Strict Compositionality and Literal Movement Grammars

Marcus Kracht[1]

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

Abstract. The principle of compositionality, as standardly defined, regards grammars as compositional that are not compositional in an intuitive sense of the word. There is, for example, no notion of a part of a string or structure involved in the formal definition. We shall therefore propose here a stricter version of compositionality. It consists in a conjunction of principles which assure among other that complex signs are in a literal sense made from simpler signs, with the meaning and syntactic type being computed in tandem. We shall argue that given this strict principle, quite powerful string handling mechanisms must be assumed. Linear Context Free Rewrite Systems (see [13]) are not enough to generate human languages, but most likely Literal Movement Grammars will do.

A grammar is *compositional* if the meaning of a (complex) expression is determined from the meaning of its (immediate) parts together with their mode of combination. A language is *compositional* if there is a compositional grammar that generates it. (See [6] for a discussion.) Recently, Zadrozny [15] has presented a proof that any meaning assignment function can go with a compositional grammar. This proof has been dismissed by Kazmi and Pelletier [7] and Westerståhl [14] on the grounds that Zadrozny is computing the meaning through a detour. He introduces new functions, one per word, and reduces the requirement of compositionality to the solution of an equation, which always exists if one assumes the existence of non–well founded sets. In fact, Zadrozny himself finds this solution formalistic and proposes restrictions under which certain pathological examples are excluded. So, there is an agreement that one should not count the proof by Zadrozny as showing us anything about the problem of compositionality of language — if the latter is understood in an intuitive sense. But in what sense can or must compositionality be understood if it is to have any significance? And what is the source of the complaints that people like Kazmi, Pelletier, Westerståhl (and others) raise? In this paper we will try to elaborate a formal setup of language as a semiotic system that allows to discuss this problem in a meaningful way. We shall propose a definition of compositionality, called strict compositionality, which is restrictive and therefore non vacuous.

Before we enter the discussion, we need to provide some basic definitions. Let F be a set (possibly empty) and Ω a function from F to the set ω of natural numbers. Throughout this paper, F will always be assumed to be finite. The pair $\langle F, \Omega\rangle$ (often denoted just by Ω) is called a *signature*. A (*partial*) *Ω–algebra* is a pair $\mathfrak{A} = \langle A, I\rangle$, where A is a non-empty set (called the *carrier set* of $\mathfrak{A}$) and I is a function with codomain F, which assigns to each $f \in F$ a (partial) $\Omega(f)$–ary function on A. A *homomorphism* from a $\mathfrak{A}$ to some (partial) Ω–algebra $\mathfrak{B} = \langle B, J\rangle$ is a function $h : A \to B$ such that for all $f \in F$ and all $\boldsymbol{a} \in A^{\Omega(f)}$:

$$h(I(f)(\boldsymbol{a})) = J(f)(h(\boldsymbol{a})) \ .$$

(Here, $h(\boldsymbol{a}) = \langle h(a_0), \dots, h(a_{\Omega(f)-1})\rangle$.) This means that the left hand side is defined iff the right hand is defined, and if both sides are defined they are equal. If $B \subseteq A$ and h is the identity map we speak of $\mathfrak{B}$ as a *subalgebra* of $\mathfrak{A}$. In that case one can show that $J(f) = I(f) \restriction B^{\Omega(f)}$. Let $X \subseteq A$ be an arbitrary subset of A. We denote by $[X]$ the least subset of A that contains X and is closed under all partial functions $I(f)$. If X has cardinality n and $[X] = A$, we say that $\mathfrak{A}$ is *n–generated.* In particular, $\mathfrak{A}$ is 0–generated if $A = [\varnothing]$.

Fix a set $V := \{x_i : i \in \omega\}$. An *$\Omega$–term* is defined inductively as follows. Every $x_i \in V$ is a term; if $f \in F$ and t_i, $i < \Omega(f)$, are terms, so is $f(t_0, t_1, \dots, t_{\Omega(f)-1})$. Now, given an algebra $\langle A, I\rangle$, the set of polynomials is the set of Ω_A–terms, where Ω_A is obtained from Ω by adding a nullary function symbol $\underline{a}$ for each $a \in A$, which is interpreted by a in A.

We assume that a language is a set of *signs.* In accordance with the syntactic literature, a sign is a triple, consisting of an exponent (this is what you can actually see of the sign), a type, and a meaning.

Definition 1 *Let E, T and M be (nonempty) sets. A* ***sign*** *over (E, T, M) is a member σ of the Cartesian product $E \times T \times M$. If $\sigma = (e, t, m)$, we call e the* ***exponent*** *of σ, t the* ***type*** *of σ and m the* ***meaning*** *of σ. A* ***language*** *over (E, T, M) is a set of (E, T, M)–signs.*

We denote the first projection from a sign by ε, the second projection by τ and the third projection by μ. Given a language L, then $\varepsilon[L] := \{\varepsilon(\sigma) : \sigma \in L\}$ is the set of exponents of L. When there is no risk of confusion we shall also speak of $\varepsilon[L]$ as a *language.* This covers the traditional usage of a subset of A^* being a language. Here are some examples of signs.

$$\begin{array}{lcl} \text{‘a’} & : & \langle \mathsf{a}, np/n, \lambda\mathcal{P}.\lambda\mathcal{Q}.(\exists x)(\mathcal{P}(x) \wedge \mathcal{Q}(x))\rangle \\ \text{‘man’} & : & \langle \mathtt{man}, n, \lambda x.\mathsf{man}'(x)\rangle \\ \text{‘walks’} & : & \langle \mathtt{walks}, np\backslash t, \lambda x.\mathsf{walk}'(x)\rangle \end{array}$$

Notice that the name of the sign (eg ‘man’) can be anything, even a number. What we can actually see of the sign is its exponent, ie `man`. (We write in typewriter font the exponent of a sign. The symbols in typewriter font are therefore true letters in print, while any other symbol is only proxy for some letter or string. Typewriter fonts are our device for quoting a string in running text without having to use quotation marks. Notice that ‘man’ is used to refer to the complete sign rather than its exponent.)

Definition 2 *A (E, T, M)–grammar G consists of a finite signature Ω and functions I_E, I_T and I_M such that $\mathfrak{E} := \langle E, I_E \rangle$, $\mathfrak{T} := \langle T, I_T \rangle$ and $\mathfrak{M} := \langle M, I_M \rangle$ are partial Ω–algebras. The* ***language*** *of G, $L(G)$, is defined by $L(G) := [\varnothing]$. We call F the set of* ***modes*** *of G. A mode is* ***proper*** *if it has nonzero arity. The* ***lexicon*** *of G is the set $\{\langle I_E(f), I_T(f), I_M(f)\rangle : \Omega(f) = 0\}$.*

This definition needs some exegesis. We think of E, T and M as independent sets, given in advance. At present, there are no conditions on these sets. A language is any subset of $E \times T \times M$. A grammar is a system by which the signs of L are built from some finite set of signs (the lexicon) by means of some finite set of functions (called proper modes). Therefore, modes correspond to grammatical rules. For example, if f is a binary mode, one thinks of f as the following grammatical rule

$$f(\sigma_1, \sigma_2) \rightarrow \sigma_1 \quad \sigma_2$$

Notice that terminal symbols of a grammar are generally not considered to be rules (they are part of the lexicon), but there is no harm in thinking of them as rules of the form $\sigma \rightarrow .$ (think of the arrow as the := in `Prolog`). Notice that our definition of grammar is modular: not any set of functions from signs to signs qualifies. Rather, we assume that the modes operate independently on the exponents, the types and the meanings of the signs. Therefore, in order to define a grammar, one needs only to specify the interpretation of the modes in each of the three sets E, T and M independently. This defines the algebras $\mathfrak{E}$, $\mathfrak{T}$ and $\mathfrak{M}$. The rest is completely determined. One forms the product $\mathfrak{E} \times \mathfrak{T} \times \mathfrak{M}$. This algebra contains a unique minimal subalgebra. Its carrier set is precisely the set of all signs that can be produced from the lexicon by the set of proper modes. If L has a grammar, it is also a partial Ω–algebra $\mathfrak{L}$ in some canonical way. We should note that if L is a (E, T, M)–language, it is also a (E', T', M')–language for any $E' \supseteq E$, $T' \supseteq T$ and $M' \supseteq M$. However, there is a different sense of extension of a language that will play a role in the discussion.

Definition 3 *Let L be a $(E, T.M)$–language and L' a (E', T', M')–language. L'* ***extends*** *L if $L' \cap E \times T \times M = L$.*

If L' extends properly L, one may think of L' as having *hidden signs*. Such a sign is of the form $\langle e, t, m\rangle$ where either $e \notin E$ or $t \notin T$ or $m \notin M$. It may be difficult to argue for a sign with exponent $e \in E$ to be hidden. Anyhow, we shall argue below that one should not postulate hidden signs. But for the moment, we leave this possibility open. The following definition embodies the notion of compositionality *as employed in the literature.*

Definition 4 *Let L be a (E, T, M)–language. L is* ***naturally compositional*** *if there is a (E, T, M)–grammar G such that $L(G) = L$. L is* ***compositional*** *if there is a L' which extends L and is naturally compositional.*

Notice that the notion of extension allows the introduction of new exponents or new types or new meanings if necessary. However, no new (E, T, M)–signs may be introduced. To see why this is the standard approach, notice that the problem is stated as follows. A language is defined a subset L of $E \times M$. The pair $\langle e, m\rangle$ corresponds to a sentence e with meaning m. With this datum, a grammar is written, introducing new types (for intermediate constituents) and assigning arbitrary meanings to signs containing them. However, in natural languages the natural parts of a sentence — the constituents — do have their own meaning, and so we cannot choose that meaning arbitrarily. Moreover, the meaning of the constituent contributes to the meaning of the sentence. Furthermore, a context–free grammar is often defined as a string generating device of the following form. The symbol $\mathtt{S}$ is the start symbol. Starting with $\mathtt{S}$, one may replace step by step each nonterminal by the right side of an appropriate rule. In our view, however, this approach suffers from a confusion between the type of a sign and its exponent. The symbol $\mathtt{NP}$ is not part of the string–algebra, which contains only sequences of terminal strings. Hence, $\mathtt{NP}$ is a type. In the course of a derivation we actually derive triples $\langle x, \mathtt{NP}, m\rangle$, where x is a string of type $\mathtt{NP}$ and meaning m.

Nevertheless, if the set of types is finite, $\mathfrak{L}$ may in fact be thought of as a many sorted algebra over sound–meaning pairs. However, the introduction of the types has the advantage to eliminate having to type the signs, and it allows to incorporate grammars with explicit type constructors, such as categorial grammars. If $\cdot$ is a mode, we write $\cdot_e$ in place of $I_E(\cdot)$, $\cdot_t$ in place of $I_T(\cdot)$ and $\cdot_m$ in place of $I_M(\cdot)$. By our definitions, $\cdot$ is defined on a tuple of signs exactly when all three functions $\cdot_e$, $\cdot_t$ and $\cdot_m$ are defined on the corresponding projections. This means that the partiality is introduced by the algebras of exponents, signs and meanings. One fundamental assumption is made.

> *Feasibility.* If $\cdot$ is a mode, then the corresponding functions $\cdot_e$, $\cdot_t$ and $\cdot_m$ are computable.

Moreover, in Montague Grammar the functions are actually polynomials in the basic functions of the algebras of strings, types and meanings, respectively. We shall assume the same here. This means that they can be expressed in λ–notation, but we shall suppress λ–notation whenever possible.

> *Polynomial Feasibility.* If $\cdot$ is a mode, then the corresponding functions $\cdot_e$, $\cdot_t$ and $\cdot_m$ are polynomial functions. Moreover, each of the functions is computable in polynomial time.

Notice that it is not clear that a polynomial function is computable in polynomial time. If, say, $\mathfrak{M}$ consists in the domain ω of natural numbers and a unary function $f : \omega \to \omega$ which is not computable, then f is a polynomial function but is not polynomially computable. The results rely only partly on this restriction of polynomial time computability.

In the simplest case, the grammar has only one mode, $\bullet$, called the *merge*. It is given by the functions $\bullet_e$, $\bullet_t$ and $\bullet_m$, which are defined by

$$\begin{array}{rcl}
\varepsilon(\sigma_1 \bullet \sigma_2) & = & \varepsilon(\sigma_1) \bullet_e \varepsilon(\sigma_2) \\
\tau(\sigma_1 \bullet \sigma_2) & = & \tau(\sigma_2) \bullet_t \tau(\sigma_2) \\
\mu(\sigma_1 \bullet \sigma_2) & = & \mu(\sigma_1) \bullet_m \mu(\sigma_2)
\end{array}$$

Let $\bullet_e$ be concatenation of strings with a blank ($\Box$) inserted, $\bullet_t$ be slash–cancellation and $\bullet_m$ be function application, and we get Montague–Grammar. To give an example, we may compose the signs 'a' and 'man'. This gives the sign $\sigma :=$ 'a' $\bullet$ 'man'. Now,

$$\begin{aligned} \varepsilon(\sigma) &= \varepsilon(\text{`a'}) \bullet_e \varepsilon(\text{`man'}) \\ &= \mathtt{a}^\frown\Box^\frown\mathtt{man} \\ &= \mathtt{a\ man} \end{aligned}$$

Likewise, $\tau(\sigma) = np$ and $\mu(\sigma) = \lambda \mathcal{Q}.(\exists x)(\mathsf{man}'(x) \wedge \mathcal{Q}(x))$ are established. Hence we get

$$\sigma = \langle \mathtt{a\ man}, np, \lambda \mathcal{Q}.(\exists x)(\mathsf{man}'(x) \wedge \mathcal{Q}(x))\rangle$$

A word on notation. We shall assume that the exponents of signs are strings or sequences thereof. These strings are represented in exactly the same way as they are written. So, each word is actually already a sequence (of letters) and concatenation of words is done by putting a blank in between the two. We use $\Box$ to denote the blank. Plain concatenation is denoted by $^\frown$, or, if no confusion arises, it is denoted simply by concatenation. Word concatenation is denoted by $\cdot$. We have $x \cdot y = x^\frown\Box^\frown y$.

Suppose that G generates L. Since $L = [\varnothing]$, any sign of the language can be represented by a constant term. This term is called its *representing term*. Terms uniquely designate derivations. So, if for a given exponent x there are m terms representing a sign with exponent x then x is said to be *m–fold structurally ambiguous*. The reader should bear in mind that the term denotes the signs only with the grammar being given.

The definitions allow that the exponents of signs may be anything we please. Yet, if by exponent we mean the visible (or audible) exponent of the sign, there is little sense in assuming that the exponents of signs are anything but strings. This is, with a minor adaptation, what we shall assume here. It has been shown that given any recursively enumerable language and any computable function assigning meanings to strings, there is a type assignment and a compositional grammar generating this language (see [6] for details). However, there is a sense in which some of these grammars may fail to be compositional in an intuitive sense. We shall therefore say that a grammar G is a *strict grammar of L* if it satisfies the following two requirements. (Two other principles will still follow, but we assume them to follow from the next two, if not in the literal sense, then at least in spirit.)

Naturalness. $L(G) = L$.

Analyticity. There is a finite set A such that $E \subseteq A^*$. For each mode f and each sequence $\boldsymbol{\sigma}$ on which f is defined there is a string polynomial $p(\boldsymbol{x})$, in which each $\boldsymbol{x}_i$, $i < \Omega(f)$, occurs at least once, such that

$$f_e(\varepsilon(\boldsymbol{\sigma})) = p(\varepsilon(\boldsymbol{\sigma}))$$

(Here, $\varepsilon(\boldsymbol{\sigma}) = \langle \varepsilon(\sigma_i) : i < \Omega(f)\rangle$. Notice that $f_e(\varepsilon(\boldsymbol{\sigma})) = \varepsilon(f(\boldsymbol{\sigma}))$, by the assumption that f is defined on $\boldsymbol{\sigma}$.)

Definition 5 *A language L is* ***strictly compositional*** *if it has a strict compositional grammar.*

These principles need a certain amount of motivation. The principle of *Naturalness* is stronger than requiring that $L(G)$ extends L', which is commonly used in the definitions of compositionality. If G is natural for L, L is actually naturally compositional, as defined earlier. So, notice that we require that all signs that the grammar generates are part of the language. In particular, the signs generated have exponents that are units of the language, and the grammar assigns those types and meanings to them that they have in that language. What signs there are is of course an empirical question. It is to a large extent also a question of methodology or personal persuasion. Not everyone will accept that λ–terms count as meanings for the words of natural language. However, even if the existence and nature of signs is unclear the question of what they are is not without meaning at all. Just like Augustinus observed with respect to time, there is a perennial problem with respect to meaning. It seems that we know perfectly well what meaning is, but when asked to give a definition, we fail. The present discussion is therefore not targeted at the question of what signs there are; rather, granted that we know that, how can a grammar for them look like? To emphasize our point once more: if the question whether languages are compositional is to have any nontrivial meaning at all, it is because we exclude the introduction of new signs.

The *Principle of Analyticity* means the following. If a sign σ is directly derived from σ_i, $i < n$, then the exponents of the σ_i are disjoint parts of the exponent of σ. We have phrased this using the language of string polynomials. It should be stressed that the polynomial may depend on the mode as well as the sequence of signs. However, if it is independent of the choice of the signs, we call the grammar *uniformly analytic*.

Definition 6 *A grammar is* ***uniformly analytic*** *if for every mode f there exists a string polynomial p, in which each variable x_i, $i < \Omega(f)$, occurs at least once, such that for all sequences $\boldsymbol{\sigma}$ of signs on which f is defined the equation*

$$f_e(\varepsilon(\boldsymbol{\sigma})) = p(\varepsilon(\boldsymbol{\sigma}))$$

holds.

However, the interpretation of the mode in A^*, f_e, need not be identical to p, it may just be a subset of p (since f_e may be a strictly partial function). In fact, *Uniform Analyticity* comes down to the existence of a polynomial p of the appropriate kind such that $f_e = p \restriction dom(f_e)$.

Example 1. To illustrate our point we shall discuss the example of Kazmi and Pelletier in [7]. Let $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, be the alphabet and let the set of exponents be $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{ca}, \mathtt{cb}\}$. Now we introduce a meaning assignment, which has the property that $\mu(\mathtt{a}) = \mu(\mathtt{b})$ but $\mu(\mathtt{ca}) \neq \mu(\mathtt{cb})$. Now, suppose first that there is only one type of expression. Then it is easily seen that there exists no strictly compositional grammar for that language. This is independent of whether we assume *Polynomial Feasibility*. For there simply is no function $\cdot_\mu$ whatsoever that satisfies

$$\begin{array}{rcl}\mu(\mathtt{ca}) & = & \mu(\mathtt{c}) \cdot_\mu \mu(\mathtt{a}) \\ \mu(\mathtt{cb}) & = & \mu(\mathtt{c}) \cdot_\mu \mu(\mathtt{b})\end{array}$$

So, restricting the class of functions to exclude this example — as Zadrozny proposes — is not necessary at all under the assumption of strict compositionality. We shall briefly outline two alternatives to the present example to shed light on the kind of restrictions that are operative here. First, we could have said that `a`, `b`, `c`, `ca` and `cb` are the exponents of primitive signs and that there is no proper mode, just five 0–ary modes. This is unintended in the example above, since we write `ca` to imply that the sign with exponent `ca` is composed from the signs with exponents `c` and `a`, respectively. Notice however that many languages have words that can be segmented as strings into 'unnatural' parts, in which case the impossibility to have a compositional account of the meaning of that word does simply not arise since the word as a sign is not conceived of as consisting of these parts. To give an example, the word 'selfish' is not the result of composing 'sell' and 'fish', although that (almost) sounds the same. Also the word 'caterpillar' is not a compound built from '(to) cater' and 'pillar'. Likewise, idioms must be considered as units from a semantic point of view. Evidently, they can often also be read literally (in which case they are decomposed), but this is not at issue here. Thus, the notion of a basic sign (ie a 0–ary mode) is different from the notion of a sign having a nonsegmentable exponent. A second variation on that theme is to allow the same string to have several types. We could, for example, allow any string to denote itself when of type, say, s. In that case we have two signs with the same exponent, for example

$$\langle \mathtt{a}, t, \mu(\mathtt{a})\rangle, \langle \mathtt{a}, s, \mathtt{a}\rangle$$

In this situation, a compositional grammar in the strict sense once again exists. Just assume two binary modes $\circ$ and $\bullet$:

$$\begin{array}{rcl}\langle x, s, m\rangle \circ \langle y, s, n\rangle & = & \langle xy, s, m^\frown n\rangle \\ \langle x, s, m\rangle \bullet \langle y, s, n\rangle & = & \langle xy, s, \mu(m^\frown n)\rangle\end{array}$$

So, we can either compose two strings qua strings, in which case we do concatenation on the semantic side. Or we can compose strings qua meaningful entities — in which case we shall step from the string to the corresponding meaning. (Actually, $\bullet$ alone would have sufficed here. One can also introduce a unary mode that transports a string to its meaning.) In this way, once again a compositional grammar exists for any language whatsoever. Or so it seems. What should be observed, however, is that we are not giving a grammar for the same language, since a language is a set of signs, and we have expanded the set of sings to include strings of type s. But how about human languages? We claim that in human languages it is not possible to have a string denote its usual meaning in addition to denoting itself. In the latter case we call the string quoted. There exist devices that allow to quote a string qua string, so actually we *do* have strings as meanings in our language. (Just anything can be the meaning of an expressions.) But the string x can never denote itself (ie x). Rather it must occur in a context that quotes it. For example, we distinguish in writing between `man` — the exponent of a sign whose meaning is, say, $\lambda x.\mathtt{man}'(x)$ — and `'man'` (using object language quotation marks here), the exponent of a sign whose meaning is `man`. So, it is never x itself that denotes x but, for example, the string x enclosed in quotes. Of course, it is only natural for a semiotic system not to allow for the possibility that its signs are all self–denoting, which would mean that we probably didn't need the signs in the first place. And precisely this saves us from vacuity.

The *Principle of Analyticity* is needed to be able to talk meaningfully about parts of a sign. Hence, we can talk about constituents and of a constituent analysis. As the principle stands, it is not unproblematic. First, we shall have to talk of occurrences of strings. The *Principle of Analyticity* is to be understood to require that $\varepsilon(\sigma)$ is made from the $\varepsilon(\sigma_i)$, and that each $\varepsilon(\sigma_i)$ occurs at least once. So, there is no deletion of any material whatsoever. Assuming for the moment that the exponents are strings, the function $\cdot_e$ shall simply be a semigroup polynomial in n–variables such that each variable occurs at least once. Examples are

$$p(x,y) := xxy, \qquad q(x,y) := yxy\mathtt{a}$$

Here, $\mathtt{a}$ is a certain constant (the letter $\mathtt{a}$). The *Principle of Analyticity* does allow for empty categories. However, we shall apply Occam's Razor and assume

Nonemptyness. No sign has empty exponent.

Now follow some more examples of grammars.

Example 2. Numbers are written as sequences of digits, where a digit is a member of $\{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}\}$. The sequence $x_{k-1}x_{k-2}\ldots x_0$ represents the number $\sum_{i<k} \mu(x_i) \cdot 10^i$, where $\mu(x_i)$ is the number assigned to each digit. There are therefore two types: *digits* (D) and *sequences* (S). The following is a strict grammar. These are the nullary modes:

$$\begin{array}{rcl} \text{`0'} & : & \langle \mathtt{0}, D, 0\rangle \\ \text{`1'} & : & \langle \mathtt{1}, D, 1\rangle \\ & & \cdots \\ \text{`9'} & : & \langle \mathtt{9}, D, 9\rangle \end{array}$$

There is a unary mode $*$ and a binary mode $\circ$. $*_e(x) := x$, $*_m(\mu) := \mu$, and $*_t(D) := S$. $*_t(S)$ is undefined. For the binary mode $\circ$ we have $x \circ_e y := xy$, $S \circ_t D := S$. $\circ_e$ is undefined otherwise. $m \circ_m n := 10m + n$. So, the term $*$'7' corresponds to the sign $\langle \mathtt{7}, S, 7\rangle$, the term '2' $\circ$ ('3' $\circ$ '7') to the sign $\langle \mathtt{237}, S, 237\rangle$.

Since the algebra of types is finite, one can actually present the algebra of signs in form of the following context–free grammar.

$$\begin{array}{lcl} \langle \mathtt{0}, D, 0\rangle \mid \ldots \mid \langle \mathtt{9}, D, 9\rangle & \to & . \\ \langle xy, S, 10 \cdot m + n\rangle & \to & \langle x, S, m\rangle \quad \langle y, D, n\rangle \\ \langle x, S, n\rangle & \to & \langle x, D, n\rangle \end{array}$$

(Here, $\mid$ denotes an alternative; the first line abbreviates therefore a total of ten rules.) Terminal symbols are denoted here by nullary rules. This grammar is left–regular.

EXAMPLE 3. As above. However, assign to a sequence x as meaning the pair $\nu(x) := \langle \ell(x), \mu(x)\rangle$.

$$\begin{array}{lll}
\langle \mathtt{0}, D, \langle 1, 0\rangle\rangle \mid \ldots \mid \langle \mathtt{9}, D, \langle 1, 9\rangle\rangle & \to & . \\
\langle yx, T, \langle \ell_1 + \ell_2, 10 \cdot m_1 + m_2\rangle\rangle & \to & \langle x, T, \langle \ell_1, m_1\rangle\rangle \quad \langle y, D, \langle \ell_2, m_2\rangle\rangle \\
\langle x, T, \langle \ell, n\rangle\rangle & \to & \langle x, D, \langle \ell, n\rangle\rangle \\
\langle x, S, n\rangle & \to & \langle x, T, \langle \ell, n\rangle\rangle
\end{array}$$

This is a compositional grammar. However, it is not strict. The problem is that the meaning of a string x simply *is* the number it represents, we are not allowed to add any more to it. It can be shown that there is no right–regular strict grammar that generates the number sequences. For a proof note that the sequences `7`, `07`, `007` etc all have the same meaning, namely 7. However, the result of prefixing `1` is different in all cases. We get `17`, `107`, `1007` etc, which all represent different numbers. However, since we have only finitely many types in a right–regular grammar, some of the sequences `7`, `07`, `007` etc must have equal type, and therefore the grammar generates two different sequences starting with `1`, which are assigned the same number. This is, however, incorrect.

It is tempting to conclude that a strict grammar for the number sequences must be left regular. This is more or less correct, but there are infinitely many grammars that can generate the number sequences compositionally even in the strict sense and each is different in the constituent analysis that it introduces. Nevertheless, the constituent analysis cannot be right regular. Strict grammars must satisfy the condition that not both a string x and $\mathtt{0}x$ are generated having identical type. So, even if the constituent analysis is not uniquely defined by the principles laid out above, there nevertheless are certain things that can be said about possible constituents.

The present definitions do not say anything about the size of the algebra of types. It may be finite or infinite. However, notice that if a grammar has infinitely many signs then the same exponent can in principle have infinitely many meanings depending on which type it has. Moreover, strings can be infinitely ambiguous, simply because they can be derived from some the same string using some unary modes. Since we do not want to exclude the number of types to be infinite, we shall rather require that unary modes must also change the exponent. Although this does not follow from *Analyticity* in a literal sense, it is nevertheless a principle of the same sort, since it requires that every step leaves a visible trace on the exponent.

> *Productivity.* If σ is composed from τ by applying a unary mode, then $\varepsilon(\tau)$ is shorter than $\varepsilon(\sigma)$.

This means that there can be no unary rules that only change the category (type) and the meaning; rather, a unary mode must introduce material into the string. The grammar in Example 2 does not comply with this restriction. There is an easy fix for that. Just assume the following additional 0–ary modes:

$$\begin{array}{rcl} \text{`}0^{\#}\text{'} & : & \langle \mathtt{0}, S, 0\rangle \\ \text{`}1^{\#}\text{'} & : & \langle \mathtt{1}, S, 1\rangle \\ & \cdots & \\ \text{`}9^{\#}\text{'} & : & \langle \mathtt{9}, S, 9\rangle \end{array}$$

Now eliminate the last rule and add instead the rule

$$\langle xy, S, 10m + n\rangle \quad \to \quad \langle x, D, m\rangle\ \langle y, D, n\rangle$$

This looks like a bad trick but is in fact quite logical. The digit `3` for example, is actually the exponent of two signs, that of the *digit* `3` and that of the *sequence* consisting of `3`. This grammar reflects the dual nature of sequences of length 1. Sequences of length > 1 can of course not be digits.

There is plenty of evidence that in language there are empty signs and also non productive modes. However, their use must obviously be highly restricted otherwise the determination of the meaning from the sound can become infeasible. So, when one looks closely at the matter it often enough appears that the use of empty signs and non productive modes can be eliminated in much the same way as it can be done in context free grammars.

Example 4. This example concerns the English number names, in a slightly simplified form. This example goes back to Arnold Zwicky, for a discussion of the formal complexity of English and Chinese number names see [11]. Each language has a largest primitive name for a number. This number varies from language to language. Let us assume that it is `million` for English. Numbers of the form 10^{6k} are represented by the k–fold iteration of the word `million`. The number 2000003000005 is therefore represented by the string

`two million million three million five`

We shall leave out the words `thousand`, `hundred` as well as `ten`, `twenty` etc and assume that our alphabet consists only of

$$\{\mathtt{zero}, \mathtt{one}, \mathtt{two}, \ldots, \mathtt{nine}, \mathtt{million}\}$$

Legitimate expressions have the following form:

$$x_0 \cdot \mathtt{million}^{k_0} \cdot x_1 \cdot \mathtt{million}^{k_2} \cdot \ldots \cdot x_{m-1} \cdot \mathtt{million}^{k_{m-1}}$$

where $k_0 > k_1 > \ldots > k_{m-1}$ and $x_j \neq \mathtt{million}$ for all $j < m$. This sequence represents the number

$$\sum_{j=0}^{m-1} x_j \cdot 10^{6k_j}$$

This language is not generable by Linear Context Free Rewrite System (otherwise known as LCFRSs); in fact, it is not even semilinear. However, it is recognizable in polynomial time. We shall propose two quite similar grammars. Each of the two have the following 0–ary modes:

$$
\begin{array}{lcl@{\qquad}lcl}
\text{`0'} & : & \langle \mathtt{zero}, D, 0\rangle & \text{`}0^{\#}\text{'} & : & \langle \mathtt{zero}, S, 0\rangle \\
\text{`1'} & : & \langle \mathtt{one}, D, 1\rangle & \text{`}1^{\#}\text{'} & : & \langle \mathtt{one}, S, 1\rangle \\
 & & \dots & & & \dots \\
\text{`9'} & : & \langle \mathtt{nine}, D, 9\rangle & \text{`}9^{\#}\text{'} & : & \langle \mathtt{nine}, S, 9\rangle \\
\text{`m'} & : & \langle \mathtt{million}, M, 10^6\rangle & \text{`}0^{\dagger}\text{'} & : & \langle \mathtt{zero}, SE, 0\rangle \\
 & & & \text{`}1^{\dagger}\text{'} & : & \langle \mathtt{one}, SE, 1\rangle \\
 & & & & & \dots \\
 & & & \text{`}9^{\dagger}\text{'} & : & \langle \mathtt{nine}, SE, 9\rangle
\end{array}
$$

Here, D is the type of a digit, S that of a number expression, and SE that of a *simple expression*, where a simple expression is of the form $x \cdot \mathtt{million}^k$, x a digit. Now, there is a binary mode(s) $\bullet$ and $\bullet^S$, operate as follows:

$$
\begin{array}{rcl}
\langle x, SE, m\rangle \bullet \ \langle y, M, n\rangle & := & \langle x \cdot y, SE, m \cdot n\rangle \\
\langle x, SE, m\rangle \bullet^S \langle y, M, n\rangle & := & \langle x \cdot y, S, m \cdot n\rangle
\end{array}
$$

Finally, the mode $\circ$ is defined as

$$\langle x, S, m\rangle \circ \langle y, SE, n\rangle := \langle x \cdot y, S, m + n\rangle$$

However, as the definitions stand the grammar generates all sequences of simple expressions, not necessarily in decreasing order. To implement the latter restriction, we have two choices: (a) we define $\circ_m$ to be $m \circ_m n := m + n$ if $m > n$, and $m \circ_m n$ undefined otherwise, (b) we define $x \circ_e y := x \cdot y$ if x ends in a larger block of `million` than y, and $x \circ_e y$ is undefined otherwise. The proposals differ slightly. If we disallow expressions of the form $\mathtt{zero}\ \mathtt{million}^k$ then they are actually equivalent, otherwise option (a) gives incorrect results. The trouble with both proposals is that we need to introduce strange partial functions. However, notice that the definition of a grammar did not tell us what the basic functions of the partial algebras are. So, rather than taking a simple merge as the (only) basic operation, we may also take merge functions as basic which require properties of strings to be checked. This requires careful formulation. We shall have to require that these functions be computable in polynomial time, otherwise Theorem 8 below is actually incorrect. In this case, if we are operating on the string algebra, testing whether one string is a substring of the other certainly takes only polynomial time. We come to our main definition:

Definition 7 *A grammar is* ***strictly compositional*** *(or* ***strict****) if it is analytic, polynomially feasible, has no nonempty signs and is productive. A language is* ***strictly compositional*** *if it has a strictly compositional grammar.*

The present restrictions can be shown to be nontrivial. Before we engage in the proof we have to fix a last detail. We shall assume that the algebra operates not necessarily on strings but on sequences of strings. Moreover, these sequences shall have bounded length, say k. It is beyond the scope of this paper to motivate exactly why we depart from the ideal model that the exponents are strings. There are basically two arguments: (a) if we allow only strings then there does not seem to be a feasible algorithm to generate those languages that are not context–free, (b) strings are not continuous but are naturally segmented (eg by pauses). Of course, it is always possible to introduce a boundary marker into the string which would function the same way. We have felt it technically more clean to allow sequences of strings. The relation between the sequences of strings and the strings themselves is fixed by the following requirement.

> *Vectorization.* The string associated with an exponent $\langle x_i : i < n\rangle$ is its concatenation $x_0x_1 \ldots x_{n-1}$.

Hence, we shall assume that the exponents of the grammar depart only mildly from strings, namely, they are allowed to be strings with some gaps. We call a grammar *vectorized* if it uses sequences of strings rather than strings.

Theorem 8 *Let $\mathfrak{A}$ be a strictly compositional vectorized grammar for L. Then the following holds:*

1. *Given a string x it can be decided in polynomial time whether it belongs to the language.*
2. *Given a string x, there are at most exponentially many derivations for x.*

Proof. The algorithm is an adaptation of the chart method. First, notice that any derivation has length at most $|x|$, where $|x|$ denotes the length of x. A representing term for x has therefore at most $|x|$ mode symbols. It is now easy to see that x is at most exponentially ambiguous. Now for the first claim. Let $n := |x|$. By analyticity, the sequence $\langle x_i : i < k\rangle$ must consist of disjoint substring occurrences of x. (This is easily established by induction.) There exist less than n^2 substrings, and hence less than n^{2k} k–sequences of substrings. In the first step, try to match a sequence against the exponent of a 0–ary mode. This takes polynomial time. This gives the list L_0. Now, given L_i, let L_{i+1} be the result of applying all possible modes to the members of L_i and matching the result against x. x corresponds to some exponent of a sign in L iff there is a $\sigma \in L_i$ for some $i < n$ such that the exponent (or its product) is x. Now, computing L_{i+1} from L_i takes time polynomial in n. For there are at most n^{2k} members, and there are finitely many modes. Each application of a single mode takes polynomial time (by polynomial feasibility). We have to compute L_i only for $i < n$. This concludes the proof of the first claim. Q. E. D.

(The polynomial bounds computed here are rather bad. They suffice for the argument, however.) So, strictness *is* restrictive. The question therefore arises: which languages are strict and which ones are not? In particular: are natural languages at all strictly compositional in the sense of the definition? We believe that the answer to the second question is positive. Notice that the rather tight constraints on putting together signs do not allow to dissociate the meaning composition and the string handling. For example, devices such as a Cooper–storage are inadmissible since they dissociate the syntactic structure from the semantic structure by introducing new meanings, something which is strictly prohibited.

It may well be that Categorial Grammar takes us out of that problem by associating enough types with a string to cover its potentials in a context. If we are not so happy with having infinitely many types, however, the only way to surround this is to postulate stronger string handling mechanisms. One promising proposal that has been made recently are the so–called Literal Movement Grammars (LMG) as have been introduced by Annius Groenink in [5]). An LMG has rules of the following form

$$A(\gamma) \to B_0(\delta_0)\ B_1(\delta_1) \ldots B_{n-1}(\delta_{n-1})$$

where A and the B_i are nonterminals, here called *types*, and γ and δ_i sequences of polynomials over the alphabet, possibly using variables. Since there are only finitely many rules, the length of these sequences is bounded. We may therefore assume that they all have the same length, adding empty strings if necessary. The advantage of LMGs is that they add explicit rules for handling the strings. By adding a third dimension, the meaning, we can turn an LMG into a grammar of signs. We call an *interpreted LMG* a grammar that has rules of the form

$$A(\gamma, q(\mu_0, \ldots, \mu_{n-1})) \to B_0(\delta_0, \mu_0)\ B_1(\delta_1, \mu_1) \ldots B_{n-1}(\delta_{n-1}, \mu_{n-1})$$

where μ_i are variables for meanings and q is a polynomial function $\mathfrak{M}^n \to \mathfrak{M}$.

An easy example of an LMG is the following

$$S(xx) \to S(x); \qquad S(\mathtt{a}) \to .$$

This grammar generates the language $\{\mathtt{a}^{2^n} : n \in \omega\}$. It is shown in [5] that any recursively enumerable language can be generated by an LMG. Therefore, these grammars are very powerful. However, certain subclasses can be identified. According to [5], an LMG is *bottom up nonerasing* if each variable on the right hand side occurs at least once on the left hand side. An LMG is *bottom up linear* if each variable on the right hand side occurs at most once on the left hand side. An LMG is *noncombinatorial* if each term on the right hand side consists of a single variable. Finally, an LMG is *simple* if it is *bottom up linear*, *bottom up nonerasing* and *noncombinatorial*. Now the following holds

Theorem 9 (Groenink) *Let $L \subseteq A^*$ be a language. L is **PTIME**–recognizable iff it can be generated by a simple LMG.*

Now, this does not mean of course that a PTIME–recognizable sign grammar can be generated by a simple interpreted LMG. But we shall present evidence below that natural languages can be generated by (more or less) simple LMGs. Notice that in the definition of *bottom up nonerasingness* one thing must be added: first, one should talk of constants, not only variables which occur on the right hand side. However, for noncombinatorial grammars this is obviously unnecessary.

Simple LMGs are not necessarily analytic, but if they are, they are uniformly analytic. For example, a clause of the form

$$A(x) \to B(x)\ C(x)$$

can occur in a simple LMG, but if we read the rules as modes this is unacceptable. The condition of analyticity requires instead that each variable occurs to the left hand side at least as often as it occurs on the right hand side. Furthermore, [5] disallows a variable to occur more than once to the left, but shows that this condition can be circumvented. We shall therefore say that an LMG is *analytic* if it is (a) noncombinatorial and (b) every variable occurs on the left hand side of a production at least as often as it occurs on the right hand side. Languages generated by analytic LMGs are also generated by simple LMGs but the converse need not hold. Notice that our first example of an LMG is actually analytic. However, it is not simple. Yet, it is easily transformed into a simple LMG (see also [5]):

$$\begin{array}{lll} S(xy) & \to & S(x)\ S(y)\ E(x,y); \quad S(\mathtt{a}) \to . \\ E(\mathtt{a},\mathtt{a}) & \to & . \quad ; \quad E(x_1y_1, x_2y_2) \to E(x_1,x_2)\ E(y_1,y_2) \end{array}$$

We notice that in Example 4, assuming that we work with pairs of strings rather than strings, under a suitable reformulation we only need to assume the existence of a binary string function of the following form:

$$p(x,y) := \begin{cases} x & \text{if } x = y \\ \uparrow & \text{else} \end{cases}$$

(Here, $\uparrow$ means that the function is undefined.)

We shall illustrate the potential of this proposal by giving an analysis of two benchmark examples. One is the cross–serial dependencies. The other are the languages with iterated cases. A particular example of the latter kind was analyzed in [8], where it was shown that the language generated in this example was not semilinear, hence not generable by a LCFRS (or a Multi–Component TAG, for that matter). (For a reference on LCFRSs see [13].)

Example 5. The following LMG generates the cross-serial dependencies of Dutch.

$$\begin{array}{lll} VR(\mathtt{zien}) & \to & . \\ VR(\mathtt{laten}) & \to & . \\ NP(\mathtt{Jan}) & \to & . \\ NP(\mathtt{Piet}) & \to & . \\ V(\mathtt{zwemmen}) & \to & . \\ VC(x_1 \cdot y_1, x_2 \cdot y_2) & \to & NP(x_1)\ VR(y_1)\ VC(x_2,y_2) \\ VC(x,y) & \to & V(y)\ NP(x) \end{array}$$

It is straightforward to transform this LMG into a sign grammar. We basically need in addition to the nullary modes, a binary mode $\bullet$ and a ternary mode $\circ$. The binary mode is defined as follows.

$$\langle\langle x_1, x_2\rangle, NP, f\rangle \bullet \langle\langle y_1, y_2\rangle, V, g\rangle := \langle\langle x_1 \cdot x_2, y_1 \cdot y_2\rangle, VC, f(g)\rangle$$

$\circ$ operates as follows.

$$\begin{array}{l} \circ(\langle\langle x_1, x_2\rangle, NP, f\rangle, \langle\langle y_1, y_2\rangle, V, g\rangle, \langle\langle z_1, z_2\rangle, VC, \mathcal{P}\rangle) \\ \quad := \langle\langle z_1 \cdot x_1 \cdot x_2, z_2 \cdot y_1 \cdot y_2\rangle, VC, g(f)(\mathcal{P})\rangle \end{array}$$

The semantics of a raising verb, here `zien` (English 'to see') is as follows:

$$\lambda x.\lambda \mathcal{P}.\lambda y.\mathsf{see}'(y, \mathcal{P}(x))$$

This generates the cross–serial dependencies. If we want to give an analysis of the German verb cluster or of the analogous English construction, we only have to modify the string polynomial $\circ_e$, nothing else needs to be changed. The analysis of cross–serial dependencies is similar to that of Calcagno [2], where a larger fragment is analyzed using head–wrapping, which was introduced in Pollard [10]. Calcagno also discusses the relationship with a proposal by Moortgat [9], who uses string equations. String equations are one way to try to avoid the use of vectorization. It is therefore worthwhile to see why it does not work. Suppose we have the following rule

$$A(u \cdot x, v \cdot y) \to B(u, v)\ C(x, y)$$

Then this must be replaced in Moortgat's system by the following rule:

$$A(p) \to B(q)\ C(r) \quad : \quad p = u \cdot x \cdot v \cdot y, q = u \cdot v, r = x \cdot y.$$

This means that if an A is analyzed as a B plus a C then the strings associated with A, B and C should satisfy the string equations shown to the right. The trouble with string equations, as Calcagno rightly points out, is that they come down to an a posteriori analysis of a single string into several components, which may or may not reflect the actual composition of the string. String equations are simply not analytic (in our sense of the word). Head–grammars and LCFRSs on the other hand mark explicit places for inserting strings, thus reflecting the actual construction of the string rather than just any other. Notice that LMGs allow both for string equations and for vectorization, but string equations are not allowed in an analytic or a simple LMG.

Example 6. We finally discuss the stacked genitives of Old Georgian (see [8]). Old Georgian displays a phenomenon called *Suffixaufnahme* or *Double Case.* Suffixaufnahme is found in many Australian languages, and it is said to be iterable beyond limitation (at least in some languages with respect to certain constructions, to be exact, see [4], [3] for examples). Old Georgian provides a generic case of Suffixaufnahme that is iterable (see Boeder [1]).

`govel-i`	`igi`	`sisxl-i`	`saxl-isa-j`	`m-is`
all-NOM	Art-NOM	blood-NOM	house-GEN-NOM	Art-GEN
	`Saul-is-isa-j`			
	Saul-GEN-GEN-NOM			

All the blood of the house of Saul

We will give a compositional LMG for this construction. However, we will simplify the matter. Full NPs shall have no article, the nominative is always `-j` and the genitive does not appear in its short form `-is`. The grammar manipulates triples $\langle x, y, z\rangle$, where x is the first noun, y its stack of case suffixes and z the remaining NP. First we write the plain LMG. (Recall here that we distinguish plain concatenation ($^\frown$) from word concatenation ($\cdot$).)

$$
\begin{array}{lll}
(1)\ \mathit{NPc}(x, y, \varepsilon) & \rightarrow & \mathit{NP}(x, \varepsilon, \varepsilon) \quad \mathit{Case}(\varepsilon, y, \varepsilon) \\
(2)\ \mathit{Case}(\varepsilon, \mathtt{isa}, \varepsilon) & \rightarrow & . \\
(3)\ \mathit{Case}(\varepsilon, \mathtt{j}, \varepsilon) & \rightarrow & . \\
(4)\ \mathit{NPs}(x, y, z) & \rightarrow & \mathit{NPc}(x, y, z) \\
(5)\ \mathit{NPs}(x, y^\frown\mathtt{isa}, \varepsilon) & \rightarrow & \mathit{NPs}(x, y, \varepsilon) \\
(6)\ \mathit{NPs}(x, y^\frown\mathtt{j}, \varepsilon) & \rightarrow & \mathit{NPs}(x, y, \varepsilon) \\
(7)\ \mathit{NPs}(x_1, y_2, x_2{}^\frown\mathtt{isa}^\frown y_2 \cdot z_2) & \rightarrow & \mathit{NPs}(x_1, y_2, \varepsilon)\ \mathit{NPs}(x_2, \mathtt{isa}^\frown y_2, z_2)
\end{array}
$$

Notice that the suffix sequence in the last rule occurs twice on the left and twice on the right hand side. This grammar is analytic (modulo massaging away unproductive rules). The semantic functions are as follows. For (4), (5) and (6) the corresponding function is the identity. For (1) and (7) it is application of the right argument to the left; (3) has the interpretation $\lambda\mathcal{P}.\mathcal{P}$ ($\mathcal{P}$ of type $\langle e, t\rangle$), but this is done for the purpose of exposition only. (2) has the semantics

$$\lambda\mathcal{P}.\lambda\mathcal{Q}.\mathsf{belong}'(y, x) \wedge \mathcal{P}(x) \wedge \mathcal{Q}(y)$$

For the nouns we take $\lambda x.\mathsf{blood}'(x)$, $\lambda x.\mathsf{house}'(x)$ and $\lambda x.x \doteq \mathsf{saul}'$ as interpretations. They have type *NP*. Our target example is

`sisxlj saxlisaj Saulisaisaj`

Here are the translations of the nouns with stacked genitives:

$$
\begin{array}{lcl}
\mathtt{sisxlj} & : & \lambda x.\mathsf{blood}'(x) \\
\mathtt{saxlisaj} & : & \lambda\mathcal{Q}.\mathsf{belong}'(y, x) \wedge \mathsf{house}'(x) \wedge \mathcal{Q}(y) \\
\mathtt{Saulisaisaj} & : & \lambda\mathcal{Q}.\mathsf{belong}'(y, x) \wedge x \doteq \mathsf{saul}' \wedge \mathcal{Q}(y)
\end{array}
$$

Notice that only the inner layer of case marking is semantically operative. The outer layers do not change the meaning. Now we compose these words. By the rules of the grammar, only the last two words can be analyzed as a constituent:

$$\lambda\mathcal{Q}.\mathsf{belong}'(y, x) \wedge x \doteq \mathsf{saul}' \wedge \mathsf{belong}'(y, z) \wedge \mathsf{house}'(x) \wedge \mathcal{Q}(z))$$

This can now be composed with the first word; this gives the following translation for our example.

$$\mathsf{belong}'(y, x) \wedge x \doteq \mathsf{saul}' \wedge \mathsf{belong}'(z, y) \wedge \mathsf{house}'(y) \wedge \mathsf{blood}'(z)$$

For this to work properly, substitution must be defined correctly. That we have free variables here is just an artifact of the simplicity of the fragment and could of course be avoided.

The last example also demonstrates how languages with stacked case marking can be treated. However, we shall note here that LMGs cannot handle such languages if they have completely free word order. It has been confirmed by Alan Dench (p. c.) that those languages which have the most extensive iterated case marking system do not allow for free word order beyond the clause boundary. Given that free word order within a clause can in principle be accounted for compositionally using LMGs — as we believe — this gives evidence that LMGs have enough string handling capacity. To show this is however beyond the scope of this paper. We shall only note that languages with stacked cases cannot simply be treated using a function that compares strings of cases, since the exponents of cases may actually be different. This means that the string handling component must in these examples rely on rather delicate functions, which are however computable in linear time. [12] has argued that German allows scrambling across any number of clause boundaries. If that is so, German could also not be handled by an interpreted LMG compositionally (in the strict sense). There is however every reason to believe that the arguments cannot follow in any order whatsoever. Rather, free word order is only present when the arguments are sufficiently distinguishable either morphologically (by their case endings) or semantically (animate vs inanimate). Otherwise, we claim, word order is fixed. If we are right, also German is not so exceptional after all.

References

1. Winfried Boeder. Suffixaufnahme in Kartvelian. In Frans Plank, editor, *Double Case. Agreement by Suffixaufnahme*, pages 151 – 215. Oxford University Press, 1995.
2. Mike Calcagno. A Sign–Based Extension to the Lambek Calculus for Discontinuous Constituents. *Bulletin of the IGPL*, 3:555 – 578, 1995.
3. Alan Dench. Suffixaufnahme and Apparent Ellipsis in Martuthunira. In Frans Plank, editor, *Double Case. Agreement by Suffixaufnahme*, pages 380 – 395. Oxford University Press, 1995.
4. Nicholas D. Evans. *A Grammar of Kayardild. With Historical–Comparative Notes on Tangkic.* Mouton de Gruyter, Berlin, 1995.
5. Annius Groenink. *Surface without Structure. Word order and tractability issues in natural language processing.* PhD thesis, Utrecht University, 1997.
6. Theo Janssen. Compositionality. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 417 – 473. Elsevier, Amsterdam, 1997.
7. A. Kazmi and F. J. Pelletier. Is Compositionality Formally Vacuous? *Linguistics and Philosophy*, 21:629 – 633, 1998.
8. Jens Michaelis and Marcus Kracht. Semilinearity as a syntactic invariant. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics (LACL '96)*, number 1328 in Lecture Notes in Computer Science, pages 329 – 345, Heidelberg, 1997. Springer.
9. Michael Moortgat. Generalized quantifiers and discontinuous type constructors. In W. Sijtsma and A. van Horck, editors, *Discontinuous Constituency.* Mouton de Gruyter, Berlin, 1993.
10. Carl J. Pollard. *Generalized Phrase Structure Grammar, Head Grammars and Natural Language.* PhD thesis, Stanford University, 1984.
11. Daniel Radzinski. Chinese Number Names, Tree Adjoining Languages, and Mild Context–Sensitivity. *Computational Linguistics*, 17:277 – 299, 1991.

12. Owen Rambow. *Formal and Computational Aspects of Natural Language Syntax.* PhD thesis, University of Pennsylvania, 1994.
13. Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context–free grammars. *Theoretical Computer Science*, 88:191 – 229, 1991.
14. Dag Westerståhl. On Mathematical Proofs of the Vacuity of Compositionality. *Linguistics and Philosophy*, 21:635 – 643, 1998.
15. Wlodek Zadrozny. From Compositional Semantics to Systematic Semantics. *Linguistics and Philosophy*, 17:329 – 342, 1994.