A Unifying Framework for Concatenation Based Grammar Formalisms

Annius V. Groenink
CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
avg@cwi.nl

September 21, 1995

Abstract

Linear Context Free Rewriting Systems (LCFRS, [Wei88]) are a general class of trans-context-free grammar systems; it is the largest well-known class of mildly context sensitive grammar; languages recognized by LCFRS strictly include those generated by the HG, TAG, LIG, CCG family. *(Parallel) Multiple Context-Free Grammar* (PMCFG, [KNSK92]) is a straightforward extension of LCFRS. *Literal Movement Grammars,* introduced by the author of this paper in [Gro95c], are a form of CFG augmented with inherited string-valued attributes. LCFRS, PMCFG and LMG are primarily aimed at the analysis of natural language. *String Attributed Grammars* are the concatenative variant of the attribute grammar formalism, which is widely used in programming language semantics. The properties of the class of attribute output languages $\mathrm{OUT}(\mathrm{SAG})$ are studied in [Eng86].

We present an attractive general purpose grammar formalism, *concatenative predicate grammar,* in which all the mentioned formalisms can be represented. This results in both a more readable notation, and an elegant hierarchical classification of the grammar formalisms.

1 Introduction

Literal Movement Grammars (LMG), introduced in [Gro95c], aim at the description of *discontinuous constituency* and *extraposition* in natural languages, while maintaining some of the pleasant properties of context free grammars, such as (polynomial) complexity of recognition, the applicability of Earley-based, left-to-right scanning parsing strategies, and the fact that the appearance of an LMG is similar to that of a CFG. In [Gro95b] we show, by a number of simple examples, that LMG is relatively adequate w.r.t. other well-known context sensitive formalisms (head grammar or tree adjoining grammar [VSW94], extraposition grammar [Per81]) in the treatment of languages such as Dutch, whose surface structure is essentially more involved than that of English.

This paper puts LMG in a familiar formal perspective by defining a notational variant, *concatenative predicate grammars* or CPG, of the *noncombinatorial* LMG as defined in [Gro95c]. This variant emphasizes its relationship to more well-known formalisms such as *linear context-free rewriting systems* (LCFRS, [Wei88]) and *parallel multiple context-free grammars* (PMCFG, [KNSK92]). It turns out that CPG not only elegantly subsume the mentioned formalisms, but also provide a simple characterization of the class of languages recognisable in deterministic polynomial time. The latter is the subject of a separate paper [Gro95a].

After an introduction to LCFRS and LMG, we will define CPG and show how CPG is a generalization of both formalisms. We then introduce *string attributed grammar* (SAG, [Eng86]) and show that it too fits into the framework of CPG.

2 Paradigm one. Linear Context Free Rewriting Systems and Parallel Multiple Context-Free Grammars

A *linear context-free rewriting system* (LCFRS, [Wei88]) is a context-free grammar in which each rule is augmented with a linear, non-erasing function over tuples of terminal words. Instead of deriving a single terminal string, an LCFRS derives tuples of terminal strings; an LCFRS derivation is a context free derivation where each node is annotated with a function. The class of languages recognized by LCFRS is equal to that recognised by multi component TAG (MCTAG), and is included in, but not equal to the class PTIME of languages that can be recognised in polynomial time [Wei88].

Definition 2.1 (LCFRS) A *linear context-free rewriting system* (LCFRS) is a tuple (N, T, μ, S, P) where N and T are sets of *nonterminal symbols* and *terminal symbols* respectively, $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, $\mu : N \rightarrow \mathbf{N}$ is the *similarity type* that assigns an arity to each nonterminal, $\mu(S) = 1$ and P is a set of *productions* of the form

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

where $m \geq 0$, $A, B_1, \ldots, B_m \in N$, and the *yield function* f is a *linear, non-erasing* function over tuples of terminal words, that is, $f : ((T^*)^{\mu(B_1)}, \ldots, (T^*)^{\mu(B_m)}) \rightarrow (T^*)^{\mu(A)}$ can be defined symbolically as

$$f(\langle x_1^1, \ldots, x_{\mu(B_1)}^1 \rangle, \ldots, \langle x_1^m, \ldots, x_{\mu(B_m)}^m \rangle) = \langle t_1, \ldots, t_{\mu(A)} \rangle$$

where t_k are strings over terminals and the variables x_j^i, and each of the x_j^i appears precisely once in $t_1, \ldots, t_{\mu(A)}$.

Definition 2.2 (LCFRS derivation) An LCFRS $G = (N, T, \mu, S, P)$ derives a terminal word $w \in T^*$ if $S \overset{G}{\Longrightarrow} \langle w \rangle$ where $\overset{G}{\Longrightarrow}$ is defined inductively, as follows: if

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

is a production in P, then

$$\frac{B_1 \overset{G}{\Longrightarrow} \langle w_1^1, \ldots, w_{\mu(B_1)}^1 \rangle \quad \cdots \quad B_m \overset{G}{\Longrightarrow} \langle w_1^m, \ldots, w_{\mu(B_m)}^m \rangle}{A \overset{G}{\Longrightarrow} f(\langle w_1^1, \ldots, w_{\mu(B_1)}^1 \rangle, \ldots, \langle w_1^m, \ldots, w_{\mu(B_m)}^m \rangle)} \ \mathbf{CFR}$$

Note that the base case is $m = 0$ (the rule then has 0 antecedents). The language recognized by an LCFRS is called a linear context-free rewriting language (LCFRL).

Example 2.3 The following LCFRS generates the language $\mathsf{a}^n \mathsf{b}^n \mathsf{c}^n$.

$$\begin{array}{lllrcl}
S & \rightarrow & e(A), & e(\langle x, y, z \rangle) & = & \langle xyz \rangle \\
A & \rightarrow & f(A), & f(\langle x, y, z \rangle) & = & \langle \mathsf{a}x, \mathsf{b}y, \mathsf{c}z \rangle \\
A & \rightarrow & g(), & g() & = & \langle \lambda, \lambda, \lambda \rangle
\end{array}$$

Remark 2.4 (Context Free Grammar) A CFG is an LCFRS, whose yield functions are limited to concatenation over 1-tuples; a context-free rule

$$A \rightarrow B_1 \cdots B_m$$

is represented as

$$A \rightarrow \mathit{concat}^m(B_1, \ldots, B_m), \quad \mathit{concat}^m(\langle w_1 \rangle, \ldots, \langle w_m \rangle) = \langle w_1 \cdots w_m \rangle$$

and a terminal rule

$$C \rightarrow w$$

as

$$C \rightarrow \mathit{term}^w(), \quad \mathit{term}^w() = \langle w \rangle$$

Example 2.5 (Copy Languages) By generating tuples containing multiple copies of the same string, we can recognize k-copy languages over any context free language $\mathcal{L}$; e.g. to generate the 2-copy language $\{ww \mid w \in \mathcal{L}\}$ over a context-free language $\mathcal{L}$, we replace the rules from the previous remark with

$$\begin{array}{lll}
A & \rightarrow & \mathit{concat}_2^m(B_1, \ldots, B_m), \\
 & & \quad \mathit{concat}_2^m(\langle v_1, w_1 \rangle, \ldots, \langle v_m, w_m \rangle) = \langle v_1 \cdots v_m, \ w_1 \cdots w_m \rangle \\
C & \rightarrow & \mathit{term}_2^w(), \\
 & & \quad \mathit{term}_2^w() = \langle w, w \rangle
\end{array}$$

Since the resulting start symbol has arity 2, we need to wrap things up by adding a new start symbol Σ and the following production:

$$\Sigma \;\rightarrow\; \mathit{flatten}_2(S), \quad \mathit{flatten}_2(\langle w_1, w_2\rangle) = \langle w_1 w_2\rangle$$

Remark 2.6 (Head Grammar) A bilinear *head grammar* is an LCFRS restricted to 2-tuples and three binary functions; i.e. $G = (N, T, \mu, S, P)$ where $\mu(A) = 2$ for $S \neq A \in N$, and for each production $A \rightarrow f(B_1, B_2)$, f is one of the functions wrap, concat_A or concat_B where

$$\begin{array}{lcl} \mathit{wrap}(\langle v_1, v_2\rangle,\ \langle w_1, w_2\rangle) & = & \langle v_1 w_1,\ w_2 v_2\rangle \\ \mathit{concat}_A(\langle v_1, v_2\rangle,\ \langle w_1, w_2\rangle) & = & \langle v_1,\ v_2 w_1 w_2\rangle \\ \mathit{concat}_B((\langle v_1, v_2\rangle,\ \langle w_1, w_2\rangle) & = & \langle v_1 v_2 w_1,\ w_2\rangle \end{array}$$

It is shown in [VSW94] that this formal counterpart of Pollard's head grammars [Pol84] is weakly equivalent to TAG, LIG and CCG. It is worth noting that this is still one of the weakest thinkable subclasses of linear context-free rewriting systems stronger than CFG; opinions vary about the question whether such severely restricted formalisms can be considered linguistically adequate. In [Gro95b] we argue that the TAG family is adequate for structural descriptions of English but fails for any more than minimal fragment of structurally complex languages such as Dutch and German.

Remark 2.7 (Constant Growth) It is easy to establish a very weak form of a pumping lemma, the *constant growth property*, for LCFRS [Wei88]: for every LCFRL $\mathcal{L}$ there is an integer c_0 and a set of constants C such that for each word $w \in \mathcal{L}$ whose length is greater than c_0, there is another word $w' \in \mathcal{L}$ such that for one of the constants $c \in C$, $|w'| = |w| + c$.

Larger classes of grammars are obtained by relaxing the constraints of non-erasingness and linearity.

Definition 2.8 (MCFG, PMCFG) A *multiple context-free grammar* (MCFG) is as an LCFRS, but a yield function f, defined as above:

$$f(\langle x_1^1, \ldots, x_{\mu(B_1)}^1\rangle, \ldots, \langle x_1^m, \ldots, x_{\mu(B_m)}^m\rangle) \;=\; \langle t_1, \ldots, t_{\mu(A)}\rangle$$

is only required to be *linear*: each of the x_j^i appears *at most once* in the sequence of terms $t_1, \ldots, t_{\mu(A)}$.

A *parallel multiple context-free grammar* [KNSK92] is as an LCFRS, but there are no restrictions to the yield functions.

A KNOWN result is that for every MCFG, a weakly equivalent LCFRS can be constructed, hence LCFRL = MCFL.[1] The same construction shows that a PMCFG can be made nonerasing. However, the following example of a PMCFG that describes a non-constant growth language shows that PMCFG has a strictly larger weak generative capacity than LCFRS and MCFG.

Example 2.9 The following PMCFG recognizes the language a^{2^n} (note that f is not linear and hence this is not an LCFRS):

$$\begin{array}{lclrcl} S & \rightarrow & f(S), & f(\langle x\rangle) & = & \langle xx\rangle \\ S & \rightarrow & g(), & g() & = & \langle \mathsf{a}\rangle \end{array}$$

Proposition 2.10 (homomorphisms) LCFRL and PMCFL are closed under arbitrary homomorphism.

PROOF Entirely straightforward. We replace all terminals in the grammar by their homomorphic images; every derivation in the resulting grammar corresponds to one in the original grammar, and vice versa. The yield functions are based on concatenation, and hence preserve homomorphic images.

[1] However, it is shown in [KNSK92] that the non-erasingness condition is relevant to the time complexity of the problem of universal recognition.

3 Paradigm two. Literal Movement Grammar

In the *literal movement grammar* formalism, introduced by the author in [Gro95c], a nonterminal is annotated with a number of terminal words. These terminal words correspond to the linguistic notion of *extraposed data*, that is, parts of a constituent which do not appear within the constituent itself. Hence as opposed to LCFRS, literal movement grammars strictly distinguish a part of a sentence as a *frame* or *backbone* relative to which some (smaller) constituents can be moved around.

Definition 3.1 (literal movement grammar) Let sets N, T and μ be given as for an LCFRS. Let V be a set of *variable symbols* disjoint with N and T.

- A *term* $t \in (T \cup V)^*$ is any sequence of terminals and variables.
- A *predicate* ϕ is a nonterminal A with $\mu(A)$ arguments, i.e. $A(t_1, \ldots, t_{\mu(A)})$ where t_i are terms. A predicate is *non-combinatorial* if it is of the form $A(x_1, \ldots, x_n)$. We write A instead of $A()$ for a predicate with zero arguments.
- An *item* is one of the following:

  A terminal a

  A variable x

  A predicate ϕ

  A slash item (ϕ/x) where x is a variable, and ϕ is a predicate

A *literal movement grammar* (LMG) is a tuple $G = (N, T, V, \mu, S, P)$, where $S \in N$; $\mu(S) = 0$ and P is a set of *productions* of the form

$$\phi \to \Psi_1 \Psi_2 \cdots \Psi_m$$

where ϕ is a predicate, and $\Psi_1 \cdots \Psi_m$ are items.

WE USE the symbols r, x, y, z for variables, v, w for terminal words, s, t for terms, ϕ, ψ for predicates, Ψ, Φ for items, and α, β, γ for sequences of items. Note that a terminal word is a term; a term is a sequence of items. The empty sequence is denoted by λ.

Definition 3.2 (semantics of LMG) An *instantiation* of an LMG production is obtained by substituting a terminal word for each variable occurring in the production; an LMG G recognizes a word w if $S \overset{G}{\Longrightarrow} w$ can be derived by the following inference rules:[2]

$$\phi \overset{G}{\Longrightarrow} \alpha \quad \text{when} \quad \phi \to \alpha \text{ is an instantiation of a rule in } G$$

$$\frac{\phi \overset{G}{\Longrightarrow} \beta\ \psi\ \gamma \qquad \psi \overset{G}{\Longrightarrow} v}{\phi \overset{G}{\Longrightarrow} \beta\ \ v\ \ \gamma}\ \textbf{LMR} \qquad \frac{\phi \overset{G}{\Longrightarrow} \beta\ \ (\psi/v)\ \ \gamma \qquad \psi \overset{G}{\Longrightarrow} v}{\phi \overset{G}{\Longrightarrow} \beta\ \ \gamma}\ \textbf{LMS}$$

ARBITRARY LMG can describe any r.e. language; hence we are usually interested in restricted forms of LMG. The following restricted form has been shown (see proposition 4.11) to describe precisely the class of polynomial time recognisable languages.

[2] The definition here is a significant simplification of the one given in [Gro95c]. We usually (e.g. in [Gro95c]) add an extra type of item $(x : \phi)$, called a *colon item* or *restricted quantifier*. According to the semantics given here it is equivalent to the sequence of two items $x\ \ (\phi/x)$. The elimination of colon items in the formal definition implies that LMG derivations no longer contain variables. As we will see further in this paper, slash items and the slash rule can also be elimiated without loss of generative capacity, but this defeats the purpose of the formalism (analysis of movement in natural language with a strict distinction between sentential frame, gaps and fillers) and requires a more substantial transformation (proposition 4.8) of the grammar.

Definition 3.3 (simple) A literal movement grammar $G = (N, T, \mu, S, P)$ is said to be *simple* when all predicates occurring on the RHS of productions $R \in P$ are non-combinatorial, and all productions

$$\phi \ \rightarrow \ \Psi_1 \Psi_2 \cdots \Psi_m$$

are *nonerasing*, that is when an item Ψ_i on the right hand side refers to a variable x, *viz.* $A(\ldots, x, \ldots)$, $A(\ldots, x, \ldots)/y$ or $A(\ldots)/x$, then x either appears in ϕ or $\Psi_j = x$ for some j.

E.g. the LMG rules

$$\begin{array}{lcll} A(x, y) & \rightarrow & B(xy) & (B(xy) \text{ is combinatorial}) \\ A & \rightarrow & B(x) \ \ C(x) & (x \text{ is erased}) \end{array}$$

are not simple; but the following rule is:

$$A(xy, z) \rightarrow (B(z)/y) \ \ r \ \ C(x, r)$$

Example 3.4 The following simple LMG recognizes the language $\mathsf{a}^n \mathsf{b}^n \mathsf{c}^n$; the start symbol S recognizes a string w followed by $R(w)$. $R(w)$ recognizes a string $\mathsf{b}^n \mathsf{c}^n$ if and only if $w = \mathsf{a}^n$.

$$\begin{array}{lcl} S & \rightarrow & x \ \ R(x) \\ R(\mathsf{a}y) & \rightarrow & \mathsf{b} \ \ R(y) \ \ \mathsf{c} \\ R(\lambda) & \rightarrow & \lambda \end{array}$$

Example 3.5 The following simple LMG recognizes the language a^{2^n}:

$$\begin{array}{lcl} S & \rightarrow & x \ \ x \ \ (S/x) \\ S & \rightarrow & \mathsf{a} \end{array}$$

Example 3.6 Let G_1 and G_2 be (simple) literal movement grammars, and let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the languages they recognise. Let S_1 and S_2 be the start symbols of the grammars. If we combine the grammars S_1 and S_2, add a new start symbol S_3 and the rule

$$S_3 \ \rightarrow \ x \ \ (S_1/x) \ \ (S_2/x)$$

then we obtain a new (simple) LMG G_3 which recognises precisely the language $\mathcal{L}_1 \cap \mathcal{L}_2$.

WE CONCLUDE this chapter with a grammar that generates the language of *arbitrary* numbers of copies from a context-free language $\mathcal{L}$, using the slash feature:

Example 3.7 Let $\mathcal{L}$ be a language, and $G = (N, T, S, P)$ be a context free grammar that describes it; then the simple literal movement grammar

$$G' = (N \ \cup \ \{\Sigma, \ \mathit{Copy}\}, \ T, \ \{x\}, \mu, \ \Sigma, \ P \ \cup \ \{R_1, R_2, R_3\}),$$

where $\mu(\mathit{Copy}) = 1, \ \mu(A) = 0$ otherwise, and the new productions R_1, R_2, R_3 are

$$\begin{array}{lcl} \Sigma & \rightarrow & x \ \ \mathit{Copy}(x) \\ \mathit{Copy}(x) & \rightarrow & x \ \ \mathit{Copy}(x) \\ \mathit{Copy}(x) & \rightarrow & (S/x) \end{array}$$

generates the language $\{w^n \mid w \in \mathcal{L}, \ n > 0\}$.

4 The framework: Concatenative Predicate Grammar

The previous sections already suggest a similarity between grammars in LCFRS, PMCFG and LMG, as some of the presented examples and terminology coincide. We now formalize their relationship by developing a definite clause style notation in which both LCFRS and LMG can be directly embedded.

Definition 4.1 (CPG) A *concatenative predicate grammar* (CPG) is a tuple $G = (N, T, V, \mu, S, P)$ where N, T, V, μ are as for an LMG, and $\mu(S) = 1$.

- *terms* t and *predicates* ϕ are as for LMG; We write Γ, Δ for sequences of predicates.
- A production $R \in P$ is of the form

  $$\phi \texttt{ :- } \psi_1 \psi_2 \cdots \psi_m$$

  where $\phi, \psi_1, \ldots, \psi_m$ are arbitrary predicates. When $m = 0$, we abbreviate the rule ($\phi \texttt{ :- } \lambda$) to

  $$\phi.$$

- An *instantiation* of a production $R \in P$ is obtained by substituting a terminal word for each of the variables occurring in the production.
- G recognizes a string w if $\vdash^G S(w)$ where $\vdash^G$ is defined inductively as follows: if

  $$A(w_1, \ldots, w_{\mu(A)}) \texttt{ :- } B_1(v_1^1, \ldots, v_{\mu(B_1)}^1) \cdots B_m(v_1^m, \ldots, v_{\mu(B_m)}^m)$$

  is an instantiation of a rule in P, then

  $$\frac{\vdash^G B_1(v_1^1, \ldots, v_{\mu(B_1)}^1) \quad \cdots \quad \vdash^G B_m(v_1^m, \ldots, v_{\mu(B_m)}^m)}{\vdash^G A(w_1, \ldots, w_{\mu(A)})} \textbf{ CPR}$$

  Note that as for LCFRS, $m = 0$ is the base case (0 antecedents).

Remark 4.2 (CPG to LMG) Every CPG can be viewed as an LMG without slash items, by identifying each CPG rule

$$\phi \texttt{ :- } \psi_1 \psi_2 \cdots \psi_m$$

with the "identical" LMG rule[3]

$$\phi \rightarrow \psi_1 \psi_2 \cdots \psi_m.$$

A sequent $\vdash^{G'} \phi$ in a CPG derivation corresponds to $\phi \overset{G}{\Longrightarrow} \lambda$ in the LMG derivation; an application of the **CPR** rule corresponds to applying m **LMR** rules in succession. Given a CPG $G = (N, T, V, \mu, S, P)$, we have a weakly equivalent LMG $G' = (N \cup \{\Sigma\}, T, V, \mu', \Sigma, P \cup R)$ where Σ is a new start symbol, and we add one new production R:

$$\Sigma \rightarrow x \quad S(x)$$

(μ' is the similarity type that agrees with μ and assigns arity 0 to the new start symbol Σ.) We now have $\vdash^{G'} S(w)$ iff $S(w) \overset{G}{\Longrightarrow} \lambda$ iff $\Sigma \overset{G}{\Longrightarrow} w$.

AS IN the case of LMG, we can construct a CPG for any recursively enumerable language. Hence the following set of restrictions.

Definition 4.3 (properties of CPG) Let $G = (N, T, V, \mu, S, P)$ be a CPG, and let $R \in P$ be one of its productions:

$$A(t_1, \ldots, t_{\mu(A)}) \texttt{ :- } B_1(s_1^1, \ldots, s_{\mu(B_1)}^1) \cdots B_m(s_1^m, \ldots, s_{\mu(B_m)}^m)$$

then

- R is *bottom-up linear*[4] if no variable x appears more than once in $t_1, \ldots, t_{\mu(A)}$.

[3] CPG can be thought of as replacing the LMG notion of *production* by the (reversed) notion of *implication*—read $\phi \texttt{ :- } \Gamma$ as $\Gamma \Rightarrow \phi$. By allowing only *predicates* on its RHS, the order of the items in a CPG production is no longer relevant, as was already the case for the *slash items* in an LMG production. Therefore the reader may feel inclined to succumb to the suggestive notation and think of Γ as a *set* of predicates rather than a sequence.

[4] The terms *left-linear* and *right-linear* are also in use here, but give rise to confusion as they are turned around w.r.t. the terminology for LCFRS yield functions; moreover these terms are also used in an entirely different sense for brands of context free grammars and languages. The choice for the top-down/bottom-up jargon is motivated by the fact that is extends straightforwardly to properties of derivation trees.

- R is *top-down linear* if no variable x appears more than once in $s_1^1, \ldots, s_{\mu(B_m)}^m$.
- R is *bottom-up nonerasing* if each variable x occurring in an s_k^j 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_k^j.
- R is *non-combinatorial* if each of the s_k^j consists of a single variable.
- R is *simple* if it is bottom-up nonerasing and non-combinatorial.

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

CLEARLY, when a CPG is simple, the corresponding LMG according to remark 4.2 is also simple. We now note a few elementary relations between the other properties, and then relate CPG to the formalisms we discussed in the previous sections.

Remark 4.4 (Bottom-up nonlinearity simulated by top-down nonlinearity)
A bottom-up nonlinear rule

$$A(\cdots x \cdots x \cdots) \text{ :- } \Gamma$$

can be replaced by the bottom-up linear, possibly top-down nonlinear rule (top-down nonlinear for x may occur in Γ and hence more than once on the RHS)

$$A(\cdots x \cdots y \cdots) \text{ :- } \Gamma \;\; Eq^T(x, y)$$

where we add the following $|T| + 1$ productions for *Eq*, so that the grammar derives precisely $Eq(w, w)$ for all terminal words w:

$$\begin{array}{lll} Eq^T(\lambda, \lambda). & & \\ Eq^T(\mathsf{a}x, \mathsf{a}y) & \text{:-} & Eq^T(x, y) \quad \text{for each } \mathsf{a} \in T \end{array}$$

When a class of grammars allows top-down nonlinearity, it is hence also capable of representing bottom-up nonlinear constructions; therefore when we are talking about *bottom-up* linear subclasses of CPG we will take this to imply that the grammars are also top-down linear, and we will simply call those grammars (fully) *linear*.

Remark 4.5 (Erasingness) We can replace a top-down erasing production

$$A(\cdots x \cdots) \text{ :- } \Gamma$$

where x does not occur in Γ, by the top-down nonerasing rule

$$A(\cdots x \cdots) \text{ :- } \Gamma \;\; Any(x)$$

where we have the following $|T| + 1$ top-down nonerasing productions for Any^T, so that the grammar derives $Any^T(w)$ for any $w \in T^*$:

$$\begin{array}{lll} Any^T(\lambda). & & \\ Any^T(\mathsf{a}x) & \text{:-} & Any^T(x) \quad \text{for each } \mathsf{a} \in T \end{array}$$

Hence top-down erasingness is irrelevant to the generative capacity of a class of grammars.

Proposition 4.6 (CPG and LCFRS) A simple, fully linear and nonerasing CPG is merely a (more compact!) notational variant of an LCFRS.

EXAMPLE The CPG corresponding to example 2.3 is

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

Proposition 4.7 (CPG and PMCFG) A non-combinatorial, top-down linear, top-down nonerasing CPG is a different notation for a PMCFG.

EXAMPLE The PMCFG for a^{2^n} from example 2.9 looks as follows in CPG notation:

$$\begin{array}{lll} S(xx) & \texttt{:-} & S(x) \\ S(\mathsf{a}). & & \end{array}$$

WE HAVE already seen that a CPG can be seen as an LMG. To prove that there is a language-preserving translation from (simple) LMG to (simple) CPG, we need to do some work.

Proposition 4.8 (CPG and LMG) For any (simple) LMG there is a weakly equivalent (simple) CPG.

PROOF The idea is to move the yield of a predicate into an extra argument position; this is analoguous to the translation of a DCG in phrase structure notation to a pure logic program. If $G = (N, T, V, \mu, S, P)$ is a (simple) LMG, then we construct a (simple) CPG $G' = (N, T, V', \mu', S, P')$ as follows: put $\mu'(A) = \mu(A) + 1$, and let $V' = V \cup \{z_1, \ldots, z_M\}$ where M is the largest number of items on the right hand side of productions in P.

Take an LMG production $A(t_1, \ldots, t_n) \to \Psi_1\Psi_2 \cdots \Psi_m$ in P. We construct a CPG rule as follows:

$$A(s_1 \cdots s_m, t_1, \ldots, t_n) \;\texttt{:-}\; \Gamma_1\Gamma_2 \cdots \Gamma_m$$

where for each item Ψ_i in the LMG rule, Γ_i is either one predicate or empty.

$$\begin{array}{lll} \textbf{if } \Psi_i = \mathsf{a} & \textbf{then } \Gamma_i = \lambda & \textbf{and } s_i = \mathsf{a} \\ \textbf{if } \Psi_i = x & \textbf{then } \Gamma_i = \lambda & \textbf{and } s_i = x \\ \textbf{if } \Psi_i = B(x_1, \ldots, x_k) & \textbf{then } \Gamma_i = B(z_i, x_1, \ldots, x_k) & \textbf{and } s_i = z_i \\ \textbf{if } \Psi_i = B(x_1, \ldots, x_k)/y & \textbf{then } \Gamma_i = B(y, x_1, \ldots, x_k) & \textbf{and } s_i = \lambda \end{array}$$

Clearly (cases 3, 4) if the LMG rule is simple, then so will its translation.

By a straightforward inductive argument, we now see that for any terminal words $v_1, \ldots, v_n, w$ and any nonterminal A, we have

$$A(v_1, \ldots, v_n) \overset{G}{\Longrightarrow} w \quad \text{iff} \quad \vdash^G A(w, v_1, \ldots, v_n).$$

A CONSEQUENCE of proposition 4.8 is that the slash items in the definition of LMG do not contribute to the formal power of the formalism.

Example 4.9 Let us illustrate this construction by translating the grammar for $\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n$ from example 3.4:

$$\begin{array}{lll} S & \to & x\ R(x) \\ R(\mathsf{a}y) & \to & \mathsf{b}\ R(y)\ \mathsf{c} \\ R(\lambda) & \to & \lambda \end{array}$$

The corresponding CPG is obtained by encoding the yield of the rules in an extra argument, as outlined in the construction for proposition 4.8:

$$\begin{array}{lll} S(xz_2) & \texttt{:-} & R(z_2, x) \\ R(\mathsf{b}z_2\mathsf{c}, \mathsf{a}y) & \texttt{:-} & R(z_2, y) \\ R(\lambda, \lambda). & & \end{array}$$

Note that the resulting grammar is simple, fully linear and non-erasing and hence satisfies the requirements for an LCFRS. It is a matter of taste whether one is to prefer the more compact LMG notation or the more formally elegant CPG. There is no doubt that both of these are more attractive than the corresponding LCFRS (the reader may wish to translate the grammar into an LCFRS to see this). The difference between LMG and CPG is very clear in this example: in LMG, part of the sentence is identified as part of the surface form ($\mathsf{b}^n\mathsf{c}^n$ in this case), and the rest of the sentence is thought of as subject to movement (the a's); in CPG (and PMCFG/LCFRS) there is no such distinction. As we will show in the following section, there are some cases where the more informal LMG notation is clearly preferable.

Example 4.10 The translation of the LMG for a^{2^n} from example 3.5 into CPG is exactly the same as the CPG equivalent of the PMCFG (example 4.7).

Proposition 4.11 [Gro95a] The class of languages recognised by simple CPG is precisely the class PTIME of languages recognisable in deterministic polynomial time.

SKETCH OF PROOFS The central observation is that in a simple CPG derivation of a string w, we are only dealing with substrings of w, and hence a recognition algorithm can encode the arguments of predicates with integers ranging from 0 to n, where n is the length of the input string w.

In [Gro95a] we show that every simple CPG can be translated immediately into an equivalent ILFP formula [Rou88], and ILFP is known to describe precisely PTIME. As to the converse, we can, again analoguous to a proof in [Rou88], simulate an arbitrary logspace-bounded alternating Turing machine (ATM) in terms of the recognition problem for simple CPG. It is a well-known result [CKS81] that alternating log space is equivalent to deterministic polynomial time.

5 Paradigm three. Attribute Grammar

This section serves to stress the idea that CPG is not only formally meaningful, but also often provides a more practical notation, as we have already seen for the case of LCFRS. We show how attribute grammars based on strings and concatenation (SAG, [Eng86]), can be written down within CPG, resulting in what we think is a more readable grammar. Unfortunately, the translation does not shed light on the formal properties of the formalism; although the output set $\mathrm{OUT}(\mathrm{SAG})$ has been shown to be strictly inside PTIME, the resulting CPG grammars are not simple.[5]

A *string based attribute grammar* (SAG) is a context free grammar (N, T, S, P) with the following additions: with each nonterminal $A \in N$ we associate a two finite sets, $\mathrm{INH}(A)$ of *inherited attribute symbols* and $\mathrm{SYN}(A)$ of *synthesized attribute symbols*. For the start symbol $S \in N$ we demand that $\mathrm{INH}(S) = \emptyset$ and $\mathrm{SYN}(A) = d$, where d is some unique *designated attribute symbol*. Now with each production $R \in P$:

$$A \rightarrow w_0 B_1 w_1 B_2 \cdots w_{m-1} B_m w_m$$

we associate a set of rules which define

- the value of the synthesized attributes of A, in terms of the values of its inherited attributes and the synthesized attributes of $B_1, \ldots, B_m$, through arbitrary concatenation (i.e. functions identical to the yield functions f as in a PMCFG, definition 2.8).
- the values of the inherited attributes of $B_1, \ldots, B_m$ in terms of the inherited attributes of A and the synthesized attributes of $B_1, \ldots, B_m$ through arbitrary concatenation.

We usually require the dependencies between the attribute values to be acyclic.

An attribute grammar as defined here accepts precisely the same sentences as its context-free backbone, but assigns a *value* to each sentence, i.e., the value of the designated attribute d after the evaluation of the attributes. The set of these "output" values is called $\mathrm{OUT}(\mathrm{SAG})$, and is studied in [Eng86].

Example 5.1 [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:

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

The context-free backbone recognizes arbitrary nonempty strings w over the alphabet $T = \{\mathsf{a}, \mathsf{b}\}$, and the output value for $w \in T^*$ is $(w\mathsf{c})^{2^{|w|}}$. Hence $\mathrm{OUT}(\mathrm{SAG})$ is the language $\{(w\mathsf{c})^n \mid w \in \{\mathsf{a}, \mathsf{b}\}^*,\ w \neq \lambda,\ n = 2^{|w|}\}$.

[5] Of course, as $\mathbf{PTIME} = \mathbf{simpleCPL}$, the existence of equivalent simple CPG is a fact, but we don't know what such a CPG will look like.

Since the sets $\mathtt{INH}(A)$ and $\mathtt{SYN}(A)$ are finite, without loss of generality we can take the attributes to be integer numbers (i.e., we don't need to give them *names*, an order is sufficient). It now follows easily that we can translate an attribute grammar into a CPG or an LMG, as we now illustrate:

Example 5.2 The SAG from example 5.1 can be represented in LMG format as follows:[6]

$$\begin{array}{lcl}
Z(x) & \rightarrow & \mathsf{a}\ A(\mathsf{a};\ x) \\
Z(x) & \rightarrow & \mathsf{b}\ A(\mathsf{b};\ x) \\
A(y;\ xx) & \rightarrow & \mathsf{a}\ A(y\mathsf{a};\ x) \\
A(y;\ xx) & \rightarrow & \mathsf{b}\ A(y\mathsf{b};\ x) \\
A(x;\ x\mathsf{c}x\mathsf{c}) & \rightarrow & \lambda
\end{array}$$

Now this is not a real LMG, as it does not have a start symbol. By adding the following rule:

$$S \rightarrow \texttt{"("}\ Z(x)\ \texttt{","}\ x\ \texttt{")"}$$

we generate the language consisting of all tuples of words in T^* and their values according to the SAG.

We can construct a corresponding CPG by adding an extra synthesized attribute that takes over the yield of the context free backbone (in other words, every SAG is output equivalent to a SAG that recognizes only the empty string).

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

Note that although this grammar accepts precisely $\mathtt{OUT}(\mathtt{SAG})$, it is not bottom-up nonerasing (y on the RHS of the S rule is erased), so it is not simple, and it seems of little formal value. However, as in the case of LCFRS, the CPG/LMG representation is (we think) a great improvement in readability. In this case, the LMG notation seems preferable, as we can still easily recognize the context free backbone of the attribute grammar, which has been "abstracted away" in the CPG.

6 Classification

We conclude this paper by summarizing how the formalisms discussed in sections 2 and 3 fit into a hierarchy of CPG grammars satisfying increasing numbers of restricting properties from definition 4.3. All formalisms are equivalent to some class of *simple* CPG (i.e. they are in PTIME). LCFRS are left and right linear, non-erasing; they satisfy the constant-growth property. PMCFG are only right linear, and do not satisfy the constant growth property (example 2.9). PMCFG are closed under arbitrary homomorphism, hence they cannot be closed under intersection (otherwise they would describe any r.e. language, which is in contradiction with the fact that fixed PMCFG recognition is polynomial-time). Simple CPG subsume PMCFG, and are closed under intersection, hence they are strictly stronger than PMCFG. Simple CPG is equivalent to simple LMG, and the class of languages generated is precisely the class PTIME of languages recognisable in polynomial time.

Formalism	*Increasing conditions on CPG form*	*Weakly equivalent to*
Generic LMG	—	CPG, all r.e. languages
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	—

[6] We use a semicolon as opposed to a comma, to separate the inherited attributes (left) and the synthesized attributes (right). This is only to improve readability; formally it should be read as a comma.

Although string-valued attribute grammars can be represented in CPG form, and we think this way of writing down such grammars is more clear than traditional ways of writing down AGs, this currently serves just as an example and has no formal consequences (moreover, the formalism does not seem to fit in this hierarchy).

Conclusions

We have outlined how literal movement grammars, and the more formal notation in the form of concatenative predicate grammars, can be viewed as a straightforward extension of the more well-known linear context-free rewriting systems. The *simple* CPG describe precisely the class PTIME. The definite clause style notation seems preferable to the notation currently found in discussions of LCFRS and Head Grammars—because it is more readable, and it sheds light on the way in which LCFRS and PMCFG relate to PTIME. This paper did not discuss formalisms between r.e. and PTIME; however, boundedness conditions along the lines of those for CLFP [Rou88] can be stated for CPG, so as to obtain the classes EXPTIME and EXP-POLYTIME.

Further research

As we have reduced the differences between a number of grammatical formalisms to parameters on the appearance of one general formalism, a number of small problems such as the question whether LCFRS is precisely the class of mildly context-sensitive languages (e.g.: is there any PMCFL that is constant growth but not an LCFRL) should in the pleasant substitutional semantics of CPG be easier to address. Since simple CPG represents precisely the tractable languages, it is worthwhile to investigate

1. the descriptive qualities of the formalism in a natural language context (typical examples of trans-mildly context-sensitive fragments are e.g. the Chinese number names from [Rad91]).
2. the possibility to add bounded information such as finite lattice-valued features so as to obtain a comfortable linguistical formalism without sacrificing tractability; it seems reasonable to believe that all sources of "unboundedness" in current feature-based formalisms are tightly connected to surface-structural unbounded dependencies, and hence a feature extension of CPG may not require unbounded constructs to be fully adequate.
3. practical implementations for simple CPG recognition and, after a suitable definition, of CPG parsing. These do not seem to be straightforward tasks, since the current proof is in terms of alternating Turing machines which bear hardly any resemblance to everyday (deterministic) computing practice. Some variant of generalized Earley recognition is under investigation.
4. the existence of tight polynomial bounds for subclasses of simple CPG in which the number of variables used in productions is bounded. Material along this lines is found in [Rou88] and [KNSK92].

Acknowledgements

I would like to thank Jan van Eijck, Bill Rounds, David Weir, Erik Aarts and an anonymous referee for discussions and comments on the LMG formalism and earlier versions of this paper. The author is supported by SION grant 612-317-420 of the Netherlands Organization for Scientific Research (NWO).

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. Paper presented at the fourth Mathematics of Language workshop (MOL4), Univ. of Pennsylvania, 1995.

[Gro95b] Annius V. Groenink. Formal Mechanisms for Left Extraposition in Dutch. Paper presented at the 5th CLIN (Computational Linguistics In the Netherlands) meeting, November 1994. Submitted for proceedings, 1995.

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

[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.

[Per81] Fernando Pereira. Extraposition Grammars. *Computational Linguistics*, 7(4):243–256, 1981.

[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.

[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.