounts for them. *Hint.* This is not so simple. First, define *schemes*, which are functions of type $s \to (s \to (s \to (s \to (s \to (s \to s)))))$. They provide a way of combining consonantism (root) and vocalism. The first three arguments form the consonantism, the remaining three the vocalism. The change in consonantism or vocalism can be defined on schemes before inserting the actual consonantism and vocalism.

Exercise 201. Write a de Saussurean grammar that generates the facts of Chinese shown in Exercise 5.3.

Exercise 202. In European languages, certain words do not inflect for case (these are adverbs, relative clauses and other) and moreover, no word can more than two cases. Define case marking functions that achieve this. (If you need concrete examples, you may elaborate the Finnish example using English substitute words.)

Exercise 203. We have talked briefly in Section 5.1 about Australian case marking systems. We shall simplify the facts (in particular the word order) as follows. We define a recursive translation from terms t in Polish notation inductively as follows. We assume so called **case endings** c_i, $i < \Omega$. For a constant term c, put $c^\diamond := c$. If F is an n–ary function symbol, and t_i, $i < n$, terms then put

$$(Ft_0 \dots t_{n-1})^\diamond := F \odot t_0^\diamond \cdot \mathsf{c}_0 \odot t_1^\diamond \cdot \mathsf{c}_1 \odot \dots \odot t_{n-1}^\diamond \cdot \mathsf{c}_{n-1}$$

Write a de Saussurean grammar that generates the signs $\langle t^\diamond, t\rangle$.

Exercise 204. In many modern theories of grammar, so called **functional elements** play a fundamental role. Functional elements are elements that are responsible for the correct shape of the structures, but have typically very little — if any — content. A particularly useful idea is to separate the content of an element from its syntactic behaviour. For example, we may introduce the morphological type of a transitive verb (tv) without specifying any selectional behaviour.

$$\textsc{see} := \langle \mathtt{see} : tv, \mathsf{see}' \rangle$$

Then we assume one or two functional elements that turn this sign into the signs $\textsc{see}_i$, $i < 6$. Show how this can be done for the particular case of the signs SEE. Can you suggest a general recipe for words of arbitrary category? Can you see suggest a solution of the problem of ablative case selection in Finnish?

Chapter 6

The Model Theory of Linguistical Structures

6.1 Categories

Up to now we have used plain nonterminal symbols in our description of syntactic categories, which have no internal structure. For many purposes this is not a serious restriction but it does not allow to capture important regularities of language. We give an example. The finite verb of German agrees with the subject in person and number. This means that the verb has different forms depending on whether the o subject is in the 1st, 2nd or 3rd person, and whether it is singular or plural. For example, the following sentences are grammatical.

(6.1) `Ich sehe.`
I see–1.Sg

(6.2) `Du siehst.`
You.Sg see–2.Sg

(6.3) `Er/Sie/Es sieht.`
He/She/It see-3.Sg

(6.4) `Wir sehen.`
We see-1.Pl

(6.5) `Ihr seht.`
You.Pl see–2.Pl

(6.6) `Sie sehen.`
They see–3.Pl

By contrast, the following sentences are ungrammatical.

(6.7) *`Ich siehst/sieht/sehen/seht.`
I see-2.Sg/see–3.Sg/see–1/3.Pl/see–2.Pl

(6.8) *`Du sehe/sieht/sehen/seht.`
You.Sg see–1.Sg/see–3.Sg/see–1/3.Pl/see–2.Pl

How can we account for this? We may simply assume that there are 6 different kinds of subjects (1st/2nd/3rd person, singular/plural) as well as 6 different kinds of verbs (of which two are homophonous, namely those of the 1st and 3rd person plural). A different way has been proposed among other in the **Generalized Phrase Structure Grammar** (**GPSG**, see (Gazdar *et al.*, 1985)). We assume to have the following uniform rule.

$$(*) \qquad \mathtt{S} \to \mathtt{NP} \quad \mathtt{VP}$$

Here the symbols `S`, `NP` and `VP` are symbols not for a single category but for a whole set of them, in fact the labels are taken to be *descriptions of categories*. This means that these 'labels' can be combined using boolean connectives, such as negation, conjunction and disjunction. For example, if we introduce the properties `1`, `2` and `3` as well as `Sg` and `Pl` then our rule $(*)$ can be refined as follows:

$$\mathtt{S} \to \mathtt{NP} \wedge \mathtt{1} \wedge \mathtt{Sg} \quad \mathtt{VP} \wedge \mathtt{1} \wedge \mathtt{Sg}$$

Furthermore, we have the following terminal rules.

$$\texttt{NP} \wedge \texttt{1} \wedge \texttt{Sg} \to \texttt{ich}, \qquad \texttt{VP} \wedge \texttt{1} \wedge \texttt{Sg} \to \texttt{sehe}$$

Here $\texttt{NP} \wedge \texttt{1} \wedge \texttt{Sg}$ is the description of a category which is a noun phrase (NP) in the first person (1) singular (Sg). This means that we can derive the sentence (6.1). In order for the sentences (6.7) and (6.1.8) not to be derivable we now have to eliminate the rule ($*$). To be able to derive the sentences (6.2) – (6.6) on the other hand we have to introduce five more rules. These can however be fused into a single schematic rule. We write in place of NP [CAT : *np*], in place of 1 [PERS : *1*], and in place of Pl we write [NUM : *pl*]. Here, we call CAT, PER and NUM **attributes**, and *np*, *vp*, *1*, and so on **values**. In the context [CAT : *np*] we say that the attribute CAT has the value *np*. A set of pairs $[A : v]$, where A is an attribute and v a value is called an **attribute–value structure**.

The rule ($*$) is now replaced by the schematic rule (‡).

$$(\ddagger) \qquad \begin{bmatrix} \text{CAT} & : & s \end{bmatrix} \to \begin{bmatrix} \text{CAT} & : & np \\ \text{PER} & : & \alpha \\ \text{NUM} & : & \beta \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & vp \\ \text{PER} & : & \alpha \\ \text{NUM} & : & \beta \end{bmatrix}$$

Here, α and β are variables. However, they have different value range; α may assume values from the set $\{1, 2, 3\}$ β values from the set $\{sg, pl\}$. This fact shall be dealt with further below. One has to see to it that the properties inducing agreement are passed on. This means that the following rule also has to be refined in a similar way.

$$(\dagger) \qquad \texttt{VP} \to \texttt{V} \quad \texttt{NP}$$

This rule says that a VP may be a constituent comprising a (transitive) verb and an NP. The agreement features have to be passed on to the verb.

$$\begin{bmatrix} \text{CAT} & : & vp \\ \text{PER} & : & \alpha \\ \text{NUM} & : & \beta \end{bmatrix} \to \begin{bmatrix} \text{CAT} & : & v \\ \text{PER} & : & \alpha \\ \text{NUM} & : & \beta \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & np \end{bmatrix}$$

Now, there are languages in which the verb not only agrees with the subject but also with the object in the same categories. This means that it does not suffice to simply write [PER : α]; we also have to say whether α concerns the subject or the object. Hence the structure relating to agreement is further embedded into the structure. Now we write as follows.

$$\begin{bmatrix} \text{CAT} & : & vp \\ \text{PER} & : & \alpha \\ \text{NUM} & : & \beta \end{bmatrix} \to \begin{bmatrix} \text{CAT} & : & v \\ \text{AGRS} & : & \begin{bmatrix} \text{PER} & : & \alpha \\ \text{NUM} & : & \beta \end{bmatrix} \\ \\ \text{AGRO} & : & \begin{bmatrix} \text{PER} & : & \alpha' \\ \text{NUM} & : & \beta' \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & np \\ \text{PER} & : & \alpha' \\ \text{NUM} & : & \beta' \end{bmatrix}$$

It is clear that this rule does the job as intended. One can make it look even nicer by assuming also for the NP an embedded structure for the agreement complex. This is what we shall do below. Notice that the value of an attribute is now not only a single value but may in turn be an entire attribute–value structure. Thus, two kinds of attributes are distinguished. *1*, *sg* are called **atomic values**. Attributes which have only atomic values are called **Type 0 attributes**, all others are **Type 1 attirbutes**. This is the basic setup of (Gazdar *et al.*, 1988). In the so called **Head Driven Phrase–Structure Grammar** by Carl Pollard and Ivan Sag (**HPSG**, see (Pollard and Sag, 1994)) this has been pushed much further. In HPSG, the entire structure is encoded using attribute–value structures of the kind just shown. They code not only the linguistic features but also the syntactic structure itself. We shall study these structures from a theoretical point of view in this chapter. This is justified since in one or the other form they do appear in any linguistic theory. Before we enter this investigation we shall move one step further. The rules that we have introduced above use variables for values of attributes. This certainly is a viable option. However, HPSG has gone into a different direction here. It introduces what are in fact structure variables, whose role is to see to it that entire attribute–value structures are shared across certain members of an attribute–value structure. To see how this goes we continue with our example. Let us now write also an NP not as a flat attribute–value structure but let us embed the agreement related attribute value pairs as the value of an attribute AGR. A 3rd person NP in the plural is now represented as follows.

$$\begin{bmatrix} \text{CAT} & : & np \\ \text{AGR} & : & \begin{bmatrix} \text{NUM} & : & pl \\ \text{PER} & : & 3 \end{bmatrix} \end{bmatrix}$$

The value of the attribute AGR is now structured in the same way as the values of AGRS and AGRO. Now we can rewrite our rules with the help of structure variables as follows. The rule (‡) now assumes this form.

$$(\ddagger) \qquad \begin{bmatrix} \text{CAT} & : & s \end{bmatrix} \to \begin{bmatrix} \text{CAT} & : & np \\ \text{AGR} & : & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & vp \\ \text{AGRS} & : & \boxed{1} \end{bmatrix}$$

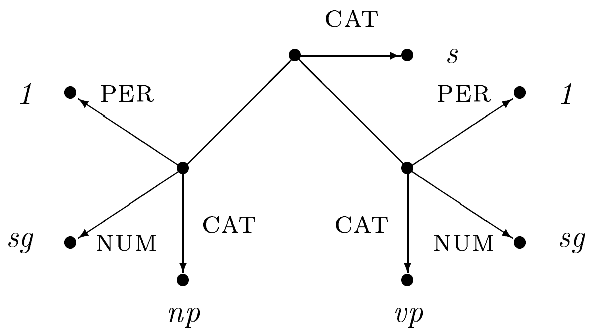
The rule that introduces the object now has this shape.

$$\begin{bmatrix} \text{CAT} & : & vp \\ \text{AGRS} & : & \boxed{1} \end{bmatrix} \to \begin{bmatrix} \text{CAT} & : & v \\ \text{AGRS} & : & \boxed{1} \\ \text{AGRO} & : & \boxed{2} \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & np \\ \text{AGR} & : & \boxed{2} \end{bmatrix}$$

The labels $\boxed{1}$ and $\boxed{2}$ are variables for attribute–value structures. If some variable occurs several times in a rule it always stands for the same attribute–value structure in all occurrences. This is how a variable works. However, this is precisely what is needed to ensure agreement. This is a possibility to avoid that agreement blows up the rule apparatus beyond recognition. The rules have become once again small and perspicuous. However, this method is only of restricted use for agreement. (The agreement facts of language are full of tiny details and exceptions, which then — unavoidably — make the introduction of more rules necessary.)

Now if attribute–value structures are only the description, then what are categories? In a nutshell, it is thought that categories are *Kripke–frames.* One assumes a set of vertices and associates

Figure 6.1: A syntactic structure



with each attribute a binary relation on this set. So, attributes are edge colours, atomic values turn into vertex colours. And a syntactic tree is not longer an exhaustively ordered tree with simple labels but an exhaustively ordered tree with labels having complex structure. Or, as it is more convenient, we shall assume that the tree structure itself also is codable by means of attribute–value structures, as we shall see later. The Figure 6.1 shows an example of a structure which — as one says — is **licensed** by the rule (‡). The literature on attribute–value structures is rich (see the books (Johnson, 1988) and (Carpenter, 1992)). In its basic appearance however it is quite simple. To begin, it is not useful to view attributes as objects. In fact, attribute–value structures are not objects, they are descriptions of objects. Moreover, they can be the values of attributes. Therefore we treat values like *np*, *1* as properties which can be combined with the usual boolean operations, for example $\neg$, $\wedge$, $\vee$ or $\rightarrow$. This has the advantage that we are now able to represent category of the German verb form `sehen` in either of the following ways.

$$\begin{bmatrix} \text{CAT} & : & v \\ \text{PER} & : & \mathit{1} \vee \mathit{3} \\ \text{NUM} & : & pl \end{bmatrix} \qquad \begin{bmatrix} \text{CAT} & : & v \\ \text{PER} & : & \neg \mathit{2} \\ \text{NUM} & : & pl \end{bmatrix}$$

The equivalence between these two follows only if we assume that the values of PER can be only *1*, *2* or *3*. This fact, however, is a fact of German, and will be part of the grammar of German. Notice that the collocation of attribute–value pairs into an attribute–value structure is nothing but the logical conjunction. So the left hand AVS can also be written down as follows.

$$[\text{CAT} : v] \wedge [\text{PER} : \mathit{1} \vee \mathit{3}] \wedge [\text{NUM} : pl]$$

One calls **underspecification** the fact that a representation does not fix an object in all detail but that it leaves certain properties unspecified. Disjunctive specifications are a case in point. However, they in fact provide not the most welcome case. Underspecification arises also when certain attributes are not contained in the AVS so that their value can actually be anything. For example, the English verb form `saw` may be represented thus.

$$\begin{bmatrix} \text{CAT} & : & v \\ \text{TEMP} & : & past \end{bmatrix}$$

This means that we have a verb in the past tense. The number and person are simply not mentioned. We can — but need not — write them down explicitly.

$$\begin{bmatrix} \text{CAT} & : & v \\ \text{TEMP} & : & past \\ \text{NUM} & : & \top \\ \text{PER} & : & \top \end{bmatrix}$$

Here $\top$ is the maximally unspecified value. We have — this is a linguistical, that is, empirical, fact —:

$$[\text{PER} : \mathit{1} \vee \mathit{2} \vee \mathit{3}]$$

From this we can deduce that `saw` also has the following representation.

$$\left[\begin{array}{lcl} \text{CAT} & : & v \\ \text{TEMP} & : & \textit{past} \\ \text{NUM} & : & \top \\ \text{PER} & : & \mathit{1} \vee \mathit{2} \vee \mathit{3} \end{array}\right]$$

Fact of language are captured by means of axioms. More on that later.

Since attribute–value pairs are propositions, we can combine them in the same way. The English verb form `see` would have the following grammatical representation.

$$\neg\left[\begin{array}{lcl} \text{CAT} & : & v \\ \text{PER} & : & \mathit{3} \\ \text{NUM} & : & sg \end{array}\right] \vee \left[\begin{array}{lcl} \text{CAT} & : & v \\ \text{NUM} & : & pl \end{array}\right]$$

This can be written down also as follows.

$$[\text{CAT} : v] \wedge (\neg([\text{PER} : \mathit{3}] \wedge [\text{NUM} : sg]) \vee [\text{NUM} : pl])$$

In turn, this can be simplified.

$$[\text{CAT} : v] \wedge (\neg[\text{PER} : \mathit{3}] \vee [\text{NUM} : pl])$$

The basis for this are the following universal laws. Since AVs are not the objects themselves but descriptions thereof, we may exchange one description of an object or class of objects by any other description of that same object or class of objects. We call an AVS **universally true** if it is always true, that is, if it holds of every object.

▷ If φ is a tautology of propositional logic then φ holds for all replacements of AVSs for the propositional variables.

▷ If φ is universally true, then so is $[\mathrm{X} : \varphi]$.

▷ $[\mathrm{X} : \varphi \to \chi]. \to .[\mathrm{X} : \varphi] \to [\mathrm{X} : \chi]$.

▷ If φ and $\varphi \to \chi$ are universally true then so is.

Most attributes are **definite**, that is, they can have at most one value in any object. For such attributes we also have

$$[\mathrm{X} : \varphi] \wedge [\mathrm{X} : \chi]. \to .[\mathrm{X} : \varphi \wedge \chi]$$

Definite attributes are the norm. Sometimes, however, one needs attributes of the standard sort; they are called **set valued** to distinguish them from the definite ones.

In the literature, two different kinds of logical languages have been proposed to talk about categories. The first is the so called **monadic second order predicate logic** (MSO), which is a fragment of **second order logic** (SO). Second order logic extends standard first order predicate logic as follows. There additionally are variables and quantifiers for predicates of any given arity $n \in \omega$. The quantifiers are also written $\forall$ and $\exists$ and the variables are P^n_i, $n, i \in \omega$. Here, n tells us that the variables is a variable for n–ary relations. So, $P := \{P^n_i : n, i \in \omega\}$ is the set of predicate variables for unary predicates and $V := \{x_i : i \in \omega\}$ the set of object variables. We write $P^n_i(\vec{x})$ to say that P_i applies to (the tuples) $\vec{x}$. If φ is a formula so are $(\forall P^n_i)\varphi$ and $(\exists P^n_i)\varphi$. The structures are the same as those of predicate logic (see Section 3.8). So, they are triples $\mathfrak{M} = \langle M, \{f^{\mathfrak{M}} : f \in F\}, \{r^{\mathfrak{M}} : r \in R\}\rangle$, where M is a nonempty set, $f^{\mathfrak{M}}$ the interpretation of the function f in $\mathfrak{M}$ and $r^{\mathfrak{m}}$ the interpretation of the relation r. A **model** is a triple $\langle \mathfrak{M}, \gamma, \beta\rangle$ where $\mathfrak{M}$ is a structure $\beta : V \to M$ a function assigning to each variable an element from M and $\gamma : P \to \wp(M)$ a function assigning to each n–ary predicate variable of P an n–ary relation on M. The relation $\langle \mathfrak{M}, \gamma, \beta\rangle \vDash \varphi$ is defined inductively.

$$\langle \mathfrak{M}, \gamma, \beta\rangle \vDash P^n_i(\vec{x}) \qquad \Leftrightarrow \qquad \beta(\vec{x}) \in \gamma(P^n_i)$$

We define $\gamma \sim_P \gamma'$ if $\gamma'(Q) = \gamma(Q)$ for all $Q \neq P$.

$$\begin{array}{lcl} \langle \mathfrak{M}, \gamma, \beta\rangle \vDash (\forall P)\varphi & \Leftrightarrow & \text{for all } \gamma' \sim_P \gamma \quad : \langle \mathfrak{M}, \gamma', \beta\rangle \vDash \varphi \\ \langle \mathfrak{M}, \gamma, \beta\rangle \vDash (\exists P)\varphi & \Leftrightarrow & \text{for some } \gamma' \sim_P \gamma : \langle \mathfrak{M}, \gamma', \beta\rangle \vDash \varphi \end{array}$$

We write $\mathfrak{M} \vDash \varphi$ if for all γ and β $\langle \mathfrak{M}, \gamma, \beta \rangle \vDash \varphi$. Monadic second order predicate logic is that fragment of SO which uses only predicate variables for unary relations ($n = 1$). When using MSO, we drop the superscript '1' in the variables P_i^1.

Another type of language that have been proposed are modal languages (see (Blackburn, 1993) and (Kracht, 1995)). We shall pick out one specific language that is actually an extension of the ones proposed in the quoted literature, namely the **quantified modal logic** (QML). This language possesses a denumerably infinite set $PV := \{p_i : i \in \omega\}$ of proposition variables, a set M of so called **modalities**, and a set C of propositional constants. And finally, there are next to the brackets the symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, $[-]$, $\langle - \rangle$, $\forall$ and $\exists$. Formulas (alias propositions) are defined inductively in the usual way. Moreover, if φ is a proposition, so is $(\forall p_i)\varphi$ and $(\exists p_i)\varphi$. The notion of Kripke–structure and Kripe–model remain the same. We define

$$\begin{array}{lcl} \langle \mathfrak{F}, \beta, x \rangle \vDash (\forall p)\varphi & \Leftrightarrow & \text{for all } \beta' \text{ with } \beta' \sim_p \beta : \langle \mathfrak{F}, \beta', x \rangle \vDash \varphi \\ \langle \mathfrak{F}, \beta, x \rangle \vDash (\exists p)\varphi & \Leftrightarrow & \text{for some } \beta' \text{ with } \beta' \sim_p \beta : \langle \mathfrak{F}, \beta', x \rangle \vDash \varphi \end{array}$$

We write $\langle \mathfrak{F}, \beta \rangle \vDash \varphi$ if for all $x \in F$ $\langle \mathfrak{F}, \beta, x \rangle \vDash \varphi$; we write $\mathfrak{F} \vDash \varphi$, if for all β we have $\langle \mathfrak{F}, \beta \rangle \vDash \varphi$.

We define an embedding of QML into MSO as follows. Let $R := \{r^m : m \in M\}$ and $C := \{Q^c : c \in K\}$. Then define $\varphi^\dagger$ inductively.

$$\begin{array}{lcl} p_i^\dagger & := & P_i(x_0) \\ c^\dagger & := & Q^c(x_0) \\ (\neg\varphi)^\dagger & := & \neg\varphi^\dagger \\ (\varphi_1 \wedge \varphi_2)^\dagger & := & \varphi_1^\dagger \wedge \varphi_2^\dagger \\ (\varphi_1 \vee \varphi_2)^\dagger & := & \varphi_1^\dagger \vee \varphi_2^\dagger \\ (\varphi_1 \rightarrow \varphi_2) & := & \varphi_1^\dagger \rightarrow \varphi_2^\dagger \\ ((\forall p_i)\varphi)^\dagger & := & (\forall P_i)\varphi^\dagger \\ ((\exists p_i)\varphi)^\dagger & := & (\exists P_i)\varphi^\dagger \\ ([m]\varphi)^\dagger & := & (\forall x_0)(r^m(x_0, x_i) \rightarrow [x_i/x_0]\varphi^\dagger) \\ (\langle m \rangle\varphi)^\dagger & := & (\exists x_0)(r^m(x_0, x_i) \wedge [x_i/x_0]\varphi^\dagger) \end{array}$$

Here in the last two clauses x_i is a variable that does not already occur in $\varphi^\dagger$. Now we have

Theorem 6.1.1 *Let φ be a formula of QML. Then $\varphi^\dagger$ is a formula of MSO over a corresponding signature, and for every Kripke–structure $\mathfrak{F}$: $\mathfrak{F} \vDash \varphi$ if and only if $\mathfrak{F} \vDash \varphi^\dagger$.*

Proof. We shall show the following. Assume $\beta : PV \to \wp(F)$ is a valuation in $\mathfrak{F}$ and that $x \in F$ and $\gamma : P \to \wp(F)$ and $\delta : V \to F$ valuations for the predicate and the object variables. Then if $\gamma(P_i) = \beta(p_i)$ for all $i \in \omega$ and $\delta(x_0) = w$ we have

$$(\ddagger) \qquad \langle \mathfrak{F}, \beta, w\rangle \vDash \varphi \qquad \Leftrightarrow \qquad \langle \mathfrak{F}, \gamma, \delta\rangle \vDash \varphi^\dagger$$

The claim now follows in this way. If β, w are given with $\langle \mathfrak{F}, \beta, w\rangle \nvDash \varphi$, then γ and δ can be named (uniquely) such that $\langle \mathfrak{F}, \gamma, \delta\rangle \nvDash \varphi^\dagger$; and if γ and δ are given with $\langle \mathfrak{F}, \gamma, \delta\rangle \nvDash \varphi^\dagger$, then β and x can be given with $\langle \mathfrak{F}, \beta, x\rangle \nvDash \varphi$. Now, however, for the proof of ($\ddagger$). The proof is done by induction. If $\varphi = p_i$ then $\varphi^\dagger = P_i(x_0)$ and the claim holds in virtue of the fact that $\beta(p_i) = \gamma(P_i)$ and $\gamma(x_0) = x$. Likewise for $\varphi = c \in C$. The steps for $\neg$, $\wedge$, $\vee$ and $\to$ are routine. Let us therefore consider $\varphi = (\exists p_i)\eta$. Let $\langle \mathfrak{F}, \beta, w\rangle \vDash \varphi$. Then for some β' which differs from β at most in p_i: $\langle \mathfrak{F}, \beta', w\rangle \vDash \eta$. Put γ' as follows: $\gamma'(P_i) := \beta'(p_i)$ for all $i \in \omega$. By induction hypothesis $\langle \mathfrak{F}, \gamma', \delta\rangle \vDash \eta^\dagger$ and γ' differs from γ at most in P_i. Therefore we have $\langle \mathfrak{F}, \gamma, \delta\rangle \vDash (\exists P_i)\eta^\dagger = \varphi^\dagger$, as desired. The argument can be reversed, and the case is therefore settled. Analogously for $\varphi = (\forall P_i)\eta$. Now for $\varphi = \langle m\rangle\eta$. Let $\langle \mathfrak{F}, \beta, w\rangle \vDash \varphi$. Then there exists a y with $w \; r^m \, y$ und $\langle \mathfrak{F}, \beta, y\rangle \vDash \eta$. Choose δ' such that $\delta'(x_0) = y$ and $\delta'(x_i) = \delta(x_i)$ for every $i > 0$. Then by induction hypothesis $\langle \mathfrak{F}, \gamma, \delta'\rangle \vDash \eta^\dagger$. If x_i is a variable that does not occur in $\eta^\dagger$ then let $\delta''(x_i) := \delta'(x_i)$, $\delta''(x_0) := w$ and $\delta''(x_j) := \delta'(x_j)$ for all $j \notin \{0, i\}$. Then $\langle \mathfrak{F}, \gamma, \delta''\rangle \vDash r^m(x_0, x_i); [x_i/x_0]\eta^\dagger = \varphi^\dagger$. Hence we have $\langle \mathfrak{F}, \gamma, \delta''\rangle \vDash \varphi^\dagger$. Now it holds that $\delta''(x_0) = w = \delta(x_0)$ and x_i is bound. Therefore also $\langle \mathfrak{F}, \gamma, \delta\rangle \vDash \varphi^\dagger$. Again the argument is reversible, and the case is proved. Likewise for $\varphi = [m]\eta$. ∎

Exercise 205. Prove Proposition **??**.

Exercise 206. Let 1, 2 and 3 be modalities. Show that a Kripke–structure satisfies the following formula if and only if $R(3) = R(1) \cup R(2)$.

$$\langle 3\rangle p \leftrightarrow \langle 1\rangle p \vee \langle 2\rangle p$$

Exercise 207. Let 1, 2 and 3 be modalities. Show that a Kripke–structure satisfies the following formula if and only if $R(3) = R(1) \circ R(2)$.

$$\langle 3\rangle p \leftrightarrow \langle 1\rangle\langle 2\rangle p$$

Exercise 208. Let r be a binary relation symbol. Show that in a model of MSO the following holds: $\langle \mathfrak{M}, \gamma, \beta\rangle \vDash Q(x, y)$ if and only if $x\ (r^{\mathfrak{M}})^*\ y$ (this means that y can be reached from x in finitely many R–steps).

$$Q(x, y) := (\forall P)(P(x) \wedge (\forall yz)(P(y) \wedge y\ r\ z. \rightarrow .P(z)). \rightarrow .P(y))$$

6.2 Axiomatic Classes I: Strings

For the purposes of this chapter we shall code strings in a new way. This way is a somewhat different formalization than the one discussed in Section 1.4. The differences are however not important.

Definition 6.2.1 *A* ***Z–structure over the alphabet*** A *is a triple of the form* $\mathfrak{L} = \langle L, \prec^{\mathfrak{L}}, \{Q_a^{\mathfrak{L}} : a \in A\}\rangle$*, where* L *is an arbitrary finite set,* $\{Q_a^{\mathfrak{L}} : a \in A\}$ *a system of pairwise disjoint subsets of* L *whose union is* L *and* $\prec^{\mathfrak{L}} \subseteq L \times L$ *a binary relation for which the following holds.*

1. $\langle L, \prec^{\mathfrak{L}}\rangle$ *is connected.*

2. *$\prec^{\mathfrak{L}}$ is cycle free.*

3. *From $x \prec^{\mathfrak{L}} y$ and $x \prec^{\mathfrak{L}} z$ follows $y = z$.*

4. *From $y \prec^{\mathfrak{L}} x$ and $z \prec^{\mathfrak{L}} x$ follows $y = z$.*

We write Q_a in place of $Q_a^{\mathfrak{L}}$ and $\prec$ in place of $\prec^{\mathfrak{L}}$ if there is no risk of confusion. Z–structures are not strings. However, it is not difficult to define a map which assigns a string to each Z–structure. However (if $L \neq \varnothing$) there are infinitely many *and a proper class* of Z–structures which have the same string associated with them.

Denote by MSO the set of sentences of monadic second order logic predicate logic which except for the logical symbols (quantifiers, boolean function symbols, equality) also contains the symbol $\underline{\prec}$, as well as a unary predicate constant $\underline{a}$ for every $a \in A$.

Definition 6.2.2 *Let $\mathcal{K}$ be a set or class of Z–structures over an alphabet A. Then $\mathsf{Th}\,\mathcal{K}$ denotes the set $\{\varphi \in \mathsf{MSO} :$ for all $\mathfrak{L} \in \mathcal{K} : \mathfrak{L} \vDash \varphi\}$. This we call the* MSO–theory of $\mathcal{K}$. *If conversely Φ is a set of sentences from MSO then let $\mathsf{Mod}\,\Phi$ be the set of all $\mathfrak{L}$ which satisfy every sentence from Φ. This we call the* ***model class of*** *Φ.*

Recall from Section 1.1 the notion of a context. It is easy to see that $\vDash$ together with the class of Z–structures and the MSO–formulae form a context. From this we directly get the following

Theorem 6.2.3 *The map $\rho : \mathcal{K} \mapsto \mathsf{Mod}\,\mathsf{Th}\,\mathcal{K}$ is a closure operator on the class of of classes Z–structures over A. Likewise $\lambda : \Phi \mapsto \mathsf{Th}\,\mathsf{Mod}\,\Phi$ is a closure operator on the set of all subsets of* MSO.

(We hope that the reader does not get irritated by the difference between classes and sets. In the usual set theory one has to distinguish between sets and classes. Model classes are except for trivial exception always classes while classes of formulae are always sets, because they are subclasses of the set of formulae. This difference can be neglected in what is to follow.) We now call the sets of the form $\lambda(\mathcal{K})$ **logics** and the classes of the form $\rho(\Phi)$ **axiomatic classes**. A class is called **finitely axiomatizable** if it has the form $\mathsf{Mod}(\Phi)$ for a finite Φ, while a logic is **finitely axiomatizable** if it is the logic of a finitely axiomatizable class. We call a class of Z–structures over A **regular** if it is the class of all Z–structures of a regular language. The logic of the class of all structures is denoted by the monadic second order order predicate logic. Formulae are called **valid** if they hold in all structures. This is equivalent to belonging to monadic second order predicate logic.

An important result of this chapter is the one from (Büchi, 1960).

Theorem 6.2.4 (Büchi) *A class of Z–structures is a finitely axiomatizable class if and only if it corresponds to a regular language which does not contain ε.*

This sentence says that with the help of monadic second order predicate logic we can only define regular classes of Z–structures. If one wants describe nonregular classes, one has to use stronger logical languages. The proof of this theorem requires a lot of work. Before we begin, we have to say something about the formulation of the theorem. By definition models are only defined on non–empty sets. This is why a model class always defines a language not containing ε. It is possible to change this but then the Z–structure of ε (which is actually unique) is a model of every formula, and then $\varnothing$ is regular but not MSO–axiomatizable. So, complete correspondence cannot be expected.

Let us begin with the simple direction. This is the claim that every regular class is finitely axiomatizable. For this end let $\mathcal{K}$ be a regular class and L the corresponding regular language. Then there exists a finite state automaton $\mathfrak{A} = \langle Q, i_0, F, \delta\rangle$ with $L(\mathfrak{A}) = L$. We may choose $Q := n$ for a natural number n and $i_0 = 0$.

Look at the following sentence.

$$
\begin{array}{lll}
\delta(\mathfrak{A}) := & (\forall xyz)(x \underline{\prec} y \wedge x \underline{\prec} z. \to y. \doteq z) & \text{(a)} \\
\wedge & (\forall xyz)(y \underline{\prec} x \wedge z \underline{\prec} x. \to .y \doteq z) & \text{(b)} \\
\wedge & (\forall P)\{(\forall xy)(x \underline{\prec} y \to (P(x) \to P(y))) \wedge (\exists x)P(x). & \\
& \quad \to .(\exists x)(P(x) \wedge (\forall y)(x \underline{\prec} y \to \neg P(y)))\} & \text{(c)} \\
\wedge & (\forall P)\{(\forall xy)(x \underline{\prec} y \to (P(y) \to P(x))) \wedge (\exists x)P(x). & \\
& \quad \to .(\exists x)(P(x) \wedge (\forall y)(y \underline{\prec} x \to \neg P(y)))\} & \text{(d)} \\
\wedge & (\forall P)\{(\forall xy)(x \underline{\prec} y \to (P(x) \leftrightarrow P(y))) \wedge (\exists x)P(x). & \\
& \quad \to .(\forall x)P(x)\} & \text{(e)} \\
\wedge & (\forall x) \bigvee \langle \underline{a}(x) : a \in A \rangle & \text{(f)} \\
\wedge & (\forall x) \bigwedge \langle \underline{a}(x) \to \neg \underline{b}(x) : a \neq b \rangle & \text{(g)} \\
\wedge & (\exists P_0 P_1 \dots P_{n-1}) & \\
& \qquad \{(\forall x)((\forall y)\neg(y \underline{\prec} x). \to .P_0(x)) & \\
& \qquad \wedge(\forall x)((\forall y)\neg(x \underline{\prec} y). \to . \bigvee \langle P_i(x) : i \in F \rangle) & \\
& \qquad \wedge(\forall xy)(x \underline{\prec} y \to \bigwedge[\langle \underline{a}(y) \wedge P_i(x). & \\
& \qquad\quad \to . \bigvee \langle P_j(y) : j \in \delta(i,a) \rangle : a \in A \rangle])\} & \text{(h)}
\end{array}
$$

Lemma 6.2.5 *Let $\mathfrak{L}$ be an* MSO*–structure. Then $\mathfrak{L} \vDash \delta(\mathfrak{A})$ if and only if the string of $\mathfrak{L}$ is in $L(\mathfrak{A})$.*

Proof. Let $\mathfrak{L} \vDash \delta(\mathfrak{A})$ and let $\mathfrak{L}$ be an MSO–structure. Then there exists a binary relation $\prec$ (the interpretation of $\underline{\prec}$) and for every $a \in A$ a subset $Q_a \subseteq L$. Because of the definition of $\delta(\mathfrak{A})$ $\prec$ is a relation for which the following holds: if $x \prec y$ and $x \prec z$ then $x \prec z$ (because of (a)); and if $y \prec x$ and $z \prec x$ then $y = z$ (because of (b)). Every nonempty subset which is closed under $\prec$–successors contains a last element (Formula (c)) and every nonempty subset which is closed under $\prec$–predecessors contains a first element (Formula (d)). Finally, every connected nonempty set is identical to L (Formula (e)). This means, as one easily shows, that L is finite and connected. Then first of all L contains a first element x_0, since it is not empty. Let inductively x_{i+1} be the (uniquely determined) $\prec$–successor of x_i. The set H of the x_i cannot be infinite, for otherwise it would be a successor closed set without last element. Hence it is finite. From this it also follows that $\prec$ is cycle free. Now it is also closed under predecessors, and therefore connected. Therefore it is equal to L. So, L is finite and connected under $\prec$.

Further we have this: every $x \in L$ is contained in at least one set Q_a (Formula (f)) and it is contained in at most one set (Formula (g)). Therefore $\mathfrak{L}$ is a Z–structure. We have to show that their string is in $L(\mathfrak{A})$. The last member of the conjunction, Formula (h), says that we can find sets $H_i \subseteq L$ for $i < n$ such that if x is the first element with respect to $\prec$ then $x \in H_0$, if x is the last element with respect to $\prec$ then $x \in H_j$ for some $j \in F$ and if $x \in H_i$, $y \in Q_a$, $x \prec y$ then $y \in H_j$ for some $j \in \delta(i, a)$. This means nothing but that the string is in $L(\mathfrak{A})$. (There is namely a biunique correspondence between accepting runs of the automaton and partitions into H_i. Under that partition $x \in H_i$ means nothing else than that the automaton is in state i at x in that run.) Now let $\mathfrak{L} \nvDash \delta(\mathfrak{A})$. Then either $\mathfrak{L}$ is not a Z–structure or there exists no accepting run of the automaton $\mathfrak{A}$. Hence the string is not in $L(\mathfrak{A})$. ☐

It is useful to understand that we can define $x \leq y$ as follows.

$$x \leq y := (\forall P)(P(x) \wedge (\forall wz)(P(w) \wedge w \underline{\prec} z. \rightarrow .P(z)) \rightarrow P(y))$$

Further we write $x < y$ for $x \leq y \wedge x \neq y$, as well as $x \prec y$ in place of $x \underline{\prec} y$. Now let the following structure be defined for a Z–structure $\mathfrak{L} = \langle L, \prec, \{Q_a : a \in A\}\rangle$.

$$M(\mathfrak{L}) := \langle L, \prec, \succ, <, >, \{Q_a : a \in A\}\rangle$$

A structure of the form $M(\mathfrak{L})$ we call an **MZ–structure**.

Now we shall prove the converse implication in the theorem of Büchi. To this end we shall make a detour. We define a modal language based on four modalities. We put $M := \{+, -, \prec, \succ\}$. The set of constants shall be $C := \{Q_a : a \in A\}$. The language so defined is called QML. Now we put

$$\begin{array}{rlll}\sigma := & \langle\prec\rangle p \to \langle +\rangle p & \wedge & \langle\succ\rangle p \to \langle -\rangle p \\ \wedge & p \to [\succ]\langle\prec\rangle p & \wedge & p \to [\prec]\langle\succ\rangle p \\ \wedge & \langle\prec\rangle p \to [\prec]p & \wedge & \langle\succ\rangle p \to [\succ]p \\ \wedge & \langle +\rangle\langle +\rangle p \to \langle +\rangle p & \wedge & \langle -\rangle\langle -\rangle p \to \langle -\rangle p \\ \wedge & [+]([+]p \to p) \to [+]p & \wedge & [-]([-]p \to p) \to [-]p \\ \wedge & \bigwedge\langle Q^a \to \neg Q^b : a \neq b\rangle & & \\ \wedge & \bigvee\langle Q^a : a \in A\rangle & & \end{array}$$

We call a structure **connected** if it is not of the form $\mathfrak{F} \oplus \mathfrak{G}$ with nonempty $\mathfrak{F}$ and $\mathfrak{G}$. As already mentioned QML–formulae cannot distinguish between connected and nonconnected structures.

Theorem 6.2.6 *Let $\mathfrak{F}$ be a connected Kripke–structure for* QML*. Put $Z(\mathfrak{F}) := \langle F, R(\prec), R(\succ), R(+), R(-), \{K(Q^a) : a \in A\}\rangle$. $\mathfrak{F} \vDash \sigma$ if and only if $Z(\mathfrak{F})$ is an MZ–structure over A.*

Proof. The proof is not hard but somewhat longish. We emphasize a few facts. (a) $\mathfrak{F} \vDash \langle\prec\rangle p \to \langle +\rangle p$ *if and only if* $R(\prec) \subseteq R(+)$. For this let $\mathfrak{F} \nvDash \langle\prec\rangle p \to \langle +\rangle p$. Then there exists a set $\beta(p) \subseteq F$ and a $x \in F$ such that $\langle\mathfrak{F}, \beta, x\rangle \vDash \langle\prec\rangle p; \neg\langle +\rangle p$. This means that there is a $y \in F$ with $y \in \beta(p)$ and $x\ R(\prec)\ y$. However, $x\ R(+)\ y$ can then not hold. This shows $R(\prec) \nsubseteq R(+)$. Let conversely $R(\prec) \nsubseteq R(+)$. Then there exist x and y with $x\ R(\prec)\ y$ but not $x\ R(+)\ y$. Now set $\beta(p) := \{y\}$. Then $\langle\mathfrak{F}, \beta, x\rangle \vDash \langle\prec\rangle p; \neg\langle +\rangle p$. (b) $\mathfrak{F} \vDash p \to [\succ]\langle\prec\rangle p$ *if and only if* $R(\succ) \subseteq R(\prec)^{\smile}$. This is shown essentially as in (a). (c) $\mathfrak{F} \vDash \langle\prec\rangle p \to [\prec]p$ *if and only if every point has at most one $R(\prec)$–successor.* Proof as in (a). (d) $\mathfrak{F} \vDash \langle +\rangle\langle +\rangle p \to \langle +\rangle p$ *if and only if $R(+)$ is transitive.* Also this is shown as in (a). (d) $\mathfrak{F} \vDash [+](p \to [+]p) \to [+]p$ *if and only if $R(+)$ is transitive and cycle free.* Because of (d) we restrict ourselves to proving this for transitive $R(+)$. Then let $\mathfrak{F} \nvDash [+]([+]p \to p) \to [+]p$. Then there exists a β and a x_0 with $\langle\mathfrak{F}, \beta, x\rangle \vDash [+](p \to [+]p); \langle +\rangle\neg p$. There exists a x_1 with $x_0\ R(+)\ x_1$ and $x_1 \notin \beta(p)$. Then $x_1 \vDash [+]p \to p$ and therefore $x_1 \vDash \langle +\rangle\neg p$. Because of the transitivity of $R(+)$ we also have

$x_1 \vDash [+]([+]p \rightarrow p)$. So we find an infinite chain $\langle x_i : i \in \omega\rangle$ such that $x_i\ R(+)\ x_{i+1}$. Therefore, $R(+)$ is not cycle free, since F is finite. Conversely, assume that $R(+)$ is not cycle free. Then there exists a set $\langle x_i : i \in \omega\rangle$ with $x_i\ R(+)\ x_{i+1}$ for all $i \in \omega$. Put $\beta(p) := \{y :$ there is $i \in \omega : y\ R(+)\ x_i\}$. Then it holds that $\langle \mathfrak{F}, \beta, x_0\rangle \vDash [+]([+]p \rightarrow p); \langle + \rangle \neg p$. For let $x\ R(+)\ y$. Case 1. For all z with $y\ R(+)\ z$ we have $z \vDash p$. Then there exists a i with $y\ R(+)\ x_i$ (for $R(+)$ is transitive). Hence $y \vDash p$. This shows $y \vDash [+]p \rightarrow p$. Since y was arbitrary we now have $x \vDash [+]([+]p \rightarrow p)$. Now $x_1 \nvDash p$ und $x_0\ R(+)\ x_1$. Hence $x_0 \vDash \langle + \rangle \neg p$, as required. The remaining properties are easy to check. ☐

Now we define an embedding of MSO to QML. To this end some preparations. As one convinces oneself easily the following laws hold.

1. $(\forall x)\varphi \leftrightarrow \neg(\exists x)(\neg\varphi)$, $\neg\neg\varphi \leftrightarrow \varphi$.
2. $(\forall x)(\varphi_1 \wedge \varphi_2) \leftrightarrow (\forall x)\varphi_1 \wedge (\forall x)\varphi_2$, $(\exists x)(\varphi_1 \vee \varphi_2) \leftrightarrow (\exists x)\varphi_1 \vee (\exists x)\varphi_2$.
3. $(\forall x)(\varphi_1 \vee \varphi_2) \leftrightarrow (\varphi_1 \vee (\forall x)\varphi_2)$, $(\exists x)(\varphi_1 \wedge \varphi_2) \leftrightarrow (\varphi_1 \wedge (\exists x)\varphi_2)$, if x does not occur freely in φ_1.
4. $(\forall x)(\varphi_1 \vee \varphi_2) \leftrightarrow (\varphi_2 \vee (\forall x)\varphi_1)$, $(\exists x)(\varphi_1 \wedge \varphi_2) \leftrightarrow (\varphi_2 \wedge (\exists x)\varphi_1)$, if x does not occur freely in φ_2.

Finally, for every variable $y \neq x$:

$$(\forall x)\varphi(x, y) \leftrightarrow (\forall x)(x < y \rightarrow \varphi(x, y)) \wedge (\forall x)(y < x \rightarrow \varphi(x, y)) \wedge \varphi(y, y).$$

We now define following quantifiers.

$$\begin{array}{lcl}
(\forall x < y)\varphi & := & (\forall x)(x < y \rightarrow \varphi), \\
(\forall x > y)\varphi & := & (\forall x)(x > y \rightarrow \varphi), \\
(\exists x < y)\varphi & := & (\exists x)(x < y \rightarrow \varphi), \\
(\exists x > y)\varphi & := & (\exists x)(x > y \rightarrow \varphi).
\end{array}$$

We call these quantifiers **restricted**. Evidently, we can replace every unrestricted quantifier $(\forall x)\varphi$ by the conjunctions of restricted quatifiers if φ contains a free variable different from x.

Lemma 6.2.7 *For every* MSO*–formula* φ *with a least one free object variable there is an* MSO*–formula* φ^g *with restricted quantifiers such that in all Z–structures* $\mathfrak{L}$*:* $\mathfrak{L} \vDash \varphi \leftrightarrow \varphi^g$.

We define functions f, f^+, g and g^+ on unary predicates as follows.

$$\begin{array}{lcl}(f(\varphi))(x) & := & (\exists y \prec x)\varphi(y), \\ (g(\varphi))(x) & := & (\exists y \succ x)\varphi(y), \\ (f^+(\varphi))(x) & := & (\exists y < x)\varphi(y), \\ (g^+(\varphi))(x) & := & (\exists y > x)\varphi(y).\end{array}$$

A somehwat more abstract approach is provided by the notion of a *universal modality.*

Definition 6.2.8 *Let* M *be a set of modailties and* $u \in M$*. Further, let* L *be a* $\mathsf{QML}(M)$*–modal logic.* u *is called a* ***universal modality of*** L *if the following formulae are contained in* L*:*

1. $[u]p \to p$, $[u]p \to [u][u]p$, $p \to [u]\langle u\rangle p$.
2. $[u]p \to [m]p$, *for all* $m \in M$.

Proposition 6.2.9 *Let* L *be a* $\mathsf{QML}(M)$*–logic,* $u \in M$ *a universal modality and further* $\mathfrak{F} = \langle F, R\rangle$ *a connected Kripke–structure with* $\mathfrak{F} \vDash L$*. Then* $R(u) = F \times F$*.*

The proof is again an exercise. The logic of Z–structures does not have a universal modality as such, but it can be defined. Namely, set

$$[u]\varphi := \varphi \wedge [+]\varphi \wedge [-]\varphi\ .$$

This satisfies the requirements above.

The obvious mismatch between MSO and QML is that the former allows for several object variables to occur freely, while the latter only has one free object variable (the world variable), which is left implicit. On condition that φ has only one free object variable, we can actually massage it into a form suitable for QML. Let P_x be a predicate variable which does not occur in φ. Define $\{P_x/x\}\varphi$ inductively as follows.

$$\begin{array}{lcll}
\{P_x/x\}(x \doteq y) & := & P_x(y), & \\
\{P_x/x\}(y \doteq x) & := & P_x(y), & \\
\{P_x/x\}(v \doteq w) & := & v \doteq w, & \text{if } x \notin \{v, w\}, \\
\{P_x/x\}(x \prec y) & := & (g(P_x))(y), & \\
\{P_x/x\}(y \prec x) & := & (f(P_x))(y), & \\
\{P_x/x\}(v \prec w) & := & v \prec w, & \text{if } x \notin \{v, w\}, \\
\{P_x/x\}(\underline{a}(x)) & := & \underline{a}(x), & \\
\{P_x/x\}(\neg\varphi) & := & \neg\{P_x/x\}\varphi, & \\
\{P_x/x\}(\varphi_1 \wedge \varphi_2) & := & \{P_x/x\}\varphi_1 \wedge \{P_x/x\}\varphi_2, & \\
\{P_x/x\}(\varphi_1 \vee \varphi_2) & := & \{P_x/x\}\varphi_1 \vee \{P_x/x\}\varphi_2, & \\
\{P_x/x\}((\exists y)\varphi) & := & (\exists y)\{P_x/x\}\varphi, & \text{if } y \neq x, \\
\{P_x/x\}((\exists x)\varphi) & := & (\exists x)\varphi, & \\
\{P_x/x\}((\forall P)\varphi) & := & (\forall P)(\{P_x/x\}\varphi), & \\
\{P_x/x\}((\exists P)\varphi) & := & (\exists P)(\{P_x/x\}\varphi). &
\end{array}$$

Let $\gamma(P_x) = \{\beta(x)\}$. Then

$$\langle \mathfrak{M}, \gamma, \beta \rangle \vDash \varphi \quad \Leftrightarrow \quad \langle \mathfrak{M}, \gamma, \beta \rangle \vDash \{P_x/x\}\varphi$$

Lemma 6.2.10 *Let*

$$\nu(P) := (\exists x)(P(x)) \wedge (\forall x)(P(x) \rightarrow \neg(f^+(P))(x)) \wedge (\forall x)(P(x) \rightarrow \neg(g^+(P))(x)) \ .$$

Then $\langle \mathfrak{M}, \gamma, \beta \rangle \vDash \nu(P)$ *if and only if* $\gamma(P) = \{x\}$ *for some* $x \in M$.

This is easy to show. Notice that $\nu(P)$ has no free occurrences of x. This is crucial, since it allows to directly translate $\nu(P)$ into a QML–formula.

Now we define an embedding of MSO into QML. Let $h : P \rightarrow PV$ be a bijection from the predicate variables of MSO to the propositional variables of QML.

$$\begin{array}{lcllcl}
(\underline{a}(x))^\diamond & := & Q^a, & (\neg\varphi)^\diamond & := & \neg\varphi^\diamond, \\
(P(y))^\diamond & := & h(P), & (\varphi_1 \wedge \varphi_2)^\diamond & := & \varphi_1^\diamond \wedge \varphi_2^\diamond, \\
(f(\varphi))^\diamond & := & \langle\succ\rangle\varphi^\diamond, & (\varphi_1 \vee \varphi_2)^\diamond & := & \varphi_1^\diamond \vee \varphi_2^\diamond, \\
(g(\varphi))^\diamond & := & \langle\prec\rangle\varphi^\diamond, & ((\exists P)\varphi)^\diamond & := & (\exists P)\varphi^\diamond, \\
(f^+(\varphi))^\diamond & := & \langle-\rangle\varphi^\diamond, & ((\forall P)\varphi)^\diamond & := & (\forall P)\varphi^\diamond, \\
(g^+(\varphi))^\diamond & := & \langle+\rangle\varphi^\diamond. & & &
\end{array}$$

For the first order quantifiers, we do the following.

$$\begin{array}{rl}
((\forall x)(\varphi(x)))^\diamond := & (\forall p_x)(((\langle u\rangle p_x \wedge (\forall p'_x)([u](p'_x \to p_x) \to ([u]\neg p_x. \vee .[u](p'_x \leftrightarrow p_x)))) \\
& \quad \to \{p_x/p\}\varphi^\diamond) \\
((\exists x)(\varphi(x)))^\diamond := & (\forall p_x)(((\langle u\rangle p_x \wedge (\forall p'_x)([u](p'_x \to p_x) \to ([u]\neg p_x. \vee .[u](p'_x \leftrightarrow p_x)))) \\
& \quad \wedge\{p_x/x\}\varphi^\diamond)
\end{array}$$

The correctness of this translation follows from the fact that the formula

$$\langle u\rangle p_x \wedge (\forall p'_x)([u][p'_x \to p_x) \to ([u]\neg p_x) \vee [u](p'_x \leftrightarrow p_x)$$

is satisfied in a Kripke–structure under a valuation β if and only if $\beta(p_x) = \{v\}$ for some $v \in F$.

Theorem 6.2.11 *Let φ be an* MSO*–formula with at most one free variable, the object variable x_0. Then there exists a* QML*–formula φ^M such that for all Z–structures $\mathfrak{L}$:*

$$\langle \mathfrak{L}, \beta\rangle \vDash \varphi(x_0) \text{ iff } \langle M(\mathfrak{Z}), \beta(x_0)\rangle \vDash \varphi(x_0)^M$$

Corollary 6.2.12 *Modulo the identification $\mathfrak{L} \mapsto M(\mathfrak{L})$* MSO *and* QML *define the same model classes of connected nonempty and finite Z–structures. Further: $\mathcal{K}$ is a finitely axiomatizable* MSO*–class of Z–structures if and only if $M(\mathcal{K})$ is a finitely axiomatizable* QML*–class of MZ–structures.*

This now shows that it would suffice if we could prove of finitely axiomatizable QML–classes of MZ–structures that they define regular languages. This we shall do now. Notice that for the proof we only have to look at grammars which have rules of the form $X \to a \mid aY$ and no rules of the form $X \to \varepsilon$. Furthermore instead of regular grammars we can work with grammars*, where we have a set of start symbols in place of a single symbol.

Let $G = \langle \Sigma, N, A, R\rangle$ be a regular grammar* and $\vec{x}$ a string. For a derivation of $\vec{x}$ we define a Z–structure over $A \times N$ (!) as follows. We consider the grammar* $G^{\times} := \langle \Sigma, N, A \times N, R^{\times}\rangle$ which consists of the following rules.

$$R^{\times} := \{X \to \langle a, X\rangle\ Y : X \to aY \in R\}$$

The map $h : A \times N \to A : \langle a, X\rangle \mapsto a$ defines a homomorphism from $(A \times M)^*$ to A^*, which we likewise denote by h. It also gives us a map from Z–structures over $A \times N$ to Z–structures over A. Every G–derivation of $\vec{x}$ uniquely defines a $G^{\times}$–derivation of a string $\vec{x}^{\times}$ with $h(\vec{x}^{\times}) = \vec{x}$ and this in turn defines a Z–structure

$$\mathfrak{M} = \langle L, \prec, \{Q^{\mathfrak{M}}_{\langle a,X\rangle} : \langle a, X\rangle \in A \times N\}\rangle\ .$$

To this model we defin e a model over the alphabet $A \cup N$ as follows.

$$\langle L, \prec^{\mathfrak{L}}, \{Q^{\mathfrak{L}}_a : a \in A\}, \{Q^{\mathfrak{L}}_X : X \in N\}\rangle$$

Here $w \in Q^{\mathfrak{L}}_a$ if and only if $w \in Q^{\mathfrak{M}}_{\langle a,X\rangle}$ for some X and $w \in Q^{\mathfrak{L}}_X$ if and only if $w \in Q^{\mathfrak{M}}_{\langle a,X\rangle}$ for some $a \in A$. We denote this model likewise by $\vec{x}^{\times}$.

Definition 6.2.13 *Let $G = \langle \Sigma, N, A, R\rangle$ be a regular grammar* and $\varphi \in$ QML a constant formula (with constants in A). We say, G is* ***faithful to*** *φ if there is a subset $H \subseteq N$ such that for every string $\vec{x}$, every $\vec{x}^{\times}$ and every w: $\langle \vec{x}^{\times}, w\rangle \vDash \varphi$ if and only if there exists $X \in H$ with $w \in Q_X$. We say, H* ***codes*** *φ* ***with respect to*** *G.*

The intention of the definition is as follows. Given a set H and a formula φ, H codes φ with respect to G if in every derivation of a string $\vec{x}$ φ is true exactly where the nonterminal H occurs. The reader may convince himself of the following facts.

Proposition 6.2.14 *Let G be a regular grammar* and let H code φ and K code χ with respect to G. Then the following holds.*

- ▷ *$N - H$ codes $\neg\varphi$ with respect to G.*
- ▷ *$H \cap K$ codes $\varphi \wedge \chi$ with respect to G.*
- ▷ *$H \cup K$ codes $\varphi \vee \chi$ with respect to G.*

We shall inductively show that every QML–formula can be coded in a regular grammar* on condition that one suitably extends the original grammar.

Definition 6.2.15 *Let $G_1 = \langle \Sigma_1, N_1, A, R_1 \rangle$ und $G_2 = \langle \Sigma_2, N_2, A, R_2 \rangle$ be regular grammars*. The let*

$$G_1 \times G_2 := \langle \Sigma_1 \times \Sigma_2, N_1 \times N_2, A, R_1 \times R_2 \rangle$$

where

$$\begin{aligned} R_1 \times R_2 := \;& \{\langle X_1, X_2 \rangle \to a \quad \langle Y_1, Y_2 \rangle : X_1 \to aY_1 \in R_1, X_2 \to aY_2 \in R_2\} \\ & \cup \{\langle X_1, X_2 \rangle \to a : X_1 \to a \in R_1, X_2 \to a \in R_2\} \end{aligned}$$

*This we call the **product** of the grammars* G_1 and G_2.*

We have

Proposition 6.2.16 *Let G_1 and G_2 be grammars* over A. Then $L(G_1 \times G_2) = L(G_1) \cap L(G_2)$.*

The following theorem is not hard to show and therefore left as an exercise.

Lemma 6.2.17 *Let φ be coded in G_2 by H. Then φ is coded in $G_1 \times G_2$ by $N_1 \times H$ and in $G_2 \times G_1$ by $H \times N_2$.*

Definition 6.2.18 *Let φ be a QML–formula. A **code** for φ is a pair $\langle G, H \rangle$ where $L(G) = A^*$ and H codes φ with respect to G. φ is called **codable** if it has a code.*

Assume that φ has a $\langle G, H\rangle$ and let G' be given. Then $L(G' \times G) = L(G')$ and φ is coded in $G' \times G$ by $N' \times H$. Therefore, it suffices to name just one code for every formula. However, we shall also draw a conclusion which makes life simpler for us.

Lemma 6.2.19 *Let Δ be a finite set of codable formulae. Then there exists a grammar* G and sets H_φ, $\varphi \in \Delta$, such that $\langle G, H_\varphi\rangle$ is a code of φ.*

Proof. Let $\Delta := \{\delta_i : i < n\}$ and let $\langle G_i, M_i\rangle$ be a code of δ_i, $i < n$. Put $G := \mathsf{X}_{i<n} G_i$ and $H_i := \mathsf{X}_{j<i} N_i \times H_i \times \mathsf{X}_{i<j<n} N_j$. Iterated application of Lemma 6.2.17 yields the claim. $\Box$

Theorem 6.2.20 *Every constant* QML*–formula is codable.*

Proof. The proof is by induction over the structure of the formula. We begin with the code of $\underline{a}$, $a \in A$. Define the following grammar* G_a. We have $N := \{X, Y\}$. $\Sigma := \{X, Y\}$. The rules are

$$X \to a \mid aX \mid aY, \quad Y \to b \mid bX \mid bY \ ,$$

where b ranges over all elements from $A - \{a\}$. The code is $\langle G_a, \{X\}\rangle$ as one easily checks. The inductive steps for $\neg$, $\wedge$, $\vee$ and $\to$ are covered by Proposition 6.2.14. Now for the case $\varphi = \langle\prec\rangle\eta$. We assume that η is codable and that $C_\eta = \langle G_\eta, H_\eta\rangle$ is a code. Now we define G_φ. Let $N_\varphi := N_\eta \times \{0, 1\}$ and $\Sigma_\varphi := \Sigma_\eta \times \{0, 1\}$. Finally, let the rules be of the form

$$\langle X, 1\rangle \to a\ \langle Y, 0\rangle, \quad \langle X, 1\rangle \to a\ \langle Y, 1\rangle \ ,$$

where $X \to aY \in R_\eta$ and $Y \in H_\eta$ and of the form

$$\langle X, 0\rangle \to a\ \langle Y, 0\rangle, \quad \langle X, 0\rangle \to a\ \langle Y, 1\rangle \ ,$$

where $X \to aY \in R_\eta$ and $Y \notin H_\eta$. Further, we have rules of the form

$$\langle X, 0\rangle \to a, \quad \langle X, 1\rangle \to a$$

for all $X \to a \in R_\eta$. One easily checks that for every rule $X \to aY$ or $X \to a$ there is a rule G_φ. The code of φ is now $\langle G_\varphi, N_\eta \times \{1\}\rangle$. Now to the case $\varphi = \langle\succ\rangle\eta$. Again we assume that η is codable and that the code is $C_\eta = \langle G_\eta, H_\eta\rangle$. Now we define G_φ. Let $N_\varphi := N_\eta \times \{0, 1\}$, $\Sigma_\varphi := \Sigma_\eta \times \{0\}$. Finally, let R_φ be the set of rules of the form

$$\langle X, 0\rangle \to a\ \langle Y, 1\rangle, \quad \langle X, 1\rangle \to a\ \langle Y, 1\rangle\ ,$$

where $X \to aY \in R_\eta$ and $X \in H_\eta$ and of the form

$$\langle X, 0\rangle \to a\ \langle Y, 0\rangle, \quad \langle X, 1\rangle \to a\ \langle Y, 1\rangle\ ,$$

where $X \to aY \in R_\eta$ and $X \notin H_\eta$; and finally for every rule $X \to a$ we take the rule

$$\langle X, 0\rangle \to a, \quad \langle X, 1\rangle \to a$$

on board. The code of φ is now $\langle G_\varphi, N_\eta \times \{1\}\rangle$. Now we look at $\varphi = \langle +\rangle\eta$. Again we put $N_\varphi := N_\eta \times \{0, 1\}$ as well as $\Sigma_\varphi := \Sigma_\eta \times \{0, 1\}$ the set of start symbols. The rules have the form

$$\langle X, 0\rangle \to a\ \langle Y, 0\rangle\ ,$$

if $X \to aY \in R_\eta$. Further they have the form

$$\langle X, 1\rangle \to a\ \langle Y, 1\rangle\ ,$$

if $X \to aY \in R_\eta$ and $Y \notin H_\eta$. Moreover, we take the rules

$$\langle X, 1\rangle \to a\ \langle Y, 0\rangle\ ,$$

if $X \to aY \in R_\eta$ and $Y \in H_\eta$, as well as finally all rules

$$\langle X, 0\rangle \to a\ ,$$

where $X \to a \in R_\eta$. The code of φ is then $\langle G_\varphi, N_\eta \times \{1\}\rangle$. Now we look at $\varphi = \langle -\rangle\eta$. Let $N_\varphi := N_\eta \times \{0, 1\}$, as well as $\Sigma_\varphi := \Sigma_\eta \times \{0\}$ the set of start symbols. The rules are of the form

$$\langle X, 1\rangle \to a\ \langle Y, 1\rangle\ ,$$

if $X \to aY \in R_\eta$. Further there are rules of the form

$$\langle X, 0\rangle \to a\ \langle Y, 1\rangle\ ,$$

if $X \to aY \in R_\eta$ and $X \in H_\eta$. Moreover, we take the rules

$$\langle X, 0\rangle \to a\ \langle Y, 0\rangle\ ,$$

if $X \to aY \in R_\eta$ and $Y \notin H_\eta$, and, finally, all rules of the form

$$\langle X, i\rangle \to a\ ,$$

where $X \to a \in R_\eta$ and $i \in \{0, 1\}$. The code of φ is then $\langle G_\varphi, N_\eta \times \{1\}\rangle$. The biggest effort goes into the last case, $\varphi = (\forall p_i)\eta$. To start, we introduce a new alphabet, namely $A \times \{0, 1\}$, and a new constant, $\underline{c}$. Let $\underline{a} := \overline{\langle a, 0\rangle} \vee \overline{\langle a, 1\rangle}$. Further let $\underline{c} \leftrightarrow \bigvee_{a \in A} \langle a, 1\rangle$. Then $\underline{\langle a, 1\rangle} \leftrightarrow \underline{a} \wedge \underline{c}$ and $\overline{\langle a, 0\rangle} \leftrightarrow \underline{a} \wedge \neg\underline{c}$. Then let $\eta' := \eta\overline{[\underline{c}/p_i]}$. We can apply the inductive hypothesis to this formula. Let Δ be the set of subformulae of η'. For an arbitrary subset $\Sigma \subseteq \Delta$ let

$$L_\Sigma := \bigwedge_{\delta \in \Sigma} \delta \wedge \bigwedge_{\delta \notin \Sigma} \neg\delta\ .$$

First of all we can find a grammar* G and sets H_δ, $\delta \in \Delta$, such that $\langle G, H_\delta\rangle$ codes δ. Hence there exist H_Σ such that $\langle G, H_\Sigma\rangle$ codes L_Σ for every $\Sigma \subseteq \Delta$. Now we first form the grammar* G^1 with the nonterminals $N \times \wp(\Delta)$ and the alphabet $A \times \{0, 1\}$. The set of rules is the set of all

$$\langle X, \Sigma\rangle \to \langle a, i\rangle \quad \langle X', \Sigma'\rangle$$

where $X \to aX' \in R$, $X \in H_\Sigma$ and $X' \in H_{\Sigma'}$; further all rules of the form

$$\langle X, \Sigma\rangle \to \langle a, i\rangle\ ,$$

where $X \to a \in R$ and $X \in H_\Sigma$. Put $H^1_\Sigma := H_\Sigma \times \{\Sigma\}$. Again one easily sees that $\langle G^1, H^1_\Sigma\rangle$ is a code for Σ for every $\Sigma \subseteq \Delta$. We now step over to the grammar* G^2 with $N^2 := N \times \{0,1\} \times \wp(\Delta)$ and $A^2 := A$ as well as all rules

$$\langle X, i, \Sigma\rangle \to a \quad \langle X', i', \Sigma'\rangle$$

where $\langle X, \Sigma\rangle \to \langle a, i\rangle \quad \langle X', \Sigma'\rangle \in R^1$ and

$$\langle X, i, \Sigma\rangle \to a$$

where $\langle X, \Sigma\rangle \to \langle a, i\rangle \in R^1$. Finally, we define the following grammar*. $N^3 := N \times \wp(\wp(\Delta))$, $A^3 := A$. Further, let

$$\langle X, \mathbb{A}\rangle \to a \quad \langle Y, \mathbb{B}\rangle$$

be a rule if and only if $\mathbb{B}$ is the set of all Σ' for which $\Sigma \in \mathbb{A}$ and there are $i, i' \in \{0,1\}$ such that

$$\langle X, i, \Sigma\rangle \to a \quad \langle Y, i', \Sigma'\rangle \in R^2 \ .$$

Likewise

$$\langle X, \mathbb{A}\rangle \to a \in R^3$$

if and only if there is a $\Sigma \in \mathbb{A}$ and some $i \in \{0,1\}$ with

$$\langle X, i, \Sigma\rangle \to a \in R^2 \ .$$

Put $H_\varphi := \{\Sigma : \eta' \in \Sigma\}$. We claim: $\langle G^3, H_\varphi\rangle$ is a code for φ. For a proof let $\vec{x}$ be a string and let a G^3–derivation of $\vec{x}$ be given. We construct a G^1–derivation. Let $\vec{x} = \prod_{i<n} x_i$. By assumption we have a derivation

$$\langle X_i, \mathbb{A}_i\rangle \to x_i \quad \langle X_{i+1}, \mathbb{A}_{i+1}\rangle$$

for $i < n - 1$ and

$$\langle X_{n-1}, \mathbb{A}_{n-1}\rangle \to x_{n-1} \ .$$

By construction there exists a j_{n-1} and a $\Sigma_{n-1} \in \mathbb{A}_{n-1}$ such that

$$\langle X_{n-1}, j_{n-1}, \Sigma_{n-1}\rangle \to a \in R^2 \ .$$

Descending we get for every $i < n - 1$ a j_i and a Σ_i with

$$\langle X_i, j_i, \Sigma_i\rangle \to a \quad \langle X_{i+1}, j_{i+1}, \Sigma_{i+1}\rangle \in R^2 \ .$$

We therefore have a G^2–derivation of $\vec{x}$. From this we immediately get a G^1–derivation. It is over the alphabet $A \times \{0, 1\}$. By assumption η' is coded G^1 by $H_{\eta'}$. Then η' holds in all nodes i with $X_i \in H_{\eta'}$. This is the set of all i with $X_i \in H_\Sigma$ for some $\Sigma \subseteq \Delta$ with $\eta' \subseteq \Delta$. This is exactly the set of all i with $\langle X_i, \mathbb{A}_i\rangle \in H_\varphi$. Hence we have $\langle X_i, \mathbb{A}_i\rangle \in H_\varphi$ if and only if the Z–structure of $\vec{x}$ satisfies φ in the given G^3–derivation at i. This however had to be shown. Likewise from a G^1–derivation of a string a G^3–derivation can be constructed, as is easily seen. □

Now we are almost done. As our last task we have to show that from the fact that a formula is codable we also get a grammar which only generates strings that satisfy this formula. So let Φ be a finite set of MSO–formulae. We may assume that they are sentences (if not, we quantify over the free variables with a universal quantifier). By Corollary 6.2.12 we can assume that in place of MSO–sentences we are dealing with QML–formulae. Further, a finite conjunction QML–formulae is again a QML–formula so that we can reduce to treating the case where Φ consists in a single QML–formula φ. By Theorem 6.2.20 there is ia code $\langle G, H\rangle$ for φ. Let $G = \langle \Sigma, N, A, R\rangle$. Put $G^\varphi := \langle \Sigma \cap H, N \cap H, A, R_H\rangle$, where

$$R_H := \{X \to aY \in R : X, Y \in H\} \cup \{X \to a \in R : X \in H\} \ .$$

Now there exists a G^φ–derivation of $\vec{x}$ if and only if $\vec{x}^\times \vDash \varphi$.

Exercise 209. Show that every (!) language is an intersection of regular languages. This intersection may actually be empty. (This means that we cannot omit the condition of finite axiomatizability in the theorem oy Büchi.)

Exercise 210. Let Φ be a finite MSO–theory, R the regular language which belongs to Φ. R is recognizable in $O(n)$–time using a finite state automaton. Give upper bounds for the number of states of a minimal automaton recognizing R. Use the proof of codability. Are the derived bounds optimal?

Exercise 211. (Continuing the previous exercise.) Give an explicit constant c_Φ such that a single tape Turing machine recognizes R in $\leq c_\Phi \cdot n$ time. How does c_Φ depend on Φ?

Exercise 212. An MSO–sentence is said to be in Σ^1_1 if it has the form

$$(\exists P_0)(\exists P_1)\ldots(\exists P_{n-1})\varphi(P_0,\ldots,P_{n-1})$$

where φ does not contain second order quantifiers. φ is said to be in Π^1_1 if it has the form $(\forall P_0)(\forall P_1)\ldots(\forall P_{n-1})\varphi(P_0,\ldots,P_{n-1})$ where φ does not contain second order quantifiers. Show the following: *Every class $\mathcal{K}$ of Z–structures is axiomatizable by a set of Σ^1_1–sentences. If $\mathcal{K}$ is finitely axiomatizable then it is finitely axiomatizable by finitely many Σ^1_1–sentences.*

6.3 Phonemicization and Two Level Phonology

In this section we shall deal with syllable structure and phonological rules. This is partly in order to exhibit some theories of phonology, and partly in order to fill a gap in our definitions of compositionality so far. Recall that we have assumed sign grammars to be completely additive: there is no possibility to remove something from an exponent that has been put there before. For example, suppose that the the root of the German noun `Vater` also is `Vater`. Since the plural is `Väter`, it cannot be formed because it will require transforming the letter `a` into `ä`. Likewise, we cannot even assume that the root contains a more abstract sound in place of `a` or `ä`, say, `A`, because this again would require rewriting that symbol. The only alternative is to postulate for German Umlaut vowels a decomposition into vowel plus a phoneme `U`, such that `aU` is realized as surface `ä`, `aU` as `ö` and `uU` as `ü`. The root then does contain the vowel `a`, and it has the form `Va` $\otimes$ `ter`. Umlauting this root consists in adding U and then 'gluing' the two pieces together. (Indeed, historically umlaut has developed from being additive to being fusional.) Final devoicing could be solved similarly by positing a decomposition of voiced consonants into voiceless consonant plus an abstract voice element. All these solutions, however, posit two levels of phonology: a surface phonology and a deep phonology. At the deep level, signs are again additive. This allows us to say that languages are compositional from the deep phonological level onwards.

In (1968), Chomsky and Halle have proposed a model of phonology that transforms deep structures into surface structures using context sensitive rewrite rules. We may illustrate these rules with German final devoicing. The rule says, roughly, that syllable final consonants (those following the vowel) are voiceless in German. However, as we have noted earlier (in Section 1.3), there is evidence to assume that some consonants are voiced and only become voiceless exactly when they end up in syllable final position. Hence Chomsky and Halle do not view this as a constraint on the structure of the syllable. They take it to be the effect of a rule that devoices consonants. Write + for the syllable boundary. Sidestepping a few difficulties, we may write the rule of devoicing as follows.

$$\mathtt{C}[+voiced] \Longrightarrow \mathtt{C}[-voiced]/\_\_[-voiced] \ or \ \_\_+$$

This says that a consonant preceding a voiceless sound or a syllable boundary becomes voiceless. Using such rules, Chomsky and Halle have formulated a theory of the sound structure of English, using a Type 1 grammar. It has been observed, however, in (Kaplan and Kay, 1994) and (Koskenniemi, 1983) that for all that language really needs the relation between deep level and surface level is a regular relation and can be effected by a finite state transducers. Before we go into the details, we shall explain something about the general abstraction process in structural linguistics, exemplified here with phonemes, and on syllable structure.

Phonetics is the study of sounds, whereas phonology is the study of the sound systems of the languages. We may simply define a phoneme as a set of sounds. Different languages group different sounds into different phonemes, so that the phonemes of languages are typically not comparable. The grouping into phonemes is far from easy. A good exposition of the method can be found in (Harris, 1963). We shall look at the process of phonemicization in some detail. Let us assume for simplicity that words or texts are realized as sequences of discrete entities called *sounds*. This is not an innocent assumption: it is not clear, often enough, whether the sequence [t] plus [ʃ], resulting in an affricate [tʃ], is to be seen as one or as two sounds. Now, denote the set of sounds by Σ. A word is not a single sequence of sounds, but rather a set of such sequences.

Definition 6.3.1 *L is a **language* over** Σ if L is a subset of $\wp(\Sigma^*)$ such that $\varnothing \notin \Sigma$ and if $W, W' \in L$ and $W \cap W' \neq \varnothing$ then $W = W'$. We call the members of L **words**. $\vec{x} \in W$ is called a **realization** of W. For two sequences $\vec{x}$ and $\vec{y}$ we write $\vec{x} \sim_L \vec{y}$ if they belong to (or realize) the same word.*

One of the aims of phonology is to simplify the alphabet in such a way that words are realized by as few as possible sequences. We proceed by choosing a new alphabet, P and a mapping $\pi : \Sigma \to P$. The map π induces a partition on Σ. If $\pi(s) = \pi(s')$ we say that s and s' are **allophones**. π induces a mapping of L onto a subset of $\wp(P^*)$ in the following way. For a word W we write $\overline{\pi}[W] := \{\overline{\pi}(\vec{x}) : \vec{x} \in W\}$. Finally, $\pi^*(L) := \{\overline{\pi}[W] : W \in L\}$. The map π must have the following property: if $\vec{x}$ and $\vec{y}$ belong to different words, then $\overline{\pi}(\vec{x}) \neq \overline{\pi}(\vec{y})$. This gives rise to the following definition.

Definition 6.3.2 *Let $\pi : P \to \Sigma$ be a map and $L \subseteq \wp(\Sigma^*)$ be a language*. π is called **discriminating for** L if whenever $W, W' \in L$ are distinct then $\overline{\pi}[W] \cap \overline{\pi}[W'] = \varnothing$.*

Lemma 6.3.3 *Let $L \subseteq \wp(\Sigma)$ be a language and $\pi : \Sigma \to P$. If π is discriminating for L, $\pi^*(L)$ is a language over P.*

Definition 6.3.4 *A* ***phonemicization*** *of L is a discriminating map $v : A \to B$ such that for every discriminating $w : A \to C$ we have $|C| \geq |B|$. We call the members of B* ***phonemes****.*

As it turns out, the phonemes are typically not mere sets of sounds. As such, they would otherwise be infinite. However, no speaker of a language has access to infinitely many sounds at any given moment. Rather, phonemes typically are defined by means of articulatory features, which tell us (in an effective way) what sound is associated with what phoneme. For example, English [p] is a sound that is voiceless (this means that the chords do not vibrate while the sound is being pronounced), it is an obstruent (it obstructs the air flow), it is a bilabial (it is pronounced by putting the lips together) and so on. The analysis of this sort ends in the establishment of an alphabet P of abstract sounds classes, defined by means of some articulatory features. These can be modeled in the logical language by means of *constants*. For example, the feature *voiced* corresponds to a constant which we call by the same name. Obviously, $\neg$*voiced* is the same as being unvoiced.

The feature are often interdependent. For example, vowels are always voiced and continuants. In English and German voiceless plosives are aspirated, while in French this is not the case; so [t] is pronounced with a subsequent [h]. (In older German books one therefore wrote `Theil` (‘part’) in place of the modern `Teil`. The aspiration is lacking when the sound is preceded by [ʃ]. In [ˈʃtʊmpf] the aspiration is lacking. In German, vowels are not simply long and short. Also the vowel quality changes with the length. Long vowels are tense. The letter `i` is pronounced [ɪ] when it is short and [i] when it is long. (For example, `Sinn` (‘sense’) [ˈzɪn] as opposed to `Tief` (‘deep’) [ˈti:f].) Likewise for the other vowels. Tabel 6.1 shows the pronunciation of the long and short vowels, drawn from (IPA, 1999), Page 87 (written by Klaus Kohler). Only [a] and [a:] are pronounced at the same place. It is therefore not easy to say which feature is distinctive: is length distinctive in German for vowels, or is it rather the tension? This is interesting

Table 6.1: Long and short vowels of German

long	short	long	short
i:	ɪ	y:	ʏ
a:	a	e:	ə
o:	ɔ	ø:	œ
u:	ʊ, ɷ	ɛ:	ɛ

in particular when speakers learn a new language, because they might be forced to keep two parameters apart that used to be cogradient. For example, in Finnish vowels are purely distinct in length, there is no cooccurring distinction in tension. If so, tension cannot be used to differentiate a long vowel from a short one. This is a potential source of difficulty for Germans if they want to learn Finnish.

If L is a language* in which every word has exactly one member, L is uniquely defined by the language $L^\diamond := \{\vec{x} : \{\vec{x}\} \in L\}$. Let us assume after suitable reductions that we have such a language*; then we may return to studying languages in the customary sense. It might be thought that languages do not possess nontrivial phonemicization maps. This is, however, not so. For example, English has two different sounds, [p] and [p^h]. The first occurs after [s], while the second appears for example word initially before a vowel. It turns out that in English [p] and [p^h] are not two but one phoneme. To see why, we offer first a combinatorial and then a logical analysis. Recall the definition of a context set.

$$\mathbb{C}_L(a) := \{\langle \vec{x}, \vec{y} \rangle : \vec{x} \cdot a \cdot \vec{y} \in L\} .$$

a and a' are said to be in **complementary distribution** if $\mathbb{C}_L(a) \cap \mathbb{C}_L(a') = \varnothing$. An example is the abovementioned [p] and [p^h]. Another example is [x] versus [χ] in German. Both are written `ch`. However, `ch` is pronounced [x] if occurring after [a], [o] and [u], while it is pronounced [ç] if occurring after other vowels and [r],

[n] or [l]. Examples are `Licht` [lıçt], `Nacht` [naxt], `echt` [eçt] and `acht` [axt]. (If you do not know German, here is a short description of the sounds. [x] is pronounced at the same place as [k] in English, but it is a fricative. [ç] is pronounced at the same place as `y` in English `yacht`, however the tongue is a little higher, that is, closer to the palatum and also the air pressure is somewhat higher, making it sound harder.) Now, from Definition 6.3.4 we extract the following.

Definition 6.3.5 *Let A be an alphabet and L a language over A. $\pi : A \to B$ is a* ***pre–phonemicization*** *if $\overline{\pi}$ is injective on L. $\pi : A \twoheadrightarrow B$ is a* ***phonemicization*** *if for all pre–phonemicizations $\pi' : A \twoheadrightarrow C$, $|C| \geq |B|$.*

The map sending [x] and [ç] to the same sound is a pre–phonemicization in German. However, notice the following. In the language $L_0 := \{\mathtt{aa}, \mathtt{bb}\}$, `a` and `b` are in complementary distribution. Nevertheless, the map sending both to the same element is not injective. So, complementary distribution is not enough to make two sounds belong to the same phoneme. We shall see below what is. Second, let $L_1 := \{\mathtt{ac}, \mathtt{bd}\}$. We may either send `a` and `b` to `e` and obtain the language $M_0 := \{\mathtt{ec}, \mathtt{ed}\}$, or we may send `c` and `d` to `f` and obtain the language $M_1 := \{\mathtt{af}, \mathtt{bf}\}$. Both maps are phonemicizations, as is easily checked. So, the result is not unique. In order to analyse the situation we have to present a few definitions. The general idea is this. Suppose that A is not minimal for L in the sense that it possesses a noninjective phonemicization. Then there is a pre–phonemicization that conflates exactly two symbols into one. The image M of this map is a regular language again. Now, given the latter we can actually recover for each member of M its preimage under this conflation. What we shall show now is that moreover if L is regular *there is an explicit procedure telling us what the preimage is.* This will be cast in rather abstract terms. We shall define here a modal language that is somewhat different from QML, namely PDL with converse. In addition to PDL, it also has a program constructor $^{\smile}$. $\alpha^{\smile}$ denotes the converse of α. Hence, in a Kripke–frame $\overline{R}(\alpha^\smile) = \overline{R}(\alpha)^\smile$. The axiomatization consists in the axioms for PDL together with the axioms $p \to [\alpha]\langle\alpha^\smile\rangle p$, $p \to [\alpha^\smile]\langle\alpha\rangle p$ for every program α. The term *dynamic logic* will henceforth refer to an extension of $\mathsf{PDL}^\smile$ by some axioms. The fragment without * is called **elementary** $\mathsf{PDL}^\smile$, and is denoted by $\mathsf{EPDL}^\smile$. An analog of Büchi's Theorem holds of $\mathsf{PDL}^\smile(\prec)$.

Theorem 6.3.6 *Let A be a finite alphabet. A class of MZ–structures over A is regular if and only if it is axiomatizable over the logic of all MZ–structures by means of constant formulae in* $\mathsf{PDL}^\smile(\prec)$ *(with constants for letters from A).*

Proof. By Kleene's Theorem, a regular language is the extension of a regular term. Such a term can be written down in $\mathsf{PDL}^\smile$ using a constant formula. Conversely, if γ is a constant $\mathsf{PDL}^\smile(\prec)$–formula it can be rewritten into an MSO–formula. $\Box$

The last point perhaps needs reflection. There is a straightforward translation of $\mathsf{PDL}^\smile$ into MSO. We only have to observe that the transitive closure of an MSO–definable relation is again MSO–definable.

$$x\ R^*\ y \quad \Leftrightarrow \quad (\forall X)(X(x)\wedge(\forall z)(\forall z')(X(z)\wedge z\ R\ z'. \to .X(z')) \to X(y))$$

Notice also that we can eliminate $^\smile$ from complex programs using the following identities.

$$\begin{array}{lcl}
\overline{R}((\alpha\cup\beta)^\smile) & = & \overline{R}(\alpha^\smile\cup\beta^\smile), \\
\overline{R}((\alpha;\beta)^\smile) & = & \overline{R}(\beta^\smile;\alpha^\smile), \\
\overline{R}((\alpha^*)^\smile) & = & \overline{R}((\alpha^\smile)^*), \\
\overline{R}((\varphi?)^\smile) & = & \overline{R}(\varphi?).
\end{array}$$

Hence, $\mathsf{PDL}^\smile(\prec)$ can also be seen as an axiomatic extension of $\mathsf{PDL}(\prec;\succ)$ by the axioms $p \to [\prec]\langle\succ\rangle p$, $p \to [\succ]\langle\prec\rangle p$. Now let Θ be a dynamic logic. Recall from Section 4.7 the definition of $\Vdash_\Theta$, the global consequence associated with Θ.

Now, we shall assume that we have a language $\mathsf{PDL}^\smile(\prec;D)$, where D is a set of constants. For simplicity, we shall assume that for each letter $a \in A$ D contains a constant $\underline{a}$. However, there may be additional constants. It is those constants that we shall be investigating here. We shall show (i) that these constants can be eliminated in an explicit way, (ii) that one can always add constants such that A can be be described purely by contact rules.

Definition 6.3.7 *Let Θ be a dynamic logic and φ a formula. We say that $\varphi(q)$* ***globally implicitly defines*** *q* ***in*** *Θ if $\varphi(q); \varphi(q') \Vdash_\Theta q \leftrightarrow q'$.*

Features (or constants, for that matter) that are implicitly defined are called **inessential**. Here the leading idea is that inessential features do not constitute a distinctive phonemic feature, because removing the distinction that this feature induces on the alphabet turns out to induce an injective map. Formally, this is spelled out as follows. Let $A \times \{0, 1\}$ be an alphabet, and assume that the second component indicates the value of the feature c. Let $\pi : A \times \{0, 1\} \to A$ be the projection onto the first factor. Suppose that the language L can be axiomatized by the constant formula $\varphi(\mathsf{c})$. $\varphi(\mathsf{c})$ defines c implicitly if $\overline{\pi} : L' \to L$ is injective. This in turn means that the map π is a pre–phonemicization. For in principle we could do without the feature. Yet, it is not clear that we can simply eliminate it. In $\mathsf{PDL}^{\smile} \oplus \varphi(\mathsf{c})$ we call c **eliminable** if there is a formula χ provably equivalent to $\varphi(\mathsf{c})$ that uses only the constants of φ without c. In the present case, however, an inessential feature is also eliminable. Notice first of all that a regular language over an alphabet B is definable by means a constant formula over the logic of all strings, with constants $\underline{b}$ for every element b of B. By Lemma 6.3.10, it is therefore enough to show the claim for the logic of all strings. Moreover, by a suitable replacement of other variables by new constants we may reduce the problem to the case where p is the only variable occurring in the formula. Now the language L is regular over the alphabet $A \times \{0, 1\}$. Therefore, $\overline{\pi}[L]$ is regular as well. This means that it can be axiomatized using a formula without the constant c. However, this only means that we can make the representation of words more compact. Ideally, we also wish to describe for given $a \in A$, in which context we find $\langle a, 0\rangle$ (an a lacking c) and in which context we find $\langle a, 1\rangle$ (an a having c). This can be done. Let $\mathfrak{A} = \langle A, Q, q_0, F, \delta\rangle$ be a finite state automaton. Then $L_{\mathfrak{A}}(q) := \{\vec{x} : q_0 \xrightarrow{\vec{x}} q\}$ is a regular language (for $L_{\mathfrak{A}}(q) = L(\langle A, Q, q_0, \{q\}, \delta\rangle)$, and the latter is a finite state automaton). Furthermore, $A^* = \bigcup_{q \in Q} L_{\mathfrak{A}}(q)$. If $\mathfrak{A}$ is deterministic, then $L_{\mathfrak{A}}(q) \cap L_{\mathfrak{A}}(q')$ whenever $q \neq q'$. Now, let $\mathfrak{B}$ be a deterministic finite state automaton over $A \times \{0, 1\}$ such that $\vec{x} \in L(\mathfrak{B})$ if and only if $\vec{x} \vDash \varphi(\mathsf{c})$. Suppose we have a constraint χ, where χ is a constant formula.

Definition 6.3.8 *The* ***Fisher–Ladner closure*** *of χ is defined as follows.*

$$\begin{array}{lll}
FL(p_i) & := & \{p_i\}, \\
FL(\gamma) & := & \{\gamma\}, \\
FL(\chi \wedge \chi') & := & \{\chi \wedge \chi'\} \cup FL(\chi) \cup FL(\chi'), \\
FL(\langle \alpha \cup \beta\rangle\chi) & := & \{\langle \alpha \cup \beta\rangle\chi\} \cup FL(\langle\alpha\rangle\chi) \cup FL(\langle\beta\rangle\chi), \\
FL(\langle \alpha; \beta\rangle\chi) & := & \{\langle \alpha; \beta\rangle\chi\} \cup FL(\langle\alpha\rangle\langle\beta\rangle\chi), \\
FL(\langle \alpha^*\rangle\chi) & := & \{\langle \alpha^*\rangle\chi\} \cup FL(\langle\alpha\rangle\langle\alpha^*\rangle\chi) \cup FL(\chi), \\
FL(\langle \varphi?\rangle\chi) & := & \{\langle \varphi?\rangle\chi\} \cup FL(\varphi) \cup FL(\chi), \\
FL(\langle \alpha\rangle\chi) & := & \{\langle \alpha\rangle\chi\} \cup FL(\chi), \qquad \alpha \textit{ basic.}
\end{array}$$

The Fisher–Ladner closure covers only $\mathsf{PDL}(\prec; \succ)$–formulae, but this is enough for our purposes. Now for each formula σ in the Fisher–Ladner closure of χ we introduce a constant $c(\sigma)$. In addition, we add the following axioms.

$$\begin{array}{lll}
c(\neg\sigma) & \leftrightarrow & \neg c(\sigma) \\
c(\sigma \wedge \tau) & \leftrightarrow & c(\sigma) \wedge c(\tau) \\
c(\langle\varphi?\rangle\sigma) & \leftrightarrow & c(\varphi) \wedge c(\sigma) \\
c(\langle\alpha \cup \beta\rangle\sigma) & \leftrightarrow & c(\langle\alpha\rangle\sigma) \vee c(\langle\beta\rangle\sigma) \\
c(\langle\alpha; \beta\rangle\sigma) & \leftrightarrow & c(\langle\alpha\rangle\langle\beta\rangle\sigma) \\
c(\langle\alpha^*\rangle\sigma) & \leftrightarrow & c(\langle\alpha\rangle\langle\alpha^*\rangle\sigma) \\
c(\langle\prec\rangle\sigma) & \leftrightarrow & \langle\prec\rangle c(\sigma) \\
c(\langle\succ\rangle\sigma) & \leftrightarrow & \langle\succ\rangle c(\sigma)
\end{array}$$

We call these formulae **cooccurrence restrictions**. After the introduction of these formulae as axioms, the equivalences $\sigma \leftrightarrow c(\sigma)$ are provable for every $\sigma \in FL(\chi)$. In particular, $\chi \leftrightarrow c(\chi)$ is provable. This means that we can eliminate χ in favour of $c(\chi)$. The formulae that we have just added do not contain any of ?, $\cup$, ;, $\smile$ or *. We only have the most simple axioms, stating that some constant is true before or after another. Now we construct the following automaton. Let ϑ be a subset of $FL(\chi)$. Then put

$$q_\vartheta := \bigwedge_{\gamma\in\vartheta} c(\gamma) \wedge \bigwedge_{\gamma\notin\vartheta} \neg c(\gamma)$$

Now let Q be the set of all consistent q_ϑ. Furthermore, put $q_\vartheta \xrightarrow{a} q_\eta$ if and only if $q_\vartheta \wedge \langle\prec; \underline{a}?\rangle q_\eta$ is consistent. Let $F := \{q_\vartheta : [\prec]\bot \in \vartheta\}$ and $B := \{q_\vartheta : [\succ]\bot \in \vartheta\}$. For every $b \in B$, $\langle A, Q, b, F, \delta\rangle$ is a finite state automaton. Then

$$L := \bigcup_{b\in B} L(\langle A, Q, b, F, \delta\rangle)$$

is a regular language. It immediately follows that the automaton above is well–defined and for every subformula α of χ the set of indices i such that $\langle\vec{x}, i\rangle \vDash \alpha$ is uniquely fixed. Hence, for every $\vec{x}$ there exists exactly one accepting run of the automaton. $\langle\vec{x}, i\rangle \vDash \psi$ if and only if ψ is in the ith state of the accepting run.

We shall apply this to our problem. Let $\varphi(\mathsf{c})$ be an implicit definition of c. Construct the automaton $\mathfrak{A}(\varphi(\mathsf{c}))$ for $\varphi(\mathsf{c})$ as just shown, and lump together all states that do not contain $c(\varphi(\mathsf{c}))$ into a single state q' and put $q' \xrightarrow{a} q'$ for every a. All states different from q' are accepting. This defines the automaton $\mathfrak{B}$. Now let $C := \{q_\vartheta : \mathsf{c} \in \vartheta\}$. The language $\bigcup_{c\in C} L_{\mathfrak{B}}(q)$ is regular, and it possesses a description in terms of the constants $\underline{a}$, $a \in A$, alone.

Definition 6.3.9 *Let Θ be a logic and $\varphi(q)$ a formula. Further, let δ be a formula not containing q. We say that δ* ***globally explicitly defines*** *q* ***in*** *Θ* ***with respect to*** *φ if $\varphi(q) \Vdash_\Theta \delta \leftrightarrow q$.*

Obviously, if δ globally explicitly defines q with respect to $\varphi(q)$ then $\varphi(q)$ globally implicitly defines q. On the other hand, if $\varphi(q)$ globally implicitly defines q then it is not necessarily the case that there is an explicit definition for it. It very much depends on the logic in addition to the formula whether there is. A logic is said to have the **global Beth–property** if for any global implicit definition there is a global explicit definition. Now suppose that we have a formula φ and that it implicitly defines q. Suppose further that δ is an explicit definition. Then the following is valid.

$$\Vdash_\Theta \varphi(q) \leftrightarrow \varphi(\delta)$$

The logic $\Theta \oplus \varphi$ defined by adding the formula φ as an axiom to Θ can therefore equally well be axiomatized by $\Theta \oplus \varphi(\delta)$. The following is relatively easy to show.

Lemma 6.3.10 *Let Θ be a modal logic, and γ a constant formula. Suppose that Θ has the global Beth–property. Then $\Theta \oplus \gamma$ also has the global Beth–property.*

Theorem 6.3.11 *Every logic of a regular string language has the global Beth–property.*

If the axiomatization is infinite, by the described procedure we get an infinite array of formulae. This does not have a regular solution in general, as the reader is asked to show in the exercises.

The procedure of phonemicization is inverse to the procedure of adding features that we have looked at in the previous section. We shall briefly look at this procedure from a phonological point of view. Assume that we have an alphabet A of phonemes, containing also the syllable boundary marker + and the word boundary marker #. These are not brackets, they are separators. Since a word boundary is also a syllable boundary, no extra marking of the syllable is done at the word boundary. Let us now ask what the rules are of syllable and word structure in a language. The minimal assumption is that any combination of phonemes may form a syllable. This turns out to be false. Syllables are in fact constrained by a number of (partly language dependent) principles. This is so since the vocal tract has a certain physiognomy that discourages certain phoneme combinations while it enhances others. These properties also lead to a deformation of sounds in contact, which is called **sandhi**, a term borrowed from Sanskrit grammar. A particular example of sandhi is assimilation ([np] > [mp]). Sandhi rules exist in nearly all languages, but the scope and character varies greatly. Here, we shall call *sandhi* any constraint that is posed on the occurrence of two phonemes (or sounds) next to each other. Sandhi rules are 2–templates in the sense of the following definition.

Definition 6.3.12 *Let A be an alphabet. An n–**template over** A (or **template of length** n) is a cartesian product of length n of subsets of A. A language L is an n–**template language** if there is a finite set $\mathcal{P}$ of length n such that L is the set of words $\vec{x}$ such that every subword of length n belongs to at least one template from $\mathcal{P}$. L is a **template language** if there is an n such that L is an n–template language.*

Obviously, an n–template language is an $n+1$–template language. Furthermore, 1–template languages have the form B^* where $B \subseteq A$. So the first really interesting class is that of the 2–template languages. It is clear that if the alphabet is finite, we may actually define an n–template to be just a member of A^n. Hence, a template language is defined by naming all those sequences of bounded length that are allowed to occur.

Proposition 6.3.13 *A language is a template language if and only if its class of A–strings is axiomatizable by finitely many positive* EPDL*–formulae.*

To make this more realistic we shall allow also boundary templates. Namely, we shall have a set $\mathcal{P}^-$ of left edge templates and a set $\mathcal{P}^+$ of right edge templates. $\mathcal{P}^-$ lists the admissible n–prefixes of a word and $\mathcal{P}^+$ the admissible n–suffixes. Call such languages

boundary template languages. Notice that phonological processes are conditioned by certain boundaries, but we have added the boundary markers to the alphabet. This effectively eliminates the need for boundary templates in the description here. We have not explored the question what would happen if they were eliminated from the alphabet.

Proposition 6.3.14 *A language is a boundary template language if and only if its class of A–strings is axiomatizable by finitely many* EPDL*–formulae.*

It follows by a result of Büchi that template languages are regular (which is easy to prove anyhow). However, the language $\mathtt{ca}^+\mathtt{c} \cup \mathtt{da}^+\mathtt{d}$ is regular but not a template language.

The set of templates effectively names the legal transitions of an automaton that uses the alphabet A itself as the set of states to recognize the language. We shall define this notion, using a slightly different concept here, namely that of a **partial finite state automaton**. This is a quintuple $\mathfrak{A} = \langle I, Q, F, A, \delta\rangle$, such that A is the **input alphabet**, Q the set (!) of internal states, I the set of initial states, F the set of accepting states and $\delta \subseteq A \times Q \to Q$ a partial function. $\mathfrak{A}$ **accepts** $\vec{x}$ if there is a computation from some $q \in I$ to some $q' \in F$ with $\vec{x}$ as input. $\mathfrak{A}$ is a **2–template** if $Q = A$ and $\delta(a, b)$ is either undefined or $\delta(a, b) = b$.

The reason for concentrating on 2–template languages is the philosophy of naturalness. Basically, grammars are natural if the nonterminal symbols is identified with a set of terminal symbols. Alternatively put: for every nonterminal X there is a terminal a such that for every X–string $\vec{x}$ we have $\mathbb{C}_L(\vec{x}) = \mathbb{C}_L(a)$. For a regular grammar this means in essence that a string beginning with a has the same distribution as the letter a itself. A moment's reflection reveals that this is the same as the property of being 2–template. Notice that the 2–template property of words and syllables was motivated from the nature of the articulatory organs, and we have described a parser that recognizes whether something is a syllable or a word. Although it seems prima facie plausible that there are also auditory constraints on phoneme sequences we know of no plausible constraint that could illustrate it. We shall therefore concentrate on the former. What we shall show now is that syllables are not 2–template. This will motivate either adding structure or adding more features to the description of syllables. These features are necessarily nonphonemic.

We shall show that nonphonemic features exist by looking at syllable structure. It is not possible to outline a general theory of syllable structure. However, the following sketch may be given (see (Grewendorf *et al.*, 1987)). The sounds are aligned into a so called **sonoricity hierarchy**, which is as follows. (vd. = voiced, vl. = voiceless.)

dark vowels	> mid vowels	> high vowels	> r–sounds
[a], [o]	[æ], [œ]	[i], [y]	[r]
> nasals; laterals	> vd. fricatives	> vd. plosives	> vl. fricatives
[s], [ʃ]	[m], [n]; [l]	[z], [ʒ]	[b], [d]
> vl. plosives			
[p], [t]			

The syllable is organized as follows.

> *Syllable Structure*. Within a syllable the sonoricity increases monotonically and then decreases.

This means that a syllable must contain at least one sound which is at least as high as all the others in the syllable. It is called the **sonoricity peak**. We shall make the following assumption that will simplify the discussion.

> *Sonoricity Peak.* The sonoricity peak can be constituted by vowels only.

This wrongly excludes the syllable [krk], or [dn]. The latter is heard in the German verb `verschwinden` [ˌfɛa ˈʃwɪndn]. (The [e] that appears in writing is hardly ever pronounced.) However, even if the assumption is relaxed, the problem that we shall address will remain.

The question is: how do we implement these constraints? There are two ways of doing so that interest us here. (a) We state them by means of $\mathsf{PDL}^{\smile}$–formulae. This is the descriptive approach. (b) We code them. This means that we add some features in such a way that the resulting restrictions become specifiable by 2–templates. The second approach has some motivation as well. The added features can be identified as states of a productive (or analytic) device. Thus, while the solution under (a) tells us what the constraint actually is, the approach under (b) gives us features by which we can identify as (sets of) states of a (finite state) machine that actually parses or produces the structures. That this can be done is a corollary of the coding theorem.

Theorem 6.3.15 *Any regular language is the homomorphic image of a boundary 2–template language.*

So, we only need to add features. Phonological string languages are regular, so this method can be applied. Let us see how we can find a 2–template solution for the sonoricity hierarchy. We introduce a feature α and its negation $-\alpha$. We start with the alphabet P, and let $C \subseteq P$ be the set of consonants. The new alphabet is

$$\Xi := P \times \{-\alpha\} \cup C \times \{\alpha\}$$

Let $son(a)$ be the sonoricity of a.

$$\begin{array}{rl} \nabla := & \{\langle\langle a, \alpha\rangle, \langle a', \alpha\rangle\rangle : son(a) \leq son(a')\} \\ \cup & \{\langle\langle a, -\alpha\rangle, \langle a', -\alpha\rangle\rangle : son(a) \geq son(a')\} \\ \cup & \{\langle\langle a, \alpha\rangle, \langle a', -\alpha\rangle\rangle : a' \notin C\} \\ \cup & \{\langle\langle a, \gamma\rangle, \langle a', \gamma'\rangle\rangle : a \in \{+, \#\}, \gamma, \gamma' \in \{\alpha, -\alpha\}\} \end{array}$$

As it is now, any subword of a word is in the language. We need to mark the beginning and the end of a sequence in a special way, as described above. This detail shall be ignored here.

α has a clear phonetic interpretation: it signals the rise of the sonoricity. It has a natural correlate in what de Saussure calls **explosive articulation**. A phoneme carrying α is pronounced with explosive articulation, a phoneme carrying $-\alpha$ is pronounced with **implosive articulation**. (See (Saussure, 1967).) So, α actually has an articulatory (and an auditory) correlate. But it is a nonphonemic feature; it has been introduced in addition to the phonemic features in order to constrain the choice of the next phoneme. As de Saussure remarks, it makes the explicit marking of the syllable boundary unnecessary. The syllable boundary is exactly where the implosive articulation changes to explosive articulation. However, some linguists (for example van der Hulst in (1984)) have provided a completely different answer. For them, a syllable is structured in the following way.

$$[\text{onset} \quad [\text{nucleus} \quad \text{coda}]]$$

So, the grammar that generates the phonological strings is actually not a regular grammar but context free (though it makes only very limited use of phrase structure rules). α marks the onset, while $-\alpha$ marks the nucleus together with the coda (which is also called **rhyme**). So, we have three possible ways to arrive at the constraint for the syllable structure: we postulate an axiom, we introduce a new feature, or we assume more structure.

We shall finally return to the question of spelling out the relation between deep and surface phonological representations. We describe here the most simple kind of a machine that transforms strings into strings, the *finite state transducer.*

Definition 6.3.16 *Let A and B be alphabets. A* ***(partial) finite state transducer from*** *A* ***to*** *B is a quadruple $\mathfrak{T} = \langle Q, i_0, F, \delta\rangle$ such that $i_0 \in Q$, $F \subseteq Q$ and $\delta : Q \times A_\varepsilon \to \wp(Q \times B^*)$ where $\delta(q, \vec{x})$ is always finite for every $\vec{x} \in A_\varepsilon$. Q is called the set of* ***states****, i_0 is called the* ***initial state****, F the set of* ***accepting states*** *and δ the* ***transition function****. $\mathfrak{T}$ is called* ***deterministic*** *if $\delta(q, a)$ contains at most one element for every $q \in Q$ and every $a \in A$.*

We call A the **input alphabet** and B the **output alphabet**. The transducer differs from a finite automaton in the transition function. This function does not only say into which state the automaton may change but also what symbol(s) it will output on going into that state. Notice that the transducer may also output an empty string and that it allows for empty transitions. These are not eliminable (as they would in the finite state automaton) since the machine may accompany the change in state by a nontrivial output. We write

$$q \xrightarrow{\vec{x}:\vec{y}} q'$$

if the transducer changes from state q with input $\vec{x}$ $(\in A^*)$ into the state q' and outputs the string $\vec{y}$ $(\in B^*)$. This is defined as follows.

$$q \xrightarrow{\vec{x}:\vec{y}} q', \quad \text{if} \begin{cases} & (q', \vec{y}) \in \delta(q, \vec{x}) \\ \text{or} & \text{for some } q'', \vec{u}, \vec{u}', \vec{v}, \vec{v}' : \\ & q \xrightarrow{\vec{u}:\vec{v}} q'' \xrightarrow{\vec{u}':\vec{v}'} q' \\ & \text{and } \vec{x} = \vec{u} \cdot \vec{u}', \vec{y} = \vec{v} \cdot \vec{v}'. \end{cases}$$

Finally one defines

$$L(\mathfrak{T}) := \{\langle \vec{x}, \vec{y} \rangle : \text{ there is } q \in F \text{ with } i_0 \xrightarrow{\vec{x}:\vec{y}} q\} .$$

Transducers can be used to describe the effect of rules. One can write, for example, a transducer $\mathfrak{Syl}$ that syllabifies a given input according to the constraints on syllable structure. Its input alphabet is $A \cup \{\Box, \sharp\}$, where A is the set of phonemes, $\Box$ the word boundary and $\sharp$ the syllable boundary. The output alphabet is $A \times \{\mathtt{o}, \mathtt{n}, \mathtt{c}\} \cup \{\Box, \sharp\}$. Here, $\mathtt{o}$ stands for *onset*, $\mathtt{n}$ for *nucleus* and $\mathtt{c}$ for *coda*. The machine annotates each phoneme stating whether it belongs to the onset of a syllable, to its nucleus or its coda. Additionally, the machine inserts a syllable boundary wherever necessary. (So, one may leave the input partially or entirely unspecified for the syllable boundaries. The machine will look which syllable segmentation can or must be introduced.) Now we write a machine $\mathfrak{AVh}$ which simulates the actions of final devoicing. It has one state, i_0, it is deterministic and the transition function consists in $\langle[\mathrm{b}], \mathtt{c}\rangle : \langle[\mathrm{p}], \mathtt{c}\rangle$, $\langle[\mathrm{d}], \mathtt{c}\rangle : \langle[\mathrm{t}], \mathtt{c}\rangle$, $\langle[\mathrm{g}], \mathtt{c}\rangle : \langle[\mathrm{k}], \mathtt{c}\rangle$ as well as $\langle[\mathrm{z}], \mathtt{c}\rangle : \langle[\mathrm{s}], \mathtt{c}\rangle$ and $\langle[\mathrm{v}], \mathtt{c}\rangle : \langle[\mathrm{f}], \mathtt{c}\rangle$. Everywhere else we have $\langle P, \alpha\rangle : \langle P, \alpha\rangle$, P a phoneme, $\alpha \in \{\mathtt{a}, \mathtt{c}, \mathtt{n}\}$.

We shall now prove a general theorem which is known as the *Transducer Theorem.* It says that the image under transduction of a regular language is again a regular language. The proof is not hard. First we can replace the function $\delta : Q \times A_\varepsilon \to \wp(Q \times B^*)$ by a function $\delta^+ : Q^\diamond \times A_\varepsilon \to \wp(Q^\diamond \times B_\varepsilon)$ by adding some more states. The details of this construction are left to the reader. Now we replace this function by $\delta^2 : Q \times A_\varepsilon \times B_\varepsilon \to \wp(Q)$. What we now have is an automaton over the alphabet $A_\varepsilon \times B_\varepsilon$. We now take over the notation from the Section 5.3 and write pairs $\vec{x} \otimes \vec{y}$. The only difference is the following: we define

$$(\vec{u} \otimes \vec{v}) \cdot (\vec{w} \otimes \vec{x}) := (\vec{u} \cdot \vec{w}) \otimes (\vec{v} \cdot \vec{x}).$$

Definition 6.3.17 *Let R be a regular term. We define $L^2(R)$ as follows.*

$$\begin{array}{lcll}
L^2(0) & := & \varnothing, & \\
L^2(\vec{x} \otimes \vec{y}) & := & \{\vec{x} \otimes \vec{y}\}, & \vec{x} \otimes \vec{y} \in A_\varepsilon \times B_\varepsilon, \\
L^2(R \cdot S) & := & \{\mathfrak{x} \cdot \mathfrak{y} : \mathfrak{x} \in L^2(R), \mathfrak{y} \in L^2(S)\}, & \\
L^2(R \cup S) & := & L^2(R) \cup L^2(S), & \\
L^2(R^*) & := & L^2(R)^*. &
\end{array}$$

*A **regular relation on** A is a relation of the form $L^2(R)$ for some regular term R.*

Theorem 6.3.18 *A relation $Z \subseteq A^* \times B^*$ is regular if and only if there is a finite state transducer $\mathfrak{T}$ such that $L(\mathfrak{T}) = Z$.*

This is essentially a consequence of the Kleene's Theorem. In place of the alphabets A we have chosen the alphabet $A_\varepsilon \times B_\varepsilon$. Now observe that the transitions $\varepsilon : \varepsilon$ do not add anything to the language. We can draw a lot of conclusions from this.

Corollary 6.3.19 (Transducer Theorem) *The following holds.*

- ▷ *Regular relations are closed unter intersection and converse.*
- ▷ *If $H \subseteq A^*$ is regular so is $H \times B^*$. If $K \subseteq B^*$ is regular so is $A^* \times K$.*
- ▷ *If $Z \subseteq A^* \times B^*$ is a regular relation, so are the projections*
  - $\pi_1[Z] := \{\vec{x} : \textit{there is } \vec{y} \textit{ with } \langle\vec{x}, \vec{y}\rangle \in Z\}$,
  - $\pi_2[Z] := \{\vec{y} : \textit{there is } \vec{x} \textit{ with } \langle\vec{x}, \vec{y}\rangle \in Z\}$.
- ▷ *If Z is a regular relation and $H \subseteq A^*$ a regular set then $Z[H] := \{\vec{y} : \textit{there is } \vec{x} \in H \textit{ with } \langle\vec{x}, \vec{y}\rangle \in Z\}$ is a regular set.*

One can distinguish two ways of using a transducer. The first is as a machine which checks for a pair of strings whether they stand in a particular regular relation. The second, whether for a given string over the input alphabet there is a string over the output alphabet that stands in the regular to it. In the first use we can always transform the transducer into a deterministic one that recognizes the same set. In the second case this is impossible. The relation $\{\langle \mathsf{a}, \mathsf{a}^n\rangle : n \in \omega\}$ is regular but there is no deterministic translation algorithm. One easily finds a language in which there is no deterministic algorithm in any of the directions. From the previous results we derive the following consequence.

Corollary 6.3.20 (Kaplan & Kay) *Let $R \subseteq A^* \times B^*$ and $S \subseteq B^* \times C^*$ be regular relations. Then $R \circ S \subseteq A^* \times C^\times$ is regular.*

Proof. By assumption and the previous theorems, both $R \times C^*$ and $A^* \times S$ are regular. Furthermore, $(R \times C^*) \cap A^* \times S) = \{\langle\vec{x}, \vec{y}, \vec{z}\rangle : \langle\vec{x}, \vec{y}\rangle \in R, \langle\vec{y}, \vec{z}\rangle \in S\}$ is regular, and so is its projection onto $A^* \times B^*$, which is exactly $R \circ S$. □

This theorem is important. It says that the composition of rules which define regular relations is again a regular relation. Effectively, what distinguishes regular relations from Type 1 grammars is that the latter allow arbitrary iterations of the same process, while the former do not.

Notes on this section. The idea of eliminating features was formulated in (Kracht, 1997) and already brought into correspondence with the notion of implicit definability.

Exercise 213. Show that for every given string in a language there is a separation into syllables that conforms to the *Syllable Structure* constraint.

Exercise 214. Let $\Pi_0 := \{\zeta_i : i < n\}$ be a finite set of basic programs. Define $M := \{\zeta_i : i < n\} \cup \{\zeta_i^{\smile} : i < n\}$. Show that for every $\mathsf{EPDL}^{\smile}$ formula φ there is a modal formula δ over the set M of modalities such that $\mathsf{PDL}^{\smile} \vdash \delta \leftrightarrow \varphi$. *Remark.* A modal formula is a formula that has no test, and no $\cup$ and ;. Whence it can be seen as a $\mathsf{PDL}^{\smile}$–formula.

Exercise 215. The results of the previous section show that there is a translation $^{\heartsuit}$ of $\mathsf{PDL}^{\smile}(\prec)$ into QML. Obviously, the problematic symbols are * and $^{\smile}$. With respect to $^{\smile}$ the technique shown above works. Can you suggest a perspicuous translation of $[\alpha^*]\varphi$? *Hint.* $[\alpha^*]\varphi$ holds if φ holds in the smallest set of worlds closed under α–successors containing the current world. This can be expressed in QML rather directly.

Exercise 216. Show that in Theorem 6.3.11 the assumption of regularity is necessary. *Hint.* For example, show that the logic of $L = \{\mathsf{a}^{2^n}\mathsf{c}\mathsf{a}^n : n \in \omega\}$ fails to have the global Beth–property.

Exercise 217. Prove Lemma 6.3.10.

Exercise 218. Historical linguistics (or one branch of it) tries to reconstruct the affiliation of languages, preferrably by reconstructing a parent language for a certain group of languages and showing how the languages of that group developed from that parent language. The success of the reconstruction lies in the establishment of so called sound correspondences. In the easiest case they take the shape of correspondences between sounds of the various languages. Let us take the Indo–European languages. Indo–European is not known directly to us. The proof of its existence is — among other — the successful establishment of such correspondences. Their reliability and range of applicability have given credibility to the hypothesis of a common proto language, called Indo–European. Its sound structure is reconstructed, and is added to the sound correspondences.

Indo–European	Sanscrit	Greek	Latin	(meaning)
g^{u}hermos	gharmah	thermos	formus	*warm*
ou̯is	avih	ois	ovis	*sheep*
su̯os	svah	hos	suus	*to be*
septm̥	sapta	hepta	septem	*seven*
dek^{u}m̥	daśa	deka	decem	*ten*
neu̯os	navah	neos	novus	*new*
ĝenos	janah	genos	genus	*gender*
su̯epnos	svapnah	hypnos	somnus	*sleep*

Some sounds of one language have exact correspondences in another. For example, [p] corresponds to [p] across all languages. With other sounds the correspondence is not so easy. The Indo–European sounds [e] and [a] become [a] in Sanscrit. So, Sanscrit [a] has multiple correspondences in other languages. Finally, sounds develop differently in different environments. In the onset, Indo–European [s] becomes Sanscrit [s], but [h] at the end of the word. The details may not interest us here. Write a transducer for all sound correspondences displayed here.

Exercise 219. (Continuing the previous exercise.) Let L_i, $i < n$, be languages over alphabets A_i. Show the following: *Suppose R is a regular relation between L_i, $i < n$. Then there is an alphabet P, a proto–language $Q \subseteq P^*$, and regular relations $R_i \subseteq P^* \times A_i^*$ such that (a) for every $\vec{x} \in P$ there is exactly one $\vec{y}$ such that $\vec{x}\, R_i\, \vec{y}$ and (b) L_i is the image of P under R_i.*

Exercise 220. Finnish has a phenomenon called *vowel harmony*. There are three kinds of vowels: back vowels ([a], [o], [u], written `a`, `o` and `u`, respectively), front vowels ([æ, ø, y], written `ä`, `ö` and `y`, respectively) and neutral vowels ([e], [i], written `e` and `i`). The principle is this.

> *Vowel harmony (Finnish).* A word contains not both a back and a front harmonic vowel.

The vowel harmony only goes up to the word boundary. So, it is possible to combine two words with different harmony. Examples are `osakeyhtiö` (*share holder company*). It consists of the back harmonic word `osake` (*share*) and the front harmonic word `yhtiö` (*society*). First, give an $\mathsf{PDL}^{\smile}$–definition of strings that satisfy Finnish vowel harmony. It follows that there is a finite automaton that recognizes this language. Construct such an automaton. *Hint.* You may need to explicitly encode the word boundary.

6.4 Axiomatic Classes II: Exhaustively Ordered Trees

The theorem by Büchi on axiomatic classes of strings has a very interesting analogon for exhaustively ordered trees. We shall prove it here; however, we shall only show those facts that are not proved in a completely similar way. Subsequently, we shall outline the importance of this theorem for syntactic theory. The reader may make himself acquainted with the notions of Section 1.4. Ordered trees are structures over a language that has two binary relation symbols, $\sqsubset$ and $<$. We also take labels from A and N (!) in the form of constants and get the language MSO^b. In this language the set of exhaustively ordered trees is a finitely axiomatizable class of structures. We consider first the postulates. $<$ is transitive and irreflexive, $\uparrow x$ is linear for every x, and there is a largest element, and every subset has a largest and a smallest element with respect to $<$. From this it follows in particular that below an arbitrary element there is a leaf. Here are now the axioms listed in the order just described.

$$\begin{aligned}
&(\forall xyz)(x < y \wedge y < z. \to .x < z),\\
&(\forall x)\neg(x < x),\\
&(\forall xyz)(x < y \wedge x < z. \to .y < z \vee y \doteq z \vee y > z),\\
&(\exists x)(\forall y)(y < x \vee y \doteq x),\\
&(\forall P)(\exists x)(\forall y)(P(x) \wedge P(y). \to .x > y \vee x \doteq y).\\
&(\forall P)(\exists x)(\forall y)(P(x) \wedge P(y). \to .x < y \vee x \doteq y)
\end{aligned}$$

In what is to follow we use the abbreviation $x \leq y := x < y \vee x \doteq y$. Now we shall lay down the axioms for the ordering. $\sqsubset$ is transitive and irreflexive, it is linear on the leaves, and we have $x \sqsubset y$ if and only if for all leaves u below x and all leaves $v \leq y$ we have $u \sqsubset v$. Finally, there are only finitely many leaves, a fact which we can express by requiring that every set of nodes has a smallest and a largest member (with respect to $\sqsubset$). We put $b(x) := \neg(\exists y)(y < x)$.

$$\begin{aligned}
&(\forall xyz)(x \sqsubset y \wedge y \sqsubset z. \to .x \sqsubset z),\\
&(\forall x)\neg(x \sqsubset x),\\
&(\forall xy)(b(x) \wedge b(y). \to .x \sqsubset y \vee x \doteq y \vee y \sqsubset x),\\
&(\forall xy)(x \sqsubset y \leftrightarrow (\forall uv)(b(u) \wedge u \leq x \wedge b(v) \wedge v \leq y. \to .u \sqsubset v)),\\
&(\forall P)\{(\forall x)(P(x) \to b(x)) \to (\exists y)(P(y) \wedge (\forall z)(P(z) \to \neg(z \sqsubset y)))\},\\
&(\forall P)\{(\forall x)(P(x) \to b(x)) \to (\exists y)(P(y) \wedge (\forall z)(P(z) \to \neg(z \sqsupset y)))\}.
\end{aligned}$$

Thirdly, we must regulate the distribution of the labels.

$$\begin{aligned}
&(\forall x)(b(x) \leftrightarrow \bigvee\langle \underline{a}(x) : a \in A\rangle),\\
&(\forall x)(b(x) \to \bigwedge\langle \underline{a}(x) \to \neg\underline{b}(x) : a \neq b\rangle),\\
&(\forall x)(\neg b(x) \to \bigvee\langle \underline{A}(x) : A \in N\rangle),\\
&(\forall x)(\neg b(x) \to \bigwedge\langle \underline{A}(x) \to \neg\underline{B}(x) : A \neq B\rangle).
\end{aligned}$$

The fact that a tree is exhaustively ordered is described by the following formula.

$$(\forall xy)(\neg(x \leq y \vee y \leq x). \to .x \sqsubset y \vee y \sqsubset x)\ .$$

Proposition 6.4.1 *The following are finitely* MSO^b*–axiomatizable classes.*

1. *The class of ordered trees.*

2. *The class of finite exhaustively ordered trees.*

Likewise we can define a quantified modal language. However, we shall change the base of relation as follows. The basis is the Exercise 1.4. We assume 8 operators, $M_8 := \{\Diamond_{\uparrow}, \Diamond_{\uparrow}^{+}, \Diamond_{\downarrow}, \Diamond_{\downarrow}^{+}, \Diamond_{\rightarrow}, \Diamond_{\rightarrow}^{+}, \Diamond_{\leftarrow}, \Diamond_{\leftarrow}^{+}\}$, which correspond with the following relations. $\prec$, $<$, $\succ$, $>$, *immediate left sister of*, *left sister of*, *immediate right sister of*, as well as *right sisters of*. There relations are MSO–definable from the original ones, and conversely the original relations can be MSO–defined from the present ones. Let $\mathfrak{B} = \langle B, <, \sqsubset\rangle$ be an exhaustively ordered tree. Then we define $R : M_8 \to B \times B$ as follows.

$$\begin{array}{lcl}
x\ R(\Diamond_{\rightarrow}^{+})\ y & := & x \sqsubset y \wedge (\exists z)(x \prec z \wedge y \prec z), \\
x\ R(\Diamond_{\rightarrow})\ y & := & x\ R(\Diamond_{\rightarrow}^{+})\ y \wedge \neg(\exists z)(x\ R(\Diamond_{\rightarrow}^{+})\ z \wedge z\ R(\Diamond_{\rightarrow}^{+})\ y), \\
x\ R(\Diamond_{\leftarrow}^{+})\ y & := & x \sqsupset y \wedge (\exists z)(x \succ z \wedge y \succ z), \\
x\ R(\Diamond_{\leftarrow})\ y & := & x\ R(\Diamond_{\leftarrow}^{+})\ y \wedge \neg(\exists z)(x\ R(\Diamond_{\leftarrow}^{+})\ z \wedge z\ R(\Diamond_{\leftarrow}^{+})\ y), \\
x\ R(\Diamond_{\uparrow}^{+})\ y & := & x > y, \\
x\ R(\Diamond_{\uparrow})\ y & := & x \succ y, \\
x\ R(\Diamond_{\downarrow}^{+})\ y & := & x < y, \\
x\ R(\Diamond_{\downarrow})\ y & := & x \prec y.
\end{array}$$

The resulting structure we call $M(\mathfrak{B})$. Now if B as well as R are given, then the relations $\prec$, $\succ$, $<$, $>$, and $\sqsubset$, as well as $\sqsupset$ are definable from these relations. First we define $\Diamond_{\uparrow}^{*}\varphi := \varphi \wedge \Diamond_{\uparrow}^{+}\varphi$, and likewise for the other relations. Then $R(\Diamond_{\uparrow}^{*}) = \Delta \cup R(\Diamond_{\uparrow}^{+})$.

$$\begin{array}{lcl}
\prec & = & R(\Diamond_{\uparrow}), \\
\succ & = & R(\Diamond_{\downarrow}), \\
< & = & R(\Diamond_{\uparrow}^{+}), \\
> & = & R(\Diamond_{\downarrow}^{+}), \\
\sqsubset & = & R(\Diamond_{\uparrow}^{*}) \circ R(\Diamond_{\rightarrow}^{+}) \circ R(\Diamond_{\downarrow}^{*}), \\
\sqsupset & = & R(\Diamond_{\uparrow}^{*}) \circ R(\Diamond_{\leftarrow}^{+}) \circ R(\Diamond_{\downarrow}^{*}).
\end{array}$$

Analogously, as with the strings we can show that the following properties are axiomatizable: (a) that $R(\Diamond_{\leftarrow}^{+})$ is transitive and irreflexive with converse relation $R(\Diamond_{\rightarrow}^{+})$; (b) that $R(\Diamond_{\leftarrow}^{+})$ is the transitive closure of $R(\Diamond_{\leftarrow})$ and $R(\Diamond_{\rightarrow}^{+})$ the transitive closure of $R(\Diamond_{\rightarrow})$. Likewise for $R(\Diamond_{\uparrow}^{+})$ and $R(\Diamond_{\uparrow})$, $R(\Diamond_{\downarrow}^{+})$ and $R(\Diamond_{\downarrow})$. With the help of the axiom below we axiomatically capture the condition that $\uparrow x$ is linear:

$$\Diamond_{\uparrow}^{+} p \wedge \Diamond_{\uparrow}^{+} q. \rightarrow .\Diamond_{\uparrow}^{+}(p \wedge q) \vee \Diamond_{\uparrow}^{+}(p \wedge \Diamond_{\uparrow}^{+} q) \vee \Diamond_{\uparrow}^{+}(q \wedge \Diamond_{\uparrow}^{+} p).$$

The other axioms are more cumbersome. Notice first the following.

Lemma 6.4.2 *Let $\langle B, <, \sqsubset \rangle$ be an exhaustively ordered tree and $x, y \in B$. Then $x \neq y$ if and only if (a) $x < y$ or (b) $x > y$ or (c) $x \sqsubset y$ or (d) $x \sqsupset y$.*

Hence the definitions.

$$\begin{array}{lcl} \langle \neq \rangle \varphi & := & \Diamond_{\uparrow}^{+} \varphi \wedge \Diamond_{\downarrow}^{+} \varphi \vee \Diamond_{\uparrow}^{*} \Diamond_{\rightarrow}^{+} \Diamond_{\downarrow}^{*} \varphi \vee \Diamond_{\uparrow}^{*} \Diamond_{\leftarrow}^{+} \Diamond_{\downarrow}^{*} \varphi, \\ \boxtimes \varphi & := & \varphi \wedge [\neq] \varphi. \end{array}$$

So we take the following additional axioms.

$$\begin{array}{ll} \boxtimes \varphi \rightarrow \Box_{\rightarrow}^{+} \varphi, & \boxtimes \varphi \rightarrow \Box_{\leftarrow}^{+} \varphi, \\ \boxtimes \varphi \rightarrow \Box_{\uparrow}^{+} \varphi, & \boxtimes \varphi \rightarrow \Box_{\downarrow}^{+} \varphi, \\ \boxtimes \varphi \rightarrow \boxtimes \boxtimes \varphi, & \boxtimes \varphi \rightarrow \varphi, \\ \varphi \rightarrow \boxtimes \neg \boxtimes \neg \varphi\ . & \end{array}$$

(Most of them are already derivable. The axiom systems if therefore not minimal.) These axioms see to it that in a structure every node is reachable from any other by means of the basic relations, moreover, that it is reachable in one step using $R(\boxtimes)$. Here we have

$$R(\boxtimes) = \{\langle x, y \rangle : x = y \textit{ or the is } z : x < z > y\}\ .$$

Notice that this always holds in a tree and that conversely it follows from the above axioms that $R(\Diamond_{\uparrow}^{+})$ possesses a largest element.

Now we put

$$b(\varphi) := \varphi \wedge \Box_{\downarrow}\bot \wedge [\neq]\neg\varphi \ .$$

$b(\varphi)$ holds at a node x if and only if x is a leaf and φ is true exactly at x. Now we can axiomatically capture the conditions that $R(\Diamond_{\rightarrow}^{+})$ must be linear on the set of leaves.

$$\Box_{\downarrow}\bot \wedge \langle\neq\rangle b(q). \to .p \vee \Diamond_{\rightarrow}^{+} p \vee \Diamond_{\leftarrow}^{+} p \ .$$

Finally, we have to take axioms which constrain the distribution of the labels. The reader will be able to supply them.

Proposition 6.4.3 *The class of exhaustively ordered trees is finitely* QML^b*–axiomatisable.*

We already know that QML^b can be embedded into MSO^b. The converse is as usual somewhat difficult. To this end we proceed as in the case of strings. We introduce an analogon of restricted quantifiers. We define functions $\Diamond_{\uparrow}$, $\Diamond_{\uparrow}^{+}$, $\Diamond_{\downarrow}$, $\Diamond_{\downarrow}^{+}$, $\Diamond_{\rightarrow}$, $\Diamond_{\rightarrow}^{+}$, $\Diamond_{\leftarrow}$, $\Diamond_{\leftarrow}^{+}$, as well as $\langle\neq\rangle$ on unary predicates, whose meaning should be self explanatory.

For example

$$\begin{array}{lcl} (\Diamond_{\uparrow}\varphi)(x) & := & (\exists y \succ x)\varphi(y) \ , \\ (\Diamond_{\uparrow}^{+}\varphi)(x) & := & (\exists y > x)\varphi(y) \ , \end{array}$$

where $y \notin fr(\varphi)$. Finally let O be defined by

$$O(\varphi) := (\forall x)\neg\varphi(x) \ .$$

Hence $O(\varphi)$ says nothing but that $\varphi(x)$ is nowhere satisfiable. Let P_x be a predicate variable which does not occur in φ. Define $\{P_x/x\}\varphi$ inductively as described in Section 6.2. Let $\gamma(P_x) = \{\beta(x)\}$. Then we have

$$\langle\mathfrak{M}, \gamma, \beta\rangle \vDash \varphi \qquad \text{if and only if} \qquad \langle\mathfrak{M}, \gamma, \beta\rangle \vDash \{P_x/x\}\varphi \ .$$

Therefore put

$$(Ex)\varphi(x) := (\exists P_x)(\neg O(P_x) \wedge O(P_x \wedge \langle\neq\rangle P_x). \to .\{P_x/x\}\varphi) \ .$$

We have to remark the following to this definition. Because of this fact we have for all exhaustively ordered trees $\mathfrak{B}$

$$\langle \mathfrak{B}, \gamma, \beta\rangle \vDash (\exists x)\varphi \qquad \text{if and only if} \qquad \langle \mathfrak{B}, \gamma, \beta\rangle \vDash (Ex)\varphi \ .$$

Let again $h : P \to PV$ be a bijection from the the set of predicate variables of MSO^b onto the set of proposition variables or QML^b.

$$\begin{array}{llllll}
(\underline{a}(x))^\diamond & := & Q^a, & (P(y))^\diamond & := & h(P), \\
(\Diamond^{\uparrow}(\varphi))^\diamond & := & \Diamond^{\uparrow}\varphi^\diamond, & (\Diamond^{\downarrow}\varphi)^\diamond & := & \Diamond^{\downarrow}\varphi^\diamond, \\
(\Diamond^{\leftarrow}\varphi)^\diamond & := & \Diamond^{\leftarrow}\varphi^\diamond, & (\Diamond^{\rightarrow}\varphi)^\diamond & := & \Diamond^{\rightarrow}\varphi^\diamond, \\
(\Diamond^{\uparrow +}\varphi)^\diamond & := & \Diamond^{\uparrow +}\varphi^\diamond, & (\Diamond^{\downarrow +}\varphi)^\diamond & := & \Diamond^{\downarrow +}\varphi^\diamond, \\
(\Diamond^{\rightarrow +}\varphi)^\diamond & := & \Diamond^{\rightarrow +}\varphi^\diamond, & (\Diamond^{\leftarrow +}\varphi))^\diamond & := & \Diamond^{\leftarrow +}\varphi^\diamond, \\
(\neg\varphi)^\diamond & := & \neg\varphi^\diamond, & (O(\varphi))^\diamond & := & \boxtimes\neg\varphi^\diamond, \\
(\varphi_1 \wedge \varphi_2)^\diamond & := & \varphi_1^\diamond \wedge \varphi_2^\diamond, & (\varphi_1 \vee \varphi_2)^\diamond & := & \varphi_1^\diamond \vee \varphi_2^\diamond, \\
((\exists P)\varphi)^\diamond & := & (\exists h(P))\varphi^\diamond, & ((\forall P)\varphi)^\diamond & := & (\forall h(P))\varphi^\diamond.
\end{array}$$

Then the desired embedding of MSO^b into QML^b is shown.

Theorem 6.4.4 *Let φ be an MSO^b–formula with at most one free variable, the object variable x_0. Then there exists a QML^b–formula φ^M such that for all exhaustively ordered trees $\mathcal{B}$:*

$$\langle \mathfrak{B}, \beta\rangle \vDash \varphi(x_0) \text{ if and only if } \langle M(\mathfrak{B}), \beta(x_0)\rangle \vDash \varphi(x_0)^M$$

Corollary 6.4.5 *Modulo the identification $\mathfrak{B} \mapsto M(\mathfrak{B})$ MSO^b and QML^b define the same model classes of exhaustively ordered trees. Further: $\mathcal{K}$ is a finitely axiomatizable class of MSO^b–structures if and only if $M(\mathcal{K})$ is a finitely axiomatizable class of QML^b–structures.*

For the purpose of definition of a code we suspend the difference between terminal and nonterminal symbols.

Definition 6.4.6 *Let $G = \langle \Sigma, N, A, R\rangle$ be a context free grammar* and $\varphi \in \mathsf{QML}^b$ a constant formula (with constants in A). We say, G is **faithful for** φ if there is a set $H_\varphi \subseteq N$ such that for every tree $\mathfrak{B}$ and every node $w \in B$: $\langle \mathfrak{B}, w\rangle \vDash \varphi$ if and only if*

$\ell(w) \in H_\varphi$. *We also say that* H_φ ***codes*** φ ***with respect to*** G. *Let* φ *be a* QML^b*–formula and* n *a natural number. An* n*–**code** for* φ *is a pair* $\langle G, H\rangle$ *such that* $L_B(G)$ *is the set of all at most* n*–ary branching, finite, exhaustively ordered trees over* $A \cup N$ *and* H *codes* φ *in* G. φ *is called* n*–**codable** if there is an* n*–code for* φ. φ *is called **codable** if there is an* n*–code for* φ *for every* n.

Notice that for technical reasons we must restrict ourselves to at most n–branching trees since we can otherwise not write down a context free grammar* as a code. Let $G = \langle \Sigma, N, A, R\rangle$ and $G' = \langle \Sigma', N', A, R'\rangle$ grammars* over A. For simplicity we assume that they are in standard form. The **product** is defined by

$$G \times G' = \langle \Sigma \times \Sigma', N \times N', A, R \times R'\rangle$$

where

$$\begin{array}{rl} R \times R' := & \{\langle X, X'\rangle \to \langle Y_0, Y_0'\rangle \dots \langle Y_{n-1}, Y_{n-1}'\rangle : \\ & \qquad X \to Y_0 \dots Y_{n-1} \in R, X' \to Y_0' \dots Y_{n-1}' \in R'\} \\ \cup & \{\langle X, X'\rangle \to a : X \to a \in R, X' \to a \in R'\}. \end{array}$$

To prove the analogon of the Coding Theorem for strings we shall have to use a trick. As one can easily show the direct extension on trees is false since we have also taken the nonterminal symbols as symbols of the language. So we proceed as follows. Let $h : N \to N'$ be a map and $\mathbb{G} = \langle B, <, \sqsubset, \ell\rangle$ a tree with labels in $B \cup N$. Then let $h[\mathfrak{B}] := \langle B, <, \sqsubset, h_A \circ \ell\rangle$ where $h_A \restriction N = h$ and $h_A(a) = a$ for all $a \in A$. Then $h[\mathfrak{B}]$ is called a **projection of** $\mathfrak{B}$. If $\mathcal{K}$ is a class of trees, then let $h[\mathcal{K}] := \{h[\mathfrak{B}] : \mathfrak{B} \in \mathcal{K}\}$. The theorem now is the following.

Theorem 6.4.7 (Thatcher & Wright, Doner) *Let* $n \in \omega$, A *be a terminal alphabet and* N *a nonterminal alphabet. A class of exhaustively ordered, at most* n*–branching finite trees over* $A \cup N$ *is finitely axiomatizable in* MSO^b *if and only if it is the projection onto* $A \cup N$ *of a context free* class of ordered trees over some alphabet.*

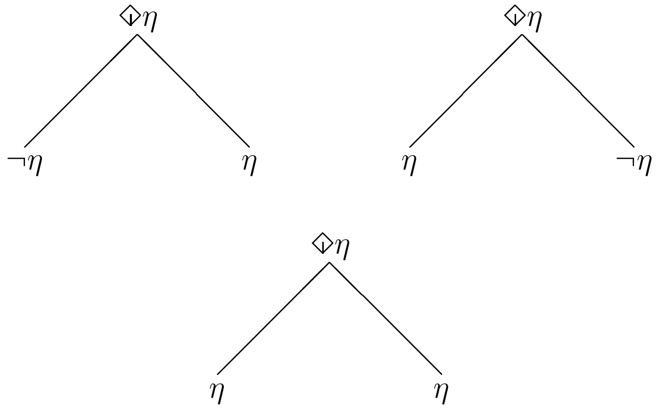
Here a class of of trees is **context free*** if it is the class if trees generated by some context free grammar*. Notice that the symbol ε is not problematic as it was for regular languages. We may look at it as an independent symbol which can be the label of a leaf. However, if this is to be admitted, we must assume that the terminal alphabet may be A_ε and not A. Notice that the union of two context free sets of trees is not necessarily itself context free. (This again is different with regular languages, since the structures did not contain the nonterminal symbols.)

From now on the proof is more or less the same. First one shows the codability of QML^b–formulae. Then one argues as follows. Let $\langle G, H\rangle$ be the code of a formula φ. We restrict the set of symbols (that is, both N as well as A) to H. In this way we get a grammar* which only generates trees that satisfy φ. Finally we define the projection $h : H \to A \cup N$ as follows. Put $h(a) := a$, $a \in A$, and $h(Y) := X$ if $L_B(G) \vDash (\forall x)(\underline{Y}(x) \to \underline{X}(x))$. In order for this to be well defined we must therefore for all $Y \in H$ have an $X \in N$ with this property. In this case we call the code **uniform**. Uniform codability follows easily from codability since we can always construct products $G \times G'$ of grammars* so that $G = \langle \Sigma, N, A, R\rangle$ and $L_B(G \times G') \vDash \underline{X}(\langle x, y\rangle)$ if and only if $L_B(G) \vDash \underline{X}(x)$. The map h is nothing but the projection onto the first component.

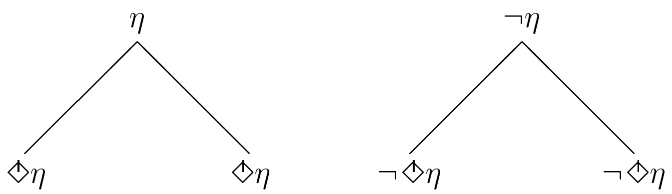
Theorem 6.4.8 *Every constant* QML^b*–formula is uniformly codable.*

Proof. We only deliver a sketch of the proof. We choose an n and show the uniform n–codability. For ease of exposition we illustrate the proof for $n = 2$. or the formulae $\underline{a}(x)$, $a \in A$, and $\underline{Y}(x)$, $Y \in N$, nothing special has to be done. Again, the boolean functors are easy. There remain the modal operators and the quantifiers. Before we begin we shall introduce a somewhat more convenient notation. As usual we assume that we have a grammar* $G = \langle \Sigma, N, A, R\rangle$ as well as some sets H_η for certain formulae. Now we take the product with a new grammar* and define H_φ. In place of explicit labels we now use the formulae themselves, where η stands for the set of labels from H_η.

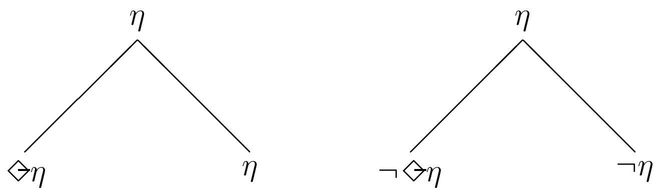
The basic modalities are as follows. We assume $\mathbf{2} = \langle\{0,1\}, \{0,1\}, A, R_2\rangle$, where R_2 consists of all possible n–branching rules of a grammar in standard form now we form the product of these two grammars*; however, we only choose a subset of rules and of the start symbols. Namely, we put $\Sigma' := \Sigma \times \{0,1\}$ and $H'_\eta := H_\eta \times \{0,1\}$, $H'_{\langle\downarrow\rangle\eta} := N \times \{1\}$. The rules are now all rules of the form



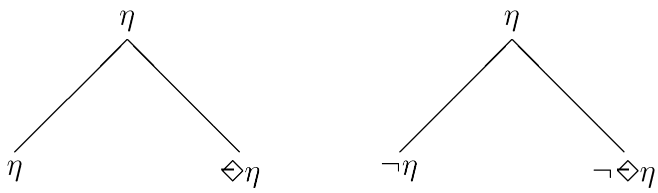
Now we proceed to $\langle\uparrow\rangle\eta$ über. Here $\Sigma'_{\langle\uparrow\rangle\eta} := N \times \{0\}$.



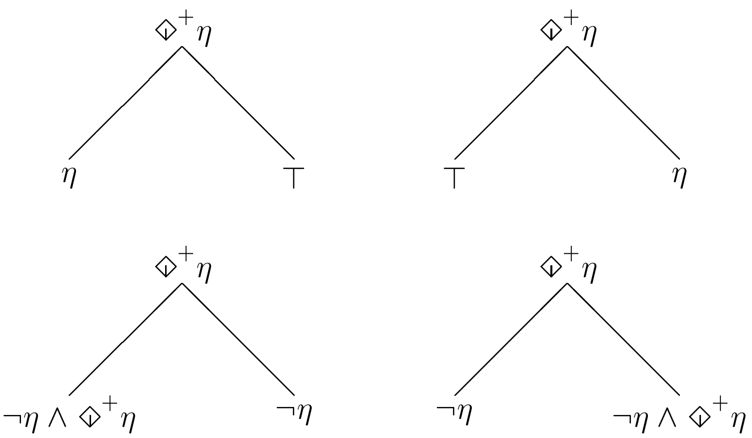
With $\langle\rightarrow\rangle\eta$ we choose $\Sigma'_{\langle\rightarrow\rangle\eta} = \Sigma \times \{0\}$.



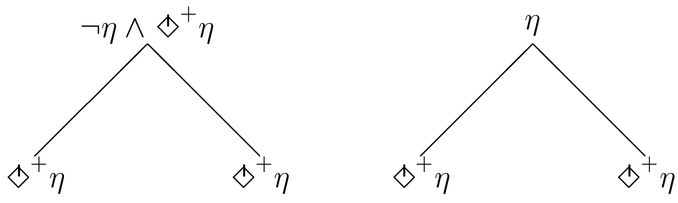
Likewise, $\Sigma'_{\langle\leftarrow\rangle\eta}$ is the start symbol of G' in the case of $\langle\leftarrow\rangle\eta$.



Now let us look at the relation $\langle\downarrow\rangle^{+}\eta$.



The set of start symbols is $\Sigma \times \{0, 1\}$.



The set of start symbols is $\Sigma' := \Sigma \times \{0\}$.

Finally we study the quantifier $(\exists p)\eta$. Let $\eta' := \eta[\underline{c}/p]$, where $\underline{c}$ is a new constant. Our terminal alphabet os therefore $A \times \{0, 1\}$, the nonterminal alphabet $N \times \{0, 1\}$. We assume that $\langle G^1, H^1_\theta \rangle$ is a uniform code for θ, θ an arbitrary subformula of η'. For every subset Σ of the set Δ of all subformulae of η' we put

$$L_\Sigma := \bigwedge_{\theta \in \Sigma} \theta \wedge \bigwedge_{\theta \in \Delta - \Sigma} \neg\theta \,.$$

Then $\langle G^1, H^1_\Sigma \rangle$ is a code for L_Σ if

$$H^1_\Sigma := \bigcap_{\theta \in \Sigma} H^1_\theta \cap \bigcap_{\theta \in \Delta - \Sigma} (N - H^1_\theta) \,.$$

Now we build a new grammar*, G^2. Let $N^2 := N \times \{0, 1\} \times \wp(N^1)$. The rules of this grammar* are all rules of the form

$$\begin{tikzcd}[column sep=small] & \langle X, i, \Sigma \rangle \arrow[dl, dash] \arrow[dr, dash] & \\ \langle Y_0, j_0, \Theta_0 \rangle & & \langle Y_1, j_1, \Theta_1 \rangle \end{tikzcd}$$

where $\langle X, i \rangle \in H^1_\Sigma$, $\langle Y_0, j_0 \rangle \in H^1_{\Theta_0}$, $\langle Y_1, j_1 \rangle \in H^1_{\Theta_1}$ and $\Sigma \to \Theta_0\Theta_1$ a rule of G^1. (This in turn is the case if there are X, Y_0 and Y_1 as well as i, j_0 and j_1 such that $\langle X, i \rangle \to \langle Y_0, j_0 \rangle \; \langle Y_1, j_1 \rangle \in R$.) Likewise for unary rules. Now we go over the grammar* G^3, with $N^3 := N \times \wp(\wp(N^1))$. Here we take all rules of the form

$$\begin{tikzcd}[column sep=small] & \langle X, \mathbb{A} \rangle \arrow[dl, dash] \arrow[dr, dash] & \\ \langle Y_0, \mathbb{B}_0 \rangle & & \langle Y_1, \mathbb{B}_1 \rangle \end{tikzcd}$$

where $\mathbb{A}$ is the set of all Σ for which there are Θ_0, Θ_1 and i, j_0, j_1 such that

$$
\begin{array}{ccc}
 & \langle X, i, \Sigma\rangle & \\
 \swarrow & & \searrow \\
\langle Y_0, j_0, \Theta_0\rangle & & \langle Y_1, j_1, \Theta_1\rangle
\end{array}
$$

is a rule of G^2. $\Box$*Notes on this section.* From complexity theory we know that context free languages, being in **PTIME**, should actually possess a description using first order logic plus inflationary fixed point operator. This means that we can describe the set of strings in $L(G)$ for a context free grammar by means of a formula that uses first order logic plus inflationary fixed points. Since we can assume G to be binary branching and invertible, it suffices to find a constituent analysis of the string. This is a set of subsets of the string, and so of too high complexity. What we need is a first order description of the constituency in terms of the string alone. The exercises describe a way to do this.

Exercise 221. Show the following: $<$ is definable from $\prec$, likewise $>$. Also, trees can be axiomatized alternatively with $\prec$ (or $\succ$). Show furthermore that in ordered trees $\prec$ is uniquely determined from $<$. Give an explicit definition.

Exercise 222. Let $x \, L \, y$ if x and y are sisters and $x \sqsubset y$. Show that in ordered trees L can be defined with $\sqsubset$ and conversely.

Exercise 223. Let $\mathfrak{T}$ be a tree over A and N such that every node that is not preterminal is at least 2–branching. Let $\vec{x} = x_0 \ldots x_{n-1}$ be the associated string. Define a set $C \subseteq n^3$ as follows. $\langle i, j, k\rangle \in C$ if and only if the least node above x_i and x_j is lower than the least node above x_i and x_k. Further, for $X \in N$, define $L_X \subseteq n^2$ by $\langle i, j\rangle \in L_X$ if and only if the least node above x_i and x_j has label X. Show that C uniquely codes the tree structure $\mathfrak{T}$ and L_X, $X \in N$, the labelling. Finally, for every $a \in A$ we have a unary relation $T_a \subseteq n$ to code the nodes of category a. Axiomatize the trees in terms of the relations C, L_X, $X \in N$, and T_a, $a \in A$.

Exercise 224. Show that a string of length n possesses 2^{cn^3} different constituent structures for some c.

6.5 Transformational Grammar

In this section we shall discuss the so called **Transformational Grammar**. Transformations have been introduced by Zellig Harris. They were operations that change one syntactic structure into another without changing the meaning. This concept has been taken over by his student, Noam Chomsky, who developed it into a very rich theory. Let us look at a simple example, a phenomenon known as **topicalization**.

(6.5.1) `Harry likes trains.`
(6.5.2) `Trains, Harry likes.`

We have two different English sentences, of which the first is in normal serialization, namely SVO, and the second in OSV order. In syntactic jargon we say that in the second sentence the object has been topicalized. (The metaphor is by the way a dynamic one. Speaking statically, one would prefer to express oneself differently.) The two sentences have different uses and probably also different meanings, but the meaning difference is hard to establish. For the present discussion this is however not really relevant. A transformation is a rule that allows us for example to transform (6.5.1) into (6.5.2). Transformations have the form SD $\Longrightarrow$ SC. Here SD stands for **structural description** and SC for **structural change**. The rule **TOP**, for **topicalization**, may be formulated as follows.

$$\mathrm{NP}_1 \ \mathrm{V} \ \mathrm{NP}_2 \ \mathrm{Y} \Longrightarrow \mathrm{NP}_2 \ \mathrm{NP}_1 \ \mathrm{V} \ \mathrm{Y}\ .$$

This means the following. If a structure can be decomposed into an NP followed by a V and a second NP followed in turn by an arbitrary string, then the rule applies and moves the second NP to the position immediately to the left of the first NP. Notice that Y is a variable for arbitrary strings while NP and V are variables for constituents of category NP and V, respectively. Since a string can possess several NPs or Vs we must have for every category a denumerable set of variables. Alternatively, we may write $[W]_{\mathrm{NP}}$. This denotes an arbitrary string which is an NP–constituent.

Analogously, we may formulate also the reversal of this rule:

$$\mathrm{NP}_2\ \mathrm{NP}_1\ \mathrm{V}\ \mathrm{Y} \Longrightarrow \mathrm{NP}_1\ \mathrm{V}\ \mathrm{NP}_2\ \mathrm{Y}\ .$$

However, one should be extremely careful with such rules. They often turn out to be too restrictive and often also too liberal. Let us look again at **TOP**. As formulated, it cannot be applied to (6.5.3) and (6.5.5), even though topicalization is admissible, as (6.5.4) and (6.5.6) show.

(6.5.3) `Harry might like trains.`
(6.5.4) `Trains, Harry might like.`
(6.5.5) `Harry certainly likes trains.`
(6.5.6) `Trains, Harry certainly likes.`

The problem is that in the SD V only stands for the verb, not for the complex consisting of the verb and the auxiliaries. So, we have to change the SD in such a way that it allows the examples above. Further, it must not be disturbed by eventually intervening adverbials.

German has a peculiarity which may be regarded as perhaps the strongest argument in favour of transformations, namely the so called V2–phenomenon. In German, the verb is at the end of the clause if that clause is subordinate. In a main clause, however, the part of the verb cluster that carries the inflection is moved to second position in the sentence. Compare the following sentences.

(6.5.7) `..., daß Hans sein Auto repariert.`
..., that Hans his car repairs.

(6.5.8) `Hans repariert sein Auto.`
Hans repairs his car.

(6.5.9) `..., daß Hans nicht in die Oper gehen will.`
..., that Hans not into the opera go wants.

(6.5.10) `Hans will nicht in die Oper gehen.`
Hans wants not into the opera go.

(6.5.11) `..., daß Hans im Unterricht selten aufpaßt.`
..., that Hans in class rarely PREF–attention.pay.

(6.5.12) `Hans paßt im Unterricht selten auf.`
Hans attention.pay in class rarely PREF.

As is readily seen, the auxiliaries and the verb are together in the subordinate clause, in the main clause the last of the series (which carries the inflection) moves into second place. Furthermore, as the last example illustrates, it can happen that certain prefixes of the verb are left behind when the verb moves. In transformational grammar one speaks of **V2–movement**. This is a transformation that takes the inflection carrier and moves it to second place in a main clause. A similar phenomenon is what might be called **damit-** or **davor–split**, which is found mainly in northern Germany.

(6.5.13) `Da hat er mich immer vor gewarnt.`
DA has he me always VOR warned.
He has always warned me of that.

(6.5.14) `Da konnte ich einfach nicht mit rechnen.`
DA could I simply not MIT reckon.
I simply could not reckon with that.

We leave it to the reader to picture the complications that arise when one wants to formulate the transformations when V2–movement and damit- or davor–split may operate. Notice also that the order of application of these rules must be reckoned with.

A big difference between V2–movement and damit–split is that the latter is optional and may apply in subordinate clauses, while the former is obligatory and restricted to main clauses.

(6.5.15) `Er hat mich immer davor gewarnt.`
(6.5.16) *`Er mich immer davor gewarnt hat.`
(6.5.17) `Ich konnte einfach nicht damit rechnen.`
(6.5.18) *`Ich damit einfach nicht rechnen konnte.`

In (6.5.16) we have reversed the effect of both transformations of (6.5.13). The sentence is ungrammatical. If we only apply V2–movement, however, we get (6.5.15), which is grammatical. Likewise for (6.5.18) and (6.5.17). In contrast to Harris, Chomsky did not construe transformations as mediating between grammatical sentences (although also Harris did allow to pass through illegitimate structures). He insisted that there is a two layered process of generation of structures. First, a simple grammar (context free, preferrably) generates so called **deep structures**. These deep structures may be seen as the canonical representations, like Polish Notation or infix notation, where the meaning can be read off immediately. However, these structures may not be legitimate objects of the language. For example, at deep structure, the verb of a German sentence appears in final position (where it arguably belongs) but alas these sentences are not grammatical as main clauses. Hence, transformations must apply. Some of them apply optionally, for example damit- and davor–split, some obligatorily, for example V2–movement. At the end of the transformational cycle stands the **surface structure**. The second process is also called (somewhat ambiguously) **derivation**. The split between these to processes has its advantages, as can be seen in the case of German. For if we assume that the main clause is not the deep structure, but derived from a deep structure that looks like a surface subordinate clause, the entire process for generating German sentences is greatly simplified. Some have even proposed that all languages have universally the same deep structure, namely SVO in (Kayne, 1994) or SVO/SOV in (Haider, 1992) and (Haider, 1994). Since the overwhelming majority of languages belongs to either of these types, such claims are not without justification. The differences that can be observed in languages are then caused not by the first process, generating the deep structure, but entirely by the second, the transformational component. However, as might be immediately clear, this is on the one hand theoretically possible but on the other hand difficult to verify empirically. Let us look at a problem. In German, the order of nominal constituents is free (within bounds).

(6.5.19) `Der Vater schenkt dem Sohn einen Hund.`
(6.5.20) `Einen Hund schenkt der Vater dem Sohn.`
(6.5.21) `Dem Sohn schenkt der Vater einen Hund.`
(6.5.22) `Dem Sohn schenkt einen Hund der Vater.`
The father gives a dog to the son.

How can we decide which of the serializations are generated at deep structure and which ones aren't? (It is of course conceivable that all of them are deep structure serializations and even that none of them is.) This question has not found a satisfactory answer to date. The problem is what to choose as a diagnostic tool to identify the deep structure serialization. In the beginning of transformational grammar it was thought that the meaning of a sentence is assigned at deep structure. The transformations are not meaning related, they only serve to make the structure 'speakable'. This is reminiscent of Harris' idea that transformations leave the meaning invariant, the only difference being that Harris' conceived of transformations as mediating between sentences of the language. Now, if we assume this then different meanings in the sentences suffice to establish that the deep structures of the corresponding sentences are different, though we are still at a loss to say which sentence has which deep structure. Later, however, the original position was given up (on evidence that surface structure did contribute to the meaning in the way that deep structure did) and a third level was introduced, the so called **Logical Form** (**LF**), which was derived from surface structure by means of further transformations. We shall not go into this, however. Suffice it to say that this increased even more the difficulty in establishing with precision the deep structure from which a given sentence originates.

Let us return to the sentences (6.5.19) – (6.5.22). They are certainly not identical. (6.5.19) sounds more neutral, (6.5.21) and (6.5.20) are somewhat marked, and (6.5.22) finally is somewhat unusual. The sentences also go together with different stress patterns, which increases the problem here somewhat. However, these differences are not exactly semantical, and indeed it is hard to say what they consist in.

Transformational grammar is very powerful. Every recursively enumerable language can be generated by a relatively simple TG. This has been shown by Stanley Peters and R. Ritchie (see (1971; 1973)). In the exercises the reader is asked to prove a variant of these theorems. The transformations that we have given above are problematic for a simple reason. The place from which material has been moved is lost. The new structure is actually not distinguishable from the old one. Of course, often we can know what the previous structure was, but only when we know which transformation has been applied. However, it has been observed that the place from which an element has been moved, influences the behaviour of the structure. For example, Chomsky has argued that `want to` can be contracted to `wanna` in American English; however, this happens only if between `want` and `to` there has been no intervening element during the course of derivation. For example, contraction is permitted in (6.5.24), in (6.5.26) however it is not, since `the man` was the subject of the lower infinitive (standing to the left of the verb), and had been raised from there.

(6.5.23) `We want to leave.`
(6.5.24) `We wanna leave.`
(6.5.25) `This is the man we want to leave us alone.`
(6.5.26) `*This is the man we wanna leave us alone.`

The problem is that the surface structure does not know that the element `the man` has once been in between `want` and `to`. Therefore, one has assumed that the moved element leaves behind a so called **trace**, written t. For other reasons the trace also got an index, which is a natural number, the same one that is given to the moved element (= antecedent of the trace). So, (6.2) 'really' looks like this.

(6.27) `Trains`$_1$`, Harry likes` t_1.

We have chosen the index 1 but any other would have done equally well. The indices as well as the t are not audible, and they are not written either (except in linguistic textbooks, of course). Now the surface structure contains traces and is therefore markedly different from what we actually hear or read. Whence one assumed — finally — that there is a further process turning a surface structure into a pronounceable structure, the so called **Phonological Form** (**PF**). PF is nothing but the phonological representation of the sentence. On PF there are no traces and no indices, no (or hardly any) constituent brackets.

One of the most important arguments in favour of traces and the instrument of coindexing was the distribution of pronouns. In the theory one distinguishes **referential expressions** (like `Harry` or `the train`) from **anaphors**. To the latter belong pronouns (`I`, `you`, `we`) as well as reflexive pronouns (`oneself`). The distribution of these three is subject to certain rules which are regulated in part by structural criteria.

(6.5.28) `Harry likes himself.`
(6.5.29) `Harry believed that John was responsible for himself.`
(6.5.30) `Harry believed to be responsible for himself.`

`Himself` is a subject–oriented anaphor. Where it appears, it refers to the subject of the same sentence. Semantically, it is interpreted as a variable, which is equal to the variable of the subject. As (6.5.29) shows, the domain of a reflexive ends with the finite sentence. The antecedent of `himself` must be taken to be John, not Harry. Otherwise, we would have to have `him` in place of `himself`. (6.5.30) shows that sometimes also phonetically empty pronouns can appear. In other languages they are far more frequent (for example in Latin or in Italian). Subject pronouns may often be omitted. One says that these languages have an empty pronoun, called pro ('little Pro'). Additionally to the question of the subject also structural factors are involved.

(6.5.31) `Heinrich was driving with his car and Peter too.`
(6.5.32) `Peter handed Karin her coat and Mary, too.`

We may understand (6.5.31) in two ways: either Peter was driving with his (= Peter's) car or with Heinrich's car. (6.5.32) allows only one reading: Peter gave Mary her own coat, not Karin's. This has arguably nothing to do with semantical factors, but only with the fact that in the first sentence, but not in the second, the pronoun is bound by its antecedent. Binding is defined as follows.

Definition 6.5.1 *Let $\mathfrak{T}$ be a tree with labelling ℓ. x **binds** y if (a) the smallest branching constituent that properly contains the x also contains y, but the smallest constituent containing x does not contain y, (b) x and y carry the same index.*

The structural condition (a) of the definition is called **c–command**. (A somewhat modified definition is found below.) The antedecent c–commands the pronoun in case of binding. In (6.5.31) the pronoun `his` is c–commanded by `Heinrich`. For the smallest constituent properly containing `Heinrich` is the entire sentence. In (6.5.32) the pronoun `her` is not c–commanded by `Karin`. (This is of course not entirely clear and must be argued for independently.)

There is a rule of distribution for pronouns that is as follows: the reflexive pronoun has to bound by the subject of the sentence. A non reflexive pronouns however may not be bound by the subject of the sentence. This applies to German as well as to English. Let us look as (6.5.33).

(6.5.33) `Himself Harry likes.`

If this sentence is grammatical, then binding is computed not only at surface structure but at some other level. For the pronoun `himself` is not c–commanded by the subject `Harry`. The structure that is being assumed is [`Himself` [`Harry likes`]]. Such consideration have played a role in the introduction of traces. Notice however that none of the arguments is inevitable. It only forces us to certain moves within a theory (because it makes certain assumptions). It has to be said though that binding was *the* central diagnostic tools of transformational grammar. Always if it was diagnosed that there was no c–command relation between an anaphor and some element one has concluded that some movement must have taken place from a position, where c–command still applied.

In the course of time the concept of transformation has undergone revision. TG allowed deletions only if they were recoverable: this means that if one has the output structure and the name of the transformation that has been applied one can reconstruct the input structure. (Effectively, this means that the transformations are partial injective functions.) In the so called **Theory of Government and Binding** (**GB**) Chomsky has banned deletion altogether from the list of options. The only admissible transformation was movement, which was later understood as copy and delete (which in effect had the same result but was theoretically a bit more elegant). The movement transformation was called **Move**–α and allowed to move any element anywhere (if only the landing site had the correct syntactic label). Everything else was regulated by conditions on the admissibility of structures.

Quite an interesting complication arose in the form of the so called **parasitic gaps**.

(6.5.34) `Which papers did you file without reading?`

We are dealing here with two verbs, which share the same direct object (`to file` and `to read`). However, at deep structure only one them could have had the overt object phrase `which papers` as its object and so at deep structure we either had something like (6.5.35) or something like (6.5.36).

(6.5.35) `you did file which papers without reading?`
(6.5.36) `you did file without reading which papers?`

One assumed that essentially (6.5.35) was the deep structure while the verb `to read` (in its form `reading`, of course) just got an empty coindexed object.

(6.5.37) `you did file which papers`$_1$ `without reading` e_1`?`

However, the empty element is not bound in this configuration. English does not allow such structures. The transformation that moves the wh–constituent at the beginning of the sentence however sees to it that a surface structure the pronoun is bound. This means that binding is not something that is decided at deep structure alone but also at surface structure. However, it cannot be one of the levels alone (see (Frey, 1993)). We have just come to see that deep structure alone gives the wrong result. If one replaces `which papers` by `which paper about yourself` then we have an example in which the binding conditions apply neither exclusively at deep structure nor exclusively at surface structure. And the example shows that traces form an integral part of the theory.

A plethora of problems have since appeared that challenged the view and the theory had to be revised over and over again in order to cope with them. One problem area were the quantifiers and their scope. In German, quantifiers have scope more or less as in the surface structure, while in English matters are different (not to mention other languages here). Another problem is coordination. In a coordinative construction we may intuitively speaking delete elements. However, deletion is not an option any more. So, one has to assume that the second conjunct contains empty elements, whose distribution must be explained. The deep structure of (6.5.38) is for example (6.5.39). For many reasons, (6.5.40) or (6.5.41) would however be more desirable.

(6.5.38) `Karl hat Maria ein Fahrrad gestohlen und Peter ein Radio.`
Karl has Maria a bicycle stolen and Peter a radio.
Karl has stolen a bicycle from Maria and a radio from Peter.

(6.5.39) $\texttt{Karl}_1$ `Maria ein Fahrrad` [`gestohlen hat`]$_2$
`und` e_1 `Peter ein Radio` e_2`.`

(6.5.40) `Karl` [[`Maria ein Fahrrad`] `und` [`Peter ein Auto`]] `gestohlen hat.`

(6.5.41) `Karl` [[`Maria ein Fahrrad gestohlen hat`]
`und` [`Peter ein Radio gestohlen hat.`]]

We shall conclude this section with a short description of GB. It is perhaps not an overstatement to say that GB has been the most popular variant of transformational grammar, so that it is perhaps most fruitful to look at this theory rather than previous ones (or even the subsequent **Minimalist Program**). GB was divided into several subtheories, so called **modules**. Each of the modules was responsible for its particular set of phenomena. There was

1. Binding Theory,
2. the ECP (Empty Category Principle),
3. Control Theory,
4. Bounding Theory,
5. the Theory of Government,
6. Case Theory,
7. Θ–Theory,
8. Projection Theory.

The following four levels of representation were distinguished.

1. D–Structure (formerly *deep structure*),
2. S–Structure (formerly *surface structure*),
3. Phonetic Form (PF) and

4. Logical Form (LF).

There was only one transformation, called **Move**–α. It takes a constituent of category α (α arbitrary) and moves it to another place either by putting it in place of an empty constituent of category α (substitution) or by adjoining it to a constituent. Binding Theory however requires that trace always have to be bound, and so movement always is into a position c–commanding the trace. Substitution was defined as follows. (Here X and Y are variables for strings α and γ category symbols. i is a variable for a natural number. It is part of the representation (more exactly, it is part of the label, which we may construe as a pair of a category symbol and a set of natural numbers). i may occur in the left hand side (SC) namey, if it figures in the label α. So, if $\alpha = \langle C, I\rangle$, C a category label and $I \subseteq \omega$, $\alpha \oplus i := \langle C, I \cup \{i\}\rangle$.)

$$\text{Substitution:} \qquad [X[[e]_\alpha Y[Z]_\alpha W]]] \Longrightarrow [X[[Z]_{\alpha\oplus i} Y[t_i]_\alpha W]]]$$

Adjunction is the following transformation.

$$\text{Adjunction:} \qquad [X[Y]_\alpha Z]_\gamma \Longrightarrow [[Y]_{\alpha\oplus i}[X[t_i]_\alpha Z]_\gamma]_\gamma$$

In both cases the constituent on the right hand side, $[X]_{\alpha\oplus i}$, is called the **antecedent** of the trace, t_i. This terminology is not arbitrary: traces in GB are considered as anaphoric elements. In what is to follow we shall not consider any adjunction since it leads to complications that go beyond the scope of this exposition. For details we refer to (Kracht, 1998). For the understanding of the basic techniques (in particular with respect to Section 6.7) it is enough if we look at substitution.

As in TG first the D–structure is generated. How this is done is not exactly clear. Chomsky assumes in (1981) that it is freely generated and then checked for conformity with the principles. Subsequently, the movement transformation operates until the conditions for an S–structure are satisfied. Then a copy of the structure is passed on to the component which transforms it into a PF. (PF is only a level of representation, therefore there must be a process to arrive at PF.) For example, symbols like t_i, e, which are empty, are deleted together with all or part of the constituent brackets. The original structure meanwhile is subjected to another transformational process until it has reached the conditions of Logical Form and is directly interpretable semantically. Quantifiers appear in their correct scope at LF. This model is also known as the **T–model**.

We begin with the phrase structure, which is conditioned by the theory of projection. The conditions of theory of projection must in fact be obeyed at all levels (with the exception of PF). This theory is also known as $\overline{X}$–**syntax**. It differentiates between simply categorial labels (for example V, N, A, P, I and C, to name the most important ones) and a level of projection. The categorial labels are either **lexical** or **functional**. Levels of projection are natural numbers, starting with 0. The higher the number the higher the level. In the most popular version one distinguishes exactly 3 levels for all categories (while in (Jackendoff, 1977) it was originally possible to specify the numbers of levels for each category independently). The levels are added to the categorial label as superscripts. So N^2 is synonymous with

$$\begin{bmatrix} \textsc{kat} & : & \mathrm{N} \\ \textsc{proj} & : & 2 \end{bmatrix}$$

If X is a categorial symbol, then XP is the highest projection. In our case NP is synonymous with N^2. The rules are at most binary branching. The non branching rules are

$$X^{j+1} \to X^j.$$

X^j is the **head of** X^{j+1}. There are, furthermore, the following rules:

$$X^{j+1} \to X^j \quad \mathrm{YP}, \qquad X^{j+1} \to \mathrm{YP} \quad X^j\ .$$

Here, YP is called the **complement** of X^j if $j = 0$, and the **specifier** if $j = 1$. Finally, we have these rules.

$$X^j \to X^j \quad \mathrm{YP}, \qquad X^j \to \mathrm{YP} \quad X^j.$$

Here YP is called the **adjunct of** X^j. The last rules create a certain difficulty. We have two occurrences of the symbol X^j. This motivated the distinction between a **category** (connected sets of nodes carrying the same label) and **segments** thereof. The complications that arise from this definition have been widely used by Chomsky in (1986). The relation **head of** is transitive. Hence x with category N^i is the head of y with N^j, if all nodes z with $x < z < y$ have category N^k for some k. By necessity, we must have $i \leq k \leq j$.

Heads possess in addition to their category label also a **subcategorization frame**. This frame determines which arguments the head needs and to which arguments it assigns case and or a θ–role. θ–roles are needed to recover an argument in the semantic representation. For example, there are roles for **agent**, **experiencer**, **theme**, **instrument** and so on. These are coded by suggestive names, such as θ_a, θ_w and so on. `see` gets for example the following subcategorization frame.

$$\texttt{see}: \qquad \langle \mathrm{NP}[\theta_w], \mathrm{NP}[\textsc{acc}, \theta_t] \rangle$$

It is on purpose that the verb does not assign case to its subject. It only assigns a θ–role. The case is assigned only by virtue of the verb getting the finiteness marker. The subcategorization frames dictate how the local structure surrounding a head looks like. One says that the head **licenses** nodes in the deep structure, namely those which correspond to entries of its subcategorization frame. It will additionally determine that certain elements get case and/or θ–roles. Case- and Θ–Theory determine which elements need case/θ–roles and how they can get them from a head. One distinguishes between **internal** and **external arguments**. There is at most one external argument, and it is signalled in the frame by underlining it. It is found at deep structure outside of the maximal projection of the head (some theorists also think that it occupies the specifier of the projection of the head, but the details do not really matter here). Further, only one of the internal arguments is a complement. This is already a consequence of

$\overline{X}$–syntax; the other arguments therefore have to be adjuncts at D–structure.

One of the great successes of the theory was the analysis of `seem`. The uninflected `seem` has the following frame.

$$\texttt{seem}: \qquad \langle \mathrm{INFL}^2[\theta_t] \rangle$$

(INFL is the symbol of inflection. This frame is valid only for the variant which selects infinitives.) This verb has an internal argument, which must be realized by the complement in the syntactic tree. The verb assigns a θ–role to this argument. Once it is inflected, it has a subject position, which is assigned case but no θ–role. A caseless NP inside the complement must be moved into the subject position of `seem` in syntax, since being an NP it needs case. It can only appear in that position, however, if at deep structure it has been assigned a θ–role. The subject of the embedded infinitive however is a canonical choice: it only gets a θ–role, but still needs case.

(6.5.42) $\texttt{Jan}_1$ `seems` [t_1 `to sleep].`
(6.5.43) *`Seems [Jan to sleep].`

It was therefore possible to distinguish two types of intransitive verbs, those which assign a θ–role to their subject (`fall`) and those which do not (`seem`). There were general laws on subcategorization frames, such as

> Burzio's Generalization. A verb assigns case to its governed NP–argument if and only it assigns a θ–role to its external argument.

The Theory of Government was responsible among other for case assignment. It was assumed that nominative and accusative could not be assigned by heads (as we — wrongly, at least according to this theory — said above) but only in a specific configuration. The simplest configuration is that between head and complement. A verb having a direct complement licenses a direct object position. This position is qua structural property (being sister to an element licensing it) assigned accusative. The following is taken from (von Stechow and Sternefeld, 1987), p. 293.

Definition 6.5.2 *x with label α **governs** y with label β if and only if (1) x and y are dominated by the same nodes with label* XP*, X arbitrary, and (2) either $\alpha = X^0$, where X is lexical or $\alpha = \mathrm{AGR}^0$ and (3) x c–commands y. x **governs** y **properly** if x governs y and either $\alpha = X^0$, X lexical, or x and y are coindexed.*

(Since labels are currently construed as pairs $\langle X^i, P\rangle$, where X^i is a category symbol with projection and P a set of natural numbers, we say that x and y are coindexed if the second component of the label of x and the second component of the label of y are not disjoint.) The ECP was responsible for the distribution of empty categories. In GB there was a whole army of different empty categories: e, for a faceless constituent into which one could move, t, for a trace, PRO and pro, which were pronouns. The ECP says among other that t must always be properly governed, while PRO may never be governed. We remark that traces are not allowed to move. In Section 6.7 we consider this restriction more closely. The Bounding Theory concerns itself with the distance that syntactic processes may cover. It (or better: notions of distance) is considered in detail in Section 6.7. Finally, we remark that Transformational Grammar also works with conditions on derivations. Transformations could not be applied in any order but had to follow certain orderings. A very important one (which was the only one to remain in GB) was **cyclicity**. Let y be the antecedent of x after movement and $z \succ y$. Then let the interval $[x, z]$ be called the **domain** of this instance of movement.

Definition 6.5.3 *Let Γ be a set of syntactic categories. x is called a **bounding node**, if the label of x is in Γ. A derivation is called **cyclic** if for any two instances of movement β_1 and β_2 and their domains B_1 and B_2 the following holds: if β_1 was applied before β_2 then every bounding node from B_1 is dominated (not necessarily properly) by some bounding node from B_2 and every bounding node* *from B_2 dominates (not necessarily properly) a bounding node from B_1.*

Principally, all finite sentences are bounding nodes. However, it has been argued by Rizzi (and others following him) that the choice of bounding categories is language dependent.

This exposition may suffice to indicate how complex the theory was. We shall not go into the details of parametrization of grammars and learnability.

Exercise 225. Coordinators like `and`, `or` and `not` have quite a flexible syntax, as was already remarked at the end of Section 3.5. We have `cat and dog`, `read and write`, `green and blue` and so on. What difficulties arise in connection with $\overline{X}$–syntax for these words? What solutions can you propose?

Exercise 226. A transformation is called *minimal* if it replaces at most two adjacent symbols by at most two adjacent symbols. Let L be a recursively enumerable language. Construct a regular grammar G and a finite set of minimal transformations such that the generated set of strings is L. Here the criterion for a derivation to be finished is that no transformation can be applied. *Hint.* If L is recursively enumerable there is a Turing machine which generates L from a given regular set of strings.

Exercise 227. (Continuing the previous exercise.) We additionally require that the deep structure generated by G as well as all intermediate structures conform to $\overline{X}$–syntax.

Exercise 228. Write a 2–LMG that accommodates German V2 and damit– and davor–split.

Exercise 229. It is believed that if traces are allowed to move, we can create unbound traces by movement of traces. Show that this is not a necessary conclusion. However, the ambiguity that arises from allowing such movement on condition that it does not make itself unbound are entirely harmless.

6.6 GPSG and HPSG

In the 80s, several alternatives to transformational grammar were being developed and they also claimed to be universal in the same way. One such development was categorial grammar, which we have discussed earlier in Chapter 3. Others were the grammar formalisms that used a declarative (or model theoretic) definition of syntactic structures. These were the Generalised Phrase Structure Grammar (mentioned already in Section 6.1) as well as the **Lexical–Functional Grammar** (**LFG**). GPSG later developed into the **Head Driven Phrase Structure Grammar** (**HPSG**). In this section we shall deal mainly with GPSG and HPSG. Our aim is twofold. On the one hand we give an overview of the expressive mechanism that is being used in these theories, on the other we show how to translate these expressive devices into a suitable polymodal logic.

In order to justify the introduction of transformational grammar, Chomsky had given several arguments, which were intended to show that traditional theories were completely inadequate. In particular, he targeted the theory of finite automata (which was very popular in the 50s) and the structuralism. His criticism of finite automata is up to now unchallenged. His critique of the structuralism, however, has never been as devastating as he made others believe. First of all, Chomsky has made a caricature of structuralism by equating it with the claim that natural languages are context free (see (Manaster-Ramer and Kac, 1990)). Even if this was not the case, his proofs of the insufficiency of context free grammars are questionable. Some linguists, notably Gerald Gazdar and Geoffrey Pullum, after reviewing these and other proofs eventually came to the conclusion that contrary to what has hitherto been believed all natural languages were context free. However, the work of Riny Huybregts and Stuart Shieber, which we have discussed already in Section 2.6 put a preliminary end to this story. Strangely enough, as James Rogers and Marcus Kracht have later shown, the theories of English proposed inside of GB actually postulated an essentially context free structure for it. Hence English is still (from a theoretical point of view) a strongly context free language (contrary to the views expressed by Chomsky).

An important argument against context free rules has been the fact that simple regularities of language such as agreement cannot be formulated in them. This was one of the main arguments by Paul Postal against the structuralists (and other people), even though strangely enough TG and GB did not have had much to say about it either. Textbooks only offer vague remarks about agreement to the effect that heads agree with their specifiers in certain features. Von Stechow and Sternefeld (1987) are more precise in this respect. In order to formulate this exactly, one needs AVSs and variables for values (and structures). These tools were introduced by GPSG into the apparatus of context free rules, as we have shown in Section 6.1. Since we have seen this already, let us go over to the word order variation. Let us note that GPSG takes over $\overline{X}$–syntax more or less without change. It does, however, not insist on binary branching. Second, GPSG separates the context free rules into two components: one is responsible for generating the dominance relation, the other for the precedence relation between sisters. The following rule determines that a node with label VP can have daughters, which may occur in any order.

$$\mathrm{VP} \to \mathrm{NP}[nom]\ \mathrm{NP}[dat]\ \mathrm{NP}[acc]\ \mathrm{V}\ .$$

This rule stands for no less than 24 different context free rules. In order to get for example the German word order of the subordinate clause we now add the following condition.

$$\mathrm{N} \prec \mathrm{V}\ .$$

This says that every daughter with label N is to the left of any daughter with label V. Hence there only remain 6 context free rules, namely those in which the verb is at the end of the clause. (See in this connection the examples (6.5.19) – (6.5.22)). For German one would however not propose this analysis since it would not allow to posit any adverbials in between the arguments of the verb. If one returns to the binary structure, then one gets back immediately the word order problems in the form of order of discharge (for which GPSG has no special mechanism). There are languages for which this is better suited. For example, Staal (1967) has argued that Sanscrit has the following word orders: SVO, SOV, VOS und OVS. If we allow the following rules without specifying the linear order, these facts are accounted for.

$$\mathrm{VP} \to \mathrm{NP}[nom] \quad \mathrm{V}^1, \qquad \mathrm{V}^1 \to \mathrm{NP}[acc] \quad V^0 .$$

All four possibilities can be generated — and no more.

Even if we ignore word order variation of the kind just described there remain a lot of phenomena that we must account for. GPSG has found a method of capturing the effect of a single movement transformation by means of a special device. It first of all defines **metarules** which generate rules from rules. For movement we do the following. Given a structure $[[\ldots Y \ldots]_W]_V$ we propose to have an additional structure $[Y_i\ [\ldots t_i \ldots]_W]_V$. Further, there shall be a unary rule $V \to Y\ W$. The introduction of these rules can be captured by a general scheme, called as we said a metarule. However, in the particular case at hand we must be a bit more careful. It is necessary to do proper bookkeeping in order not to lose sight of where we had moved from. In analogy to categorial grammar, we introduce a category W/Y and in place of the rule $V \to Y\ W$ the rule $V \to Y\ W/Y$. However, in place of W/Y one writes in GPSG

$$\begin{bmatrix} W & & \\ \textsc{slash} & : & Y \end{bmatrix}$$

This says that what we have is a constituent of category W/Y, which is a constituent of category W missing a Y. How do we see to it that the feature [SLASH : Y] is correctly distributed? Also here GPSG has tried to come up with a principled answer. GPSG distinguishes **foot features** from **head features**. Their behaviour is quite distinct. Every feature is either a foot feature or a head feature. The attribute SLASH is classified as a foot feature.

(It is perhaps unfortunate that it is called a feature and not an attribute, but this is a minor issue.) For a foot feature such as SLASH, the SLASH–features of the mother are the union of the SLASH–features of the daughters. Let us look more closely into that. If W is an AVS and f a feature then we denote by $f(W)$ the value of f in W.

Definition 6.6.1 *Let G be a set of rules over AVSs. f is a* ***foot feature in*** *G if for every maximally instantiated rule $A \to B_0 \ldots B_{n-1}$ the following holds.*

$$f(A) = \bigwedge_{i<n} f(B_i) \ .$$

So, what this says is that the SLASH–feature can be passed on from mother to any number of its daughters. In this way the authors of (Gazdar *et al.*, 1985) have seen to it that parasitic gaps can also be explained (see the previous section for this phenomenon). However, extreme care is needed. For the rules do not allow to count how many constituents of the same category have been extracted, so there might still be room for error. **Head features** are being distributed roughly as follows.

> [Head Feature Convention] If f is a head feature then for every rule $A \to B_0 \ldots B_{n-1}$. $f(A) = f(B_i)$, where B_i is the head of that rule.

The exact formulation of the distribution scheme for head features however is much more complex than for foot features. We shall not go into the details here.

This finishes our short introduction to GPSG. It is immediately clear that the languages generated by GPSG are context free if there are only finitely many category symbols. In order for this to be the case, the syntax of paths in an AVS was severely restricted.

Definition 6.6.2 *Let A be an AV–structure. A* ***path in*** *A is a sequence $\langle f_i : i < n\rangle$ such that $f_{n-1} \circ \ldots \circ f_0(A)$ is defined. The value of this expression is the* ***value*** *of the path.*

In (Gazdar *et al.*, 1985) it was required that only those paths were legitimate in which no attribute occurs twice. In this way the finiteness is a simple matter. The following is left to the reader as an exercise.

Proposition 6.6.3 *Let A be a finite set of attributes and F a finite set of paths over A. Then every set of pairwise non–equivalent AVSs is finite.*

Subsequently to the discovery on the word order of Dutch and Swiss German this restriction finally had to fall. Further, some people have anyway argued that the syntactic structure of the verbal complex is much different, and that this applies also to German. The verbs in a sequence of infinitives were argued to form a constituent, the so called **verb cluster**. This has been claimed in the GB framework for German and Dutch. Also, Bresnan et. al. (1987) argued for a much different analysis. The theory they were proposing was LFG. Central to LFG is the assumption that there are two (nowadays three) distinct structures that are being built simultaneously:

1. **c–structure** or constituent structure: this is the structure where the linear precedence is encoded and also syntactic structure.

2. **f–structure** or functional structure: this is the structure where the grammatical relations (subject, object) but also discourse relations (topic) are encoded.

3. **a–structure** or argument structure: this is the structure that encodes argument relations (θ–roles).

A rule specifies a piece of c–structure together with correspondences between the c–structure and the other structures. For simplicity we shall ignore a–structure from now on. An example is provided by the following grammar.

$$\begin{array}{cccc}
\mathrm{S} & \to & \mathrm{NP} & \mathrm{VP} \\
 & & (\uparrow \textsc{subj} =\downarrow) & \uparrow=\downarrow \\[2ex]
\mathrm{NP} & \to & \mathrm{Det} & \mathrm{N} \\
 & & \uparrow=\downarrow & \uparrow=\downarrow \\[2ex]
\mathrm{VP} & \to & \mathrm{V} & \mathrm{PP} \\
 & & \uparrow=\downarrow & (\uparrow \textsc{obj} =\downarrow)
\end{array}$$

The rules have two lines: the upper line specifies a context free phrase structure rule of the usual kind. The lower line tells us how the c–structure relates to the f–structure. These correspondences will allow to define a unique f–structure (together with the universal rules of language). The rule if applied creates a local tree in the c–structure, consisting of three nodes, say 0, 00, 01, with label S, NP and VP, respectively. The corresponding f–structure is different. This is indicated by the equations. To make the ideas precise, we shall assume two sets of nodes in the universe, C and F, which are the c–structure and f–structure nodes, respectively. And we assume a function FUNC, which maps C to F. It is clear now how to translate context free rules into first–order formulae. We directly turn to the f–structure statements. C–structure is a tree, F–structure is an AVS. Using the function UP to map a node to its mother, the equation $(\uparrow \textsc{subj} =\downarrow)$ is translated as follows:

$$(\uparrow \textsc{subj} =\downarrow)(x) := \textsc{subj} \circ \textsc{func} \circ \textsc{up}(y) \doteq \textsc{func}(x))$$

In simpler terms: I am my mother's subject. The somewhat simpler statement $\uparrow=\downarrow$ is translated

$$(\uparrow=\downarrow)(x) := \textsc{func} \circ \textsc{up}(x) \doteq \textsc{func}(x)$$

Thus the f–structure has only two nodes, since the predicate installs itself into the root node (by the second condition), while the subject NP is its SUBJ–value. Notice that the statements are local path equations, which are required to hold of the c–structure node under which they occur. LFG uses the fact that f–structure is flatter than c–structure to derive the Dutch and Swiss German sentences using rules of this kind, despite the fact that the c–structures are not context free.

While GPSG and LFG still assume a phrase structure skeleton that plays an independent role in the theory, HPSG actually offers a completely homogeneous theory that makes no distinction between the sources from which a structure is constrained. What made this possible is the insight that the attribute value formalism can also encode structure. A very simple possibility of taking care of the structure is the following. Already in GPSG there was a feature SUBCAT whose value was the subcategorization frame of the head. Since the subcategorization frame must map into a structure we require that in the rules

$$X \to Y[\text{SUBCAT} : A] \quad B$$

$B \to A$ must be universally valid. (Notice that the order does not play any role.) This means nothing but that B is being subsumed under A that is to say that it is a special A. The difference with GPSG is now that we allow to stack the feature SUBCAT arbitrarily deep. For example, we can attribute to the German word `geben` ('to give') the following category.

$$\begin{bmatrix} \text{CAT} & : & v \\ \text{SUBCAT} & : & \begin{bmatrix} \text{CAT} & : & \text{NP} \\ \text{CASE} & : & nom \\ \text{SUBCAT} & : & \begin{bmatrix} \text{CAT} & : & \text{NP} \\ \text{CASE} & : & dat \\ \text{SUBCAT} & : & \begin{bmatrix} \text{CAT} & : & \text{NP} \\ \text{CASE} & : & acc \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The rules of combination for category symbols have to be adapted accordingly. This requires some effort but is possible without problems. HPSG essentially follows this line, however pushing the use of AVSs to the limit. Not only the categories, also the entire geometrical structure is now coded using AVSs. HPSG also uses structure variables. This is necessary in particular for the semantics, which HPSG treats in the same way as syntax. (In this it differs from GPSG. The latter uses a Montagovian approach, pairing syntactic rules with semantical rules. In HPSG, the semantics is coded up like syntax.) Parallel to the development of GPSG and related frameworks, the so called *constraint based approaches* to natural language processing were introduced. (Shieber, 1992) provides a good reference.

Definition 6.6.4 *A **basic constraint language** is a finite set F of unary function symbols. A **constraint** is either an equation $s(x) \doteq t(x)$ or a statement $s(x)\uparrow$. A **constraint model** is a partial algebra $\mathfrak{A} = \langle A, \Pi\rangle$ for the signature. We write $\mathfrak{A} \vDash s \doteq t$ if and only if for every a, $s^{\mathfrak{A}}(a)$ and $t^{\mathfrak{A}}(a)$ are defined and equal. $\mathfrak{A} \vDash s(x)\uparrow$ if and only if for every a, $s^{\mathfrak{A}}(a)$ is defined.*

Often, one has a particular constant $\underline{o}$, which serves as the root, and one considers equations of the form $s(\underline{o}) \doteq t(\underline{o})$. Whereas the latter type of equation holds only at the root, the above type of equations are required to hold *globally*. We shall deal only with globally valid equations. Notice that we can encode atomic values into this language by interpreting an atom p as a unary function f_p with the idea being that f_p is defined at b if and only if p holds of b. (Of course, we wish to have $f_p(f_p(b)) = f_p(b)$ if the latter is defined, but we need not require that.) We give a straightforward interpretation of this language into modal logic. For each $f \in F$, take a modality, which we call by the same name. Every f satisfies $\langle f\rangle p \wedge \langle f\rangle q. \rightarrow .\langle f\rangle(p \wedge q)$. (This logic is known as $\mathsf{K.alt}_1$, see (Kracht, 1995).) With each term t we associate a modality in the obvious way: $f^\mu := f$, $(f(t))^\mu := f; t^\mu$. Now, the formula $s \doteq t$ is translated by

$$\begin{array}{lcl} (s^\mu \doteq t^\mu) & := & \langle s^\mu\rangle p \leftrightarrow \langle t^\mu\rangle p \\ (\uparrow s)^\mu & := & \langle s^\mu\rangle\top \end{array}$$

Finally, given $\mathfrak{A} = \langle A, \Pi \rangle$ we define a Kripke–frame $\mathfrak{A}^\mu := \langle A, R \rangle$ with $x\ R(f)\ y$ if and only if $f(x)$ is defined and equals y. Then

$$\mathfrak{A} \vDash s \doteq t \quad \Leftrightarrow \quad \mathfrak{A}^\mu \vDash (s \doteq t)^\mu$$

Further,

$$\mathfrak{A} \vDash (\uparrow s) \quad \Leftrightarrow \quad \mathfrak{A}^\mu \vDash (\uparrow s)^\mu$$

Now, this language of constraints has been extended in various ways. The attribute–value structures of Section 6.1 effectively extend this language by boolean connectives. $[C\ :\ A]$ is a shorthand for $\langle C \rangle A^\mu$, where A^μ is the modal formula associated with A. Moreover, following the discussion of Section 6.4 we use $\Diamond_{\leftarrow}$, $\Diamond_{\rightarrow}$, $\Diamond_{\uparrow}$ and $\Diamond_{\downarrow}$ to steer around in the phrase structure skeleton. HPSG uses a different encoding. It assumes an attribute called DAUGHTERS, whose value is a list. A list in turn is an AVS which is built recursively using the predicates FIRST and REST. (The reader may write down the path language for lists.) The notions of Kripke–structure and generalized Kripke–structure are then defined as usual. The Kripke–structures take the role of the actual syntactic objects, while the AVSs are simply formulae to talk about them.

The logic L_0 is of course not very interesting. What we want to have is a theory of the existing objects, not just all conceivable ones. A particular concern in syntactical theory is therefore the formulation of an adequate theory of the linguistic objects, or be it a universal theory of all linguistic objects, be it a theory of the linguistic objects of a particular language. We may cast this in logical terms in the following way. We start with a set (or class) $\mathcal{K}$ of Kripke–structures. The theory of that class is $\mathsf{Th}\,\mathcal{K}$. It would be most preferrably if for any given Kripke–frame $\mathfrak{F}$ we had $\mathfrak{F} \vDash \mathsf{Th}\,\mathcal{K}$ if and only if $\mathfrak{F} \in \mathcal{K}$. Unfortunately, this is not always the case. We shall see, however, that the situation is as good as one can hope for. Notice the implications of the setup. Given, say, the admissible structure of English, we get a modal logic $L_M(\text{Eng})$, which is an extension of L_0. Moreover, if $L_M(\text{Univ})$ is the modal logic of all existing linguistic objects, then $L_M(\text{Eng})$ furthermore is an axiomatic extension of $L_M(\text{Univ})$. There are sets Γ and E of formulae such that

$$L_0 \subseteq L_M(\text{Univ}) = L_0 \oplus \Gamma \subseteq_M (\text{Eng}) = L_0 \oplus \Gamma \oplus E$$

If we want to know, for example, whether a particular formula φ is satisfiable in a structure of English it is not enough to test it against the postulates of the logic L_0, nor for those of $L_M(\text{Univ})$. Rather, we must show that it is consistent with $L_M(\text{Eng})$. These problems can have very different complexity. While L_0 is decidable, this need not be the case for $L_M(\text{Eng})$ nor for $L_M(\text{Univ})$. The reason is that in order to know whether there is a structure for a logic that satisfies the axioms we must first guess that structure before we can check the axioms on it. If we have no indication of its size, this can turn out to be impossible. The exercises shall provide some examples. Another way to see that there is a problem is this. φ is a theorem of $L_M(\text{Eng})$ if it can be derived from $L_0 \cup \Gamma \cup E$ using modus ponens (MP), substitution and (MN). However, $\Gamma \cup E \Vdash_{L_0} \varphi$ if and only if φ can be derived from $L_0 \cup \Gamma \cup E$ using (MP) and (MN) alone. Substitution, however, is very powerful. Here we shall be concerned with the difference in expressive power of the basic constraint language and the modal logic. The basic constraint language allows to express that two terms (called paths for obvious reasons) are identical. There are two ways in which such an identity can be enforced. (a) As an axiom: then it must hold of all structures under consideration. An example is provided by the agreement rules of a language. (b) As a datum: then we are asked to satisfy the equation in a particular structure. In modal logic, only equations as axioms are expressible. Except for trivial cases there is no formula $\varphi(s, t)$ in polymodal $\mathsf{K.alt}_1$ such that

$$\langle \mathfrak{A}^\mu, \beta, x \rangle \vDash \varphi \Leftrightarrow s^{\mathfrak{A}}(x) \doteq t^{\mathfrak{A}}(x)$$

Hence, modal logic is expressibly weaker than predicate logic, in which such a condition is easily expressible. Yet, it is not clear that such conditions are at all needed in natural language. All that is needed is to be able to state conditions of that kind on all structures — which we can in fact do. (See (Kracht, 1995) for an extensive discussion.)

HPSG also uses *types*. Types are properties of nodes. As such, they can be modelled by unary predicates in MSO, or by boolean constants in modal logic. For example, we have represented the atomic values by proposition constants. In GPSG, the atomic values were assigned only to type 0 features. HPSG goes further than that by typing any AVS. Since the AVS is interpreted in a Kripke–frame, this creates no additional difficulty. Reentrancy is modelled by path equations in constraint languages, and can of course be naturally expressed using modal languages, as we have seen. As an example, we consider the agreement rule of Section 6.1 again.

$$(\ddagger) \qquad \begin{bmatrix} \text{CAT} & : & s \end{bmatrix} \to \begin{bmatrix} \text{CAT} & : & np \\ \text{AGR} & : & \boxed{1} \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & : & vp \\ \text{AGRS} & : & \boxed{1} \end{bmatrix}$$

In the earlier (Pollard and Sag, 1987), the idea of reentrancy was motivated by *information sharing*. What the label $\boxed{1}$ says is that any information available under that node in one occurrence is available at any other occurrence. One way to make this true is to simply say that the two occurrences of $\boxed{1}$ are not distinct in the structure. (An analogy might hep here. In the notation $\{a, \{a, b\}\}$ the two occurrences of a do not stand for different things of the universe: they both denote a, just that the linear notation forces us to write it down twice.) There is a way to enforce this in modal logic. Consider the following formula.

$$\begin{aligned} &\langle \mathsf{cat}\rangle \mathsf{np} \wedge \Diamond_{\downarrow}(\Diamond_{\leftarrow}\bot \wedge \langle \text{CAT}\rangle \mathsf{np} \wedge \Diamond_{\rightarrow}(\langle \text{CAT}\rangle \mathsf{vp} \wedge \Diamond_{\rightarrow}\bot)) \\ &\qquad \to \Diamond_{\downarrow}(\Diamond_{\leftarrow}\bot \wedge \langle \text{AGR}\rangle p \leftrightarrow \Diamond_{\rightarrow}\langle \text{AGRS}\rangle p) \end{aligned}$$

This formula says that if we have an S which consists of an NP and a VP, then whatever is the value of AGR of the NP also is the value of AGRS of the VP.

The constituency structure that the rules specify can be written down using quantified modal logic. As an exercise further down shows, QML is so powerful that the codability of any syntactic fact is not surprising. Recall from Section 1.1 the definition of ZFC. As said, it is a first–order theory. In $\mathsf{MSO}(\in, \doteq)$ one can write down an axiom that forces sets to be well–founded with respect to $\in$ and even write down the axioms of NBG (von Neumann–Gödel–Bernays Set Theory), which differs from ZFC in having a simpler scheme for set comprehension. In its place we have this axiom.

> Class Comprehension. $(\forall P)(\forall x)(\exists y)(\forall z)(z \in y \leftrightarrow z \in x \wedge P(z))$.

It says that from a set x and an arbitrary subset of the universe P (which does not have to be a set) there is a *set* of all things that belong to both x and P. In presence of the results by Thatcher, Donner and Wright all this may sound paradoxical. However the introduction of structure variables has made the structures into acyclic graphs rather than trees. However, our reformulation of HPSG is not expressed in QML but in the much weaker polymodal logic. Thus, theories of linguistic objects are extensions of polymodal K. However, as (Kracht, 2001a) shows, by introducing enough modalities one can axiomatize a logic such that a Kripke–frame $\langle F, R\rangle$ is a frame for this logic if and only if $\langle F, R(\in)\rangle$ is a model of NGB. This means that effectively any higher logic can be encoded into HPSG notation, since it is reducible to set theory, and thereby to polymodal logic. Although this is not per se an argument against using the notation, it shows that anything goes and that a claim to the effect that such and such phenomenon can be accounted for in HPSG is empirically vacuous.

Notes on this Section. One of the seminal works in GPSG besides (Gazdar *et al.*, 1985) is the study of word order in German by Hans Uszkoreit (1987). The constituent structure of the continental Germanic languages has been a focus of considerable debate between the different grammatical frameworks. The discovery of Swiss German actually put an end to the debate whether or not context free rules are appropriate. In GSPG it is assumed that the dominance and the precedence relations are specified separately.

Rules contain a dominance skeleton and a specification that says which of the orderings is admissible. However, as Almerindo Ojeda has shown in (1988) GPSG can also generate cross serial dependencies of the Swiss German type. One only has to relax the requirement that the daughters of a node must be linearly ordered to a requirement that the yield of the tree must be so ordered.

Exercise 230. Show that all axioms of ZFC and also $\mathsf{Class\ Comprehension}$ are expressible in $\mathsf{MSO}(\varepsilon, \doteq)$.

Exercise 231. Show that the logic L_0 of any number of basic modal operators satisfying $\Diamond p \wedge \Diamond q \to \Diamond(p \wedge q)$ is decidable. This shows the decidability of L_0. *Hint.* Show that any formula is equivalent to a disjunction of conjunctions of statements of the form $\langle\delta\rangle\pi$, where δ is a sequence of modalities and π is either nonmodal or of the form $[m]\bot$.

Exercise 232. Write a grammar using LFG–rules of the kind described above to generate the crossing dependencies of Swiss German.

Exercise 233. Let A be an alphabet, T a Turing machine over A. The computation of T can be coded onto a grid of numbers $\mathbb{Z} \times \mathbb{N}$. Take this grid to be a Kripke–structure, with basic relations the immediate horizontal successor and predecessor, the transitive closure of these relations, and the vertical successor. Take constants c_a for every $a \in A \cup Q \cup \nabla$. c_∇ codes the position of the read write head. Now formulate an axiom φ_T such that a Kripke–structure satisfies φ_T if and only if it represents a computation of T.

6.7 Formal Structures of GB

We shall close this chapter with a survey of the basic mathematical constructs of GB. The first complex is the constraining of syntactic structures. GB has many types of conditions on syntactic structures. It has for example many principles that describe the geometrical configuration within which an element can operate. A central definition is that of c–command.

Definition 6.7.1 *Let $\mathfrak{T} = \langle T, < \rangle$ be a tree, $x, y \in T$. x* ***c–commands*** *y if the following holds: for every $z > x$ we have $z \geq y$. A constituent $\downarrow x$* ***c–commands*** *a constituent $\downarrow y$ if x c–commands y.*

In (1986), Jan Koster has proposed an attempt to formulate GB without the use of movement transformations. The basic idea was that the traces in the surface structure leave enough indications of the deep structure that we can replace talk of deep structure and derivations by talk about the surface structure alone. The general principle that he proposed was as follows. Let x be a node with label δ, and let δ be a so called dependent element. (Dependency is probably defined with reference to the category.) Then there must exist a uniquely defined node y with label α which c–commands x, and is local to x. Koster required in addition that α and δ shared a property. However, in formulating this condition it turns out to be easier to constrain the possible choices of δ and α. In addition to the parameters α and δ it remains to say what locality is. Anticipating our definitions somewhat we shall say that we have $x \ R \ y$ for a certain relation R. Barker and Pullum have surveyed the notions of locality that enter in the definition of R that were used in the literature and given a definition of command relation in (Barker and Pullum, 1990). Using this, Kracht (1993) developed a mathematical theory of command relations that we shall outline here.

Definition 6.7.2 *Let $\langle T, < \rangle$ be a tree and $R \subseteq T^2$ a relation. R is called a* ***command relation*** *if there is a function $f_R : T \to T$ such that (1) – (3) hold. R is called a* ***monotone command relation*** *if in addition it satisfies (4), and it is called* ***tight*** *if it satisfies (1) – (5).*

1. *$R_x := \{y : x \ R \ y\} = \downarrow f_R(x)$.*
2. *$x < f_R(x)$ for all $x < r$.*

3. $f_R(r) = r$.

4. *If* $x \leq y$ *then* $f_R(x) \leq f_R(y)$.

5. *If* $x < f_R(y)$ *then* $f_R(x) \leq f_R(y)$.

The first class that we shall study is the class of tight command relations. Let $\mathfrak{T}$ be a tree and $P \subseteq T$. We say, x **P–commands** y if for every $z > x$ with $z \in P$ we have $z \geq y$. We denote the relation of P–command by $K(P)$. If we choose $P = T$ we exactly get c–command. The following theorem is left as an exercise.

Proposition 6.7.3 *Let* R *be a binary relation on the tree* $\langle T, < \rangle$. R *is a tight command relation if and only if* $R = K(P)$ *for some* $P \subseteq T$.

Let $\mathfrak{T}$ be a tree. We denote by $\mathrm{MCr}(\mathfrak{T})$ the set of monotone command relations on $\mathfrak{T}$. This set is closed under intersection, union and relation composition. We even have

$$\begin{array}{rcl} f_{R\cup S}(x) & = & \max\{f_R(x), f_S(x)\}, \\ f_{R\cap S}(x) & = & \min\{f_R(x), f_S(x)\}, \\ f_{R\circ S}(x) & = & (f_S \circ f_R)(x). \end{array}$$

For we have $(R \cup S)_x = R_x \cup S_x = \downarrow f_R(x) \cup \downarrow f_S(x) = \downarrow(\max\{f_R(x), f_S(x)\})$, and likewise for intersection. For relation composition we need the monotony. For if $x\, R \circ S\, y$ we can conclude that $x\, R\, f_R(x)$ and $f_R(x)\, S\, y$. Hence $x\, R \circ S\, y$ if and only if $y \leq f_S(f_R(x))$, from which the claim now follows. Now we set

$$\mathfrak{MCr}(\mathfrak{T}) = \langle \mathrm{MCr}(\mathfrak{T}), \cap, \cup, \circ \rangle\ .$$

$\mathfrak{MCr}(\mathfrak{T})$ is a distributive lattice with respect to $\cap$ and $\cup$. What is more, there are additional laws of distribution concerning relation composition.

Proposition 6.7.4 *Let* R, S, T *be from* $\mathrm{MCr}(\mathfrak{T})$. *Then*

1. $R \circ (S \cap T) = (R \circ S) \cap (R \circ T)$, $\quad (S \cap T) \circ R = (S \circ R) \cap (T \circ R)$.
2. $R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$, $\quad (S \cup T) \circ R = (S \circ R) \cup (T \circ R)$.

Proof. Let R, S and T be monotone command relations and x an element of the tree. Then

$$\begin{array}{rcl} f_{R\circ(S\cap T)}(x) & = & f_{S\cap T} \circ f_R(x) \\ & = & \min\{f_S(f_R(x)), f_T(f_R(x))\} \\ & = & \min\{f_{R\circ S}(x), f_{R\circ T}(x)\} \\ & = & f_{(R\circ S)\cap(R\circ T)}(x) \end{array}$$

The other claims can be shown analogously. □

Definition 6.7.5 *Let $\mathfrak{T}$ be a tree, $R \in \mathrm{MCr}(\mathfrak{T})$. R is called **generated** if R can be produced from tight command relations by means of $\cap$, $\cup$ and $\circ$. R is called **chain like** if it can be generated from tight relations with $\circ$ alone.*

Theorem 6.7.6 *R is generated if and only if R is an intersection of chain line command relations.*

Proof. Because of Proposition 6.7.4 we can move $\circ$ to the inside of $\cap$ and $\cup$. Furthermore, we can move $\cap$ outside of the scope of $\cup$. It remains to be shown that the intersection of two chain like command relations is an intersection of chain like command relations. This follows from Lemma 6.7.10. □

Lemma 6.7.7 *Let $R = K(P)$ and $S = K(Q)$ be tight. Then*

$$R \cup S = (R \circ S) \cap (S \circ R) \cap K(P \cap Q) \, .$$

Proof. Let x be given. We look at $f_R(x)$ and $f_S(x)$. Case 1. $f_R(x) < f_S(x)$. Then $f_{R\cup S}(x) = f_S(x)$. On the right hand side we have $f_S \circ f_R(x) = f_S(x)$, since S is tight. $f_R \circ f_S(x) \geq f_S(x)$, as well as $f_{K(P\cap Q)}(x) \geq f_S(x)$. Case 2. $f_S(x) < f_R(x)$. Analogously.

Case 3. $f_S(x) = f_R(x)$. Then $f_{S\cup R}(x) = f_R(x) = f_S(x)$, whence $f_R \circ f_S(x), f_S \circ f_R(x) \geq f_{S\cup R}(x)$. The smallest node above x which is both in P and in Q is clearly in $f_S(x)$. Hence we have $f_{K(P\cap Q)}(x) = f_S(x)$. Hence equality holds in all cases. □

We put

$$K(P) \bullet K(Q) := K(P \cap P)$$

The operation $\bullet$ is defined only on tight command relations. If $\langle R_i : i < m\rangle$ is a sequence of command relations, then $R_0 \circ R_1 \circ \ldots \circ R_{n-1}$ is called its **product**. In what is to follow we shall characterize a union of chain like relations as the intersection of products. To this end we need some definitions. The first is that of a **shuffling**. This operation mixes two sequences in such a way that the liner order inside the sequences is respected.

Definition 6.7.8 *Let $\rho = \langle a_i : i < m\rangle$ and $\sigma = \langle b_j : j < n\rangle$ be sequences of objects. A **shuffling** of ρ and σ is a sequence $\langle c_k : k < m+n\rangle$ such that there are injective monotone functions $f : n \to m+n$ and $g : m \to m+n$ such that $im(f) \cap im(g) = \varnothing$ and $im(f) \cup im(g) = m+n$, as well as $c_{f(i)} = a_i$ for all $i < m$ and $c_{g(j)} = b_j$ for all $j < n$. f and g are called the **embeddings** of the shuffling.*

Definition 6.7.9 *Let $\rho = \langle R_i : i < m\rangle$ and $\sigma = \langle S_j : j < n\rangle$ be sequences of tight command relations. Then T is called **weakly associated with** ρ and σ if there is a shuffling $\tau = \langle T_i : i < m+n\rangle$ of ρ and σ together with embeddings f and g such that*

$$T = T_0 \circ^0 T_1 \circ^1 T_2 \ldots \circ^{n-2} T_{n-1}$$

where $\circ^i \in \{\circ, \bullet\}$ for every $i < n-1$ and $\circ^i = \circ$ always if $\{i, i+1\} \subseteq im(f)$ or $\{i, i+1\} \subseteq im(g)$.

If $m = n = 2$, we have the following shufflings.

$$\begin{array}{lll} \langle R_0, R_1, S_0, S_1\rangle, & \langle R_0, S_0, R_1, S_1\rangle, & \langle R_0, S_0, S_1, R_1\rangle, \\ \langle S_0, R_0, R_1, S_1\rangle, & \langle S_0, R_0, S_1, R_1\rangle, & \langle S_0, S_1, R_0, R_1\rangle\ . \end{array}$$

The sequence $\langle R_1, S_0, S_1, R_0\rangle$ is not a shuffling because the order of the Rs is not respected. In general there exist $\binom{m+n}{n}$ different shufflings. For every shuffling there are up to 2^{n-1} weakly associated command relations (if $n \leq m$). For example the following command relations are weakly associated to the third shuffling.

$$R_0 \bullet S_0 \circ S_1 \bullet S_1, \quad R_0 \circ S_0 \circ S_1 \circ R_1.$$

The relation $R_0 \circ S_0 \bullet S_1 \circ R_1$ is however not weakly associated to it since $\bullet$ may not occur in between two S.

Lemma 6.7.10 *Let $\rho = \langle R_i : i < m\rangle$ and $\sigma = \langle S_i : i < n\rangle$ be sequences of tight command relations with product T and U, respectively. Then $T \cap U$ is the intersection of all chain like command relations which are products of sequences weakly associated with a shuffling of ρ and σ.*

In practice one has restricted attention to command relations which are characterized by certain sets of nodes, such as the set of all maximal projections, the set of all finite sentences, the set of all sentences in the indicative mood and so on. If we choose P to be the set of nodes carrying a label subsuming the category of finite sentences, then we get the following: if x is a reflexive anaphor, it has to be c–commanded by a subject, which it in turn P–commands. (The last condition makes sure that the subject is a subject of the same sentence.) There is a plethora of similar examples where command relations play a role in defining the range of phenomena. Here, one took not just any old set of nodes but those that where *definable.* To precisify this, let $\langle \mathfrak{T}, \ell\rangle$ with $\ell : T \to N$ be a labelled tree and $Q \subseteq N$. Then $K(Q) := K(\ell^{-1}(Q))$ is called a **definable tight command relation**.

Definition 6.7.11 *Let $\mathfrak{T}$ be a tree and $R \subseteq T \times T$. P is called a (**definable**) **command relation** if it is can be produced from definable tight command relations by means of composition, union and intersection.*

In follows from the previous considerations that the union of definable relations is an intersection of chains of tight relations. A particular role is played by subjacency. The antecedent of a trace must be 1–subjacent to a trace. As is argued in (Kracht, 1998) on the basis of (Chomsky, 1986) this relation is exactly

$$K(\mathrm{CP}) \circ K(\mathrm{IP}) .$$

The movement and copy–transformations create so called *chains*. They connect elements in different positions with each other. The mechanism inside the grammar is coindexation. For as we have said in Section 6.5 traces must be properly governed, and this means that an antecendent must c–command its trace in addition to being coindexed with it. This is a restriction on the structures as well as on the movement transformations. Using coindexation one also has the option of associating antecedent and trace without assuming that anything has ever moved. The transformational history can anyway be projected form the S–structure up to minor (in fact inessential) variations. This means that we need not care whether the S–structure has been obtained by transformations or by some other process introducing the indexation (this is what Koster has argued for). The association between antecedent and trace can also be done in a different way, namely by collecting sets of constituents. We call a chain a certain set of constituents. If we have a chain, the members of it may be thought of as being coindexed, but this is no longer necessary. Chomsky has once again introduced the idea in the 90s that movement is the sequence of copying and deletion and made this one of the main innovations of the reform in the Minimalist Program (see (Chomsky, 1993)). Deletion here is simply marking as phonetically empty (so the copy stays but is marked). However, the same idea can be introduced into GB without substantial change. Let us do this here and introduce in place of Move–α the transformation *Copy*–α. It will turn out that it is actually not necessary to which of the members of the chain has been obtained by copying from which other member. The reason is simple: the copy (= antecedent) c–commands the original (= trace) but the latter does not c–command the former. Knowing who is with whom in a chain is therefore enough. This is the central insight that is used in the theory of chains in (Kracht, 2001b) which we shall now outline. We shall see below that copying gives more information on the derivation than movement, so that we must be careful in saying that nothing has changed by introducing copy–movement.

Recall that constituents are subtrees. In what is to follow we shall not distinguish between a set of nodes and the constituent that is based on that set.

Definition 6.7.12 *Let $\mathfrak{T}$ be a tree. A subset Δ of $\mathfrak{T}$ which is linearly ordered with respect to c–command is called a* ***chain of*** *$\mathfrak{T}$. The element which is highest with respect to c–command is called the* ***head of*** *Δ, the lowest the* ***foot****. Δ is a* ***copy chain*** *if any two members are isomorphic. Δ is a* ***trace chain*** *if all non heads are traces.*

The definition of chains can be supplemented with more detail in the case of copy chains. This will be needed in the sequel.

Definition 6.7.13 *Let $\mathfrak{T}$ be a tree. A* ***copy chain**** ***in*** *$\mathfrak{T}$ is a pair $\langle \Delta, \Phi \rangle$ for which the following holds.*

1. *Δ is a set of pairwise disjoint constituents ordered linearly by c–command.*
2. *$\Phi = \{\varphi_{\mathfrak{C},\mathfrak{D}} : \mathfrak{C}, \mathfrak{D} \in \Delta\}$ is a family of isomorphisms such that for all $\mathfrak{C}, \mathfrak{D}, \mathfrak{A} \in \Delta$ we have*
   - *(a)* $\varphi_{\mathfrak{C},\mathfrak{C}} = 1_{\mathfrak{C}}$,
   - *(b)* $\varphi_{\mathfrak{C},\mathfrak{A}} = \varphi_{\mathfrak{C},\mathfrak{D}} \circ \varphi_{\mathfrak{D},\mathfrak{A}}$.

The ***chain associated with*** *$\langle \Delta, \Phi \rangle$ is Δ.*

Often we shall identify a chain* with its associated chain. The isomorphisms give explicit information which elements of the various constituents are counterparts of which others.

Definition 6.7.14 *Let $\mathfrak{T}$ be a tree and $\mathcal{D} = \langle \Delta, \Phi \rangle$ a copy chain***. Then we put $x \approx_{\mathcal{D}} y$ if there is a map $\varphi \in \Phi$ such that $\varphi(x) = y$. We put $[x]_{\mathcal{D}} := \{y : x \approx_{\mathcal{D}} y\}$. If C is a set of copy chains** *then let $\approx_C$ be the smallest equivalence relation generated by all $\approx_{\mathcal{D}}$, $\mathcal{D} \in C$. Further, let $[x]_C := \{y : x \approx_C y\}$.*

Definition 6.7.15 *Let $\langle \Delta, \Phi \rangle$ be a copy chain***, $\mathfrak{C}, \mathfrak{D} \in \Delta$. $\mathfrak{C}$ is said to be* ***immediately above*** *$\mathfrak{D}$ if there is no $\mathfrak{E} \in \Delta$ distinct from $\mathfrak{C}$ and $\mathfrak{D}$ which c–commands $\mathfrak{D}$ and is c–commanded by $\mathfrak{C}$. A* ***link of*** *Δ is a triple $\langle \mathfrak{C}, \varphi_{\mathfrak{D},\mathfrak{C}}, \mathfrak{D} \rangle$ where $\mathfrak{C}$ is immediately above $\mathfrak{D}$. φ is called a* ***link map****, if it occurs in a link. An* ***ascending map*** *is a composition of link maps.*

Lemma 6.7.16 *Let φ be a link map. Then $t(\varphi(x)) < t(x)$.*

Proof. Let $\varphi = \varphi_{\mathfrak{C},\mathfrak{D}}$, $\mathfrak{C} = {\downarrow}v$, $\mathfrak{D} = {\downarrow}w$. Further, let $t_{\mathfrak{C}}(x)$ be the depth of x in $\mathfrak{C}$, $t_{\mathfrak{D}}(\varphi(x))$ the depth of $\varphi(x)$ in $\mathfrak{D}$. Then $t_{\mathfrak{C}}(x) = t_{\mathfrak{D}}(\varphi(x))$, since φ is an isomorphism. On the other hand $t(x) = t(v) + t_{\mathfrak{C}}(x)$ and $t(\varphi(x)) = t(w) + t_{\mathfrak{D}}(\varphi(x)) = t(w) + t_{\mathfrak{C}}(x)$. The claim now follows from the next lemma given the remark that v c–commands w, but w does not c–command v. □

Lemma 6.7.17 *Let $\mathfrak{T} = \langle T, < \rangle$ be a tree. Let $x, y \in T$ be incomparable elements and that x c–commands y. Then $t(x) \leq t(y)$.*

Proof. There exists a uniquely defined z with $z \succ x$. By definition of c–command we have $z \geq y$. But $y \neq z$, since y is not comparable with x. Hence $y < z$. Now we have $t(x) = t(z) + 1$ and $t(y) > t(z) = t(x) - 1$. Whence the claim. □

We call a pair $\langle \mathfrak{T}, C \rangle$ a **copy chain tree** (**CCT**) if C is a set of copy chains* on $\mathfrak{T}$, $\mathfrak{T}$ a finite tree. We consider among other the following constraints.

> *Uniqueness*. Every constituent of $\mathfrak{T}$ is contained in exactly one chain.
> *Liberation*. Let Γ, Δ be chain, $\mathfrak{C} \in \Gamma$ and $\mathfrak{D}_0, \mathfrak{D}_1 \in \Delta$ with $\mathfrak{D}_0 \neq \mathfrak{D}_1$ such that $\mathfrak{D}_0, \mathfrak{D}_1 \subseteq \mathfrak{C}$. Then $\mathfrak{C}$ is the foot of Γ.

Lemma 6.7.18 *Let K be a CCT which satisfies Uniqueness and Liberation. Further, let φ and φ' be link maps with $im(\varphi) \cap im(\varphi') \neq \varnothing$. Then already $\varphi = \varphi'$.*

Proof. Let $\varphi : \mathfrak{C} \to \mathfrak{D}$, $\varphi' : \mathfrak{C}' \to \mathfrak{D}'$ be link maps. If $im(\varphi) \cap im(\varphi') \neq \varnothing$ then $\mathfrak{D} \subseteq \mathfrak{D}'$ or $\mathfrak{D}' \subseteq \mathfrak{D}$. Without loss of generality we may assume the first. Let $\mathfrak{D} \subsetneq \mathfrak{D}'$. Then also $\mathfrak{C} \subseteq \mathfrak{D}'$, since $\mathfrak{D}$ c–commands $\mathfrak{C}$. Because of *Uniqueness* $\mathfrak{D}'$ is the foot of its chain, in contradiction to our assumption. Hence we have $\mathfrak{D} = \mathfrak{D}'$. $\mathfrak{C}$, $\mathfrak{C}'$ and $\mathfrak{D}$ are now in the same chain. Since φ and φ' are link maps, we additionally have $\mathfrak{C} = \mathfrak{C}'$. Hence $\varphi = \varphi'$. ☐

Definition 6.7.19 *Let K be a CCT. x is called a **root** if x is not in the image of a link map.*

Then proof of the following theorem is now easy to provide. It is left for the reader.

Proposition 6.7.20 *Let K be a CCT which satisfies Uniqueness and Liberation. Let x be an element and τ_i, $i < m$, φ_j, $j < n$, link maps, and y, z roots such that*

$$x = \tau_{m-1} \circ \tau_{m-2} \circ \ldots \circ \tau_0(y) = \varphi_{n-1} \circ \varphi_{n-2} \circ \ldots \circ \varphi_0(z) .$$

Then we have $y = z$, $m = n$ and $\tau_i = \varphi_i$ for all $i < n$.

Hence, for given x a uniquely defined root x_r with $x \approx_C x_r$. Further, there exists a uniquely defined sequence $\langle \varphi_i : i < n \rangle$ of link maps such that x is the image of $\varphi_{n-1} \circ \ldots \circ \varphi_0$. This sequence we call the **canonical decomposition of** x.

Proposition 6.7.21 *Let K be a CCT satisfying Uniqueness and Liberation. Then the following are equivalent.*

1. $x \approx_C y$.
2. $x_r = y_r$.
3. *There exist two ascending maps χ and τ with $y = \tau \circ \chi^{-1}(x)$.*

Proof. (1) $\Rightarrow$ (3). Let $x \approx_C y$. It is not hard to see that then there exists a sequence $\langle \sigma_i : i < p \rangle$ of link maps or inverses thereof such that $y = \sigma_{p-1} \circ \ldots \circ \sigma_0(x)$. Now if σ_i is a link map and σ_{i+1} an inverse link map, then $\sigma_{i+1} = \sigma_i^{-1}$. Hence we may assume that for some $q \leq p$ all σ_i, $i < q$, are inverse link maps and all σ_i, $p > i \geq q$, are link maps. Now put $\tau := \sigma_p \circ \sigma_{p-1} \ldots \circ \sigma_q$ and $\chi := \sigma_0 \circ \sigma_1 \circ \ldots \circ \sigma_{q-1}$. χ and τ are ascending maps. So, (3) obtains. (3) $\Rightarrow$ (2). Let ascending maps χ and τ be given with $y = \tau \circ \chi^{-1}(x)$. Put $u := \chi^{-1}(x)$. Then $u = \rho(u_r)$ for some ascending map ρ. Further, $x = \chi(u) = \chi \circ \rho(u_r)$ and $y = \tau(u) = \tau \circ \rho(u_r)$. Now, u_r is a root and x as well as y are images of u_r under ascending maps. Hence u_r is a root of x and y. This however means that $u_r = x_r = y_r$. Hence, (2) obtains. (2) $\Rightarrow$ (1) is straightforward. □

The proof by the way shows the following fact.

Lemma 6.7.22 *Every ascending map is a canonical decomposition. Every composition of maps equals a product $\tau \circ \chi^{-1}$ where τ and χ are ascending maps. A minimal composition of link maps and their inverses is unique.*

Let x be an element and $\langle \varphi_i : i < n \rangle$ its canonical composition. Then we call

$$T_K(x) := \{\varphi_{j-1} \circ \varphi_{j-2} \circ \ldots \circ \varphi_0(x) : j \leq n\}$$

the **trajectory of** x. The trajectory mirrors the history of x in the process of derivation. We call **root line** of x the set

$$W_K(x) := \{y : y \in T_K(x), y \text{ c–commands } x_r\}$$

Notice that x_r c–commands itself. The **peak of** x is the element of $W_K(x)$ of smallest depth. We write x_p for the peak of x and π_x for the ascending map which sends x to x_π.

Definition 6.7.23 *Let K be a CCT satisfying Uniqueness and Liberation. If r is the root of the tree then r is the **zenith of** r, the **zenith map** is $\zeta_r := 1_T$. If $x \neq r$ then the **zenith map** is the composition $\zeta_y \circ \pi_x$, where $y \succ x_p$. The **zenith of** x equals $\zeta_y \circ \pi_x(x)$. We write x_ζ for the zenith of x.*

Definition 6.7.24 *A link map is called* ***orbital*** *if it occurs in a minimal decomposition of the zenith map.*

At last we can formulate the following restriction on CCTs.

> *No Recycling.* All link maps are orbital.

The effect of a copy transformation is that (1) it adds a new constituent and (2) this constituent is added to an already existing chain as a head. Hence the whole derivation can be thought of as a process which generates a tree together with its chains. These can be explicitly described and this eliminates the necessity of talking about transformations.

Definition 6.7.25 *A* ***copy chain structure (CCS)*** *is a CCT $K = \langle \mathfrak{T}, C \rangle$ which satisfies Uniqueness, Liberation and No Recycling.*

Everything that one wants to say about transformations and derivations can be said also about copy chain structures. The reason for this is the following fact. We call a CCT simply a **tree** if every chain consists of a single constituent. Then also this tree is a copy chain structure. A transformation can naturally be defined as an operation between CCSs. Now it turns out that *Copy–α* turns a CCS into a CCS. The reason for this is that traces have to be bound and may not be moved. (Only in order to reflect this in the definition of the CCSs the condition *No Recycling* has been introduced. Otherwise it was unnecessary.) The following now holds.

Theorem 6.7.26 *A CCT is a CCS if and only if it is obtained from a tree by successive application of Copy–α.*

This legitimates the use of copy chain structures.

Transformational grammar and HPSG are not as different as one might think. The appearance to the contrary is created by the fact that TG is written up using trees, while HPSG has acyclic structures, which need not be trees. In this section we shall show that GB actually defines structures that are more similar to acyclic graphs than to trees. The basis for the alternative formulation is the idea that instead of movement transformations we define an operation that changes the dominance relation. If the daughter constituent z of x moves and becomes a daughter constituent of y then we can simply add to the dominance relation the pair $\langle z, y\rangle$. This rather simple idea has to be worked out carefully. For first we have to change from using the usual transitive dominance relation the immediate dominance relation. Second one has to take care of the linear order of the elements at the surface since it is now not any more represented.

Definition 6.7.27 *A* ***directed acyclic graph*** *is a pair $\langle G, \prec\rangle$, where the transitive closure $<$ of $\prec$ is irreflexive. r is called a* ***root*** *if for every $x \in G$ with $x \neq r$ we have $x < r$. A* ***multidominance structure (MDS)*** *is a triple $\langle M, \prec, r\rangle$ such that $\langle M, \prec\rangle$ is a directed acyclic graph with root r and for every $x < r$ the set $M(x) := \{y : x \prec y\}$ is linearly ordered by $<$.*

With an MDS we only have coded the dominance relation between the constituents. In order to include order we cannot simply add another relation as we did with trees. Depending on the branching number, a fair number of new relations will have to be added, which represent the relations *the ith daughter of* (where $i < n$, the maximum branching number). Since we are dealing with binary branching trees we need only two of these relations.

Definition 6.7.28 *An* ***ordered (binary branching) multidominance structure (OMDS)*** *is a quadruple $\langle M, \prec_0, \prec_1, r\rangle$ such that the following holds:*

1. *$\langle M, \prec_0 \cup \prec_1, r\rangle$ is an MDS.*
2. *From $x \succ_0 y$ and $x \succ_0 z$ follows $y = z$.*
3. *From $x \succ_1 y$ and $x \succ_1 z$ follows $y = z$.*

4. If $x \succ_1 z$ for some z then there exists a $y \neq z$ with $x \succ_0 y$.

(The reader may verify that (2) and (4) together imply that $\succ_0 \cap \succ_1 = \varnothing$.) Let $\langle \mathfrak{T}, <, \sqsubset \rangle$ be a binary branching ordered tree. Then we put $x \prec_0 y$ if x is a daughter of y and there is no daughter z of y with $z \sqsubset x$. Further, we write $x \prec_1 y$ if x is a daughter of y but not $x \prec_0 y$.

Theorem 6.7.29 *Let $K = \langle \mathfrak{T}, C \rangle$ be a CCS over an ordered binary branching tree with root r. Put $M := [x]_C$, $x \in T$, as well as $[x]_C \prec_0 [y]_C$ if and only if there is an $x' \approx_C x$ and an $y' \approx_C y$ with $x' \prec_0 y'$; and put $[x]_C \prec_1 [y]_C$ if and only if there is an $x' \approx_C x$ and an $y' \approx_C y$ such that $x' \prec_1 y'$. Finally let $M(K) := \langle M, \prec_0, \prec_1, [r]_K \rangle$. Then $M(K)$ is an OMDS.*

Now we want to deal with the problem of finding the CCS from the OMDS.

Definition 6.7.30 *Let $\langle M, \prec_0, \prec_1, r \rangle$ be an ordered MDS. An* ***identifier*** *is a sequence $I = \langle x_i : i < n \rangle$ such that $r \succ x_0$ and $x_i \succ x_{i+1}$ for all $i \in n$. $\mathfrak{I}(\mathfrak{M})$ denotes the set of all identifiers of $\mathfrak{M}$. (We agree that $r = x_{-1}$.) The* ***address of*** *I is that sequence $\langle \gamma_i : i < n \rangle$ such that we have for all $i < n$ $x_i \prec_{\gamma_i} x_{i-1}$.*

The following is easy to see.

Proposition 6.7.31 *The addresses of an ordered MDS form a tree domain.*

This means that we have already identified the tree structure. What remains to do is to find the chains. The order is irrelevant, so we ignore it. At first we want to establish which elements are overt. In a CCS an element x is called **overt** if for every $y \geq x$ the constituent $\downarrow y$ is the head of its chain. This we can also describe in the associated MDS. We say a pair $\langle x, y \rangle$ is a **link in** $\langle M, \prec, r \rangle$ if $x \prec y$. The link is **maximal** if y is maximal with respect to $<$ in $M(x)$. An **S–identifier** is an identifier $I = \langle x_i : i < n \rangle$ where $\langle x_{i-1}, x_i \rangle$ is a maximal link for all $i < n$. The overt elements are exactly the S–identifiers.

Definition 6.7.32 *Let $\mathfrak{M} = \langle M, \prec, r\rangle$ and $\mathfrak{M}' = \langle M', \prec', r'\rangle$ be MDSs. Then $\mathfrak{M}'$ is called a* ***link extension of*** *$\mathfrak{M}$ if $M' = M$, $r' = r$ and $\prec' = \prec \cup \{\langle x, y\rangle\}$, where $\langle x, y\rangle$ is maximal in $\mathfrak{M}'$.*

One finds out easily that if K' is derived from K by simple copying then $M(K')$ is isomorphic to a link extension of $M(K)$. Let conversely $\mathfrak{M}'$ be a link extension of $\mathfrak{M}$ and K a CCS such that $M(K) \cong \mathfrak{M}$. Then we claim that there is a CCS K' for which $M(K') \cong \mathfrak{M}'$ and which results by copying from K. This is unique up to isomorphism. The tree is given by $\mathfrak{I}(\mathfrak{M}')$ further let the tree of K be exactly $\mathfrak{I}(\mathfrak{M})$. First we have $\mathfrak{I}(\mathfrak{M}) \subset \mathfrak{I}(\mathfrak{M}')$, and the identity is an embedding whose image contains all identifiers which do not contains the subsequence $x; y$. Let now y' be maximal with respect to $<$ in $\mathfrak{M}$. Further, let I be the S–identifier of y and I' the S–identifier of y' in $\mathfrak{M}$. Then $I' = I; J$ for some J since $y' < y$. Define $\varphi : I; J; x; K \mapsto I; x; K$. This is an isomorphism of the constituent $\downarrow I; J; x$ onto the constituent $\downarrow I; x$. Now we define the chains* on $\mathfrak{I}(\mathfrak{M}')$. Let $\mathcal{D} = \langle \Delta, \Phi\rangle$ be the chain of K which contains the constituent $\downarrow I; J; xK$. Then let $\mathcal{D}' := \langle \Delta \cup \{im(\varphi)\}, \Phi'\rangle$, where $\Phi' := \Phi \cup \{\varphi \circ \chi : \chi \in \Phi\} \cup \{\chi \circ \varphi^{-1} : \chi \in \Phi\rangle$. For every other chain $\mathcal{C}$ let $\mathcal{C}' := \mathcal{C}$. Finally for an identifier $L < I; J; x; K$ we put $\mathcal{K}_L := \langle \{\downarrow L, \{1_{\downarrow L}\}\rangle$. Then we put

$$K' := \langle \mathfrak{I}(\mathfrak{M}'), <, \varepsilon, \{\mathcal{C}' : \mathcal{C} \in C\} \cup \{\mathcal{K}_L : L < I; J; x; K\}\rangle \ .$$

This is a CCS. Evidently it satisfies *Uniqueness*. Further, *Liberation* is satisfied as one easily checks. *No Recycling* costs us some more effort. We only have to show that the new link map is orbital. For this it suffices that $I; J; x$ is an overt element in K. Then the constituent may be moved. However, $I; J$ was overt by assumption and the link $\langle x, y\rangle$ maximal. So, $I; J; x$ is overt as well in $\mathfrak{M}$.

Now, how does one define the kinds of structures that are usual in GB? One approximation is the following. We say a **trace chain structure** is a pair $\langle \mathfrak{T}, C\rangle$ where C is a set of trace chains. If we have a CCS we get the trace chain structure relatively easily. To this end we replace all maximal nonovert constituents in a tree by a trace (which is a one node tree). This however deletes some chain members! Additionally it may happen that some traces are not any more bound. Hence we say that a trace chain structure is a pair $\langle \mathfrak{T}, C\rangle$ which results from a CCS by deleting overt constituents. Now one can define trace chain structures also from MDSs, and it turns out that if two CCSs K and K' have isomorphic MDSs then their trace chain structures are isomorphic. This is for the following reason. An MDS is determined from $\mathfrak{T}$ and $\approx_C$ alone. We can determine the root of every element from $\mathfrak{T}$ and $\approx_C$, and further also the root line. From this we can define the peak of every element and therefore also the zenith. The overt elements are exactly the elements in zenith position. The trace chain structure contains except of the overt elements also the traces. These are exactly the overt daughters of the overt elements.

Let us summarize. There exists a biunique correspondence between derivations of trace chain structures, derivations of CCSs and derivations of MDSs. Further, there is a biunique correspondence between MDSs and trace chain structures. Insofar the latter two structures are exactly equivalent. CCSs contain more information over the derivation (see the exercises).

Exercise 234. The following example shall show why we cannot use the ordering $<$ in the MDSs. Let $\mathfrak{M} = \langle\{0,1,2\}, \prec, 0\rangle$ and $\mathfrak{M}' = \langle\{0,1,2\}, \prec', 0\rangle$ with $\prec = \{\langle 2,1\rangle, \langle 1,0\rangle\}$, but $\prec' = \{\langle 2,1\rangle, \langle 1,0\rangle, \langle 2,0\rangle\}$. Evidently $\prec^+ = \prec'^+$. Construct $\mathfrak{I}(\mathfrak{M})$ and $\mathfrak{I}(\mathfrak{M}')$ as well as the connected CCS.

Exercise 235. Prove Proposition 6.7.3.

Exercise 236. Show Lemma 6.7.10.

Exercise 237. We say, x **ac–commands** y if (i) x and y are incomparable, (ii) x c–commands y and (iii) y does *not* c–command x. Show that ac–command is transitive.

Exercise 238. Show Proposition 6.7.20.

Exercise 239. Let the CCS in Figure 6.2 be given. The members of a chain are annotated by the same upper case Greek letter. Trivial chains are not shown. Let the link maps be $\varphi_\Gamma : 2 \mapsto 4$, $\varphi_\Delta : i \mapsto i + 6(i < 6)$, and $\varphi_\Theta : i \mapsto i + 13(i < 13)$. Compute $[i]_C$ for every i. If instead of φ_Δ we take the map φ'_Δ how do the equivalence classes change?

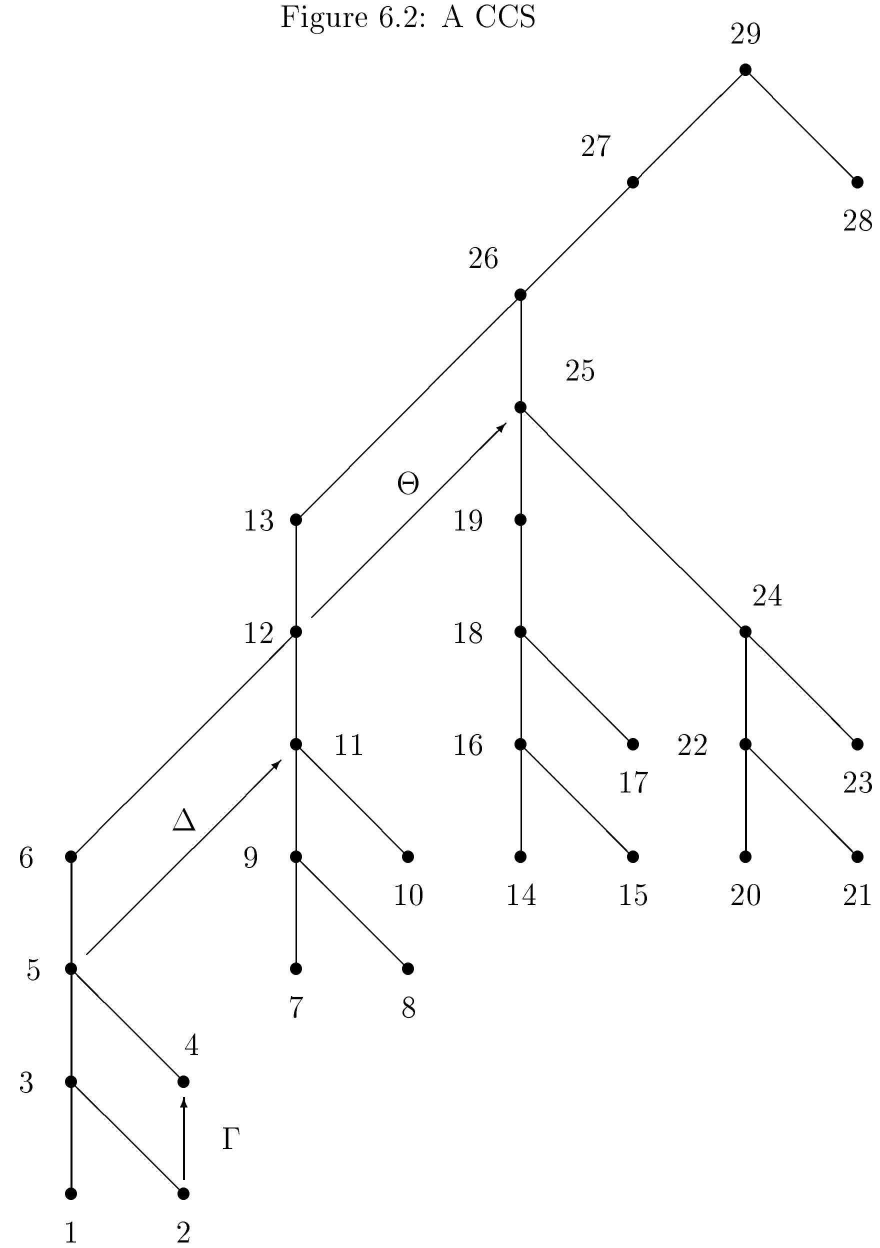
$$\varphi'_\Delta : 1 \mapsto 8, 2 \mapsto 7, 3 \mapsto 9, 4 \mapsto 10, 5 \mapsto 11$$

Determine the peak and the zenith of every element and the maps.

Exercise 240. Compute the maps of the MDS associated with Figure 6.2.

Exercise 241. Let $d(n)$ be the largest number of nonisomorphic CCSs which have (up to isomorphism) the same MDS. Show that $d(n) \in O(2^n)$.

Figure 6.2: A CCS



Index

Bibliography

(Barendregt, 1985) Henk Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* Number 103 in Studies in Logic. Elsevier, Amsterdam, 2 edition, 1985.

(Barker and Pullum, 1990) Chris Barker and Geoffrey Pullum. A theory of command relations. *Linguistics and Philosophy*, 13:1–34, 1990.

(Bird and Ellison, 1994) Steven Bird and Mark Ellison. One–level phonology: Autosegmental representations and rules as finite automata. *Computational Linguistics*, 20:55 – 90, 1994.

(Blackburn, 1993) Patrick Blackburn. Modal logic and attribute value structures. In Maarten de Rijke, editor, *Diamonds and Defaults*, Synthese No. 229, pages 19 –65. Kluwer, 1993.

(Blok and Pigozzi, 1990) Wim J. Blok and Don J. Pigozzi. Algebraizable logics. *Memoirs of the Americal Mathematical Society*, 77(396), 1990.

(Bochvar, 1938) D. A. Bochvar. On a three–valued logical calculus and its application to the analysis of contradictions. *Mathematicheskii Sbornik*, 4:287 – 308, 1938.

(Bresnan *et al.*, 1987) Joan Bresnan, Ronald M. Kaplan, Stanley Peters, and Annie Zaenen. Cross–Serial Dependencies in Dutch. In Walter Savitch, Emmon Bach, William Marsch, and

Gila Safran-Naveh, editors, *The Formal Complexity of Natural Language*, pages 286 – 319. Reidel, Dordrecht, 1987.

(Büchi, 1960) J. Büchi. Weak second–order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:66 – 92, 1960.

(Burris and Sankappanavar, 1981) Stanley Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Number 78 in Graduate Texts in Mathematics. Springer, 1981.

(Carpenter, 1992) Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press, 1992.

(Chandra *et al.*, 1981) A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28:114 – 133, 1981.

(Chomsky and Halle, 1968) Noam Chomsky and Morris Halle. *The sound pattern of English*. Harper and Row, New York, 1968.

(Chomsky, 1959) Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2:137 – 167, 1959.

(Chomsky, 1962) Noam Chomsky. Context–free grammars and pushdown storage. *MIT Research Laboratory of Electronics Quarterly Progress Report*, 65, 1962.

(Chomsky, 1981) Noam Chomsky. *Lecture Notes on Government and Binding*. Foris, Dordrecht, 1981.

(Chomsky, 1986) Noam Chomsky. *Barriers*. MIT Press, Cambrigde (Mass.), 1986.

(Chomsky, 1993) Noam Chomsky. A minimalist program for linguistic theory. In K. Hale and Keyser S. J., editors, *The View* *from Building 20: Essays in Honour of Sylvain Bromberger*, pages 1 – 52. MIT Press, 1993.

(Church, 1933) Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 2:346 – 366, 1933.

(Church, 1940) Alonzo Church. A formulation of the simply theory of types. *Journal of Symbolic Logic*, 5:56 – 68, 1940.

(Culy, 1987) Christopher Culy. The Complexity of the Vocabulary of Bambara. In Walter Savitch, Emmon Bach, William Marsch, and Gila Safran-Naveh, editors, *The Formal Complexity of Natural Language*, pages 345 – 351. Reidel, Dordrecht, 1987.

(Curry, 1930) Haskell B. Curry. Grundlagen der kombinatorischen Logik. *American Journal of Mathematics*, 52:509 – 536, 789 – 834, 1930.

(Davey and Priestley, 1990) B. A. Davey and H. A. Priestley. *Lattices and Order*. Cambridge University Press, Cambridge, 1990.

(Deutsch *et al.*, 2000) David Deutsch, Artur Ekert, and Rossella Luppacchini. Machines, logic and quantum physics. *Bulletin of Symbolic Logic*, 6:265 – 283, 2000.

(Doner, 1970) J. E. Doner. Tree acceptors and some of their applications. *Journal of Computer and Systems Sciences*, 4:406 – 451, 1970.

(Dowty *et al.*, 1981) David R. Dowty, Robert E. Wall, and Stanley Peters. *Introduction to Montague Semantics*. Number 11 in Synthese Library. Reidel, Dordrecht, 1981.

(Driesner, 2001) Eli Driesner. Tarski's Restricted Form and Neale's Quantificational Treatment of Proper Names. *Linguistics and Philosophy*, 24:405 – 415, 2001.

(Ebbinghaus and Flum, 1995) Hans-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer, 1995.

(Ebert and Kracht, 2000) Christian Ebert and Marcus Kracht. Formal syntax and semantics of case stacking languages. In *Proceedings of the EACL 2000*, 2000.

(Eisenberg, 1973) Peter Eisenberg. A Note on 'Identity of Constituents'. *Linguistic Inquiry*, 4:417 – 420, 1973.

(Fiengo and May, 1994) Robert Fiengo and Robert May. *Indices and Identity.* Number 24 in Linguistic Inquiry Monographs. MIT Press, Cambridge, Massachussetts, 1994.

(Fine, 1992) Kit Fine. Transparency, Part I: Reduction. Unpublished manuscript, UCLA, 1992.

(Frege, 1962) Gottlob Frege. Funktion und Begriff. In Günther Patzig, editor, *Funktion, Begriff, Bedeutung. Fünf logische Studien.* Vandenhoeck & Ruprecht, Göttingen, 1962.

(Frey, 1993) Werner Frey. *Syntaktische Bedingungen für die semantische Interpretation.* Number 35 in Studia Grammatica. Akademie Verlag, Berlin, 1993.

(Gamut, 1991a) L. T. F. Gamut. *Logic, Language and Meaning*, volume 2: Intensional Logic and Logical Grammar. University of Chicago Press, Chicago, 1991.

(Gamut, 1991b) L. T. F. Gamut. *Logic, Language and Meaning*, volume 1: Introduction to Logic. University of Chicago Press, Chicago, 1991.

(Gärdenfors, 1988) Peter Gärdenfors. *Knowledge in Flux.* MIT Press, Cambridge (Mass.), 1988.

(Gazdar *et al.*, 1985) Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar.* Blackwell, London, 1985.

(Gazdar *et al.*, 1988) G. Gazdar, G. Pullum, R. Carpenter, T. Hukari, and R. Levine. Category structures. *Computational Linguistics*, 14:1 – 19, 1988.

(Geach, 1972) Peter Geach. A Program for Syntax. In Donald Davidson and Gilbert Harman, editors, *Semantics for Natural Language*, number 40 in Synthese Library. Reidel, Dordrecht, 1972.

(Geller and Harrison, 1977) M. M. Geller and M. A. Harrison. On $LR(k)$ grammars and languages. *Theoretical Computer Science*, 4:245 – 276, 1977.

(Geurts, 1998) Bart Geurts. Presupposition and Anaphors in Attitude Contexts. *Linguistics and Philosophy*, 21:545 – 601, 1998.

(Ginsburg, 1975) Seymour Ginsburg. *Algebraic and Automata–Theoretic Properties of Formal Languages.* North–Holland, Amsterdam, 1975.

(Goldstern and Judah, 1995) Martin Goldstern and Haim Judah. *The Incompleteness Phenomenon.* AK Peters, Wellesley, Massachussetts, 1995.

(Grätzer, 1968) George Grätzer. *Universal Algebra.* van Nostrand, New York, 1968.

(Grätzer, 1971) George Grätzer. *Lattice Theory: First Concepts and Distributive Lattices.* Freeman, 1971.

(Greibach, 1967) Sheila A. Greibach. A new normal–formal theroem for context–free phrase structure grammars. *Journal of the Association for Computing Machinery*, 13:42 – 52, 1967.

(Grewendorf *et al.*, 1987) Günter Grewendorf, Fritz Hamm, and Wolfgang Sternefeld. *Sprachliches Wissen. Eine Einführung in moderne Theorien der grammatischen Beschreibung.* Number 695 in suhrkamp taschenbuch wissenschaft. Suhrkamp Verlag, 1987.

(Groenendijk and Stokhof, 1991) Jeroen Groenendijk and Martin Stokhof. Dynamic predicate logic. *Linguistics and Philosophy*, 14:39 – 100, 1991.

(Groenink, 1997) Annius Groenink. *Surface without Structure. Word Order and Tractability Issues in Natural Language Analysis.* PhD thesis, University of Utrecht, 1997.

(Haider, 1992) Hubert Haider. Branching and discharge. Technical Report 23, SFB 340, Universität Stuttgart, 1992.

(Haider, 1994) Hubert Haider. Detached clauses – the later the deeper. Technical Report 41, SFB 340, Universitiät Stuttgart, 1994.

(Halmos, 1956) Paul Halmos. Homogeneous locally finite polyadic boolean algebras of infinite degree. *Fundamenta Mathematicae*, 43:255 – 325, 1956.

(Hamm and van Lambalgen, 2001) Friedrich Hamm and Michiel van Lambalgen. Events, nominalization and the progressive. manuscript, 2001.

(Harkema, 2001) Henk Harkema. A Characterization of Minimalist Languages. In Philippe de Groote, Glyn Morrill, and Christian Retoré, editors, *Logical Aspects of Computational Linguistics (LACL '01)*, number 2099 in Lecture Notes in Artificial Intelligence, pages 193 – 211, Berlin, Heidelberg, 2001. Springer.

(Harris, 1963) Zellig S. Harris. *Structural Linguistics.* The University of Chicago Press, 1963.

(Harrison, 1978) Michael A. Harrison. *Introduction to Formal Language Theory.* Addison Wesley, Reading (Mass.), 1978.

(Heim, 1983) Irene Heim. On the projection problem for presuppositions. In M. Barlow and D. Flickinger, D. Westcoat, editors, *Proceedings of the 2nd West Coast Conference on Formal Linguistics*, pages 114 – 126, Stanford University, 1983.

(Henkin *et al.*, 1971) Leon Henkin, Donald Monk, and Alfred Tarski. *Cylindric Algebras. Part 1.* Number 64 in Studies in Logic and the Foundation of Mathematics. North–Holland, Amsterdam, 1971.

(Hindley and Longo, 1980) J. R. Hindley and G. Longo. Lambda calculus models and extensionality. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 26:289 – 310, 1980.

(Hindley *et al.*, 1972) J. R. Hindley, B. Lercher, and J. P. Seldin. *Introduction to Combinatory Logic.* Number 7 in London Mathematical Society Lecture Notes. Oxford University Press, Oxford, 1972.

(Hintikka, 1962) Jaakko Hintikka. *Knowledge and Belief. An Introduction into the logic of the two notions.* Cornell University Press, Ithaca, 1962.

(Hodges, 2001) Wilfrid Hodges. Formal features of compositionality. *Journal of Logic, Language and Information*, 10:7 – 28, 2001.

(Hopcroft and Ullman, 1969) John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata.* Addison Wesley, Reading (Mass.), 1969.

(Hulst, 1984) Harry van der Hulst. *Syllable Structure and Stress in Dutch.* Foris, Dordrecht, 1984.

(Huybregts, 1984) Riny Huybregts. Overlapping Dependencies in Dutch. *Utrecht Working Papers in Linguistics*, 1:3 – 40, 1984.

(IPA, 1999) IPA. *Handbook of the International Phonetic Association.* Cambridge University Press, Cambridge, 1999.

(Ireland and Rosen, 1990) Kenneth Ireland and Michael Rosen. *A Classical Introduction to Modern Number Theory.* Number 84 in Graduate Texts in Mathematics. Springer, Berlin, Heidelberg, 2 edition, 1990.

(Jackendoff, 1977) Ray Jackendoff. *$\overline{X}$–Syntax: A Study of Phrase Structure.* Number 2 in Linguistic Inquiry Monographs. MIT Press, Cambridge (Mass.), 1977.

(Johnson, 1969) J. S. Johnson. Nonfinitizabilitity of classes of representable polyadic algebras. *Journal of Symbolic Logic*, 34:344 – 352, 1969.

(Johnson, 1988) Mark Johnson. *Attribute–Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes.* CSLI, 1988.

(Jones, 1955) Burton W. Jones. *The Theory of Numbers.* Holt, Rinehart and Winston, New York, 1955.

(Joshi, 1985) Aravind K. Joshi. How much context–sensitivity is needed for characterizing structural descriptions: Tree adjoining grammars. In David Dowty, Lauri Karttunen, and Arnold Zwicky, editors, *Natural language parsing: Psychological, computational and theoretical perspectives.* Cambdrige University Press, Cambridge, 1985.

(Just and Weese, 1996) Winfried Just and Martin Weese. *Discovering Modern Set Theory. I. The Basics*, volume 8 of *Graduate Studies in Mathematics.* AMS, 1996.

(Just and Weese, 1997) Winfried Just and Martin Weese. *Discovering Modern Set Theory. II. Set–Theoretic Tools for Every Mathematician*, volume 18 of *Graduate Studies in Mathematics*. AMS, 1997.

(Kamp and Reyle, 1993) Hans Kamp and Uwe Reyle. *From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Language and Discourse Representation*. Kluwer, Dordrecht, 1993.

(Kamp, 1981) Hans Kamp. A theory of truth and semantic representation. In Jeroen Groenendijk, editor, *Formal methods in the study of language*. Mathematisch Centrum, 1981.

(Kaplan and Kay, 1994) Ron M. Kaplan and Martin Kay. Regular Models of Phonological Rule Systems. *Computational Linguistics*, 20:331 – 378, 1994.

(Karttunen, 1974) Lauri Karttunen. Presuppositions and linguistic context. *Theoretical Linguistics*, 1:181 – 194, 1974.

(Kasami, 1965) Tadao Kasami. An efficient recognition and syntax–analysis algorithm for context–free languages. Technical report, Air Force Cambridge Research Laboratory, Bedford, Mass., 1965. Science Report AFCRL–65–758.

(Kayne, 1994) Richard S. Kayne. *The Antisymmetry of Syntax*. Number 25 in Linguistic Inquiry Monographs. MIT Press, 1994.

(Keenan and Faltz, 1985) Edward P. Keenan and Leonard L. Faltz. *Boolean Semantics for Natural Language*. Reidel, Dordrecht, 1985.

(Kempson, 1975) Ruth Kempson. *Presupposition and the delimitation of semantics*. Cambridge University Press, Cambridge, 1975.

(Kleene, 1956) Stephen C. Kleene. Representation of events in nerve nets. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3 – 40. Princeton University Press, 1956.

(Knuth, 1965) Donald Knuth. On the translation of languages from left to right. *Information and Control*, 8:607 – 639, 1965.

(Koskenniemi, 1983) Kimmo Koskenniemi. Two–level morphology. a general computational model for word–form recognition. Technical Report 11, Department of General Linguistics, University of Helsinki, 1983.

(Koster, 1986) Jan Koster. *Domains and Dynasties: the Radical Autonomy of Syntax.* Foris, Dordrecht, 1986.

(Kracht, 1993) Marcus Kracht. Mathematical aspects of command relations. In *Proceedings of the EACL 93*, pages 241 – 250, 1993.

(Kracht, 1994) Marcus Kracht. Logic and control: How they determine the behaviour of presuppositions. In Jan van Eijck and Albert Visser, editors, *Logic and Information Flow*, pages 88 – 111. MIT Press, Cambridge, Massachusetts, 1994.

(Kracht, 1995) Marcus Kracht. Is there a genuine modal perspective on feature structures? *Linguistics and Philosophy*, 18:401 – 458, 1995.

(Kracht, 1997) Marcus Kracht. Inessential Features. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics (LACL '96)*, number 1328 in Lecture Notes in Artificial Intelligence, pages 43 – 62, Heidelberg, 1997. Springer.

(Kracht, 1998) Marcus Kracht. Adjunction Structures and Syntactic Domains. In Uwe Mönnich and Hans-Peter Kolb, editors, *The Mathematics of Sentence Structure. Trees and Their Logics*, number 44 in Studies in Generative Grammar, pages 259 – 299. Mouton–de Gruyter, Berlin, 1998.

(Kracht, 1999) Marcus Kracht. *Tools and Techniques in Modal Logic*. Number 142 in Studies in Logic. Elsevier, Amsterdam, 1999.

(Kracht, 2001a) Marcus Kracht. Modal Logics That Need Very Large Frames. *Notre Dame Journal of Formal Logic*, 42:141 – 173, 2001.

(Kracht, 2001b) Marcus Kracht. Syntax in chains. *Linguistics and Philosophy*, 24:467 – 529, 2001.

(Kracht, 2002) Marcus Kracht. Against the feature bundle theory of case. In Ellen Brandner and Heike Zinsmeister, editors, *New Perspcetives on Case*. CSLI, 2002.

(Kuroda, 1964) S.-Y. Kuroda. Classes of languages and linear bounded automata. *Information and Control*, 7:207 – 223, 1964.

(Lamb, 1966) Sydney M. Lamb. *Outline of Stratificational Grammar*. Washington, 1966.

(Lambek, 1958) Joachim Lambek. The Mathematics of Sentence Structure. *The American Mathematical Monthly*, 65:154 – 169, 1958.

(Landweber, 1963) Peter S. Landweber. Three theorem on phrase structure grammars of type 1. *Information and Control*, 6:131 – 137, 1963.

(Langholm, 2001) Tore Langholm. A Descriptive Characterisation of Indexed Grammars. *Grammars*, 4:205 – 262, 2001.

(Lehmann, 1993) Winfred P. Lehmann. *Theoretical Bases of Indo–European Linguistics*. Routledge, London, 1993.

(Leibniz, 2000) Gottfried Wilhelm Leibniz. *Die Grundlagen des logischen Kalküls. (Lateinisch–Deutsch)*. Number 525 in Philosophische Bibliothek. Meiner Verlag, Hamburg, 2000.

(Levelt, 1991) Willem P. Levelt. *Speaking. From Intention to Articulation.* MIT Press, Cambridge (Mass.), 2 edition, 1991.

(Lyons, 1968) John Lyons. *Einführung in die moderne Linguistik.* Beck'sche Elementarbücher. Beck Verlag, München, 1968.

(Lyons, 1978) John Lyons. *Semantics. Vol. 1.* Cambridge University Press, Cambridge, 1978.

(Manaster-Ramer and Kac, 1990) Alexis Manaster-Ramer and Michael B. Kac. The Concept of Phrase Structure. *Linguistics and Philosophy*, 13:325 – 362, 1990.

(Manaster-Ramer, 1986) Alexis Manaster-Ramer. Copying in natural languages, context–freeness and queue grammars. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 85 – 89, 1986.

(Markov, 1947) A. A. Markov. On the impossibility of certain algorithms in the theory of associative systems (russian). *Doklady Akadémii Nauk SSSR*, 55:587 – 590, 1947.

(Marsh and Partee, 1987) William Marsh and Barbara H. Partee. How Non–Context Free is Variable Binding? In Walter Savitch, Emmon Bach, William Marsch, and Gila Safran-Naveh, editors, *The Formal Complexity of Natural Language*, pages 369 – 386. Reidel, Dordrecht, 1987.

(Mel'cuk, 1993 200) Igor Mel'cuk. *Cours de Morphologie Générale.*, volume 1 – 5. Les Presses de l'Université de Montréal, 1993 – 200.

(Meyer, 1982) A. R. Meyer. What is a model of the lambda calculus? *Information and Control*, 52:87 – 122, 1982.

(Michaelis, 2001) Jens Michaelis. *On Formal Properties of Minimalist Grammars.* PhD thesis, Universität Potsdam, 2001.

(Miller, 1999) Philip H. Miller. *Strong Generative Capacity. The Semantics of Linguistic Formalisms.* CSLI, Stanford, 1999.

(Mitchell, 1990) J. C. Mitchell. Type systems for programming languages. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol B. Formal Models and Semantics*, pages 365 – 458. Elsevier Science Publishers, Amsterdam, 1990.

(Monk, 1969) Donald J. Monk. Nonfinitizabilitity of classes of representable cylindric algebras. *Journal of Symbolic Logic*, 34:331 – 343, 1969.

(Mönnich, 1999) Uwe Mönnich. On cloning context–freeness. In Hans-Peter Kolb and Uwe Mönnich, editors, *The Mathematics of Sentence Structure. Trees and their Logics*, number 44 in Studies in Generative Grammar, pages 195 – 229. Mouton de Gruyter, Berlin, 1999.

(Moschovakis, 1994) Yannis Moschovakis. Sense and denotation as algorithm and value. In Juha Oikkonen and Jouko Väänänen, editors, *Proceedings of the ASL Meeting 1990, Helsinki*, number 2 in Lecture Notes in Logic, pages 210 – 249, Berlin and Heidelberg, 1994. Springer.

(Muskens *et al.*, 1997) Reinhard Muskens, Johan van Benthem, and Albert Visser. Dynamics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 587 – 648. Elsevier, Amsterdam, 1997.

(Myhill, 1960) John Myhill. Linear bounded automata. Technical report, Wright–Patterson Air Force Base, 1960.

(Ojeda, 1988) Almerindo E. Ojeda. A Linear Precedence Account of Cross-Serial Dependencies. *Linguistics and Philosophy*, 11:457 – 492, 1988.

(Parikh, 1961) Rohit Parikh. Language generating devices. *MIT Research Laboratory of Electronics Quarterly Progress Report*, 60:199 – 212, 1961.

(Pentus, 1995) Mati Pentus. Models for the Lambek calculus. *Annals of Pure and Applied Logic*, 75:179 – 213, 1995.

(Pentus, 1997) Mati Pentus. Product–Free Lambek–Calculus and Context–Free Grammars. *Journal of Symbolic Logic*, 62:648 – 660, 1997.

(Peters and Ritchie, 1971) Stanley P. Peters and R. W. Ritchie. On restricting the base component of transformational grammars. *Information and Control*, 18:483 – 501, 1971.

(Peters and Ritchie, 1973) Stanley P. Peters and R. W. Ritchie. On the generative power of transformational grammars. *Information Sciences*, 6:49 – 83, 1973.

(Pigozzi and Salibra, 1995) Don Pigozzi and Antonino Salibra. The Abstract Variable–Binding Calculus. *Studia Logica*, 55:129 – 179, 1995.

(Pigozzi, 1991) Don J. Pigozzi. Fregean Algebraic Logic. In Hajnal Andréka, Donald Monk, and István Németi, editors, *Algebraic Logic*, number 54 in Colloquia Mathematika Societatis János Bolyai, pages 475 – 504. János Bolyai Matematikai Társulat and North–Holland, 1991.

(Pollard and Sag, 1987) Carl Pollard and Ivan Sag. *Information–Based Syntax and Semantics. Vol. 1.* Number 13 in Lecture Notes. CSLI, Stanford, 1987.

(Pollard and Sag, 1994) Carl Pollard and Ivan Sag. *Head–Driven Phrase Structure Grammar.* The University of Chicago Press, Chicago, 1994.

(Pollard, 1984) Carl J. Pollard. *Generalized Phrase Structure Grammar, Head Grammars and Natural Language*. PhD thesis, Stanford University, 1984.

(Post, 1936) Emil L. Post. Finite combinatory processes – formulation. *Journal of Symbolic Logic*, 1:103 – 105, 1936.

(Post, 1947) Emil L. Post. Recursive unsolvability of a problem of Thue. *Journal of Symbolic Logic*, 11:1 – 11, 1947.

(Prucnal and Wroński, 1974) T. Prucnal and A. Wroński. An algebraic characterization of the notion of structural completeness. *Bulletin of Section Logic of the Polish Academy of Sciences*, 3:20 – 33, 1974.

(Quine, 1960) Willard van Orman Quine. Variables explained away. *Proceedings of American Philosophical Society*, 104:343 – 347, 1960.

(Radzinski, 1990) Daniel Radzinski. Unbounded Syntactic Copying in Mandarin Chinese. *Linguistics and Philosophy*, 13:113 – 127, 1990.

(Rounds, 1988) William C. Rounds. LFP: A Logic for Linguistic Description and an Analysis of its Complexity. *Computational Linguistics*, 14:1 – 9, 1988.

(Russell, 1905) Bertrand Russell. On denoting. *Mind*, 14:479 – 493, 1905.

(Sain and Thompson, 1991) Ildikó Sain and Richard S. Thompson. Finite Schema Axiomatization of Quasi–Polyadic Algebras. In Hajnal Andréka, Donald Monk, and István Németi, editors, *Algebraic Logic*, number 54 in Colloquia Mathematika Societatis János Bolyai, pages 539 – 571. János Bolyai Matematikai Társulat and North–Holland, 1991.

(Salomaa, 1978) Arto K. Salomaa. *Formale Sprachen.* Springer Verlag, Berlin, 1978.

(Saussure, 1967) Ferdinand de Saussure. *Grundfragen der allgemeinen Sprachwissenschaft.* Walter de Gruyter, Berlin, 2 edition, 1967.

(Schönfinkel, 1924) Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305 – 316, 1924.

(Seki *et al.*, 1991) Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context–free grammars. *Theoretical Computer Science*, 88:191 – 229, 1991.

(Sestier, 1960) A. Sestier. Contributions à une théorie ensembliste des classifications linguistiques. In *Actes du Ier Congrès de l'AFCAL*, pages 293 – 305, Grenoble, 1960.

(Shieber, 1985) Stuart Shieber. Evidence against the Context–Freeness of Natural Languages. *Linguistics and Philosophy*, 8:333 – 343, 1985.

(Shieber, 1992) Stuart Shieber. *Constraint–Based Grammar Formalisms.* MIT Press, Cambridge (Mass.), 1992.

(Staal, 1967) J. F. Staal. *Word Order in Sanskrit and Universal Grammar.* Foundations of Language, Supplementary Series No. 5. Reidel, Dordrecht, 1967.

(Stabler, 1997) Edward P. Stabler. Derivational minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics (LACL '96)*, number 1328 in Lecture Notes in Artificial Intelligence, pages 68 – 95, Heidelberg, 1997. Springer.

(Steedman, 1990) Mark Steedman. Gapping as constituent coordination. *Linguistics and Philosophy*, 13:207 – 263, 1990.

(Steedman, 1996) Mark Steedman. *Surface Structure and Interpretation*. Number 30 in Linguistic Inquiry Monographs. MIT Press, Cambridge, Massachussetts, 1996.

(Tarski, 1983) Alfred Tarski. The concept of truth in formalized languages. In J. Corcoran, editor, *Logic, Semantics, Metamathematics*, pages 152 – 178. Hackett Publishing, Indianapolis, 1983.

(Tesnière, 1982) Lucien Tesnière. *Eléments de syntaxe structurale*. Klincksieck, Paris, 4 edition, 1982.

(Thatcher and Wright, 1968) J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second–order logic. *Mathematical Systems Theory*, 2:57 – 81, 1968.

(Thue, 1914) Axel Thue. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln. *Skrifter utgit av Videnskapsselkapet i Kristiania, I. Mathematisk–naturvidenskabelig klasse*, 10, 1914.

(Trakhténbrodt, 1950) B. A. Trakhténbrodt. Névozmožnost' algorifma dlá problémy razréšimosti na konéčnyh klassah. *Doklady Akadémii Nauk SSSR*, pages 569 – 572, 1950.

(Turing, 1936) Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematicsl Society*, 42:230 – 265, 1936.

(Uszkoreit, 1987) Hans Uszkoreit. *Word Order and Constituent Order in German*. Number 8 in Lecture Notes. CSLI, 1987.

(van der Sandt, 1988) Rob A. van der Sandt. *Context and Presupposition*. Croom Helm, London, 1988.

(van Eijck and Kamp, 1997) Jan van Eijck and Hans Kamp. Representing discourse in context. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 179 – 237. Elsevier, Amsterdam, 1997.

(van Eijck, 1994) Jan van Eijck. Presupposition failure: a comedy of errors. *Formal Aspects of Computing*, 3, 1994.

(Vaught, 1995) Robert L. Vaught. *Set Theory. An Introduction.* Birkhäuser, Basel, 2 edition, 1995.

(Veltman, 1985) Frank Veltman. *Logics for Conditionals.* PhD thesis, Department of Philosophy, University of Amsterdam, 1985.

(Vermeulen, 1995) Kees F. M. Vermeulen. Merging without mystery or: Variables in dynamic semantics. *Journal of Philosophical Logic*, 24:405 – 450, 1995.

(Vijay-Shanker *et al.*, 1987) K. Vijay-Shanker, David J. Weir, and Aravind Joshi. Characterizing structural descriptions produced by various grammar formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics*, pages 104 – 111, 1987.

(von Stechow and Sternefeld, 1987) Arnim von Stechow and Wolfgang Sternefeld. *Bausteine syntaktischen Wissens. Ein Lehrbuch der generativen Grammatik.* Westdeutscher Verlag, Opladen, 1987.

(Wartena, 1999) Christian Wartena. *Storage Structures and Conditions on Movement in Natural Language Syntax.* PhD thesis, Institut für allgemeine Sprachwissenschaft, Universität Potsdam, 1999.

(Younger, 1967) D. H. Younger. Recognition and parsing of context–free languages in time n^3. *Information and Control*, 10:189 – 208, 1967.

(Zadrozny, 1994) Wlodek Zadrozny. From Compositional Semantics to Systematic Semantics. *Linguistics and Philosophy*, 17:329 – 342, 1994.

(Zimmermann, 1999) Thomas Ede Zimmermann. Meaning Postulates and the Model–Theoretic Approach to Natural Language Semantics. *Linguistics and Philosophy*, 22:529 – 561, 1999.