RANGE CONCATENATION GRAMMARS

Pierre Boullier
INRIA-Rocquencourt
BP 105
78153 Le Chesnay Cedex, France
Pierre.Boullier@inria.fr

Abstract

In this paper we present Range Concatenation Grammars, a syntactic formalism which possesses many attractive features among which we underline here, power and closure properties. For example, Range Concatenation Grammars are more powerful than Linear Context-Free Rewriting Systems though this power is not reached to the detriment of efficiency since its sentences can always be parsed in polynomial time. Range Concatenation Languages are closed both under intersection and complementation and these closure properties may allow to consider novel ways to describe some linguistic processings. We also present a parsing algorithm which is the basis of our current prototype implementation.

1 Introduction

The great number of syntactic formalisms upon which natural language (NL) processing is based may be interpreted in two ways: on one hand this shows that this research field is very active and on the other hand it shows that, at the evidence, 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, it must allow the description of features that have been identified so far in various NLs, while staying computationally tractable. We know that, due to their lack of expressiveness, context-free grammars (CFGs) cannot play this role (See [Shieber, 1985]). Yet, context sensitive grammars are powerful enough although they are too greedy in computer time. A first answer is given by the notion of mild context-sensitivity. This notion is an attempt to express the formal power needed to define NLs (see [Joshi, 1985] and [Weir, 1988]). However, there exist some phenomena such as large Chinese numbers or word scrambling that are outside the power of mildly context-sensitive (MCS) formalisms. In this paper, we present a convincing alternative: the range concatenation grammars (RCGs). RCG is a syntactic formalism which is a variant of the simple version of literal movement grammar (LMG), described in [Groenink, 1997], and which is also related to the framework of LFP developed in [Rounds, 1988]. In fact it may be considered to lie halfway between their respective *string* and *integer* versions; RCGs retain from the string version of LMGs or LFPs the notion of concatenation, applying it to ranges (couples of integers which denote occurrences of substrings in a source text) rather than strings, and from their integer version the ability to handle only (part of) the source text (this later feature is the key to tractability). The rewriting rules of RCGs, called *clauses*, apply to composite objects named *predicates* which can be seen as nonterminal symbols with arguments. We have shown that the positive version of RCGs, as simple LMGs or integer indexing LFPs, exactly covers the class *PTIME* of languages recognizable in deterministic polynomial time. Since the composition operations of RCGs are not restricted to be linear and non-erasing, its languages (RCLs) are not semi-linear. Therefore, RCGs are *not* MCS and they are more powerful than linear context-free rewriting systems (LCFRS) [Vijay-Shanker, Weir, and Joshi, 1987],[1] while staying computationally tractable: its sentences can be parsed in polynomial time. However, this formalism shares with LCFRS the fact that its derivations are CF (i.e., the choice of the operation performed at each step only depends on the object to be derived from). As in the CF case, its derived trees can be packed into polynomial sized parse forests. Besides its power and efficiency, this formalism possesses many other attractive properties. Let us emphasize in this introduction the fact that RCLs are both closed under intersection and complementation, and, like CFGs, RCGs can act as syntactic backbones upon which can be grafted decorations from other domains (probabilities, logical terms, feature structures).

This paper studies the full class of RCGs and presents a polynomial parse time algorithm.

2 Positive Range Concatenation Grammars

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

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

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

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

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

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

2.1 Ranges & Bindings

If we consider a derivation in a CFG, headed at the start symbol and leading to some sentence, we know that each nonterminal occurrence is responsible for the generation of the substring laying between two indexes say i and j. For a given input string $w = a_1 \dots a_n$, such a couple (i, j) is called a *range*. We know that in CF theory, ranges play a central role. For example the (unbounded number of) parse trees, associated with some input string w can be represented by a CFG, called *shared forest* [Lang, 1994]. Its nonterminal symbols have the form (A, i, j) where A is a nonterminal of the initial grammar and (i, j) is a range in w. In an analogous way, ranges are the core of our formalism.

In the sequel terminal symbol in T are denoted by early occurring lower case letters such as $a, b, c, \dots$, while variables in V are denoted by late occurring upper case letters such as X, Y, Z. If $w \in T^*$, $|w| = n$, its set of ranges is $\mathcal{R}_w = \{\rho \mid \rho = (i, j), 0 \leq i \leq j \leq n\}$. We shall sometimes use vectors to denote elements of cartesian products. For example elements in $\mathcal{R}_w^k$ (i.e., tuple of ranges of length k) may be denoted by $\vec{\rho}$ and clauses by $A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots A_m(\vec{\alpha_m})$, $\vec{\alpha_i} \in ((T \cup V)^*)^{k_i}$, $k_i = arity(A_i)$.

[1] In [Boullier, 1999a], we argue that this extra power can be used in NL processing.

We shall use several equivalent denotations for ranges in $\mathcal{R}_w$. If $w = w_1 w_2 w_3$ with $w_1 = a_1 \ldots a_i$, $w_2 = a_{i+1} \ldots a_j$ and $w_3 = a_{j+1} \ldots a_n$, the range (i, j) can be denoted either by an explicit dotted term $w_1 \bullet w_2 \bullet w_3$, or by $\langle i..j \rangle_w$, or even by $\langle i..j \rangle$ when w is understood or of no importance. Given a range $\langle i..j \rangle$, the integer i is its *lower bound*, j is its *upper bound* and $j - i$ is its *size*. A range such that $i = j$ is an *empty* range. 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 $w^{\langle j..n \rangle}$. Therefore we have, $w^{\langle j..j \rangle} = \varepsilon$, $w^{\langle j-1..j \rangle} = a_j$ and $w^{\langle 0..n \rangle} = w$. If $\vec{\rho} = \rho_1, \ldots, \rho_i, \ldots, \rho_p$ is a vector of ranges, by definition $w^{\vec{\rho}}$ denotes the tuple of strings $w^{\rho_1}, \ldots, w^{\rho_i}, \ldots, w^{\rho_p}$.

In any PRCG, terminals, variables and arguments in a clause are supposed to be bound to ranges by a substitution mechanism. Any couple (X, ρ) is called a *variable binding* denoted by X/ρ, ρ is the *range instantiation* of X, and w^ρ is its *string instantiation*. A set $\sigma = \{X_1/\rho_1, \ldots, X_p/\rho_p\}$ of variable bindings is a *variable substitution* iff $X_i/\rho_i \neq X_j/\rho_j \Rightarrow X_i \neq X_j$. A couple (a, ρ) is a *terminal binding* denoted by a/ρ iff $\rho = \langle j-1..j \rangle$ and $a = a_j$.

The *concatenation of ranges* is a partial (associative) binary operation on $\mathcal{R}_w$ defined by $\langle i_1..j_1 \rangle_w \langle i_2..j_2 \rangle_w = \langle i_1..j_2 \rangle_w$ iff $j_1 = i_2$. If we consider a string $w \in T^*$, a string $\alpha = u_1 \ldots u_i \ldots u_p \in (T \cup V)^*$, a variable substitution σ and a range $\rho \in \mathcal{R}_w$, the couple (α, ρ) is a *string binding* for σ, denoted α/ρ iff

- for each u_i there exists a range ρ_i s.t.
  - if $u_i \in V$, $u_i/\rho_i \in \sigma$,
  - if $u_i \in T$, u_i/ρ_i is such that $u_i = w^{\rho_i}$,
- and $\rho_1 \ldots \rho_i \ldots \rho_p = \rho$

For a given variable substitution σ, a set $\omega = \{\alpha_1/\rho_1, \ldots, \alpha_p/\rho_p\}$ is called a *string substitution* (for $\{\alpha_1, \ldots, \alpha_p\}$), iff each α_i/ρ_i is a string binding for σ. A clause c is *instantiable* by ω iff there is a string substitution ω for the set of arguments of its predicates. If a clause is instantiable by ω, and if each of its arguments α is replaced by the range ρ s.t. $\alpha/\rho \in \omega$, we get a *positive instantiated clause* whose components are *positive instantiated predicates*. For example, $A(\langle g..h \rangle, \langle i..j \rangle, \langle k..l \rangle) \rightarrow B(\langle g{+}1..h \rangle, \langle i{+}1..j{-}1 \rangle, \langle k..l{-}1 \rangle)$ is a positive instantiation of the clause $A(aX, bYc, Zd) \rightarrow B(X, Y, Z)$ if the source text $a_1 \ldots a_n$ is such that $a_{g+1} = a, a_{i+1} = b, a_j = c$ and $a_l = d$. In this case, the variables X, Y and Z are bound to $\langle g{+}1..h \rangle$, $\langle i{+}1..j{-}1 \rangle$ and $\langle k..l{-}1 \rangle$ respectively.[2]

For some $w \in T^*$, the set of *positive instantiated predicates* IP^+_w is defined by $IP^+_w = \{A(\vec{\rho}) \mid A \in N, \vec{\rho} \in \mathcal{R}^k_w, k = arity(A)\}$.

2.2 Derivation, Language, Derived Tree & Shared Forest

For a PRCG $G = (N, T, V, P, S)$ and a source text w, we define, on strings of positive instantiated predicates, a binary relation called *positive derive* and denoted by $\underset{G,w}{\overset{+}{\Rightarrow}}$. If Γ_1 and Γ_2 are strings in $(IP^+_w)^*$, we have

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

[2] Often, for a variable binding X/ρ, instead of saying that *ρ is the range which is bound to X or denoted by X*, we shall say, the range X, or even for w^ρ, instead of saying *the string whose occurrence is denoted by the range which is bound to X*, we shall say the string X.

iff $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \ldots\ A_m(\vec{\rho_m})$ is the instantiation of some clause $A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m})$ in P for some string substitution.

A sequence of strings of positive instantiated predicates $\Gamma_0, \ldots, \Gamma_{i-1}, \Gamma_i, \ldots, \Gamma_l$ s.t. $\forall i, 1 \leq i \leq l$: $\Gamma_{i-1} \underset{G,w}{+\!\!\Rightarrow} \Gamma_i$ is called a *derivation*, or more precisely a Γ_0*-derivation* or even a Γ_0/Γ_l*-derivation*. Each consecutive couple of strings (Γ_{i-1}, Γ_i) is a *derivation step*.

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

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

An input string $w \in T^*$, $|w| = n$ is a sentence iff the empty string (of positive instantiated predicates) can be derived from $S(\langle 0..n \rangle)$, the positive instantiation of the start predicate on the whole source text.

More generally, we define the string language of a nonterminal A by $\mathcal{L}(A) = \cup_{w \in T^*} \mathcal{L}(A, w)$ where $\mathcal{L}(A, w) = \{w^{\vec{\rho}} \mid \vec{\rho} \in \mathcal{R}_w^h, h = arity(A), A(\vec{\rho}) \overset{+}{\underset{G,w}{+\!\!\Rightarrow}} \varepsilon\}$. However, with this definition, we can note that $\mathcal{L}(S)$ and $\mathcal{L}(G)$ are different. Consider the grammar G s.t.

$$\begin{array}{rcl} S(X) & \rightarrow & A(Xa) \\ A(X) & \rightarrow & \varepsilon \end{array}$$

We can see that we have $\mathcal{L}(G) = \emptyset$ and $\mathcal{L}(S) = T^*$.[3] In fact, we have $\mathcal{L}(G) \subset \mathcal{L}(S)$, and the equality is reached for non-increasing grammars.[4]

As in the CF case, if we consider a 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). All possible derivation strategies can be captured in a single canonical tree structure which abstracts all possible orders and which is called a *derived tree* (or *parse tree*). For any given $A(\vec{\rho})$-derivation, we can associate a single derived tree whose root is labeled $A(\vec{\rho})$. Conversely, if we consider a derived tree, there may be associated derivations which depend upon the way the tree is traversed (for example a top-down left-to-right traversal leads to a leftmost derivation). Note 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 this occurrence is known, the clause 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 an instantiation of different clauses. But, of course, each of these interpretations is a valid one.

Consider a k-PRCG $G = (N, T, V, P, S)$, a terminal string w and the set $\mathcal{D}_w$ of all complete $S(\bullet w \bullet)/\varepsilon$-derivations. We define a terminal-free CFG $G_w = (N \times \mathcal{R}_w^k, \emptyset, P_w, S(\bullet w \bullet))$ whose set of rules P_w is formed by all the instantiated clauses $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho}_1) \ldots A_m(\vec{\rho_m})$ used in the derivation steps in $\mathcal{D}_w$. This CFG is called the *shared forest* for w w.r.t. G. Note that, if $\mathcal{D}_w$ is not empty, the language of a shared forest for w is not $\{w\}$, as in the CF case, but is $\{\varepsilon\}$.

Moreover, this shared forest of polynomial size may be viewed as an exact packed representation of all the (unbounded number of) derived (parse) trees in G for w: the set of parse trees for G on the input w and the set of parse trees of its associated CF shared forest G_w (on the input ε), are identical.

---

[3] For every $w = u_1 u_2 a v, u_1 u_2 v \in T^*$, we have $S(u_1 \bullet u_2 \bullet av) \overset{+}{\underset{G,w}{+\!\!\Rightarrow}} \varepsilon$.

[4] A grammar is *non-increasing* iff for every instantiated clause and for each argument binding α'/ρ', $\rho' = \langle j..k \rangle$ in its RHS there exists in its LHS an argument binding α/ρ, $\rho = \langle i..l \rangle$ such that $i \leq j \leq k \leq l$. Non-increasing grammars represent an important and fairly large subclass of RCGs

The arguments of a given predicate may denote discontinuous or even overlapping ranges. Fundamentally, a predicate name A defines a notion (property, structure, dependency, ...) between its arguments, whose associated ranges can be arbitrarily scattered over the source text. PRCGs are therefore well suited to describe long distance dependencies. Overlapping ranges arise as a consequence of the non-linearity of the formalism. For example, the same variable (denoting the same range) may occur in different arguments in the RHS of some clause, expressing different views (properties) of the same portion of the source text.

Note that the order of predicate calls in the RHS of a clause is of no importance (in fact, RHS of clauses are sets of predicate calls rather than lists).

Example 1 *As an example of a PRCG, the following set of clauses describes the three-copy language* $\{www \mid w \in \{a,b\}^*\}$ *which is not a CFL and even lies beyond the formal power of tree adjoining grammars (TAGs).*

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

Below, we check that the input string $w = a_1b_2a_3b_4a_5b_6$ *is a sentence. Of course, indices are not part of the input letters, they are used to improve readability of ranges: for each couple (letter, index), we know where letter occurs in the source text. At each derivation step, we have made clear both the clause and the variable substitution used.*

$$\begin{array}{llll} S(a_1b_2a_3b_4a_5b_6) & \overset{S(XYZ)\rightarrow A(X,Y,Z)}{\underset{G,w}{\overset{+}{\Rightarrow}}} & A(a_1b_2,a_3b_4,a_5b_6) & \{X/a_1b_2, Y/a_3b_4, Z/a_5b_6\} \\ & \overset{A(aX,aY,aZ)\rightarrow A(X,Y,Z)}{\underset{G,w}{\overset{+}{\Rightarrow}}} & A(b_2,b_4,b_6) & \{X/b_2, Y/b_4, Z/b_6\} \\ & \overset{A(bX,bY,bZ)\rightarrow A(X,Y,Z)}{\underset{G,w}{\overset{+}{\Rightarrow}}} & A(\varepsilon,\varepsilon,\varepsilon) & \{X/\langle 2..2\rangle, Y/\langle 4..4\rangle, Z/\langle 6..6\rangle\} \\ & \overset{A(\varepsilon,\varepsilon,\varepsilon)\rightarrow\varepsilon}{\underset{G,w}{\overset{+}{\Rightarrow}}} & \varepsilon & \emptyset \end{array}$$

Example 2 *Another way to define the previous language is with the following set of clauses:*

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

where the equality predicate eq *is defined by*

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

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

Example 3 *The power of this formalism is shown by the following grammar that defines the non semi-linear language* $\mathcal{L} = \{a^{2^p} \mid p \geq 0\}$

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

3 Negative Range Concatenation Grammars

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

A predicate call of the form $\overline{A(\alpha_1, \ldots, \alpha_p)}$ is said to be a *negative predicate call.*

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

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

In a NRCG, the intended meaning of a negative predicate call is to define the complement language (w.r.t. T^*) of its positive counterpart: an instantiated negative predicate succeeds iff its positive counterpart (always) fails. This definition is based on a "negation by failure" rule.

More formally, let $G = (N, T, V, P, S)$ be a RCG, and let w be a string in T^*. The set of *negative instantiated predicate* IP_w^- is defined by $IP_w^- = \{\overline{A(\vec{\rho})} \mid A(\vec{\rho}) \in IP_w^+\}$,[5] and the set of *instantiated predicate* IP_w is defined by $IP_w = IP_w^+ \cup IP_w^-$.

For RCGs, we redefine the *positive derive* relation $\underset{G,w}{+\!\Rightarrow}$ in the following way. If Γ_1 and Γ_2 are strings in $(IP_w)^*$, we have

$$\Gamma_1 \; A_0(\vec{\rho_0}) \; \Gamma_2 \quad \underset{G,w}{+\!\Rightarrow} \quad \Gamma_1 \; \phi_1 \ldots \phi_m \; \Gamma_2$$

iff $A_0(\vec{\rho_0}) \to \phi_1 \ldots \phi_m$ is the instantiation of some clause $A_0(\vec{\alpha_0}) \to \psi_1 \ldots \psi_m$ in P for some string substitution ω. If $\vec{\alpha_i}/\vec{\rho_i} \in \omega$,[6] we have, either if $\psi_i = A_i(\vec{\alpha_i})$ then $\phi_i = A_i(\vec{\rho_i})$, or if $\psi_i = \overline{A_i(\vec{\alpha_i})}$ then $\phi_i = \overline{A_i(\vec{\rho_i})}$.

Note that negative instantiated predicates cannot be derived further on by positive derive relations.

We also define on strings of instantiated predicates a *negative derive* relation, denoted by $\underset{G,w}{-\!\Rightarrow}$. If Γ_1 and Γ_2 are strings in $(IP_w)^*$, we have

$$\Gamma_1 \; \overline{A(\vec{\rho})} \; \Gamma_2 \quad \underset{G,w}{-\!\Rightarrow} \quad \Gamma_1 \Gamma_2$$

iff $\overline{A(\vec{\rho})}$ is a negative instantiated predicate such that $(A(\vec{\rho}), \varepsilon) \not\in \overset{+}{\underset{G,w}{+\!\Rightarrow}}$.

Note that this definition of $\underset{G,w}{-\!\Rightarrow}$ "erases" negative instantiated predicates whose positive counterpart is not related to the empty string (of instantiated predicates) by the transitive closure of $\underset{G,w}{+\!\Rightarrow}$. As a consequence of this definition, the structure (parse tree) associated with a negative derivation step is void, and, more generally, the structure of the (complement) language associated with a negative predicate call is void. In other words, within the RCG formalism, we cannot define any structure between a negative predicate call and the parts of an input string that it selects.

Let $\underset{G,w}{-\!-\!\Rightarrow}$ be any subset of $\underset{G,w}{-\!\Rightarrow}$. We define a *positive/negative derive* relation $\underset{G,w}{\pm\!\Rightarrow}$ by

$$\underset{G,w}{\pm\!\Rightarrow} \quad = \quad \underset{G,w}{+\!\Rightarrow} \cup \underset{G,w}{-\!-\!\Rightarrow}$$

We say that $\underset{G,w}{\pm\!\Rightarrow}$ is *consistent* (otherwise *inconsistent*) iff for each $A(\vec{\rho}) \in IP_w^+$ we have either $A(\vec{\rho}) \overset{+}{\underset{G,w}{\pm\!\Rightarrow}} \varepsilon$ or $\overline{A(\vec{\rho})} \underset{G,w}{\pm\!\Rightarrow} \varepsilon$, but not both. Note that the existence of both such derivations would show that the tuple of strings $w^{\vec{\rho}}$ simultaneously belongs to the language of A and to its complement!

[5] The previous definition of IP_w^+ still holds for (N)RCGs.

[6] $\vec{\alpha_i}/\vec{\rho_i}$ is a shorthand notation for $\{\alpha_{i1}/\rho_{i1}, \ldots, \alpha_{ip_i}/\rho_{ip_i}\}$ if $\vec{\alpha_i} = \alpha_{i1}, \ldots, \alpha_{ip_i}$ and $\vec{\rho_i} = \rho_{i1}, \ldots, \rho_{ip_i}$.

We say that a grammar G is *consistent* if for every $w \in T^*$, there exists a consistent relation $\underset{G,w}{\pm\!\!\Rightarrow}$ (otherwise G is *inconsistent*).

A consistent positive/negative derive relation is simply called *derive* and is denoted by $\underset{G,w}{\Rightarrow}$. If a derive relation exists, we can show that it is unique. However, such a derive relation does not always exist, and thus some grammars are inconsistent.

Consider the RCG G whose set of clauses is the singleton $P = \{S(X) \rightarrow \overline{S(X)}\}$. By definition, for any $w \in T^*$ and for any $\rho \in \mathcal{R}_w$, we have

$$\begin{array}{lcl} \{(S(\rho), \overline{S(\rho)})\} & \subset & \underset{G,w}{+\!\!\Rightarrow} \\ \{(\overline{S(\rho)}, \varepsilon)\} & \subset & \underset{G,w}{-\!\!\Rightarrow} \end{array}$$

There are two possibilities for $\underset{G,w}{-\!\!-\!\!\Rightarrow}$: the couple $(\overline{S(\rho)}, \varepsilon)$ either does not belong or does belong to $\underset{G,w}{-\!\!-\!\!\Rightarrow}$. In the first case, $\underset{G,w}{\pm\!\!\Rightarrow}$ is inconsistent since we have both $(S(\rho), \varepsilon) \not\in \overset{+}{\underset{G,w}{\pm\!\!\Rightarrow}}$ and $(\overline{S(\rho)}, \varepsilon) \not\in \underset{G,w}{\pm\!\!\Rightarrow}$. In the second case, $\underset{G,w}{\pm\!\!\Rightarrow}$ is also inconsistent since we have

$$\begin{array}{lcl} \{(S(\rho), \overline{S(\rho)}), (\overline{S(\rho)}, \varepsilon)\} & \subset & \underset{G,w}{\pm\!\!\Rightarrow} \\ \{(S(\rho), \overline{S(\rho)}), (\overline{S(\rho)}, \varepsilon), (S(\rho), \varepsilon)\} & \subset & \overset{+}{\underset{G,w}{\pm\!\!\Rightarrow}} \end{array}$$

thus G is inconsistent.

In the sequel, we shall only consider consistent RCGs. Of course, PRCGs are always consistent.

Definition 5 *The (string)* language *of a RCG* $G = (N, T, V, P, S)$ *is the set*

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

The language of inconsistent RCGs is undefined.

In the sequel, without loss of generality, we will both prohibit clauses whose RHS contains arguments that are in T^*, and assume that no argument has more than one instance of any variable.

4 A Parsing Algorithm for RCGs

In our prototype system, we have implemented a RCG parser which is based upon the algorithm depicted in Table 1. Let $G = (N, T, V, P, S)$ be a k-PRCG for which we also consider P as a sequence of clauses denoted by $\vec{P}$. In this algorithm, the functions *clause* and *prdct* are both memoized: their returned values are kept in auxiliary $1+k$-dimensional matrixes Γ and Π which are indexed by elements in $P \times \mathcal{R}_w^k$ and $N \times \mathcal{R}_w^k$. We assume that their elements are all initialized to **unset**. The internal loop at lines #5, #6 and #7 is executed for each possible instantiation of the current clause, with the only constraint that its LHS arguments must always be bound to the parameter $\vec{\rho_0}$. In function *prdct* at line #4, P_A designates the set of A-clauses. It is not difficult to see that we have implemented a top-down recognizer[7] for any PRCG if the function *prdct* is called, for some input string w, with $prdct(S, \bullet w \bullet)$.

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

(6') **output** $(A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \ldots A_j(\vec{\rho_j}) \ldots A_m(\vec{\rho_m}))$

[7] Bottom-up algorithms can also be devised.

(1) **function** *clause* $(i, \vec{\rho_0})$ **return boolean**
(2) **if** $\Gamma[i, \vec{\rho_0}] \neq$ **unset then return** $\Gamma[i, \vec{\rho_0}]$
(3) **let** $A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_j(\vec{\alpha_j}) \ldots A_m(\vec{\alpha_m}) = \vec{P}[i]$
(4) *ret-val* $:= \Gamma[i, \vec{\rho_0}] :=$ **false**
(5) **foreach** $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \ldots A_j(\vec{\rho_j}) \ldots A_m(\vec{\rho_m})$ **do**
(6) *ret-val* $:=$ *ret-val* $\vee$ (*prdct* $(A_1, \vec{\rho_1}) \wedge \ldots$ *prdct* $(A_j, \vec{\rho_j}) \wedge \ldots$ *prdct* $(A_m, \vec{\rho_m})$)
(7) **end foreach**
(8) **return** $\Gamma[i, \vec{\rho_0}] :=$ *ret-val*
(9) **end function**

(1) **function** *prdct* $(A, \vec{\rho})$ **return boolean**
(2) **if** $\Pi[A, \vec{\rho_0}] \neq$ **unset then return** $\Pi[A, \vec{\rho_0}]$
(3) *ret-val* $:=$ **false**
(4) **foreach** i **such that** $\vec{P}[i] \in P_A$ **do**
(5) *ret-val* $:=$ *ret-val* $\vee$ *clause* $(i, \vec{\rho})$
(6) **end foreach**
(7) **return** $\Pi[A, \vec{\rho_0}] :=$ *ret-val*
(8) **end function**

Table 1: A Recognition Algorithm for PRCGs.

after line #6 in *clause*, statement which must only be executed if the expression (*prdct* $(A_1, \vec{\rho_1}) \wedge \ldots \wedge$ *prdct* $(A_m, \vec{\rho_m})$) succeeds.

In order to handle the full class of RCGs, we simply have to change line #6 in *clause* by something like

(6) *ret-val* $:=$ *ret-val* $\vee$ (*prdct* $(A_1, \vec{\rho_1}) \wedge \ldots \overline{prdct(A_j, \vec{\rho_j})} \wedge \ldots$ *prdct* $(A_m, \vec{\rho_m})$)

if $\vec{P}[i]$ has the form $A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots \overline{A_j(\vec{\alpha_j})} \ldots A_m(\vec{\alpha_m})$.[8]

We can also note that the assignment of $\Gamma[i, \vec{\rho_0}]$ to **false** at line #4 in *clause* allows this recognizer to handle cyclic grammars.[9]

4.1 Its Parse Time Complexity

For a given k-RCG, and an input string $w \in T^*$, $|w| = n$, the number of instantiations of a given clause $\vec{P}[i]$, is less than or equal to $n^{2k(1+l_i)}$ where l_i is the number of predicate calls in the RHS of $\vec{P}[i]$. Thus, for a given clause, thanks to the memoization mechanism, the number of calls of the form *prdct* $(A_j, \vec{\rho_j})$ in line #6 of *clause*, is less than or equal to $l_i n^{2k(1+l_i)} \leq l_i n^{2k(1+l)}$ where l is the length of the longest clause. Thus, for all possible clauses this number is $\sum_{i=1}^{|P|} l_i n^{2k(1+l)} = |G| n^{2k(1+l)}$ if $|G| = \sum_{i=1}^{|P|} l_i$ is the *size* of the grammar. Thus, if we assume that each use of the function *prdct* takes a constant time, the time complexity of this algorithm is at most $\mathcal{O}(|G| n^{2k(1+l)})$,[10] and its space complexity is $\mathcal{O}(|P| n^{2k})$, the size of the memoization matrixes Γ and Π.

We emphasize the fact that a linear dependency upon the grammar size is extremely important in NL processing where we handle huge grammars and small sentences.

[8] Note that in the corresponding shared forest, *negative nodes* of the form $\overline{A_j(\vec{\rho_j})}$ must be considered either as leaves or, equivalently, we must add in the parser a statement such as **output** $(\overline{A_j(\vec{\rho_j})} \rightarrow \varepsilon)$.

[9] I.e., grammars for which there exist derivations such that $A(\vec{\rho}) \overset{+}{\underset{G,w}{\Rightarrow}} \Gamma_1 A(\vec{\rho})\, \Gamma_2$. However, in order to get a parser for cyclic grammars, this simple mechanism had to be improved.

[10] In fact we have $\mathcal{O}(|G| n^{2r})$ where r is the maximum number of arguments in a clause.

In the above evaluation, we have assumed that arguments in a clause are all independent; this is rarely the case. If we consider a predicate argument $\alpha = u_1 \dots u_p \in (V \cup T)^*$ and the string binding α/ρ for some variable substitution σ, each position $0, 1, \dots, p$ in α is mapped onto a *source index* (a position in the source text) $i_0, i_1, \dots, i_p$ s.t. $i_0 \leq i_1 \leq \dots \leq i_p$ and $\rho = \langle i_0..i_p \rangle$. These source indexes are not necessarily independent (free). In particular, if for example $u_j \in T$, we have $i_j = i_{j-1} + 1$. Moreover, most of the time, in a clause, variables have multiple occurrences. This means that, in a clause instantiation, the lower source indexes and the upper source indexes associated with all the occurrences of the same variable are always the same, and thus are not free. In fact, the degree d of the polynomial which expresses the maximum parse time complexity associated with a clause is equal to the number of free bounds in that clause. For any RCG G, if d is its maximum number of free bounds, the parse time of an input string of length n takes at worst $\mathcal{O}(|G|n^d)$.

If we consider a bottom-up non-erasing[11] k-RCG G, by definition, there is no free bound in its RHS. Thus, in this case, the number of free bounds is less than or equal to $d = \max_{\vec{P}[i]}(k_i + v_i)$, where k_i and v_i are respectively the arity and the number of (different) variables in the LHS predicate of $\vec{P}[i]$.

In Example 2, we have defined a predicate named *eq* which may be useful in many grammars, thus, in our prototype implementation, we decided to predefine it, together with some others, among which we quote here *len* and *eqlen*:

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

$eqlen(X, Y)$**:** checks that the sizes of X and Y are equal;

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

It must be noted that these predefined predicates do not increase the formal power of RCGs insofar as each of them can be defined by a pure RCG. Their introduction is justified by the fact that they are more efficiently implemented than their RCG-defined counterpart and, more significantly, because they convey static information which can be used to decrease the number of free bounds and may thus lead to an improved parse time.

Consider again Example 2. This grammar is bottom-up non-erasing, and the most complex clause is the first one. Its number of free bounds is four (one argument with three variables), and thus its parse time complexity is at worst $\mathcal{O}(n^4)$. In fact, this complexity is at worst quadratic since neither the lower bound nor the upper bound of the argument XYZ is free since their associated source index values are always 0 and n respectively. Note that the lower bound of the definitions of the unary predicate L is not free either, since its value is always the source index zero. The parse time complexity of this grammar further decreases to linear if *eq* is predefined, because in that case, we statically know that the sizes of X, Y and Z must be equal (there is no more free bound within this first clause which is thus executed in constant time). The linear time comes from the upper bound of the LHS argument in the last three clauses. We can also check that the real parse time complexity of Example 3 is logarithmic in the length of the source text!

5 Closure Properties & Modularity

We shall show below that RCLs are closed under union, concatenation, Kleene iteration, intersection and complementation.

[11]A RCG is *bottom-up non-erasing* if, for each clause, all variables that occur in RHS also occur in LHS.

Let $G_1 = (N_1, T_1, V_1, P_1, S_2)$ and $G_2 = (N_2, T_2, V_2, P_2, S_2)$ be two RCGs defining the languages L_1 and L_2 respectively. Without loss of generality, we assume that $N_1 \cap N_2 = \emptyset$ and that S is a unary predicate name not in $N_1 \cup N_2$. Consider two RCGs $G' = (N_1 \cup \{S\}, T_1, V_1 \cup \{X\}, P_1 \cup P', S)$ and $G'' = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, V_1 \cup V_2 \cup \{X\}, P_1 \cup P_2 \cup P'', S)$ defining the languages L' and L'' respectively. By careful definition of the additional sets of clauses P' and P'', we can get $L'' = L_1 \cup L_2$, $L'' = L_1 L_2$ or $L'' = L_1 \cap L_2$ and $L' = L_1^*$ or $L' = \overline{L_1}$.

Union: $P'' = \{S(X) \rightarrow S_1(X), S(X) \rightarrow S_2(X)\}$

Concatenation: $P'' = \{S(XY) \rightarrow S_1(X)\ S_2(Y)\}$

Intersection: $P'' = \{S(X) \rightarrow S_1(X)\ S_2(X)\}$

Kleene iteration: $P' = \{S(\varepsilon) \rightarrow \varepsilon, S(XY) \rightarrow S_1(X)\ S(Y)\}$

Complementation: $P' = \{S(X) \rightarrow \overline{S_1(X)}\}$

In [Boullier, 1999d], we have shown that the emptiness problem for RCLs is undecidable and that RCLs are not closed under homomorphism. In fact, we have shown that a polynomial parse time formalism that extends CFGs cannot be closed both under homomorphism and intersection and we advocate that, for a NL description formalism, it is worth being closed under intersection rather than under homomorphism. This is specially true when this closure property is reached without changing the component grammars.

Let G_1 and G_2 be two grammars in some formalism $\mathcal{F}$, their sets of rules are P_1 and P_2 and they define the languages L_1 and L_2 respectively. We say that $\mathcal{F}$ is *modular* w.r.t. some closure operation f if the language $L = f(L_1, L_2)$ can be defined by a grammar G in $\mathcal{F}$ whose set of rules P is such that $P_1 \cup P_2 \subset P$. The idea behind this notion of sub-grammar is to preserve the structures (parse trees for G_1 and G_2) built by the component grammars. In that sense, we can say that CFGs are modular w.r.t. the union operation since CFGs have, on the one hand, the formal property to be closed under union and, on the other hand, this union is described without changing the component grammars G_1 and G_2 (we simply have to add the two rules $S \rightarrow S_1$ and $S \rightarrow S_2$). Conversely, CFGs are not modular w.r.t. intersection or complementation since we know that CFLs are not closed under intersection or complementation. If we now 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, let us 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.

Following our definition, we see that RCLs are modular w.r.t. union, concatenation, Kleene iteration, intersection and complementation. Of course it is of a considerable benefit for a formalism to be modular w.r.t. intersection and complementation.

Modularity w.r.t. intersection allows one to directly define a language with the properties $P_1 \wedge P_2$, 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) allows for example to model the paradigm "general rule with exceptions". Assume that we have a property P defined by a general rule R with some exceptions E to this general rule. Thus, formally we have $P = R - E = R \cap \overline{E}$.

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

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

assuming that P, R and E are unary predicate names. If, moreover, these exceptions are described by some rules say D, we simply have to add the clause

$$P(X) \quad \rightarrow \quad D(X)$$

6 Conclusion

In [Boullier, 1999d], we have shown that the 1-RCG subclass of RCGs with a single argument, is already a powerful extension of CFGs which can be parsed in cubic time and which contains both the intersection and the complement of CFLs. In [Boullier, 1999b&c], we have shown that unrestricted TAGs and set-local multi-component TAGs can be translated into equivalent PRCGs. Moreover, these transformations do not induce any over-cost. For example we have a linear parse time for regular CFGs, a cubic parse time for CFGs and a $\mathcal{O}(n^6)$ parse time for TAGs.

In this paper we present the full class of RCGs in which we can express several NL phenomena which are outside the formal power of MCS formalisms, 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. Moreover, in a given grammar, only complicated (many arguments, many variables) clauses produce higher parse times whereas simpler clauses induce lower times.

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

As CFGs, RCGs may themselves be considered as a syntactic backbone upon which other formalisms such as Herbrand's domain or feature structures can be grafted.

And lastly, we have seen that RCGs are modular This allows to imagine libraries of generic linguistic modules in which any language designer can pick up at will when he wants to specify such and such phenomena.

All these properties seem to advocate that RCGs might well have the right level of formal power needed in NL processing.

References

[Boullier, 1999a] Boullier P. (June 1999). Chinese Numbers, MIX, Scrambling, and Range Concatenation Grammars In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, Bergen, Norway. See also *Research Report No 3614* at `http://www.inria.fr/RRRT/RR-3614.html`, INRIA-Rocquencourt, France, Jan. 1999, 14 pages.

[Boullier, 1999b&c] Boullier P. (July 1999). On TAG Parsing *and* On Multicomponent TAG Parsing. In $6^{ème}$ *conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'99)*, Cargèse, Corse, France, pages 75–84 and pages 321–326. See also *Research Report No 3668* at `http://www.inria.fr/RRRT/RR-3668.html`, INRIA-Rocquencourt, France, Apr. 1999, 39 pages.

[Boullier, 1999d] Boullier P. (July 1999). A Cubic Time Extension of Context-Free Grammars In *Sixth Meeting on Mathematics of Language (MOL6)*, University of Central Florida, Orlando, Florida, USA. See also *Research Report No 3611* at `http://www.inria.fr/RRRT/RR-3611.html`, INRIA-Rocquencourt, France, Jan. 1999, 28 pages.

[Groenink, 1997] Groenink A. (Nov. 1997). Surface without Structure Word order and tractability in natural language analysis. PhD thesis, Utrecht University, The Nederlands, 250 pages.

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

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

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

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

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

[Weir, 1988] Weir D. (1988). Characterizing Mildly Context-Sensitive Grammar Formalisms. PhD thesis, University of Pennsylvania, Philadelphia, PA.