# A Simple Uniform Semantics for Concatenation-based Grammar

Annius V. Groenink
CWI, P.O. Box 94079
1090 GB Amsterdam
avg@cwi.nl

January 3, 1996

**Abstract**

We define a more formal version of literal movement grammar (LMG) as outlined in [Gro95c], in such a way that it provides a simple framework that incorporates a large family of grammar formalisms (Head Grammar [Pol84], LCFRS, [Wei88]), PMCFG, [KNSK92] and String Attributed Grammars [Eng86]). The semantics is (both in rewriting and least fixed point definitions) simple and elegant, and sheds some new light on shared properties of the mentioned formalisms. We then define a restricted version called *simple* LMG and show that it generates languages that are not mildly context sensitive, yet preserves the polynomial time recognition property of LCFRS.

## Introduction

This paper consists of three parts. In the first, we propose a more elementary notation of the LMG formalism as introduced the first LMG paper [Gro95c], and call it *predicate literal movement grammar*. The generalization has a twofold purpose. First, it allows us to give a more elementary semantics, both in rewriting style and as a fixed point operator on sets of tuples of terminal strings. Then, we see how it allows us to put a series of tuple-based grammar formalisms of increasing recognising power (LCFRS [Wei88], MCFG, PMCFG [KNSK92], and LMG) in a uniform semantic framework.

In the second part we look at least fixed point interpretations, followed by a discussion on complexity of recognition. We introduce a restricted version of our formalism, *simple* LMG, and show that it strictly extends LCFRS and PMCFG, yet preserves polynomial time recognition. More precisely, the class of languages described by simple LMG is exactly the class PTIME. The PTIME fragment can be extended to cover input data in the form of a lattice (such as in speech analysis) or arbitrary ordered finite structures (think of pattern recognition in vector based images).

The third part wraps up the story with a classification of the formalisms that have been discussed, and a discussion on mild context-sensitivity and polynomial time. Among other things, we give an example showing that LMG can give accounts of essential structural phenomena in Natural Language known to be beyond the scope of linear context-free rewriting systems.

## 1 The Predicate LMG Framework

*Literal movement grammar* (LMG, henceforth *slash-style* LMG) was introduced by the author in [Gro95c], as a formalism which takes a strong left-to-right top-down view on "literal" filler-gap relocation, i.e. passing the terminal words scanned in filler positions down the derivation until they are matched up by a correspondingly typed gap, in the form of what in that paper was called a 'slash item'.

We will redefine LMG here in a version which has more formal appeal.

**Definition 1** A *predicate literal movement grammar* (LMG[1]) is a tuple $G = (N, T, V, S, P)$ where $N, T$ and $V$ are mutually disjoint sets of *nonterminal symbols*, *terminal symbols* and *variable symbols*, respectively, $S \in N$,

[1] The previous papers [Gro96] and [Gro95b] refer to predicate LMG as CPG (concatenative predicate grammar), but I agree with the readers who thought that giving what is essentially a different representation of the same formalism, a name of its own, might lead to unnecessary confusion.

$$\dfrac{\mathrm{NP}(\texttt{John}) \quad \dfrac{\dfrac{\mathrm{VT}(\texttt{saw}) \quad \mathrm{NP}(\texttt{Mary})}{\mathrm{VP}(\texttt{saw, Mary})} \quad \dfrac{\mathrm{VT}(\texttt{admired}) \quad \mathrm{NP}(\texttt{Mary})}{\mathrm{VP}(\texttt{admired, Mary})}}{\mathrm{VP}(\texttt{saw and admired, Mary})}}{\mathrm{S}(\texttt{Mary, John saw and admired})}$$

Figure 1: Derivation of *Mary, John saw and admired*

and $P$ is a set of *clauses* (which correspond to the notion of a *production* in CFG or the slash-style LMG from [Gro95c]) of the form

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

where $\phi, \psi_1 \ldots \psi_m$ are *predicates* of the form

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

where $t_i \in (T \cup V)^*$.

A predicate LMG clause is *instantiated* by substituting a string $w \in T^*$ for each of the variables occurring in the clause.

**Definition 2 (semantics)** Let $G = (N, T, V, S, P)$ be a predicate LMG. Then $G$ is said to *recognise* a string $w$ if $\vdash^G S(w)$ where $\vdash^G$ is defined inductively as follows: if

$$\begin{aligned} A(w_1, \ldots, w_n) \ \texttt{:-} \ & B_1(v_{11}, \ldots, v_{1n_1}), \\ & \ldots, \\ & B_m(v_{m1}, \ldots, v_{mn_m}) \end{aligned}$$

is an instantiation of a clause in $P$, and for each $1 \leq k \leq m$

$$\vdash^G B_i(v_{k1}, \ldots, v_{kn_k})$$

then

$$\vdash^G A(w_1, \ldots, w_n)$$

Note that $m = 0$ is the base case (zero antecedents).

**Example 1 (Topicalization/Conjunction)** An example of an LMG is the following grammar

$$\begin{aligned} &\mathrm{S}(n \text{ ", " } mv) && \texttt{:-} \ \mathrm{NP}(m), \ \mathrm{VP}(v, n) \\ &\mathrm{VP}(v_1 \text{ "and" } v_2, \ n) && \texttt{:-} \ \mathrm{VP}(v_1, n), \ \mathrm{VP}(v_2, n) \\ &\mathrm{VP}(v, n) && \texttt{:-} \ \mathrm{VT}(v), \ \mathrm{NP}(n) \end{aligned}$$

which derives topicalized sentences of the form

*Mary, John saw and admired and kissed.*

An example derivation is shown in figure 1.

**Example 2 (CFGs)** Every CFG can be defined as a predicate LMG; this is analogous to the translation of a CFG into Prolog. Every nonterminal will have one argument which represents its yield. E.g. $\mathrm{S} \rightarrow \mathrm{NP\ VP}$ will become $\mathrm{S}(xy) \rightarrow \mathrm{NP}(x), \mathrm{VP}(y)$.

We can do the same to slash style LMG [Gro95c].

**Example 3 (Slash-style LMG)** The original version of LMG using slash items [Gro95c] [Gro96] can be translated directly into predicate LMG, by introducing an extra argument for each nonterminal. Vice versa, in some predicate LMG, such as the grammar from example 1, one argument that clearly encodes simple left-to-right phrase structure can be removed so as to give a slash-style LMG; the slash-style LMG notation for the grammar in the example is

$$\begin{array}{lcl} \mathrm{S} & \rightarrow & n \text{ ","} \ \mathrm{NP} \ \mathrm{VP}(n) \\ \mathrm{VP}(n) & \rightarrow & \mathrm{VP}(n) \text{ "and"} \ \mathrm{VP}(n) \\ \mathrm{VP}(n) & \rightarrow & \mathrm{VT} \ (\mathrm{NP}/n) \end{array}$$

Note how the slash-style notation is, especially in the case of natural language grammars, more appealing to traditional intuition (but also actually more readable, as it involves less variables per rule).

**Example 4 (Head Grammar)** A head grammar is a CPG that operates on pairs. There are three types of (modified) head grammar rule: a *wrapping* production:

$$A \rightarrow \mathrm{wrap}(B, C)$$

is represented in LMG as

$$A(b_1 c_1,\ a_2 b_2) \ \texttt{:-}\ B(b_1, b_2),\ C(c_1, c_2)$$

and one of the two types of concatenating rule as

$$A(a_1,\ a_2 b_1 b_2) \ \texttt{:-}\ B(b_1, b_2),\ C(c_1, c_2)$$

**Example 5 (LCFRS and PMCFG)** Linear context-free rewriting systems (LCFRS) and parallel multiple context-free grammars (PMCFG) are no more than complex notation for restrictions of predicate LMG. Productions in both LCFRS [Wei88] and PMCFG [KNSK92] are of the form

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

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

$$\begin{array}{l} f(\langle x_{11}, \ldots, x_{1n_1} \rangle, \\ \quad \ldots, \\ \quad \langle x_{m1}, \ldots, x_{mn_m} \rangle) \ = \ \langle t_1, \ldots, t_n \rangle \end{array}$$

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 $t_1, \ldots, t_n$. For PMCFG, there is no such restriction.

In the predicate notation of these formalisms, the separate function definition disappears; we simple write a rule as

$$\begin{array}{ll} A(t_1, \ldots, t_n) \ \texttt{:-} & B_1(x_{11}, \ldots, x_{1n_1}), \\ & \ldots, \\ & B_m(x_{m1}, \ldots, x_{mn_m}) \end{array}$$

**Example 6 (Attribute Grammar)** [Eng86] We have two nonterminals: a start symbol $Z$ with a designated attribute $d$; and $A$ with an inherited attribute $i$ and a synthesized attribute $s$. The grammar and the attribute rules are as follows:

$$\begin{array}{lcll} Z & \rightarrow & \mathsf{a}\ A & Z.d := A.s,\ \ A.i := \mathsf{a} \\ Z & \rightarrow & \mathsf{b}\ A & Z.d := A.s,\ \ A.i := \mathsf{b} \\ A^{(1)} & \rightarrow & \mathsf{a}\ A^{(2)} & A^{(1)}.s := A^{(2)}.s\ A^{(2)}.s, \\ & & & A^{(2)}.i := A^{(1)}.i\ \mathsf{a} \\ A^{(1)} & \rightarrow & \mathsf{b}\ A^{(2)} & A^{(1)}.s := A^{(2)}.s\ A^{(2)}.s, \\ & & & A^{(2)}.i := A^{(1)}.i\ \mathsf{b} \\ A & \rightarrow & \lambda & A.s := A.i\ \mathsf{c}\ A.i\ \mathsf{c} \end{array}$$

The context-free backbone recognises arbitrary nonempty strings $w$ over the alphabet $T = \{\mathsf{a}, \mathsf{b}\}$, and the output value for $w \in T^*$ is $(w\mathsf{c})^{2^{|w|}}$.

The grammar is represented in predicate LMG as

$$\begin{array}{lll} S(x) & \text{:-} & Z(x, y) \\ Z(x,\ \mathsf{a}z) & \text{:-} & A(\mathsf{a};\ x,\ z) \\ Z(x,\ \mathsf{b}z) & \text{:-} & A(\mathsf{b};\ x,\ z) \\ A(y;\ xx,\ \mathsf{a}z) & \text{:-} & A(y\mathsf{a};\ x,\ z) \\ A(y;\ xx,\ \mathsf{b}z) & \text{:-} & A(y\mathsf{b};\ x,\ z) \\ A(x;\ x\mathsf{c}x\mathsf{c}, \lambda). & & \end{array}$$

Note that there is no formal difference between inherited and synthesized attributes—this is in line with the observation that designating an attribute as synthesized or inherited is, once we look at the equations as *constraints*, semantically irrelevant, and should rather be considered as a hint toward a concrete program or parser that evaluates the "value" of a sentence. The translated SAG is a curious type of LMG, as the S production "throws away" the value $y$. It will turn out that this type of LMG has less favourable computational properties than the ones defined so far; the example serves mainly to stress that LMG provides a very simple semantics, and a compact notation for attribute grammars.

# 2 Least Fixed Points and Complexity

The examples have already shown how LMG subsumes the chain $\mathrm{CFG} \subseteq \mathrm{HG} \subseteq \mathrm{LCFRG} \subseteq \mathrm{PMCFG}$ of formalisms of increasing generative capacity. It is actually known that these are all strict inclusions; moreover the fixed recognition problems for all these formalisms are in PTIME. We will now answer the question how, in the spirit of these formalisms, we can characterize PTIME itself.

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 with the Chandra-Kozen-Stockmeyer [CKS81] result on the correspondence between deterministic and alternating Turing machine computations then gives a deterministic PTIME complexity for recognition.

## 2.1 Fixed point interpretations of LMG

Before we redefine LMG to talk about integer positions in strings, let's present the semantics of LMG in such a way that we interpret the nonterminals as relations over terminal strings. Let $G = (N, T, V, S, P)$ be an LMG. Let NA be the set of *assignments* to the nonterminals: functions $\rho$ 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 interpretation function as an argument and producing a new function, defined as follows:

If

$$\begin{array}{lll} A(w_1, \ldots, w_n) & \text{:-} & B_1(v_{11}, \ldots, v_{1n_1}), \\ & & \ldots, \\ & & B_m(v_{m1}, \ldots, v_{mn_m}) \end{array}$$

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

It is easily seen that $[\![G]\!]$ is a continuous and monotonic operator on the complete partial order $(\mathrm{NA}, \sqsubseteq)$ defined by

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

let $\rho_1, \rho_2$ be two assignments, and $\rho_1 \sqsubseteq \rho_2$. Let $a \in ([\![G]\!]\rho_1)(A)$. Then we have the clause $R$ and the tuples $b_1, \ldots, b_m$ from the definition, and $b_i \in \rho_1(B_i)$. It now follows that for each $i$, $b_i \in \rho_2(B_i)$, hence $a \in ([\![G]\!]\rho_2)(A)$. So $[\![G]\!]$ is monotonic. Because the partial order is defined as componentwise set inclusion over a sufficiently general universe, it follows automatically that $[\![G]\!]$ is also continuous.

We can now validly define the interpretation of a grammar to be the least fixed point of $[\![G]\!]$:

$$\mathcal{I}_G \;=\; \bigsqcup_{k=0}^{\infty} [\![G]\!]^k(\lambda A.\emptyset)$$

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 recognised by the LMG in the traditional sense.

It is easy to check that the rewriting semantics and the fixed point semantics are equivalent. The fixed point semantics is a useful tool for several purposes. First of all, it gives a more detailed yet mathematically elegant interpretation of grammar. More detailed, because it does not merely characterize the language generated by a single designated start symbol, but characterizes the derivational behaviour of the grammar, without looking at single derivations in particular.

We want 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.

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 $\mathrm{NA}_w$ is the set of *integer nonterminal assignments* $\rho$ mapping a nonterminal to a set of tuples of pairs of integers. Then $[G]_w$ is defined

If

$$\begin{array}{l} A(a_{i_1}\cdots a_{j_1}, \ldots, a_{i_n}\cdots a_{j_n}) \\ \quad :\text{-}\;\; B_1(a_{i_{11}}\cdots a_{j_{11}}, \ldots, a_{i_{1n_1}}\cdots a_{j_{1n_1}}), \\ \qquad \ldots, \\ \qquad B_m(a_{i_{m1}}\cdots a_{j_{m1}}, \ldots, a_{i_{mn_m}}\cdots a_{j_{mn_m}}) \end{array}$$

is an instantiation of a clause in $P$, and for each $1 \le k \le m$, $\langle (i_{k1}, j_{k1}), \ldots, (i_{kn_k}, j_{kn_k})\rangle \in [G]_w(B_k)$, then $\langle (i_1, j_1)\ldots, (i_n, j_n)\rangle \in [G]_w(A)$.

It is important to see that what is done here, is *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

$$\begin{array}{lll} A(w_1, \ldots, w_n) & :\text{-} & B_1(v_{11}, \ldots, v_{1n_1}), \\ & & \ldots, \\ & & B_m(v_{m1}, \ldots, v_{mn_m}) \end{array}$$

such that $w_1, \ldots, w_n$ are substrings of $w$, but the $v_{ij}$ are not, then this instantiation will be ignored in the integer LFP semantics. Hence we want to rule out this type of clause. I.e., we want to make sure that $w_1, \ldots, w_n$ 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 rule that is talking about integer positions instead of strings.

**Definition 3 (simple LMG)** An LMG is called *simple* if its clauses $R \in P$ are all of the form

$$\begin{array}{lll} A(t_1, \ldots, t_n) & :\text{-} & B_1(x_{11}, \ldots, x_{1n_1}), \\ & & \ldots, \\ & & B_m(x_{m1}, \ldots, x_{mn_m}) \end{array}$$

and each of the $x_{ij}$ appears at precisely once in $t_1, \ldots, t_n$.

## 2.2 Simple LML is in PTIME

There are two ways to show that the languages generated by simple LMG can be recognised 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]; this is quite simple and is sketched in [Gro95a]. The VP rules in the grammar for English topicalization from conjunctive verb phrases

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

for example, are translated into a single formula

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

The correspondence between the languages defined by ILFP and those recognised by logspace-bounded alternating Turing machines (ATM) shown in [Rou88] then completes the argument.

Here we will sketch a more informal recognition algorithm, which however gives a better indication of what a possible implementation would look like (as it is a deterministic algorithm).

We take the integer representation of LMG grammars, in which every argument is represented as a pair $(l, r)$ of integer indices, as a point of departure (in fact the ILFP translation given above will serve for this purpose).

Given an input string $w$ of length $n$, construct memo tables containing a boolean value for each possible predicate $A(l_1, r_1,\ \ldots,\ l_n, r_n)$, where $l_i, r_i$ are integer values ranging from 0 to $n$. Reset all the table entries to zero. 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:

```
VP(i,j,k,l):
   if VP(i,j,k,l) memoed
   then
      return memoed value
   else
      memo VP(i,j,k,l) as False

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

If $p$ is the largest number of integer predicate arguments and $m$ is the largest number of bound variables in each disjunct of the ILFP version of an LMG rule, then the recogniser needs to do $O(n^m)$ calls or look-ups for each

of the predicates. Since there are $O(n^p)$ predicates, recognition can be performed in deterministic $O(n^{p+m})$ time and $O(n^p)$ memoing storage. Constructing a minimally informative parse forest would require $O(n^{p+m})$ space.[2]

The bound given here seems tight. The rules of a binary, modified head grammar, such a the wrapping rule:

$$A(a_1 b_1, b_2 a_2) \ :- \ B(a_1, a_2),\ C(b_1, b_2)$$

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

$$\begin{aligned} A(i, j,\ l, k) \ \Leftrightarrow \ \exists i', l'.\ (\ & i \leq i' \leq j \\ & \wedge\ l \leq l' \leq k \\ & \wedge\ B(i, i',\ l', k) \\ & \wedge\ C(i', j,\ l, l')\ ) \end{aligned}$$

The general recogniser for HG we obtain by applying the sketched algorithm has the well known upper time bound of $O(n^6)$.

## 2.3 Simple LML subsumes PTIME

We proceed exactly as in [Rou88]. It is a known result that $PTIME = ASPACE(\log\ n)$. 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 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

$$\begin{aligned} & S(xz) \ :- \ q_0(x, z, z, z, z, x),\ LengthZero(z) \\ & LengthZero(\lambda). \end{aligned}$$

The informal idea is that the grammar recognises a word $w$ if and only if $S(w)$ is derived, hence $q_0(w, \lambda, \lambda, \lambda, \lambda, w)$ holds,[3] 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 $SameLength(x, y)$ which produces exactly the tuples $(w_1, w_2)$ where $|w_1| = |w_2|$, $EmptyOrLengthOne(x)$, $TwiceAsLong(x, y)$, etc. We can then easily define the arithmetical operations, e.g. if we define

$$\begin{aligned} Mult2(xy, z) \ :- \ & TwiceAsLong(x, z), \\ & NextState(x) \end{aligned}$$

then $Mult2(w, v_1)$ is derived by the grammar if and only if it derives $NextState(v_2)$, 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

[2] 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.

[3] 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.

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.

### 2.4 Recognition of nonlinear finite structures

We can eliminate the terminals ($T$) in the definition of LMG, instead talking about how we can recognise *derived relations* (the phrases) between *positions* in a sentence given axioms defining a set of *basic relations* (the words) between these positions.

**Definition 4 (terminal free LMG)** A *terminal-free* LMG is a tuple $(N, V, S, P)$, where $N, V$ and $S$ are as before; productions are as for predicate LMG, but the arguments of nonterminals are now only allowed to be single variables $x \in V$. Let $U$ be an arbitrary universe (a set); a terminal free LMG clause is *instantiated* by substituting an element from $U$ for each of the variables. The semantics is then as follows; for any instantiated predicate $\phi$, we have

$$\phi \vdash^G \phi$$

and if

$$\begin{aligned} A(w_1, \ldots, w_n) \ :- \ & B_1(v_{11}, \ldots, v_{1n_1}), \\ & \ldots, \\ & B_m(v_{m1}, \ldots, v_{mn_m}) \end{aligned}$$

where $w_i, v_{ij} \in U$, is an instantiation of a clause in $P$, and for each $1 \leq k \leq m$

$$\Gamma_k \vdash^G B_i(v_{k1}, \ldots, v_{kn_k})$$

then

$$\Gamma_1, \ldots, \Gamma_k \vdash^G A(w_1, \ldots, w_n)$$

(where $\Gamma_k$ are sets of predicates).

String-based LMG is an instance of this very general definition; we take $U$ to be the set of nonnegative integers and we encode the string $w = a_0 a_1 \cdots a_{n-1}$ by adding $a_0, \ldots, a_{n-1}$ to the set of nonterminals $N$, and postulating the axioms $a_0(0, 1), \ldots, a_{n-1}(n-1, n)$. We transform the grammar $G$ to a grammar $G'$ over integer positions instead of strings, as in the formulae in section 2.2; the notion of derivability ($\vdash^G \mathrm{S}(w)$) is replaced with

$$a_0(0, 1), \ldots, a_{n-1}(n-1, n) \vdash^G \mathrm{S}(0, n).$$

Clearly this is not the only interpretation we can imagine. As the form of the axioms allows us to define any finite structure over points in an arbitrary universe $U$, we are now no longer prohibited from defining a *string lattice* or even any graph; if $U$ (or the part of $U$ addressed in the axioms) is finite, the sketched recognition algorithm will still be polynomial in terms of the number of points. So it seems that this definition extends the scope of tuple-based grammar to the discussion of complexity of more general forms of pattern recognition.

## 3 Classification & Discussion

We have seen examples of how CFG, HG, LCFRS and PMCFG are represented in the predicate LMG framework. It is known that these are of strictly growing generative capacity: HG can generate the 3-counting language $a^n b^n c^n$ which is not context-free; LCFRS can generate arbitrary counting languages $a_1^n a_2^n \cdots a_k^n$ (for any $k$), but the languages generated by LCFRS satisfy an extended form of pumping lemma, pumping an (even) number $k$ of substrings.

**Lemma 1 (pumping for LCFRS/MCTAG)** *Let the language $L$ be generated by an LCFRS. Then there are constants $n, k$ such that for any $w \in L$ with $|w| > n$, there are 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 any $p \geq 1$, $u_0 v_1^p u_1 v_2^p u_2 \cdots u_{k-1} v_k^p u_k \in L$.*

| *Formalism* | *Increasing conditions on CPG form* | *Weakly equivalent to* |
|---|---|---|
| Generic LMG | — | recursive enumerability |
| SAG | First argument of nonterminals does not interact with the others, and is limited to concatenation—i.e. a context free grammar | — |
| Bounded LMG | Length of terminal strings in derivations is polynomially (linearly) bounded in terms of the length of the input string | EXP-POLY time (CLFP, EXPTIME) |
| 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 |
| CFG | Singletons | — |

Figure 2: Hierarchical classification

The PMCFG

$S(xx)$ :- $S(x)$
$S(a)$.

generates the language $a^{2^k}$, which does not grow constantly and hence clearly does not satisfy the pumping lemma.

However, as with context free grammars and LCFRS, subderivations of PMCFG can be freely substituted, hence PMCFG is still closed under arbitrary homomorphism.

To show that simple LMG is of strictly stronger generative capacity, we make two observations. First, simple LMG is closed under intersection: Take two simple LMGs $G_1$ and $G_2$ whose start symbols are $\mathrm{S}_1$ and $\mathrm{S}_2$ respectively. Then combine the clauses of $G_1$ and $G_2$ (renaming nonterminals where necessary) and add the clause

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

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 generates the intersection of $G_1$ and $G_2$.

The second observation is that we can translate any PMCFG to a simple LMG. Simple LMG does not allow a variable to appear more than once on the LHS of a clause: e.g. the PMCFG clause

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

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 clause by the fragment

$\mathrm{A}(x, yz)$ :- $\mathrm{B}(x),\ \mathrm{C}(y),\ \mathrm{Eq}(y, z)$
$\mathrm{Eq}(\mathsf{a}x, \mathsf{a}y)$ :- $\mathrm{Eq}(x, y)$ for every $\mathsf{a} \in T$
$\mathrm{Eq}(\lambda, \lambda)$.

So simple LML subsumes PMCFL. Now suppose PMCFL and simple LML would be equal, then they would be closed under homomorphism and intersection, which implies that they generate all r.e. languages, and would not be decidable. So we must conclude that PMCFL is not closed under intersection, LMG is not closed under homomorphism, and LML strictly includes PMCFL.

For a full classification of the different formalisms in their predicate LMG versions, we introduce some terminology.

**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:

$$A(t_1, \ldots, t_n) \text{ :- } B_1(s_{11}, \ldots, s_{1n_1}),\ \ldots,\ B_m(s_{m1}, \ldots, s_{mn_m})$$

then

- $R$ is *bottom-up linear* if no variable $x$ appears more than once in $t_1, \ldots, t_n$.
- $R$ is *top-down linear* if no variable $x$ appears more than once in $s_{11}, \ldots, s_{mn_m}$.
- $R$ 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* 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.

So 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.

In short, we have the hierarchical classification shown in figure 2.[4]

## 3.1 Mild Context-Sensitivity and Polynomial Time

So far we have seen that LCFRS and PMCFG can be extended to simple LMG, which generates a strictly larger class of grammars, but still has polynomial time recognisability; moreover the top-down recognition algorithms for the different types of grammar are not essentially different. Does simple LMG give us an essential increase in expressivity?

Presentations of LCFRS usually go with the definition of *mild context-sensitivity* (MCS), outlined by Joshi as the class of languages

1. with a limited capacity for describing crossed dependencies
2. recognisable in polynomial time
3. satisfying the constant growth property, that is [Wei88] the language $L$ has associated to it 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 $c \in C$.

The constant growth property is in a sense a more general statement of the LCFRS pumping lemma. The statement of MCS is motivated by the desire to define classes of grammar which are severely limited in capacity, yet have sufficient strength to describe the basic structure of natural language syntax. It has always been proposed as an *attempt* to characterize such a class, and there has in particular been a number of arguments that the constant growth property is not satisfactory: Manaster-Ramer [Rad91] pointed out that while $\{\mathtt{a}^n \mid n \text{ is prime}\}$ does not have the constant growth property, its perverted cousin $\{\mathtt{b}^*\mathtt{a}^n \mid n \text{ is prime}\}$ does.

The following example shows a fragment of Dutch which is constant growth but does not satisfy the LCFRS pumping lemma.

## 3.2 Example

[MR87] gives the following example of a trans-tree adjoining fragment of Dutch, containing sentences such as:

. . . dat Jan Piet Marie liet opbellen,
that made call
hoorde uitnodigen,
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]*

[4] The figure includes a class (bounded LMG) not treated in this paper, which corresponds to the least fixed point calculus CLFP in [Rou88].

The fragment can be characterized as follows:

$$
\begin{aligned}
\ldots \text{ dat Jan Piet Marie } \mathrm{NP}^k & \\
\text{liet } \mathrm{VR}^k & \text{ opbellen,} \\
\text{hoorde } \mathrm{VR}^k & \text{ uitnodigen,} \\
\text{hielp } \mathrm{VR}^k & \text{ ontmoeten} \\
\text{en zag } \mathrm{VR}^k & \text{ omhelzen.}
\end{aligned}
$$

The fragment does not satisfy the pumping lemma for TAG and Head Grammar which says that we can create constantly growing subfragments by pumping 4 substrings. Obviously, in the example, 5 strings will need to be pumped.

The pumping lemma for LCFRS states that there is a number $k$ such that at most $k$ strings need to be pumped. Increasing the number of conjuncts in Manaster-Ramer's example hence provides, given any LCFRS, an argument that it cannot describe the fragment.

The following simple (slash-style) LMG does generate Manaster-Ramer's fragment [MR87] as sketched in the examples, with an unbounded number of conjuncts.

$$
\begin{array}{lcl}
\mathrm{S} & \rightarrow & \texttt{...dat NP } n \text{ VP}(n) \\
 & & \\
\mathrm{VP}(n) & \rightarrow & \overline{\mathrm{V}}(n) \\
\mathrm{VP}(n) & \rightarrow & \mathrm{VP}^C(n) \\
 & & \\
\mathrm{VP}^C(n) & \rightarrow & \overline{\mathrm{V}}(n) \texttt{ "," } \mathrm{VP}^C(n) \\
\mathrm{VP}^C(n) & \rightarrow & \overline{\mathrm{V}}(n) \texttt{ en } \overline{\mathrm{V}}(n) \\
 & & \\
\overline{\mathrm{V}}(n) & \rightarrow & \mathrm{VT}\ (\mathrm{NP}/n) \\
\overline{\mathrm{V}}(nm) & \rightarrow & \mathrm{VR}\ (\mathrm{NP}/n)\ \overline{\mathrm{V}}(m) \\
 & & \\
\mathrm{NP} & \rightarrow & \texttt{Jan, Marie, Piet} \\
\mathrm{VT} & \rightarrow & \texttt{opbellen, uitnodigen...} \\
\mathrm{VR} & \rightarrow & \texttt{zag, horen, hielp...}
\end{array}
$$

## 3.3 Revising MCS

The ability to describe crossed dependencies in conjunctive VPs is clearly a desirable feature of a grammar formalism in the spirit of LCFRS. This could be seen as an argument that the constant growth property in the definition of LCFRS should in fact *not* be strengthened to a pumping lemma.

The remaining question is whether there are clearly 'unnatural' languages that are in PTIME but which we want to rule out; one may think of $a^{2^n}$. If we do not rule these out, then the 'limited capacity for describing crossed dependencies' is obvious, and mild context-sensitivity collapses into the single predicate 'recognisable in polynomial time' which is equivalent to 'generated by a simple LMG'.

The pumping lemma for LCFRS is too weak (it doesn't rule out the prefixed prime language $\{\mathsf{b}^*\mathsf{a}^n \mid n \text{ is prime}\}$), whereas LCFRS does not cater for the example of crossed dependencies and unbounded conjunction.

I believe that the 'flaws' in the definition of constant growth and pumping lemma should be circumvented by claiming that there is a fixed bound to the size of the 'unpumped' part of the string, i.e. there is some form of a finite 'basis'. A revised pumping lemma would be along the lines of

**Lemma 2 (strong finite pumping)** *Let $L$ be a language. Then $L$ is* strongly finitely pumpable *if there are constants $n, k$ such that for any $w \in L$ with $|w| > n$, there are strings $u_0, \ldots, u_k$ and $v_1, \ldots, v_k$ such that $\sum u_i + \sum v_i < n$, there is a $p$ such that $w = u_0 v_1^p u_1 v_2^p u_2 \cdots u_{k-1} v_k^p u_k$, and for any $p \geq 1$, $u_0 v_1^p u_1 v_2^p u_2 \cdots u_{k-1} v_k^p u_k \in L$.*

This does rule out the prefixed prime language because instead of claiming that we can make *larger* strings from a given string, we are saying that it can be pumped from a string shorter than a constant fixed for the language.

Since this is a stronger pumping lemma than that known for LCFRS/MCTAG, it is not what we are after, since it will again rule out the unbounded conjunctions. However, if we could weaken this version of the pumping lemma into a revised definition of constant growth, it would seem to characterize a valuable property.

### 3.4 Conclusions

We have outlined a formal version of the LMG formalism as presented earlier, and shown that we can define a restriction which models exactly the polynomial time recognisable languages. Moreover, this restriction, the *simple* LMG, can describe essential fragments of Dutch verb structure which cannot be described by any known smaller classes of grammars within PTIME.

While LCFRS was previously the best known approximation of the ideal class of 'mildly context-sensitive' grammars, we believe to have shown by our examples of Dutch, that it is not strong enough; however the alternative presented here is clearly too strong—unnatural languages such as $a^{2^n}$ can now be described. So now the question should be raised how we can exploit the extra power (hidden in the ability to do intersection), without allowing the reduplication given by PMCFG (multiple occurrence of variables on the LHS), which seems to give rise to the 'unnatural' languages. It should be noted that LMG grammars which generate these unnatural languages, contain 'equality' predicates which consist of one clause for every symbol in the terminal alphabet, which could indeed be seen as an 'unnatural grammar'.

The proposed general recogniser for LCFRS/LMG which gives the proper bounds for the class of HG (including TAG, LIG and CCG), if informally presented, is as far as we know the first of its kind written on paper. One of the reasons there have not been attempts to define such algorithms before is the claim [KNSK92] that universal recognition of LCFRS is PSPACE-complete and universal recognition of PMCFG is EXP-POLY time complete. However, these results involve constructions which generate grammars whose size is proportional to a given input string, and hence provide only a limited picture of computational reality. Thoughts on possible improvement by reducing top-down prediction based on terminal corners are in progress, and an implementation of a parser based on LMG is to be expected in the near future.

## Acknowledgements

The author is supported by SION grant 612-317-420 of the Netherlands Organization for Scientific Research (NWO). Part of this paper is an elaborated version of the extended abstract [Gro95a] of my talk at the fourth Mathematics of Language workshop in Philadelphia, November 1995. Other papers are available webwise through `http://www.cwi.nl/~avg/`.

## References

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

[Eng86] Joost Engelfriet. The Complexity of Languages Generated by Attribute Grammars. *SIAM J. Comput.*, 15(1):70–86, 1986.

[Gro95a] Annius V. Groenink. An Elegant Grammatical Formalism for the Polynomial-time recognisable Languages. Extended abstract of paper presented at the fourth Mathematics of Language workshop (MOL4), Univ. of Pennsylvania, 10 1995.

[Gro95b] 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*, 11 1995.

[Gro95c] Annius V. Groenink. Literal Movement Grammars. In *Proceedings of the 7th EACL Conference, University College, Dublin*, 3 1995.

[Gro96] Annius V. Groenink. A Unifying Framework for Concatenation-based Grammar Formalisms. In *Proceedings of Accolade 1995, Amsterdam*, 1996.

[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.

[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, Standford 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.

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