Another Facet of LIG Parsing

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

Abstract

In this paper[1] we present a new parsing algorithm for linear indexed grammars (LIGs) in the same spirit as the one described in (Vijay-Shanker and Weir, 1993) for tree adjoining grammars. For a LIG L and an input string x of length n, we build a non ambiguous context-free grammar whose sentences are all (and exclusively) valid derivation sequences in L which lead to x. We show that this grammar can be built in $\mathcal{O}(n^6)$ time and that individual parses can be extracted in linear time with the size of the extracted parse tree. Though this $\mathcal{O}(n^6)$ upper bound does not improve over previous results, the average case behaves much better. Moreover, practical parsing times can be decreased by some statically performed computations.

1 Introduction

The class of mildly context-sensitive languages can be described by several equivalent grammar types. Among these types we can notably cite tree adjoining grammars (TAGs) and linear indexed grammars (LIGs). In (Vijay-Shanker and Weir, 1994) TAGs are transformed into equivalent LIGs. Though context-sensitive linguistic phenomena seem to be more naturally expressed in TAG formalism, from a computational point of view, many authors think that LIGs play a central role and therefore the understanding of LIGs and LIG parsing is of importance. For example, quoted from (Schabes and Shieber, 1994) "The LIG version of TAG can be used for recognition and parsing. Because the LIG formalism is based on augmented rewriting, the parsing algorithms can be much simpler to understand and easier to modify, and no loss of generality is incurred". In (Vijay-Shanker and Weir, 1993) LIGs are used to express the derivations of a sentence in TAGs. In (Vijay-Shanker, Weir and Rambow, 1995) the approach used for parsing a new formalism, the D-Tree Grammars (DTG), is to translate a DTG into a Linear Prioritized Multiset Grammar which is similar to a LIG but uses multisets in place of stacks.

LIGs can be seen as usual context-free grammars (CFGs) upon which constraints are imposed. These constraints are expressed by stacks of symbols associated with non-terminals. We study parsing of LIGs, our goal being to define a structure that verifies the LIG constraints and codes all (and exclusively) parse trees deriving sentences.

Since derivations in LIGs are constrained CF derivations, we can think of a scheme where the CF derivations for a given input are expressed by a shared forest from which individual parse trees which do not satisfied the LIG constraints are erased. Unhappily this view is too simplistic, since the erasing of individual trees whose parts can be shared with other valid trees can only be performed after some unfolding (unsharing) that can produced a forest whose size is exponential or even unbounded.

In (Vijay-Shanker and Weir, 1993), the context-freeness of adjunction in TAGs is captured by giving a CFG to represent the set of all possible derivation sequences. In this paper we study a new parsing scheme for LIGs based upon similar principles and which, on the other side, emphasizes as (Lang, 1991) and (Lang, 1994), the use of grammars (shared forest) to represent parse trees and is an extension of our previous work (Boullier, 1995).

This previous paper describes a recognition algorithm for LIGs, but not a parser. For a LIG and an input string, all valid parse trees are actually coded into the CF shared parse forest used by this recognizer, but, on some parse trees of this forest, the

[1] See (Boullier, 1996) for an extended version.

checking of the LIG constraints can possibly failed. At first sight, there are two conceivable ways to extend this recognizer into a parser:

1. only "good" trees are kept;

2. the LIG constraints are [re-]checked while the extraction of valid trees is performed.

As explained above, the first solution can produce an unbounded number of trees. The second solution is also uncomfortable since it necessitates the reevaluation on each tree of the LIG conditions and, doing so, we move away from the usual idea that individual parse trees can be extracted by a simple walk through a structure.

In this paper, we advocate a third way which will use (see section 4), the same basic material as the one used in (Boullier, 1995). For a given LIG L and an input string x, we exhibit a non ambiguous CFG whose sentences are all possible valid derivation sequences in L which lead to x. We show that this CFG can be constructed in $\mathcal{O}(n^6)$ time and that individual parses can be extracted in time linear with the size of the extracted tree.

2 Derivation Grammar and CF Parse Forest

In a CFG $G = (V_N, V_T, P, S)$, the *derives* relation $\underset{G}{\Rightarrow}$ is the set $\{(\sigma B \sigma', \sigma \beta \sigma') \mid B \rightarrow \beta \in P \wedge V = V_N \cup V_T \wedge \sigma, \sigma' \in V^*\}$. A *derivation* is a sequence of strings in V^* s.t. the relation derives holds between any two consecutive strings. In a *rightmost* derivation, at each step, the rightmost non-terminal say B is replaced by the right-hand side (RHS) of a B-production. Equivalently if $\sigma_0 \underset{G}{\overset{r_1}{\Rightarrow}} \ldots \underset{G}{\overset{r_n}{\Rightarrow}} \sigma_n$ is a rightmost derivation where the relation symbol is overlined by the production used at each step, we say that $r_1 \ldots r_n$ is a rightmost σ_0/σ_n-derivation.

For a CFG G, the set of its rightmost S/x-derivations, where $x \in \mathcal{L}(G)$, can itself be defined by a grammar.

Definition 1 *Let $G = (V_N, V_T, P, S)$ be a CFG, its rightmost derivation grammar is the CFG $D = (V_N, P, P^D, S)$ where $P^D = \{A_0 \rightarrow A_1 \ldots A_q r \mid r = A_0 \rightarrow w_0 A_1 w_1 \ldots w_{q-1} A_q w_q \in P \wedge w_i \in V_T^* \wedge A_j \in V_N\}$*

¿From the natural bijection between P and P^D, we can easily prove that

$$\mathcal{L}(D) = \{r_n \ldots r_1 \mid \\ r_1 \ldots r_n \text{ is a rightmost } S/x\text{-derivation in } G\}$$

This shows that the rightmost derivation language of a CFG is also CF. We will show in section 4 that a similar result holds for LIGs.

Following (Lang, 1994), CF parsing is the intersection of a CFG and a finite-state automaton (FSA) which models the input string x[2]. The result of this intersection is a CFG $G^x = (V_N^x, V_T^x, P^x, [S]_0^n)$ called a *shared parse forest* which is a specialization of the initial CFG $G = (V_N, V_T, P, S)$ to x. Each production $r_i^j \in P^x$, is the production $r_i \in P$ up to some non-terminal renaming. The non-terminal symbols in V_N^x are triples denoted $[A]_p^q$ where $A \in V_N$, and p and q are states. When such a non-terminal is productive, $[A]_p^q \underset{G^x}{\overset{+}{\Rightarrow}} w$, we have $q \in \delta(p, w)$.

If we build the rightmost derivation grammar associated with a shared parse forest, and we remove all its useless symbols, we get a reduced CFG say D^x. The CF recognition problem for (G, x) is equivalent to the existence of an $[S]_0^n$-production in D^x. Moreover, each rightmost S/x-derivation in G is (the reverse of) a sentence in $\mathcal{L}(D^x)$. However, this result is not very interesting since individual parse trees can be as easily extracted directly from the parse forest. This is due to the fact that in the CF case, a tree that is derived (a parse tree) contains all the information about its derivation (the sequence of rewritings used) and therefore there is no need to distinguish between these two notions. Though this is not always the case with non CF formalisms, we will see in the next sections that a similar approach, when applied to LIGs, leads to a shared parse forest which is a LIG while it is possible to define a derivation grammar which is CF.

3 Linear Indexed Grammars

An indexed grammar is a CFG in which stack of symbols are associated with non-terminals. LIGs are a restricted form of indexed grammars in which the dependence between stacks is such that at most one stack in the RHS of a production is related with the stack in its LHS. Other non-terminals are associated with independant stacks of bounded size.

Following (Vijay-Shanker and Weir, 1994)

Definition 2 *$L = (V_N, V_T, V_I, P_L, S)$ denotes a LIG where V_N, V_T, V_I and P_L are respectively finite sets of non-terminals, terminals, stack symbols and productions, and S is the start symbol.*

In the sequel we will only consider a restricted

[2] if $x = a_1 \ldots a_n$, the states can be the integers $0 \ldots n$, 0 is the initial state, n the unique final state, and the transition function δ is s.t. $i \in \delta(i-1, a_i)$ and $i \in \delta(i, \varepsilon)$.

form of LIGs with productions of the form

$$P_L \quad = \quad \{A() \to w\} \cup \{A(..\alpha) \to \Gamma_1 B(..\alpha')\Gamma_2\}$$

where $A, B \in V_N$, $w \in V_T^* \wedge 0 \leq |w| \leq 2$, $\alpha\alpha' \in V_I^* \wedge 0 \leq |\alpha\alpha'| \leq 1$ and $\Gamma_1\Gamma_2 \in V_T \cup \{\varepsilon\} \cup \{C() \mid C \in V_N\}$.

An element like $A(..\alpha)$ is a *primary constituent* while $C()$ is a *secondary constituent*. The stack schema $(..\alpha)$ of a primary constituent matches all the stacks whose prefix (bottom) part is left unspecified and whose suffix (top) part is α; the stack of a secondary constituent is always empty.

Such a form has been chosen both for complexity reasons and to decrease the number of cases we have to deal with. However, it is easy to see that this form of LIG constitutes a normal form.

We use $r()$ to denote a production in P_L, where the parentheses remind us that we are in a LIG!

The *CF-backbone* of a LIG is the underlying CFG in which each production is a LIG production where the stack part of each constituent has been deleted, leaving only the non-terminal part. We will only consider LIGs such there is a bijection between its production set and the production set of its CF-backbone[3].

We call *object* the pair denoted $A(\alpha)$ where A is a non-terminal and (α) a stack of symbols. Let $V_O = \{A(\alpha) \mid A \in V_N \wedge \alpha \in V_I^*\}$ be the set of objects. We define on $(V_O \cup V_T)^*$ the binary relation *derives* denoted $\underset{L}{\Rightarrow}$ (the relation symbol is sometimes overlined by a production):

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

$$\Gamma_1' A()\Gamma_2' \quad \overset{A()\to w}{\underset{L}{\Rightarrow}} \quad \Gamma_1' w \Gamma_2'$$

In the first above element we say that the object $B(\alpha''\alpha')$ is the *distinguished child* of $A(\alpha''\alpha)$, and if $\Gamma_1\Gamma_2 = C()$, $C()$ is the *secondary object*. A *derivation* $\Gamma_1, \ldots, \Gamma_i, \Gamma_{i+1}, \ldots, \Gamma_l$ is a sequence of strings where the relation derives holds between any two consecutive strings

The language defined by a LIG L is the set:

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

As in the CF case we can talk of rightmost derivations when the rightmost object is derived at each step. Of course, many other derivation strategies may be thought of. For our parsing algorithm, we need such a particular derives relation. Assume that at one step an object derives both a distinguished child and a secondary object. Our particular derivation strategy is such that this distinguished child will always be derived after the secondary object (and its descendants), whether this secondary object lays to its left or to its right. This derives relation is denoted $\underset{\ell,L}{\Rightarrow}$ and is called *linear*[4].

A *spine* is the sequence of objects $A_1(\alpha_1) \ldots A_i(\alpha_i)\ A_{i+1}(\alpha_{i+1}) \ldots A_p(\alpha_p)$ if, there is a derivation in which each object $A_{i+1}(\alpha_{i+1})$ is the distinguished child of $A_i(\alpha_i)$ (and therefore the *distinguished descendant* of $A_j(\alpha_j), 1 \leq j \leq i$).

4 Linear Derivation Grammar

For a given LIG L, consider a linear $S()/x$-derivation

$$S() \overset{r_n()}{\underset{\ell,L}{\Rightarrow}} \cdots \overset{r_i()}{\underset{\ell,L}{\Rightarrow}} \cdots \overset{r_1()}{\underset{\ell,L}{\Rightarrow}} x$$

The sequence of productions $r_1() \ldots r_i() \ldots r_n()$ (considered in reverse order) is a string in P_L^*. The purpose of this section is to define the set of such strings as the language defined by some CFG.

Associated with a LIG $L = (V_N, V_T, V_I, P_L, S)$, we first define a bunch of binary relations which are borrowed from (Boullier, 1995)

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

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

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 holds: $\alpha_i = \alpha_{i+1}$, $\alpha_i\gamma = \alpha_{i+1}$, or $\alpha_i = \alpha_{i+1}\gamma$.

The $\underset{+}{-\!\diamond\!-}$ relation select pairs of non-terminals (A_1, A_p) s.t. $\alpha_1 = \alpha_p = \varepsilon$ along non trivial spines.

[3] r_p and $r_p()$ with the same index p designate associated productions.

[4] linear reminds us that we are in a LIG and relies upon a linear (total) order over object occurrences in a derivation. See (Boullier, 1996) for a more formal definition.

If the relations $\overset{\gamma}{\underset{+}{\succ}}$ and $\approx$ are defined as $\overset{\gamma}{\underset{+}{\succ}} = \overset{\gamma}{\underset{1}{\succ}} \cup \underset{+}{\Diamond}\overset{\gamma}{\underset{1}{\succ}}$ and $\approx = \bigcup_{\gamma \in V_I} \overset{\gamma}{\underset{1}{\prec}}\overset{\gamma}{\underset{+}{\succ}}$, we can see that the following identity holds

Property 1

$$\underset{+}{\Diamond} \;=\; \underset{1}{\Diamond} \cup \approx \cup \underset{1}{\Diamond}\underset{+}{\Diamond} \cup \approx \underset{+}{\Diamond}$$

In (Boullier, 1995) we can found an algorithm[5] which computes the $\underset{+}{\Diamond}$, $\overset{\gamma}{\underset{+}{\succ}}$ and $\approx$ relations as the composition of $\underset{1}{\Diamond}$, $\overset{\gamma}{\underset{1}{\prec}}$ and $\overset{\gamma}{\underset{1}{\succ}}$ in $\mathcal{O}(|V_N|^3)$ time.

Definition 3 *For a LIG $L = (V_N, V_T, V_I, P_L, S)$, we call linear derivation grammar (LDG) the CFG D_L (or D when L is understood) $D = (V_N^D, V_T^D, P^D, S^D)$ where*

- $V_N^D = \{[A] \mid A \in V_N\} \cup \{[A\rho B] \mid A, B \in V_N \wedge \rho \in \mathcal{R}\}$, *and $\mathcal{R}$ is the set of relations* $\{\overset{\gamma}{\underset{1}{\prec}}, \underset{1}{\Diamond}, \overset{\gamma}{\underset{1}{\succ}}, \underset{+}{\Diamond}, \approx, \overset{\gamma}{\underset{+}{\succ}}\}$[6]
- $V_T^D = P_L$
- $S^D = [S]$
- *Below, $[\Gamma_1\Gamma_2]$ denotes either the non-terminal symbol $[X]$ when $\Gamma_1\Gamma_2 = X()$ or the empty string ε when $\Gamma_1\Gamma_2 \in V_T^*$. P^D is defined as being*

$$\begin{aligned}
&\{[A] \to r() \mid r() = A() \to w \in P_L\} && (1)\\
&\cup\{[A] \to r()[A \underset{+}{\Diamond} B] \mid\\
&\qquad r() = B() \to w \in P_L\} && (2)\\
&\cup\{[A \underset{+}{\Diamond} C] \to [\Gamma_1\Gamma_2]r() \mid\\
&\qquad r() = A(..) \to \Gamma_1 C(..)\Gamma_2 \in P_L\} && (3)\\
&\cup\{[A \underset{+}{\Diamond} C] \to [A \approx C]\} && (4)\\
&\cup\{[A \underset{+}{\Diamond} C] \to [B \underset{+}{\Diamond} C][\Gamma_1\Gamma_2]r() \mid\\
&\qquad r() = A(..) \to \Gamma_1 B(..)\Gamma_2 \in P_L\} && (5)\\
&\cup\{[A \underset{+}{\Diamond} C] \to [B \underset{+}{\Diamond} C][A \approx B]\} && (6)\\
&\cup\{[A \approx C] \to [B \overset{\gamma}{\underset{+}{\succ}} C][\Gamma_1\Gamma_2]r() \mid\\
&\qquad r() = A(..) \to \Gamma_1 B(..\gamma)\Gamma_2 \in P_L\} && (7)\\
&\cup\{[A \overset{\gamma}{\underset{+}{\succ}} C] \to [\Gamma_1\Gamma_2]r() \mid\\
&\qquad r() = A(..\gamma) \to \Gamma_1 C(..)\Gamma_2 \in P_L\} && (8)\\
&\cup\{[A \overset{\gamma}{\underset{+}{\succ}} C] \to [\Gamma_1\Gamma_2]r()[A \underset{+}{\Diamond} B] \mid\\
&\qquad r() = B(..\gamma) \to \Gamma_1 C(..)\Gamma_2 \in P_L\} && (9)
\end{aligned}$$

The productions in P^D define all the ways linear derivations can be composed from linear subderivations. This compositions rely on one side upon property 1 (recall that the productions in P_L, must be produced in reverse order) and, on the other side, upon the order in which secondary spines (the $\Gamma_1\Gamma_2$-spines) are processed to get the linear derivation order.

In (Boullier, 1996), we prove that LDGs are not ambiguous (in fact they are SLR(1)) and define

$$\begin{aligned}\mathcal{L}(D) \;=\; &\{r_1()\dots r_n() \mid S() \overset{r_n()}{\underset{\ell,L}{\Rightarrow}} \dots \overset{r_1()}{\underset{\ell,L}{\Rightarrow}} x\\ &\wedge x \in \mathcal{L}(L)\}\end{aligned}$$

If, by some classical algorithm, we remove from D all its useless symbols, we get a reduced CFG say $D' = (V_N^{D'}, V_T^{D'}, P^{D'}, S^{D'})$. In this grammar, all its terminal symbols, which are productions in L, are useful. By the way, the construction of D' solve the emptiness problem for LIGs: L specify the empty set iff the set $V_T^{D'}$ is empty[7].

5 LIG parsing

Given a LIG $L = (V_N, V_T, V_I, P_L, S)$ we want to find all the syntactic structures associated with an input string $x \in V_T^*$. In section 2 we used a CFG (the shared parse forest) for representing all parses in a CFG. In this section we will see how to build a CFG which represents all parses in a LIG.

In (Boullier, 1995) we give a recognizer for LIGs with the following scheme: in a first phase a general CF parsing algorithm, working on the CF-backbone builds a shared parse forest for a given input string x. In a second phase, the LIG conditions are checked on this forest. This checking can result in some subtree (production) deletions, namely the ones for which there is no valid symbol stack evaluation. If the resulting grammar is not empty, then x is a sentence. However, in the general case, this resulting grammar is not a shared parse forest for the initial LIG in the sense that the computation of stack of symbols along spines are not guaranteed to be consistent. Such invalid spines are not deleted during the check of the LIG conditions because they could be

[5] Though in the referred paper, these relations are defined on constituents, the algorithm also applies to non-terminals.

[6] In fact we will only use *valid* non-terminals $[A\rho B]$ for which the relation ρ holds between A and B.

[7] In (Vijay-Shanker and Weir, 1993) the emptiness problem for LIGs is solved by constructing an FSA.

composed of sub-spines which are themselves parts of other valid spines. One way to solve this problem is to unfold the shared parse forest and to extract individual parse trees. A parse tree is then kept iff the LIG conditions are valid on that tree. But such a method is not practical since the number of parse trees can be unbounded when the CF-backbone is cyclic. Even for non cyclic grammars, the number of parse trees can be exponential in the size of the input. Moreover, it is problematic that a worst case polynomial size structure could be reached by some sharing compatible both with the syntactic and the "semantic" features.

However, we know that derivations in TAGs are context-free (see (Vijay-Shanker, 1987)) and (Vijay-Shanker and Weir, 1993) exhibits a CFG which represents all possible derivation sequences in a TAG. We will show that the analogous holds for LIGs and leads to an $\mathcal{O}(n^6)$ time parsing algorithm.

Definition 4 *Let $L = (V_N, V_T, V_I, P_L, S)$ be a LIG, $G = (V_N, V_T, P_G, S)$ its CF-backbone, x a string in $\mathcal{L}(G)$, and $G^x = (V_N^x, V_T^x, P_G^x, S^x)$ its shared parse forest for x. We define the LIGed forest for x as being the LIG $L^x = (V_N^x, V_T^x, V_I, P_L^x, S^x)$ s.t. G^x is its CF-backbone and its productions are the productions of P_G^x in which the corresponding stack-schemas of L have been added. For example $r_p^q() = [A]_i^k(..\alpha) \rightarrow [B]_i^j(..\alpha')[C]_j^k() \in P_L^x$ iff $r_p^q = [A]_i^k \rightarrow [B]_i^j[C]_j^k \in P_G^x \wedge r_p = A \rightarrow BC \in G \wedge r_p() = A(..\alpha) \rightarrow B(..\alpha')C() \in L$.*

Between a LIG L and its LIGed forest L^x for x, we have:

$$x \in \mathcal{L}(L) \iff x \in \mathcal{L}(L^x)$$

If we follow(Lang, 1994), the previous definition which produces a LIGed forest from any L and x is a (LIG) parser[8]: given a LIG L and a string x, we have constructed a new LIG L^x for the intersection $\mathcal{L}(L) \cap \{x\}$, which is the shared forest for all parses of the sentences in the intersection. However, we wish to go one step further since the parsing (or even recognition) problem for LIGs cannot be trivially extracted from the LIGed forests.

Our vision for the parsing of a string x with a LIG L can be summarized in few lines. Let G be the CF-backbone of L, we first build G^x the CFG shared parse forest by any classical general CF parsing algorithm and then L^x its LIGed forest. Afterwards, we build the reduced LDG D_{L^x} associated with L^x as shown in section 4.

The recognition problem for (L, x) (i.e. is x an element of $\mathcal{L}(L)$) is equivalent to the non-emptiness of the production set of D_{L^x}.

Moreover, each linear $S()/x$-derivation in L is (the reverse of) a string in $\mathcal{L}(D_{L^x})$[9]. So the extraction of individual parses in a LIG is merely reduced to the derivation of strings in a CFG.

An important issue is about the complexity, in time and space, of D_{L^x}. Let n be the length of the input string x. Since G is in binary form we know that the shared parse forest G^x can be build in $\mathcal{O}(n^3)$ time and the number of its productions is also in $\mathcal{O}(n^3)$. Moreover, the cardinality of V_N^x is $\mathcal{O}(n^2)$ and, for any given non-terminal, say $[A]_p^q$, there are at most $\mathcal{O}(n)$ $[A]_p^q$-productions. Of course, these complexities extend to the LIGed forest L^x.

We now look at the LDG complexity when the input LIG is a LIGed forest. In fact, we mainly have to check two forms of productions (see definition 3). The first form is production (6) ($[A \underset{+}{\prec\!\succ} C] \rightarrow [B \underset{+}{\prec\!\succ} C][A \approx B]$), where three different non-terminals in V_N are implied (i.e. A, B and C), so the number of productions of that form is cubic in the number of non-terminals and therefore is $\mathcal{O}(n^6)$.

In the second form (productions (5), (7) and (9)), exemplified by $[A \approx C] \rightarrow [B \overset{\gamma}{\underset{+}{\succ}} C][\Gamma_1\Gamma_2]r()$, there are four non-terminals in V_N (i.e. A, B, C, and X if $\Gamma_1\Gamma_2 = X()$) and a production $r()$ (the number of relation symbols $\overset{\gamma}{\underset{+}{\succ}}$ is a constant), therefore, the number of such productions seems to be of fourth degree in the number of non-terminals and linear in the number of productions. However, these variables are not independant. For a given A, the number of triples $(B, X, r())$ is the number of A-productions hence $\mathcal{O}(n)$. So, at the end, the number of productions of that form is $\mathcal{O}(n^5)$.

We can easily check that the other form of productions have a lesser degree.

Therefore, the number of productions is dominated by the first form and the size (and in fact the construction time) of this grammar is $\mathcal{O}(n^6)$.

This (once again) shows that the recognition and parsing problem for a LIG can be solved in $\mathcal{O}(n^6)$ time.

For a LDG $D = (V_N^D, V_T^D, P^D, S^D)$, we note that for any given non-terminal $A \in V_N^D$ and string $\sigma \in \mathcal{L}(A)$ with $|\sigma| \geq 2$, a single production $A \rightarrow X_1X_2$ or $A \rightarrow X_1X_2X_3$ in P^D is needed to "cut" σ into two or three non-empty pieces σ_1, σ_2, and σ_3, such that

[8] Of course, instead of x, we can consider any FSA.

[9] In fact, the terminal symbols in D_{L^x} are productions in L^x (say $R_p^q()$), which trivially can be mapped to productions in L (here $r_p()$).

$X_i \overset{*}{\underset{D}{\Rightarrow}} \sigma_i$, except when the production form number (4) is used. In such a case, this cutting needs two productions (namely (4) and (7)). This shows that the cutting out of any string of length l, into elementary pieces of length 1, is performed in using $\mathcal{O}(l)$ productions. Therefore, the extraction of a linear $S()/x$-derivation in L is performed in time linear with the length of that derivation. If we assume that the CF-backbone G is non cyclic, the extraction of a parse is linear in n. Moreover, during an extraction, since D_{L^x} is not ambiguous, at some place, the choice of another A-production will result in a different linear derivation.

Of course, practical generations of LDGs must improve over a blind application of definition 3. One way is to consider a top-down strategy: the X-productions in a LDG are generated iff X is the start symbol or occurs in the RHS of an already generated production. The examples in section 6 are produced this way.

If the number of ambiguities in the initial LIG is bounded, the size of D_{L^x}, for a given input string x of length n, is linear in n.

The size and the time needed to compute D_{L^x} are closely related to the actual sizes of the $\underset{+}{-\!\!\diamond\!\!-}$, $\overset{\gamma}{\underset{+}{\succ}}$ and $\approx$ relations. As pointed out in (Boullier, 1995), their $\mathcal{O}(n^4)$ maximum sizes seem to be seldom reached in practice. This means that the average parsing time is much better than this $\mathcal{O}(n^6)$ worst case.

Moreover, our parsing schema allow to avoid some useless computations. Assume that the symbol $[A \underset{+}{-\!\!\diamond\!\!-} B]$ is useless in the LDG D_L associated with the initial LIG L, we know that any non-terminal s.t. $[[A]_i^j \underset{+}{-\!\!\diamond\!\!-} [B]_k^l]$ is also useless in D_{L^x}. Therefore, the static computation of a reduced LDG for the initial LIG L (and the corresponding $\underset{+}{-\!\!\diamond\!\!-}$, $\overset{\gamma}{\underset{+}{\succ}}$ and $\approx$ relations) can be used to direct the parsing process and decrease the parsing time (see section 6).

6 Two Examples

6.1 First Example

In this section, we illustrate our algorithm with a LIG $L = (\{S, T\}, \{a, b, c\}, \{\gamma_a, \gamma_b, \gamma_c\}, P_L, S)$ where P_L contains the following productions:

$$\begin{array}{ll}
r_1() = S(..) \rightarrow S(..\gamma_a)a & r_2() = S(..) \rightarrow S(..\gamma_b)b \\
r_3() = S(..) \rightarrow S(..\gamma_c)c & r_4() = S(..) \rightarrow T(..) \\
r_5() = T(..\gamma_a) \rightarrow aT(..) & r_6() = T(..\gamma_b) \rightarrow bT(..) \\
r_7() = T(..\gamma_c) \rightarrow cT(..) & r_8() = T() \rightarrow c
\end{array}$$

It is easy to see that its CF-backbone G, whose production set P_G is:

$$\begin{array}{llll}
S \rightarrow Sa & S \rightarrow Sb & S \rightarrow Sc & S \rightarrow T \\
T \rightarrow aT & T \rightarrow bT & T \rightarrow cT & T \rightarrow c
\end{array}$$

defines the language $\mathcal{L}(G) = \{wcw' \mid w, w' \in \{a, b, c\}^*\}$. We remark that the stacks of symbols in L constrain the string w' to be equal to w and therefore the language $\mathcal{L}(L)$ is $\{wcw \mid w \in \{a, b, c\}^*\}$.

We note that in L the key part is played by the middle c, introduced by production $r_8()$, and that this grammar is non ambiguous, while in G the symbol c, introduced by the last production $T \rightarrow c$, is only a separator between w and w' and that this grammar is ambiguous (any occurrence of c may be this separator).

The computation of the relations gives:

$$\begin{array}{ccccccc}
 & & & & \underset{1}{-\!\!\diamond\!\!-} & = & \{(S, T)\} \\
\underset{1}{\overset{\gamma_a}{\prec}} & = & \underset{1}{\overset{\gamma_b}{\prec}} & = & \underset{1}{\overset{\gamma_c}{\prec}} & = & \{(S, S)\} \\
\underset{1}{\overset{\gamma_a}{\succ}} & = & \underset{1}{\overset{\gamma_b}{\succ}} & = & \underset{1}{\overset{\gamma_c}{\succ}} & = & \{(T, T)\} \\
 & & & & \underset{+}{-\!\!\diamond\!\!-} & = & \{(S, T)\} \\
 & & & & \approx & = & \{(S, T)\} \\
\underset{+}{\overset{\gamma_a}{\succ}} & = & \underset{+}{\overset{\gamma_b}{\succ}} & = & \underset{+}{\overset{\gamma_c}{\succ}} & = & \{(T, T), (S, T)\}
\end{array}$$

The production set P^D of the LDG D associated with L is:

$$\begin{array}{lcll}
[S] & \rightarrow & r_8()[S \underset{+}{-\!\!\diamond\!\!-} T] & (2) \\
[S \underset{+}{-\!\!\diamond\!\!-} T] & \rightarrow & r_4() & (3) \\
[S \underset{+}{-\!\!\diamond\!\!-} T] & \rightarrow & [S \approx T] & (4) \\
[S \approx T] & \rightarrow & [S \underset{+}{\overset{\gamma_a}{\succ}} T]r_1() & (7) \\
[S \approx T] & \rightarrow & [S \underset{+}{\overset{\gamma_b}{\succ}} T]r_2() & (7) \\
[S \approx T] & \rightarrow & [S \underset{+}{\overset{\gamma_c}{\succ}} T]r_3() & (7) \\
[S \underset{+}{\overset{\gamma_a}{\succ}} T] & \rightarrow & r_5()[S \underset{+}{-\!\!\diamond\!\!-} T] & (9) \\
[S \underset{+}{\overset{\gamma_b}{\succ}} T] & \rightarrow & r_6()[S \underset{+}{-\!\!\diamond\!\!-} T] & (9) \\
[S \underset{+}{\overset{\gamma_c}{\succ}} T] & \rightarrow & r_7()[S \underset{+}{-\!\!\diamond\!\!-} T] & (9)
\end{array}$$

The numbers (i) refer to definition 3. We can easily checked that this grammar is reduced.

Let $x = ccc$ be an input string. Since x is an element of $\mathcal{L}(G)$, its shared parse forest G^x is not empty. Its production set P_G^x is:

$$\begin{array}{ll}
r_3^1 = [S]_0^3 \rightarrow [S]_0^2 c & r_4^2 = [S]_0^3 \rightarrow [T]_0^3 \\
r_3^3 = [S]_0^2 \rightarrow [S]_0^1 c & r_4^4 = [S]_0^2 \rightarrow [T]_0^2 \\
r_4^5 = [S]_0^1 \rightarrow [T]_0^1 & r_7^6 = [T]_0^3 \rightarrow c[T]_1^3 \\
r_7^7 = [T]_1^3 \rightarrow c[T]_2^3 & r_8^8 = [T]_2^3 \rightarrow c \\
r_7^9 = [T]_0^2 \rightarrow c[T]_1^2 & r_8^{10} = [T]_1^2 \rightarrow c \\
r_8^{11} = [T]_0^1 \rightarrow c &
\end{array}$$

We can observe that this shared parse forest denotes in fact three different parse trees. Each one corresponding to a different cutting out of $x = wcw'$ (i.e. $w = \varepsilon$ and $w' = cc$, or $w = c$ and $w' = c$, or $w = cc$ and $w' = \varepsilon$).

The corresponding LIGed forest whose start symbol is $S^x = [S]_0^3$ and production set P_L^x is:

$$\begin{array}{llll}
r_3^1() & = & [S]_0^3(..) & \rightarrow & [S]_0^2(..\gamma_c)c \\
r_4^2() & = & [S]_0^3(..) & \rightarrow & [T]_0^3(..) \\
r_3^3() & = & [S]_0^2(..) & \rightarrow & [S]_0^1(..\gamma_c)c \\
r_4^4() & = & [S]_0^2(..) & \rightarrow & [T]_0^2(..) \\
r_4^5() & = & [S]_0^1(..) & \rightarrow & [T]_0^1(..) \\
r_7^6() & = & [T]_0^3(..\gamma_c) & \rightarrow & c[T]_1^3(..) \\
r_7^7() & = & [T]_1^3(..\gamma_c) & \rightarrow & c[T]_2^3(..) \\
r_8^8() & = & [T]_2^3() & \rightarrow & c \\
r_7^9() & = & [T]_0^2(..\gamma_c) & \rightarrow & c[T]_1^2(..) \\
r_8^{10}() & = & [T]_1^2() & \rightarrow & c \\
r_8^{11}() & = & [T]_0^1() & \rightarrow & c
\end{array}$$

For this LIGed forest the relations are:

$$\begin{array}{lll}
\underset{1}{\diamond} & = & \{([S]_0^3, [T]_0^3), ([S]_0^2, [T]_0^2), ([S]_0^1, [T]_0^1)\} \\
\overset{\gamma_c}{\underset{1}{\prec}} & = & \{([S]_0^3, [S]_0^2), ([S]_0^2, [S]_0^1)\} \\
\overset{\gamma_c}{\underset{1}{\succ}} & = & \{([T]_0^3, [T]_1^3), ([T]_1^3, [T]_2^3), ([T]_0^2, [T]_1^2)\} \\
\approx & = & \{([S]_0^3, [T]_1^2)\} \\
\underset{+}{\diamond} & = & \underset{1}{\diamond} \cup \approx \\
\overset{\gamma_c}{\underset{+}{\succ}} & = & \overset{\gamma_c}{\underset{1}{\succ}} \cup \{([S]_0^3, [T]_1^3), ([S]_0^2, [T]_1^2)\}
\end{array}$$

The start symbol of the LDG associated with the LIGed forest L^x is $[[S]_0^3]$. If we assume that an A-production is generated iff it is an $[[S]_0^3]$-production or A occurs in an already generated production, we get:

$$\begin{array}{llll}
[[S]_0^3] & \rightarrow & r_8^{10}()[[S]_0^3 \underset{+}{\diamond} [T]_1^2] & (2) \\
[[S]_0^3 \underset{+}{\diamond} [T]_1^2] & \rightarrow & [[S]_0^3 \approx [T]_1^2] & (4) \\
[[S]_0^3 \approx [T]_1^2] & \rightarrow & [[S]_0^2 \overset{\gamma_c}{\underset{+}{\succ}} [T]_1^2]r_3^1() & (7) \\
[[S]_0^2 \overset{\gamma_c}{\underset{+}{\succ}} [T]_1^2] & \rightarrow & r_7^9()[[S]_0^2 \underset{+}{\diamond} [T]_0^2] & (9) \\
[[S]_0^2 \underset{+}{\diamond} [T]_0^2] & \rightarrow & r_4^4() & (3)
\end{array}$$

This CFG is reduced. Since its production set is non empty, we have $ccc \in \mathcal{L}(L)$. Its language is $\{r_8^{10}()r_7^9()r_4^4()r_3^1()\}$ which shows that the only linear derivation in L is $S() \underset{\ell,L}{\overset{r_3()}{\Rightarrow}} S(\gamma_c)c \underset{\ell,L}{\overset{r_4()}{\Rightarrow}} T(\gamma_c)c \underset{\ell,L}{\overset{r_7()}{\Rightarrow}} cT()c \underset{\ell,L}{\overset{r_8()}{\Rightarrow}} ccc$.

In computing the relations for the initial LIG L, we remark that though $T \overset{\gamma_a}{\underset{+}{\succ}} T$, $T \overset{\gamma_b}{\underset{+}{\succ}} T$, and $T \overset{\gamma_c}{\underset{+}{\succ}} T$, the non-terminals $[T \overset{\gamma_a}{\underset{+}{\succ}} T]$, $[T \overset{\gamma_b}{\underset{+}{\succ}} T]$, and $[T \overset{\gamma_c}{\underset{+}{\succ}} T]$ are not used in P^D. This means that for any LIGed forest L^x, the elements of the form $([T]_p^q, [T]_{p'}^{q'})$ do not need to be computed in the $\overset{\gamma_a}{\underset{+}{\succ}}$, $\overset{\gamma_b}{\underset{+}{\succ}}$, and $\overset{\gamma_c}{\underset{+}{\succ}}$ relations since they will never produce a useful non-terminal. In this example, the subset $\overset{\gamma_c}{\underset{1}{\succ}}$ of $\overset{\gamma_c}{\underset{+}{\succ}}$ is useless.

The next example shows the handling of a cyclic grammar.

6.2 Second Example

The following LIG L, where A is the start symbol:

$$\begin{array}{ll}
r_1() = A(..) \rightarrow A(..\gamma_a) & r_2() = A(..) \rightarrow B(..) \\
r_3() = B(..\gamma_a) \rightarrow B(..) & r_4() = B() \rightarrow a
\end{array}$$

is cyclic (we have $A \overset{+}{\Rightarrow} A$ and $B \overset{+}{\Rightarrow} B$ in its CF-backbone), and the stack schemas in production $r_1()$ indicate that an unbounded number of push γ_a actions can take place, while production $r_3()$ indicates an unbounded number of pops. Its CF-backbone is unbounded ambiguous though its language contains the single string a.

The computation of the relations gives:

$$\begin{array}{lll}
\underset{1}{\diamond} & = & \{(A, B)\} \\
\overset{\gamma_a}{\underset{1}{\prec}} & = & \{(A, A)\} \\
\overset{\gamma_a}{\underset{1}{\succ}} & = & \{(B, B)\} \\
\underset{+}{\diamond} & = & \{(A, B)\} \\
\approx & = & \{(A, B)\} \\
\overset{\gamma_a}{\underset{+}{\succ}} & = & \{(A, B), (B, B)\}
\end{array}$$

The start symbol of the LDG associated with L is $[A]$ and its productions set P^D is:

$$\begin{array}{llll}
[A] & \rightarrow & r_4()[A \underset{+}{\diamond} B] & (2) \\
[A \underset{+}{\diamond} B] & \rightarrow & r_2() & (3) \\
[A \underset{+}{\diamond} B] & \rightarrow & [A \approx B] & (4) \\
[A \approx B] & \rightarrow & [A \overset{\gamma_a}{\underset{+}{\succ}} B]r_1() & (7) \\
[A \overset{\gamma_a}{\underset{+}{\succ}} B] & \rightarrow & r_3()[A \underset{+}{\diamond} B] & (9)
\end{array}$$

We can easily checked that this grammar is reduced.

We want to parse the input string $x = a$ (i.e. find all the linear $S()/a$-derivations).

Its LIGed forest, whose start symbol is $[A]_0^1$ is:

$$\begin{array}{lcllcl}
r_1^1() & = & [A]_0^1(..) & \rightarrow & [A]_0^1(..\gamma_a) \\
r_2^2() & = & [A]_0^1(..) & \rightarrow & [B]_0^1(..) \\
r_3^3() & = & [B]_0^1(..\gamma_a) & \rightarrow & [B]_0^1(..) \\
r_4^4() & = & [B]_0^1() & \rightarrow & a
\end{array}$$

For this LIGed forest L^x, the relations are:

$$\begin{array}{rcl}
\underset{1}{\diamond} & = & \{([A]_0^1, [B]_0^1)\} \\
\underset{1}{\overset{\gamma_a}{\prec}} & = & \{([A]_0^1, [A]_0^1)\} \\
\underset{1}{\overset{\gamma_a}{\succ}} & = & \{([B]_0^1, [B]_0^1)\} \\
\approx & = & \{([A]_0^1, [B]_0^1)\} \\
\underset{+}{\diamond} & = & \{([A]_0^1, [B]_0^1)\} \\
\underset{+}{\overset{\gamma_a}{\succ}} & = & \{([A]_0^1, [B]_0^1), ([B]_0^1, [B]_0^1)\}
\end{array}$$

The start symbol of the LDG associated with L^x is $[[A]_0^1]$. If we assume that an A-production is generated iff it is an $[[A]_0^1]$-production or A occurs in an already generated production, its production set is:

$$\begin{array}{llll}
[[A]_0^1] & \rightarrow & r_4^4()[[A]_0^1 \underset{+}{\diamond} [B]_0^1] & (2) \\
[[A]_0^1 \underset{+}{\diamond} [B]_0^1] & \rightarrow & r_2^2() & (3) \\
[[A]_0^1 \underset{+}{\diamond} [B]_0^1] & \rightarrow & [[A]_0^1 \approx [B]_0^1] & (4) \\
[[A]_0^1 \approx [B]_0^1] & \rightarrow & [[A]_0^1 \underset{+}{\overset{\gamma_a}{\succ}} [B]_0^1] r_1^1() & (7) \\
[[A]_0^1 \underset{+}{\overset{\gamma_a}{\succ}} [B]_0^1] & \rightarrow & r_3^3()[[A]_0^1 \underset{+}{\diamond} [B]_0^1] & (9)
\end{array}$$

This CFG is reduced. Since its production set is non empty, we have $a \in \mathcal{L}(L)$. Its language is $\{r_4^4()\{r_3^3()\}^k r_2^2()\{r_1^1()\}^k \mid 0 \leq k\}$ which shows that the only valid linear derivations w.r.t. L must contain an identical number k of productions which push γ_a (i.e. the production $r_1()$) and productions which pop γ_a (i.e. the production $r_3()$).

As in the previous example, we can see that the element $[B]_0^1 \underset{+}{\overset{\gamma_a}{\succ}} [B]_0^1$ is useless.

7 Conclusion

We have shown that the parses of a LIG can be represented by a non ambiguous CFG. This representation captures the fact that the values of a stack of symbols is well parenthesized. When a symbol γ is pushed on a stack at a given index at some place, this very symbol must be popped some place else, and we know that such (recursive) pairing is the essence of context-freeness.

In this approach, the number of productions and the construction time of this CFG is at worst $\mathcal{O}(n^6)$, though much better results occur in practical situations. Moreover, static computations on the initial LIG may decrease this practical complexity in avoiding useless computations. Each sentence in this CFG is a derivation of the given input string by the LIG, and is extracted in linear time.

References

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.

Pierre Boullier. 1996. Another Facet of LIG Parsing (extended version). In *Research Report No 2858* at http://www.inria.fr/RRRT/RR-2858.html, INRIA-Rocquencourt, France, Apr. 1996, 22 pages.

Bernard Lang. 1991. Towards a uniform formal framework for parsing. In *Current Issues in Parsing Technology*, edited by M. Tomita, Kluwer Academic Publishers, pages 153–171.

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

Yves Schabes, Stuart M. Shieber. 1994. An Alternative Conception of Tree-Adjoining Derivation. In *ACL Computational Linguistics*, Vol. 20, No. 1, pages 91–124.

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

K. Vijay-Shanker, David J. Weir. 1993. The Used of Shared Forests in Tree Adjoining Grammar Parsing. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL'93)*, Utrecht, The Netherlands, pages 384–393.

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

K. Vijay-Shanker, David J. Weir, Owen Rambow. 1995. Parsing D-Tree Grammars. In *Proceedings of the fourth international workshop on parsing technologies (IWPT'95)*, Prague and Karlovy Vary, Czech Republic, pages 252–259.