Proposal for a Natural Language Processing Syntactic Backbone

Pierre Boullier

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Proposal for a Natural Language Processing Syntactic Backbone

Pierre Boullier

N ° 3342

Janvier 1998

THÈME 3

ISSN 0249-6399

INRIA
ROCQUENCOURT

Proposal for a Natural Language Processing Syntactic Backbone

Pierre Boullier*

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

Rapport de recherche n ° 3342 — Janvier 1998 — 41 pages

Abstract: The purpose of this paper is to present a grammatical formalism that extends context-free grammars and aims at being a convincing challenger as a syntactic base for various tasks, especially in natural language processing. These grammars are powerful, they strictly include mildly context-sensitive languages, while staying computationally tractable, since sentences are parsed in polynomial time. Moreover, this formalism allows a form of modularity which may lead to the design of libraries of reusable generic grammatical components. And, last, it can act as a syntactic backbone upon which decorations from other domains (say feature structures) can be grafted.

Key-words: grammar formalisms, context-sensitive parsing, shared forests, complexity of parsing, modularity.

(Résumé : tsvp)

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

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Proposition d'un support syntaxique pour le traitement des langues naturelles

Résumé : Cet article présente un formalisme grammatical, plus général que les grammaires non contextuelles, suffisamment convaincant pour servir de base à différentes tâches, particulièrement en traitement de la langue. Ces grammaires sont puissantes, elles incluent strictement les langages modérément contextuels, tout en restant utilisables en pratique ; leurs phrases sont analysées en temps polynomial. De plus ce formalisme permet une forme de modularité qui peut conduire à la conception de librairies de composants grammaticaux génériques réutilisables. Finalement, il peut être utilisé comme structure syntaxique supportant des décorations à valeur d'en d'autres domaines (par exemple les structures de trait).

Mots-clé : formalismes grammaticaux, analyse syntaxique contextuelle, forêts partagées, complexité de l'analyse, modularité.

1 Introduction

The great number of syntactic formalisms upon which natural language processing is based may be interpreted in two ways: on one hand it shows that this research field is very active and on the other hand it shows that, manifestly, there is no consensus for a single formalism and that the one with the *right* properties is still to be discovered. What properties should have such an ideal formalism ? Of course, this formalism must allow the description of the difficult features which have been identified so far in various natural languages while staying computationally tractable. We know that context-free grammars (CFGs) cannot play this role due to their lack of expressiveness. On the other hand, context sensitive grammars are too greedy in computer time.

A negative point which must be noted with usual grammars is their lack of modularity. To be modular, a formalism must possess both some formal properties and some structure preserving properties. Let's take an example to illustrate this point. Assume we have two context-free grammars (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 can 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 structure of its components (parse trees of G_1 and G_2). Conversely, 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 properties, since the structure is not preserved in any sense. For example, take a regular CFG G, defining the language L, we know that it is possible to construct a regular CFG whose language is $\overline{L}$, but its parse trees are not related with the parse trees of G.

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, assuming that we have 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 mix the two specifications.

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 in order to construct its own description.

The purpose of this paper, which is based upon the work of Groenink (see [Groenink 97]), is to present a simple modular (in the above sense) grammatical formalism, which is parsed in polynomial time and which allows to describe, at a pure syntactic level, many linguistic phenomena. Moreover, this formalism can be considered itself as a syntactic backbone upon which decorations taken from other domains (Herbrand, feature structures, ...) can be added.

2 Conventions & Notations

If E is an enumerable ordered set, $E = \{e_1, e_2, \dots\}$ the corresponding sequence $e_1, e_2, \dots$ of elements in E can be denoted by a vector notation $\vec{E}$ and the selection of vector components (i.e. indexing) $\vec{E}[1], \vec{E}[2], \dots$ denote respectively the elements e_1, e_2, ... In some cases E can be mapped to nonnegative integers, $E = \{e_0, e_1, \dots\}$, in this case $\vec{E}[0]$ denotes e_0. If E is finite, its cardinality is denoted by l_E or $l_{\vec{E}}$. The vector notation may also be used to convey the idea of sequences (*tuples*).

3 Positive Range Concatenation Grammar

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

$$\psi_0 \rightarrow \psi_1 \dots \psi_m$$

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

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

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.

PRCGs can be considered as a variant of Literal Movement Grammars (LMGs) of Groenink (see [Groenink 97])[1].

We shall see that the language defined by such a grammar does not depend either upon the order of the predicates $\psi_1\psi_2 \dots \psi_m$ occurring in the right-hand-side (RHS) of a clause nor upon duplicate predicates (multiple occurrence). Therefore the RHS of clauses must be considered as sets rather than sequences.

The scope of variables is local to a clause and as such we will assume that the clauses in P are all different after any variable renaming or permutation of its RHS predicates.

A predicate whose nonterminal part is A is called an A-predicate. If the left-hand side (LHS) of a clause is an A-predicate, we have an A-clause. The set of all A-clauses in P is denoted by P_A.

For a given nonterminal A, this definition assigns a fixed arity to all A-predicates. The arity of the S-predicates (associated with the start symbol) is one.

The arity of a clause is the arity of its LHS predicate.

The arity $h \geq 1$ of a PRCG is the maximum arity of its clauses, in such a case we have an h-PRCG.

Since P, clauses and arguments may respectively be seen as sequences of clauses, predicates and strings (over $(T \cup V)^*$), they can be denoted by vector notations. $\vec{P}[i], 1 \leq i \leq l_P$ is a clause. A clause $\psi_0 \rightarrow \psi_1\psi_2 \dots \psi_m$ can be denoted by $\vec{\Psi}$ where $\vec{\Psi}[0]$ denotes the LHS predicate ψ_0, and each of $\vec{\Psi}[i], 1 \leq i \leq m$ denotes a RHS predicate ψ_i. A predicate $A(\alpha_1, \dots, \alpha_p)$ can be denoted by $A(\vec{\alpha})$ and each $\vec{\alpha}[i], 1 \leq i \leq p$ is an argument.

If $\alpha \in (T \cup V)^*$, $V^\alpha = \{X \mid \alpha = uXv \wedge X \in V\}$ is the set of its variables. If $\psi = A(\alpha_1, \dots, \alpha_p)$, is a predicate, V^ψ denotes the set $V^{\alpha_1 \dots \alpha_p}$ and if $\vec{\Psi} = \psi_0 \rightarrow \psi_1\psi_2 \dots \psi_m$ is a clause, $V^{\vec{\Psi}}$ is the set $V^{\psi_0\psi_1\dots\psi_m}$ of variables occurring in $\vec{\Psi}$.

Let $G = (N, T, V, P, S)$ be a PRCG. The CFG $G' = (N, \emptyset, P', S)$ is a *(CF) skeleton* of G iff there is a mapping between P and P' such that for each clause $A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m}) \in P$ there is a production $A_0 \rightarrow A'_1 \dots A'_m \in P'$ in which $A'_1 \dots A'_m$ is a permutation of $A_1 \dots A_m$. Of course, the CF skeletons associated with one PRCG are weakly equivalent, their languages being such that $\mathcal{L}(G') \subseteq \{\varepsilon\}$. In the sequel we will assume that the PRCGs are such that their CF skeletons are reduced[2]. Under that assumption, if $P' \neq \emptyset$, the language of any CF skeleton is not empty and contains only the empty string.

3.1 Ranges

The notion of ranges defined in this section is fundamental in PRCGs theory since their associated derivations and languages will be based upon ranges. A range is merely a couple (i, j) of non negative integers denoted by $\langle i..j \rangle$. A range is used to formalize the occurrence of a substring in a string or a couple of states in a Finite State Automaton (FSA).

For a given string $w = a_1a_2 \dots a_n \in T^*$, the set of its *ranges* is defined by $\mathcal{R}_w = \{\langle i..j \rangle \mid 0 \leq i \leq j \leq n\}$. A *range* $\langle i..j \rangle$ in $\mathcal{R}_w$ designates the triple (w_1, w_2, w_3) such that $w = w_1w_2w_3$ with $w_1 = a_1 \dots a_i$, $w_2 = a_{i+1} \dots a_j$ and $w_3 = a_{j+1} \dots a_n$. We will use several equivalent denotations for (string) ranges: an explicit dotted notation as $w_1 \bullet w_2 \bullet w_3$ or $\langle i..j \rangle_w$ or $\langle i..j \rangle$ when w is understood. The three substrings w_1, w_2 and w_3 associated with $\langle i..j \rangle$ are respectively denoted by $w^{\langle 0..i \rangle}$, $w^{\langle i..j \rangle}$ and $w^{\langle j..n \rangle}$. A range where $i = j$ is called an *empty* range. The range $\langle j-1..j \rangle$ can be denoted by $\langle j \rangle$. Therefore we have, $w^{\langle j..j \rangle} = \varepsilon$, $w^{\langle j \rangle} = a_j$ and $w^{\langle 0..n \rangle} = w$.

If $\vec{\rho}$ is a vector of ranges $\vec{\rho}[1] = \rho_1, \dots, \vec{\rho}[i] = \rho_i, \dots, \vec{\rho}[p] = \rho_p$ where $1 \leq i \leq p, \rho_i \in \mathcal{R}_w$, by definition $w^{\vec{\rho}}$ denotes the sequence (vector) of strings $w^{\rho_1}, \dots, w^{\rho_i}, \dots, w^{\rho_p}$.

For a given FSA $\mathcal{A} = (Q, T, q_0, \delta, F)$, the set of its *ranges* is defined as $\mathcal{R}_\mathcal{A} = \{\langle i..j \rangle \mid i, j \in Q \wedge \exists x \in T^*, j \in \delta^*(i, x)\}$ where δ^* is the extension from terminal symbols to strings of the transition function δ.

We can remark that a string $w = a_1a_2 \dots a_n$ is a trivial case of deterministic FSA where $Q = \{i \mid 0 \leq i \leq n\}$, $T = \{a \mid w = w_1aw_2\}$, $q_0 = 0$, $F = \{n\}$ and the deterministic transition function δ is only defined between consecutive states by $\delta(i-1, a_i) = i, 0 < i \leq n$.

This generalization from strings to FSAs can be very useful in NLP since regular languages can express different phenomena like ill-formed, incomplete or ambiguous (multi tagged/multi part of speech or word lattice) inputs. However, in the sequel, we will assume that the input is a string, but it must be clear that (most of) our results also hold with FSAs.

[1] See Section 11 for more details.

[2] A CFG is reduced if all its symbols are useful, i.e. are accessible from the start symbol and produce terminal strings.

3.2 Bindings

Variables are bound to ranges by the following mechanism of variable substitution.

Let $w = a_1 \dots a_n$ be a string in T^*, the ρ's are ranges in $\mathcal{R}_w$ and the X's are variables in V. Any couple (X, ρ) is called a *variable binding* for X denoted by X/ρ. The range ρ is the *range instantiation* of X and the string w^ρ is the *string instantiation* of X. Two variable bindings X_i/ρ_i and X_j/ρ_j are *consistent* iff $X_i = X_j \Longrightarrow \rho_i = \rho_j$. A set $\sigma = \{X_1/\rho_1, \dots, X_n/\rho_n\}$ of variable bindings is a *variable substitution* iff its elements are consistent.

This notion of binding is extended below from variables to terminals, strings, predicates and clauses.

A couple (ε, ρ) is an *empty binding* denoted ε/ρ iff $\rho = \langle j..j \rangle_w$ for some j.

A couple (a, ρ) with $a \in T$ is a *terminal binding* denoted a/ρ iff $\rho = \langle j \rangle$ and $a = a_j$.

More generally, a couple (α, ρ) with $\alpha \in (T \cup V)^*$ and $\rho \in \mathcal{R}_w$ for some w in T^* is a *string binding* iff $\alpha = x_0 X_1 x_1 \dots X_k x_k \dots X_p x_p$ with $\forall k, 0 \leq k \leq p : x_k \in T^*$ and $\forall k, 1 \leq k \leq p : X_k \in V$, and there is a variable substitution $\sigma = \{Y_1/\tau_1, \dots, Y_q/\tau_q\}$ such that $w^\rho = x_0 w^{\rho_1} x_1 \dots w^{\rho_k} x_k \dots w^{\rho_p} x_p$ with $X_1/\rho_1, \dots, X_k/\rho_k, \dots, X_p/\rho_p \in \sigma$. Such a string binding is denoted by $\alpha/_\sigma \rho$ or simply α/ρ when σ is understood or of no importance. The string w^ρ is called the (string) *instantiation* of α by ρ. A string $\alpha \in (T \cup V)^*$ is *instantiable* iff there is a string $w \in T^*$, a range $\rho \in \mathcal{R}_w$ and a variable substitution σ s.t. $\alpha/_\sigma \rho$.

It is not difficult to see that any string $\alpha = x_0 X_1 x_1 \dots X_k x_k \dots X_p x_p$ is instantiable, except when there are two different indices i and j s.t. $i < j$, $X_i = X_j$ and $x_i x_{i+1} \dots x_{j-1} \neq \varepsilon$. Therefore, in the sequel, without loss of generality, we disallow duplicate variables in individual strings (predicate arguments).

The couple $(\vec{\alpha}, \vec{\rho})$, where $\vec{\alpha} = \alpha_1, \dots, \alpha_p$ is a vector of strings and $\vec{\rho} = \rho_1, \dots, \rho_p$ is a vector of ranges is a *string vector binding* iff there is a variable substitution σ such that $\forall k, 1 \leq k \leq p$, we have $\alpha_k/_\sigma \rho_k$. A string vector binding is denoted $\vec{\alpha}/_\sigma \vec{\rho}$ or simply $\vec{\alpha}/\vec{\rho}$. In such a case, we say that $\vec{\alpha}$ is *instantiable*. The string *instantiation* of $\vec{\alpha}$ by $\vec{\rho}$ is the vector of strings $w^{\vec{\rho}} = w^{\rho_1}, \dots, w^{\rho_p}$.

A vector of instantiable strings is not always instantiable. For example, consider the two arguments (aX, bX). They are not instantiable since for any $w = a_1 \dots a_n$ and any variable binding $X/\langle i..j \rangle_w$ we should have $a_i = a$ and $a_i = b$, which is impossible. However, we can design an algorithm which checks whether a string vector is instantiable.

If $\vec{\alpha}/_\sigma \vec{\rho}$ is a string vector binding and if $A(\vec{\alpha})$ is a predicate, $A(\vec{\alpha})/_\sigma \vec{\rho}$ denotes a *predicate* binding and we say that the couple $A(\vec{\rho})$ is a (range) *instantiation* of $A(\vec{\alpha})$. In this case, the predicate $A(\vec{\alpha})$ is *instantiable*.

Last, consider a clause $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m})$ and a vector of range vectors $\vec{\Omega} = \vec{\rho_0}, \vec{\rho_1}, \dots, \vec{\rho_m}$, the couple $(\vec{\Psi}, \vec{\Omega})$ is a *clause binding* iff there is a variable substitution σ such that $\forall k, 0 \leq k \leq m$, we have $A_k(\vec{\alpha_k})/_\sigma \vec{\rho_k}$. A clause binding is denoted by $\vec{\Psi}/_\sigma \vec{\Omega}$ or more simply $\vec{\Psi}/\vec{\Omega}$. Of course we say that $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \dots A_m(\vec{\rho_m})$ is a (range) *instantiation* of $\vec{\Psi}$ by $\vec{\Omega}$ and that $\vec{\Psi}$ is *instantiable*.

In the sequel we will only consider *instantiable* clauses.

3.3 Derivations, Derivation Trees & Shared Forests

3.3.1 The CF Case

Let $G = (N, T, P, S)$ be a CFG, *derive* is a binary relation on $(T \cup N)^*$, denoted by $\underset{G}{\Rightarrow}$, defined by

$$\Gamma_1 \, A \, \Gamma_2 \underset{G}{\Rightarrow} \Gamma_1 \, \alpha \, \Gamma_2$$

where Γ_1 and Γ_2 are elements of $(T \cup N)^*$ and $A \rightarrow \alpha$ is a production in P.

Let $d = (\Gamma_0, \Gamma_1, \dots, \Gamma_l)$ be a sequence of strings in $(T \cup N)^*$ such that $\forall i, 1 \leq i \leq l : \Gamma_{i-1} \underset{G}{\Rightarrow} \Gamma_i$, d is called a *derivation* of length l, Γ_0 is the *head* and Γ_l is the *word*. A derivation can also be denoted by $(\Gamma_0 \underset{G}{\Rightarrow} \Gamma_1 \overset{*}{\underset{G}{\Rightarrow}} \Gamma_l)$. Each couple (Γ_{i-1}, Γ_i) is a *derivation step*. A Γ-*derivation* is a derivation whose head is Γ. A Γ_1-derivation whose word is Γ_2 is a Γ_1/Γ_2-*derivation*. The elements of a Γ-derivation are called Γ-*phrases*. Any S-phrase is a *sentential form*. A *sentence* is a sentential form in T^*. A derivation whose word is a sequence in T^* is a *terminal or closed* derivation.

If we consider the derivation as a rewriting process, at each step, the choice of the nonterminal to be derived does not depend of its neighbors (the derivation process is context-free). We can talk of *leftmost* (resp. *rightmost*) derivations when at each step the leftmost (resp. rightmost) nonterminal is substituted. Of course many other derivation strategies may be thought of. All theses strategies can be captured in a single canonical tree structure which abstracts all possible orders and called *derivation tree* (or *parse tree*). For any given A-derivation, we can associate a single derivation tree whose root is labeled A. Conversely, if we consider a

derivation tree, there may have several associated derivations which depend upon the way the tree is walked (for example a top-down left-to-right walk leads to a leftmost derivation).

Remark that from a derivation step (Γ, Γ'), it is not always possible to determine which nonterminal occurrence in Γ has been derived (consider the derivation step $A\ A \underset{G}{\Rightarrow} A$ with $A \to \varepsilon \in P$). However, if the occurrence is known, the production used can be determined. This will not be the case with PRCGs (see below).

If we consider a S/x-derivation say $d = (S \overset{*}{\underset{G}{\Rightarrow}} \Gamma_1\ A\ \Gamma_3 \overset{+}{\underset{G}{\Rightarrow}} x)$, the nonterminal occurrence of A in $\Gamma_1\ A\ \Gamma_3$ is responsible for the generation of the portion x_2 of x, independently of Γ_1 and Γ_3 which are respectively responsible for the production of x_1 and x_3 if we assume that $x = x_1x_2x_3$. In a S/x-derivation, we denote by A^ρ, where the range $\rho = x_1 \bullet x_2 \bullet x_3$, the fact that (some occurrence of) A is responsible for the generation of x_2. Therefore, to each S/x-derivation d, we can associate a new sequence in $(T \cup (N \times \mathcal{R}_x))^*$ denoted by $\langle d \rangle$ in which each occurrence of a nonterminal A is replaced by some A^ρ: $\langle d \rangle = (S^{\bullet x \bullet} \overset{*}{\underset{G}{\Rightarrow}} \langle \Gamma_1 \rangle A^{x_1 \bullet x_2 \bullet x_3} \langle \Gamma_3 \rangle \overset{*}{\underset{G}{\Rightarrow}} x_1x_2x_3)$ where $\langle \Gamma_1 \rangle$ and $\langle \Gamma_3 \rangle$ are strings produced from Γ_1 and Γ_3 in which each nonterminal occurrence is associated with its corresponding range.

Let's denote $\mathcal{D}_x$ the set of all S/x-derivations for a given $x = a_1 \dots a_n \in T^*$ and a CFG $G = (N, T, P, S)$, and $\langle \mathcal{D}_x \rangle = \{\langle d \rangle \mid d \in \mathcal{D}_x\}$ the corresponding set. We construct a new CFG $G_x = (N \times \mathcal{R}_x, T, P_x, S^{\bullet x \bullet})$ whose production set is defined by

$$\begin{aligned} P_x \quad = \quad & \{A_0^{\rho_0} \to y_0 A_1^{\rho_1} \dots A_m^{\rho_m} y_m \mid A_0 \to y_0 A_1 \dots A_m y_m \in P \wedge \\ & (S^{\bullet x \bullet} \overset{*}{\underset{G}{\Rightarrow}} \langle \Gamma_1 \rangle A_0^{\rho_0} \langle \Gamma_3 \rangle, \langle \Gamma_1 \rangle y_0 A_1^{\rho_1} \dots A_m^{\rho_m} y_m \langle \Gamma_3 \rangle \overset{*}{\underset{G}{\Rightarrow}} x) \in \langle \mathcal{D}_x \rangle\} \end{aligned}$$

This CFG is called a *shared forest*. This is the structure which is constructed, implicitly or explicitly, by any general CF parsing algorithm.

This shared forest has some nice properties among which we can note

- This shared forest is the intersection of a CFG (the grammar G) and a FSA (the trivial FSA defined in Section 3.1 whose language is $\{x\}$). It can be built in cubic time in n and its size is also cubic if P is in binary form.
- $x \in \mathcal{L}(G) \iff \{x\} = \mathcal{L}(G_x)$.
- $\langle \mathcal{D}_x \rangle$ is in fact the set of all $S^{\bullet x \bullet}/x$-derivations in G_x. Moreover, G_x may be viewed as a polynomial size representation of the (unbounded) set of derivation trees in G for x.

In fact, shared forests seem to be the right structure when we consider the output of a CF parser, both at a formal or at a practical level.

Below, we generalize this notion of shared forests to PRCGs.

3.3.2 The PRCG Case

Definition 2 *For a given PRCG $G = (N, T, V, P, S)$, and a string w in T^*, we define on strings of instantiated predicates[3] a binary relation named derive, denoted $\underset{G,w}{\Rightarrow}$ by*

$$\Gamma_1\ A_0(\vec{\rho_0})\ \Gamma_2 \underset{G,w}{\Rightarrow} \Gamma_1\ A_1(\vec{\rho_1}) \dots A_m(\vec{\rho_m})\ \Gamma_2$$

where Γ_1 and Γ_2 are sequences of instantiated predicates, $\vec{\Psi} = A_0(\vec{\alpha_0}) \to A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m})$ is a clause in P, $\vec{\Omega} = (\vec{\rho_0}, \vec{\rho_1}, \dots, \vec{\rho_m})$ is such that $\vec{\Psi}/_\sigma\Omega$ is a clause binding for some variable binding σ and $A_0(\vec{\rho_0}) \to A_1(\vec{\rho_1}) \dots A_m(\vec{\rho_m})$ is the range instantiation of $\vec{\Psi}$ by $\vec{\Omega}$.

Let $d = (\Gamma_0, \Gamma_1, \dots, \Gamma_l)$ be a sequence of strings of instantiated predicates such that $\forall i, 1 \leq i \leq l : \Gamma_{i-1} \underset{G,w}{\Rightarrow} \Gamma_i$ for some $w \in T^*$, d is called a *derivation* of length l, Γ_0 is the *head* and Γ_l is the *word*. Each couple (Γ_{i-1}, Γ_i) is a *derivation step*. A Γ-*derivation* is a derivation whose head is Γ. A Γ_1-derivation whose word is Γ_2 is a Γ_1/Γ_2-*derivation*. The elements of a Γ-derivation are called Γ-*phrases*. Any $S(\bullet w \bullet)$-phrase is a *sentential form*. A derivation whose word is the empty sequence ε is a *terminal or closed* derivation. A $S(\bullet w \bullet)/\varepsilon$-derivation is *complete*. A string $w \in T^*$ is a *sentence* of G iff there is a (complete) $S(\bullet w \bullet)/\varepsilon$-derivation. Remark that in PRCGs a sentence is not a particular sentential form.

[3] The ranges are elements of $\mathcal{R}_w$.

Definition 3 *The string language of a PRCG $G = (N, T, V, P, S)$ is the set of its sentences*

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

As in the CF case, if we consider the derivation as a rewriting process, at each step, the choice of the (instantiated) predicate to be derived does not depend of its neighbors (the derivation process is context-free). We can talk of *leftmost* (resp. *rightmost*) derivations when at each step the leftmost (resp. rightmost) predicate is substituted. Of course many other derivation strategies may be thought of. All these strategies can be captured in a single canonical tree structure which abstracts all possible orders and which is called *derivation tree* (or *parse tree*). For any given $A(\vec{\rho})$-derivation, we can associate a single derivation tree whose root is labeled $A(\vec{\rho})$. Conversely, if we consider a derivation tree, there may have several associated derivations which depend upon the way the tree is walked (for example a top-down left-to-right walk leads to a leftmost derivation).

Remark that from a derivation step (Γ, Γ'), it is not always possible to determine which predicate occurrence in Γ has been derived. Moreover, even if the occurrence is known, the clause $\vec{\Psi}$ used cannot be determined in the general case. This is due to the fact that $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \ldots A_m(\vec{\rho_m})$ may be the instantiation of different clauses $\vec{\Psi}_1$, $\vec{\Psi}_2$, ... by the clause bindings $\vec{\Psi}_1/\vec{\Omega}$, $\vec{\Psi}_2/\vec{\Omega}$, ... where $\vec{\Omega} = (\vec{\rho_0}, \vec{\rho_1}, \ldots, \vec{\rho_m})$. But, of course, each of these interpretations is a valid one.

Consider the set $\mathcal{D}_w$ of all $S(\bullet w \bullet)/\varepsilon$-derivations for a given $w = a_1 \ldots a_n \in T^*$ and a k-PRCG $G = (N, T, V, P, S)$. We define a CFG $G_w = (N \times \mathcal{R}_w^k, \emptyset, P_w, S^{\bullet w \bullet})$ whose production set is

$$P_w \quad = \quad \{A_0^{\vec{\rho_0}} \rightarrow A_1^{\vec{\rho_1}} \ldots A_m^{\vec{\rho_m}} \mid S^{\bullet w \bullet} \underset{G,w}{\overset{*}{\Rightarrow}} \Gamma_1 \; A_0(\vec{\rho_0})\, \Gamma_3 \underset{G,w}{\Rightarrow} \Gamma_1 \; A_1(\vec{\rho_1}) \ldots A_m(\vec{\rho_m})\, \Gamma_3 \underset{G,w}{\overset{*}{\Rightarrow}} \varepsilon \in \mathcal{D}_w\}$$

This CFG is called the *shared forest* for w w.r.t. G.

Note that, opposite to the CF case, if $\mathcal{D}_w$ is not empty, the language of a shared forest is not $\{w\}$ but is $\{\varepsilon\}$.

We shall see in Section 13 how this shared forest can be constructed by our parsing algorithm in polynomial time.

Moreover, this shared forest of polynomial size, may be viewed as an exact packed representation of all the (unbounded number of) derivation (parse) trees in G for some w: the set of parse trees for (G, w) and the set of parse trees of its shared forest G_w are identical[4].

Definition 4 *A PRCG is ambiguous if there is at least one sentence for which there is more than one derivation tree.*

3.3.3 Generalization to other cases

Lang generalizes the notion of shared forest from CFG to other families of formalisms. Quoted from [Lang 94] *[the purpose is,] given a family Φ of grammatical formalisms, to derive the construction of dynamic programming parsers for Φ from a constructive proof that the family Φ is closed under intersection with regular sets. More precisely, given a Φ-grammar $\mathcal{G}$ and a finite state automaton $\mathcal{A}$, if we can construct from them a new Φ-grammar $\mathcal{F}$ for the intersection $\mathcal{L}(\mathcal{G}) \cap \mathcal{L}(\mathcal{A})$, we define this new grammar $\mathcal{F}$ to be the shared forest for all parses of the sentences in the intersection.* Afterwards, he successfully applies this vision to TAG parsing, and he claims that *this approach extends nicely to such formalisms as ... Linear Indexed Grammars ....* In [Boullier 95], we construct a *LIGed forest* which is exactly a shared forest à la Lang (it is merely the shared forest of the CF backbone of the initial LIG, decorated with its stack schemas). But doing so (in time $\mathcal{O}(n^3)$), we are far from getting a LIG parser or even a recognizer. The LIG constraints still have to be checked. In [Boullier 96] we propose a LIG parser whose output is a CFG (built in time $\mathcal{O}(n^6)$), whose language is the set of valid LIG derivations from which each derivation can be extracted in linear time. As a caricature, if we apply Lang's vision to RCG, their parsing is for free: let S_1 be the start symbol of a RCG G_1, $w = a_1 \ldots a_n$ be an input string and the output of a RCG parser for w is the RCG whose set of clauses is the set of clauses of G_1 to which the two clauses $S_2(w) \rightarrow \varepsilon$ and $S(X) \rightarrow S_1(X)\ \ S_2(X)$ have been added.

Though the fact that the input and the output of a parser are expressed in the same formalism seems to be a conceptual appealing feature, we part from this approach and propose for shared forests a much more practical vision. What we really want as an output for a parser is a *packed structure from which individual derivation (parse) tree can easily be extracted.* To be more precise, by packed, we mean a structure whose size is polynomial in the length of the input string, even when the number of individual derivation trees is unbounded. By easily extracted, we mean a linear time extraction in the size of the extracted tree.

[4] Of course we assume that $A(\vec{\rho})$ and $\vec{\rho}$ denote identical couples.

In that sense, the Linear Derivation Grammars of [Boullier 96] are shared forests for LIGs.

Of course, polynomial size CFGs are ideal candidates for being shared forests. This is the case of RCGs, as we already have seen in the previous section.

3.4 Positive Range Concatenation Languages (PRCLs)

In Definition 3 we have defined the string language $\mathcal{L}$ of a PRCG which covers the usual meaning of language defined by a grammar. Here we will define another type of language associated with nonterminals of a PRCG, namely the *range language* denoted by Λ.

For a given $w \in T^*$, the elements of a range language are range vectors $\vec{\rho} \in \mathcal{R}_w^*$ and the elements of a string language are string vectors $w^{\vec{\rho}}, \vec{\rho} \in \mathcal{R}_w^*$. Between any range language Λ and the corresponding string language $\mathcal{L}$ we have the following property:

$$\forall \vec{\rho} \in \mathcal{R}_w^* \quad \vec{\rho} \in \Lambda \implies w^{\vec{\rho}} \in \mathcal{L}$$

.

Definition 5 *Let $G = (N, T, V, P, S)$ be a PRCG.*

- *The range language of a nonterminal A for some $w \in T^*$ is*

$$\Lambda(A, w) = \{\vec{\rho} \mid \vec{\rho} \in \mathcal{R}_w^* \wedge A(\vec{\rho}) \underset{G,w}{\overset{+}{\Rightarrow}} \varepsilon\}$$

- *The (string) language of a nonterminal A for some $w \in T^*$ is*

$$\mathcal{L}(A, w) = \{w^{\vec{\rho}} \mid \vec{\rho} \in \Lambda(A, w)\}$$

- *The range language of a nonterminal A is*

$$\Lambda(A) = \cup_{w \in T^*} \Lambda(A, w)$$

- *The (string) language of a nonterminal A is*

$$\mathcal{L}(A) = \cup_{w \in T^*} \mathcal{L}(A, w) = \{w^{\vec{\rho}} \mid \vec{\rho} \in \Lambda(A)\}$$

- *The range language defined by G is*

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

- *The (string) language[5] defined by G is*

$$\mathcal{L}(G) = \{w \mid \bullet w \bullet \in \Lambda(G)\}$$

We must note that, with these definitions, the language of a grammar is not the language of its start symbol, we have

$$\begin{array}{rcl} \Lambda(G) & \subseteq & \Lambda(S) \\ \mathcal{L}(G) & \subseteq & \mathcal{L}(S) \end{array}$$

since $\Lambda(S) = \{w_1 \bullet w_2 \bullet w_3 \mid S(w_1 \bullet w_2 \bullet w_3) \underset{G,w_1 w_2 w_3}{\overset{+}{\Rightarrow}} \varepsilon\}$ and $\mathcal{L}(S) = \{w_2 \mid w_1 \bullet w_2 \bullet w_3 \in \Lambda(S)\}$.

[5] Of course, this is equivalent with Definition 3.

Example 1 *Consider the PRCG with the following set of clauses*

$$\begin{array}{lrcl} c_1: & S(XcY) & \rightarrow & A(X,Y) \\ c_2: & A(\varepsilon,\varepsilon) & \rightarrow & \varepsilon \\ c_3: & A(Xa,Ya) & \rightarrow & A(X,Y) \\ c_4: & A(Xb,Yb) & \rightarrow & A(X,Y) \\ c_5: & A(Xc,Yc) & \rightarrow & A(X,Y) \end{array}$$

We check that the string $w = abcab$ is a sentence

$$\begin{array}{rcl} S(\bullet abcab\bullet) & \underset{G,w}{\overset{c_1}{\Rightarrow}} & A(\bullet ab \bullet cab, abc \bullet ab\bullet) \\ & \underset{G,w}{\overset{c_4}{\Rightarrow}} & A(\bullet a \bullet bcab, abc \bullet a \bullet b) \\ & \underset{G,w}{\overset{c_3}{\Rightarrow}} & A(\bullet \bullet abcab, abc \bullet \bullet ab) \\ & \underset{G,w}{\overset{c_2}{\Rightarrow}} & \varepsilon \end{array}$$

The clause used at each derivation step is stacked over the corresponding derived symbol.

One can easily see that at each step the ranges ρ_1 and ρ_2 of the predicate A are such that $w^{\rho_1} = w^{\rho_2}$ so the language is

$$\mathcal{L} = \{xcx \mid x \in \{a,b,c\}^*\}$$

which is not CF.

Example 2 *Another way to define the language of the Example 1 is with the following PRCG:*

$$\begin{array}{lcl} S(XcY) & \rightarrow & L(X)\ Eq(X,Y) \\ L(\varepsilon) & \rightarrow & \varepsilon \\ L(Xa) & \rightarrow & L(X) \\ L(Xb) & \rightarrow & L(X) \\ L(Xc) & \rightarrow & L(X) \end{array}$$

where the equality predicate Eq is defined by the two clauses

$$\begin{array}{lcl} Eq(Xt,Yt) & \rightarrow & Eq(X,Y) \\ Eq(\varepsilon,\varepsilon) & \rightarrow & \varepsilon \end{array}$$

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

We see that Eq acts as a generic predicate which can be used in any grammar when we need to check the equality of two strings. See Section 12 for a generalization of this grammar definition process.

Example 3 *The power of this formalism is shown by the next grammar whose sentences do not express a "constant growth property"*

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

$\mathcal{L} = \{a^{2^p} \mid p \geq 0\}$

4 (Negative) Range Concatenation Grammars

In this section we define the general notion of range concatenation grammar (RCG), which is an extension of PRCGs in which the predicates occurring in RHS of clauses may be "negative" (denoted by an overlined nonterminal symbol) with the intuitive meaning of negation or complementation. The term negative range concatenation grammar (NRCG) will be used to enforce the presence of negative predicate.

Definition 6 *A range concatenation grammar (RCG) $G = (N,T,V,P,S)$ is a PRCG except that predicate symbols in RHS of clauses are in the set $N \cup \overline{N}$ where $\overline{N} = \{\overline{A} \mid A \in N\}$.*

Therefore we may talk of positive or negative nonterminals (predicates), $N \cup \overline{N}$ is the set of *extended* nonterminals (predicates).

Let $G = (N, T, V, P, S)$ be a RCG. The PRCG $G^+ = (N, T, V, P^+, S)$ associated with G is such that there is a mapping from P to P^+ s.t. the negative nonterminals in a clause (if any) are transformed into their positive counterpart ($P^+ = \{A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m}) \mid A_0(\vec{\alpha_0}) \rightarrow e_1(\vec{\alpha_1}) \dots e_m(\vec{\alpha_m}) \in P \wedge (\forall i, 1 \leq i \leq m :$ if $e_i \in N$ then $A_i = e_i \vee$ if $e_i \in \overline{N}$ then $A_i = \overline{e_i})\}$)[6]. As usual we will assume that the CF skeleton of the PRCG associated with a given RCG is reduced.

In order to define the language of such a grammar, we first extend the derive relation.

The relation derive holds between strings whose elements are instantiated extended predicates.

For a given string $w \in T^*$, let D_0 be the set of couples $(\Gamma_1 \ A_0(\vec{\rho_0}) \ \Gamma_2, \Gamma_1 \ A_1(\vec{\rho_1}) \dots A_m(\vec{\rho_m}) \ \Gamma_2)$ where Γ_1 and Γ_2 are sequences of instantiated extended predicates and such that there is an A_0-clause $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m}) \in P$ and a binding $\vec{\Psi}/\vec{\Omega}$ with $\vec{\rho_i} = \vec{\Omega}[i], 0 \leq i \leq m$.

The (extended) derive relation, also denoted $\underset{G,w}{\Rightarrow}$, is defined as the smallest set which verifies the equation

$$\underset{G,w}{\Rightarrow} \quad = \quad D_0 \cup \{(\Gamma_1 \ \overline{A_0}(\vec{\rho_0}) \ \Gamma_2, \Gamma_1 \ \Gamma_2) \mid (A_0(\vec{\rho_0}), \varepsilon) \notin \overset{+}{\underset{G,w}{\Rightarrow}}\}$$

Intuitively, negative instantiated predicates such as $\overline{A_0}(\vec{\rho_0})$ disappear within derivations, when $\vec{\rho_0}$ is not in the range language of A_0.

Of course, this definition is the Definition 2 when we only consider instantiated positive predicates. However, some inconsistency may occur if we want to keep the view that a negative predicate represents the complement of its positive counterpart. A derive relation in which we have both $A(\vec{\rho}) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$ and $\overline{A}(\vec{\rho}) \underset{G,w}{\Rightarrow} \varepsilon$ is *inconsistent* (otherwise *consistent*).

Example 4 *Consider the RCG G which contains the two clauses*

$$\begin{array}{lcl} S(a) & \rightarrow & \varepsilon \\ S(X) & \rightarrow & \overline{S}(X) \end{array}$$

We have $D_0 = \{(\Gamma_1 S(\rho_1)\Gamma_2, \Gamma_1\Gamma_2) \mid \rho_1 \in \mathcal{R}_w \wedge w^{\rho_1} = a\} \cup \{(\Gamma_1 S(\rho_2)\Gamma_2, \Gamma_1\overline{S}(\rho_2)\Gamma_2) \mid \rho_2 \in \mathcal{R}_w\}$, for some $w \in T^*$, where Γ_1 and Γ_2 are strings in $(N \times \mathcal{R}_w^+)^*$. Therefore, if $\rho_3 \neq \rho_1$ we have $(\Gamma_1\overline{S}(\rho_3)\Gamma_2, \Gamma_1\Gamma_2) \in \underset{G,w}{\Rightarrow}$. This shows that the relation $\underset{G,w}{\Rightarrow}$ is inconsistent since if $\rho_3 \neq \rho_1$, we have both $S(\rho_3) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$ and $\overline{S}(\rho_3) \underset{G,w}{\Rightarrow} \varepsilon$.

In the sequel we will proscribe inconsistent derive relations. If the CF skeleton of a RCG is such that $A \overset{+}{\Rightarrow} \Gamma_1 \ \overline{A} \ \Gamma_2$ (a nonterminal is defined in term of its own complement), we can easily see that the derive relations (for all $w \in T^*$) are consistent. Under this assumption, for a derive relation, a nonterminal A and a vector of ranges $\vec{\rho}$ we have either $A(\vec{\rho}) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$ or $\overline{A}(\vec{\rho}) \underset{G,w}{\Rightarrow} \varepsilon$.

Remark that, the structure (derivation trees) of negative components is empty. This property is not in contradiction with our purpose to use negative predicates to describe the exceptions of a general rule. Assume that a property P is defined by a general rule (the predicate R) with some exceptions to that rule (described by E). The clause $P(\vec{\alpha}) \rightarrow R(\vec{\alpha}) \ \overline{E}(\vec{\alpha})$ describes such a specification. It is clear that the structure of the elements of E (which therefore are not in P) has no interest. Conversely, the structure of the elements of P (which are in R and not in E), is the structure of the elements of R.

Definition 7 *Let $G = (N, T, V, P, S)$ be a RCG.*

- *The range language of a nonterminal A for some $w \in T^*$ is*

$$\Lambda(A, w) = \{\vec{\rho} \mid \vec{\rho} \in \mathcal{R}_w^* \wedge A(\vec{\rho}) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon\}$$

- *The range language of a negative nonterminal $\overline{A}$ for some $w \in T^*$ is*

$$\Lambda(\overline{A}, w) = \{\vec{\rho} \mid \vec{\rho} \in \mathcal{R}_w^+ \wedge \vec{\rho} \notin \Lambda(A, w)\} = \{\vec{\rho} \mid \vec{\rho} \in \mathcal{R}_w^+ \wedge \overline{A}(\vec{\rho}) \underset{G,w}{\Rightarrow} \varepsilon\}$$

[6] Of course we have $\overline{\overline{A}} = A$.

- *The (string) language of a nonterminal A for some $w \in T^*$ is*

$$\mathcal{L}(A, w) = \{w^{\vec{\rho}} \mid \vec{\rho} \in \Lambda(A, w)\}$$

- *The (string) language of a negative nonterminal $\overline{A}$ for some $w \in T^*$ is*

$$\mathcal{L}(\overline{A}, w) = \{w^{\vec{\rho}} \mid \vec{\rho} \in \Lambda(\overline{A}, w)\}$$

- *The range language of an extended nonterminal $e \in N \cup \overline{N}$ is*

$$\Lambda(e) = \cup_{w \in T^*} \Lambda(e, w)$$

- *The (string) language of an extended nonterminal $e \in N \cup \overline{N}$ is*

$$\mathcal{L}(e) = \cup_{w \in T^*} \mathcal{L}(e, w) = \{w^{\vec{\rho}} \mid \rho \in \Lambda(e)\}$$

- *The range language defined by G is*

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

- *The (string) language defined by G is*

$$\mathcal{L}(G) = \{w \mid \bullet w \bullet \in \Lambda(G)\}$$

Borrowed from [Groenink 97], we define the following.

Definition 8 *Let $G = (N, T, V, P, S)$ be a RCG and let $\vec{\Psi} \in P$ be one of its clause:*

$$\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m})$$

- *$\vec{\Psi}$ is non-combinatorial if each of the $\vec{\alpha_j}[k], 1 \leq j \leq m$ consists of a single variable.*
- *$\vec{\Psi}$ is bottom-up linear if no variable appears more than once in $\vec{\alpha_0}$.*
- *$\vec{\Psi}$ is top-down linear if no variable appears more than once in $\vec{\alpha_1}, \ldots, \vec{\alpha_m}$.*
- *$\vec{\Psi}$ is bottom-up non-erasing if each variable occurring in an $\vec{\alpha_1}, \ldots, \vec{\alpha_m}$, also occurs in $\vec{\alpha_0}$.*
- *$\vec{\Psi}$ is top-down non-erasing if each variable occurring in $\vec{\alpha_0}$, also appears in $\vec{\alpha_1}, \ldots, \vec{\alpha_m}$.*
- *$\vec{\Psi}$ is linear (resp. non-erasing) if it is both bottom-up and top-down linear (resp. non-erasing).*

These definitions extend from clause to set of clauses.

Property 1 *For any RCG, there is an equivalent non-combinatorial RCG.*

Proof: Let $G = (N, T, V, P, S)$ be a combinatorial RCG and $G' = (N', T, V', P', S')$ the non-combinatorial RCG construct in the following way. We assume that each time a new nonterminal or a new variable appears in some clause, it is added to the set N' or V'.

We assume that W is a variable not in V which is bound, throughout G' to the range $\bullet w \bullet$ for some $w \in T^*$.

P' is initialized with $S'(W) \rightarrow S(W, W)$.

For each clause $c = A_0(\alpha_0^1, \ldots, \alpha_0^{l_0}) \rightarrow A_1(\alpha_1^1, \ldots, \alpha_1^{l_1}) \ldots A_m(\alpha_m^1, \ldots, \alpha_m^{l_m})$ in P

- If c is non-combinatorial, $A_0(W, \alpha_0^1, \ldots, \alpha_0^{l_0}) \rightarrow A_1(W, \alpha_1^1, \ldots, \alpha_1^{l_1}) \ldots A_m(W, \alpha_m^1, \ldots, \alpha_m^{l_m})$ is added to P'.
- If c is combinatorial, we add three new clauses to P'

1. $A_0(W, X_0^1, \ldots, X_0^{l_0}) \rightarrow c(W, X_0^1, \ldots, X_0^{l_0}, W, \ldots, W)$ where the arity of c (c is a nonterminal for G') is $1 + l_0 + l_1 + \ldots + l_m$.
2. $c(W, X_0^1, \ldots, X_0^{l_0}, Y_1^1 X_1^1 Z_1^1, \ldots, Y_1^{l_1} X_1^{l_1} Z_1^{l_1}, \ldots, Y_m^1 X_m^1 Z_m^1, \ldots, Y_m^{l_m} X_m^{l_m} Z_m^{l_m}) \rightarrow c'(X_0^1, \ldots, X_0^{l_0}, X_1^1, \ldots, X_1^{l_1}, \ldots, X_m^1, \ldots, X_m^{l_m})\ A_1(W, X_1^1, \ldots, X_1^{l_1}) \ldots A_m(W, X_m^1, \ldots, X_m^{l_m})$ where the arity of c' is $l_0 + l_1 + \ldots + l_m$ and the arity of each $A_i, 1 \leq i \leq m$ is $1 + l_i$.
3. $c'(\alpha_0^1, \ldots, \alpha_0^{l_0}, \alpha_1^1, \ldots, \alpha_1^{l_1}, \ldots, \alpha_m^1, \ldots, \alpha_m^{l_m}) \rightarrow \varepsilon$

It is not difficult to see that G and G' are equivalent.

∎

Note that G' is not top-down linear.

Property 2 *For any non-combinatorial bottom-up erasing RCG G, there is an equivalent non-combinatorial bottom-up non-erasing RCG G'.*

We will only show how we can force a single variable say Y_2 in the RHS of a bottom-up erasing clause to appear in the LHS of this clause. This method easily generalizes to any number of such variables. Assume we have in G a clause $A_0(\vec{\alpha_0}) \rightarrow \ldots A_i(\vec{\alpha_i}) \ldots$ with $Y_2 \notin V^{\vec{\alpha_0}}$ and $Y_2 \in V^{\vec{\alpha_i}}$. We construct a new clause where the predicate A_0 has two more arguments, the first one and the second one, which are both bound to the range $\bullet w \bullet$: $A_0(W, Y_1 Y_2 Y_3, \vec{\alpha_0}) \rightarrow \ldots A_i(\vec{\alpha_i}) \ldots$. In fact the additional number of arguments for that clause is the number of erased variables from the RHS plus one. Of course, the arity of each predicate must be consistent throughout G' (use the variable W, as much as necessary). At last we only have to add the clause $S'(W) \rightarrow S(W, \ldots, W)$, with the right new arity for S.

Note that G' is not top-down linear and top-down erasing.

Property 3 *For any non-combinatorial bottom-up non-erasing top-down erasing RCG G, there is an equivalent non-combinatorial non-erasing RCG G'.*

Each clause $A_0(\vec{\alpha_0}) \rightarrow \Psi$ in G with a variable Y s.t. $Y \in V^{\vec{\alpha_0}}$ and $Y \notin V^{\Psi}$ is changed to the new clause in G' $A_0(\vec{\alpha_0}) \rightarrow \Psi\ Any(Y)$ where Any is a predicate defined by

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

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

Therefore, in the sequel, we may assume at will that any RCG at hand is non-combinatorial and non-erasing.

However, these equivalence properties do not hold for linearity (i.e. in the general case, there is no linear grammar equivalent to a non-linear one). In other words, non-linearity brings some formal power to the formalism. In particular, we shall see in Section 12 that RCGs are closed by intersection is due to its non-linearity.

We can show that if a non-combinatorial, non-erasing top-down linear RCG is reduced and ε-free, then this grammar is (bottom-up) linear. This is due to the fact that under those assumptions, the ranges are not empty and non-overlapping. Therefore, a useful A_0-clause cannot have two occurrences of the same variable in two arguments of the LHS predicate A_0.

Note that the clauses

$$\begin{array}{lcl} A(X, X) & \rightarrow & B(X) \\ A(X, Y) & \rightarrow & B(X)\ Eq(X, Y) \end{array}$$

and

$$\begin{array}{lcl} A(X) & \rightarrow & B(X)\ C(X) \\ A(X) & \rightarrow & B(X)\ C(Y)\ Eq(X, Y) \end{array}$$

are not equivalent.

5 Simple (Positive) Range Concatenation Grammars

Definition 9 *We say that a RCG is simple (sRCG) if it is linear, non-erasing and non-combinatorial*[7] *.*

In other words, the arguments in the RHS of any clause are different variables and all these variables (and no others) must occur exactly one time in the LHS arguments.

In this section we shall see some algorithms and properties of simple PRCGs (sPRCGs). We emphasize this particular form of RCG both for its simplicity and because there is a great variety of usual syntactic formalisms (see Sections 7, 8, 9 and 10) which can be transformed into an equivalent sPRCG.

Definition 10 *We say that an A-clause $\vec{\Psi}$ is productive (for some RCG G) iff there is for some $w \in T^*$ and for some range vector $\vec{\rho} \in \mathcal{R}_w^*$ a terminal derivation headed at $A(\vec{\rho})$ whose first step uses an instantiation of $\vec{\Psi}$. We say that a nonterminal A is productive iff there is a productive A-clause.*

Let $G = (N, T, V, P, S)$ be a RCG. We assume that G is in non-combinatorial form. The following algorithm computes an array named Pr in examining at turn each clause in P which is supposed to have the form $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m})$.

1. Initially, $Pr[\vec{\Psi}]$ is set to *false* for all $\vec{\Psi} \in P$.
2. level := 0. If $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow \varepsilon$ then set $Pr[\vec{\Psi}]$ to *true*.
3. level := level+1. If for each $A_j, 1 \leq j \leq m$ there is an A_j-clause say c such that $Pr[c]$ is *true*, then set $Pr[\vec{\Psi}]$ to *true*.
4. Step 3) is iterated until stability.

This algorithm terminates since we monotonically iterate over finite sets.

Property 4 *If G is a sPRCG, $\vec{\Psi}$ is productive iff $Pr[\vec{\Psi}]$ is set to true by the previous algorithm.*

Proof: Assume that $Pr[\vec{\Psi}] = true$. We are going to show by induction on the level associated with $Pr[\vec{\Psi}]$ that there is a terminal derivation $A_0(\vec{\rho_0}) \underset{G,w}{\overset{+}{\Rightarrow}} \varepsilon$ if $\vec{\Psi}$ is an A_0-clause.

basis. If this assignment has been set at level 0, there is a terminal derivation of length 1, using the clause binding $\vec{\Psi}/\vec{\rho_0}$ for some $\vec{\rho_0}$ (G is instantiable).

Induction step. Assume that all clauses associated with a level $0, 1, \dots, l-1$ are productive and consider a clause $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m})$ associated with level l (if any) such that $Pr(\vec{\Psi}) = true$. We know (see step 3), that there are m A_j-clauses $\vec{\Psi}_1, \dots, \vec{\Psi_m}$ s.t. $Pr[\vec{\Psi_j}] = true, \forall j, 1 \leq j \leq m$ and whose associated level is less than l. Therefore, by the induction hypothesis, there are m strings w_j and m terminal $A_j(\vec{\rho_j})$-derivations, using the derive relation $\underset{G,w_j}{\Rightarrow}$. In order to construct a derivation whose first step uses an instantiation of the clause $\vec{\Psi}$ and whose further sub-derivations are made from the m terminal $A_j(\vec{\rho_j})$-derivations, we have to construct a single input string say w, since in a derivation all the ranges must be elements of a single $\mathcal{R}_w$. We assume that the k^{th} argument of the j^{th} predicate in the RHS of $\vec{\Psi}$ (i.e. $\vec{\alpha_j}[k]$) is the variable X_j^k. The new string w is the concatenation from left to right of the arguments $\vec{\alpha_0}$ of the LHS predicate A_0 in which each variable, say Y, is substituted by the substring $w_j^{\vec{\rho_j}[k]}$ of w_j if $Y = X_j^k$ and terminal symbols are left as such given $w = z_1 \dots z_{l_{\vec{\alpha_0}}}$ where each z_i is the participation of $\vec{\alpha_0}[i]$ in w. Therefore, $\forall j, 1 \leq j \leq m, \forall k, 1 \leq k \leq l_{\vec{\alpha}_j} : \exists w_1, w_2$ s.t. $w = w_1 w_j^{\vec{\rho_j}[k]} w_3$ and if $(j,k) \neq (j',k')$ the substrings $w_j^{\vec{\rho_j}[k]}$ and $w_{j'}^{\vec{\rho_j}[k']}$ are non-overlapping in w. Assume that $w_j = x_1 y_1 y_2 y_3 x_3$ and that the range $\vec{\rho_j}[k]$ is $x_1 \bullet y_1 y_2 y_3 \bullet x_3$. Any subrange $\rho = x_1 y_1 \bullet y_2 \bullet y_3 x_3$ of $\vec{\rho_j}[k]$ is changed into the range $w_1 y_1 \bullet y_2 \bullet y_3 w_3$ of $\mathcal{R}_w$. In particular, the ranges such as $\vec{\rho_j}[k]$ are changed into $w_1 \bullet w_j^{\vec{\rho_j}[k]} \bullet w_3$. Let $\vec{\rho_0} = (\bullet z_1 \bullet \dots z_{l_{\vec{\alpha_0}}}, \dots, z_1 \dots \bullet z_i \bullet \dots z_{l_{\vec{\alpha_0}}}, \dots, z_1 \dots \bullet z_{l_{\vec{\alpha_0}}} \bullet)$ be a range vector, it is not difficult to see that we have $\vec{\alpha_0}/\vec{\rho_0}$. Hence, there is a terminal derivation whose first step uses an instantiation of the clause $\vec{\Psi}$ which is therefore productive.

[7] Note that our definition for simplicity is stronger than the one of [Groenink 97].

We show by induction on the length of terminal derivations that the converse also holds. The basic step is true since derivations of length 1 are processed at step 2 of the algorithm. Assume that our property holds for terminal derivation of length $l-1$ and we consider a terminal derivation of length l whose first step is characterized by the binding $\vec{\Psi}/\vec{\Omega}$. There are m independent terminal derivations headed at $A_j(\vec{\Omega}[j])$ whose first step used the clause say $\vec{\Psi}_j$ and whose length is less than l. Therefore we have $Pr[\vec{\Psi}_j] = true$ by the induction hypothesis and $Pr[\vec{\Psi}] = true$ by step 3) of our algorithm.

∎

Remark that our algorithm gives only a necessary condition for a clause to be productive if the grammar at hand is not top-down linear. The reason is that it cannot check the fact that the string instantiation of different arguments are identical, when these arguments are identical variables.

Example 5 *Consider the not top-down linear PRCG whose clauses are:*

$$\begin{array}{lcl} S(X) & \rightarrow & A(X)\ B(X) \\ A(a) & \rightarrow & \varepsilon \\ B(b) & \rightarrow & \varepsilon \end{array}$$

our algorithm will set $Pr[S(X) \rightarrow A(X)\ B(X)]$ to true though its language is empty.

Property 5 *The emptiness problem is solvable for any sPRCG.*

We only have to look at the value of $Pr[\vec{\Psi}]$ for some S-clause $\vec{\Psi}$.

In Sections 8, 9 and 10 we will show that a variety of usual syntactic formalisms can be transformed into an equivalent sPRCG. The previous property also shows that the emptiness problem is solvable for all these formalisms:

Property 6 *The emptiness problem is solvable for LIGs, TAGs, HGs, CCFGs and LCFRS.*

Definition 11 *We say that a clause is accessible (otherwise, it is inaccessible) if, for some $w \in T^*$, it is is used in a $S(\bullet w \bullet)$-derivation.*

Recall that we only consider RCGs which are instantiable and whose CF skeleton is reduced. If moreover we consider the non-combinatorial linear version, each clause in this grammar is accessible. If the grammar is not top-down linear, the checking that different RHS arguments with identical variables match identical ranges cannot be performed.

Definition 12 *We say that a clause is useful (otherwise, it is useless) if it used in a $S(\bullet w \bullet)/\varepsilon$-derivation for some $w \in T^*$.*

Definition 13 *A RCG is reduced if all its clauses are useful.*

If the grammar is simple, we can remove all its useless clauses by a variant of the classical algorithm for CFGs.

If this algorithm is applied to a non linear RCG, we get an equivalent RCG which may be not reduced.

5.1 ε-freeness

Definition 14 *A clause $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m})$ is an ε-clause if one argument of a clause $\vec{\alpha_i}[k], 0 \leq i \leq m, 1 \leq k \leq l_{\vec{\alpha_i}}$ is the empty string.*

Definition 15 *We say that a RCG $G = (N, T, V, P, S)$ is ε-free if either*

1. *P has no ε-clauses, or*
2. *There is exactly one ε-clause $S(\varepsilon) \rightarrow \varepsilon$ and S or $\overline{S}$ do not appear in the RHS of any clause in P.*

Our purpose is, as in the CF case, to transform RCGs into an equivalent ε-free form.

In this section we will use a particular type of substitution, in which variables are substituted by the empty string. The set of all these variables is called an *ε-substitution*. Let v be an ε-substitution, $\alpha = w_0X_1w_1\ldots w_{p-1}X_pw_p$ be a string in $(V\cup T)^*$. The *ε-instantiation* of α by v is the string $\alpha' = w_0Y_1w_1\ldots w_{p-1}Y_pw_p$ where $\forall j, 1\leq j\leq p$ we have $Y_j = \varepsilon$ if $X_j\in v$ or $Y_j = X_j$ if $X_j\notin v$. This notion of ε-instantiation extends from strings to vector of strings. To designate the components of a vector of strings $\vec{\alpha}$ whose ε-instantiation by some v is ε, we introduce the notion of *characteristic vector*. It is a vector of booleans (in $\{0,1\}^*$) denoted by $\vec{\imath}$ whose size is the size of $\vec{\alpha}$ and whose value $\vec{\imath}[k]$ is 0 iff the ε-instantiation of $\vec{\alpha}[k]$ by v is ε. By convention, the vectors $\vec{0}$ and $\vec{1}$ designate sequences of 0's or 1's of appropriate length. For a given couple $(\vec{\alpha}, v)$, there is a single characteristic vector $\vec{\imath}$.

In a dual manner, consider both a vector of strings $\vec{\alpha}$ and a boolean vector $\vec{\imath}$ of the same size. If there is a set of variables v such that the characteristic vector of $\vec{\alpha}$ by v is $\vec{\imath}$, we say that v is an *ε-substitution* and $\vec{\imath}$ is a *characteristic vector* of $\vec{\alpha}$. This notion extends to sequence of vector of strings and sequence of boolean vectors when there are $\vec{\alpha_1},\ldots,\vec{\alpha_p}$, $\vec{\imath_1},\ldots,\vec{\imath_p}$ and a set of variables v which is an ε-substitution for each couple $(\vec{\alpha_j},\vec{\imath_j})$.

For each RCG $G = (N,T,V,P,S)$, the following algorithm computes a set called *Empty* whose elements are couples of nonterminals and boolean vectors $(A,\vec{\imath})$ where the size of $\vec{\imath}$ is the arity of A. The idea is to gather, for each nonterminal A, from all its A-clauses its characteristic vectors which are compatible with the characteristic vectors of its RHS. In other words, if $(A_0,\vec{\imath_0})\in$ *Empty*, this means that there is in P an A_0-clause $A_0(\vec{\alpha_0})\rightarrow A_1(\vec{\alpha_1})\ldots A_m(\vec{\alpha_m})$ and there exists an ε-substitution v such that the characteristic vectors of $\vec{\alpha_0},\vec{\alpha_1},\ldots,\vec{\alpha_m}$ by v are respectively $\vec{\imath_0},\vec{\imath_1},\ldots,\vec{\imath_m}$ and of course $(A_1,\vec{\imath_1}),\ldots,(A_m,\vec{\imath_m})$ are also elements of *Empty*.

Algorithm 1 *Empty* **construction.**
Let $\vec{\Psi} = A_0(\vec{\alpha_0})\rightarrow A_1(\vec{\alpha_1})\ldots A_j(\vec{\alpha_j})\ldots A_m(\vec{\alpha_m})$ be the positive clause associated to any clause in P.

1. *Initially Empty $=\emptyset$.*
2. *If the RHS of $\vec{\Psi}$ is empty, we construct the boolean vector $\vec{\imath_0}$ such that $\vec{\imath_0}[k]=0\Longleftrightarrow\vec{\alpha_0}[k]=\varepsilon$ and we add $(A_0,\vec{\imath_0})$ to Empty.*
3. *For each ε-substitution v such that the characteristic vectors of each $\vec{\alpha_j}, 1\leq j\leq m$ by v is $\vec{\imath_j}$ and $(A_j,\vec{\imath_j})\in$ Empty, we add $(A_0,\vec{\imath_0})$ to Empty where $\vec{\imath_0}$ is the characteristic vector of $\vec{\alpha_0}$ by v.*
4. *Step 3) is repeated until stability.*

Remark that *Empty* may contain some $(A,\vec{0})$ elements.

Let $\vec{u}$ be a vector, and $\vec{\imath}$ be a boolean vector of equal length. The *instantiation* of $\vec{u}$ by $\vec{\imath}$ is the vector of the same length denoted $\vec{u}^{\vec{\imath}}$ where the components of $\vec{u}$ which correspond to a null component in $\vec{\imath}$ have been changed in ε ($\vec{\imath}[j]=0\Longrightarrow\vec{u}^{\vec{\imath}}[j]=\varepsilon, \vec{\imath}[j]=1\Longrightarrow\vec{u}^{\vec{\imath}}[j]=\vec{u}[j]$). We call *$\varepsilon$-restriction* of $\vec{u}$ by $\vec{\imath}$ the vector denoted by $\langle\vec{u}\rangle_{\vec{\imath}}$ where all components of $\vec{u}$ which correspond to a null component in $\vec{\imath}$ have been erased. More formally, we have $l_{\langle\vec{u}\rangle_{\vec{\imath}}} = |\{j\mid\vec{\imath}[j]=1\}|$ and for each $k, 1\leq k\leq l_{\langle\vec{u}\rangle_{\vec{\imath}}}$: $\langle\vec{u}\rangle_{\vec{\imath}}[k]=\vec{u}[j]\Longleftrightarrow\vec{\imath}[j]=1\wedge k=|\{h\mid\vec{\imath}[h]=1\wedge h\leq j\}|$.

Now, consider a clause $\vec{\Psi} = A_0(\vec{\alpha_0})\rightarrow A_1(\vec{\alpha_1})\ldots A_j(\vec{\alpha_j})\ldots A_m(\vec{\alpha_m})$, a vector of boolean vectors $\vec{I} = \vec{\imath_0},\vec{\imath_1},\ldots,\vec{\imath_j},\ldots,\vec{\imath_m}$ whose sizes are respectively the arities of $A_0,A_1,\ldots,A_j,\ldots,A_m$ and an ε-substitution v for each $(\vec{\alpha_j},\vec{\imath_j}), 0\leq j\leq m$. The clause $A_0^{\vec{\imath_0}}(\langle\vec{\alpha}_0\rangle_{\vec{\imath_0}})\rightarrow A_1^{\vec{\imath_1}}(\langle\vec{\alpha}_1\rangle_{\vec{\imath_1}})\ldots A_m^{\vec{\imath_m}}(\langle\vec{\alpha}_m\rangle_{\vec{\imath_m}})$ where the $A_j^{\vec{\imath_j}}$'s are new nonterminals, is called the *ε-restriction* of $\vec{\Psi}$ by $\vec{I}$ and is denoted by $\langle\vec{\Psi}\rangle_{\vec{I}}$. Of course, a negative nonterminal occurrence in $\vec{\Psi}$ produced its negative counterpart in $\langle\vec{\Psi}\rangle_{\vec{I}}$. For example if the j^{th} predicate of $\vec{\Psi}$ is $\overline{A_j}(\vec{\alpha_j})$, the j^{th} predicate of $\langle\vec{\Psi}\rangle_{\vec{I}}$ is $\overline{A_j^{\vec{\imath_j}}}(\langle\vec{\alpha_j}\rangle_{\vec{\imath_j}})$.

Algorithm 2 Conversion to an ε-free grammar.

Input. *A simple RCG $G = (N,T,V,P,S)$.*

Output. *An equivalent ε-free RCG $G' = (N',T,V\cup\{X\},P',S')$.*

Method. 1. *Construct Empty by the previous algorithm. Initially we have $P' = \{S'(X)\rightarrow S^{\vec{1}}(X)\}$.*

2. *If $\vec{\Psi} = A_0(\vec{\alpha_0})\rightarrow A_1(\vec{\alpha_1})\ldots A_j(\vec{\alpha_j})\ldots A_m(\vec{\alpha_m})$ is in P, add to P' all the ε-restrictions $\langle\vec{\Psi}\rangle_{\vec{I}}$ where $\vec{I} = \vec{\imath_0},\vec{\imath_1},\ldots,\vec{\imath_j},\ldots,\vec{\imath_m}$ and the boolean vectors $\vec{\imath_j}, 0\leq j\leq m$ are such that $(A_j,\vec{\imath_j})\in$ Empty and there is an ε-substitution v such that the characteristic vectors of each $\vec{\alpha_j}, 0\leq j\leq m$ by v is $\vec{\imath_j}$. By*

convention we have $A_j^{\vec{\imath_j}}(\langle\vec{\alpha_j}\rangle_{\vec{\imath_j}}) = \varepsilon$ *if* $\vec{\imath_j} = \vec{0}$. *Be sure not to add* $A_0^{\vec{0}}(\langle\vec{\alpha_0}\rangle_{\vec{0}}) \rightarrow \varepsilon$. *Note that in the general case it should be possible to add to* P' *a "clause" whose LHS is an empty string while its RHS is not empty. The restrictions on the input RCG (i.e. non-combinatorial and bottom-up non-erasing) are only set to prohibit such a case.*

3. *If* $(S, \vec{0}) \in$ *Empty, add to* P' *the clause* $S'(\varepsilon) \rightarrow \varepsilon$.

Remark that if a clause in P does not fulfill the conditions of step 2), this clause is not productive and nothing is added to P'.

In the proof we must show that both G' is an ε-free grammar and is equivalent to G. The first part results from the property that in step 2), we have $\forall k : \vec{\alpha_0}[k] = \varepsilon \Longrightarrow \vec{\imath_0}[k] = 0$ which means that if the k^{th} argument of $A_0(\vec{\alpha_0})$ is the empty string, all the $\vec{\imath_0}$'s have a 0 in the k^{th} position and therefore the constructed clauses are ε-free.

The equivalence between G and G' can be shown as follows.

The case $\varepsilon \in \mathcal{L}(G) \Longleftrightarrow \varepsilon \in \mathcal{L}(G')$ is evident. If $w \neq \varepsilon \wedge w \in \mathcal{L}(G)$, there exists a derivation $d = S(\bullet w \bullet) \underset{G,w}{\overset{+}{\Rightarrow}} \varepsilon$. The idea is to mimic d by a derivation d' in G' which is built from right to left in the following way. Consider any derivation step in d

$$\Gamma_1\ A_0(\vec{\rho_0})\ \Gamma_2 \underset{G,w}{\Rightarrow} \Gamma_1\ A_1(\vec{\rho_1}) \ldots A_m(\vec{\rho_m})\ \Gamma_2$$

the corresponding step in d' is

$$\Gamma'_1\ A_0^{\vec{\imath_0}}(\langle\vec{\rho_0}\rangle_{\vec{\imath_0}})\ \Gamma'_2 \underset{G',w}{\Rightarrow} \Gamma'_1\ A_1^{\vec{\imath_1}}(\langle\vec{\rho_1}\rangle_{\vec{\imath_1}}) \ldots A_m^{\vec{\imath_m}}(\langle\vec{\rho_m}\rangle_{\vec{\imath_m}})\ \Gamma'_2$$

such that $(A_0, \vec{\imath_0}) \in$ *Empty* and the empty ranges of $\vec{\rho_0}$ correspond to the null components of $\vec{\imath_0}$ (of course, by hypothesis, we have $\forall j, 1 \leq j \leq m : (A_j, \vec{\imath_j}) \in$ *Empty* and the empty ranges of $\vec{\rho_j}$ correspond to the null components of $\vec{\imath_j}$). It is not difficult to see that if $\vec{\Psi}/\vec{\Omega}$ is the clause binding used in d, then $\langle\vec{\Psi}\rangle_{\vec{I}}/\langle\vec{\Omega}\rangle_{\vec{I}}$ is the clause binding used in d' where $\vec{I} = \vec{\imath_0}, \vec{\imath_1}, \ldots, \vec{\imath_m}$, except when the $\vec{\imath_j}$'s are all null vectors. In such a case, the derivation parts in d associated with a null characteristic vector merely disappear in d'.

The converse to get a d from a d' also holds. We will assume, without lost of generality that from any clause $\langle\vec{\Psi}\rangle_{\vec{I}}$ in P' we know both $\vec{\Psi}$, the initial clause in P and $\vec{I}$, the sequence of characteristic vectors. Therefore, for each binding $\langle\vec{\Psi}\rangle_{\vec{I}}/\langle\vec{\Omega}\rangle_{\vec{I}}$ in d' we know the corresponding clause binding $\vec{\Psi}/\vec{\Omega}$ in d. However, there is a double difficulty here: the null components of the characteristic vectors give empty ranges in $\vec{\Omega}$, and for each, we have to find one among the $n+1$ possible $(\langle 0\rangle_w, \ldots, \langle n\rangle_w)$ and second, we may have to invent some derivation parts in d, which do not exist in d', and which correspond to null characteristic vectors. If we examine d', from left to right, assume that the clause binding used at one step is $\langle\vec{\Psi}\rangle_{\vec{I}}/\langle\vec{\Omega}\rangle_{\vec{I}}$, where $\vec{I} = \vec{\imath_0}, \vec{\imath_1}, \ldots, \vec{\imath_m}$, $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m})$, and that in d the empty ranges corresponding to the null components of the LHS characteristic vector $\vec{\imath_0}$ are known. From d', we also know the (non empty) ranges of the variables which correspond to unit values in the $\vec{\imath_j}$'s, therefore, since G is simple, the variable binding is known and therefore the clause binding $\vec{\Psi}/\vec{\Omega}$ is also known. If an instantiated predicate $A^{\vec{\imath}}(\langle\vec{\rho}\rangle_{\vec{\imath}})$ (which is the range instantiation of $A^{\vec{\imath}}(\langle\vec{\alpha}\rangle_{\vec{\imath}})$) has disappeared in d', this means that $\vec{\imath} = \vec{0}$ and that all the components of $\vec{\rho}$ are empty ranges. By the previous statement, we know that these empty ranges are exactly known. Therefore, we have to graft in d a terminal derivation part headed at $A(\vec{\rho})$. By taking into account the informations in *Empty* and G, we are capable to construct such a derivation involving only empty ranges.

Example 6 *The following non* ε*-free grammar*

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

produced the set $Empty = \{(A,10), (A,01), (S,1)\}$ *and is transformed by our algorithm into*

$$\begin{array}{lcl} S'(X) & \rightarrow & S^1(X) \\ S^1(X) & \rightarrow & A^{10}(X) \\ S^1(Y) & \rightarrow & A^{01}(Y) \\ A^{10}(a) & \rightarrow & \varepsilon \\ A^{01}(b) & \rightarrow & \varepsilon \end{array}$$

Example 7 *The following non ε-free grammar*

$$\begin{array}{lcl} S(XYZ) & \rightarrow & A(X,Y,Z) \\ A(X_1X_2,Y_1Y_2,Z_1Z_2) & \rightarrow & A(Y_1,Z_1,X_1)\ \ B(Z_2,X_2,Y_2) \\ A(a,b,\varepsilon) & \rightarrow & \varepsilon \\ A(c,\varepsilon,d) & \rightarrow & \varepsilon \\ B(e,\varepsilon,\varepsilon) & \rightarrow & \varepsilon \end{array}$$

produced the set $Empty = \{(B,100),(A,110),(A,101),(A,111),(A,011),(S,1)\}$ *and is transformed by our algorithm into*

$$\begin{array}{lcl} S'(X) & \rightarrow & S^1(X) \\ S^1(XYZ) & \rightarrow & A^{111}(X,Y,Z) \\ S^1(XY) & \rightarrow & A^{110}(X,Y) \\ S^1(XZ) & \rightarrow & A^{101}(X,Z) \\ S^1(YZ) & \rightarrow & A^{011}(Y,Z) \\ A^{111}(X_1,Y_1,Z_1Z_2) & \rightarrow & A^{111}(Y_1,Z_1,X_1)\ \ B^{100}(Z_2) \\ A^{111}(X_1,Y_1,Z_2) & \rightarrow & A^{101}(Y_1,X_1)\ \ B^{100}(Z_2) \\ A^{101}(X_1,Z_1Z_2) & \rightarrow & A^{011}(Z_1,X_1)\ \ B^{100}(Z_2) \\ A^{011}(Y_1,Z_1Z_2) & \rightarrow & A^{110}(Y_1,Z_1)\ \ B^{100}(Z_2) \\ A^{110}(a,b) & \rightarrow & \varepsilon \\ A^{101}(c,d) & \rightarrow & \varepsilon \\ B^{100}(e) & \rightarrow & \varepsilon \end{array}$$

6 2-var Form

A RCG is in *k-var form* if there is at most k variable occurrences in any given argument.

Property 7 *For each RCG, there is an equivalent RCG in 2-var form.*

We know that we may assume that the initial RCG is in non-combinatorial form (see Property 1).

Assume that the clause $A(\vec{\alpha}) \rightarrow \Psi$ is not in 2-var form, because, for some j, the argument $\alpha = \vec{\alpha}[j]$ is not. We simply split α in substrings $\alpha = \alpha_1 \ldots \alpha_h \ldots \alpha_l$, each α_h being in 2-var form and we produce a sequence of clauses using new predicates $A_2 \ldots A_l$ of increasing arities as follows. Underlined strings denotes new variables.

$$\begin{array}{lcl} A(\ldots,\underline{\alpha_1\,\alpha_2\ldots\alpha_l},\ldots) & \rightarrow & A_2(\ldots,\underline{\alpha_1},\underline{\alpha_2\ldots\alpha_l},\ldots) \\ \ldots & & \\ A_h(\ldots,\underline{\alpha_1},\ldots,\underline{\alpha_h\,\alpha_{h+1}\ldots\alpha_l},\ldots) & \rightarrow & A_{h+1}(\ldots,\underline{\alpha_1},\ldots,\underline{\alpha_h},\underline{\alpha_{h+1}\ldots\alpha_l},\ldots) \\ \ldots & & \\ A_{l-1}(\ldots,\underline{\alpha_1},\ldots,\underline{\alpha_{l-1}\,\alpha_l},\ldots) & \rightarrow & A_l(\ldots,\underline{\alpha_1},\ldots,\underline{\alpha_{l-1}},\underline{\alpha_l},\ldots) \\ A_l(\ldots,\alpha_1,\ldots,\alpha_j,\ldots,\alpha_l,\ldots) & \rightarrow & \Psi \end{array}$$

This transformation is performed while there is an argument which is not in 2-var form.

In some cases the transformation into a 2-var form can be performed without increasing the arity of the grammar.

This is always the case when the arity of the initial grammar is one, there is a transformation (very similar to the transformation in Chomsky normal form for CFGs) which construct an equivalent 1-RCG in 2-var form. But when the arity is greater than one, the pairing of variables in different arguments may or may not be favorable. In the first example below, there is a transformation which preserves the initial arity

Example 8

$$\begin{array}{lcl} A(XYZ,UVW) & \rightarrow & B(X,W)\ C(Z,V)\ D(Y,U) \\ & \Longrightarrow & \\ A(X\underline{YZ},\underline{UV}W) & \rightarrow & B(X,W)\ B'(\underline{YZ},\underline{UV}) \\ B'(YZ,UV) & \rightarrow & C(Z,V)\ D(Y,U) \end{array}$$

But this is not always the case. It is not possible to transform the following clause in 4-var form into a set of equivalent clauses in 2-var form without increasing the number of arguments.

Example 9

$$\begin{array}{lll} A(XYZT,UVWS) & \rightarrow & B(X,V)\ C(Y,S)\ D(Z,U)\ E(T,W) \\ & \Longrightarrow & \\ & ??? & \end{array}$$

7 CFGs & sPRCGs

Property 8 *For any CFG, there is an equivalent 1-sPRCG*

Let $A \rightarrow w_0 B_1 w_1 \ldots B_p w_p$ be a production in some CFG (N, T, P, S) where $w_i \in T^*$ and $B_i \in N$ and $A(w_0 X_1 w_1 \ldots X_p w_p) \rightarrow B_1(X_1) \ldots B_p(X_p)$ a corresponding clause where the X_i's are p different variables. By construction, the arities of its predicates are one and moreover this clause is in simple form. If we apply this transformation to all the productions of any CFG, due to the bijection between P and the set of clauses clauses, it is not difficult to see that we get an equivalent 1-PRCG in simple form.

The converse is equally true. Note however that the order of the RHS predicates in a clause is free, but when we built a CF production from a simple clause whose arity is 1, the order of the RHS nonterminals (if any) in the production is given by the order of the variables in the LHS arguments of the clause.

Therefore, the languages defined by the class of simple PRCGs with a single argument are the context-free languages:

Property 9

$$\mathit{CFL} \quad = \quad \mathit{1\text{-}sPRCL}$$

This equivalence does not hold when we consider 1-PRCGs which are strictly more powerful than 1-sPRCGs.

This property comes from the fact that RCGs are closed under intersection (see Section 12). For example, this closure property may allow to describe the language $\mathcal{L} = \{a^n b^n c^n \mid n \geq 0\}$ by a 1-PRCG as being the intersection of $\mathcal{L}_1 = \{a^n b^n c^k \mid n, k \geq 0\}$ and $\mathcal{L}_2 = \{a^k b^n c^n \mid n, k \geq 0\}$

where $\mathcal{L}_1$ is defined by

$$\begin{array}{lcl} S_1(XY) & \rightarrow & A_1(X)\ B_1(Y) \\ A_1(\varepsilon) & \rightarrow & \varepsilon \\ A_1(aXb) & \rightarrow & A_1(X) \\ B_1(\varepsilon) & \rightarrow & \varepsilon \\ B_1(cX) & \rightarrow & B_1(X) \end{array}$$

where $\mathcal{L}_2$ is defined by

$$\begin{array}{lcl} S_2(XY) & \rightarrow & A_2(X)\ B_2(Y) \\ A_2(\varepsilon) & \rightarrow & \varepsilon \\ A_2(aX) & \rightarrow & A_2(X) \\ B_2(\varepsilon) & \rightarrow & \varepsilon \\ B_2(bXc) & \rightarrow & B_2(X) \end{array}$$

and $\mathcal{L}$ is defined by

$$S(X) \quad \rightarrow \quad S_1(X)\ S_2(X)$$

The same language $\mathcal{L} = \{a^n b^n c^n \mid n \geq 0\}$ can also be defined by the following combinatorial 1-PRCG

$$\begin{array}{lcl} S(XYZ) & \rightarrow & A(XY)\ B(YZ) \\ A(aXb) & \rightarrow & A(X) \\ A(\varepsilon) & \rightarrow & \varepsilon \\ B(bXc) & \rightarrow & B(X) \\ B(\varepsilon) & \rightarrow & \varepsilon \end{array}$$

8 LIGs, TAGs, HGs & sPRCGs

In [Vijay-Shanker and Weir 94a] Vijay-Shanker and Weir have shown that linear indexed grammars, tree adjoining grammars and head grammars are equivalent formalisms[8], though they appear to be quite different. In this section we will show that each of this formalism can be transformed into an equivalent 2-sPRCG.

We will not discuss here the relevance of these formalisms to describe linguistic phenomena. Many references can be found in [Vijay-Shanker and Weir 94a]

8.1 Linear Indexed Grammars & sPRCGs

Indexed grammars are syntactic formalisms which are extensions of CFGs in which a stack of symbols is associated with each nonterminal. These grammars express both a CF rewriting system and the way these stacks evolve within derivations. In a linear indexed grammar (LIG), which is a restricted form of indexed grammar, there is a single distinguished stack associated with a RHS nonterminal which is related with the stack associated with the LHS nonterminal, all other stacks are independent and of bounded size. Of course, this distinguished stack may change between the LHS and the RHS, some symbols may be pushed or popped.

Following [Vijay-Shanker and Weir 94b]

Definition 16 *A LIG, L is denoted by (V_N, V_T, V_I, P_L, S) where:*

- *V_N is a non-empty finite set of nonterminal symbols;*
- *V_T is a finite set of terminal symbols, V_N and V_T are disjoint;*
- *V_I is a finite set of stack symbols (I stands for indices);*
- *P_L is a finite set of productions;*
- *$S \in V_N$ is the start symbol.*

A string of stack symbols is an element of V_I^*. In this section, we adopt the convention that α denotes members of V_I^* and a denotes elements of V_I. In fact, in a LIG production, the structure associated with a nonterminal can be either a stack or a stack schema. A *stack schema* denoted by $(..\alpha)$ matches all the stacks whose prefix (bottom) part is left unspecified and whose suffix (top) part is α. In a LIG production, we call *primary constituent* the pair denoted $A(..\alpha)$, consisting of a nonterminal A, and a stack schema $(..\alpha)$ and *secondary constituent* the pair denoted $A(\alpha)$, consisting of a nonterminal A, and a string of stack symbols α. If $\alpha = \varepsilon$, we have an *empty secondary constituent* denoted by $A()$.

Without any loss of generality, we assume that productions in P_L have the following forms

$$\begin{array}{lcl} A() & \rightarrow & x \\ A(..a) & \rightarrow & \Gamma_1 \;\; B(..a')\; \Gamma_2 \end{array}$$

where $A, B \in V_N$, $aa' \in V_I \cup \{\varepsilon\}$, $x \in V_T^*$ and $\Gamma_1\Gamma_2 \in V_T^* \cup \{D() \mid D \in V_N\}$.

We call *object* the pair denoted by $A(\alpha)$ where A is a nonterminal and (α) a stack of symbols. If $\alpha = \varepsilon$, we have an *empty object* denoted by $A()$. Let V_O be the set of objects $V_O = \{A(\alpha) \mid A \in V_N \wedge \alpha \in V_I^*\}$. We define on $(V_O \cup V_T)^*$ the binary relation *derive* denoted $\underset{L}{\Rightarrow}$ (the relation symbol is sometimes overlined by a production in P_L) by:

$$\Gamma_1'\; A(\alpha a)\; \Gamma_2' \quad \overset{A(..a)\rightarrow\Gamma_1\; B(..a')\; \Gamma_2}{\underset{L}{\Rightarrow}} \quad \Gamma_1'\; \Gamma_1\; B(\alpha a')\; \Gamma_2\; \Gamma_2'$$

$$\Gamma_1'\; A()\; \Gamma_2' \quad \overset{A()\rightarrow x}{\underset{L}{\Rightarrow}} \quad \Gamma_1'\; x\; \Gamma_2'$$

In the first above element we say that the object $B(\alpha a')$ is the *distinguished child* of $A(\alpha a)$, and if $\Gamma_1\Gamma_2 = D()$, $D()$ is the *secondary object*. For a derivation the reflexive transitive closure of the distinguished child relation is the *distinguished descendent* relation. Let $\Gamma_1, \ldots, \Gamma_i, \Gamma_{i+1}, \ldots, \Gamma_l$ be strings in $(V_O \cup V_T)^*$ such that $\forall i, 1 \leq i < l : \Gamma_i \underset{L}{\Rightarrow} \Gamma_{i+1}$ then the sequence of strings $(\Gamma_1, \ldots, \Gamma_i, \Gamma_{i+1}, \ldots, \Gamma_l)$ is called a *derivation*.

The sequence of objects $A_1(\alpha_1) \ldots A_i(\alpha_i) A_{i+1}(\alpha_{i+1}) \ldots A_p(\alpha_p)$ is called a *spine* if, there is a derivation in which each object $A_{i+1}(\alpha_{i+1})$ is the distinguished child of $A_i(\alpha_i)$.

The language defined by a LIG L is the set:

[8] In fact, this equivalence also holds with a fourth formalism, the combinatory categorial grammars.

$$\mathcal{L}(L) = \{w \mid S() \overset{+}{\underset{L}{\Rightarrow}} w \wedge w \in V_T^*\}$$

As in the CF case we can talk of rightmost (resp. leftmost) derivations when the rightmost (resp. leftmost) object is derived at each step.

Associated with a LIG $L = (V_N, V_T, V_I, P_L, S)$, we define a bunch of binary relations which are borrowed from [Boullier 95]

$$\begin{aligned}
\underset{1}{=} &= \{(A,B) \mid A(..) \rightarrow \Gamma_1 B(..)\Gamma_2 \in P_L\} \\
\overset{a}{\underset{1}{\prec}} &= \{(A,B) \mid A(..) \rightarrow \Gamma_1 B(..a)\Gamma_2 \in P_L\} \\
\overset{a}{\underset{1}{\succ}} &= \{(A,B) \mid A(..a) \rightarrow \Gamma_1 B(..)\Gamma_2 \in P_L\} \\
\underset{+}{=} &= \{(A_1,A_p) \mid A_1() \overset{+}{\underset{L}{\Rightarrow}} \Gamma_1\ A_p()\ \Gamma_2 \text{ and } A_p() \text{ is a distinguished descendent of } A_1()\}
\end{aligned}$$

The *1-level* relations simply indicate, for each production, which operation can be apply to the stack associated with the LHS nonterminal to get the stack associated with its distinguished child; $\underset{1}{=}$ indicates equality, $\overset{a}{\underset{1}{\prec}}$ the pushing of a, and $\overset{a}{\underset{1}{\succ}}$ the popping of a.

If we look at the evolution of a stack along a spine $A_1(\alpha_1)\ldots A_i(\alpha_i)A_{i+1}(\alpha_{i+1})\ldots A_p(\alpha_p)$, between any two objects, one of the following equalities holds: $\alpha_i = \alpha_{i+1}$, $\alpha_i a = \alpha_{i+1}$, or $\alpha_i = \alpha_{i+1}a$.

The $\underset{+}{=}$ relation select pairs of nonterminals (A_j, A_k) along spines s.t. $j < k$, $\alpha_j = \alpha_k$ and $\forall l, j \leq l \leq k$: $\alpha_l = \alpha_j\alpha', \alpha' \in V_I^*$.

If the relations $\overset{a}{\underset{+}{\succ}}$ and $\underset{+}{\diamond}$ are defined as

$$\begin{aligned}
\overset{a}{\underset{+}{\succ}} &= \overset{a}{\underset{1}{\succ}} \cup \underset{+}{=}\overset{a}{\underset{1}{\succ}} \\
\underset{+}{\diamond} &= \bigcup_{a \in V_I} \overset{a}{\underset{1}{\prec}}\overset{a}{\underset{+}{\succ}}
\end{aligned}$$

we can easily see that the following identity holds

Property 10

$$\underset{+}{=} = \underset{1}{=} \cup \underset{+}{\diamond} \cup \underset{1}{=}\underset{+}{=} \cup \underset{+}{\diamond}\underset{+}{=}$$

In [Boullier 95] we can found an algorithm[9] which computes the $\underset{+}{=}$, $\overset{a}{\underset{+}{\succ}}$ and $\underset{+}{\diamond}$ relations as the composition of $\underset{1}{=}$, $\overset{a}{\underset{1}{\prec}}$ and $\overset{a}{\underset{1}{\succ}}$ in $\mathcal{O}(|V_N|^3)$ time. Of course, the maximum size of these relations is $\mathcal{O}(|V_N|^2)$.

Property 11 *For any LIG, there is an equivalent 2-sPRCG in 2-var form.*

Let $L = (V_N, V_T, V_I, P_L, S)$ be a LIG. We are going to construct an equivalent 2-sPRCG $G = (N, T, V, P, \langle S \rangle)$ in 2-var form.

- The set of nonterminal symbols is $N = \{\langle A \rangle \mid A \in V_N\} \cup \{\langle A, B \rangle_\rho \mid A, B \in V_N \wedge \rho \in \mathcal{R}\}$, where $\mathcal{R}$ is the set of relations $\underset{+}{=}$, $\underset{+}{\diamond}$, $\overset{a}{\underset{+}{\succ}}$. In fact we will only use *valid* nonterminals $\langle A, B \rangle_\rho$ for which the relation ρ holds between A and B. The arity of the nonterminals in $\langle A \rangle$ form (resp. $\langle A, B \rangle_\rho$ form) is one (resp. two).
- The terminal symbols of G are the terminal symbols of L: $T = V_T$
- The variables will be taken among $V = \{W, X, Y, Z\}$.
- The set P is construct from P_L and the relations $\underset{+}{=}$, $\underset{+}{\diamond}$ and $\overset{a}{\underset{+}{\succ}}$. Remember that in a LIG production like $A(..a) \rightarrow \Gamma_1\ \ B(..a')\ \Gamma_2$, the string $\Gamma_1\Gamma_2$ may designate either a terminal string or a secondary constituent. The notations used below will depend of these two cases:

[9] Though in the referred paper, these relations are defined on constituents, the algorithm also applies to nonterminals.

1. If $\Gamma_1\Gamma_2 \in T^*$, we have $\gamma_1 = \Gamma_1$, $\gamma_2 = \Gamma_2$, $\langle\Gamma_1\rangle = \langle\Gamma_2\rangle = \varepsilon$ (the empty predicate string).
2. If $\Gamma_1\Gamma_2 = D()$, we have

$$\gamma_1 = \begin{cases} X & \text{if } \Gamma_1 = D() \\ \varepsilon & \text{if } \Gamma_1 = \varepsilon \end{cases}, \gamma_2 = \begin{cases} Y & \text{if } \Gamma_2 = D() \\ \varepsilon & \text{if } \Gamma_2 = \varepsilon \end{cases}, \langle\Gamma_1\rangle = \begin{cases} \langle D\rangle(X) & \text{if } \Gamma_1 = D() \\ \varepsilon & \text{if } \Gamma_1 = \varepsilon \end{cases}$$

and

$$\langle\Gamma_2\rangle = \begin{cases} \langle D\rangle(Y) & \text{if } \Gamma_2 = D() \\ \varepsilon & \text{if } \Gamma_2 = \varepsilon \end{cases} .$$

$$\begin{aligned}
P \;=\; & \{\langle A\rangle(x) \to \varepsilon \mid A() \to x \in P_L\} \cup && (1)\\
& \{\langle A\rangle(XxY) \to \langle A,B\rangle^{=}_{+}(X,Y) \mid B() \to x \in P_L\} \cup && (2)\\
& \{\langle A,C\rangle^{=}_{+}(\gamma_1,\gamma_2) \to \langle\Gamma_1\rangle\ \langle\Gamma_2\rangle \mid A(..) \to \Gamma_1 C(..)\Gamma_2 \in P_L\} \cup && (3)\\
& \{\langle A,C\rangle^{=}_{+}(X,Y) \to \langle A,C\rangle^{\diamond}_{+}(X,Y)\} \cup && (4)\\
& \{\langle A,C\rangle^{=}_{+}(\gamma_1 W, Z\gamma_2) \to \langle\Gamma_1\rangle\ \langle B,C\rangle^{=}_{+}(W,Z)\ \langle\Gamma_2\rangle \mid A(..) \to \Gamma_1 B(..)\Gamma_2 \in P_L\} \cup && (5)\\
& \{\langle A,C\rangle^{=}_{+}(XW,ZY) \to \langle A,B\rangle^{\diamond}_{+}(X,Y)\ \langle B,C\rangle^{=}_{+}(W,Z)\} \cup && (6)\\
& \{\langle A,C\rangle^{\diamond}_{+}(\gamma_1 W, Z\gamma_2) \to \langle\Gamma_1\rangle\ \langle B,C\rangle^{a}_{+}(W,Z)\ \langle\Gamma_2\rangle \mid A(..) \to \Gamma_1 B(..a)\Gamma_2 \in P_L\} \cup && (7)\\
& \{\langle A,C\rangle^{a}_{+}(\gamma_1,\gamma_2) \to \langle\Gamma_1\rangle\langle\Gamma_2\rangle \mid A(..a) \to \Gamma_1 C(..)\Gamma_2 \in P_L\} \cup && (8)\\
& \{\langle A,C\rangle^{a}_{+}(W\gamma_1,\gamma_2 Z) \to \langle\Gamma_1\rangle\ \langle A,B\rangle^{=}_{+}(W,Z)\ \langle\Gamma_2\rangle \mid B(..a) \to \Gamma_1 C(..)\Gamma_2 \in P_L\} && (9)
\end{aligned}$$

By construction, G is a 2-sPRCG in 2-var form.
Now, we are going to show that $\mathcal{L}(L) = \mathcal{L}(G)$.

Proof: To show this result we use a particular kind of derivations called linear derivations (see [Boullier 96]), denoted $\underset{l,L}{\Rightarrow}$ and which indicates at each step which object is going to be derived. Informally a secondary object (if any) and its descendents are derived before its corresponding primary object, whether it lays to its left or to its right.

In this proof we will consider several types of linear derivations.

- The *closed* derivations such as $A() \overset{+}{\underset{l,L}{\Rightarrow}} w$.
- The *balanced* derivations such as $A() \overset{+}{\underset{l,L}{\Rightarrow}} w_1\ B()\ w_3$ where $B()$ is a distinguished descendent of $A()$. In such a case we know that, by definition, the relation $\underset{+}{=}$ holds between A and B. Moreover, among the balanced derivation we distinguish the *push-pop* derivations in which the relation $\underset{+}{\diamond}$ also holds between A and B.
- The *pop-a* derivations such as $A(a) \overset{+}{\underset{l,L}{\Rightarrow}} w_1\ B()\ w_3$ where $B()$ is a distinguished descendent of $A(a)$ and such $A \overset{a}{\underset{+}{\succ}} B$.

The equivalence between our two languages will be proved by induction on the lengths of derivations. Our induction hypothesis are denoted by (H_i) (and their inverse by $\overline{H_i}$) with

H_1: If there is a closed derivation $A() \overset{+}{\underset{l,L}{\Rightarrow}} w$ in L then w is in the (string) language of $\langle A\rangle$ (i.e. $w \in \mathcal{L}(\langle A\rangle)$).

H_2: If there is a balanced derivation $A() \overset{+}{\underset{l,L}{\Rightarrow}} w_1\ B()\ w_3$ then the vector of strings (w_1, w_3) is in the (string) language of $\langle A,B\rangle^{=}_{+}$ (i.e. $(w_1,w_3) \in \mathcal{L}(\langle A,B\rangle^{=}_{+})$).

H_3: If there is a push-pop derivation $A() \underset{l,L}{\overset{+}{\Rightarrow}} w_1\ B()\ w_3$ then the vector of strings (w_1, w_3) is in the (string) language of $\langle A, B\rangle_{\underset{+}{\diamond}}$ (i.e. $(w_1, w_3) \in \mathcal{L}(\langle A, B\rangle_{\underset{+}{\diamond}})$).

H_4: If there is a pop-a derivation $A(a) \underset{l,L}{\overset{+}{\Rightarrow}} w_1\ B()\ w_3$ then the vector of strings (w_1, w_3) is in the (string) language of $\langle A, B\rangle_{\underset{+}{\overset{a}{\succ}}}$ (i.e. $(w_1, w_3) \in \mathcal{L}(\langle A, B\rangle_{\underset{+}{\overset{a}{\succ}}})$).

Proof of the H_i's. The proof of this part is performed by induction on the length of derivations in L.

basis. We first check that the H_i's are true for derivations of minimum length in L. This minimum length is one, except for push-pop derivations (hypothesis H_3) for which it is two.

H_1: Let $A() \underset{l,L}{\Rightarrow} w$ be a closed derivation of length one. Such a derivation exists iff there is in P_L a production of the form $A() \rightarrow w$. If it is the case, we know by (1) that the clause $\langle A\rangle(w) \rightarrow \varepsilon$ is in P and that therefore we have $w \in \mathcal{L}(\langle A\rangle)$.

H_2: Let $A() \underset{l,L}{\Rightarrow} w_1\ B()\ w_3$ be a balanced derivation of length 1. Such a derivation exists iff there is in P_L a production of the form $A(..) \rightarrow w_1\ B(..)\ w_3$. If it is the case, we know by (3) that the clause $\langle A, B\rangle_{\underset{+}{=}}(w_1, w_3) \rightarrow \varepsilon$ is in P and that therefore we have $(w_1, w_3) \in \mathcal{L}(\langle A, B\rangle_{\underset{+}{=}})$.

H_3: Let $A() \underset{l,L}{\Rightarrow} w'_1\ B(a)\ w'_3 \underset{l,L}{\Rightarrow} w'_1\ w''_1\ C()\ w''_3\ w'_3$ be a push-pop derivation of length 2. Such a derivation exists iff there are in P_L two derivations of the form $A(..) \rightarrow w'_1\ B(..a)\ w'_3$ and $B(..a) \rightarrow w''_1\ C(..)\ w''_3$. If it is the case, we know by (7) and (8) that there are in P two clauses $\langle A, C\rangle_{\underset{+}{\diamond}}(w'_1\ W, Z\ w'_3) \rightarrow \langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}}(W, Z)$ and $\langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}}(w''_1, w''_3) \rightarrow \varepsilon$. This second clause shows that $(w''_1, w''_3) \in \mathcal{L}(\langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}})$ and the first one shows that if (x, y) is in $\mathcal{L}(\langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}})$, then $(w'_1\ x, y\ w'_3)$ is in $\mathcal{L}(\langle A, C\rangle_{\underset{+}{\diamond}})$. Therefore we have $(w'_1\ w''_1, w''_3\ w'_3) \in \mathcal{L}(\langle A, C\rangle_{\underset{+}{\diamond}})$.

H_4: Let $d = A(a) \underset{l,L}{\Rightarrow} w_1\ B()\ w_3$ be a pop derivation of length 1. Such a derivation exists iff there is in P_L a production of the form $A(..a) \rightarrow w_1\ B(..)\ w_3$. If it is the case, we know by (8) that the clause $\langle A, B\rangle_{\underset{+}{\overset{a}{\succ}}}(w_1, w_3) \rightarrow \varepsilon$ is in P and that therefore we have $(w_1, w_3) \in \mathcal{L}(\langle A, B\rangle_{\underset{+}{\overset{a}{\succ}}})$.

Induction step. We assume that for all closed, balanced, push-pop and pop-a derivations in L of length 1, 2, ... and l the H_i's hold. We will prove that they also hold for all derivations of length $l+1$. Of course, all derivations of length $l+1$ have at least an inside step.

H_1: Let $d = A() \underset{l,L}{\overset{+}{\Rightarrow}} w$ be a closed derivation of length $l+1$. Since $l+1 \geq 2$, there is a last step which is not the first one and therefore we have $d = A() \underset{l,L}{\overset{+}{\Rightarrow}} w_1 B() w_3 \underset{l,L}{\Rightarrow} w_1 w_2 w_3$ with $w_1 w_2 w_3 = w$. This shows that first $d_1 = A() \underset{l,L}{\overset{+}{\Rightarrow}} w_1 B() w_3$ is a balanced derivation and second the production $B() \rightarrow w_2$ is in P_L. Therefore the conditions of (2) are fulfilled and the clause $\vec{\Psi} = \langle A\rangle(X w_2 Y) \rightarrow \langle A, B\rangle_{\underset{+}{=}}(X, Y)$ is in P. Moreover, the length of d_1 is less or equal to l, then H_2 holds for d_1 and $(w_1, w_3) \in \mathcal{L}(\langle A, B\rangle_{\underset{+}{=}})$. Combining with $\vec{\Psi}$ we get $w = w_1 w_2 w_3 \in \mathcal{L}(\langle A\rangle)$.

H_3: Let $d = A() \underset{l,L}{\Rightarrow} \Gamma_1\ \ B(a)\ \Gamma_3 \underset{l,L}{\overset{+}{\Rightarrow}} w_1\ C()\ w_3$ be a push-pop derivation of length $l+1$. The first step of d indicates that the production $A(..) \rightarrow \Gamma_1\ B(..a)\ \Gamma_3$ is in P_L. Therefore the conditions of (7) are fulfilled and the clause $\vec{\Psi} = \langle A, C\rangle_{\underset{+}{\diamond}}(\gamma_1 W, Z\gamma_3) \rightarrow \langle\Gamma_1\rangle\ \langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}}(W, Z)\ \langle\Gamma_3\rangle$ is in P. Moreover d may be decomposed in two parts d_1 and d_2 with $d_1 = A() \underset{l,L}{\Rightarrow} \Gamma_1\ B(a)\ \Gamma_3 \underset{l,L}{\overset{*}{\Rightarrow}} w'_1\ B(a)\ w'_3$ and $d_2 = B(a) \underset{l,L}{\overset{+}{\Rightarrow}} w''_1\ C()\ w''_3$ assuming that $w_1 = w'_1 w''_1$ and $w_3 = w''_3 w'_3$. Since the length of d_2 is less or equal to l, by H_4 we know that the property $p_1 : (w''_1, w''_3) \in \mathcal{L}(\langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}})$

holds. From d_1, we can deduce that there are two closed derivations $\Gamma_1 \underset{l,L}{\overset{*}{\Rightarrow}} w'_1$ and $\Gamma_3 \underset{l,L}{\overset{*}{\Rightarrow}} w'_3$ whose lengths are less or equal than l and which fulfill H_1[10]. Therefore we have $p_2 : w'_1 \in \mathcal{L}(\langle\Gamma_1\rangle)$ and $p_3 : w'_3 \in \mathcal{L}(\langle\Gamma_3\rangle)$. Properties p_1, p_2 and p_3 together with clause $\vec{\Psi}$ show that we have $(w'_1 w''_1, w''_3 w'_3) = (w_1, w_3) \in \mathcal{L}(\langle A, C\rangle \underset{+}{\multimap})$.

H_2: Let $d = A() \underset{l,L}{\overset{+}{\Rightarrow}} w_1\ C()\ w_3$ be a balanced derivation of length $l+1$. Since $l+1 \geq 2$, there is a first step which is not the last one, but two cases can occur at this first step

1. A symbol, say a is pushed on the primary stack. Hence we have $d = A() \underset{l,L}{\Rightarrow} \Gamma_1\ A'(a)\ \Gamma_3 \underset{l,L}{\overset{+}{\Rightarrow}} w_1\ C()\ w_3$. We know that somewhere in d there is a component $w'_1\ B()\ w'_3$ where $B()$ is a distinguished descendent of $A()$ and such that we have $A \underset{+}{\multimap} B$. Once again two cases can occur
   (a) $w'_1\ B()\ w'_3 = w_1\ C()\ w_3$. Here, d is a push-pop derivation. Therefore the conditions of (4) are fulfilled and the clause $\vec{\Psi} = \langle A, C\rangle_{\underset{+}{=}}(X, Y) \rightarrow \langle A, C\rangle_{\underset{+}{\multimap}}(X, Y)$ is in P. This clause indicates that $(x, y) \in \mathcal{L}(\langle A, B\rangle_{\underset{+}{\multimap}}) \Longrightarrow (x, y) \in \mathcal{L}(\langle A, B\rangle_{\underset{+}{=}})$. Since we have seen that H_3 is true for push-pop derivations of length $l+1$, H_2 is therefore also true for balanced derivations of length $l+1$ which are also push-pop derivations.
   (b) $w'_1\ B()\ w'_3 \neq w_1\ C()\ w_3$. Here d may be considered as the composition of a push-pop derivation say $d_1 = A() \underset{l,L}{\Rightarrow} \Gamma_1\ A'(a)\ \Gamma_3 \underset{l,L}{\overset{+}{\Rightarrow}} w'_1\ B()\ w'_3$ followed by a balanced derivation say $d_2 = B() \underset{l,L}{\overset{+}{\Rightarrow}} w''_1\ C()\ w''_3$ with $w_1 = w'_1 w''_1$ and $w_3 = w''_3 w'_3$. Since conditions (6) are fulfilled, the clause $\vec{\Psi} = \langle A, C\rangle_{\underset{+}{=}}(XW, ZY) \rightarrow \langle A, B\rangle_{\underset{+}{\multimap}}(X, Y)\ \ \langle B, C\rangle_{\underset{+}{=}}(W, Z)$ is construct. This clause indicates that if $(w, z) \in \mathcal{L}(\langle B, C\rangle_{\underset{+}{=}})$ and if $(x, y) \in \mathcal{L}(\langle AB\rangle_{\underset{+}{\multimap}})$, then $(xw, zy) \in \mathcal{L}(\langle A, C\rangle_{\underset{+}{=}})$. Moreover, each of d_1 and d_2 has a length less or equal than l and hence H_3 and H_2 apply. This shows that $(w'_1 w''_1, w''_3 w'_3) = (w_1, w_3) \in \mathcal{L}(\langle A, C\rangle_{\underset{+}{=}})$.
2. No symbol is pushed (and the primary stack stays empty). Hence we have $d = A() \underset{l,L}{\Rightarrow} \Gamma_1\ B()\ \Gamma_3 \underset{l,L}{\overset{*}{\Rightarrow}} w'_1\ B()\ w'_3 \underset{l,L}{\overset{+}{\Rightarrow}} w_1\ C()\ w_3$. The derivation d may be decomposed in two parts $d_1 = A() \underset{l,L}{\Rightarrow} \Gamma_1\ B()\ \Gamma_3 \underset{l,L}{\overset{*}{\Rightarrow}} w'_1\ B()\ w'_3$ and $d_2 = B() \underset{l,L}{\overset{+}{\Rightarrow}} w''_1\ C()\ w''_3$ with $w_1 = w'_1 w''_1$ and $w_3 = w''_3 w'_3$. The first step of d_1 shows that the production $A() \rightarrow \Gamma_1\ B()\ \Gamma_3 \in P_L$ and d_2 shows that $B \underset{+}{=} C$. Therefore conditions (5) are fulfilled and the clause $\vec{\Psi} = \langle A, C\rangle_{\underset{+}{=}}(\gamma_1 W, Z\gamma_3) \rightarrow \langle\Gamma_1\rangle\ \langle B, C\rangle_{\underset{+}{=}}(W, Z)\ \langle\Gamma_3\rangle$ is constructed. This clause indicates that if $x \in \mathcal{L}(\langle\Gamma_1\rangle)$, $y \in \mathcal{L}(\langle\Gamma_3\rangle)$ and $(w, z) \in \mathcal{L}(\langle B, C\rangle_{\underset{+}{=}})$, then $(xw, zy) \in \mathcal{L}(\langle A, C\rangle_{\underset{+}{=}})$. Moreover we have $\Gamma_1 \underset{l,L}{\overset{*}{\Rightarrow}} w'_1$, $\Gamma_3 \underset{l,L}{\overset{*}{\Rightarrow}} w'_3$ and the length of d_2 is less or equal to l, therefore H_1 and H_2 apply. This shows that $(w'_1 w''_1, w''_3 w'_3) = (w_1, w_3) \in \mathcal{L}(\langle A, C\rangle_{\underset{+}{=}})$.

H_4: Let $d = A(a) \underset{l,L}{\overset{+}{\Rightarrow}} w'_1\ B(a)\ w'_3 \underset{l,L}{\Rightarrow} w'_1\ \Gamma_1\ C()\ \Gamma_3\ w'_3 \underset{l,L}{\overset{*}{\Rightarrow}} w'_1 w''_1\ C()\ w''_3 w_3$ be a pop derivation of length $l+1$. This derivation may be decomposed in $d_1 = A(a) \underset{l,L}{\overset{+}{\Rightarrow}} w'_1\ B(a)\ w'_3$ and $d_2 = B(a) \underset{l,L}{\Rightarrow} \Gamma_1\ C()\ \Gamma_3 \underset{l,L}{\overset{*}{\Rightarrow}} w''_1\ C()\ w''_3$. The derivation d_1 shows that there is a balanced derivation $d'_1 = A() \underset{l,L}{\overset{+}{\Rightarrow}} w'_1\ B()\ w'_3$ where a, the bottom symbol in the stacks of the spine, has been erased. The derivation d_2 shows that we have $B(..a) \rightarrow \Gamma_1\ C()\ \Gamma_3 \in P_L$. Moreover d_2 contains two closed derivations $\Gamma_1 \underset{l,L}{\overset{*}{\Rightarrow}} w'_1$ and $\Gamma_3 \underset{l,L}{\overset{*}{\Rightarrow}} w'_3$. The conditions (9) are fulfilled and the clause $\vec{\Psi} = \langle A, C\rangle_{\underset{+}{\overset{a}{\succ}}}(W\gamma_1, \gamma_2 Z) \rightarrow \langle\Gamma_1\rangle\ \ \langle A, B\rangle_{\underset{+}{=}}(W, Z)\ \ \langle\Gamma_2\rangle$ is constructed. This clause indicates that

[10] If $\Gamma \in T^*$, we assume that $\mathcal{L}(\langle\Gamma\rangle) = \{\Gamma\}$.

if $x \in \mathcal{L}(\langle\Gamma_1\rangle)$, $y \in \mathcal{L}(\langle\Gamma_3\rangle)$ and $(w,z) \in \mathcal{L}(\langle A,B\rangle_{\underset{+}{=}})$, then $(wx,yz) \in \mathcal{L}(\langle A,C\rangle_{\underset{+}{\overset{a}{\succ}}})$. Moreover each of $\Gamma_1 \overset{*}{\underset{l,L}{\Rightarrow}} w_1''$, $\Gamma_3 \overset{*}{\underset{l,L}{\Rightarrow}} w_3''$ and d_1 has a length less or equal to l and hence H_1 and H_2 apply. This shows that $(w_1'w_1'', w_3''w_3') = (w_1,w_3) \in \mathcal{L}(\langle A,C\rangle_{\underset{+}{\overset{a}{\succ}}})$.

Proof of the $\overline{H_i}$'s. The proof of this part is performed by induction on the length of derivations in G. In this proof, the clause $\vec{\Psi}$ used in some derivation step may overlined the derive symbol (i.e. we may have $\overset{\vec{\Psi}}{\underset{G,w}{\Rightarrow}}$) and all the ranges ρ are elements of $\mathcal{R}_w$ for some $w \in T^*$.

basis. We first check that the $\overline{H_i}$'s are true for derivations of minimum length in G. This minimum length is one for any kind of nonterminal, except for the $\langle A,B\rangle_{\underset{+}{\diamond}}$'s (hypothesis H_3) for which it is two.

$\overline{H_1}$: Consider a terminal $\langle A\rangle$-derivation of length one in G $\langle A\rangle(\rho) \underset{G,w}{\Rightarrow} \varepsilon$ where $w^\rho = w_2$. Such a derivation exists iff there is in P a clause of the form $\langle A\rangle(w_2) \to \varepsilon$. If it is the case, we know by (1) that this clause has been produced iff the production $A() \to w_2$ is in P_L. Therefore, $\overline{H_1}$ holds since we have the closed derivation $A() \underset{l,L}{\Rightarrow} w_2$.

$\overline{H_2}$: Consider a terminal $\langle A,B\rangle_{\underset{+}{=}}$-derivation of length one in G $\langle A,B\rangle_{\underset{+}{=}}(\rho_1,\rho_3) \underset{G,w}{\Rightarrow} \varepsilon$ where $w^{\rho_1} = w_1$ and $w^{\rho_3} = w_3$. Such a derivation exists iff there is in P a clause of the form $\langle A,B\rangle_{\underset{+}{=}}(w_1,w_3) \to \varepsilon$. If it is the case, we know by (3) that this clause has been produced iff the production $A(..) \to w_1\ B(..)\ w_3$ is in P_L. Therefore, $\overline{H_2}$ holds since we have the balanced derivation $A() \underset{l,L}{\Rightarrow} w_1\ B()\ w_3$.

$\overline{H_3}$: We include in this basis step the $\langle A,B\rangle_{\underset{+}{\diamond\!\!-\!\!\diamond}}$-derivations of length two in G since this is the minimum length of these kinds of derivations. Consider the derivation $d = \langle A,B\rangle_{\underset{+}{\diamond}}(\rho_1',\rho_3') \overset{\vec{\Psi}_1}{\underset{G,w}{\Rightarrow}} \langle B,C\rangle_{\underset{+}{\overset{a}{\succ}}}(\rho_1'',\rho_3'') \overset{\vec{\Psi}_2}{\underset{G,w}{\Rightarrow}} \varepsilon$ where $\vec{\Psi}_1 = \langle A,B\rangle_{\underset{+}{\diamond}}(w_1'W, Zw_3') \to \langle B,C\rangle_{\underset{+}{\overset{a}{\succ}}}(W,Z)$, $\vec{\Psi}_2 = \langle B,C\rangle_{\underset{+}{\overset{a}{\succ}}}(w_1'',w_3'') \to \varepsilon$, $w^{\rho_1''} = w_1''$, $w^{\rho_3''} = w_3''$, $w^{\rho_1'} = w_1'$ and $w^{\rho_3'} = w_3'$. The derivation d exists iff the clauses $\vec{\Psi}_1$ and $\vec{\Psi}_2$ exist. If it is the case, they have been produced respectively in (7) and (8), this mean that the productions $A(..) \to w_1'\ B(..a)\ w_3'$ and $B(..a) \to w_1''\ C(..)\ w_3''$ are in P_L. Therefore, $\overline{H_3}$ holds since we have the push-pop derivation $A() \underset{l,L}{\Rightarrow} w_1'\ B(a)\ w_3' \underset{l,L}{\Rightarrow} w_1'\ w_1''\ C()\ w_3''\ w_3'$.

$\overline{H_4}$: Consider a terminal $\langle A,B\rangle_{\underset{+}{\overset{a}{\succ}}}$-derivation of length one in G $\langle A,B\rangle_{\underset{+}{\overset{a}{\succ}}}(\rho_1,\rho_3) \underset{G,w}{\Rightarrow} \varepsilon$ where $w^{\rho_1} = w_1$ and $w^{\rho_3} = w_3$. Such a derivation exists iff there is in P a clause of the form $\langle A,B\rangle_{\underset{+}{\overset{a}{\succ}}}(w_1,w_3) \to \varepsilon$. If it is the case, we know by (8) that this clause has been produced iff the production $A(..a) \to w_1\ B(..)\ w_3$ is in P_L. Therefore, $\overline{H_4}$ holds since we have the pop-a derivation $A(a) \underset{l,L}{\Rightarrow} w_1\ B()\ w_3$.

Induction step. We assume that for all derivations in G of length $1, 2, \ldots, l$ the $\overline{H_i}$'s hold, we will prove that they also hold for all derivations of length $l+1$. Of course, all derivations of length $l+1$ have at least an inside step.

$\overline{H_1}$: First we consider a terminal $\langle A\rangle$-derivation $d = \langle A\rangle(\rho) \overset{\vec{\Psi}_1}{\underset{G,w}{\Rightarrow}} \langle A,B\rangle_{\underset{+}{=}}(\rho_1,\rho_2) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$ in G of length $l+1$, where $\vec{\Psi}_1 = \langle A\rangle(Ww_2Z) \to \langle A,B\rangle_{\underset{+}{=}}(W,Z)$, $w_1 = w^{\rho_1}$, $w_3 = w^{\rho_2}$ and $w_1w_2w_3 = w^\rho$. If such a derivation exists, the clause $\vec{\Psi}_1$ has been produced in (2). This means that we have both $A \underset{+}{=} B$ and $B() \to w_2 \in P_L$. If we consider the suffix of d of length l $\langle A,B\rangle_{\underset{+}{=}}(\rho_1,\rho_2) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$,

this derivation shows that $(w_1, w_3) \in \mathcal{L}(\langle A, B\rangle_{\underset{+}{=}})$, and by the induction hypothesis we know that there is in L a balanced derivation $A() \overset{+}{\underset{l,L}{\Rightarrow}} w_1 \;\; B() \; w_3$ which gives the closed derivation $A() \overset{+}{\underset{l,L}{\Rightarrow}} w_1 \;\; B() \; w_3 \overset{B() \to w_2}{\underset{l,L}{\Rightarrow}} w_1 \;\; w_2 \; w_3$.

$\overline{H_3}$: We consider a terminal $\langle A, C\rangle_{\underset{+}{\diamond\!\!-\!\!\diamond}}$-derivation $d = \langle A, C\rangle_{\underset{+}{\diamond\!\!-\!\!\diamond}}(\rho_1, \rho_2) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$ of length $l+1$ where $w^{\rho_1} = w_1$ and $w^{\rho_3} = w_3$.
The first step of such a derivation uses $\vec{\Psi} = \langle A, C\rangle_{\underset{+}{\diamond\!\!-\!\!\diamond}}(\gamma_1 W, Z\gamma_2) \to \langle \Gamma_1\rangle \;\; \langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}}(W, Z) \; \langle \Gamma_2\rangle$ produced in (7), therefore we have $d = \langle A, C\rangle_{\underset{+}{\diamond\!\!-\!\!\diamond}}(\vec{\Omega}[0]) \overset{\vec{\Psi}}{\underset{G,w}{\Rightarrow}} \langle \Gamma_1\rangle(\vec{\Omega}[1]) \; \langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}}(\vec{\Omega}[2]) \; \langle \Gamma_3\rangle(\vec{\Omega}[3]) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$ where $\vec{\Omega}[0] = (\rho_1, \rho_2)$. Let $w^{\vec{\Omega}[1]}$ and $w^{\vec{\Omega}[3]}$ be respectively the strings w_1' and w_3'. Since the lengths of the subderivations in d starting at $\langle \Gamma_1\rangle(\vec{\Omega}[1])$ and $\langle \Gamma_3\rangle(\vec{\Omega}[3])$ are less or equal than l, we know by $\overline{H_1}$ that there are two closed derivations in L, $\Gamma_1 \overset{*}{\underset{l,L}{\Rightarrow}} w_1'$ and $\Gamma_2 \overset{*}{\underset{l,L}{\Rightarrow}} w_3'$. Moreover, the length of the subderivation of d starting at $\langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}}(\vec{\Omega}[2])$ is also less or equal to l. Therefore, if we assume that $(w_1'', w_3'') \in \mathcal{L}(\langle B, C\rangle_{\underset{+}{\overset{a}{\succ}}})$, we know by $\overline{H_4}$ that there is in L a pop-a derivation $B(a) \overset{+}{\underset{l,L}{\Rightarrow}} w_1'' \; C() \; w_3''$. At last, the clause $\vec{\Psi}$ is produced iff $A(..) \to \Gamma_1 \; B(..a) \; \Gamma_2 \in P_L$. We can therefore exibit a push-pop derivation in $L \; A() \underset{l,L}{\Rightarrow} \Gamma_1 \; B(a) \; \Gamma_2 \overset{*}{\underset{l,L}{\Rightarrow}} w_1' \; B(a) \; w_3' \overset{+}{\underset{l,L}{\Rightarrow}} w_1' \; w_1'' \; C() \; w_3'' \; w_3'$ where $w_1' \; w_1'' = w_1$ and $w_3'' \; w_3' = w_3$.

$\overline{H_2}$: Now we consider a terminal $\langle A, C\rangle_{\underset{+}{=}}$-derivation $d = \langle A, C\rangle_{\underset{+}{=}}(\rho_1, \rho_3) \overset{\vec{\Psi}}{\underset{G,w}{\Rightarrow}} \ldots \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$ of length $l+1$ where $w^{\rho_1} = w_1$, $w^{\rho_3} = w_3$ and $\vec{\Psi}$ is the clause produced in (3), (4), (5) or (6). We will not repeat the arguments concerning the fact that the induction hypothesis apply on the subderivations of d headed on each element of the instantiation of the RHS of $\vec{\Psi}$ by some $\vec{\Omega}$ with $\vec{\Omega}[0] = (\rho_1, \rho_3)$, since they are identical to the ones used in proving $\overline{H_3}$.

Case (3). We have $A(..) \to \Gamma_1 \;\; C(..) \; \Gamma_3 \in P_L$, $\Gamma_1 \overset{*}{\underset{l,L}{\Rightarrow}} w_1$ and $\Gamma_3 \overset{*}{\underset{l,L}{\Rightarrow}} w_3$. Therefore, we can exhibit a balanced derivation $A() \underset{l,L}{\Rightarrow} \Gamma_1 \; C() \; \Gamma_3 \overset{*}{\underset{l,L}{\Rightarrow}} w_1 \;\; C() \; w_3$.

Case (4). Here we simply indicate that a push-pop derivation is a particular case of balanced derivation and that our hypothesis extends from push-pop to balanced.

Case (5). We have $A(..) \to \Gamma_1 \; B(..) \; \Gamma_3 \in P_L$, $\Gamma_1 \overset{*}{\underset{l,L}{\Rightarrow}} w_1'$, $\Gamma_3 \overset{*}{\underset{l,L}{\Rightarrow}} w_3'$ and $B() \overset{+}{\underset{l,L}{\Rightarrow}} w_1'' \; C() \; w_3''$. Therefore, we can exhibit the balanced derivation $A() \underset{l,L}{\Rightarrow} \Gamma_1 \; B() \; \Gamma_2 \overset{*}{\underset{l,L}{\Rightarrow}} w_1' \; B() \; w_3' \overset{+}{\underset{l,L}{\Rightarrow}} w_1' \; w_1'' \; C() \; w_3'' \; w_3'$ where $w_1' \; w_1'' = w_1$ and $w_3'' \; w_3' = w_3$.

Case (6). We have $A() \overset{+}{\underset{l,L}{\Rightarrow}} w_1' \; B() \; w_3'$ and $B() \overset{+}{\underset{l,L}{\Rightarrow}} w_1'' \; C() \; w_3''$. Therefore, we get the balanced derivation $A() \underset{l,L}{\Rightarrow} \Gamma_1 \; B() \; \Gamma_2 \overset{*}{\underset{l,L}{\Rightarrow}} w_1' \; B() \; w_3' \overset{+}{\underset{l,L}{\Rightarrow}} w_1' \; w_1'' \; C() \; w_3'' \; w_3'$ where $w_1' \; w_1'' = w_1$ and $w_3'' \; w_3' = w_3$.

$\overline{H_4}$: Last, we consider a terminal $\langle A, C\rangle_{\underset{+}{\overset{a}{\succ}}}$-derivation $d = \langle A, C\rangle_{\underset{+}{\overset{a}{\succ}}}(\rho_1, \rho_2) \overset{\vec{\Psi}}{\underset{G,w}{\Rightarrow}} \ldots \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon$ of length $l+1$ where $w^{\rho_1} = w_1$, $w^{\rho_3} = w_3$ and $\vec{\Psi}$ is the clause produced in (8) or (9).

Case (8). We have $A(..a) \to \Gamma_1 \;\; C(..) \; \Gamma_3 \in P_L$, $\Gamma_1 \overset{*}{\underset{l,L}{\Rightarrow}} w_1$ and $\Gamma_3 \overset{*}{\underset{l,L}{\Rightarrow}} w_3$. Therefore, we can exhibit the pop-a derivation $A(a) \underset{l,L}{\Rightarrow} \Gamma_1 \; C() \; \Gamma_3 \overset{*}{\underset{l,L}{\Rightarrow}} w_1 \;\; C() \; w_3$.

Case (9). We have $B(..a) \to \Gamma_1 \; C(..) \; \Gamma_3 \in P_L$, $\Gamma_1 \overset{*}{\underset{l,L}{\Rightarrow}} w_1''$, $\Gamma_3 \overset{*}{\underset{l,L}{\Rightarrow}} w_3''$ and $A() \overset{+}{\underset{l,L}{\Rightarrow}} w_1' \; B() \; w_3'$. Of course from this last derivation, we can produce another derivation where each stack on the spine headed at $A()$ is prefixed by any string of symbols, and in particular by the

single symbol a, giving $A(a) \overset{+}{\underset{l,L}{\Rightarrow}} w'_1\ B(a)\ w'_3$. Therefore, we get the derivation $A(a) \overset{+}{\underset{l,L}{\Rightarrow}} w'_1\ B(a)\ w'_3 \underset{l,L}{\Rightarrow} w'_1\ \Gamma_1\ C()\ \Gamma_3\ w'_3 \overset{+}{\underset{l,L}{\Rightarrow}} w'_1\ w''_1\ C()\ w''_3\ w'_3$ where $w'_1\ w''_1 = w_1$ and $w''_3\ w'_3 = w_3$.

∎

Example 10 *We illustrate our transformation with a LIG* $L = (\{S,T\},\{a,b,c\},\{a,b,c\},P_L,S)$ *where* P_L *contains the following productions:*

$$\begin{array}{llll} S(..) \to S(..a)\ a & S(..) \to S(..b)\ b & S(..) \to S(..c)\ c & S(..) \to T(..) \\ T(..a) \to a\ T(..) & T(..b) \to b\ T(..) & T(..c) \to c\ T(..) & T() \to c \end{array}$$

It is easy to see that the language $\mathcal{L}(L)$ *is* $\{xcx \mid x \in \{a,b,c\}^*\}$.
We note that in L *the key part is played by the middle* c*, introduced by the production* $T() \to c$.
The computation of the relations gives:

$$\begin{array}{lllll} & & & \underset{1}{=} & = \{(S,T)\} \\ \overset{a}{\underset{1}{\prec}} & = & \overset{b}{\underset{1}{\prec}} & = \overset{c}{\underset{1}{\prec}} & = \{(S,S)\} \\ \overset{a}{\underset{1}{\succ}} & = & \overset{b}{\underset{1}{\succ}} & = \overset{c}{\underset{1}{\succ}} & = \{(T,T)\} \\ & & & \underset{+}{=} & = \{(S,T)\} \\ & & & \underset{+}{\lessdot\gtrdot} & = \{(S,T)\} \\ \overset{a}{\underset{+}{\prec}} & = & \overset{b}{\underset{+}{\prec}} & = \overset{c}{\underset{+}{\prec}} & = \{(T,T),(S,T)\} \end{array}$$

The production set P *of the PRCG* G *associated with* L *is:*

$$\begin{array}{lll} \langle S\rangle(XcY) & \to \langle S,T\rangle_{\underset{+}{=}}(X,Y) & (2) \\ \langle S,T\rangle_{\underset{+}{=}}(\varepsilon,\varepsilon) & \to \varepsilon & (3) \\ \langle S,T\rangle_{\underset{+}{=}}(X,Y) & \to \langle S,T\rangle_{\underset{+}{\lessdot\gtrdot}}(X,Y) & (4) \\ \langle S,T\rangle_{\underset{+}{\lessdot\gtrdot}}(W,Za) & \to \langle S,T\rangle_{\overset{a}{\underset{+}{\prec}}}(W,Z) & (7) \\ \langle S,T\rangle_{\underset{+}{\lessdot\gtrdot}}(W,Zb) & \to \langle S,T\rangle_{\overset{b}{\underset{+}{\prec}}}(W,Z) & (7) \\ \langle S,T\rangle_{\underset{+}{\lessdot\gtrdot}}(W,Zc) & \to \langle S,T\rangle_{\overset{c}{\underset{+}{\prec}}}(W,Z) & (7) \\ \langle S,T\rangle_{\overset{a}{\underset{+}{\succ}}}(Wa,Z) & \to \langle S,T\rangle_{\underset{+}{=}}(W,Z) & (9) \\ \langle S,T\rangle_{\overset{b}{\underset{+}{\succ}}}(Wb,Z) & \to \langle S,T\rangle_{\underset{+}{=}}(W,Z) & (9) \\ \langle S,T\rangle_{\overset{c}{\underset{+}{\succ}}}(Wc,Z) & \to \langle S,T\rangle_{\underset{+}{=}}(W,Z) & (9) \end{array}$$

The (i)*'s refer to the numbers used to label the various form of clauses produced during the proof of Property 11.*

We know, by Example 3, that 2-PRCGs in 2-var form which are not in simple form are much more powerful than LIGs.

8.2 Tree Adjoining Grammars & sPRCGs

Let $L = (V_N, V_T, \mathcal{I}, \mathcal{A}, S)$ be a Tree Adjoining Grammar (TAG).

V_N and V_T are finite disjoint sets of nonterminal and terminal symbols, $\mathcal{I}$ is the finite set of *initial trees* and $\mathcal{A}$ is the finite set of *auxiliary trees*.

Initial trees are such that their root is labeled by the start symbol S, inside nodes are labeled by nonterminals and each leaf is labeled by an element of $V_T^\varepsilon = V_T \cup \{\varepsilon\}$.

Auxiliary trees are such that their root is labeled by a nonterminal, say A, inside nodes are labeled by nonterminals, and each leaf is labeled by an element of V_T^ε, except for one leaf which is labeled by a nonterminal whose name is the root name (i.e. A). This particular leaf is called the *foot*. Such a tree is called an *auxiliary A-tree*. The path from root to foot is called its *spine*. We call *terminal tree* a non-trivial tree where every leaf is labeled by an element of V_T^ε. The string whose components are the symbols labeling the leaves of a tree and gathered by a left to right walk is called its *yield*.

A node whose label is B is an $\mathcal{A}$-node iff there is an auxiliary B-tree. For a given tree, the set of its inside nodes are partitionned in two: the $\mathcal{A}$-nodes and the others.

The *adjunct* operation is defined on trees. It takes as operands a terminal tree and an auxiliary tree and gives as result a terminal tree. Let τ be a terminal tree, n be an inside $\mathcal{A}$-node labeled by the nonterminal A and τ_1 an auxiliary A-tree. The adjunct operation is (operationally) defined as follows.

First note that the nonterminal A labels the node n, the root and the foot of τ_1. Excise the subtree τ' of τ occurring in n, replace it with a copy of τ_1, and graft the excised subtree τ' to the foot of this copy of τ_1.

The (string) language of a TAG is the set of the yields of terminal trees which are built by the adjunct operation from the initial trees.

We add the following two conditions which do not change the power of TAGs:

1. The start symbol only occurs as root of initial trees;
2. No adjunction can occur at the root or at the foot of an auxiliary tree.

Property 12 *For any TAG, there is an equivalent 2-sPRCG.*

The corresponding PRCG $G = (N, T, V, P, S)$ is such that: $T = V_T$ and their start symbols are identicals. Its nonterminals N, except for S, are the labels of the roots of the auxiliary trees. Its predicates are all binaries, except the start symbol S which is unary.

Let τ be a tree (initial or auxiliary), I_τ the set of its inside $\mathcal{A}$-nodes, and $|I_\tau|$ its cardinal. Let V_τ the set of $2|I_\tau|$ variables X_i, Y_i, $1 \leq i \leq |I_\tau|$. We assign to each internal $\mathcal{A}$-node a couple of variables X_i and Y_i. X_i is supposed to be a *left* decoration and Y_i a *right* decoration. The root and the foot (if any) have no decoration. Each terminal leaf $a \in V_T^\varepsilon$ has a single decoration which is itself.

If we consider the process of gathering decorations in a tree by a top-down left-to-right walk, we collect the left variable X_i when passing an $\mathcal{A}$-node top-down and the right variable Y_i, when passing this $\mathcal{A}$-node bottom-up. Therefore, for every tree τ, we collect a string $d_\tau \in (T \cup \{X_i \mid 1 \leq i \leq |I_\tau|\} \cup \{Y_i \mid 1 \leq i \leq |I_\tau|\})^*$. For an auxiliary tree τ, this string d_τ can be splitten in two parts d_τ^1 and d_τ^2 s.t. $d_\tau = d_\tau^1 d_\tau^2$ where d_τ^1 is the part gathered before the foot of τ and d_τ^2 is the part gathered after the foot.

To each initial or auxiliary tree in L, we associate a clause in P which is constructed in the following way.

The LHS of a clause associated with an initial tree τ is the predicate $S(d_\tau)$.

The LHS of a clause associated with an auxiliary A-tree τ is the predicate $A(d_\tau^1, d_\tau^2)$.

The RHS of a clause associated with a tree τ (initial or auxiliary) is a sequence of $|I_\tau|$ predicates $p_1 \ldots p_i \ldots p_{|I_\tau|}$, all differents, with $p_i = A_i(X_i, Y_i)$ where A_i is the label of the i^{th} $\mathcal{A}$-node in τ[11].

For each auxiliary A-tree we add the clause $A(\varepsilon, \varepsilon) \rightarrow \varepsilon$ (if not already there). The idea behind such clauses is to allow each predicate to be productive since we have assumed that adjunction is not mandatory in TAGs.

If we forget about the $A(\varepsilon, \varepsilon) \rightarrow \varepsilon$ clauses, there is a bijection between the trees in L and the clauses in G, that is also the case for their respective derivations. The proof of their equivalence is made easier by this property and is omitted here.

Example 11 *The TAG with $\mathcal{I} = \{S(A(\varepsilon))\}$ and $\mathcal{A} = \{A(a, A(b, A, c))\}$ defined the string language $\{a^n b^n c^n \mid n \geq 0\}$. The set of clauses of its corresponding 2-sPRCG is :*

$$\begin{array}{lcl} S(X_1 Y_1) & \rightarrow & A(X_1, Y_1) \\ A(aX_1 b, cY_1) & \rightarrow & A(X_1, Y_1) \\ A(\varepsilon, \varepsilon) & \rightarrow & \varepsilon \end{array}$$

[11] In fact, the order of the predicates in the RHS is irrelevant.

8.3 Head Grammars & sPRCGs

For completion reasons, we add this section which is mainly extracted from [Groenink 96], using our conventions.

Head Grammars (HG) were introduced in [Pollard 84] and can be viewed as a generalization of CFGs in which a wrapping operation is used in addition to concatenation. The nonterminals of a HG derive couples of terminal strings (v, w) denoted $v \uparrow w$. We restrict our attention to a weakly equivalent form of HGs, in which productions are *bilinear*.

Definition 17 *A (bilinear) head grammar is a 4-tuple $L = (N, T, P_L, S)$, as in a CFG, except that its productions have two forms*

$$\begin{array}{lcll} A & \rightarrow & w_1 \uparrow w_2 & \textit{(terminal form)} \\ A & \rightarrow & f(B_1, B_2) & \textit{(nonterminal form)} \end{array}$$

where $A, B_1, B_2 \in N$, $w_1, w_2 \in T^*$ and the yield function f is one of the function symbols wrap, $concat_1$ *or* $concat_2$.

On HGs, we define a derive relation $\underset{L}{\Rightarrow}$ by

- if $A \rightarrow w_1 \uparrow w_2$ is a production then $A \underset{L}{\Rightarrow} w_1 \uparrow w_2$;
- if $A \rightarrow f(B_1, B_2)$ is a production, $B_1 \overset{+}{\underset{L}{\Rightarrow}} v_1 \uparrow v_2$ and $B_2 \overset{+}{\underset{L}{\Rightarrow}} w_1 \uparrow w_2$, then $A \underset{L}{\Rightarrow} f(v_1 \uparrow v_2, w_1 \uparrow w_2)$ where

$$\begin{array}{lcl} wrap(v_1 \uparrow v_2, w_1 \uparrow w_2) & = & w_1 v_1 \uparrow v_2 w_2 \\ concat_1(v_1 \uparrow v_2, w_1 \uparrow w_2) & = & v_1 \uparrow v_2 w_1 w_2 \\ concat_2(v_1 \uparrow v_2, w_1 \uparrow w_2) & = & v_1 v_2 w_1 \uparrow w_2 \end{array}$$

and the language generated by L is $\mathcal{L}(L) = \{vw \mid S \overset{+}{\underset{L}{\Rightarrow}} v \uparrow w\}$.

We can easily check that in applying the following translation from P_L into clauses

$$\begin{array}{lcl@{\quad}lcl} A & \rightarrow & w_1 \uparrow w_2 & \Longrightarrow & A(w_1, w_2) & \rightarrow & \varepsilon \\ A & \rightarrow & wrap(B_1, B_2) & \Longrightarrow & A(WX, YZ) & \rightarrow & B_1(X, Y)\ B_2(W, Z) \\ A & \rightarrow & concat_1(B_1, B_2) & \Longrightarrow & A(X, YWZ) & \rightarrow & B_1(X, Y)\ B_2(W, Z) \\ A & \rightarrow & concat_2(B_1, B_2) & \Longrightarrow & A(XYW, Z) & \rightarrow & B_1(X, Y)\ B_2(W, Z) \end{array}$$

and adding the clause $S'(XY) \rightarrow S(X, Y)$, where S' is a new nonterminal (it is the start symbol of G), we get a 2-sPRCG.

Property 13 *For any HG, there is an equivalent 2-sPRCG.*

A proof of this property can easily be based upon the bijection (if we forget about the $S'(XY) \rightarrow S(X, Y)$ clause) between the productions of the HG and the clauses of the corresponding 2-sPRCG.

Example 12 *The HG $L = (\{S, T, AC, B\}, \{a, b, c\}, P_L, S)$ where*

$$\begin{array}{l} P_L = \{ \\ \qquad \begin{array}{lcl} S & \rightarrow & \varepsilon \uparrow \varepsilon, \\ S & \rightarrow & wrap(B, T), \\ B & \rightarrow & b \uparrow \varepsilon, \\ T & \rightarrow & wrap(S, AC), \\ AC & \rightarrow & a \uparrow c \end{array} \\ \} \end{array}$$

generates the language $\mathcal{L}(L) = \{a^n b^n c^n \mid n \geq 0\}$.

By the previous transformation we get the equivalent set of clauses

$$\begin{array}{l} P = \{ \\ \qquad \begin{array}{lcl} S'(XY) & \rightarrow & S(X, Y), \\ S(\varepsilon, \varepsilon) & \rightarrow & \varepsilon, \\ S(WX, YZ) & \rightarrow & B(X, Y)\ \ T(W, Z), \\ B(b, \varepsilon) & \rightarrow & \varepsilon, \\ T(WX, YZ) & \rightarrow & S(X, Y)\ \ AC(W, Z), \\ AC(a, c) & \rightarrow & \varepsilon \end{array} \\ \} \end{array}$$

9 Coupled Context-Free Grammars & sPRCGs

Coupled context-free grammars (CCFGs) are a generalization of CFGs (see for example [Hotz and Pitsch 94]). In this formalism, nonterminals are simultaneously substituted if they correspond to a (correctly nested) system of parentheses. If we look at the generative capacity of CCFGs, we obtain an infinite hierarchy of languages, characterized by an integer number, its *rank* (the rank of a CCFG, is the number of elements rewritten simultaneously), whose first of rank 1 are the CFLs and the second of rank 2 are the languages generated by TAGs, LIGs or HGs.

CCFGs are defined over extended semi-Dyck sets which are a generalization of semi-Dyck sets. Elements of these sets can be regarded as sequences of parentheses which are correctly nested.

$\mathcal{K} = \{(k_i^1, \ldots, k_i^j, \ldots, k_i^{m_i}) \mid i, j, m_i \in \mathbb{N}\}$ is a set of *tuple of parentheses* if it is finite and if it satisfies $k_i^j \neq k_l^m$ for $i \neq l$ or $j \neq m$. The set of parentheses is $comp(\mathcal{K}) = \{k_i \mid (k_1, \ldots, k_i, \ldots, k_r) \in \mathcal{K}\}$.

The sets $\mathcal{K}[r] = \{(k_i^1, \ldots, k_i^j, \ldots, k_i^{m_i}) \mid m_i = r\}$ contain the tuple of parentheses of length r. We assume that $\mathcal{K}[0] = \{\varepsilon\}$.

Let $\mathcal{K}$ be a set of tuples of parentheses and T be an arbitrary set where $T \cap \mathcal{K} = T \cap comp(\mathcal{K}) = \emptyset$. The *extended semi-Dyck set over* $\mathcal{K}$ *and* T, $ED(\mathcal{K}, T)$ is defined inductively by:

1. $T^* \subseteq ED(\mathcal{K}, T)$
2. $\mathcal{K}[1] \subseteq ED(\mathcal{K}, T)$
3. $u_1, \ldots, u_r \in ED(\mathcal{K}, T), (k_1, \ldots, k_{r+1}) \in \mathcal{K}[r+1] \Longrightarrow k_1 u_1 \ldots k_r u_r k_{r+1} \in ED(\mathcal{K}, T)$
4. $u, v \in ED(\mathcal{K}, T) \Longrightarrow uv \in ED(\mathcal{K}, T)$
5. $ED(\mathcal{K}, T)$ is the smallest set fulfilling conditions 1) to 4).

A *parentheses rewriting system over* $ED(\mathcal{K}, T)$ is a finite, non empty set P of productions of the form

$$\{(k_1, \ldots, k_r) \rightarrow (\alpha_1, \ldots, \alpha_r) \mid (k_1, \ldots, k_r) \in \mathcal{K} \wedge \alpha_1 \ldots \alpha_r \in ED(\mathcal{K}, T)\}$$

Definition 18 *A CCFG is an ordered 4-tuple* $(\mathcal{K}, T, P, S)$ *where* $\mathcal{K}$ *is a parentheses set and* P *is a parentheses rewriting system over* $ED(\mathcal{K}, T)$ *and* $S \in \mathcal{K}[1]$.

The term *coupled* expresses that, at each step, some CF rewritings must be performed in parallel and are controlled by $\mathcal{K}$. Therefore $\mathcal{K}$ may be seen as a set of coupled nonterminals.

Let $G_c = (\mathcal{K}, T, P_c, S)$ be a CCFG, we define on $(comp(\mathcal{K}) \cup T)^*$ the *derive* relation denoted $\underset{G_c}{\Rightarrow}$: $\Phi \underset{G_c}{\Rightarrow} \Psi$ holds for $\Phi, \Psi \in (comp(\mathcal{K}) \cup T)^*$, iff there exist $u_1, u_{r+1} \in V^*$, $u_2, \ldots, u_r \in ED(\mathcal{K}, T)$ and $(k_1, \ldots, k_r) \rightarrow (\alpha_1, \ldots, \alpha_r) \in P_c$ such that

$$\begin{aligned} \Phi &= u_1 k_1 u_2 k_2 \ldots u_r k_r u_{r+1} \\ \Psi &= u_1 \alpha_1 u_2 \alpha_2 \ldots u_r \alpha_r u_{r+1} \end{aligned}$$

The language defined by G_c is

$$\mathcal{L}(G_c) = \{w \mid S \underset{G_c}{\overset{+}{\Rightarrow}} w \wedge w \in T^*\}$$

For any CCFG $G_c = (\mathcal{K}, T, P_c, S)$, we define the *rank* of G as $\text{rank}(G) = \max\{r \mid (k_1, \ldots, k_r) \in \mathcal{K}\}$.

Property 14 *For any CCFG of rank* h*, there is an equivalent* h*-sPRCG.*

For any CCFG $G_c = (\mathcal{K}, T, P_c, S)$, we construct a PRCG $G = (\mathcal{K}, T, V, P, S)$ in the following way. Let $(k_1, \ldots, k_r) \rightarrow (\alpha_1, \ldots, \alpha_r)$ be a production in P_c where $(k_1, \ldots, k_r) \in \mathcal{K}$ and $\alpha_1 \ldots \alpha_r \in ED(\mathcal{K}, T)$ such that $\alpha_1 = w_0^1 k^1 w_1^1 \ldots k^{s_1} w_{p_1}^1$, $\alpha_i = w_0^i k^{1+s_i} w_1^i \ldots k^{p_i+s_i} w_{p_i}^i$ and $\alpha_r = w_0^r k^{1+s_r} w_1^r \ldots k^{p_i+s_r} w_{p_r}^r$ where $s_1 = p_1$ and $s_k = \sum_{1 \leq j < k} p_j, 2 \leq k \leq r$.

We successively defined the LHS and the RHS of the corresponding clause

The LHS part. The nonterminal part of the LHS is $(k_1, \ldots, k_r)$ and its arguments are merely a rewriting of the RHS of the production where each occurrence of a parenthesis element is replaced by a brand new variable. More formally, let replace each occurrence of a $comp(\mathcal{K})$ element in all α_i's (e.g. the k^l's) by a variable X_l and therefore transforming each α_i into $\langle\alpha_i\rangle$ such that $\langle\alpha_1\rangle = w_0^1 X_1 w_1^1 \ldots X_{p_1} w_{p_1}^1$, ..., $\langle\alpha_i\rangle = w_0^i X_{1+s_i} w_1^i \ldots X_{p_i+s_i} w_{p_i}^i$, ..., $\langle\alpha_r\rangle = w_0^r X_{1+s_r} w_1^r \ldots X_{p_i+s_r} w_{p_r}^r$. The LHS of the corresponding clause in P is therefore: $(k_1, \ldots, k_r)(\langle\alpha_1\rangle, \ldots, \langle\alpha_r\rangle)$. Remark that the length r of the parentheses becomes the arity of the nonterminal $(k_1, \ldots, k_r)$.

The RHS part. The RHS of the clause is constructed from the α_i's and only depends upon the parentheses and their positions in the RHS of the initial production. Consider the string of parentheses $\kappa = k^1 \dots k^{s_1} k^{s_1+1} \dots k^{s_2} \dots k^{s_{r-1}+1} \dots k^{s_r}$ constructed from $\alpha_1 \dots \alpha_r$, where all the elements of T (the w's) have been erased. Of course, we still have $\kappa \in ED(\mathcal{K}, T)$. This sequence of parentheses, κ, may (multiply) be seen as a string $u_0 k_1 u_1 \dots u_{i-1} k_i u_i \dots k_m u_m$ where $u_i \in ED(\mathcal{K}, T)$ and $(k_1, \dots, k_m) \in \mathcal{K}[m]$. To each parenthesis k_i we associate its position $p_i = |u_0 k_1 u_1 \dots u_{i-1}| + 1$ in κ. We define a set $D = \{(k_1, \dots, k_j, \dots, k_m)(Y_1, \dots, Y_j, \dots, Y_m) \mid \kappa = u_0 k_1 u_1 \dots k_j u_j \dots k_m u_m \wedge u_j \in ED(\mathcal{K}, T) \wedge (k_1, \dots, k_m) \in \mathcal{K}[m] \wedge Y_j = X_{p_j}\}$[12]. If, as usual, we look at this set as a vector $\vec{D}$, the RHS of the constructed clause is $\vec{D}$.

Proof: By construction, each clause in P is in simple form and its arity is at most h. The bijections between the productions in P_c and the clauses in P and between the derivations in G_c and G allow to easily show this result.

∎

Example 13 *The CCFG* $G_c = (\{(S), (L, R)\}, \{a, b, c\}, P_c, (S))$ *where the productions in* P_c *are*

$$\begin{array}{lcl} (S) & \rightarrow & (L, R) \\ (L, R) & \rightarrow & (aLb, cR) \\ (L, R) & \rightarrow & (\varepsilon, \varepsilon) \end{array}$$

defines the language $\mathcal{L}(G_c) = \{a^n b^n c^n \mid n \geq 0\}$.
The associated PRCG is $G = (\{(S), (L, R)\}, \{a, b, c\}, \{X_1, X_2\}, P, (S))$ *where the clauses in* P *are*

$$\begin{array}{lcl} (S)(X_1X_2) & \rightarrow & (L, R)(X_1, X_2) \\ (L, R)(aX_1b, cX_2) & \rightarrow & (L, R)(X_1, X_2) \\ (L, R)(\varepsilon, \varepsilon) & \rightarrow & \varepsilon \end{array}$$

10 Linear Context-Free Rewriting Systems & sPRCGs

Linear context-free rewriting systems (LCFRS) have been introduced in [Weir 88] and are a generalization of HGs since they allow nonterminals to derive tuples (and not only couples) of terminal strings and instead of the wrapping and concatenation operator, they allow any operator which is linear and non-erasing. A production in a LCFRS has the form

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

where f is a function over tuples of terminal strings defined by

$$f(\vec{x_1}, \dots, \vec{x_m}) = \vec{t}$$

where the $x_{ij} = \vec{x_i}[j]$'s are variables and the $\vec{t}[k]$'s are terms (by the concatenation operation) over terminals and variables of the LHS. The linearity (both top-down and bottom-up), the non-erasingness (both top-down and bottom-up) and the fact that the $\vec{x_i}[j]$'s are variables (non-combinatorial) are exactly, in the RCG terminology, a simple clause

$$A(\vec{t}) \rightarrow B_1(\vec{x_1}) \dots B_m(\vec{x_m})$$

therefore, we trivially have

Property 15 *LCFRS and simple PRCGs are equivalent.*

[12] The RHS of a production in a CCFG is called its *drain*.

11 Literal Movement Grammars & PRCGs

As already mentioned, PRCGs are a variant of literal movement grammars (LMGs) see [Groenink 97]. The string version of generic LMGs may be seen either as a mini-prolog acting on a flat domain or as PRCGs where variables are not bound to ranges but to strings. As such, LMGs are not specialized to parsing, in the sense that some predicate arguments may be instantiated in strings which are not substrings of its sentences. At the contrary, in RCGs, the string instantiation of an argument w^ρ, for some range $\rho \in \mathcal{R}_w$, is by definition a substring of the input w. Therefore, generic LMGs could better be considered as a string manipulation formalism. In [Groenink 97] it is shown that the language defined by arbitrary LMG is the set of recursively enumerable languages. Therefore, to get a computationally tractable formalism, Groenink defined a subclass called *simple* LMGs in which clauses are non-combinatorial, bottom-up non-erasing and bottom-up linear. In fact the idea behind the restrictions from generic LMGs to simple LMGs are set to only handle substrings of the input in instantiated clauses. Within these restrictions, it can be shown that two occurrences of the same variable (in RHS of a clause), designate identical substrings occurring at the same index of the input. This exactly covers the notion of range. In order to implement LMG recognizers, Groenink defined a variant of simple LMGs called index LMGs which handles indexes (in the source text) instead of strings. He has shown that simple LMGs and index LMGs are weakly equivalent and that they exactly cover the class *PTIME* of languages recognizable in deterministic polynomial time. It is not very difficult to show that we have the following inclusions

simple LMG $\subseteq$ PRCG $\subseteq$ index LMG = simple LMG

This shows that formally PRCGs exactly cover the class *PTIME*.

In fact PRCG formalism combines both the advantages of simple LMGs and index LMGs. Moreover, the fact that RCG clauses are not forced to be non-combinatorial, bottom-up non-erasing or bottom-up linear, may add some flexibility in grammar design. However it does not seem that this increased flexibility, even when we consider negative predicates, had any formal power. The argument being that recognition is still performed in polynomial time for (general) RCGs.

There is a strict subclass of simple LMGs which is of interest, namely the parallel multiple context-free grammars (PMCFGs, [Kaji et al. 92]) which themselves strictly extend LCFRS in allowing only rules which are non-combinatorial, top-down non-erasing and top-down linear.

Therefore, we get the following hierarchy, w.r.t. their formal power

LCFRS $\subset$ PMCFG $\subset$ simple LMG = PRCG $\subset$ LMG

12 Closure Properties and Modularity

RCGs possess many interesting closure properties, some are classical (let's quote union, concatenation, Kleene-closure, ...), some are less classical (k-copy, $*$-copy, ...) and the two last (intersection and complementation) are the basis for our argumentation in favor of the modularity of this formalism.

Property 16 *RCLs are closed under k-copy and $*$-copy.*

The *k-copy* language of $\mathcal{L}$ for a given k is the set $\{w^k \mid w \in \mathcal{L}\}$. Let $G_1 = (N, T, V, P, S_1)$ be a RCG, the k-copy language of $\mathcal{L}(G_1)$ is defined by the RCG $G_k = (N \cup \{S_k, Eq\}, T, V \cup \{X_1, \ldots, X_k\}, P \cup \{\vec{\Psi}\}, S_k)$ where the start symbol S_k is a new nonterminal, Eq is the equality predicate defined in Example 2 and

$$\vec{\Psi} \; : \quad S_k(X_1 \ldots X_k) \quad \rightarrow \quad Eq(X_1, X_2) \ldots Eq(X_1, X_k) \;\; S_1(X_1)$$

For the $*$-copy language ($\cup_{k \geq 0} \mathcal{L}(G_k)$), we add to P the following set of clauses:

$$\begin{array}{lcl} S(\varepsilon) & \rightarrow & \varepsilon \\ S(X) & \rightarrow & S_1(X) \\ S(XYZ) & \rightarrow & Eq(X,Y)\; S_1(X)\; K(X,Z) \\ K(X,\varepsilon) & \rightarrow & \varepsilon \\ K(X,YZ) & \rightarrow & Eq(X,Y)\; K(X,Z) \end{array}$$

where K is a new nonterminal.

Property 17 *RCLs are closed under scrambled-copy.*

The *scrambled-copy* language of $\mathcal{L}$ is the set $\tilde{\mathcal{L}} = \{w \mid w = w_1 w_2 \wedge w_1 \in \mathcal{L} \wedge w_2 \in \Pi(w_1)\}$, where $\Pi(w_1)$ are the set of permutations of w_1. Assume that $\mathcal{L}$ is described by the RCG $G = (N, T, V, P, S)$, $\tilde{\mathcal{L}}$ is defined by a RCG whose start symbol is $\tilde{S}$ and the following clauses are added to P

$$
\begin{array}{lcl}
\tilde{S}(W_1 W_2) & \rightarrow & S(W_1)\ Scrmbl(W_1, W_1, W_2) \\
Scrmbl(T_0 X, W_1, W_2) & \rightarrow & Term(T_0)\ A(T_0, W_1, W_2)\ Scrmbl(X, W_1, W_2) \\
Scrmbl(\varepsilon, W_1, W_2) & \rightarrow & \varepsilon \\
A(T_0, T_1 W_1, T_2 W_2) & \rightarrow & Eq(T_0, T_1)\ Eq(T_0, T_2)\ A(T_0, W_1, W_2) \\
A(T_0, T_1 W_1, T_2 W_2) & \rightarrow & Eq(T_0, T_1)\ \overline{Eq}(T_0, T_2)\ A(T_0, T_1 W_1, W_2) \\
A(T_0, T_1 W_1, T_2 W_2) & \rightarrow & \overline{Eq}(T_0, T_1)\ Eq(T_0, T_2)\ A(T_0, W_1, T_2 W_2) \\
A(T_0, T_1 W_1, T_2 W_2) & \rightarrow & \overline{Eq}(T_0, T_1)\ \overline{Eq}(T_0, T_2)\ A(T_0, W_1, W_2) \\
A(T_0, \varepsilon, \varepsilon) & \rightarrow & \varepsilon
\end{array}
$$

where we assume that the predicate $Term$ checks if its argument is a terminal symbol. The first clause checks that the prefix w_1 of the input string $w = w_1 w_2$ is in L_1 and that its suffix w_2 is a scrambling of w_1. The idea behind the scrambling test is simply to check that the numbers of occurrences of each terminal symbol are equal in w_1 and w_2.

However, it is not clear how this closure property can be related to (long-distance) scrambling since we need to have some model (the string w_1) to be able to check for a permutation (the string w_2).

Property 18 *RCLs are closed under intersection.*

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 respectively defining the languages L_1 and L_2. Since we may rename nonterminals at will without changing the generated language we assume that $N_1 \cap N_2 = \emptyset$. Let S be a new nonterminal not in $N_1 \cup N_2$.

We construct the grammar $G = (N, T, V, P, S)$ where $N = N_1 \cup N_2 \cup \{S\}$, $T = T_1 \cup T_2$, $V = V_1 \cup V_2 \cup \{X\}$ and $P = P_1 \cup P_2 \cup \{S(X) \rightarrow S_1(X)\ \ S_2(X)\}$.

Trivially we have $\mathcal{L}(G) = L_1 \cap L_2$.

The fact that RCGs are closed under intersection and that such an intersection can be reached by a trivial operation (i.e. add a new clause after perhaps some nonterminal renaming) opens some interesting perspective in the ease and modular description of linguistic phenomena: it allows the direct definition of a conjunction of structures. One view of a given structural (grammatical) phenomenon is expressed by a first grammar an other view of the same phenomenon is given by a second grammar, the device which describes simultaneously the two previous views is merely the intersection of these two grammars.

Property 19 *The positive and negative range languages of nonterminals are complementary.*

The proof that

1. If $\vec{\rho_0} \in \Lambda(A) \Longrightarrow \vec{\rho_0} \notin \Lambda(\overline{A})$
2. If $\vec{\rho_0} \notin \Lambda(\overline{A}) \Longrightarrow \vec{\rho_0} \in \Lambda(A)$

directly results from the fact that derive relations are assumed to be consistent.
therefore, we have

Property 20 *The RCLs are closed under complementation.*

If $G = (N, T, V, P, S)$ is a RCG, the grammar $G' = (N \cup \{S'\}, T, V \cup \{X\}, P \cup \{S'(X) \rightarrow \overline{S}(X)\}, S')$ is a RCG s.t. we have $\mathcal{L}(S') = \mathcal{L}(\overline{S})$.

The reason why we are interested by this property is to model the paradigm "general rule with exceptions". The general rule is described by some predicate say R, the exceptions to this rule by an other predicate say E the whole property P being described by $R - E$ (i.e. $R \cap \overline{E}$).

Within the RCG formalism, we simply have to add a clause of the form

$$P(\vec{\alpha}) \quad \rightarrow R(\vec{\alpha})\ \overline{E}(\vec{\alpha})$$

We know that the complementary of a CFL is not CF. However when you give an erroneous input string to a general CF parser, this parser will always detect an error. More generally, if we have a parser for a given formalism which terminates on any input, this algorithm will recognize any erroneous input (i.e. string belonging

to the complementary), though the complementary language may not belong to the same class. But doing so, we get a recognizer (not a parser) for the complementary language. However, within the vision "general rule with exceptions", the structure (parse tree) of the complementary (negative) parts of RCGs are of no importance since if P is true (i.e. R is true and E is false), the structure of P is the structure of R (the general case).

We think that modularity[13] w.r.t. the intersection and complementation operations brings a new insight into the way syntactic backbones of NLs can be designed.

Example 14 *Assume you 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 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 our problem (if ever possible). Contrary, 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.

13 A Parsing Algorithm for RCGs

In a first time we will only consider a recognizer for positive RCGs and second how this algorithm can be generalized to also handle negative RCGs. Afterwards, our recognizer will be extended into a parser.

13.1 The positive RCG case

The recognizer is in Table 1. The function *prdct* must be called from the outside, for some input string w by $prdct(S, \bullet w \bullet)$.

```
(1)   function clause (i, ρ⃗0) return boolean
(2)     if K[i, ρ⃗0] ≠ unset then return K[i, ρ⃗0]
(3)     let P⃗[i] : A0(α⃗0) → A1(α⃗1) ... Am(α⃗m)
(4)     ret-val := K[i, ρ⃗0] := false
(5)     foreach Ω⃗ such that Ω⃗[0] = ρ⃗0 ∧ P⃗[i]/Ω⃗ do
(6)       ret-val := ret-val ∨ prdct (A1, Ω⃗[1]) ∧ ... ∧ prdct (Am, Ω⃗[m])
(7)     end foreach
(8)   return K[i, ρ⃗0] := ret-val
(9)   end function

(10)  function prdct (A, ρ⃗) return boolean
(11)    ret-val := false
(12)    foreach i such that P⃗[i] ∈ PA do
(13)      ret-val := ret-val ∨ clause (i, ρ⃗)
(14)    end foreach
(15)    return ret-val
(16)  end function
```

Table 1: The Recognition Algorithm for PRCGs.

[13] Recall that to be modular we need both a closure and a structure preserving property.

First remark that the function *clause* is memoized, its return value is kept in an auxiliary 2-dimensional matrix K. The elements of K are initialized to **unset**. As usual, $\vec{P}$ designates the clauses of the PRCG at hand.

We will show that this set of two recursive functions terminates on each finite input w of length n. In such a case the number of different values for the arguments of *clause* or *prdct* is bounded (it is $\mathcal{O}(n^{2h})$ if the arity of the PRCG is h). Therefore, an infinite loop can only occur if the same function is called with the same argument values. This case is avoided since first *clause* is memoized (see line (8)) and its body is executed only once when the memoized value is **unset** (see lines (2) and (4)) and second this memoization extends to *prdct* at a bounded extra cost ($\mathrm{Max}_{A\in N}|P_A|$). The interpretation of the three memoized values for a given couple of arguments $(i,\vec{\rho_0})$ is

unset. These arguments are considered for the first time, execute the body.

false. *clause* $(i,\vec{\rho_0})$ has already been called but here two cases can occur

1. This value has been set at line (4), the stack of recursive calls already contains the same call (the PRCG is cyclic) and a reexecution must be avoided. Moreover, up to the current knowledge, the value is **false**, meaning that, for the time being, no terminal derivation starting at $A(\vec{\rho_0})$ and whose first step uses the clause $\vec{P}[i]$ has been found.
2. This value has been over written at line (8), and the meaning is a total failure, no terminal derivation starting at $A(\vec{\rho_0})$ and whose first step uses the clause $\vec{P}[i]$ exists.

true. This value has been over written at line (8), and the meaning is a success: there is at least one terminal derivation starting at $A(\vec{\rho_0})$ and whose first step uses the clause $\vec{P}[i]$.

A call to *prdct* $(A,\vec{\rho})$ returns true iff there is at least one terminal derivation starting at $A(\vec{\rho})$. A call to *clause* $(i,\vec{\rho_0})$ returns true iff there is at least one clause binding $\vec{P}[i]/\vec{\Omega}$ such that $\vec{\Omega}[0]=\vec{\rho_0}$ and for which each instantiated predicate $A_k(\vec{\Omega}[k])$ of the RHS of the instantiated clause $A_0(\vec{\Omega}[0])\rightarrow A_1(\vec{\Omega}[1])\ldots A_m(\vec{\Omega}[m])$ is the head of at least one terminal derivation.

13.2 The negative RCG case

In order to get a recognition algorithm which also works for negative RCGs, we simply modify the line (6) in changing every call to $prdct(A_k,\vec{\Omega}[k])$ by $\neg\ prdct(A_k,\vec{\Omega}[k])$ when the corresponding predicate $\overline{A_k}(\vec{\alpha_k})$ in the RHS of $\vec{P}[i]$ is negative.

13.3 The parser case

To turn this recognition algorithm into a parser, we simply add the statement

(6') $\qquad$ **output**$(A_0^{\vec{\rho_0}}\rightarrow A_1^{\vec{\Omega}[1]}\ldots A_m^{\vec{\Omega}[m]})$

after line (6) which must be executed only if the call to $prdct\ (A_1,\vec{\Omega}[1])\wedge\ldots\wedge prdct\ (A_m,\vec{\Omega}[m])$ succeeds.

If we are in the negative case, say $\overline{A_k}(\vec{\alpha_k})$, we know that the call to $prdct(A_k,\vec{\Omega}[k])$ failed and its structure is empty, so is the structure of its complement. Therefore $\overline{A_k}^{\vec{\Omega}[k]}$ must be considered as a leaf. Equivalently we can output an additional $\overline{A_k}^{\vec{\Omega}[k]}\rightarrow\varepsilon$ production or even erase the negative nonterminals in the RHS of the production output at line (6').

13.4 Complexity considerations

We first compute the maximum number of $\vec{\Omega}$'s at line (5) for a given couple of arguments $(i,\vec{\rho_0})$. The number of ranges in the RHS is $a_i^r=\sum_{k=1}^{m}Arity(A_k)$ where $Arity(A_k)$ is the arity of the nonterminal A_k. Since each range is a couple of integers, line (6) is executed $\mathcal{O}(n^{2a_i^r})$. The function $clause(i,\vec{\rho_0})$ is executed $\mathcal{O}(n^{2a_i^l})$ where a_i^l is the arity of the LHS ($a_i^l=Arity(A_0)$). Therefore for a given clause $\vec{P}[i]$, the line (6) is executed $\mathcal{O}(n^{2a_i})$ times where $a_i=a_i^l+a_i^r$ is the arity of $\vec{P}[i]$. The maximum value of a_i is $h(l+1)$ if we consider an h-RCG and if l is the maximum number of predicates in RHS of clauses. The maximum time complexity of our parser is therefore $\mathcal{O}(|P|n^{2h(l+1)})$. This complexity is also the size complexity of the output shared forest.

Note that these complexities are linear in the size of the grammar. Such a result is of importance: let's quote from [Hotz and Pitsch 94] *When analyzing natural languages one often has to deal with rather short sentences while the underlying grammar is one of an enormous size. Therefore, the time complexity's dependency on the size of the grammar is important. Even for large ranks, here, the time complexity depends only linear on the size of the grammar. In contrast to our algorithm, all parsing procedures for TAGs and Coupled-Context-Free Grammars of rank 2, respectively, known so far depend at least quadratic on this size as their time complexity.*

13.4.1 The non-combinatorial and bottom-up non-erasing case

However, this previous view is very pessimistic since we have assumed a complete independence between the LHS and the RHS ranges. Assume that our RCG is non-combinatorial and bottom-up non-erasing, that is arguments of the RHS are variables which occur in the LHS. Let v_α be the number defined by

$$v_\alpha = \begin{cases} 0 & \text{if } \alpha \in T^* \\ k+1 & \text{if } \alpha = x_0 X_1 \dots X_k x_k \end{cases}$$

$v_{\vec{\alpha_0}} = \sum_i v_{\vec{\alpha_0}[i]}$ and $v = \text{Max}\, v_{\vec{\alpha_0}}$ where $\vec{\alpha_0}$ denotes the arguments of the LHS predicates. Under that conditions, the time and size complexity of our parser is $\mathcal{O}(|P|n^v)$.

In establishing this complexity, we have assumed that

1. different variables carry unrelated ranges, and that
2. the two bounds of each range are also unrelated.

We shall see below that in some particular cases this pessimistic view can be enhanced, but in a first time, we shall look at some practical improvements whose purpose is to decrease the number of time the loop at line (5) of Table 1 is executed. The idea is to statically compute (at grammar compilation time) two functions *First* and *Last* whose purpose is to record for each argument of every predicate the prefix and the suffix of the strings which can be derived. These functions can be seen as a generalization of the LL conditions in deterministic CF parsing, the length of these prefix and suffix can be increased at will. We will examine the length one case.

13.4.2 First & Last

We define the function *First (resp. Last)* in $N \times \mathbb{N} \mapsto \mathcal{P}(T \cup \{\varepsilon\})$ by $t \in First(A,k)$ (resp. $t \in Last(A,k)$) iff $S(\bullet w \bullet) \underset{G,w}{\overset{*}{\Rightarrow}} \Gamma_1\ A(\rho_1,\dots,\rho_k,\dots,\rho_p)\ \Gamma_2 \underset{G,w}{\overset{+}{\Rightarrow}} \varepsilon$ where $\rho_k = \langle i..j\rangle_w \in \mathcal{R}_w \wedge (i \neq j \wedge a_{i+1} = t$ (resp. $a_j = t) \vee i = j \wedge t = \varepsilon)$.

Algorithm 3 *We assume that G is non-combinatorial and non-erasing. Its clauses are denoted by $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1})\dots A_j(\vec{\alpha_j})\dots A_m(\vec{\alpha_m})$.*

We compute the 2-dimensional array First (resp. Last).

1. *We assume that the set Empty (see Algorithm 1) has been computed.*
2. *$\forall A,k : First[A,k] = \emptyset$ (resp. $Last[A,k] = \emptyset$).*
3. *if $\vec{\alpha_0}[k] = a\alpha'$ (resp. $\vec{\alpha_0}[k] = \alpha' a$), $a \in T$, add a to $First[A,k]$ (resp. to $Last[A,k]$).*
4. *if $\vec{\alpha_0}[k] = X_1 \dots X_h \dots X_q Y \alpha'$ (resp. $\vec{\alpha_0}[k] = \alpha' Y X_1 \dots X_h \dots X_q$) and for each $X_h, 1 \leq h \leq q$ there is an argument in the RHS say $\vec{\alpha_j}[l]$ such that $\vec{\alpha_j}[l] = X_h$, $(A_j, \vec{i_j}) \in Empty$ and $\vec{i_j}[l] = 0$ then*
   (a) *If $Y = a$, we add a to $First[A,k]$ (resp. to $Last[A,k]$).*
   (b) *If $Y \in V$, and if Y is the say l^{th} argument of $A_j, j > 0$, we unioned $First[A,k]$ (resp. $Last[A,k]$) with $First[A_j,l]$ (resp. $Last[A_j,l]$)*[14].
5. *Step 3) is iterated until stability.*
6. *For each $(A,\vec{i}) \in Empty$ and $\vec{i}[k] = 0$, we add ε to $First[A,k]$ (resp. to $Last[A,k]$).*

[14] If G is erasing and Y is a variable which does not occur in the RHS of $\vec{\Psi}$, we set $First[A,k]$ (resp. $Last[A,k]$) to T. Such a conservative approach may also be considered at places where clauses are combinatorial.

Property 21 $First(A,k) \subseteq First[A,k]$

Note that if $First[A,k] = \emptyset$ or $Last[A,k] = \emptyset$, this means that the A-predicates are useless.

Assume that the arrays $First$ and $Last$ have been (statically) computed, then we can activate the body of the loop at step (5) of the algorithm in Table 1 only when each range ρ in $\vec{\Omega}$ verifies the $First$ and $Last$ constraints: assume $\rho = \vec{\Omega}[j][k] = \langle h..i \rangle_w$, then we must have $a_{h+1} \in First[A_j,k]$ and $a_i \in Last[A_j,k]$.

Another statically computed property which may decrease the complexity of our parsing algorithm implies predicates whose multiple arguments have related sizes.

13.4.3 Compare

The idea here is to statically know whether the sizes of at least two arguments of a given predicate always verify some length constraints. Here, for simplicity reasons, we shall only look at an equality length relation. Assume that we know that the two arguments of the predicate A are always of equal length, we can therefore constraint the ranges say ρ_1 and ρ_2 computed for A at step 5) of the algorithm in Table 1 to be of equal sizes[15]. Doing so, the number of free bounds decreases from four to three. Of course, this improvement is independent of the $First$ and $Last$ ones.

We define a *Compare* function whose domain is $(N \times \mathbb{N}^2)$ and codomain is $\{=, \neq\}$. The idea is to perform a static evaluation on the grammar in order to keep track of the comparison of the sizes between couples of arguments for every predicate.

Algorithm 4 *Compare* **construction.**
For simplicity reasons, we will assume that the grammar at hand is non-combinatorial.
This algorithm computes the 3-dimensional array Compare.

1. *Initially* $\forall A,i,j$: $Compare[A,i,j]$ *is set to* $=$.
2. *Let* $\vec{\Psi} = A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_j(\vec{\alpha_j}) \ldots A_m(\vec{\alpha_m})$ *be the positive clause associated to any clause in* P. *Let* $\alpha' = \vec{\alpha_0}[i]$ *and* $\alpha'' = \vec{\alpha_0}[j], i \neq j$, *the two arguments of* A_0 *we wish to compare. If the numbers of terminal symbols in* α' *and* α'' *are different, then set* $Compare[A_0,i,j]$ *to* $\neq$. *If equal, we*
   - *first suppress in* α' *and* α'' *the variables which have multiple occurrences (recall that this variables can only be bound to empty ranges);*
   - *second, variables which occur in both strings are also erased (they are always bound to the same range).*

   If the lengths of these resulting strings, say α_1 *and* α_2, *are different, then set* $Compare[A_0,i,j]$ *to* $\neq$. *If equal, we look for a bijection between the variables of* α_1 *and the variables of* α_2 *such that for each couple* $U \leftrightarrow V$, *where* U *is a variable of* α_1 *and* V *is a variable of* α_2, *we have* $Compare[A_k,r,s]$ *set to* $=$ *and* $U,V \in \{\vec{\alpha_k}[r], \vec{\alpha_k}[s]\}$. *If such a bijection does not exist, we set* $Compare[A_0,i,j]$ *to* $\neq$.
3. *Step 2) is repeated until stability.*

Obviously if for a given predicate we know, by a static computation, that the sizes of its arguments are equal, this may decrease the time complexity of the parsing algorithm.

For example, by a static computation, it is not difficult to see that the two arguments of the equality predicate Eq defined in Example 2 are (always) of equal size. Therefore, for each equality check, in any given recursive call to Eq, there is only a single free value, say the upper bound u_1 of the first range argument. The lower bounds l_1 and l_2 do not change and the upper bound of the second argument is $l_2 + h_1 - l_1$. This shows that each complete string equality is performed in time linear with the common size of its two arguments by the Eq predicate.

Therefore the bindings for X and Y, assuming that the binding of XcY is known and that the sizes of X and Y must be equal, is unique (moreover, if the size of the input range is not an odd number, the function can immediately return false). If we combine this property to the fact that in this example the lower and upper bounds of the ranges of the predicates S and L are constrained (S is called a single time with the range $\bullet w \bullet$, for each call of L, the lower bound of the argument is always 1, and for each call of R, the lower bound of the argument is always $\frac{n-1}{2} + 1$. Therefore we get a linear time parser for $\mathcal{L}(G) = \{wcw \mid w \in \{a,b,c\}^*\}$.

[15] The size of a range $\langle i..j \rangle_w$ is $j - i$.

Of course, due to the modularity, it is possible to conceive a library of predefined languages that any grammar designer can reuse in its own language definition. Moreover, some of these grammars may have their own specialized recognition function. It is clearly the case for the equality predicate which can be simply and efficiently implemented as a simple string comparison.

We have seen, in previous sections, that some grammar types may be transformed into an equivalent sPRCG form. In the next section, we compare the complexities of the best known parsing techniques for these grammars and the complexity of their sPRCG counterpart.

Recall that sPRCGs are non-combinatorial and bottom-up non-erasing and therefore our parsing algorithm complexity is $\mathcal{O}(|P|n^v)$ (see Section 13.4.1 for a definition of v).

13.4.4 The CFG case

If we assume a CFG in binary form, we have seen that it can be transformed into a 1-sPRCG (predicates are all unaries) where the length of each argument is at most 2. Therefore we have $v = 3$ and we get the classical cubic time for CFG's parsing. However, this cubic time is a worst case and there are particular cases where this time can decrease.

In particular, if the language is regular and is defined by a right-linear or left-linear) grammar, (each production is of the form $A \rightarrow xB$ or $A \rightarrow x$, where $A, B \in N$ and $x \in T^*$, our parsing algorithm is linear, though the general formula gives a quadratic time ($v = 2$). The reason is the following. Due to the particular forms of right-linear productions, the corresponding clauses have at most one variable. Moreover, the upper bound of the single range argument is always n and hence the corresponding complexity decreases from quadratic to linear.

One can wonder whether the classical linear parse time for the LL or LR subclasses of CFGs can also be reached. This seems to be difficult with the proposed algorithm since variable bindings are performed a priori (top-down).

13.4.5 The LIG, TAG, HG cases

If we assume a binary branching LIG, we have seen in Section 8.1 that the most complicated case happens in form (6) of Property 11 when we apply the "wrapping" conditions exemplified by the clause:

$$A(XZ, TY) \quad \rightarrow \quad B(X, Y)\ C(Z, T)$$

If we consider a bilinear HG, we have seen in Section 8.3 that the associated clauses have the form

$$\begin{array}{lcl} A(WX, YZ) & \rightarrow & B_1(X, Y)\ \ B_2(W, Z) \\ A(X, YWZ) & \rightarrow & B_1(X, Y)\ B_2(W, Z) \\ A(XYW, Z) & \rightarrow & B_1(X, Y)\ B_2(W, Z) \end{array}$$

In Section 8.2, we have not assumed that TAGs have some binary branching form. However, we know that there is a binary branching normal form for TAGs (see [Vijay-Shanker and Weir 94a]) and moreover the maximum number of inside $\mathcal{A}$-node in each tree can be less or equal than two. Under such conditions, the forms of the associated clauses are exactly those produced for the above HG; we get the first one when the inside nodes are on the spine, the second and the third when a node lays respectively to the right or to the left of the spine.

This shows that, starting from a LIG, a HG or a TAG, the value of the constant v in the equivalent 2-sPRCG is 6. Therefore parsing is performed by our algorithm in time $\mathcal{O}(|P|n^6)$ which is the best time complexity of their initial counterpart grammar.

We can remark that, for example, the transformation of a LIG into an equivalent RCG allows to understand the reason why the analysis time is in $\mathcal{O}(n^6)$. In particular, if the conditions to apply the transformation (6) are not fulfilled for some LIG L, the parsing time for L is less than $\mathcal{O}(n^6)$. It reaches at most $\mathcal{O}(n^5)$ since any string $\gamma_1\gamma_2$ contains at most one variable. Also note that supplementary conditions may also decrease the LIG parsing time, even when the conditions (6) apply (assume that the size of the ranges bound to variables X and Y or W and Z are always equal).

13.4.6 The Coupled Context-Free Grammar case

We have seen in Section 9 that for any CCFG of rank h, there is an equivalent h-sPRCG. For each CCFG there is a binary form (the generalized Chomsky normal form) for which the worst case time complexity is $\mathcal{O}(|P|n^{3h})$

(see [Hotz and Pitsch 94]). Starting from this normal form, we get clauses where the number of arguments is at most h and the number of variables in each LHS argument is at most 2. Therefore the value of v is less or equal to $3h$ and the worst time complexity of the equivalent h-sPRCG is also $\mathcal{O}(|P|n^{3h})$.

Definition 19 *A non-combinatorial h-RCG $G = (N, T, V, P, S)$ in 2-var form is i-limited for some $1 \leq i \leq h$, if for all clauses $A(\alpha_1, \ldots, \alpha_p) \rightarrow \Psi$ in P, we have*

$$|\{j \mid |V^{\alpha_j}| = 2, 1 \leq j \leq p\}| \leq i$$

This means that i is the maximum number of arguments for a given predicate which have exactly two variables.

Property 22 *Let $G = (N, T, V, P, S)$ be an i-limited non-combinatorial h-RCG in 2-var form. Our recognition algorithm solves the word problem for any $w \in T^*$, $n = |w|$, in time $\mathcal{O}(|P|n^{2h+i})$.*

This results from the fact that each argument which has exactly two variables induces a cubic time and there are at most i such arguments which therefore induce a $\mathcal{O}(|P|n^{3i})$ time. The $h - i$ other arguments have exactly one variable, each such argument induces a quadratic time, so the 1-var arguments induce a $\mathcal{O}(|P|n^{2(h-i)})$ time. Finally since $\mathcal{O}(|P|n^{3i})\mathcal{O}(|P|n^{2(h-i)}) = \mathcal{O}(|P|n^{2h+i})$ and since the recognition time is linear with the size of the grammar, we get our result.

Moreover, in [Hotz and Pitsch 94], it is shown that for any Coupled-Context-Free grammar G of rank $h \geq 2$, there exists an equivalent 2-limited G' of rank h in generalized Chomsky normal form. It is trivial to see that the RCG equivalent with G' is a 2-limited h-sPRCG in 2-var form which therefore has, as its original CCFG, a worst case time complexity of $\mathcal{O}(|P|n^{2h+2})$.

13.4.7 The linear context-free rewriting systems case

In Section 10 we have seen that sPRCGs may be viewed as another writing of LCFRS. Therefore we have a proof that LCFRS can be parsed in polynomial time.

13.4.8 Beyond LCFRS

The Example 3, reprinted below

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

whose language is $\{a^{2^p} \mid p \geq 0\}$ shows that the formal power of RCGs exceeds the power of LCFRS (mildly context-sensitive formalisms) since its sentences do not express a constant growth property. In fact, its parse trees are all the complete binary trees. Remark that the added formal power is due to the non-linearity (the variables X occurs more than once in the RHS of the first clause). We can also note that this added power does not necessarily have to be paid by some increase in parsing time complexity.

If we look at this grammar, it is not difficult to see that our parsing algorithm performs $\mathcal{O}(p)$ calls of the function *clause* on the first clause and each call is performed in time $\mathcal{O}(2^i)$ if 2^i is the size of the input range. Since we know that the Eq predicate is executed in linear time with the common size of its two arguments, we get a linear parse time for this grammar since $\sum_i \mathcal{O}(2^i) = \mathcal{O}(2^p) = \mathcal{O}(n)$.

Example 15 *It is possible to generalize the language $\{a^n b^n c^n \mid n >= 0\}$ to the language $\{a_1^n \ldots a_k^n \mid \forall n, k \geq 0 \wedge a_i \in T\}$ where the number of substrings a_i^n of length n is undefined and a given substring may occur several times. This language is defined by the following PRCG:*

$$\begin{array}{lcl} S(XY) & \rightarrow & A(X)\ B(X,Y) \\ B(X,YZ) & \rightarrow & EqLen(X,Y)\ A(Y)\ B(X,Z) \\ B(X,\varepsilon) & \rightarrow & \varepsilon \\ A(X) & \rightarrow & A_i(X) \\ A_i(a_iX) & \rightarrow & A_i(X) \\ A_i(\varepsilon) & \rightarrow & \varepsilon \end{array}$$

where the predefined predicate $EqLen$ is true *iff its two arguments denote ranges of equal size and the three last clauses are schema over terminals $a_i \in T$. It is not difficult to see that each cutting in X, Y of the source range in the first clause is processed in time linear with n. Therefore this grammar is parsed in quadratic time with n.*

Let's take a last example which shows the power of RCG and its computational tractability.

Example 16 *Let's call prime string a string whose length is a prime number. The set of all prime strings is a language which can be defined, for some vocabulary T, by the following RCG clauses:*

$$
\begin{array}{lllll}
1: & S(X) & \rightarrow & Prime(X) & \\
2: & Prime(X) & \rightarrow & \overline{NotPrime}(X) & \\
3: & NotPrime() & \rightarrow & \varepsilon & \\
4: & NotPrime(X_1X_2YZ) & \rightarrow & Len(2,X_1)\ EqLen(X_1X_2,Y) & \\
 & & & XYleqZ(Y,Y,X_1X_2YZ)\ Mul(Y,Z) & \\
5: & XYleqZ(X,\varepsilon,Z) & \rightarrow & \varepsilon & \\
6: & XYleqZ(X,Y_1Y_2,Z_1Z_2) & \rightarrow & Len(1,Y_2)\ EqLen(X,Z_2)\ XYleqZ(X,Y_1,Z_1) & \\
7: & Mul(X,\varepsilon) & \rightarrow & \varepsilon & \\
8: & Mul(X,Y_1Y_2) & \rightarrow & EqLen(X,Y_2)\ Mul(X,Y_1) &
\end{array}
$$

We choose to define prime strings as the complementary of non prime strings (see clause #2). Clause #3 indicates that 0 is not a prime number. A strictly positive integer number n is not a prime number iff there are two integers x and k, $1 < x,k < n$ s.t. $n = kx$. More precisely, we know that a tighten upper bound for x and k is the square root of n (i.e. $2 \leq k,x \leq \sqrt{n}$). Since 1, 2 and 3 are prime numbers, a number $n \geq 4$ is not a prime number iff we have the following conditions: there are three numbers x, y and z s.t. $x+y+z=n$, $2 \leq x$, $x = y$, $x^2 \leq n$ and z is a multiple of x ($\exists k \geq 0 : z = kx$). Assume that the predefined predicate $Len(p,X)$ is true iff the length of the range X is the integer p, the clause #4 may be read as: if the input range of length n may be decomposed into four consecutive ranges X_1, X_2, Y and Z such that $|X_1| + |X_2| \geq 2$ (since $|X_1| = 2$ by $Len(2,X_1)$), $|X_1| + |X_2| = |Y|$ (by $EqLen(X_1X_2,Y)$), $(|X_1| + |X_2|)^2 \leq n$ (checked by the predicate $XYleqZ$) and Z is a multiple of $|X_1| + |X_2|$ (checked by Mul) then n is not a prime number. The idea behind the clauses #5 and #6, defining the predicate $XYleqZ(X,Y,Z)$ is the following. $XYleqZ(X,\varepsilon,Z)$ is always true (clause #5) since $|X| * 0 \leq |Z|$. Clause #6 simply indicates that if X, Y and Z are such that $|Y| = |Y_1| + 1$, $|Z| = |Z_1| + |X|$ and $|X| * |Y_1| \leq |Z_1|$ then we have $|X| * |Y| \leq |Z|$. Clauses #7 and #8 define the predicate $Mul(X,Y)$ which is true iff the length of Y is a multiple of the length of X. We know that 0 is a multiple of $|X|$ (clause #7) and if $|Y| = |Y_1| + |X|$ and if $|Y_1|$ is a multiple of $|X|$, then $|Y|$ is also a multiple of $|X|$ (clause #8).

We can first remark that this definition is very modular and in particular the predicates Mul, $XYleqZ$ and $Prime$ itself do not depend on any particular terminal symbol and as such may be part of a library of modules. Of course, for a particular (main) grammar the set T of terminal symbols must be defined.

Now, let's have a look at its time complexity. We shall assume that each predefined predicate Len or $EqLen$ takes a unit time. If we consider the Mul clauses, we can easily see that a test to know whether two ranges $|X|$ and $|Y|$ are such that $|Y|$ is a multiple of $|X|$ necessitates exactly $|Y|/|X| + 1$ calls and then takes $\mathcal{O}(|Y|/|X|)$ time. Analogously we can see that the test for the predicate $XYleqZ(X,Y,Z)$ takes $\mathcal{O}(|Y|)$ when it succeeds or $\mathcal{O}(|Z|/|Y|)$ when it fails. Now, let us consider clause #4. Here, we can consider that the only unknown is the upper bound of X_2 since the lower bound of X_1 is 0, the upper bound of Z is n, $|X_1| = 2$, $|Y| = |X_1| + |X_2|$ and $|Z| = n - (|X_1| + |X_2| + |Y|)$. Let $X = X_1X_2$, therefore, the size of X takes at most $n/2$ different values. The complexity due to the execution of the predicate $XYleqZ$ is the addition of two terms, one when its execution succeeds and the other when its execution fails. We know that the execution succeeds if $|X| \leq \sqrt{n}$ or fails otherwise. Therefore the execution time for the predicate $XYleqZ$ is $\Sigma_{x=2}^{\sqrt{n}} x + \Sigma_{x=\sqrt{n}}^{n/2} n/x$ whose limit is $\mathcal{O}(n) + \mathcal{O}(n \log n) \approx \mathcal{O}(n \log n)$. The execution time for the predicate Mul is $\Sigma_{x=2}^{n/2} n/x$ whose limit is $\mathcal{O}(n \log n)$[16]. Therefore, time complexity for the language of prime strings is $\mathcal{O}(n \log n)$.

14 Conclusion

In this paper we present a syntactic formalism which is an extension of CFGs and whose formal power allows to express numerous phenomena occurring in natural language processing, while staying computationally tractable. The associated parsers work in time polynomial with the size of the input string and in time linear with the size of the grammar. In a given grammar, only complicated (many arguments, many variables) clauses can produce higher parsing times whereas simpler clauses induce lower times.

[16] In fact, if we assume that the RHS predicates are executed from left to right, the upper bound of the Σ for the predicate Mul is $\sqrt{n}$ instead of $n/2$ since Mul is only executed when $XYleqZ$ succeeds. However, this consideration does not improve over the $\mathcal{O}(n \log n)$ limit.

In fact, PRCGs exactly cover the class *PTIME* of languages recognizable in deterministic polynomial time and as such they are strictly more powerful than TAGs or LCFRS. We have shown that many grammatical formalisms (CFG, LIG, TAG, HG, CCFG, LCFRS, ...) may be directly translated into a weakly equivalent PRCG form. Note that this formal power does not induce any overcost since the complexity of their PRCG counterpart is exactly their best known complexity. For example we get a linear time for regular CFG, a cubic time for CFG and an $\mathcal{O}(n^6)$ parse time for LIG, TAG or HG. Moreover, it may bring some new insight into these formalisms and may for example help to understand why the complexity of a LIG recognizer is $\mathcal{O}(n^6)$.

In this paper, we have introduced some predefined predicates as *Eq*, *EqLen* or *Len* whose usage, when appropriate, may decrease the parsing time in avoiding internal loops within the recognition procedure. A prototype version of the recognizer in Table 1 has been implemented and it has for example allowed to check that the practical recognition time for the *prime strings* language of Example 16 is the theoretical time $\mathcal{O}(n \log n)$[17]. The *Compare* array of § 13.4.3 may be thought of as an attempt to automatically extract the *EqLen* predefined predicate from a grammar.

Though the worst case upper bound is not improved by the usage, within the parser, of the statically computed structures *First* and *Last*, we are convinced that practical parse times will be greatly enhanced by this technique. Moreover, our parsing algorithm seems to be well suited to allow parallel implementations.

For a given input string, the exponential or even unbounded set of derivation trees which are the output of our parser, can be represented into a packed structure, the shared forest, which is a CFG of polynomial size and from which each individual derivation tree can be extracted in linear time with its own size.

These RCGs may themselves be considered as a syntactic backbone upon which other formalisms as Herbrand's domain or feature structures can be grafted in the very same way as these formalisms decorate CFGs.

At last, we have seen that RCGs are closed under intersection and complementation in a way which preserves the structure of their individual components. These properties indicate that RCGs are modular and allow to imagine libraries of generic linguistic modules in which any language designer can pick up at will when he wants to define such and such phenomenon.

All these properties seems to advocate that RCGs are perhaps the right level of syntactic context-sensitivity needed in NLP. Moreover, this formalism may have applications in other domains. We here think of biology and particularly about RNA pseudoknot problem and the multiple sequence alignment problem.

References

[Boullier 95] Pierre Boullier. 1995. Yet another $\mathcal{O}(n^6)$ recognition algorithm for mildly context-sensitive languages. In *Proceedings of the fourth international workshop on parsing technologies (IWPT'95)*, Prague and Karlovy Vary, Czech Republic, pages 34–47. See also *Research Report No 2730* at `http://www.inria.fr/RRRT/RR-2730.html`, INRIA-Rocquencourt, France, Nov. 1995, 22 pages.

[Boullier 96] Pierre Boullier. 1996. Another Facet of LIG Parsing In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL96)*, University of California Santa Cruz, California, USA, pages 87–94. See also *Research Report No 2858* at `http://www.inria.fr/RRRT/RR-2858.html`, INRIA-Rocquencourt, France, Apr. 1996, 22 pages.

[Groenink 96] Annius V. Groenink. 1996. Mild Context-Sensitivity and Tuple-based Generalizations of Context-Free Grammar. In *Report CS-R9634*, CWI Amsterdam, The Nederlands, Sep. 1996, 22 pages.

[Groenink 97] Annius Victor Groenink. 1997. Surface without Structure Word order and tractability in natural language analysis. PhD thesis, Utrecht University, The Nederlands, Nov. 1977, 250 pages.

[Hotz and Pitsch 94] Günter Hotz, Gisela Pitsch. 1994. Fast Uniform Analysis of Coupled-Context-Free languages. In *Proc. 21th Internat. Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 820 (Springer, Berlin 1994), pages 412–423.

[Kaji et al. 92] Y. Kaji, R. Nakanishi, H. Seki, T. Kasami 1992. The Universal Recognition Problems for Parallel Multiple Context-Free Grammars and for Their Subclasses. In *IEICE*, E75-D(4), pages 499–508.

[Lang 94] Bernard Lang. 1994. Recognition can be harder than parsing. In *Computational Intelligence*, Vol. 10, No. 4, pages 486–494.

[17] During this test, performed on an old 40Mhz Sun SS10, it takes one second to check that a string of 19489 a's is a sentence (i.e. that 19489 is a prime number).

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

[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

[Vijay-Shanker and Weir 94b] K. Vijay-Shanker, David J. Weir. 1994. Parsing some constrained grammar formalisms. In *ACL Computational Linguistics*, Vol. 19, No. 4, pages 591–636.

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

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

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