Centrum voor Wiskunde en Informatica
REPORTRAPPORT

Mild context-sensitivity and tuple-based generalizations of context-free grammar

A.V. Groenink

Computer Science/Department of Interactive Systems

CS-R9634 1996

Report CS-R9634
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Mild Context-Sensitivity and Tuple-based Generalizations of Context-Free Grammar

Annius V. Groenink (avg@cwi.nl)

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

This paper classifies a family of grammar formalisms that extend context-free grammar by talking about *tuples* of terminal strings, rather than independently combining single terminal words into larger single phrases. These include a number of well-known formalisms, such as head grammar and linear context-free rewriting systems, but also a new formalism, *(simple) literal movement grammar*, which strictly extends the previously known formalisms, while preserving polynomial time recognizability.

The descriptive capacity of simple literal movement grammars is illustrated both formally through a weak generative capacity argument and in a more practical sense by the description of conjunctive cross-serial relative clauses in Dutch. After sketching a complexity result and drawing a number of conclusions from the illustrations, it is then suggested that the notion of *mild context-sensitivity* currently in use, that depends on the rather loosely defined concept of *constant growth*, needs a modification to apply sensibly to the illustrated facts; an attempt at such a revision is proposed.

AMS Subject Classification (1991): 03D10, 68Q15, 68Q45, 68Q50, 68Q52, 68S05

CR Subject Classification (1991): D3.1, F1.2, F1.3, F4.2, F4.3

Keywords & Phrases: Grammar formalisms; mild context-sensitivity; complexity of language recognition; weak generative capacity; conjunctions and crossed dependencies

Note: The author is supported by SION grant 612-317-420 of the Netherlands Organization for Scientific Research (NWO). To appear in a special edition of *Linguistics and Philosophy* of papers read at the fourth Mathematics of Language meeting in Philadelphia, October 1995.

1. Introduction

In recent years there has been considerable interest for 'light' grammar formalisms aimed at describing linguistic *structure* only, whose descriptive capacity is minimally larger than that of a context free grammar. A well known example is *tree adjoining grammar* (TAG, see e.g. [Wei88]); less common are *linear indexed grammar* (LIG), *head grammar* (HG, [Pol84]) and *combinatory categorial grammar* (CCG). HG, LIG, CCG and TAG form a mutually equivalent group [VSW94] in a hierarchy of languages described by a tuple-based formalism called *linear context-free rewriting systems* (LCFRS, [Wei88]); this class is further extended by *parallel multiple context-free grammars* (PMCFG) in [KNSK92].

One of the motivations for such light grammar formalisms is that in order to study the real, theory-independent nature of linguistic structure, it is valuable to investigate precisely how much formal power is required for an adequate structural description of (configurationally oriented) natural languages.

There have been many arguments that the underlying structure of natural languages is beyond the descriptive capacity of context-free languages; papers such as [MR87] and [Rad91] show that even stronger formalisms like TAG can be considered inadequate. Joshi [Jos85] proposed an informal outline of a class of languages structural linguistic investigations could be limited to: a *mildly context-sensitive* (MCS) *language* is one that has the following three properties

1. limited crossed dependencies
2. constant growth
3. polynomial parsing

where a language L is *constant growth* if there is a constant c_0 and a finite set of constants C such that for all $w \in L$ where $|w| > c_0$ there is a $w' \in L$ such that $|w| = |w'| + c$ for some $c \in C$. One can think of constant growth as a weak, alternative form of a pumping lemma.

The class of mildly context-sensitive languages seems to be most adequately approached by LCFRS. This paper will show, along the lines of an argument in [MR87], how "limited crossed dependencies" probably implies that even a stronger formalism like LCFRS is not fully adequate for the description of basic linguistic structure.[1] I will define a framework which captures the grammar formalisms currently known to satisfy the MCS constraint, and a few stronger ones, and then observe how some of the stronger formalisms could possibly qualify for 'describing structurally essential phenomena' outside the class of the mildly context-sensitive languages as currently defined. I proceed with an outline of a discussion about alternative definitions of language classes aimed at a refinement of the notion of MCS.

1.1 Dependency and Discontinuous Constituency

It is well known that natural languages exhibit dependency between different phrases within a sentence that are not 'next to each other'. Some examples are nominal topicalization, auxiliary inversion and Dutch/Swiss German crossed dependencies.

I take as a point of departure the idea that phrases that depend on each other, i.e. need to be checked against each other by a parser in some potentially unbounded manner, should be 'close to each other' in the sense that there should be a structural analysis (say a tree) in which these components are dominated by the same node. In other words, I consider the feasibility of a minimal, one-level *domain of locality.*

This is equivalent to saying that we try and keep *dependency* local to *constituents.* This has repercussions on the notion of a constituent, which varies over different linguistic theories. The approach taken in this paper is one of *discontinuous constituency.* This may not be easy to accept if the reader is used to the views imposed by other linguistic paradigms; in transformational accounts of language, the meaning of the word *constituent* varies depending on what structure one is looking at (s-structure, d-structure, LF); in LFG, the sentence corresponding to a *c-structure* (assuming that the **c** stands for constituent) is read off the context-free tree analysis from left to right, hence a constituent in LFG always seems to form an uninterrupted substring of a sentence, and the existence of *discontinuous* constituents is meaningless if not contradictory.

Furthermore the grammar formalisms presented in this paper explore how far we can get without attaching more than one tree-shaped analysis to the same sentence, which is done most abundantly in GB-style approaches, but also for example in TAG. For many discontinuity and movement phenomena, it seems that a single tree or graph like analysis can characterize a sufficient amount of both logical and deep structure; the surface form is then obtained by nonstructural, compositional string-generating operations over this tree analysis (this is somewhat similar to the way strings are produced in a Montague-style approach).

[1] Another inadequacy (inability to describe Chinese number names) of LCFRS is discussed in [Rad91].

1.2 Examples

A popular example of what is called a discontinuous constituent in the literature is the phrase *easy · to please* in (1.1). The underlying idea is that there is a generally accepted tree structure inspired by functional concepts such as heads, complements and modifiers. Constituents are then obtained by putting together the words at the leaves of arbitrary subtrees. A sensible account of English will recognize *easy to please* as an adjectival phrase that modifies the nominal projection *person*, hence *person* and *easy to please* must be represented by separate subtrees, and they must be separate constituents.

(1.1) Kim is an **easy** person **to please**.

(1.2) Kim is **easy · to please**.

Another way of reasoning is that we want the adjectival phrase *easy to please* in sentence (1.1) to be represented in the same way as in (1.2). There are roughly two explanations in existence. The transformational view is that *to please* is *extraposed* or *moved* rightward out of its deep-structural position. The other explanation will speak of a phenomenon of *discontinuous constituency* and says that the constituent *easy to please* will *wrap* itself around a noun phrase when it modifies it. Let's say that (*easy*, *to please*) is a constituent whose surface form consists of two *clusters* which may wrap themselves around other words in a full sentence.

Now consider sentences (1.3) and (1.4).

(1.3) **Did** Eve **eat the apple**

(1.4) Eve **did · eat the apple**

An elegant description of these would probably consider the phrase headed by *eat* as a complement to *did*, and assign the same (deep-) structural analysis to *did eat the apple* in both examples. Again, this might equally well be explained as movement of the verb *did* or by saying that (*did*, *eat the apple*) is a discontinuous VP constituent that can wrap itself around its subject.

The same remarks can be made about topicalized sentences. (1.5–1.7) are from [Pol84] on head grammar:

(1.5) Smith **sent Jones to Minsk**

(1.6) **Minsk**, Smith **sent Jones to ·**

(1.7) **Jones**, Smith **sent · to Minsk**

Here we could also pose that *sent Jones to Minsk* should be the same constituent in all three sentences. But the *wrapping* analysis is no longer suitable for these examples; we cannot split up the VP into less than 4 clusters, and there is hardly a common order in the 3 examples, except that *sent* precedes *to*. While e.g. a head grammar will be able to describe the previous two examples, this last example is already beyond its descriptive scope (one would need to add a SLASH feature to describe these different topicalized sentences [Pol84]).

Finally, a very challenging structural phenomenon has always been that of *crossed dependencies* in Dutch and Swiss German relative clauses; as opposed to the simple English verb phrase (1.8), the corresponding Dutch relative clause (1.9) shows how in Dutch we see a potentially unbounded number of constituents whose mutual dependencies, assumed to be the same in Dutch as they are in English, are *crossing*.

(1.8) ...that Marie heard$_i$ Jan$_i$ help$_j$ Fred$_j$ convince$_k$ Anne$_k$

(1.9) ...dat Marie Jan$_i$ Fred$_j$ Anne$_k$ hoorde$_i$ · helpen$_j$ · overtuigen$_k$ ·
that heard help convince

Nevertheless, the informational content of the English and the Dutch phrases are the same, so preferably we should model them in a way which emphasizes the underlying structural correspondence between the two.

1.3 Generative capacity of languages and k-pumpability
A comfortable way of assessing the formal complexity of linguistic constructions is based on extensions of the well known *pumping lemma* for context free languages. The following version of a pumping property can be proved for the class of languages recognized by *linear context-free rewriting systems* [VS87] [SMFK91] (discussed in section 2.2).

Definition 1 (k-pumpability) Let L be a language. Then L is (universally) *k-pumpable* if there are constants c_0, k such that for any $w \in L$ with $|w| > c_0$, there are strings $u_0, \ldots, u_k$ and $v_1, \ldots, v_k$ such that $w = u_0v_1u_1v_2u_2 \cdots u_{k-1}v_ku_k$, for each i: $1 \leq |v_i| < c_0$, and for any $p \geq 0$, $u_0v_1^pu_1v_2^pu_2 \cdots u_{k-1}v_k^pu_k \in L$.

Regular languages are 1-pumpable; Context-free languages are 2-pumpable [HU79]; tree adjoining or head languages are 4-pumpable [VS87]. For every language recognized by an LCFRS, there is a k such that it is k-pumpable [SMFK91]. So if a language can be shown not to be 2-pumpable, then it is not a CFL; if it is not 4-pumpable, it is not a TAL, and so forth.

A fragment containing cross-serial sentences such as (1.9) *sec* is 2-pumpable, since we can pump one co-indexed pair, say *Jan* and *hoorde*, to create another valid sentence. Indeed Dutch cross-serial clauses in isolation are *weakly* context free: there is a context free grammar that derives precisely the set of valid clauses. But such a grammar will necessarily generate an *embedded* structure:

(1.10) ...dat Marie Jan$_i$ Fred$_j$ Anne$_k$ hoorde$_k$ · helpen$_j$ · overtuigen$_i$ ·

I.e., a context-free grammar does not model a real *dependency* between these words, except that they are of the same category.

One way of showing the inadequacy of context-free grammars in describing crossed dependencies is to look at a formal language that resembles Dutch in its dependencies.[2] Another, example-based illustration is given in section 3.

[2] Take for example the fragment (1.11), which represents the "NP"s and "VP"s as identical copies of words in the context-free grammar $\mathtt{c}^n\mathtt{d}^n$, so as to enforce a structural closeness between the 'verbs' and their 'complements':

(1.11) $\{w_1\mathtt{a}w_2\mathtt{a}\cdots\mathtt{a}w_n\mathtt{a}w_1\mathtt{b}w_2\mathtt{b}\cdots\mathtt{b}w_n\mathtt{b} \mid n > 0,\ w_1, \ldots, w_n \in \mathtt{c}^n\mathtt{d}^n\}$

In this *copy language model* we see that in general we need $k = 4$: the pumped parts v_i according to definition 1 need to be shorter than c_0, so if the w_i are sufficiently long, we cannot simply duplicate corresponding pairs of w_i's to form a new sentence; so we need to apply pumping to two identical w_i's, which require 2-pumping. So in total we need to pump 4 substrings, and this corresponds to the fact that crossed dependencies can be *strongly* generated by a TAG or HG, which is 4-pumpable.

The most formally adequate method however is a weak generative capacity argument: it can be shown that when crossed dependencies interact with other phenomena, we get string sets that cannot be described by a CFG. If we want to take account of *conjunctions*, we can even show that the full class of languages satisfying a universal k-pumping lemma will not be sufficiently powerful. An example given in [MR87] showing that Dutch is beyond the descriptive capacity of TAG, can be extended to an argument showing that Dutch[3] is beyond the weak generative capacity of LCFRS.

(1.12) ...dat Jan Piet Marie liet opbellen, hoorde uitnodigen,
that made call heard invite
hielp ontmoeten en zag omhelzen
helped meet saw embrace

...*that Jan made Piet call Marie, heard [him] invite [her],*
helped [him] meet [her] and saw [him] embrace [her]

(1.13) ...dat Jan Piet Marie$^+$ liet laten$^+$ opbellen, hoorde horen$^+$ uitnodigen,
hielp helpen$^+$ ontmoeten en zag zien$^+$ omhelzen

Manaster-Ramer's argument is as follows:[4] a fragment containing sentences with 4 conjuncts like (1.12) can be obtained as the intersection of Dutch with a regular language (1.13). This fragment is not 4-pumpable,[5] since we need to pump 5 substrings (*Marie*, *laten*, *horen*, *helpen* and *zien*) to obtain another sentence in the fragment. So the fragment is not a TAL, and because the tree adjoining languages are closed under intersection with regular languages, Dutch cannot be a TAL.

Let's now extend the argument to cover *arbitrary* numbers of conjuncts, as in (1.14). This fragment is obtained[6] by intersecting Dutch with the regular language (1.15).

(1.14) { ...dat Jan Piet Marie Fred^n (hoorde leren^n uitnodigen,)$^+$
en zag leren^n omhelzen. | $n \geq 0$ }

(1.15) ...dat Jan Piet Marie Fred* (hoorde leren* uitnodigen,)$^+$
en zag leren* omhelzen.

In this example, we can't simply say that for a sentence with k conjuncts, we need to pump $k+1$ substrings, because we can always repeat a single conjunct. However, LCFRS satisfy the pumping property of definition 1, which also states that the size of the pumped substrings has a fixed upper bound c_0. Suppose the fragment is an LCFRL, then there must be a k such that it is k-pumpable. But in sentence (1.16), the conjuncts are longer than c_0 so we are not allowed to pump them as a whole; so we can only create new sentences within the fragment by pumping $k+1$ sequences: substrings of Fred^{c_0} and each of the VR sequences leren^{c_0}.

(1.16) ...dat Jan Piet Marie Fred^{c_0} (hoorde leren^{c_0} uitnodigen,)$^{k-1}$
en zag leren^{c_0} omhelzen.

[3] The argument will also work for German; what matters is not the crossed dependencies, but the fact that there is a fragment in which all sentences satisfy the condition that all object NPs precede all the verbs.

[4] The fragments in [MR87] contain more natural sentences than the ones given here, because the regular languages Manaster-Ramer intersects Dutch with allow a more free choice of verbs.

[5] Manaster-Ramer's fragment is *weakly* 5-pumpable, but the corresponding copy language model would introduce extra dependencies between the NP's and the VR's, and thus even require 10-pumping. So a binary conjunction would be enough to create a strongly trans-tree adjoining language.

[6] This process of "obtaining" simple fragments by intersecting Dutch with regular languages is described much more thoroughly in [MR87]. See e.g. the remarks on the possibility of dropping some of the NPs in [MR87] ("omitting subjects").

Now assume that Dutch is recognized by an LCFRS, then so must fragment (1.14), because LCFRL is closed under intersection with regular languages. But we have just shown that (1.14) is not k-pumpable for any k, so Dutch cannot be described by an LCFRS.

2. A series of generalizations of CFG

Tuple-based extensions of context-free grammar are best explained along the lines of the typical translation of a CFG to a logic program. E.g. the simple context free grammar (2.17) can be represented in Prolog as (2.18).

(2.17)
$$\begin{array}{lll} \mathrm{S} & \rightarrow & \mathrm{NP}\ \mathrm{VP} \\ \mathrm{NP} & \rightarrow & \texttt{Mary} \\ \mathrm{VP} & \rightarrow & \texttt{left} \end{array}$$

(2.18)
$$\begin{array}{l} \mathrm{s}(Z) \texttt{ :- } \mathrm{np}(X), \mathrm{vp}(Y), \mathrm{append}(X, Y, Z). \\ \mathrm{np}([\mathrm{Mary}]). \\ \mathrm{vp}([\mathrm{left}]). \end{array}$$

In this paper I will use a notation, illustrated in (2.19), for grammars that is half way between the conventions for writing down context-free grammars and Prolog—we don't use the Prolog conventions for variables, and polish away the use of **append** by allowing associative concatenation in predicate arguments.

(2.19)
$$\begin{array}{l} \mathrm{S}(xy) \texttt{ :- } \mathrm{NP}(x), \mathrm{VP}(y). \\ \mathrm{NP}(\texttt{Mary}). \\ \mathrm{VP}(\texttt{left}). \end{array}$$

(2.20)
$$\begin{array}{l} \mathrm{A}(\texttt{b}x\texttt{c}) \texttt{ :- } \mathrm{A}(x). \\ \mathrm{A}(\lambda). \end{array}$$

(2.21)
$$\begin{array}{ll} \mathrm{S}(yx) & \texttt{:- } \mathrm{A}(x, y). \\ \mathrm{A}(\texttt{b}x\texttt{c}, \texttt{a}y) & \texttt{:- } \mathrm{A}(x, y). \\ \mathrm{A}(\lambda, \lambda). & \end{array}$$

As said in the introduction, in natural languages, sometimes separate parts of the *yield* of a constituent wind up in different places in a sentence. Hence we use the insight given by Prolog that we need not stop here, but we could also for example write a grammar like (2.21)[7] which describes the language $\texttt{a}^n\texttt{b}^n\texttt{c}^n$, in a manner straightforwardly extending that of its context free relatives. While it derives a string $\texttt{b}^n\texttt{c}^n$ just like the context-free grammar (2.20), a separate component $\texttt{a}^n$ is generated in parallel, and prefixed to the $\texttt{b}^n\texttt{c}^n$ part in the S clause.

We will soon see that most of these grammars are actually linear context-free rewriting systems in a 'funny', streamlined notation; we will also see that there is a benefit in staying more close to the Prolog notation. Examples of grammars in the formalisms I define here are deferred to the next section.

2.1 Literal Movement Grammar

Literal movement grammar is the formal version of the class of grammars just described. This is a rather large class (any r.e. language can be described), but we can easily isolate significant subclasses, some but not all of which will correspond to other known grammar formalisms.

[7] λ is the empty string.

Definition 2 A *(predicate) literal movement grammar* (pLMG) is a tuple $G = (N, T, V, S, P)$ where N, T and V are finite, mutually disjoint sets of *nonterminal symbols*, *terminal symbols* and *variable symbols* respectively, $S \in N$, and P is a finite set of *clauses*

$$\phi \;\texttt{:-}\; \psi_1, \psi_2, \ldots, \psi_m.$$

where $m \geq 0$ and each of $\phi, \psi_1 \ldots \psi_m$ is a *predicate*

$$A(t_1, \ldots, t_p)$$

where $p \geq 1$, $A \in N$ and $t_i \in (T \cup V)^*$. As in Prolog, we leave out the :- symbol when $m = 0$.

A predicate LMG clause is *instantiated* by substituting a string $w \in T^*$ for each of the variables occurring in the clause. E.g. the rule $\mathrm{S}(xy) \texttt{ :- } \mathrm{NP}(x), \mathrm{VP}(y)$ could be instantiated to rules over only terminals such as

S(**Mary pinched the cat**) :- NP(**Mary**), VP(**pinched the cat**).
S(**Mary pinched the cat**) :- NP(**Mary pinched the**), VP(**cat**).

Definition 3 (rewrite semantics) Let $G = (N, T, V, S, P)$ be a predicate LMG. Then G is said to *recognize*[8] a string w if $\vdash^G S(w)$ where $\vdash^G$ is defined inductively as follows: if $w_i,\ v_{ki} \in T^*$ and

$$A(w_1, \ldots, w_p) \;\texttt{:-}\; B_1(v_{11}, \ldots, v_{1p_1}), \ldots, B_m(v_{m1}, \ldots, v_{mp_m}).$$

is an instantiation of a clause in P, and for each $1 \leq k \leq m$ we have $\vdash^G B_k(v_{k1}, \ldots, v_{kp_k})$, then $\vdash^G A(w_1, \ldots, w_p)$. Note that $m = 0$ is the base case (zero antecedents).

Although it is a rather simple example ($m = 1$), let's check how the LMG in (2.21) derives aabbcc:

$$\begin{array}{llll} & \vdash^G \mathrm{A}(\lambda,\ \lambda) & \text{by } \mathrm{A}(\lambda, \lambda). & \\ \Rightarrow & \vdash^G \mathrm{A}(\mathsf{bc},\ \mathsf{a}) & \text{by } \mathrm{A}(\mathsf{b}x\mathsf{c}, \mathsf{a}y) \texttt{ :- } \mathrm{A}(x, y). & (x = y = \lambda) \\ \Rightarrow & \vdash^G \mathrm{A}(\mathsf{bbcc},\ \mathsf{aa}) & \text{by } \mathrm{A}(\mathsf{b}x\mathsf{c}, \mathsf{a}y) \texttt{ :- } \mathrm{A}(x, y). & (x = \mathsf{bc},\ y = \mathsf{a}) \\ \Rightarrow & \vdash^G \mathrm{S}(\mathsf{aabbcc}) & \text{by } \mathrm{S}(yx) \texttt{ :- } \mathrm{A}(x, y). & (x = \mathsf{bbcc},\ y = \mathsf{aa}) \end{array}$$

2.2 Embedding other formalisms into LMG

Given that much of the design of LMG is inspired by the translation of a CFG to a logic program, it should be obvious how a CFG is translated to an equivalent LMG: if the RHS of a CFG production contains n nonterminals and m terminal symbols, we use n different variables $x_1, \ldots, x_n$ for the nonterminals; the terminal symbols are moved to the left hand side of the clause; e.g. the context free production (2.22) will be translated to the LMG clause (2.23).

(2.22) $\quad \mathrm{VP} \rightarrow \mathrm{VP}\ \mathsf{and}\ \mathrm{VP}$

(2.23) $\quad \mathrm{VP}(x_1\ \mathsf{and}\ x_2) \;\texttt{:-}\; \mathrm{VP}(x_1),\ \mathrm{VP}(x_2).$

It is more interesting to look at how stronger grammar formalisms are translated to predicate LMG.

Head Grammar The weakest class of formalisms exceeding CFG in recognizing power is that of HG, TAG, LIG and CCG (for the proofs of mutual equivalence see [VSW94]). We look at head grammar in its most simplified form, weakly equivalent to the one of [Pol84]: we view headed strings as pairs of terminal strings, and we look at bilinear productions only.

[8] This definition of LMG does not assign an *arity* to a nonterminal symbol; this is to simplify the definitions, especially in section 4.1. In principle, a nonterminal can appear with different arities in the same LMG; in particular if the start symbol doesn't appear with the arity 1, the language recognized by the grammar is empty.

Definition 4 (head grammar) A bilinear *head grammar* (HG) is a tuple (N, T, S, P) as in a context free grammar, but where the productions in P are terminal, that is of the form (2.24) or nonterminal (2.25),

(2.24) $\quad A \to \langle w_1, w_2 \rangle$

(2.25) $\quad A \to f(B_1, B_2)$

where $A, B_1, B_2 \in N$, $w_1, w_2 \in T^*$, and the *yield function*, f, is one of the function symbols *wrap*, *concat*$_1$ or *concat*$_2$.

A head grammar G recognizes pairs of terminal strings, as follows:

Base case If $A \to \langle w_1, w_2 \rangle$ is a grammar rule, then $A \overset{G}{\Longrightarrow} \langle w_1, w_2 \rangle$

Inductive case If $A \to f(B_1, B_2)$ is a rule in G, $B_1 \overset{G}{\Longrightarrow} \langle v_1, v_2 \rangle$ and $B_2 \overset{G}{\Longrightarrow} \langle w_1, w_2 \rangle$, then $A \overset{G}{\Longrightarrow} f(\langle v_1, v_2 \rangle, \langle w_1, w_2 \rangle)$ where

$$\begin{aligned}
\mathit{wrap}(\langle v_1, v_2 \rangle, \langle w_1, w_2 \rangle) &= \langle w_1 v_1,\ v_2 w_2 \rangle \\
\mathit{concat}_1(\langle v_1, v_2 \rangle, \langle w_1, w_2 \rangle) &= \langle v_1,\ v_2 w_1 w_2 \rangle \\
\mathit{concat}_2(\langle v_1, v_2 \rangle, \langle w_1, w_2 \rangle) &= \langle v_1 v_2 w_1,\ w_2 \rangle
\end{aligned}$$

The underlying intuition is that a tuple $\langle w_1, w_2 \rangle$ represents a constituent $w_1 w_2$ whose *head* is the first terminal of w_2.
A head grammar is an LMG that operates on pairs. A wrapping production (2.26) is represented in LMG as (2.27) and a concatenating rule like (2.28) as (2.29).

(2.26) $\quad A \to \mathrm{wrap}(B, C)$

(2.27) $\quad A(y_1 x_1,\ x_2 y_2)$:- $B(x_1, x_2),\ C(y_1, y_2)$.

(2.28) $\quad A \to \mathrm{concat}_1(B, C)$

(2.29) $\quad A(x_1,\ x_2 y_1 y_2)$:- $B(x_1, x_2),\ C(y_1, y_2)$.

A terminal production $A \to \langle w_1, w_2 \rangle$ is translated as $A(w_1, w_2)$. We then have $A \overset{G}{\Longrightarrow} \langle w_1, w_2 \rangle$ in the HG if and only if $\vdash^G A(w_1, w_2)$ in the LMG translation.

Linear context-free rewriting systems Linear context-free rewriting systems (LCFRS, [Wei88]) generalize head grammars by allowing arbitrary tuples instead of just pairs, and instead of the wrapping and concatenation operators, any operator which is *linear* and *nonerasing*. A production in an LCFRS looks just like one in head grammar:

$$A \to f(B_1, \ldots, B_m)$$

where f is a function over tuples of terminal words defined (symbolically) as

$$f(\langle x_{11}, \ldots, x_{1p_1} \rangle, \ldots, \langle x_{m1}, \ldots, x_{mp_m} \rangle) = \langle t_1, \ldots, t_p \rangle$$

where t_i are terms over the variables x_{ij} and terminal symbols. In LCFRS, f is required to be *linear* and *nonerasing*, that is every one of the x_{ij} should appear *exactly once* in the $t_1, \ldots, t_p$.
Clearly every head grammar is an LCFRS; the notion of LCFRS derivation is analogous to that for HG. Just as with the head grammars, we can write the rules in predicate LMG notation instead:

$$A(t_1, \ldots, t_p) \ \text{:-}\ B_1(x_{11}, \ldots, x_{1p_1}),\ \ldots,\ B_m(x_{m1}, \ldots, x_{mp_m}).$$

The conditions on the use of variables remain the same; the x_{ij} on the right hand side are distinct variables, which is to say the antecedents B_i of the clause are *independent*; it follows that we can freely substitute LCFRS subderivations, and hence the formalism is closed under homomorphism. The conditions of linearity and nonerasingness are responsible for the familiar results of constant growth and k-pumpability. A first example of an LCFRS is (2.21).

Parallel multiple context-free grammars A *parallel multiple context-free grammar* (PMCFG, [KNSK92]) is like an LCFRS, except that rules need not be linear and nonerasing. The traditional notation for a PMCFG is just like that of LCFRS, but we can also choose to define PMCFG to be an LMG that satisfies the syntactic property that in each clause

$$A(t_1, \ldots, t_p) \;\text{:-}\; B_1(x_{11}, \ldots, x_{1p_1}),\; \ldots,\; B_m(x_{m1}, \ldots, x_{mp_m}).$$

all the RHS variables x_{ij} are distinct (which means closure under homomorphism).

2.3 Classification

It is easy to see that the formalisms described in the previous section are of strictly growing generative capacities: HG can generate the 3-counting language $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$ which is not context-free; LCFRS can generate arbitrary counting languages $a_1^n a_2^n \cdots a_k^n$ (for any k), not generated by a head grammar when $k > 4$. It is also a known result that if L is recognized by an LCFRS, then L is k-pumpable for some finite k [SMFK91]. This is not true for PMCFG; grammar (2.30) generates the language a^{2^n}, which does not allow k-pumping for any finite k.

(2.30) $\mathrm{S}(xx)$:- $\mathrm{S}(x)$.
$\mathrm{S}(a)$.

For a full classification of the different formalisms in their predicate LMG versions, we introduce some terminology; the 'bottom-up', 'top-down' terminology in definition 5 reflects the behaviour in the perspective of derivation trees, where the root (S-) node is the top of the derivation.

Definition 5 (properties of LMG) Let $G = (N, T, V, S, P)$ be a LMG, and let $R \in P$ be one of its productions:

$$A(t_1, \ldots, t_p) \;\text{:-}\; B_1(s_{11}, \ldots, s_{1p_1}),\; \ldots,\; B_m(s_{m1}, \ldots, s_{mp_m}).$$

then

- R is *bottom-up linear* if no variable x appears more than once in $t_1, \ldots, t_p$.
- R is *top-down linear* if no variable x appears more than once in $s_{11}, \ldots, s_{mp_m}$.
- R is *bottom-up nonerasing* if each variable x occurring in an s_{jk} also occurs in at least one of the t_i.
- R is *top-down nonerasing* if each variable x occurring in one of the t_i also appears in one of the s_{jk}.
- R is *non-combinatorial* (cf. [Gro95b]) if each of the s_{jk} consists of a single variable.
- R is *simple* if it is bottom-up nonerasing, bottom-up linear and non-combinatorial.

For all these properties, G has the property if and only if all $R \in P$ have the property.

In the terminology of definition 5, an LCFRS is a noncombinatorial, top-down and bottom-up linear, top-down and bottom-up nonerasing LMG. A PMCFG is only top-down nonerasing and top-down linear. Note that all literal movement grammars considered in this paper are noncombinatorial (RHS predicates have only single variables in argument positions). A summary is given in figure 1.

Formalism	*Increasing conditions on LMG form*
Generic LMG	—
Simple LMG	Bottom-up nonerasing, non-combinatorial
(Nonerasing) PMCFG	Top-down linear, top-down nonerasing
LCFRS	Bottom-up linear
HG	Pairs only, restricted operations
CFG	Singletons

Figure 1: Hierarchical classification 1

Generic and simple LMG It is easy to see that the languages LML recognized by arbitrary LMG is the set of recursively enumerable languages. First observe that LMG are closed under intersection: take two simple LMGs G_1 and G_2 whose start symbols are S_1 and S_2 respectively. Then combine the clauses of G_1 and G_2 (renaming nonterminals where necessary) and add the clause (2.31) which says "$\mathrm{S}(x)$ can be derived if we can derive both $\mathrm{S}_1(x)$ and $\mathrm{S}_2(x)$." Clearly the resulting grammar with start symbol S generates the intersection of G_1 and G_2.

(2.31) $\mathrm{S}(x)$:- $\mathrm{S}_1(x)$, $\mathrm{S}_2(x)$.

Then use the familiar result (see e.g. [Gin75] p. 125) that any r.e. language L can be described as $h(L_1 \cap L_2)$ where h is a homomorphism and the languages $L_1 = L(G_1)$ and $L_2 = L(G_2)$ are context-free. The clauses in (2.32), together with the clauses for G_1 and G_2 then generate L.

(2.32) $\mathrm{S}(y)$:- $\mathrm{S}_1(x)$, $\mathrm{S}_2(x), \mathrm{H}(x, y)$.
$\mathrm{H}(\mathrm{a}x, wy)$:- $\mathrm{H}(x, y)$. where $w = h(\mathrm{a})$ [for each $\mathrm{a} \in T$]
$\mathrm{H}(\lambda, \lambda)$.

However, there is at least one class smaller than LML but larger than PMCFG that is of special interest, and this is what I will be focusing on from this point; that is *simple* LMG (definition 5). The properties of a simple LMG make that it literally subsumes LCFRS, but not PMCFG. However, we can show that any PMCFG can be translated to a weakly equivalent LMG (which may have an $O(|T|)$ times large number of grammar rules).

Simple LMG does not allow a variable to appear more than once on the LHS of a clause: e.g. the PMCFG clause (2.33) is not a valid simple LMG clause. However (and contrary to PMCFG) simple LMG does allow variables to appear on the *right hand side* more than once. So we can replace the single PMCFG clause by the simple LMG clause (2.34) that refers to an *equality predicate* defined by two clauses (2.35) one of which is a *schema* over all terminals. This is straightforwardly generalized into a proof that for any PMCFG there is an equivalent simple LMG.

(2.33) $\mathrm{A}(x, yy)$:- $\mathrm{B}(x)$, $\mathrm{C}(y)$.

(2.34) $\mathrm{A}(x, yz)$:- $\mathrm{B}(x)$, $\mathrm{C}(y)$, $\mathrm{Eq}(y, z)$.

(2.35) $\mathrm{Eq}(\mathrm{a}x, \mathrm{a}y)$:- $\mathrm{Eq}(x, y)$. for every $\mathrm{a} \in T$
$\mathrm{Eq}(\lambda, \lambda)$.

To show that simple LMG is of a strictly stronger generative capacity than PMCFG, observe that clause (2.31) is simple, so simple LMG is closed under intersection, and that any PMCFG can be translated to an equivalent simple LMG; now suppose PMCFG and simple LMG would be equally powerful, then they would generate a class of languages including CFL, that is closed under arbitrary homomorphism and intersection, that is all r.e. languages. But we know [KNSK92] that the languages recognized by PMCFG are in PTIME, so a fortiori they are decidable. This is a contradiction, so we conclude that PMCFL is not closed under intersection, and hence simple LML *strictly* includes PMCFL. It is shown in section 4.3 that simple LML is in PTIME.

3. A descriptive case study

We have already seen how Manaster-Ramer's 1987 example of a very elementary trans-tree adjoining fragment of Dutch can be extended to challenge the generative capacity of LCFRS. However, this is a formal argument, and it talks about *weak* generative capacity. In practice, the grammar formalisms fall short of descriptive adequacy a bit sooner than can be proved by a weak capacity argument, so it's worthwhile to investigate the practical capacity of the various formalisms by looking at a few small concrete grammars.

3.1 Head grammar: crossed dependencies

[Pol84] presents an extensive treatment of discontinuous constituency in English, including examples such as *Kim is easy to please* (1.1). Here I will repeat, in a slightly modified version, a description of Dutch crossed dependencies Pollard gives in an appendix.

Recall example (1.9). Just as suggested in the introduction, the HG analysis will assume that the Dutch cross-serial relative clause (3.37) has the same functional head-complement structure as its English counterpart (3.36), the only difference being that English has an exceptionally simple, 'linear', surface form.

(3.36) John [$_{\mathrm{VP}}$[$_{\overline{\mathrm{V}}}$ saw Mary [$_{\overline{\mathrm{V}}}$ drink coffee]]]

(3.37) ... dat Jan [$_{\mathrm{VP}}$[$_{\mathrm{NC}}$ Marie koffie] [$_{\mathrm{VC}}$ zag drinken]]
that saw drink
...*that John saw Mary drink coffee*

In Dutch, we need to think of the VP as divided into two *clusters*: the nominal cluster NC and the verbal cluster VC. The yield of each of the embedded verbal projections is then somehow divided into one part that selects to appear in the NC, and another part that selects to appear in the VC.

$$\begin{array}{lll}
\overline{\mathrm{V}} & \rightarrow & \mathit{concat}_2(\mathrm{NP},\ \mathrm{VT}) \\
\mathrm{VT} & \rightarrow & \mathit{wrap}(\mathrm{VR},\ \overline{\mathrm{V}}) \\
 & & \\
\mathrm{NP} & \rightarrow & \langle \lambda, \mathtt{Marie} \rangle \mid \langle \lambda, \mathtt{koffie} \rangle \\
\mathrm{VT} & \rightarrow & \langle \lambda, \mathtt{drinken} \rangle \\
\mathrm{VR} & \rightarrow & \langle \lambda, \mathtt{zag} \rangle
\end{array}$$

Figure 2: HG for transitive and raising verbs in Dutch.

A head grammar that closely resembles the example in [Pol84] is shown in figure 2. The head of a $\overline{\mathrm{V}}$ is its leading verb, so the yield of a $\overline{\mathrm{V}}$ is a tuple whose first component is a sequence of NPs (the nominal cluster) and whose second component is a series of verbs (the verb cluster). The first of the noun phrases is the direct object of the head verb. The derivation tree for the verb phrase *Marie koffie zag drinken* (saw Mary drink coffee) is shown in figure 3.

The HG analysis emphasizes both the underlying functional or deep structure of the verb phrase, and the way the surface form of a Dutch VP can be constructed from that deep structure, without the

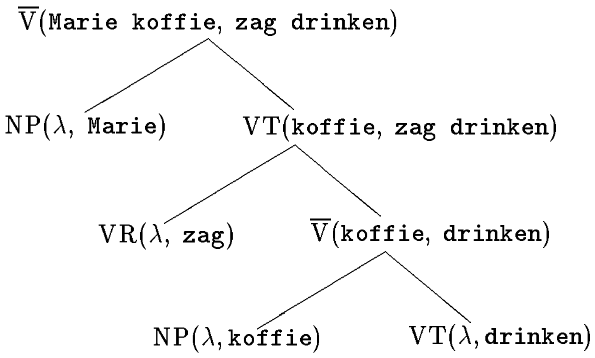

Figure 3: HG derivation.

need for a distinct surface *structure*. Why is it, then, that no efforts are made in the literature to show that the HG account can be extended to cover other similar phenomena, such as verb second forms and leftward nominal extraposition? The crucial point is that in these cases the position of the head is no longer sufficient as a handle on what parts the yield of a constituent should be separated into, and we have to revert to more arbitrary, less linguistically well-motivated cluster divisions. This does not seem to turn out problematic; nor can I find an argument in Pollard's introduction [Pol84] to head grammar making a claim beyond the observation that "many discontinuity phenomena can be described by [such] head wrapping operations".

3.2 Linear context-free rewriting systems: combining phenomena

LCFRS straightforwardly extend HG by allowing operations over n-tuples; in other words, by allowing constituents to be divided into more than 2 clusters. Apart from the loss of the linguistically flavoured motivation in terms of the role of heads, one of the reasons such grammars have not been looked at in practical descriptions is that the traditional presentation of LCFRS makes the grammars a bit hard to read. The LMG notation can help somewhat in making LCFRS a better tool for concrete description. Suppose we want to extend the account of the VP to produce full Dutch sentential forms (verb second in (3.38b) and (3.38c) and topicalization of the first object in (3.38d)):

(3.38) a. ...dat Jan **Marie koffie zag drinken**
...that John saw Mary drink coffee
b. Jan **zag Marie koffie drinken**
John saw Mary drink coffee
c. **Zag** Jan **Marie koffie drinken**?
Did John see Mary drink coffee
d. **Wie zag** jij **koffie drinken**?
Who did you see drink coffee?

If we want these four examples to get the same VP analysis, we see that we need to split up the VP into at least *three* components, i.e. the finite verb *zag*, its direct object *wie/Marie* and the remaining frame *koffie drinken.*

The example in figure 4 is an LCFRS in its predicate LMG form which describes the dutch cross-

$$\begin{array}{llll}
\text{s-rel:} & \mathrm{S}(\ldots\texttt{dat}\ s\ do\ hv) & :- & \mathrm{NP}(s),\ \overline{\mathrm{V}}(d,\ o,\ h,\ v).\\
\text{s-decl:} & \mathrm{S}(s\ h\ do\ v) & :- & \mathrm{NP}(s),\ \overline{\mathrm{V}}(d,\ o,\ h,\ v).\\
\text{s-inter:} & \mathrm{S}(h\ s\ do\ v) & :- & \mathrm{NP}(s),\ \overline{\mathrm{V}}(d,\ o,\ h,\ v).\\
\text{s-topic:} & \mathrm{S}(d\ h\ s\ o\ v) & :- & \mathrm{NP}(s),\ \overline{\mathrm{V}}(d,\ o,\ h,\ v).\\
 & & & \\
 & \overline{\mathrm{V}}(d,\ \lambda,\ h,\ \lambda) & :- & \mathrm{VT}(h),\ \mathrm{NP}(d).\\
 & \overline{\mathrm{V}}(n,\ do,\ r,\ hv) & :- & \mathrm{VR}(r),\ \mathrm{NP}(n),\ \overline{\mathrm{V}}(d,\ o,\ h,\ v).
\end{array}$$

$$\begin{array}{lll}
\mathrm{NP}(\texttt{Jan}). & \mathrm{NP}(\texttt{Marie}). & \mathrm{NP}(\texttt{koffie}).\\
\mathrm{NP}(\texttt{jij}). & \mathrm{NP}(\texttt{wie}). & \\
\mathrm{VT}(\texttt{drinken}). & \mathrm{VR}(\texttt{zag}). & \\
\ldots & &
\end{array}$$

Figure 4: An LCFRS in predicate notation for Dutch sentential forms.

serial VP and four sentential forms including verb second and topicalization.[9] It strictly refines the HG from the previous section, in that it still divides the verb phrase into a *nominal cluster* and a *verb cluster*; but it splits up both clusters into two components. A verb phrase is now a four-tuple $\langle d, o, h, v\rangle$ consisting of a direct object d, the rest of the object cluster o, the head verb h, and the rest of the verb cluster v. The subject has suggestively been assigned the variable s, so a correspondence to the SOV/SVO terminology helps reading the grammar. An example derivation is given in figure 5.[10]

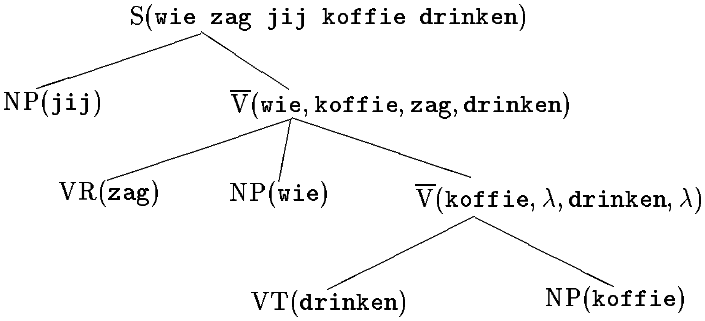
S(wie zag jij koffie drinken)
NP(jij)
V̄(wie, koffie, zag, drinken)
VR(zag)
NP(wie)
V̄(koffie, λ, drinken, λ)
VT(drinken)
NP(koffie)

Figure 5: LCFRS derivation of *Wie zag jij koffie drinken?*

[9] The original LCFRS would have rules like

$$\overline{\mathrm{V}} \ \rightarrow\ f_6(\mathrm{VR}, \mathrm{NP}, \overline{\mathrm{V}})\ \text{where} f_6(r, n, \langle d,\ o,\ h,\ v\rangle)\ =\ \langle n,\ do,\ r,\ hv\rangle)$$

which is certainly a little more cumbersome to read than the predicate LMG form.

[10] Note that when looking at the derivations, abstracting away from surface order, the SOV structure from the HG analysis has been changed to SVO in the LCFRS example—preferable for reasons of uniformity as this is the underlying structure of English; the order of the elements on the RHS of an LCFRS production is irrelevant, so the distinction SVO/SOV in an LCFRS setting is no more than an issue of cosmetics—though of course it still makes sense to talk about a VP-in/external subject.

3.3 PMCFG: reduplication

PMCFG extends LCFRS in providing a mechanism for *reduplication.* As such it can give accounts for a number of phenomena that involve counting. Examples of such phenomena are Chinese number names [Rad91], *respectively*-constructs, and Old Georgian genitive suffix stacking [MK96]. The fragment (1.14) repeated here as (3.39) is described by the PMCFG in figure 6.

(3.39) $\{$ ...dat Jan Piet Marie Fred^n (hoorde leren^n uitnodigen,)$^+$
en zag leren^n omhelzen. $\mid n \geq 0$ $\}$

The fragment can be described easily by a PMCFG because we used only one infinite raising verb (*leren*), so that we can generate the sequence `leren`^{n} once and then reduplicate it; this technique will fail as soon as we allow two different infinite raising verbs. Moreover, the analysis can hardly be thought of as assigning the right underlying structures.

S(`...dat Jan Piet Marie` f c `en zag` l `omhelzen.`) :- Cj(c, f, l).

Cj(`hoorde` l `uitnodigen` "`,`" c, f, l) :- Cj(c, f, l).
Cj(`hoorde` l `uitnodigen`, f, l) :- Fl(f, l).

Fl(`Fred` f, `leren` l) :- Fl(f, l).
Fl(λ, λ).

Figure 6: PMCFG for unbounded conjunctions.

3.4 Simple LMG: conjunction through sharing

The added strength of a simple LMG over a PMCFG is that it does assign a well-motivated analysis to co-ordinating crossed dependency phrases. The grammar in figure 7 makes use of shared variables on the right hand side of a clause to extrapose the same sequence of NP objects from the VP more than once; it describes sentences like those in the introduction with an unbounded number of conjuncts. As such it covers a slightly larger and more realistic fragment[11] than (3.39).

The advantage of simple LMG over LCFRS remains limited to to ability to describe the combination of conjunction and cross-serial clauses—even the very complex grammars in [Gro95a] use multiple occurrences of variables on the RHS of productions only in rules describing conjunction. Figure 8 is a summary of the conclusions of this section.

4. Fixed point interpretations and complexity

The examples have already shown how LMG subsumes the chain CFG $\subseteq$ HG $\subseteq$ LCFRS $\subseteq$ PMCFG of formalisms of increasing generative capacities. The fixed recognition problems for these formalisms are known to be decidable in polynomial time. We will now show that *simple* LMG is not only favourable in its capacity to describe conjunctions, but is also interesting from a formal point of view: it describes exactly the class PTIME of languages recognizable in deterministic polynomial time.

Calculi that describe PTIME have been known for quite some time. The calculus ILFP (integer least fixed point) is introduced in [Rou88]; it applies knowledge about the relationship between bounded arithmetic and complexity to language recognition. The underlying idea is that by talking about *positions in* the input string, as opposed to about the strings themselves, we can store intermediate steps in the search for a derivation in logspace, which by the Chandra-Kozen-Stockmeyer [CKS81] result on the correspondence between deterministic and alternating Turing machine computations then gives a deterministic PTIME complexity for recognition.

[11] For the sake of simplicity, the grammar shown here makes no distinction between en and a comma, and does not distinguish finite and infinite verbs.

$$\begin{array}{lll}
\mathrm{S}(\ldots\texttt{dat}\ s\ o\ v) & \texttt{:-} & \mathrm{NP}(s), \mathrm{VP}(o, v). \\
 & & \\
\mathrm{VP}(o, v_1\ \texttt{en}\ v_2) & \texttt{:-} & \overline{\mathrm{V}}(o, v_1), \mathrm{VP}(o, v_2). \\
\mathrm{VP}(o, v) & \texttt{:-} & \overline{\mathrm{V}}(o, v). \\
 & & \\
\overline{\mathrm{V}}(o, v) & \texttt{:-} & \mathrm{VT}(v), \mathrm{NP}(o). \\
\overline{\mathrm{V}}(no, rv) & \texttt{:-} & \mathrm{VR}(r), \mathrm{NP}(n), \overline{\mathrm{V}}(o, v). \\
 & & \\
\mathrm{NP}(\texttt{Jan}). & & \\
\mathrm{NP}(\texttt{Marie}). & & \\
\mathrm{VT}(\texttt{opbellen}). & & \\
\mathrm{VR}(\texttt{zag}). & & \\
\ldots & &
\end{array}$$

Figure 7: Simple LMG for unbounded conjunctions.

Formalism	*Concrete descriptive capacity*
Generic LMG	—
Simple LMG	Conjunctions and multiple extraposition
PMCFG	Reduplication, number sequences
6-LCFRS	Substantial fragments of Dutch verb structure [Gro95a]
4-LCFRS	Conjunction-free crossed dependencies, topicalization and verb 2nd simultaneously
HG	Isolated, simple Dutch relative clauses
CFG	English verb phrases without inversion

Figure 8: Hierarchical classification 2

Least fixed point interpretations of rule based grammar are attractive in many ways; in the case of LMG, the LFP interpretation will give a link to the ILFP calculus, as well as some insight into why the property of simplicity is a sensible restriction to the general form of LMG rules in order to get tractable recognition.

4.1 Fixed point interpretations of LMG

Let $G = (N, T, V, S, P)$ be an LMG. Let NA be the set of *assignments* to the nonterminals: functions ρ mapping a nonterminal to a set of arbitrary tuples of strings over T. The set of productions P can then be viewed as an operator $[G]$ taking an assignment as an argument and producing a new assignment, defined as follows: if ρ is an assignment, and

$$A(w_1, \ldots, w_p) \ \texttt{:-}\ B_1(v_{11}, \ldots, v_{1p_1}), \ldots, B_m(v_{m1}, \ldots, v_{mp_m}).$$

is an instantiation of a clause in P, and for each $1 \leq k \leq m$, $(v_{k1}, \ldots, v_{kp_k}) \in \rho(B_k)$, then $(w_1, \ldots, w_p) \in ([G]\rho)(A)$.

Define the complete partial order $(\mathrm{NA}, \sqsubseteq)$ by (4.40), its bottom element, the empty assignment ρ_0 by (4.41) and the join $\sqcup$ by (4.42). It is easily seen that $[G]$ is a continuous and monotonic operator on $(\mathrm{NA}, \sqsubseteq)$.

(4.40) $\rho_1 \sqsubseteq \rho_2 \ \Leftrightarrow\ \forall A \in N.\ \rho_1(A) \subseteq \rho_2(A):$

(4.41) $\rho_0(A) = \emptyset$ for all $A \in N$

(4.42) $t \in (\rho_1 \sqcup \rho_2)(A) \;\Leftrightarrow\; t \in \rho_1(A) \cup \rho_2(A)$

The interpretation of a grammar will now be the least fixed point of $[G]$:

$$\mathcal{I}_G \;=\; \bigsqcup_{k=0}^{\infty} [G]^k \rho_0$$

i.e. a function which takes a nonterminal and yields a set of tuples of strings; If S is the start symbol of G, and its arity throughout the grammar is 1, then $\mathcal{I}_G(S)$ will be the language recognized by the LMG in the traditional sense.

4.2 Motivation for the simplicity constraint

Our aim is to find out how we can restrict the LMG grammars in such a way that recognition can be performed as an alternating search in logspace. For a given string of length n, in log space we can encode a bounded set of numbers ranging from 0 to n (in binary encoding). This means that we have to encode the arguments of an LMG predicate in a derivation each with a bounded set of numbers. Since in the original interpretation the arguments are strings, the most obvious choice is to encode the arguments as pairs of integers ranging 0 to n encoding a *substring* of the input.

To put this formally: if $\rho \in \mathrm{PA}$ then let $\rho \sqcap w$ be defined by: $(w_1, \ldots, w_n) \in (\rho \sqcap w)(A)$ iff $(w_1, \ldots, w_n) \in \rho(A)$, and $w_1, \ldots, w_n$ are substrings of w; furthermore put $([G] \sqcap w)(\rho) = ([G]\rho) \sqcap w$. The question then is: how can I make sure that

$$\left(\bigsqcup_{k=0}^{\infty} [G]^k \rho_0 \right) \sqcap w \;=\; \bigsqcup_{k=0}^{\infty} ([G] \sqcap w)^k \rho_0?$$

Redefine the fixed point semantics as follows; let $w = a_0 a_1 \cdots a_{n-1}$ be a terminal string of length n; then NA_w is the set of *integer nonterminal assignments* ρ *for the input* w, mapping a nonterminal to a set of tuples of pairs of integers between 0 and n. Then we define $[G]_w$ as follows: if ρ is an integer assignment and

$$\begin{array}{l} A(a_{l_1} \cdots a_{r_1},\; \ldots,\; a_{l_p} \cdots a_{r_p}) \\ \quad \texttt{:-}\;\; B_1(a_{l_{11}} \cdots a_{r_{11}},\; \ldots,\; a_{l_{1p_1}} \cdots a_{r_{1p_1}}),\; \ldots,\; B_m(a_{l_{m1}} \cdots a_{r_{m1}},\; \ldots,\; a_{l_{mp_m}} \cdots a_{r_{mp_m}}). \end{array}$$

is an instantiation of a clause in P, and for each $1 \le k \le m$, $(\langle l_{k1}, r_{k1} \rangle, \ldots, \langle l_{kp_k}, r_{kp_k} \rangle) \in \rho(B_k)$, then $(\langle l_1, r_1 \rangle \ldots, \langle l_p, r_p \rangle) \in ([G]_w \rho)(A)$.

It is important to see that what is done here, is in the general case *not* the same as taking the string-based LFP interpretation, and intersecting the sets of tuples with the domain of substrings of a given w. If we have an instantiated clause

$$A(w_1, \ldots, w_p) \;\;\texttt{:-}\;\; B_1(v_{11}, \ldots, v_{1p_1}),\; \ldots,\; B_m(v_{m1}, \ldots, v_{mp_m}).$$

such that $w_1, \ldots, w_p$ are substrings of w, but the v_{ij} are not, then this instantiation will be ignored in the integer LFP semantics: even though all w_i are substrings and $A(w_1, \ldots, w_p)$ is derived by the grammar, it cannot be 'reached' with a derivation in which only substrings of the input appear.

Hence we want to rule out this type of clause. I.e., we want to make sure that $w_1, \ldots, w_p$ are substrings of the input, so are the v_{ij}. Thus *simple LMG* is defined by disallowing terms other than single variables on the *right hand side* of the clauses. This way we can uniquely replace each rule by a clause that is talking about integer positions instead of strings. Note that we must also disallow variables on the RHS that do not appear on the LHS because then they could be instantiated with any string.[12]

[12] The decision not to allow variables to appear on the LHS more than once is more arbitrary.

Definition 6 (simple LMG, repeated) An LMG is called *simple* if its clauses $R \in P$ are all of the form

$$A(t_1, \ldots, t_p) \;\; \text{:-} \;\; B_1(x_{11}, \ldots, x_{1p_1}), \; \ldots, \; B_m(x_{m1}, \ldots, x_{mp_m}).$$

where the x_{ij} are disjoint, and each of the x_{ij} appears precisely once in $t_1, \ldots, t_p$.

4.3 Simple LML is in PTIME

There are two ways to show that the languages generated by simple LMG can be recognized in polynomial time. The first, most formal argument shows that every LMG can be translated into an equivalent formula in the integer string position calculus ILFP [Rou88]; we will not go into the definition of ILFP here, but an example translation of a cluster (4.43) of clauses for a nonterminal VP grammar fragment would be (4.44). The correspondence between the languages defined by ILFP and those recognized by logspace-bounded alternating Turing machines (ATM) proved in [Rou88] then completes the argument.

(4.43)
$$\begin{array}{ll} \mathrm{VP}(v_1 \text{ "}\mathbf{and}\text{" } v_2,\ n) & \text{:-}\ \mathrm{VP}(v_1, n),\ \mathrm{VP}(v_2, n). \\ \mathrm{VP}(v, n) & \text{:-}\ \mathrm{VT}(v),\ \mathrm{NP}(n). \end{array}$$

(4.44)
$$\begin{array}{l} \mathrm{VP}(i, j,\ k, l) \\ \Leftrightarrow\ \ \exists i', j'.\ (i \leq i' < j' \leq j \wedge \mathbf{and}(i', j') \wedge \mathrm{VP}(i, i',\ k, l) \wedge \mathrm{VP}(j', j,\ k, l)) \\ \qquad \vee\ (\mathrm{VT}(i, j) \wedge \mathrm{NP}(k, l)) \end{array}$$

In the context of this paper however it is useful to sketch a more practically minded recognition algorithm, whose presentation and complexity heuristics are much in the spirit of the those for the "functional parsing" techniques in [Lee93]. This will give a better indication of what a possible real world implementation would look like; an alternating Turing machine does not immediately correspond to a technique used in practice—let's try and get a bit more than just the 'foundational security' of a Turing machine construction.

Take an integer representation of LMG grammars, in which every argument is represented as a pair $\langle l, r\rangle$ of integer indices, as a point of departure; the ILFP translation given above will be excellent for this purpose.

Given an input string w of length n, construct memo tables containing a value **True**, **False** or **Unknown** for each possible predicate $A(l_1, r_1,\ \ldots,\ l_p, r_p)$, where l_i, r_i are integer values ranging from 0 to n. Reset all the table entries to **Unknown**. Now start with the predicate $S(0, n)$, and recursively check, using the memo table where possible, all possible instantiations of the bound variables (i' and j' in the example) in all applicable rules.

The procedure for VP rule we just translated is as follows:[13]

```
VP(i,j,k,l):
   if memo table entry for VP(i,j,k,l) != Unknown
   then
      return memoed value
   else
      memo VP(i,j,k,l) as False        // avoid redundant recursion

      loop i' = 0...n
      loop j' = 0...n
         if i <= i'       and
```

[13] Although I have successfully extended this algorithm to full LMG parsers, it disregards cyclic derivations, because it memoes a predicate as **False** before computing its value.

```
      j' <= j        and
      j' = i' + 1    and
      a[i'] = "and"  and
      VP(i,i',k,l)   and
      VP(j',j,k,l)
   then
      memo VP(i,j,k,l) as True
      return True

if VT(i,j)  and
   NP(k,l)
then
   memo VP(i,j,k,l) as True
   return True

return False
```

Given a simple literal movement grammar and its ILFP translation, let $|G|$ be the number of clauses in the grammar (which is w.l.o.g. also a bound on the number $|N|$ of nonterminal symbols), let m be the largest number of predicates on the RHS of a clause; let p be the largest number of integer predicate arguments and let q be the largest number of bound variables in each disjunct of the ILFP formula. Then the recognizer needs to do $O(m \cdot n^q)$ function calls (in the inner loop) for each of the predicates. Since there are $O(|G| \cdot n^p)$ predicates, recognition can be performed in deterministic $O(|G|mn^{p+q})$ time and $O(|G|n^p)$ memoing storage. Constructing a minimally informative parse forest would require $O(|G|n^{p+q})$ space.[14]
The bound given here seems tight. The rules of a binary, modified head grammar, such a the wrapping rule:

$$A(x_1y_1, y_2x_2) \;\; \texttt{:-} \;\; B(x_1, x_2), \;\; C(y_1, y_2).$$

are translated into integer based rules with 6 variables ($p = 4, \; q = 2$):

$$A(i,j, \; l,k) \;\Leftrightarrow\; \exists i', l'. \, (i \leq i' \leq j \wedge \; l \leq l' \leq k \wedge \; B(i,i', \; l',k) \wedge \; C(i',j, \; l,l'))$$

The general recognizer for HG we obtain by applying the sketched algorithm has the well known upper time bound of $O(n^6)$ for bilinearized head grammars. A similar argument gives a bound of $O(n^3)$ for bilinear context free grammars.

4.4 Simple LML subsumes PTIME

We proceed exactly as in [Rou88]. It is a known result that $\mathit{PTIME} = \mathit{ASPACE}(\log n)$. Let M be an alternating Turing machine [CKS81] with a read-only input tape and one binary working tape (the argument can then be extended to cover an arbitrary number of binary working tapes). Let M be space bounded by $\log n$, where n is the length of its input w.
Instantaneous descriptions (ID) of the ATM can be described by a state symbol q and a tuple (h, l, r, ll, rr) of integers ranging from 0 to n; h is the position of the input head, l and r describe the contents of the binary work tape left and right of its head, and ll and rr represent the amount of work tape space left and right of its head. As Rounds argues, an ID predicate $q(h, l, r, ll, rr)$ is defined in

[14] It should be admitted here that there is a certain amount of handwaving in this argument—the algorithm is recursive, with a maximum recursion depth of $O(n^p)$—extra storage and time required to do this recursion is not incorporated into the sketch. Nevertheless I have tried to be very precise in the particular sizes of the grammar entities in the total time complexity. The only quantities that are contribute to an exponential complexity are p and q; this is quite a bit more indicative than claims proved in many texts on universal recognition. Most importantly, [KNSK92] prove that "universal PMCFG recognition is EXP-POLY time hard" only in terms of the *sum* of the "size" of the grammar and the input, and even then these sizes do not seem to have been taken to be completely independent.

terms of other ID predicates through a disjunction (existential states) or conjunction (universal states) of other predicates, where the arguments of the predicates are built from h, l, r, ll and rr through the arithmetical constants and operations 0, 1, $n-1$, $+1$, -1, $*2$ and $/2$. The applicability of the moves is checked by equality and nonequality over values derived from h, l, r, ll, rr by the operators.

We now simulate the ATM in a simple LMG by introducing a 6-ary nonterminal for each state q; its first argument is a copy of the input w; the last five are arbitrary substrings of w, whose length corresponds to the values of h, l, r, ll, rr. The start rule of the grammar is

$$S(xz) \ \text{:-}\ q_0(x, z, z, z, z, x),\ \mathit{LengthZero}(z).$$
$$\mathit{LengthZero}(\lambda).$$

The informal idea is that the grammar recognizes a word w if and only if $S(w)$ is derived, hence $q_0(w, \lambda, \lambda, \lambda, \lambda, w)$ holds,[15] which will correspond precisely to the machine M halting in an accepting state when given the string w on the input tape, a blank work tape, and its heads in 0 position. The copy of w will be passed to each state nonterminal, and will be used both for checking elements of the input and to generate copies of strings for doing arithmetic over $0 \ldots n$.

We define a number of auxiliary predicates, such as a schema of clauses defining $\mathit{SameLength}(x, y)$ which produces exactly the tuples (w_1, w_2) where $|w_1| = |w_2|$, $\mathit{EmptyOrLengthOne}(x)$, $\mathit{TwiceAsLong}(x, y)$, etc. We can then easily define the arithmetical operations, e.g. if we define

$$\mathit{Mult2}(xy, z) \ \text{:-}\ \mathit{TwiceAsLong}(x, z), \mathit{NextState}(x).$$

then $\mathit{Mult2}(w, v_1)$ is derived by the grammar if and only if it derives $\mathit{NextState}(v_2)$, where w is an auxiliary copy of the input, and v_2 is any string twice as long as v_1 (but no longer than $|w|$).

Similar constructions define the other arithmetical operations. For each universal state symbol, we introduce a single production that rewrites it to a number of new states. For each existential state, we will have a number of productions which each rewrite it to a single new state. In both cases, a number of extra rules is necessary for evaluation of conditions; the transition itself must be broken up into a series of steps, each step corresponding to the application of one arithmetical operation; each step passes a sufficient number of copies of w to the next step to preserve the ability of doing modulo n arithmetic.

Hence we build a grammar that generates w if and only if the ATM M accepts w, completing the construction.

5. Discussion

The formal results obtained in this paper are summarized in figure 9. The conclusion that simple LMG is an important extension of LCFRS and PMCFG because it adds descriptions of the crucial constructions of co-ordination and crossed dependencies, but does not result in a noticeable increase in computational complexity, suggests that we might reconsider the details of the outlined class of mildly context-sensitive languages.

The essential property in the definition of mild context-sensitivity is of course the constant growth property. It is in a sense a more general statement of a k-pumping lemma. However, the pumping lemma spots the similarity between the languages $\{\mathtt{a}^n \mid n \text{ is prime}\}$ and $\{\mathtt{b}^*\mathtt{a}^n \mid n \text{ is prime}\}$, whereas the second language is not ruled out by the constant growth property. Attempts at refining the definition of constant growth without stating a complete pumping lemma have so far failed.

On the other hand, a pumping lemma seems to be too strong—for the extended Manaster-Ramer fragment does not satisfy k-pumpability for any fixed k; yet I feel it should be a mildly context-sensitive language, as it displays some form of 'linear growth of structures' in the same sense in which ordinary cross-serial dependent clauses do. It has been argued (see e.g. [Rad91]) that this linear growth of *structures* is really what 'constant growth' is meant to say.

[15] Note that this amounts to initializing rr with the value n rather than $\log n$. Although we could initialize it with $\log n$ (by adding a fairly complicated set of SLMG rules to compute that value), this is not necessary for the construction to succeed.

Formalism	*Increasing conditions on LMG form*	*Weakly equivalent to*
Generic LMG	—	recursive enumerability
Simple LMG	Bottom-up nonerasing, non-combinatorial	ILFP, PTIME
Nonerasing PMCFG	Top-down linear, top-down nonerasing	Standard PMCFG
LCFRS	Bottom-up linear	MCFG, MC-TAG
HG	Pairs only, restricted operations	TAG, LIG, CCG[VSW94]
CFG	Singletons	—

Figure 9: Hierarchical classification 3

Clearly PTIME, that is the languages described by simple LMG, is a family of languages larger than the class we want to define, because we don't want to call a language such as a^{2^n} mildly context-sensitive. On the other hand, a language such as the Chinese number series in [Rad91] does seem to satisfy the informal idea of 'linear growth of structures'. This leads to the conjecture that the interesting class of languages is the one that satisfies a property along the lines of the following *non*-uniform k-pumpability condition:

Definition 7 (finite pumpability) *Let L be a language. Then L is* finitely pumpable *if there is a constant c_0 such that for any $w \in L$ with $|w| > c_0$, there are a finite number k and strings $u_0, \ldots, u_k$ and $v_1, \ldots, v_k$ such that $w = u_0 v_1 u_1 v_2 u_2 \cdots u_{k-1} v_k u_k$, and for each i, $1 \leq |v_i| < n$, and for any $p \geq 0$, $u_0 v_1^p u_1 v_2^p u_2 \cdots u_{k-1} v_k^p u_k \in L$.*

This definition excludes exponentially growing languages, while admitting the conjoined cross-serial phrases and the offensive variants [Rad91] of the Chinese number series. This leads to the following rough attempt at a revised definition of mild context sensitivity:[16]

1. limited crossed dependencies, reduplication and parallelism
2. finite pumpability (definition 7)
3. polynomial parsing

Conclusions and further research

I have outlined a mathematically simplified version of the LMG formalism as presented in earlier work, and shown that the restricted class of *simple* LMG models exactly the polynomial time recognizable languages, and adequately describes essential properties (conjunction) of Dutch verb structure not described by any known smaller classes of grammars within PTIME.

However, simple LMG still describes a rather large class of grammars, including unnatural languages such as a^{2^n}. So now the question should be raised how we can exploit the extra power (hidden in the ability to do intersection), without allowing the unlimited reduplication given by PMCFG (multiple occurrence of variables on the LHS). It is perhaps a useful idea that LMG describing these unnatural languages contain potentially large 'equality' schemas, which could be considered as 'unnatural grammars'. The relative inadequacy of PMCFG w.r.t. simple LMG has not been made very concrete.

[16] I have chosen the word 'parallelism' in favour of conjunction, to stress that the essential property that makes conjunction hard to describe is the 'structural similarity' it seems to require of the conjuncts. The class satisfying just properties 2. and 3. intuitively implies property 1. without "limited", and satisfies a number of abstract closure properties, see e.g. [Gin75] (but it seems to lack closure under intersection with regular languages). As in the original definition of MCS, the first item is kept vague; as such it probably characterizes properties that remain to be formalized; perhaps by stating 1. as "including a number of formal languages" such as the copy language (1.11) and adding a requirement 4. Is a (pre/semi) AFL [Gin75].

The proposed revised definition of mild context-sensitivity in section 5 could be seen as a guideline for finding the proper middle way between LCFRS and simple LMG.

The informal "predicate descent" recognizer for LCFRS/LMG is a deterministic PTIME algorithm that specifies concrete, and apparently tight upper bounds in terms of the sizes of both input and grammar. It also gives the most simple solution to the problem of left-recursion in context-free parsing, while still maintaining a time complexity of $O(n^3)$. I have written a complete parser package which achieves a higher efficiency than the example in this paper by calculating terminal corner and occur tables.

The fixed-point interpretation of LMG using integer positions can be extended easily to cover polynomial-time recognition from descriptions of *nonlinear* finite structures, by replacing the integer positions with the nodes of an arbitrary finite structure, such as a simple line drawing or a string lattice.

ACKNOWLEDGEMENTS

The author is supported by SION grant 612-317-420 of the Netherlands Organization for Scientific Research (NWO). Marcus Kracht, Jens Michaelis and Jan van Eijck made important suggestions on a number of crucial issues addressed in this paper.

REFERENCES

[CKS81] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *JACM*, 28:114–133, 1981.

[Gin75] S. Ginsburg. *Formal Languages.* North-Holland/Am. Elsevier, Amsterdam, Oxford/New York, 1975.

[Gro95a] Annius V. Groenink. Formal Mechanisms for Left Extraposition in Dutch. In *Proceedings of the 5th CLIN (Computational Linguistics In the Netherlands) meeting, November 1994, Enschede*, November 1995.

[Gro95b] Annius V. Groenink. Literal Movement Grammars. In *Proceedings of the 7th EACL Conference, University College, Dublin*, March 1995.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979.

[Jos85] Aravind Joshi. Tree Adjoining Grammars: How much Context-Sensitivity is Required to Provide Reasonable Structural Descriptions? In A. Zwicky, editor, *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, pages 206–250. Cambridge University Press, 1985.

[KNSK92] Y. Kaji, R. Nakanishi, H. Seki, and T. Kasami. The Universal Recognition Problems for Parallel Multiple Context-Free Grammars and for Their Subclasses. *IEICE*, E75-D(4):499–508, 1992.

[Lee93] René Leermakers. *The Functional Treatment of Parsing.* Kluwer, The Netherlands, 1993.

[MK96] Jens Michaelis and Marcus Kracht. Semilinearity as a syntactic invariant. Paper read at the Logical Aspects of Computanional Linguistics conference, Nancy, 23–25 september, 1996.

[MR87] Alexis Manaster-Ramer. Dutch as a formal language. *Linguistics and Philosophy*, 10:221–246, 1987.

[Pol84] Carl J. Pollard. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language.* PhD thesis, Stanford University, 1984.

[Rad91] Daniel Radzinski. Chinese Number-Names, Tree Adjoining Languages, and Mild Context-Sensitivity. *Computational Linguistics*, 17(3):277–299, 1991.

[Rou88] William C. Rounds. LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. *Computational Linguistics*, 14(4):1–9, 1988.

[SMFK91] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229, 1991.

[VS87] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars.* PhD thesis, Univ. of Pennsylvania, 1987.

[VSW94] K. Vijay-Shanker and D. J. Weir. The Equivalence of Four Extensions of Context-Free Grammar. *Math. Systems Theory*, 27:511–546, 1994.

[Wei88] David J. Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* PhD thesis, University of Pennsylvania, 1988.