A Cubic Time Extension of Context-Free Grammars

Pierre Boullier

N° 3611

Janvier 1999

THÈME 3

ISSN 0249-6399

A Cubic Time Extension of Context-Free Grammars

Pierre Boullier*

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet Atoll

Rapport de recherche n˚ 3611 — Janvier 1999 — 28 pages

Abstract: Context-free grammars and cubic time parsing are so related in people's minds that they often think that parsing any extension of context-free grammars must need some extra time. Of course, this is not true and this paper presents a generalization of context-free grammars which keeps a cubic time complexity. This extension, which defines a sub-class of context-sensitive languages, has both a theoretical and a practical interest. The class of languages defined by these grammars is closed under both intersection and complementation (in fact it is the class containing the intersection and the complementation of context-free languages). On the other hand, these grammars can be considered as being mildly context-sensitive and can therefore be used in natural language processing.

Key-words: context-free grammars, mild context-sensitive grammars, parsing time complexity, closure properties, grammar modularity.

(Résumé : tsvp)

* E-mail: Pierre.Boullier@inria.fr

Une extension des grammaires non-contextuelles analysable en temps cubique

Résumé : Les grammaires non-contextuelles et l'analyse en temps cubique sont si étroitement associées dans l'esprit des gens qu'ils pensent souvent que toute extension nécessiterait obligatoirement un temps d'analyse supérieur. Bien sûr, c'est une erreur, et ce papier présente une généralisation des grammaires non-contextuelles qui conserve un temps d'analyse cubique. Cette extension, qui définit une sous-classe des langages contextuels, peut présenter un intérêt à la fois théorique et pratique. La classe de langages définie par ces grammaires est close par intersection et complémentation (en fait c'est la classe à laquelle appartiennent les intersections et le complémentaire des langages non-contextuels). D'un autre côté, ces grammaires peuvent être considérées comme étant un formalisme modérément contextuel et, en conséquence, être utilisées en traitement de la langue naturelle.

Mots-clé : grammaires non-contextuelles, grammaires modérément contextuelles, complexité en temps d'analyse, propriétés de fermeture, modularité grammaticale.

1 Motivation

It is a well known result that the intersection of two context-free languages (CFLs) is not a CFL and that the complement of a CFL is not a CFL either [Hopcroft and Ullman 79], but both results are contained into the context-sensitive class. The recognition problem for context-sensitive languages (CSLs) is PSPACE-complete, this means that no polynomial-time solution is known. However, very simple algorithms can be designed to solve both the intersection and complementation problems for CFLs. Let G_1 and G_2 be two context-free grammars (CFGs) defining respectively the languages L_1 and L_2, and let M be a cubic time CF recognizer such as the Earley's algorithm for example [Earley 70]. Any input string w is in $L_1 \cap L_2$ iff it belongs simultaneously to (G_1, M) and to (G_2, M). Obviously such a check is performed in cubic time. In the same vein w is in $\overline{L_1}$, iff it does not belong to (G_1, M), this second check being also performed in cubic time. Therefore, the recognition problem for $L_1 \cap L_2$ and $\overline{L_1}$ can be performed in cubic time, this indicates that these languages may perhaps belong to some sub-class of CSLs.

On the other hand, in [Boullier 98a], we presented range concatenation grammars (RCGs), a syntactic formalism which is a variant of literal movement grammars (LMGs), described in [Groenink 97], and which is also related to the framework of LFP developed by [Rounds 88]. This formalism extends CFGs and is even more powerful than linear context-free rewriting systems (LCFRS) [Vijay-Shanker, Weir, and Joshi 87], while staying computationally tractable: its sentences can be parsed in polynomial time. The rewriting rules of RCGs, called *clauses*, apply to composite objects named *predicates* which can be seen as nonterminal symbols with arguments.

The purpose of this paper is to study the sub-class of RCGs with a single argument, the 1-RCGs. We shall see that the intersection of two CFLs is a range concatenation language of arity one (1-RCL), that the complement of a CFL is a 1-RCL and that 1-RCGs can arguably be considered as a mildly context-sensitive formalism [Joshi, Vijay-Shanker, and Weir 91].

Section 2 introduces the notion of RCG while Section 3 presents the 1-RCG subclass and its relationships with CFG. Section 4 studies whether 1-RCG is a mildly context-sensitive formalism and Section 5 gives some concluding remarks together with some experiment results.

2 Range Concatenation Grammars

The purpose of this section is to introduce the notion of RCG in a rather informal way and to present some of its properties. Some new results are presented here and some others already appear in more details in [Boullier 98a].

Definition 1 *A positive range concatenation grammar (PRCG) $G = (N, T, V, P, S)$ is a 5-tuple where N is a finite set of* predicate names, *T and V are finite, disjoint sets of* terminal symbols *and* variable symbols *respectively, $S \in N$ is the start predicate name, and P is a finite set of* clauses

$$\psi_0 \to \psi_1 \dots \psi_m$$

where $m \geq 0$ and each of $\psi_0, \psi_1, \dots, \psi_m$ is a predicate of the form

$$A(\alpha_1, \dots, \alpha_p)$$

where $p \geq 1$ is its arity, $A \in N$ and each of $\alpha_i \in (T \cup V)^*$, $1 \leq i \leq p$, is an argument.

Each occurrence of a predicate in the RHS of a clause is a predicate *call*, it is a predicate *definition* if it occurs in its LHS. Clauses which define predicate A are called A-clauses. This definition assigns a fixed arity to each predicate name. The arity of the start predicate name is one. The *arity* k of a grammar (we have a k-PRCG), is the maximum arity of its predicates.

Early occurring lower case letters such as $a, b, c, \dots$ will denote terminal symbols while late occurring upper case letters such as X, Y, Z will denote elements of V.

The language defined by a PRCG is based on the notion of *range*. For a given input string $w = a_1 \dots a_n$ a range is a couple (i, j), $0 \leq i \leq j \leq n$ of integers which denotes the occurrence of some substring $a_{i+1} \dots a_j$ in w. The number $j - i$ is its *size*. We will use several equivalent denotations for ranges: an explicit dotted notation like $w_1 \bullet w_2 \bullet w_3$ or, if w_2 extends from positions $i+1$ through j, by $\langle i..j \rangle_w$ or $\langle i..j \rangle$ when w is understood or of no importance. For a range $\langle i..j \rangle$, i is its *lower bound* and j is its *upper bound*. If $i = j$, we have an *empty* range. Of course, only consecutive ranges can be concatenated into new ranges. In any PRCG, terminals, variables and arguments in a clause are supposed to be bound to ranges by a substitution mechanism. An *instantiated clause* is a clause in which variables and arguments are consistently (w.r.t. the concatenation operation) replaced by ranges; its components are *instantiated predicates*.

For example, $A(\langle g..h \rangle, \langle i..j \rangle, \langle k..l \rangle) \to B(\langle g{+}1..h \rangle, \langle i{+}1..j{-}1 \rangle, \langle k..l{-}1 \rangle)$ is an instantiation of the clause $A(aX, bYc, Zd) \to B(X, Y, Z)$ if the source text $a_1 \dots a_n$ is such that $a_{g+1} = a, a_{i+1} = b, a_j = c$ and $a_l = d$. In this case, the variables X, Y and Z are bound to $\langle g{+}1..h \rangle$, $\langle i{+}1..j{-}1 \rangle$ and $\langle k..l{-}1 \rangle$ respectively.[1]

A *derive* relation, denoted by $\underset{G,w}{\Rightarrow}$, is defined on strings of instantiated predicates. If an instantiated predicate is the LHS of some instantiated clause, it can be replaced by the RHS of that instantiated clause.

Definition 2 *The language of a PRCG $G = (N, T, V, P, S)$ is the set*

$$\mathcal{L}(G) \quad = \quad \{w \mid S(\bullet w \bullet) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon\}$$

An input string $w = a_1 \dots a_n$ is a sentence iff the empty string (of instantiated predicates) can be derived from $S(\langle 0..n \rangle)$.

[1]Often, instead of saying *the range which is bound to X or denoted by X*, we will say, the range X, or even instead of *the string whose occurrence is denoted by the range which is bound to X*, we will say the string X.

The arguments of a given predicate may denote discontinuous or even overlapping ranges. Fundamentally, a predicate name A defines a notion (property, structure, dependency, ...) between its arguments whose ranges can be arbitrarily scattered over the source text. What is "between" its arguments is *not* the responsibility of A, and is described (if at all) somewhere else. PRCGs are therefore well suited to describe long distance dependencies. Overlapping ranges arise as a consequence of the non-linearity of the formalism. For example, the same variable (denoting the same range) may occur in different arguments in the RHS of some clause, expressing different views (properties) of the same portion of the source text.

Note that the order of RHS predicates in a clause is of no importance.

As an example of a PRCG, the following set of clauses describes the three-copy language $\{www \mid w \in \{a,b\}^*\}$ which is not a CFL and even lies beyond the formal power of tree adjoining grammars (TAGs) [Vijay-Shanker 87].

$$\begin{array}{lcl} S(XYZ) & \rightarrow & A(X,Y,Z) \\ A(aX,aY,aZ) & \rightarrow & A(X,Y,Z) \\ A(bX,bY,bZ) & \rightarrow & A(X,Y,Z) \\ A(\varepsilon,\varepsilon,\varepsilon) & \rightarrow & \varepsilon \end{array}$$

Definition 3 *A negative range concatenation grammar (NRCG) $G = (N,T,V,P,S)$ is a 5-tuple, like a PRCG, except that some predicates occurring in RHS, have the form $\overline{A(\alpha_1,\ldots,\alpha_p)}$, $A \in N$.*

A predicate call of the form $\overline{A(\alpha_1,\ldots,\alpha_p)}$ is said to be a *negative predicate (call).* The intuitive meaning is that a negative predicate succeeds iff its positive counterpart (always) fails. Therefore this definition is based on a "negation by failure" rule. However, in order to avoid inconsistencies occurring when an instantiated predicate is defined in terms of its negative counterpart, we prohibit derivations exhibiting this possibility. Thus we only allow so called *consistent* derivations. We say that a grammar is consistent if all of its derivations are consistent.

Definition 4 *A range concatenation grammar (RCG) is a PRCG or a NRCG.*

The PRCG (resp. NRCG) term will be used to underline the absence (resp. presence) of negative predicates.

2.1 Parse Time Complexity

In [Boullier 98a], we presented a parsing algorithm which, for an RCG G and an input string of length n, produces a parse forest in time polynomial with n and linear with $|G|$. The degree of this polynomial is at most the number of free bounds in any clause.

Intuitively, if we consider the instantiation of a clause, all its terminal symbols, variable, arguments are bound to ranges. This means that each position in its arguments is mapped onto a *source index*, a position in the source text. However, the knowledge of all couples (argument position, source index) is usually not necessary to fully defined the mapping. For example, if XaY is some argument, if $X \bullet aY$ denotes a position in this argument, and if $(X \bullet aY, i)$ is an element of the mapping, we know that $(Xa \bullet Y, i+1)$ must be another element. Argument positions $p_1, \dots, p_j$ for which there is a mapping m s.t. the source index $m(p_i)$ of some p_i cannot be deduced from the images of the other argument positions, are called *free bounds*. Of course, number of free bounds, means *minimum* number of argument positions needed to deduce all the other positions in the clause.

2.1.1 Predefined Predicates

The current implementation has a certain number of predefined predicate names among which we quote here *len*, *eqlen* and *eq*:

$len(l, X)$: checks that the size of the range denoted by the variable X is the integer l.

$eqlen(X, Y)$: checks that the size of X and Y are equal.

$eq(X, Y)$: checks that the substrings selected by the ranges X and Y are equal.

It must be noted that these predefined predicates do not increase the formal power of RCGs insofar as each of them can be defined by a pure RCG. For example, the predicate $len(1, X)$ is equivalent to $len_1(X)$, and the predicate len_1 can be defined by clause schema such as $len_1(t) \to \varepsilon$ over all terminals $t \in T$. Their introduction is justified by the fact that they are more efficiently implemented than their RCG defined counterpart and, more significantly that they convey static information which can be used to decrease the number of free bounds and thus lead to an improved parse time.

For example, the predicate call $len(3, X)$, indicates that the size of X is 3, and this means that the positions in the arguments before and after any occurrence of the variable X are not both free. In an analogous manner, $eq(X, Y)$ $(eq \Longrightarrow eqlen)$, implies that the four lower and upper bounds of X and Y are not all free.

2.1.2 Non Polynomial Parse Time

The question whether there exists some non polynomial parse time for RCGs seems a little strange since the degree of the polynomial is directly related to the number of free bounds. However, recall that this number of free bounds is a *maximum* polynomial parse time and that the maximum actual parse time can be lesser.

Consider the language $\{x \mid |x| = p^2, p \geq 0\}$ defined on some vocabulary T whose word lengths are perfect squares. This language can be defined by the following RCG

$$\begin{array}{lll}
S(XY) & \to & p^2(X, X, XY) \\
p^2(X, \varepsilon, \varepsilon) & \to & \varepsilon \\
p^2(X, T_1Y_1, Z_1Z_2) & \to & len(1, T_1)\ eqlen(X, Z_1)\ p^2(X, Y_1, Z_2)
\end{array}$$

The idea behind this grammar is the following, the first clause cuts the input text x into a prefix denoted by X and a suffix denoted by Y and $p^2(X, Y, Z)$ checks that $|Z| = |X||Y|$

in erasing at each call (see the third clause) one symbol in its second argument (the T_1) and $|X|$ symbols in its third argument (the Z_1).

The predefined predicate $len(1, T_1)$ checks that the length of T_1 is one (a single terminal symbol) and the predefined predicate $eqlen(X, Z_1)$ checks that $|X| = |Z_1|$. Note that this grammar is *not* mildly context-sensitive (see [Joshi, Vijay-Shanker, and Weir 91]) since its language does not verify the constant growth property, and is thus beyond the formal power of LCFRS. Because of the predicates *len* and *eqlen*, the maximum number of free bounds is six (because of the third clause), and thus we get a maximum parse time of $\mathcal{O}(n^6)$. Now, we will examine more precisely what happens during a parse.

For any given call $p^2(X, Y, Z)$, the predicate p^2 delivers its answer (true or false) in $1 + \min(|Y|, \frac{|Z|}{|X|})$ steps. Therefore, the total number of steps τ performed by the predicate p^2 is $\sum_{i=0}^{n} 1 + \min(i, \frac{n}{i})$. Since $\min(i, \frac{n}{i})$ is equal to i if $i \leq \sqrt{n}$ and is equal to $\frac{n}{i}$ if $i \geq \sqrt{n}$, we get $\tau \leq n + 1 + \sum_{i=0}^{\sqrt{n}} i + \sum_{i=\sqrt{n}}^{n} \frac{n}{i}$ whose asymptotic behavior is $\mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n \log n) = \mathcal{O}(n \log n)$. Therefore this grammar has an $\mathcal{O}(n \log n)$ parse time.

Note that the replacement of $eqlen(X, Z_1)$ in the last clause by $eq(X, Z_1)$ which checks that the substrings denoted by X and Z_1 are equal, gives a RCG which defines the language $\{w^{|w|} \mid w \in T^*\}$ which is a non trivial generalization of the k-copy language.

2.1.3 Sub-linear Parsing

The existence of sub-linear parse times seems strange since at least, each symbol in the source has to be read. However, it does make sense if we take the point of view that, before parsing, the source text w has been processed by a scanner whose processing time is not taken into account within the parse time. Of course, we assume that one of the purposes of this scanning phase is to check that the input word w is in T^*.

Under this assumption, for example, the language T^* can be defined by a single clause

$$S(X) \quad \rightarrow \quad \varepsilon$$

whose parse time is $\mathcal{O}(1)$.

More convincingly, consider the language $\{a^{2^l} \mid l \geq 0\}$. This language does not have the constant growth property and thus is *not* mildly context-sensitive.

However, assuming that the source text is in a^*, it can be described by the RCG

$$\begin{array}{lcl} S(XY) & \rightarrow & eqlen(X, Y)\ S(X) \\ S(a) & \rightarrow & \varepsilon \end{array}$$

By the predefined predicate *eqlen*, we know that the sizes of X and Y in the first clause are equal, each input range is cut in two equal parts, and, at each call, only the prefix part is processed. Therefore, for each input string a^n s.t. $2^{l-1} < n \leq 2^l$, S gives its answer in at most l steps. And then we get an $\mathcal{O}(\log n)$ parse time.

We can also reconsider the language of perfect squares already introduced in Paragraph 2.1.2. If we use the fact that the p^{th} perfect square is the sum of the first p odd integers, the RCG with clauses

$$\begin{array}{lcl} S(\varepsilon) & \rightarrow & \varepsilon \\ S(TX) & \rightarrow & len(1,T)\ p^2(T,X) \\ p^2(X,YTZ) & \rightarrow & len(2,T)\ eqlen(X,Y)\ p^2(YT,Z) \\ p^2(X,\varepsilon) & \rightarrow & \varepsilon \end{array}$$

defines this language. The size of the first argument of p^2 takes successively all the odd values. These number start from one (see the call $p^2(T,X)$ in the second clause) and if $|X|$ is an odd number, $|YT| = |X| + 2$ is the next odd number (see the third clause). When the last clause is verified, we can note that we have $|X| = 2p - 1$. It is not difficult to see that for any string in T^* of length $n \leq p^2$, the parser performs at most $\mathcal{O}(p)$ steps, and this gives a sub-linear $\mathcal{O}(\sqrt{n})$ parse time.

2.2 Proper and Simple RCGs

In this section we will introduce a few terms concerning various forms of RCGs.

In the sequel, without loss of generality, we will prohibit empty arguments in clause RHS's. It is always possible to replace any empty argument by a new variable, say X, and to add in this RHS a predicate call, say *empty*(X), assuming that *empty* is defined by the following clause:

$$empty(\varepsilon) \quad \rightarrow \quad \varepsilon$$

We will also assume that no argument has more than one instance of a given variable. As a matter of fact two occurrences of the same variable (in a clause) denote the same range and, an argument of the form $\alpha_1 X \alpha_2 X \alpha_3$ can only be instantiated if X and α_2 are bound to the same empty range. Thus, this argument can be replaced by $\alpha_1 X \alpha_2 \alpha_3$, by adding to the RHS the predicate call *empty*$(X\alpha_2)$.

Partly borrowed from [Groenink 97], we define the following.

Definition 5 *A clause is:*

- combinatorial *if at least one argument of its RHS predicates does not consist of a single variable;*
- bottom-up linear *(resp.* top-down linear*) if no variable appears more than once in its LHS (resp. RHS) arguments;*
- bottom-up erasing *(resp.* top-down erasing*) if there is at least one variable occurring in its RHS (resp. LHS) which does not appear in its LHS (resp. RHS);*
- linear *(resp.* erasing*) if it is both bottom-up linear and top-down linear (resp. bottom-up erasing and top-down erasing);*
- proper *if it is both non-combinatorial and non-erasing;*

- simple *if it is both proper and linear.*[2]

These definitions extend from clause to set of clauses (i.e. grammars).

Property 1 *For any RCG, there is an equivalent proper RCG.*

The proof of this result is in [Boullier 98a]. It must be noted that the transformed grammar is top-down non-linear (if we assume that the original is combinatorial or bottom-up erasing)[3], even when the original one is top-down linear. Moreover the proposed transformation increases the arity of the new grammar though its parsing time complexity (the number of free bounds) is not changed.

Definition 6 *An RCG is in k-*var *form if there is at most k variable occurrences in any given predicate argument.*

Property 2 *For any RCG, there is an equivalent RCG in 2-var form.*

Once again the arity of this equivalent RCG in 2-var form may be greater than the arity of the original one.

Remark: if the grammar is bottom-up non-erasing, the polynomial time complexity does not depend on its RHS's but only on the number of its LHS free bounds.

2.3 Closure Properties

Property 3 *The class of RCLs is closed under union, concatenation, iteration and intersection with a regular set.*

Let $G_1 = (N_1, T_1, V_1, P_1, S_1)$ and $G_2 = (N_2, T_2, V_2, P_2, S_2)$ be two RCGs which respectively define the languages L_1 and L_2. Without loss of generality, we assume that N_1 and N_2 are disjoint. For each closure property we will build a new RCG with the afore mentioned property and whose start predicate name is S.

union $L_1 \cup L_2$: add the two clauses $S(X) \rightarrow S_1(X)$ and $S(X) \rightarrow S_2(X)$.

concatenation $L_1 L_2$: add the clause $S(XY) \rightarrow S_1(X)\ S_2(Y)$.

iteration L_1^*: Add the two clauses $S(X) \rightarrow \varepsilon$ and $S(XY) \rightarrow S_1(X)\ S(Y)$.

[2]It is not the Groenink's definition of simple.

[3]The transformation from top-down erasing to top-down non-erasing does not change neither the arity, nor the linearity. When in a clause a variable of the LHS, say X disappears in the RHS, we simply add to this RHS the predicate $any(X)$ where any is a new predicate name defined by the two clauses

$$\begin{array}{lcl} any(aX) & \rightarrow & any(X) \\ any(\varepsilon) & \rightarrow & \varepsilon \end{array}$$

where the first clause is a schema over all terminals $a \in T$.

intersection with a regular set $L_1 \cap L_2$**:** add the clause $S(X) \rightarrow S_1(X)\ S_2(X)$ (here, we assume that L_2 is a regular language, and can therefore be defined by an RCG (see discussion of Property 11 below).

Property 4 *RCLs are closed under intersection and complementation.*

As for the proof of Property 3, we have

intersection $L_1 \cap L_2$**:** add the clause $S(X) \rightarrow S_1(X)\ S_2(X)$.

complementation $\overline{L_1}$**:** add the clause $S(X) \rightarrow \overline{S_1(X)}$.

Property 5 *The parse time of the intersection of two RCGs G_1 and G_2 is the maximum parse time of G_1 and G_2.*

The parse time of the complement of a RCG G is the parse time of G.

Those interesting properties trivially result from the fact that the supplementary clauses, introduced during the discussion of Property 4 are executed in constant time since they both appear at most one time in any derivation (whether it is valid or not).

Property 6 *The emptiness problem for (P)RCGs is undecidable.*

The proof directly results from two properties: it is undecidable whether or not the intersection of any two CFLs is empty (see [Hopcroft and Ullman 79]), and the intersection of two CFLs is an RCL (this will be shown later on at Property 12).

Below, we recall some definitions before stating a few more properties.

A *substitution* f is a mapping of an alphabet Σ onto subsets of T^*, for some alphabet T. Thus f associates a language with each symbol of Σ. The mapping f is extending from symbol to strings as follows:

1. $f(\varepsilon) = \varepsilon$
2. $f(xa) = f(x)f(a)$

The mapping f is extended from strings to languages in defining $f(L) = \cup_{x \in L} f(x)$.

An *homomorphism* h is a substitution such that $h(a)$ contains a single string for each a. We take $h(a)$ to be the string itself.

If h is an homomorphism, the *inverse homomorphic image* of a language L is $h^{-1}(L) = \{x \mid h(x) \in L\}$.

An abstract family of languages (AFL) is a collection of languages which is closed under union, concatenation, iteration, intersection with a regular set, homomorphism and inverse homomorphism.

Property 7 *The class of RCLs forms a pre-AFL (i.e. closed under marked product $L_1 a L_2$, marked iteration $(Lc)^*$, inverse homomorphism and intersection with regular sets).*

The notion of pre-AFL comes from [Ginsburg, Greibach, and Hopcroft 69]. All these closure properties, except the inverse homomorphism are obvious. This property has been proved by [Groenink 97] for simple LMGs, thus it is also true for PRCLs since simple LMGs and PRCGs are equivalent formalisms. However, a direct proof can easily be shown in slightly changing the RCG definition. The closure under inverse homomorphism trivially holds if we add in the definition of RCGs the homomorphism h itself. The language of such an extension is defined by $\{w \mid S(\bullet h(w) \bullet) \underset{G,h(w)}{\overset{+}{\Rightarrow}} \varepsilon\}$. The usual case corresponds to the identity homomorphism. Thus, if language L is defined by the *extended* RCG (N, T, V, P, S, h) where (N, T, V, P, S) is an RCG and h is an homomorphism of Σ onto T^*, and if h' is an homomorphism of some alphabet Δ onto Σ^*, the extended RCG $(N, T, V, P, S, h' \circ h)$ defines the inverse homomorphic image of L by h'.

Property 8 *RCLs are* not *closed under homomorphism (or substitution).*

It is a well known result (see e.g. [Ginsburg 75]) that any recursively enumerable language L can be described as $h(L_1 \cap L_2)$ where h is an homomorphism and the languages L_1 and L_2 are CF. Since, by Property 12, the intersection of two CFLs is an RCL, it follows that the closure of RCLs under homomorphism would imply that recursively enumerable language are included into RCLs!

However, it is interesting to see where an attempt to directly prove that RCLs are closed under homomorphism fails.

For an RCG G defining $\mathcal{L}(G)$, and an homomorphism h, the idea is to build a new RCG G' defining $\mathcal{L}(G') = h(\mathcal{L}(G))$ in simply replacing each occurrence of any terminal a by $h(a)$. Without loss of generality (see Property 1), we assume that the initial grammar G is in proper form, thus terminal symbols only occur in LHS arguments and G is top-down non-erasing. We construct G' as follows. For each clause of G, each occurrence of any terminal a is replaced by the terminal string $h(a)$.

We will prove that $h(\mathcal{L}(G)) \subseteq \mathcal{L}(G')$. We assume that G is in a normal form in which the occurrences of terminal symbols in (LHS) arguments are only grouped within clauses of the form

$$t_i(t_i) \quad \rightarrow \quad \varepsilon$$

one such clause for each $t_i \in T$. Here t_i also designates a predicate name, hereafter called *terminal predicate* which only defines the terminal t_i. To get such a normal form, we simply have to replace in each clause each occurrence of any terminal t_i by a new variable say X_{t_i} and to add in the RHS, if not already there, the predicate call $t_i(X_{t_i})$. Therefore, the only difference between G and G' is that the clauses

$$t_i(t_i) \quad \rightarrow \quad \varepsilon$$

of G are changed in G' by

$$t_i(h(t_i)) \quad \rightarrow \quad \varepsilon$$

If $w \in \mathcal{L}(G)$ and $x = h(w)$, we are going to show that $x \in \mathcal{L}(G')$.

Since $w = a_1 \dots a_j \dots a_n \in \mathcal{L}(G)$, there is a derivation $S(\bullet w \bullet) \underset{G,w}{\overset{+}{\Rightarrow}} \Gamma \underset{G,w}{\overset{+}{\Rightarrow}} \varepsilon$ where $\Gamma = a_1(w_1 \bullet a_1 \bullet w'_1) \dots a_j(w_j \bullet a_j \bullet w'_j) \dots a_n(w_n \bullet a_n \bullet w'_n)$ s.t. $\forall j, 1 \leq j \leq n$, $w_j a_j w'_j = w$. We can then mimic this derivation with G' on the source text $x = h(w)$, we get $S(\bullet x \bullet) \underset{G',x}{\overset{+}{\Rightarrow}} \Gamma'$ with $\Gamma' = a_1(h(w_1) \bullet h(a_1) \bullet h(w'_1)) \dots a_j(h(w_j) \bullet h(a_j) \bullet h(w'_j)) \dots a_n(h(w_n) \bullet h(a_n) \bullet h(w'_n))$ and, our result holds because trivially we have $\Gamma' \underset{G',x}{\overset{+}{\Rightarrow}} \varepsilon$.

Now, we want to show that $\mathcal{L}(G') \subseteq h(\mathcal{L}(G))$ i.e. $\forall x \in \mathcal{L}(G'), \exists w, x = h(w), w \in \mathcal{L}(G)$. Since $x \in \mathcal{L}(G')$, there is a derivation $S(\bullet x \bullet) \underset{G',x}{\overset{+}{\Rightarrow}} \Gamma' \underset{G',x}{\overset{+}{\Rightarrow}} \varepsilon$ where $\Gamma' = t_1(x'_1 \bullet x_1 \bullet x''_1) \dots t_k(x'_k \bullet x_k \bullet x''_k) \dots t_m(x'_m \bullet x_m \bullet x''_m)$ s.t. $\forall k, 1 \leq k \leq m$, $x'_k x_k x''_k = x$, and the t_k's are terminal predicate occurrences. If $t_k(x'_k \bullet x_k \bullet x''_k)$ is a predicate instantiation in Γ', we know that $t_k(x'_k \bullet x_k \bullet x''_k) \underset{G',x}{\Rightarrow} \varepsilon$, $x_k = h(t_k)$, and all the x_k's "cover" the string x. However, we cannot build a string w s.t. $x = h(w)$. The reason is that two x_k's can overlap. Assume we have $h(t_1) = ab$, $h(t_2) = ba$, $x = aba$, and the derivation $S(\bullet aba \bullet) \underset{G',x}{\overset{+}{\Rightarrow}} t_1(\bullet ab \bullet a)\ t_2(a \bullet ba \bullet) \underset{G',x}{\overset{+}{\Rightarrow}} \varepsilon$, there is no w s.t. $x = h(w)$. This problem is due to the non linearity of the formalism.

This phenomenon is illustrated by the following simple example.

Consider the language $L = b^*ab^* \cap a^*ba^*$, of course we have $L = \{ab, ba\}$. Using the first form, we define L by the RCG G

$$\begin{array}{lll}
S(X) & \rightarrow & b^*ab^*(X)\ a^*ba^*(X) \\
b^*ab^*(XaY) & \rightarrow & b^*(X)\ b^*(Y) \\
a^*ba^*(XbY) & \rightarrow & a^*(X)\ a^*(Y) \\
a^*(\varepsilon) & \rightarrow & \varepsilon \\
a^*(aX) & \rightarrow & a^*(X) \\
b^*(\varepsilon) & \rightarrow & \varepsilon \\
b^*(bX) & \rightarrow & b^*(X)
\end{array}$$

Now, if we take the homomorphism h s.t. $h(a) = u$ and $h(b) = u$, it is not difficult to see that, replacing the terminal symbols a and b by u, we get a grammar G' whose language is u^+. Therefore, if we take $x = u$, we have $u \in \mathcal{L}(G')$ but $u \neq h(ab)$ and $u \neq h(ba)$.

Therefore we have the negative result that

Property 9 *The class of RCLs are not an AFL.*

From what precedes, we know that, for a class of languages extending CFLs, these three properties are all incompatible:

1. the membership problem is decidable in polynomial time;
2. the class is closed under homomorphism;

3. the class is closed under intersection.

Therefore, for a formalism which aims at NL processing, the question is to choose at most two of these properties. For efficiency reasons, the first one seems unavoidable and stands as such as one criterion in the MCS formalisms definition. Now, the choice between the second and the third property is not evident. As [Savitch 89] comments, "*...in some circles [being a full AFL] invests the class [of languages] with a certain respectability.*" In fact, it is clear that the closure under homomorphism allows elegant proofs but practical applications of this property seem more seldom. On the other hand, closure under intersection, which is a novel property in both powerful and tractable formalisms, allows for new syntactic designs. Moreover, as with RCGs, if this property is reached without changing the structure of the component languages, we get an interesting modularity property (see Paragraph 2.4). Thus, we choose to lean in favor of the closure under intersection vs. closure under homomorphism.

2.4 Modularity

We also claim that RCGs are *modular*, in a sense now to be defined. If we consider the previous closure properties, they all are evidenced *without* changing the component grammars. We simply have to add a few clauses, leaving unchanged the initial grammars.

Let's take an example to illustrate this point. Assume we have two CFGs $G_1 = (N_1, T_1, P_1, S_1)$ and $G_2 = (N_2, T_2, P_2, S_2)$ which respectively define the languages L_1 and L_2. We can assume that $N_1 \cap N_2 = \emptyset$. If we try to build a CFG $G = (N, T, P, S)$ which defines the language $L = L_1 \cup L_2$, we have $N = N_1 \cup N_2 \cup \{S\}$, $T = T_1 \cup T_2$ and $P = P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}$. Here we can say that CFGs are *modular* w.r.t. the union operation since CFGs have the formal property to be closed under union and moreover this union is described by a method (here the additional rules $S \rightarrow S_1$ and $S \rightarrow S_2$) which preserves the components (grammars G_1 and G_2). However, CFGs are not modular w.r.t. intersection or complementation since we know that CFLs are not closed under intersection or complementation. If now we consider regular languages, we know that they possess the formal property of being closed under intersection and complementation; however we cannot say that they are modular w.r.t. these operations, since the initial grammars are not preserved in any sense in the final result. For example, if $G1$ and $G2$ are two regular CFGs, defining the languages L_1 and L_2, we know that it is possible to construct a regular CFG whose language is $L_1 \cap L_2$, but the resulting grammar G strongly differs from both G_1 and G_2.

Therefore, in this sense, to be modular, a syntactic formalism must have both closure properties and structure preserving properties.

Of course it would be of considerable benefit for a formalism to be modular w.r.t. intersection and complementation. The modularity w.r.t. intersection could allow one to directly define a language with the properties P_1 *and* P_2, starting from two grammars G_1 and G_2 describing P_1 and P_2, without changing neither G_1 nor G_2. Modularity w.r.t. complementation (or difference) could allow for example to describe on the one hand a general rule and, on the other hand, the exceptions to this general rule and then to group the two specifications.

Assume we want to write a grammar for a language which has the properties (structures described by) P_1 or P_2 and Q_1 or Q_2 but from which R_1 or R_2 are excluded. In a classical grammatical formalism the fact that each grammar for $P_1, \ldots, R_2$ is known does not bring any clue to solve this problem (if ever possible). In contrast, within the RCG formalism, the grammar:

$$\begin{array}{lcl}
S(X) & \rightarrow & P(X)\ Q(X)\ \overline{R(X)} \\
P(X) & \rightarrow & P_1(X) \\
P(X) & \rightarrow & P_2(X) \\
Q(X) & \rightarrow & Q_1(X) \\
Q(X) & \rightarrow & Q_2(X) \\
R(X) & \rightarrow & R_1(X) \\
R(X) & \rightarrow & R_2(X)
\end{array}$$

trivially gives the solution.

Such modular properties, could even allow us to design libraries of grammars describing such and such linguistic feature, in order to help a grammar writer who could pickup modules from this library at will to construct his own descriptions.

We can also remark that RCGs can easily deal with partial descriptions. Let L be a language defined by some RCG G and assume that "some part" of L is defined by a predicate p (and its derivatives). If we erase in G all the occurrences of p (in LHS *and* RHS) and its useless derivatives, we get a top-down erasing grammar G', which still accepts the sentences of L. This shows that we can construct an under-specified partial grammar G' for a full language L having the property $L \subseteq \mathcal{L}(G')$.

3 CFGs & 1-RCGs

RCLs can be viewed as a hierarchy of languages which relies upon the arity k of their grammars.[4]

In this section, we will study RCGs whose arity is one, the 1-RCGs, and concentrate on their relationships with CFGs.

Property 10 *Any non-combinatorial, bottom-up non-erasing* 1-*RCG can be parsed in cubic time.*

[4]We can also note that RCLs form a parse time based hierarchy w.r.t. several closure operations. This means that if L_1 and L_2 are two RCLs which both have an $\mathcal{O}(f(n))$ parse time complexity, the composition of L_1 and L_2 by some closure operation ρ is also parsable in time $\mathcal{O}(f(n))$. The closure operations ρ for which the above property applies, without any restriction on the parse time function f, are intersection, complementation, union, intersection with a regular set, and inverse homomorphism. This property also holds for concatenation and Kleene closure, but, in this case, the parse time function must be a polynomial which is at least cubic. These results directly follow from the discussion of Properties 3 and 4.

We can assume that the initial 1-RCG is also top-down non-erasing since the transformation (see Property 1) does not change the arity. We are going to transform any clause which is not in 2-var form into a set of equivalent clauses in 2-var form. A clause, where $p > 2$

$$\begin{array}{llll} c: & A_0(w_0X_1w_1\dots X_pw_p) & \rightarrow & A_1(X_1)\ A'_1(X_1)\dots \\ & & & A_2(X_2)\ A'_2(X_2)\dots \\ & & & \dots \\ & & & A_p(X_p)\ A'_p(X_p)\dots \end{array}$$

in which the order of the RHS predicates have been sorted according to the order of the variables in the LHS argument can be transformed into the following set of clauses where underlined strings denote new variables and the c_i's denote new predicate names (associated with the clause c)

$$\begin{array}{lll} A_0(w_0X_1w_1\underline{X_2w_2\dots X_pw_p}) & \rightarrow & A_1(X_1)\ A'_1(X_1)\dots \\ & & c_2(\underline{X_2w_2\dots X_pw_p}) \\ & \dots & \\ c_i(X_iw_i\underline{X_{i+1}w_{i+1}\dots X_pw_p}) & \rightarrow & A_i(X_i)\ A'_i(X_i)\dots \\ & & c_{i+1}(\underline{X_{i+1}w_{i+1}\dots X_pw_p}) \\ & \dots & \\ c_{p-1}(X_{p-1}w_{p-1}X_pw_p) & \rightarrow & A_{p-1}(X_{p-1})\ A'_{p-1}(X_{p-1})\dots \\ & & A_p(X_p)\ A'_p(X_p)\dots \end{array}$$

The cubic parse time results directly from the fact that in a bottom-up non-erasing 1-RCG in 2-var form there are at most three free bounds per clause.

Property 11 • *For any CFG, there is an equivalent simple 1-PRCG which can be parsed in cubic time.*

- *For any linear CFG*[5]*, there is an equivalent simple 1-PRCG which can be parsed in quadratic time.*
- *For any regular grammar, there is an equivalent simple 1-PRCG which can be parsed in linear time.*

Let $A \rightarrow w_0B_1w_1\dots B_pw_p$ be a production in some CFG (N,T,P,S) where $w_i \in T^*$ and $B_i \in N$ and let $A(w_0X_1w_1\dots X_pw_p) \rightarrow B_1(X_1)\dots B_p(X_p)$ be a corresponding clause where the X_i's are p different variables (even if some B_i's are identical predicate names). By construction, the arities of its predicates are one and this clause is simple. If we apply the proposed transformation to all the productions of any CFG, due to the bijection between P and the set of clauses, it is not difficult to see that we get a strongly equivalent simple 1-PRCG.

[5]no RHS of a production contains more than one instance of a nonterminal.

The converse is equally true, we can transform any simple 1-RCG into an equivalent CFG. Though the order of the RHS predicates in a clause is free, note that when we built a CF production from a simple clause whose arity is 1, the order of the RHS nonterminal symbols (if any) in this production is given by the order of the variables in the LHS argument of the original clause.

Therefore, the languages defined by the class of simple 1-PRCGs are the CFLs.

$$\text{CFLs} \quad = \quad \text{simple 1-PRCLs}$$

The cubic parse time complexity is a particular case of Property 10.

If the initial CFG is such that the number of nonterminal symbols in the RHS of any production is at most two (*binary branching form*), the corresponding simple 1-PRCG is in 2-var form. Of course, this 2-var form directly induces a cubic parse time.

If the CFG at hand is linear, the proposed transformation gives a simple 1-PRCG in 1-var form. Thus, each clause has at most two free bounds and the corresponding parsing is performed in quadratic time.

If the CFG at hand is regular (right or left-linear), the proposed transformation gives a simple 1-PRCG in 1-var form. But, in this case the number of free bounds reduces to one. Indeed, if the original is left-linear (resp. right-linear), the lower bound (resp. upper bound) of each range, is always the constant 0 (resp. n), if the length of the input string is n. Therefore, parsing of regular languages can be performed by simple 1-PRCGs in linear time.

Property 12 *The intersection of two CFLs is a proper 1-PRCL which can be parsed in cubic time.*

Let $G_1 = (N_1, T_1, P_1, S_1)$ and $G_2 = (N_2, T_2, P_2, S_2)$ be two CFGs defining the languages L_1 and L_2. We assume that $N_1 \cap N_2 = \emptyset$. Let $G'_1 = (N_1, T_1, V_1, P'_1, S_1)$ and $G'_2 = (N_2, T_2, V_2, P'_2, S_2)$ be the associated simple 1-PRCGs defining the same languages L_1 and L_2 (see discussion of Property 11). If S is a new (start) predicate name, the 1-PRCG

$$G' \quad = \quad (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, V_1 \cup V_2 \cup \{X\}, P'_1 \cup P'_2 \cup \{S(X) \rightarrow S_1(X)\ S_2(X)\}, S)$$

defines the language $L = L_1 \cap L_2$. Moreover, G' is proper since both G'_1 and G'_2 are proper, and so is the additional clause $S(X) \rightarrow S_1(X)\ S_2(X)$. However, G is *not* (top-down) linear. This non-linearity is due to the extra clause $S(X) \rightarrow S_1(X)\ S_2(X)$, which is in fact the rewriting rule which "computes" the intersection.

The cubic parse time is a particular case of Property 10.

Property 13 *The complement of a CFL is a simple 1-NRCL which can be parsed in cubic time.*

Let $G = (N, T, P, S)$ be a CFG defining the language L. Let $G' = (N, T, V, P', S)$ be the associated simple 1-PRCG defining the same languages L.

If S' is a new (start) predicate name, the 1-NRCG

$$G'' \quad = \quad (N \cup \{S'\}, T, V \cup \{X\}, P \cup \{S'(X) \to \overline{S(X)}\}, S')$$

defines the language $\overline{L}$. Obviously G'' is simple and negative.

The cubic parse time is again a particular case of Property 10.

Note that, by construction, G'' is consistent.

Remark that Property 12 shows that proper 1-PRCGs (as opposed to simple) are strictly more powerful than CFGs and by Property 13 that this is also the case for simple 1-NRCGs.

Property 14 *The class of 1-RCLs is closed under union, concatenation, iteration and intersection with a regular set.*

All these closure properties are trivially true when we restrict ourselves to the arity one (see Property 3).

Property 15 *The class of 1-RCLs is closed under intersection and complementation. The parsing time complexity of the resulting 1-RCG is the maximum complexity of its component grammars.*

Property 4 and 5 also apply to the 1-RCL subclass. Moreover, the intersection and complementation operations do not increase the parse time complexity. For example the intersection of two cubic time grammars has also a cubic time.

The 1-RCG class is thus also modular (see Paragraph 2.4).

Almost all properties of RCGs also apply to 1-RCGs. However, if we transform a combinatorial 1-RCG into a non-combinatorial one, our general algorithm will give a k-RCG, with $k > 1$, outside our subclass. The transformation of a combinatorial k-RCG into an equivalent non-combinatorial k-RCG, for any k is an open problem.

The next section will investigate the formal descriptive power of 1-RCGs.

4 Mildly Context-Sensitive Formalisms & 1-RCGs

The notion of mild context-sensitivity is an attempt by [Joshi 85] to express the formal power needed to define natural languages.

Definition 7 *A grammar (language) is* mildly context-sensitive *(MCSG, MCSL) if it has the following properties:*

1. *CFL are properly contained in MCSL;*
2. *a MCSL can be parsed in polynomial time;*
3. *MCSGs capture these three basic non CF constructions of natural languages, that is* multiple agreements*,* cross agreements *and* duplication*;*

4. a MCSL has the constant growth property.

The first point is motivated by the fact that CFGs are not adequate to define natural language since some phenomena are beyond their power [Shieber 85].

For efficiency reasons, the second point seems quite natural.

The third point simply exhibits some "classical" non CF linguistic phenomena that such formalisms must be able to handle.

Let us examine the fourth point. A language has a constant growth property if its sentences, sorted by increasing length, are such that two consecutive lengths do not differ by arbitrary numbers. This fourth point is sometimes changed in requiring the semi-linearity property, which is a slightly stronger property (semi-linearity implies constant growth). Both notions try to formalize the linguistic intuition that sentences of natural languages are built in applying linear operations to initial bounded structures. Obviously, the class of linguistic phenomena covered by this definition is far from being precise enough and leaves room for interpretation.

In [Boullier 98a] and [Boullier 98b] we showed that, on the one hand, RCGs are strictly more powerful than MCSGs and, on the other hand, how some MCSGs (let's cite tree adjoining grammars (TAGs) [Vijay-Shanker 87], linear indexed grammars (LIGs) [Gazdar 85], head grammars (HGs) [Pollard 84], linear context-free rewriting systems (LCFRS) [Vijay-Shanker, Weir, and Joshi 87]) can be translated into an equivalent simple PRCG, while preserving their parse time complexity.

In this section we shall study the relationships between 1-RCGs and MCSGs.

We already know that Point 1 (see Properties 11, 12 and 13) and Point 2 (see Property 10) of MCSGs are fulfilled by proper 1-RCGs thus now we shall concentrate on Points 3 and 4.

4.1 Multiple Agreements, Cross Agreements and Duplication

In this part we will show that multiple agreements, cross agreements and duplication properties are all captured by proper 1-RCGs.

If we look at the current hierarchy of MCSGs, we find at the bottom four equivalent formalisms (i.e. TAGs, LIGs, HGs and combinatorial categorial grammars [Steedman 87]) (see [Vijay-Shanker and Weir 94a]) which can be parsed in $\mathcal{O}(n^6)$ time, and at the top of the hierarchy, two equivalent formalisms LCFRS and multicomponent TAGs (MCTAGs) [Weir 88] which can be parsed in polynomial time (the degree of which is at least six). Another (linguistically serious) candidate are D-Tree grammars (DTGs) [Rambow, Vijay-Shanker, and Weir 95], but its belonging to the MCS class is questionable since they cannot express the duplication property. If it happens that proper 1-RCGs are MCS, since they can be parsed in cubic time (Property 10), they will become, the most efficient MCS formalism. However, it is clear that TALs, for example, are not included in proper 1-RCLs, because this will show that TALs can be parsed in cubic time.[6] Conversely, we will show that 1-RCLs are *not* included in TALs or even in LCFRLs.

[6]Of course, this argument is not a proof and the existence of a TAL which is not a proper 1-RCG still has to be shown.

Multiple agreements, crossed agreements and duplication used to be abstracted respectively by the three languages $L_1 = \{a^n b^n c^n \mid n \geq 1\}$, $L_2 = \{a^n b^m c^n d^m \mid m, n \geq 1\}$ and $L_3 = \{ww \mid w \in \{a, b\}^*\}$. We shall see now that L_1, L_2 and L_3 are all proper 1-RCLs.

- L_1 can be seen as the intersection of two CFLs $L_1' = \{a^n b^n c^m \mid m, n \geq 1\}$ and $L_1'' = \{a^n b^m c^m \mid m, n \geq 1\}$. Thus L_1 is a proper 1-RCL (Property 12).
- L_2 can be seen as the intersection of two CFLs $L_2' = \{a^n b^m c^n d^l \mid l, m, n \geq 1\}$ and $L_2'' = \{a^n b^m c^l d^m \mid l, m, n \geq 1\}$. Thus L_2 is a proper 1-RCL (Property 12).
- It can be easily shown that $\overline{L_3}$ is a CFL. Let $L_3^t = \{w_1 t w_2 \mid |w_1| = |w_2|, t \in \{a, b\}\}$, be the CFL whose sentences are of odd size and their middle letter is a t. Of course L_3^a and L_3^b are both included in $\overline{L_3}$. Let $L_3'' = \overline{L_3} - (L_3^a \cup L_3^b)$ be the part of $\overline{L_3}$ whose strings have an even length. A string $w = w_1 t w_2 w_1' t' w_2'$ is in L_3'' iff we have $|w_1| = |w_1'|$, $|w_2| = |w_2'|$, $t, t' \in \{a, b\}$, and $t \neq t'$. Let $w' = w_1 t w_1' w_2 t' w_2'$. Since $w' \in L_3'' \iff w \in L_3''$, we have $L_3'' = L_3^a L_3^b \cup L_3^b L_3^a$. L_3'' is therefore a CFL and $\overline{L_3} = L_3^a \cup L_3^b \cup L_3^a L_3^b \cup L_3^b L_3^a$ is also CF. Thus, L_3 which is the complement of a CFL is a proper 1-RCL (Property 13).

Point 3 of the MCSG definition is thus fulfilled. Moreover, the corresponding languages can be parsed in cubic time at worst (Properties 12 and 13). In defining these grammars, we shall see below that L_1, L_2 and L_3 can be parsed in less than cubic time.

4.1.1 Multiple Agreements

The following set of proper clauses defines $L_1 = \{a^n b^n c^n \mid n \geq 1\}$

$$\begin{array}{lcl}
S_1(X) & \rightarrow & a^n b^n c^+(X)\ a^+ b^m c^m(X) \\
a^n b^n c^+(Xc) & \rightarrow & a^n b^n c^*(X) \\
a^n b^n c^*(Xc) & \rightarrow & a^n b^n c^*(X) \\
a^n b^n c^*(aXb) & \rightarrow & a^n b^n(X) \\
a^n b^n(\varepsilon) & \rightarrow & \varepsilon \\
a^n b^n(aXb) & \rightarrow & a^n b^n(X) \\
a^+ b^m c^m(aX) & \rightarrow & a^* b^m c^m(X) \\
a^* b^m c^m(aX) & \rightarrow & a^* b^m c^m(X) \\
a^* b^m c^m(bXc) & \rightarrow & b^m c^m(X) \\
b^m c^m(X) & \rightarrow & \varepsilon \\
b^m c^m(bXc) & \rightarrow & b^m c^m(X)
\end{array}$$

It is not difficult to see that each terminal symbol in X is scanned exactly one time[7] by the predicate $a^n b^n c^+(X)$ and its derivatives, and also one time by $a^+ b^m c^m(X)$, leading to a total linear parse time (see Table 2 below for practical results).

[7]if the source text is a sentence or at most one time, if the source text is not a sentence.

4.1.2 Cross Agreements

The following set of proper clauses defines $L_2 = \{a^n b^m c^n d^m \mid m, n \geq 1\}$

$$\begin{array}{lcl}
S_2(X) & \rightarrow & a^n b^+ c^n d^+(X)\ a^+ b^m c^+ d^m(X) \\
a^n b^+ c^n d^+(Xd) & \rightarrow & a^n b^+ c^n d^*(X) \\
a^n b^+ c^n d^*(Xd) & \rightarrow & a^n b^+ c^n d^*(X) \\
a^n b^+ c^n d^*(aXc) & \rightarrow & a^n b^+ c^n(X) \\
a^n b^+ c^n(aXc) & \rightarrow & a^n b^+ c^n(X) \\
a^n b^+ c^n(bX) & \rightarrow & b^*(X) \\
b^*(bX) & \rightarrow & b^*(X) \\
b^*(\varepsilon) & \rightarrow & \varepsilon \\
a^+ b^m c^+ d^m(aX) & \rightarrow & a^* b^m c^+ d^m(X) \\
a^* b^m c^+ d^m(aX) & \rightarrow & a^* b^m c^+ d^m(X) \\
a^* b^m c^+ d^m(bXd) & \rightarrow & b^m c^+ d^m(X) \\
b^m c^+ d^m(bXd) & \rightarrow & b^m c^+ d^m(X) \\
b^m c^+ d^m(cX) & \rightarrow & c^*(X) \\
c^*(cX) & \rightarrow & c^*(X) \\
c^*(\varepsilon) & \rightarrow & \varepsilon
\end{array}$$

As for L_1, we can show that the corresponding parse time is linear.

4.1.3 Duplication

The following set of proper clauses defines $L_3 = \{ww \mid w \in \{a, b\}^*\}$

$$\begin{array}{lcl}
S_3(X) & \rightarrow & \overline{w_1 w_2(X)} \\
w_1 w_2(X) & \rightarrow & w_1 a w_2(X) \\
w_1 w_2(X) & \rightarrow & w_1 b w_2(X) \\
w_1 w_2(XY) & \rightarrow & w_1 a w_2(X)\ w_1 b w_2(Y) \\
w_1 w_2(XY) & \rightarrow & w_1 b w_2(X)\ w_1 a w_2(Y)
\end{array}$$

where the predicate names $w_1 a w_2$ and $w_1 b w_2$ satisfy the following clause schema

$$\begin{array}{lcl}
w_1 t w_2(t) & \rightarrow & \varepsilon \\
w_1 t w_2(aXa) & \rightarrow & w_1 t w_2(X) \\
w_1 t w_2(aXb) & \rightarrow & w_1 t w_2(X) \\
w_1 t w_2(bXa) & \rightarrow & w_1 t w_2(X) \\
w_1 t w_2(bXb) & \rightarrow & w_1 t w_2(X)
\end{array}$$

which simply checks that the string selected by its input range has an odd size and that its middle symbol is t.

This grammar has a quadratic parse time behavior. The number of predicate calls in the fourth and fifth clause is linear in the length of the source text, and each predicate call $w_1 t w_2(X)$ gives its answer in time linear with $|X|$.

However, using 1-RCGs, we do not know how to get a linear parse time, though this linear parse time can easily be reached if we define L_3 by a the 2-RCG

$$S_3(XY) \quad \rightarrow \quad eq(X, Y)$$

4.2 Generalizations of Multiple Agreements, Cross Agreements and Duplication

In the previous section, we have seen how languages which abstract multiple agreements, cross agreements and duplication properties can be defined with proper 1-RCGs. It is a well known result that these languages are also TALs. In this section we shall see some generalizations which are still proper 1-RCGs, but are beyond the formal power of TAGs.

4.2.1 Generalization of Multiple Agreements

The multiple agreement property can easily be generalized to any fixed number of components. The language $\{a_1^n \dots a_k^n \mid n \geq 1, k \geq 3\}$ where k is fixed and $a_i \neq a_{i+1}$, $1 \leq i < k$ can be defined as the intersection of two CFLs

$$\{a_1^{n_1} a_2^{n_1} \dots a_{2i-1}^{n_i} a_{2i}^{n_i} \dots a_k^{\lfloor \frac{k+1}{2}}\} \cap \{a_1^{n_1} a_2^{n_2} a_3^{n_2} \dots a_{2i}^{n_{i+1}} \; a_{2i+1}^{n_{i+1}} \dots a_k^{\lfloor \frac{k}{2}+1}\}$$

Each of these CFL can be defined by a linear time proper 1-RCG. Thus, by Property 5, their intersection can be defined by a proper 1-RCG parsable in linear time.

For example, the non TAL language $\{a^n b^n c^n d^n e^n \mid n \geq 1\}$ with five components, can be defined as the intersection of two CFLs $\{a^n b^n c^m d^m e^l \mid l, m, n \geq 1\}$ and $\{a^l b^n c^n d^m e^m \mid l, m, n \geq 1\}$ which can be both defined by proper 1-RCGs. The following linear time proper 1-RCG, for example, defines the language $a^n b^n c^m d^m e^l$.

$$\begin{array}{lcl}
a^n b^n c^m d^m e^+(Xe) & \rightarrow & a^n b^n c^m d^m e^*(X) \\
a^n b^n c^m d^m e^*(Xe) & \rightarrow & a^n b^n c^m d^m e^*(X) \\
a^n b^n c^m d^m e^*(Xd) & \rightarrow & a^n b^n c^+ d^*(X) \; a^+ b^+ c^m d^{m-1}(X) \\
a^n b^n c^+ d^*(Xd) & \rightarrow & a^n b^n c^+ d^*(X) \\
a^n b^n c^+ d^*(Xc) & \rightarrow & a^n b^n c^*(X) \\
a^n b^n c^*(Xc) & \rightarrow & a^n b^n c^*(X) \\
a^n b^n c^*(aXb) & \rightarrow & a^n b^n(X) \\
a^n b^n(\varepsilon) & \rightarrow & \varepsilon \\
a^n b^n(aXb) & \rightarrow & a^n b^n(X) \\
a^+ b^+ c^m d^{m-1}(aX) & \rightarrow & a^* b^+ c^m d^{m-1}(X) \\
a^* b^+ c^m d^{m-1}(aX) & \rightarrow & a^* b^+ c^m d^{m-1}(X) \\
a^* b^+ c^m d^{m-1}(bX) & \rightarrow & b^* c^m d^{m-1}(X) \\
b^* c^m d^{m-1}(bX) & \rightarrow & b^* c^m d^{m-1}(X) \\
b^* c^m d^{m-1}(cX) & \rightarrow & c^m d^m(X) \\
c^m d^m(\varepsilon) & \rightarrow & \varepsilon \\
c^m d^m(cXd) & \rightarrow & c^m d^m(X)
\end{array}$$

4.2.2 Generalization of Cross Agreements

The previous result is also true for the generalization of the crossed agreements property. The language $\{a_1^{n_1} \dots a_k^{n_k} b_1^{n_1} \dots b_k^{n_k} \mid n_i > 0, k \geq 2\}$ where k is fixed and $a_i \neq a_{i+1}$, $1 \leq i < k$, $b_j \neq b_{j+1}$, $1 \leq j < k$ and $a_k \neq b_1$ can be defined as the intersection of k CFLs $\bigcap_{1 \leq i \leq k} a_1^+ \dots a_{i-1}^+ a_i^{n_i} a_{i+1}^+ \dots a_k^+ b_1^+ \dots b_{i-1}^+ b_i^{n_i} b_{i+1}^+ \dots b_k^+$. Each of these CFL, and thus their intersections, can be defined by a linear time proper 1-RCG.

For example the non TAL language $\{a^n b^m c^l d^n e^m f^l \mid l, n, m \geq 1\}$ can be defined by $a^n b^+ c^+ d^n e^+ f^+ \cap a^+ b^m c^+ d^+ e^m f^+ \cap a^+ b^+ c^l d^+ e^+ f^l$ where, for example, the middle language can be defined by

$$\begin{array}{lcl}
a^+b^mc^+d^+e^mf^+(aX) & \rightarrow & a^*b^mc^+d^+e^mf^+(X) \\
a^*b^mc^+d^+e^mf^+(aX) & \rightarrow & a^*b^mc^+d^+e^mf^+(X) \\
a^*b^mc^+d^+e^mf^+(bX) & \rightarrow & b^{m-1}c^+d^+e^mf^+(X) \\
b^{m-1}c^+d^+e^mf^+(Xf) & \rightarrow & b^{m-1}c^+d^+e^mf^*(X) \\
b^{m-1}c^+d^+e^mf^*(Xf) & \rightarrow & b^{m-1}c^+d^+e^mf^*(X) \\
b^{m-1}c^+d^+e^mf^*(Xe) & \rightarrow & b^mc^+d^+e^m(X) \\
b^mc^+d^+e^m(bXe) & \rightarrow & b^mc^+d^+e^m(X) \\
b^mc^+d^+e^m(cXd) & \rightarrow & c^*d^*(X) \\
c^*d^*(\varepsilon) & \rightarrow & \varepsilon \\
c^*d^*(cX) & \rightarrow & c^*d^*(X) \\
c^*d^*(dX) & \rightarrow & d^*(X) \\
d^*(\varepsilon) & \rightarrow & \varepsilon \\
d^*(dX) & \rightarrow & d^*(X)
\end{array}$$

The linear behavior can also be understood if we remark that each terminal symbol is scanned at most k times.

4.2.3 Generalization of Duplication

However, we do not know how to generalize the duplication property within the proper 1-RCG framework, though a quadratic parsing time can easily be reached for the (ambiguous) language $\{ww^* \mid w \in \{a, b\}^*\}$, with the following 2-RCG:

$$\begin{array}{lcl}
S(\varepsilon) & \rightarrow & \varepsilon \\
S(XY) & \rightarrow & \overline{len(0, X)}\ \overline{len(0, Y)}\ w^*(X, Y) \\
w^*(X, \varepsilon) & \rightarrow & \varepsilon \\
w^*(X, YZ) & \rightarrow & eq(X, Y)\ w^*(X, Z)
\end{array}$$

where *len* and *eq* are predefined predicate names (see Paragraph 2.1.1).

Note that $w^*(X, Y)$ gives its answer in time linear with $\max(|X|, |Y|)$. The quadratic behavior is due to the fact that the number of calls $w^*(X, Y)$ in the second clause is linear with $n = |X| + |Y|$ and that $\max(|X|, |Y|) \leq |X| + |Y|$.

The parse time for the previous language can be (slightly) enhanced by the following grammar where the second clause is changed by

$$S(XY) \quad \to \quad \overline{len(0,X)}\ \overline{len(0,Y)}\ mul(X,Y)\ w^*(X,Y)$$

where the predicate name *mul* checks whether the size of its second argument Y is a multiple of the size of its first argument X

$$\begin{array}{lcl} mul(X,\varepsilon) & \to & \varepsilon \\ mul(X,YZ) & \to & eqlen(X,Z)\ mul(X,Y) \end{array}$$

For a given (X,Y), on one side, the predicate name *mul* executes in $\frac{|Y|}{|X|}$ steps. Thus, for all the X's s.t. $1 \leq |X| \leq n = |Y|$, the total time is $\Sigma_{|X|=1}^{|X|=n} \frac{n}{|X|}$ which is of the order $\mathcal{O}(n \log n)$. On the other side the predicate $w^*(X,Y)$ executes itself in linear time with $|Y|$. If we assume a left-to-right evaluation order strategy, predicate $w^*(X,Y)$ in the (new) second clause is called iff $mul(X,Y)$ succeeds. Let $d(n)$ be the number of w's in a source text of length n ($d(n) = |\{p \mid \exists k > 0, p > 1, n = pk\}|$). Therefore the total time taken by this clause (and its derivatives) is $\mathcal{O}(n \log n) + \mathcal{O}(nd(n))$. Let $n = p_1^{q_1} \dots p_m^{q_m}$ be the integer factorization of n where $p_1 \dots p_m$ are the distinct prime factors (greater than one) of n and $q_1 \dots q_m$ are their multiples, we have $d(n) = (q_1+1)(q_2+1)\dots(q_m+1) - 1$. We can show that $1 \leq d(n) < n$, the lower bound is reached when n is a prime number (there are exactly n substrings w of length 1), and the upper bound is reached when we have $n = p!$. Therefore $\mathcal{O}(nd(n))$ is sub-quadratic and so is the total parse time.

A slightly modified version $\{w\{cw\}^+ \mid w \in \{a,b\}^*\}$ can be defined by a sub-quadratic parse time proper 1-RCG or by a linear time RCG.

4.3 Constant Growth Property

This part study whether 1-RCGs have the constant growth property.

We already know that general RCGs do not have the constant growth property since languages such $\{a^{2^p} \mid p \geq 0\}$ and $\{a^{p^2} \mid p \geq 0\}$ are RCLs.

We shall show below that proper 1-RCLs do not have the constant growth property either (and are thus non semi-linear).

Consider the language $\{ab^k ab^{k-1} ... ab^2 ab \mid k \geq 1\}$. This language does not have the constant growth property since the difference of the lengths of two consecutive sentences, one for $k = i-1$ and the other for $k = i$ is $|ab^k| = i+1$, is not bounded by a constant.

However, the following proper 1-RCG defines this language.

$$\begin{array}{lcl}
S(ab) & \rightarrow & \varepsilon \\
S(aX) & \rightarrow & b^k \dots b^2 ab(X) \\
b^k \dots b^2 ab(bbab) & \rightarrow & \varepsilon \\
b^k \dots b^2 ab(X) & \rightarrow & b^k ab^{k-1} \dots (X) \ \dots b^k \dots b^2 ab(X) \\
b^k ab^{k-1} \dots (XaY) & \rightarrow & b^k ab^{k-1}(X) \ any(Y) \\
b^k ab^{k-1}(ba) & \rightarrow & \varepsilon \\
b^k ab^{k-1}(bXb) & \rightarrow & b^k ab^{k-1}(X) \\
\dots b^k \dots b^2 ab(bX) & \rightarrow & \dots b^k \dots b^2 ab(X) \\
\dots b^k \dots b^2 ab(aX) & \rightarrow & b^k \dots b^2 ab(X)
\end{array}$$

The idea behind this grammar is very simple. The predicate name $b^k \dots b^2 ab$ checks that the string denoted by its input range either has the form "*bbab*" or has the form $XaYaZ$ where X is b^k and Y is b^{k-1}, and YaZ has (recursively) the same form.

The corresponding parse time is quadratic since in $b^k ab^{k-1} \dots (XaY)$ the upper bound of Y is always n. We can note that this language can be defined by a linear time RCG.

We have just shown that proper 1-RCLs do not have the constant growth property and thus, strictly speaking, 1-RCLs are not MCSLs. However, recall that the intention behind this last property was to approximate the linguistic intuition that sentences of natural languages are built by linear operations acting on initial structures of bounded size. On the other hand, note that the previous language can be seen as an abstraction of sentences of the form "five thousand thousand thousand five thousand thousand five thousand" (i.e. 5 005 005 000) and, as pointed out by [Groenink 97], we can wonder whether such a language should not be allowed to count as not excessively growing and could thus be included within the scope of MCSLs.

5 Conclusion

In this paper we present 1-RCGs, a subclass of RCGs, whose formal power allows to express numerous phenomena occurring in natural languages, while staying computationally efficient.

Its possibilities to describe natural languages are illustrated by the fact that it is both a strict extension of CFGs and it allows the processing of multiple agreements, crossed agreements and duplication phenomena. Moreover, this extra power is for free since the corresponding parse time complexity is at worst cubic in the length of the input string.[8] It is thus a candidate to be a mildly context-sensitive formalism, even if it is, in some sense, too powerful, since it can define languages which do not have the constant growth property. However, as pointed out in Paragraph 4.3, this extra power may be useful to describe some "not excessively growing" natural language phenomena. If we assume that proper 1-RCGs are MCSGs, it is certainly the most efficient mildly context-sensitive formalism since it seems

[8] As for RCGs, the time is linear in the size of the grammar.

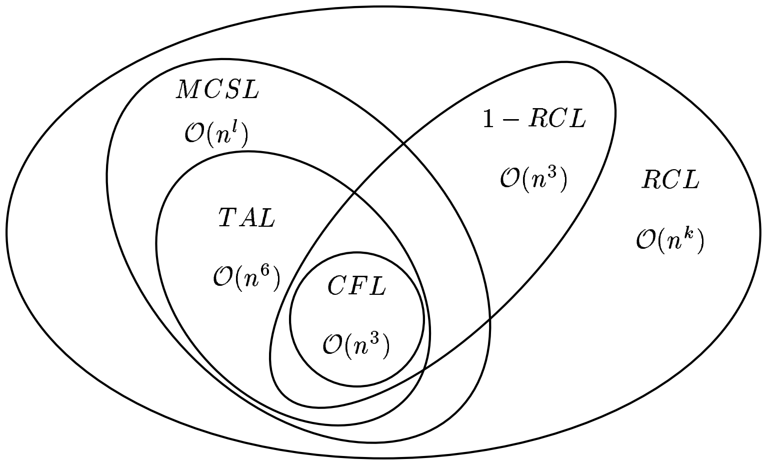

Table 1: Summary of formal powers and time complexities.

unlikely to find an extension of CFGs with less than a cubic parse time. In Table 1, we have depicted the respective formal power and parse time of RCLs and MCSLs.

On the other hand, we have shown that the class of 1-RCLs are closed under union, concatenation, iteration, intersection and complementation. Since these closure properties are reached in preserving their basic grammar components, 1-RCGs, as RCGs, can be qualified of modular and allow to imagine libraries of linguistic modules in which any language designer can pick up at will when he wants to define such and such phenomenon. We have remarked that (1-)RCGs can easily deal with partial descriptions.

We have also characterized the subclass of CSLs including the intersection of two CFLs and the complement of a CFL as being the proper 1-RCL class.

A prototype version which takes as input an RCG and produces a parser/recognizer for its language has been implemented. The example grammars of this report, and their corresponding complexity have all been checked. Table 2 gives a summary of these results. The tests have been performed on a 40Mhz Sun SS10. We used the recognizer version for all these tests, except for the ambiguous language ww^+ where a parser was used. For this language, with a^{2700} as a source text, the parser produces 35 parse trees.[9] The extra formal power of full RCGs over 1-RCGs is not only for free (costly constructions only induce a higher polynomial degree at the point of their usage), but, as illustrated in Table 2, the parse time, for some languages, may even decrease dramatically.

We firmly believe that full RCGs have the right level of syntactic power even if it may be argued that their level of context-sensitivity seems too high to describe what is needed in natural language processing.

[9]We have $2700 = 2^2 3^3 5^2$, and thus $d(n) = (2+1)(3+1)(2+1) - 1 = 35$ (see Paragraph 4.2.3).

Language	Type	Complexity	Time	Source length
$a^n b^n c^n$	1-RCG	linear	1 s	$(n = 25\,000)$ 75 000
$a^n b^n c^n d^n e^n$	1-RCG	linear	1 s	$(n = 8\,000)$ 40 000
$a^n b^m c^n d^m$	1-RCG	linear	1 s	$(n = 15\,000, m = 20\,000)$ 70 000
$a^n b^m c^l d^n e^m f^l$	1-RCG	linear	1 s	$(n = 6\,000, m = l = 7\,000)$ 40 000
ww	1-RCG	quadratic	1 s	$(\|w\| = 230)$ 460
ww	RCG	linear	1 s	$> 10^6$
$ab^k ab^{k-1} \dots ab$	1-RCG	quadratic	1 s	$(k = 76)$ 3 002
$ab^k ab^{k-1} \dots ab$	RCG	linear	0.160 s	$(k = 1\,000)$ 501 500
$w\{cw\}^k$	1-RCG	sub-quadratic	1 s	$(\|w\| = 20, k = 61)$ 1 303
$w\{cw\}^k$	RCG	linear	0.280 s	$(\|w\| = 1\,000, k = 999)$ 1 000 999
ww^k	RCG	quadratic	1 s	$(\|w\| = 20, k = 135)$ 2 700
a^{p^2}	RCG	$\mathcal{O}(n \log n)$	1 s	$(p = 540)$ 291 600
a^{p^2}	RCG	$\mathcal{O}(\sqrt{n})$	neglect.	$(p = 540)$ 291 600

Table 2: Implementation results.

The properties of RCGs (power, efficiency, modularity, ...) make of this formalism, in our opinion, a good challenger to replace CFGs. A supplementary argument is that RCGs can be seen as a syntactic backbone upon which decorations, taken from other domains (probabilities, logical terms, feature structures, ...), can be grafted.

Therefore, RCGs may, as such, be seen as a (linguistic) description formalism. However, since many popular natural language description formalisms are on one hand already widely used and, on the other hand, can be (often easily) transformed into an equivalent RCG, as efficient as the initial grammar, RCGs can also be seen as an (high level) implementation formalism.

References

[Boullier 98a] Pierre Boullier. 1998. Proposal for a Natural Language Processing Syntactic Backbone. In *Research Report No 3342* at `http://www.inria.fr/RRRT/RR-3342.html`, INRIA-Rocquencourt, France, Jan. 1998, 41 pages.

[Boullier 98b] Pierre Boullier. 1998. A Generalization of Mildly Context-Sensitive Formalisms. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, University of Pennsylvania, Philadelphia, Pennsylvania, 1–3 August, pages 17–20.

[Earley 70] Jan Earley. 1970. An Efficient Context-Free Parsing Algorithm In *Communication of the ACM*, Vol. 13, No 2, Feb. 1970, pages 94–102.

[Gazdar 85] G. Gazdar. 1985. Applicability of Indexed Grammars to Natural Languages. In *Technical Report CSLI-85-34*, Center for Study of Language and Information, 1985.

[Ginsburg, Greibach, and Hopcroft 69] Seymour Ginsburg, Sheila Greibach, John E. Hopcroft. 1969. Studies in Abstract Families of Languages. In *Memoirs of the American Mathematical Society*, Providence, Rhode Island, No 87.

[Ginsburg 75] Seymour Ginsburg. 1975. Formal Languages. North-Holland/Am. Elsevier, Amsterdam, Oxford/New-York.

[Groenink 97] Annius V. Groenink. 1997. SURFACE WITHOUT STRUCTURE Word order and tractability issues in natural language analysis. PhD thesis, Utrecht University, The Netherlands, Nov. 1977, 250 pages.

[Hopcroft and Ullman 79] John E. Hopcroft, Jeffrey D. Ullman. 1979. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, Mass.

[Joshi 85] Aravind K. Joshi. 1985. How much context-sensitivity is necessary for characterizing structural descriptions — Tree Adjoining Grammars. In *Natural Language Processing — Theoritical, Computational and Psychological Perspective*, D. Dowty, L. Karttunen, and A. Zwicky, editors, Cambridge University Press, New-York, NY.

[Joshi, Vijay-Shanker, and Weir 91] Aravind K. Joshi, K. Vijay-Shanker, David Weir. 1991. The convergence of mildly context-sensitive grammatical formalisms. In *Foundational Issues in Natural Language Processing*, P. Sells, S. Shieber, and T. Wasow editors, MIT Press, Cambridge, Mass.

[Pollard 84] Carl J. Pollard. 1984. Generalized Phrase Structure Grammars, Head Grammars and Natural Language. PhD thesis, Stanford University.

[Rambow, Vijay-Shanker, and Weir 95] Owen Rambow, K. Vijay-Shanker, David J. Weir. 1995. D-Tree Grammars. In *Proceedings of the 33th Annual Meeting of the Association for Computational Linguistics (ACL95)*, pages 151–158.

[Rounds 88] William C. Rounds. 1988. LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. In *ACL Computational Linguistics*, Vol. 14, No. 4, pages 1–9.

[Savitch 89] W. Savitch. 1989. A Formal Model for Context-Free Languages Augmented with Reduplication. In *Computational Linguistics*, 15, pages 250–261.

[Shieber 85] Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. In *Linguistics and Philosophy*, Vol. 8, pages 333–343.

[Steedman 87] M. Steedman. 1987. Combinatory grammars and parasitic gaps. In *Natural language and Linguistic Theory*, 1987.

[Vijay-Shanker 87] K. Vijay-Shanker. 1987. A study of tree adjoining grammars. *PhD thesis*, University of Pennsylvania.

[Vijay-Shanker, Weir, and Joshi 87] K. Vijay-Shanker, David J. Weir, Aravind K. Joshi. 1987. Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics (ACL'87),* Stanford University, CA, pages 104–111.

[Vijay-Shanker and Weir 94a] K. Vijay-Shanker, David J. Weir. 1994. The equivalence of four extensions of context-free grammars. In *Math. Systems Theory*, Vol. 27. pages 511–546

[Weir 88] David J. Weir. 1988. Characterizing Mildly Context-Sensitive Grammar Formalisms. PhD thesis, University of Pennsylvania, Philadelphia, PA.

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399