A Generalization of Mildly Context-Sensitive Formalisms

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

1 Introduction

In (Boullier 98), we presented range concatenation grammars (RCGs), a syntactic formalism which is a variant of literal movement grammars (LMGs), described in (Groenink 97), and which is also related to the framework of LFP developed by (Rounds 88). 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 rather than strings, and from their integer version the ability to handle only (part of) the source text. The basis of RCGs is the notion of *range*, a couple of integers $(i \mathrel{..} j)$ which denotes the occurrence of some substring $a_{i+1} \dots a_j$ in an input string $a_1 \dots a_n$. Of course, only consecutive ranges can be concatenated into a new range[1]. This formalism, which extends CFGs, aims at being a convincing challenger as a syntactic base for various tasks, especially in natural language processing. 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* mildly context-sensitive (Joshi, Vijay-Shanker, and Weir 91) and are more powerful than linear context-free rewriting systems (LCFRS) (Vijay-Shanker, Weir, and Joshi 87), while staying computationally tractable: its sentences can be parsed in polynomial time. However, our formalism shares with LCFRS the fact that derivations are context-free (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 parse forests (Lang 94). Let ρ be a range. The nodes of a CFG parse forest are couples (A, ρ) while for an RCG they have the form $(A, \vec{\rho})$ where $\vec{\rho}$ is a vector (list) of ranges. Besides its power and efficiency, this formalism possesses many other attractive properties. RCLs are closed under intersection and complementation[2]. Since this closure property can be reached without changing the structure (grammar) of the constituents (i.e. we can get the intersection of two grammars G_1 and G_2 without changing neither G_1 nor G_2), it allows for a form of modularity which may lead to the design of libraries of reusable generic grammatical components. Moreover, like CFGs, this formalism can act as a syntactic backbone upon which decorations from other domains (probabilities, logical terms, feature structures) can be grafted, and last, in our opinion, it is very elegant and understandable.

2 RCGs

The rewrite rules $\psi_0 \rightarrow \psi_1 \dots \psi_m$ of an RCG are called *clauses.* Each component $\psi_i = A(\alpha_1, \dots, \alpha_p)$ is a *predicate.* Each *argument* α_i of a predicate is a string of terminal symbols

[1] Ranges can be generalized to denote couples of states in some FSA representing ill-formed, incomplete or ambiguous (multi tagged/multi part of speech or word lattice) input.

[2] The set $T^* - L$, complementary of L, is defined on the basis of "negation by failure" rules.

and *variables.* Variables and arguments in a clause are supposed to be bound to ranges by a substitution mechanism. An *instantiated clause* is a clause in which arguments and variables are consistently replaced by ranges; its components are *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 an instantiation of the clause $A(aX, bYc, Zd) \rightarrow B(X, Y, Z)$ if the source text $a_1 \dots a_n$ is such that $a_{g+1} = a, a_{i+1} = b, a_j = c$ and $a_l = d$. A *derive* relation is defined on strings of instantiated predicates. If an instantiated predicate is the LHS of some instantiated clause, it can be replaced by the RHS of that instantiated clause. An input string $a_1 \dots a_n$ is a sentence iff the empty string (of instantiated predicates) can be derived from $S(\langle 0 .. n\rangle)$ where S is the start symbol. The arguments of predicates may denote discontinuous or even overlapping ranges. Fundamentally, a predicate A defines a notion (property, structure, dependency, ...) between its arguments whose ranges may be scattered over the source text. What is "between" its arguments is *not* the responsibility of A, and is described (if at all) somewhere else. RCGs are therefore well suited to describe long distance dependencies. Overlapping ranges are due to the non-linearity of the formalism. For example, the same variable may occur in different arguments in the RHS of some clause, expressing different views (properties) of the same portion of the source text.

As an example of an RCG, the following set of clauses describes the three-copy language $\{www \mid w \in \{a,b\}^*\}$ which is known to be beyond the formal power of TAGs.

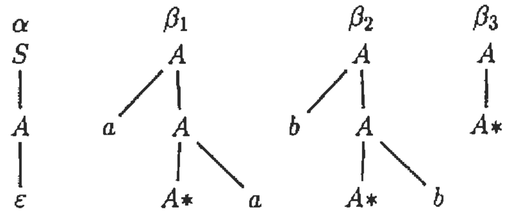
$$(\mathrm{I}) \quad \left\{ \begin{array}{lcl} S(XYZ) & \rightarrow & A(X,Y,Z) \\ A(aX,aY,aZ) & \rightarrow & A(X,Y,Z) \\ A(bX,bY,bZ) & \rightarrow & A(X,Y,Z) \\ A(\varepsilon,\varepsilon,\varepsilon) & \rightarrow & \varepsilon \end{array} \right.$$

3 RCGs & TAGs

Within the TAG formalism, if we consider an auxiliary tree τ and the way it evolves until no more adjunction/substitution is possible, we realize that some properties of the final tree are already known on τ. The yield derived by the part of τ to the left (resp. to the right) of its spine are contiguous and, the *left yield* (produced by the left part) lies to the left of the *right yield* in the input string. Thus, for any tree τ (initial or auxiliary) consider its m internal nodes where adjunction is allowed[3]. We decorate each such node with two variables L_i and R_i $(1 \leq i \leq m)$ which are supposed to capture respectively the left and right yield of this i^{th} node. The root and foot of auxiliary trees have no decoration. Each terminal leaf has a single decoration which is its terminal symbol or ε. Afterwards, we collect into a string d_τ the decorations gathered during a top-down left-to-right walk in τ. If τ is an auxiliary tree, let d^l_τ and d^r_τ be the part of d_τ gathered before and after the foot of τ has been hit. With each tree, we associate an RCG clause constructed as follows:

- Its LHS is the predicate $S(d_\tau)$ if τ is an initial tree (S is the start predicate).
- Its LHS is the predicate $A(d^l_\tau, d^r_\tau)$ if τ is an auxiliary A-tree.
- Its RHS is $\psi_1 \dots \psi_m$ with $\psi_i = A_i(L_i, R_i)$ if A_i is the label of the i^{th} inside node.

For example, the following TAG



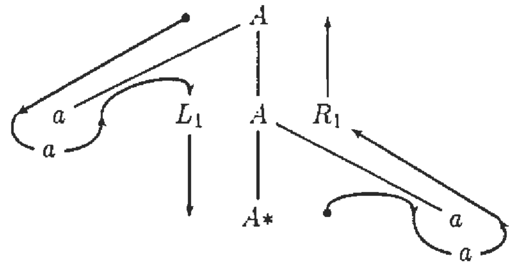
where α is the initial tree and β_1, β_2 and β_3 are the auxiliary trees[4], defines the language $\{ww \mid w \in \{a,b\}^*\}$, which is translated into the strongly equivalent RCG

$$\begin{array}{lcl} S(L_1R_1) & \rightarrow & A(L_1,R_1) \\ A(aL_1,aR_1) & \rightarrow & A(L_1,R_1) \\ A(bL_1,bR_1) & \rightarrow & A(L_1,R_1) \\ A(\varepsilon,\varepsilon) & \rightarrow & \varepsilon \end{array}$$

[3] In TAGs, we assumed that initial trees are all labeled by a unique start symbol, say S, which is not used somewhere else, that adjunction is not allowed at the root or at the foot of any auxiliary tree but is mandatory on inside nodes.

[4] Each foot is marked by an $*$.

As an example, the arguments of the LHS predicate of the second clause have been gathered during the following walk in β_1



We know (Vijay-Shanker and Weir 94) that TAGs, LIGs and HGs are three weakly equivalent formalisms though they appear to have quite different external forms. Groenink has shown that HGs can be translated into equivalent LMGs. We have shown that transformation from TAGs to RCGs also exists. In (Boullier 98) we have proposed a transformation from LIGs into equivalent RCGs. While the process involved to get an equivalent RCG for a TAG or an HG is rather straightforward, the equivalence proof for LIG is much more complex and relies upon our work described in (Boullier 96). This is due to the fact that an RCG is a purely syntactic formalism in the sense that it only handles (part of) the source text, exclusive of any other symbol. Therefore the stack symbols of LIGs have no direct equivalent in RCGs and the translation process needs to understand what the structural properties induced by these stack symbols are. An interesting property of all these translations is that the power of RCGs comes for free. In particular, if the input TAG or LIG is in some normal form[5], the corresponding RCG can be parsed in $\mathcal{O}(n^6)$ time at worst. Moreover, in RCGs, the incidence of each clause on the total parsing time can be isolated. Of course, complicated clauses induce high polynomial exponents. If we look at the clauses generated by the translation, some are simple, and few (if any) are complicated (and therefore induce an exponent of 6). In fact these translations bring new insight and help to understand why and at which point the maximum complexity is introduced.

4 RCGs & RNRGs

Ranked node rewriting grammars (RNRGs) (Abe and Mamitsuka 97) are an extension of TAGs. They are used to predict the protein secondary structure from their amino acid sequence patterns. These secondary structures, the so-called β-sheet regions in particular, form a kind of long distance dependency which can be captured by RNRGs. More precisely, it is a stochastic version of RNRGs which is used in this application[6]. The probability of each rewrite rule is set by training over a protein whose structure is known (corpus) and then used to analyze other proteins. RNRGs form a strictly growing hierarchy of grammars and languages (RNRLs) which is characterized by an integer called its *rank*. For any $k \geq 1$, $RNRL(k)$ properly contains $RNRL(k\text{-}1)$. $RNRL(0)$ are the CFLs and $RNRL(1)$ are the TALs.

An RNRG is a labeled tree rewriting system that consists of a *starting tree* and a finite set of rewriting rules, $A \rightarrow \alpha$, where A is a nonterminal symbol and α is a tree structure, which specifies how a node ν, labeled A, can be rewritten. Some leaves in α, called *empty leaves*, are labeled by a ♮ sign. Empty leaves are placeholders which indicate where the children of ν must be grafted. The number of children of ν and the number of empty leaves in α must be equal. This number is the *rank*. After rewriting, the children of a node are attached to these empty leaves in the same order as before rewriting. A tree whose nodes are only labeled by terminal symbols is a *terminal tree*. The *tree language* of an RNRG is the set of terminal trees which can be derived from the starting tree after a finite number of applications of its rewriting rules. Its *string language* is the set of yields of its tree language. Note that if an internal node is labeled by a terminal symbol, this node cannot be rewritten and its label does not contribute

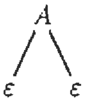
[5] Auxiliary trees in TAGS are such that there are at most two internal nodes where the adjunct operation can take place or the number of objects in the right-hand side of LIG rules is at most two.

[6] In fact, for computational considerations, only a subclass of RNRGs is processed.

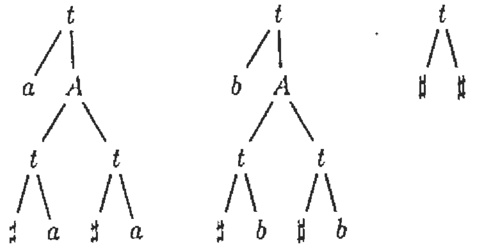
to the string language.

It is not difficult to transform an RNRG of any rank into an equivalent RCG. In fact the algorithm is a generalization of the one used for TAGs. Once again, no complexity penalty is induced by this transformation.

The previous three-copy language can be described by an RNRG of rank 2 whose initial tree is

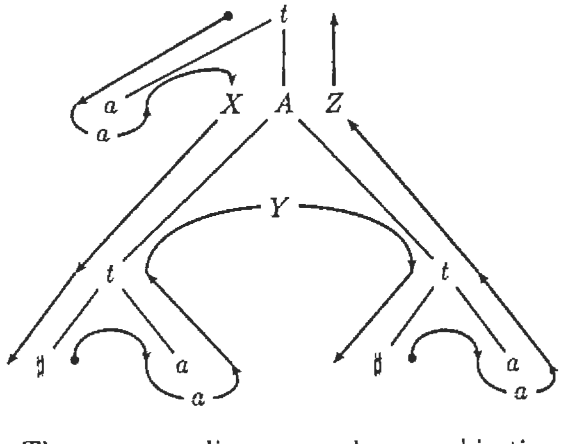


and the set of rewrite rules for the node A is



where t stands for an anonymous terminal symbol which labels non leaf nodes.

Our algorithm exactly yields the RCG labeled (I). As an example, the arguments of the LHS predicate of the second clause have been gathered during the following walk on the tree structure of the first rewrite rule for A. The variables X, Y and Z denote the left, bottom[7] and right environment of A.



The corresponding parser has a cubic time complexity. This global parsing time can be reduced to linear if we remark that the ranges substituted to the variables X, Y and Z in the first clause are of equal sizes. Such a property can be automatically discovered or explicitly specified.

[7] For a node with $l + 1$ sons, there will be $Y_1, \ldots Y_l$ "bottom" variables.

References

Naoki Abe, Hiroshi Mamitsuka. 1997. Predicting Protein Secondary Structure Using Stochastic Tree Grammars. In *Machine Learning*, 29, Dec. 1997, pages 275–301.

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.

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

Annius V. Groenink. 1997. Surface without Structure, word order and tractability in natural language analysis. PhD thesis, Utrecht University, The Nederlands, Nov. 1977, 250 pages.

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

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

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

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

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.