Another Facet of LIG Parsing (extended version)

Pierre Boullier

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Another Facet of LIG Parsing (extended version)

Pierre Boullier

N° 2858

Avril 1996

THÈME 3

ISSN 0249-6399

INRIA
ROCQUENCOURT

Another Facet of LIG Parsing (extended version)

Pierre Boullier*

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

Rapport de recherche n°2858 — Avril 1996 — 22 pages

Abstract: In this paper 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.

Key-words: mildly context-sensitive parsing, ambiguity, parse tree, shared parse forest.

(Résumé : tsvp)

This report is an extended version of the ACL'96 paper (Boullier, 1996).

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

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

Une autre facette de l'analyse syntaxique des LIG (version étendue)

Résumé : Nous présentons dans ce rapport un nouvel algorithme d'analyse syntaxique pour les grammaires indexées linéaires (LIG) dans le même esprit que celui décrit dans (Vijay-Shanker and Weir, 1993) pour les grammaires d'arbres adjoints. Étant donné une LIG L et une chaîne d'entrée x, nous construisons une grammaire non contextuelle et non ambigüe dont les phrases sont les séquences de dérivations de L (et uniquement celles-ci), qui conduisent à x. Nous montrons que cette grammaire peut être construite en temps $\mathcal{O}(n^6)$ et que chaque arbre d'analyse peut en être extrait en temps linéaire par rapport à sa taille. Bien que cette borne supérieure en $\mathcal{O}(n^6)$ n'améliore pas les résultats déjà connus, le comportement du cas moyen est bien meilleur. De plus, en pratique, le temps d'analyse peut être amélioré par des calculs effectués statiquement.

Mots-clé : analyse faiblement contextuelle, ambiguïté, arbre d'analyse, forêt d'analyse partagée.

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 by 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 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

The goal of this section is to set up the vocabulary and to define a grammatical vision of derivations and parse trees.

Let $G = (V_N, V_T, P, S)$ be a CFG where:

- V_N is a non-empty finite set of *non-terminal* symbols.
- V_T is a finite set of *terminal* symbols; V_N and V_T are disjoint; $V = V_N \cup V_T$ is the *vocabulary*.
- S is an element of V_N called the *start symbol*.
- $P \subseteq V_N \times V^*$ is a finite set of productions. Each production is denoted by $A \to \sigma$; such a production is called an A-production.

We adopt the convention that A, B, C denote non-terminals, a, b, c denote terminals, w, x denote elements of V_T^*, X denotes elements of V, β, σ denote elements of V^* and r refer to productions.

On V^* we define a binary relation named *derives* and denoted by $\underset{G}{\Rightarrow}$ (or simply $\Rightarrow$ when G is understood) as the set $\{(\sigma B \sigma', \sigma\beta\sigma') \mid B \to \beta \in P\}$.

Let $\sigma_1, \ldots, \sigma_i, \sigma_{i+1}, \ldots, \sigma_l$ be strings in V^* such that $\forall i, 1 \leq i < l, \sigma_i \Rightarrow \sigma_{i+1}$ then the sequence of strings $(\sigma_1, \ldots, \sigma_i, \sigma_{i+1}, \ldots, \sigma_l)$ is called a *derivation*.

A *σ-derivation* is a derivation whose first element is σ. A σ-derivation whose last element in the sequence is β is called a *σ/β-derivation*. The elements of a σ-derivation are called *σ-phrases*. A σ-phrase in V_T^* is a *σ-sentence*. On the other hand an S-phrase is a *sentential form* and an S-sentence is a *sentence*.

The language defined by G is the set of its sentences:

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

In a *rightmost (leftmost)* derivation, at each step the rightmost (leftmost) non-terminal say B is replaced by the RHS of a B-production. A rightmost (leftmost) derivation can equivalently be specified by its first and last string and by the sequence of productions used at each step. For example if $S \overset{r_1}{\underset{G}{\Rightarrow}} \ldots \overset{r_n}{\underset{G}{\Rightarrow}} x$ is a rightmost S/x-derivation in which at each

step the relation symbol is overlined by the production used, we will also say that $r_1 \dots r_n$ is a rightmost S/x-derivation.

A symbol X is *accessible* (from the start symbol) if X appears in some sentential form, *productive* if $X \stackrel{*}{\Rightarrow} x$ and *useful* if it is both accessible and productive. A production is useful if all its symbols are useful. A grammar is *reduced* when all its productions are useful. We will see how these classical definitions extend to LIGs in section 3.

For a CFG G, the set of its (say) rightmost derivations can itself be defined by a grammar called rightmost derivation grammar.

Definition 1 *Let $G = (V_N, V_T, P, S)$ be a CFG, its rightmost derivation grammar is the CFG D_G (or D when G is understood) $D = (V_N^D, V_T^D, P^D, S^D)$ where*

- $V_T^D = P$
- $V_N^D = V_N$.
- $S^D = S$
- $P^D = \{A_0 \rightarrow A_1 \dots A_q r \mid r = A_0 \rightarrow w_0 A_1 w_1 \dots w_{q-1} A_q w_q \in P \wedge w_i \in V_T^* \wedge A_j \in V_N\}$

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

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

This shows that the rightmost derivation language of a CFG is also CF. We will see in section 4 that a similar result holds for LIGs. Note that this grammar is reduced iff G is reduced.

A *shared parse forest* is the intersection of a CFG and a non-deterministic finite state automaton (FSA). From the language theory we know that this intersection is itself a CFG. In (Lang, 1994) B. Lang gives this definition to denote the set of parse trees since any input string x can be modeled by an FSA[1]. Moreover, the consideration of an FSA as input rather than a simple linear sequence can be interesting in natural language processing since regular languages can express different phenomena like ill-formed or ambiguous inputs. Below, we formally define a shared parse forest when the input is an FSA.

Definition 2 *Let $G = (V_N, V_T, P, S)$ be a CFG, $G' = (V_N \cup \{S'\}, V_T, P \cup \{S' \rightarrow S\}, S')$ its augmented grammar, and $\mathcal{M} = (Q, \Sigma, q_0, \delta, F)$ be an FSA. The shared parse forest for $\mathcal{M}$ (w.r.t. G) is the CFG, $G^{\mathcal{M}} = (V_N^{\mathcal{M}}, V_T^{\mathcal{M}}, P^{\mathcal{M}}, [S'])$ where:*

- $V_N^{\mathcal{M}} = \{[S']\} \bigcup \{[X]_i^j \mid X \in V_N \wedge i, j \in Q\}$.
- $V_T^{\mathcal{M}} = \{a \mid a \in V_T \cap \Sigma\}$.

[1] if $x = a_1 \dots a_n$, the states of the FSA can be the integers $0 \dots n$, 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 $i \in \delta(i, \varepsilon)$.

- $P^{\mathcal{M}} = \{[S'] \rightarrow [S]_{q_0}^{q_f} \mid q_f \in F\} \bigcup \{[X]_i^j \rightarrow \varepsilon \mid X \rightarrow \varepsilon \in P \wedge j \in \delta(i,\varepsilon)\} \bigcup$
  $\{[X_0]_{i_0}^{i_p} \rightarrow Y_1 \dots Y_k \dots Y_p \mid X_0 \rightarrow X_1 \dots X_k \dots X_p \in P \wedge i_0 \in Q \wedge \forall k, 1 \leq k \leq p \exists i_k \in Q$
  s. t. $(Y_k = [X_k]_{i_{k-1}}^{i_k} \wedge X_k \in V_N \vee Y_k = X_k \wedge X_k \in V_T \cap \Sigma) \wedge \delta(i_{k-1}, X_k) = i_k\}$.

This grammar described the intersection of G and $\mathcal{M}$:

$$\mathcal{L}(G^{\mathcal{M}}) = \mathcal{L}(G) \bigcap \mathcal{L}(\mathcal{M})$$

Each production $r_p^q \in P^{\mathcal{M}}$, is denoted by a double index (p,q) where the lower one p is such that r_p is the associated production in $P \cup \{S' \rightarrow S\}$.

We call *canonical* shared parse forest the reduced CFG whose production set $P_c^{\mathcal{M}}$ is the subset of $P^{\mathcal{M}}$ where all productions containing useless symbols have been eliminated. In fact, any CFG $G_i^{\mathcal{M}} = (V_N^{\mathcal{M}}, V_T^{\mathcal{M}}, P_i^{\mathcal{M}}, S^{\mathcal{M}})$ s.t. $P_c^{\mathcal{M}} \subseteq P_i^{\mathcal{M}} \subseteq P^{\mathcal{M}}$ is called a shared parse forest.

It should be noticed that this definition is completely independent of the way (i.e left-to-right, top-down, bottom-up, ...) the forest is built.

A "blind" implementation, simply based upon combinatorial considerations, which does not rely upon dependencies from one production to the other, and which generates $P^{\mathcal{M}}$, leads to a parser which may be qualified of global. The forest is built without any left-to-right, top-down or bottom-up bias. In such a case inaccessible and non-productive symbols can be produced.

The bottom-up version, where a production is generated only when its RHS symbols have already been computed leads to a parser in the CKY style. In this case, in the generated forest, all symbols are productive but some of them can be inaccessible.

The counterpart, where all the productions, having a given non-terminal in LHS, (and all possible combination of symbols in RHS,) are produced only when this LHS non-terminal has already been generated (or is the start symbol) leads to a top-down parser. In this case, in the generated forest, all symbols are accessible but some of them can be non-productive.

In all cases, the recognition problem of x in G is to decide whether the language of this parse forest grammar is empty or equivalently whether the start symbol $[S']$ is useful.

If we build the rightmost derivation grammar associated with a shared parse forest, and we remove all its useless symbols by the classical algorithm, we get a reduced grammar D^x. The CF recognition problem is now equivalent to the existence of an $[S']$-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. This is not always the case with non CF formalisms, and 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.

In the sequel we will assume that only linear inputs are processed, but all results stay valid when they are transposed to FSAs.

3 Linear Indexed Grammars (LIGs)

Indexed grammars are syntactic formalisms which are extensions of CFGs in which a stack of symbols is associated with each non-terminal. Besides this CF property, these grammars express the way these stacks evolve. In a LIG, which is a restricted form of indexed grammar, in productions with (at least) a non-terminal in RHS, the same stack is associated with both the LHS symbol and this RHS symbol. Of course, the content of this stack can change between its LHS and its RHS occurrence: some symbols may be pushed or popped. Other RHS non-terminal symbols, if any, are associated with (new) stacks of bounded size.

Following (Vijay-Shanker and Weir, 1994)

Definition 3 *A LIG, L is denoted by (V_N, V_T, V_I, P_L, S) where:*

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

A string of stack symbols is an element of V_I^*. We adopt the convention that α will denote members of V_I^* and γ elements of V_I. In fact, in a LIG production, the structure associated with a non-terminal can be either a stack or a stack schema. A *stack schema* denoted $(..\alpha)$ matches all the stacks whose prefix (bottom) part is left unspecified and whose suffix (top) part is α. In a LIG production, we call *primary constituent* the pair denoted $A(..\alpha)$, consisting of a non-terminal A, and a stack schema $(..\alpha)$ and *secondary constituent* the pair denoted $A(\alpha)$, consisting of a non-terminal A, and a string of stack symbols α.

In the sequel we will only consider a restricted form of LIGs with productions of the form

$$\begin{aligned} P_L \quad = \quad & \{A() \rightarrow w \mid A \in V_N \wedge w \in V_T^* \wedge 0 \leq |w| \leq 2\} \cup \\ & \{A(..\alpha) \rightarrow \Gamma_1 B(..\alpha')\Gamma_2 \mid A, B \in V_N \wedge \alpha\alpha' \in V_I^* \wedge 0 \leq |\alpha\alpha'| \leq 1\} \end{aligned}$$

where $\Gamma_1\Gamma_2 \in V_T \cup \{\varepsilon\} \cup \{C() \mid C \in V_N\}$.

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 its 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[2].

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

$$\begin{array}{lll} \Gamma_1' A(\alpha''\alpha)\Gamma_2' & \overset{A(..\alpha)\rightarrow\Gamma_1 B(..\alpha')\Gamma_2}{\underset{L}{\Rightarrow}} & \Gamma_1'\Gamma_1 B(\alpha''\alpha')\Gamma_2\Gamma_2' \\ \Gamma_1' A()\Gamma_2' & \overset{A()\rightarrow w}{\underset{L}{\Rightarrow}} & \Gamma_1' w\Gamma_2' \end{array}$$

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*. Let $\Gamma_1, \ldots, \Gamma_i, \Gamma_{i+1}, \ldots, \Gamma_l$ be strings in $(V_O \cup V_T)^*$ such that $\forall i, 1 \leq i < l, \exists r_i() \in P_L, \Gamma_i \overset{r_i()}{\underset{L}{\Rightarrow}} \Gamma_{i+1}$ then the sequence of strings $(\Gamma_1, \ldots, \Gamma_i, \Gamma_{i+1}, \ldots, \Gamma_l)$ is called a *derivation*.

The 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 (resp. leftmost) derivations when the rightmost (resp. leftmost) 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. But we first need to define an ordering relation which will be used to choose in a string Γ which object to derive. This relation is defined on *addresses*. An address is an element of $\{1, 2\}^*$ denoted by $i.j.k \ldots$.

The ordering relation $\leq$ is defined by:

$$\begin{array}{rcl} \eta & \leq & \eta.\eta_1 \\ \eta.1.\eta_1 & \leq & \eta.2.\eta_2 \end{array}$$

$\forall \eta, \eta_1, \eta_2 \in \{1, 2\}^*$. We can easily check that $\leq$ defines a total order over $\{1, 2\}^*$ (it is a lexicographic order).

Let $A_0(\alpha_0) \overset{*}{\underset{L}{\Rightarrow}} \Gamma_1 A_1(\alpha_1)x_1 \overset{r_i()}{\underset{L}{\Rightarrow}} \Gamma_1\Gamma_2 x_1$ be a rightmost $A_0(\alpha_0)$-derivation, we will associate an address with any object occurrence in the following way:

- The address of the initial object occurrence $A_0(\alpha_0)$ is ε.

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

- Assume that each object occurrence in $\Gamma_1 A_1(\alpha_1)x_1$ has an address and the address of $A_1(\alpha_1)$ is η, we examine the addresses of $\Gamma_1\Gamma_2 x_1$. The addresses associated with Γ_1 are identical in left and right hand sides, and the addresses associated with (the object occurrences of) Γ_2 depend upon the kind of the production $r_i()$ used. When $r_i() = A_1() \rightarrow w$ is used ($\alpha_1 = \varepsilon$ and $\Gamma_2 = w$) this addressing is pointless. In the other case the addresses of the object occurrences in Γ_2 are $\eta.2$ for the distinguished child and $\eta.1$ for the secondary object, if any.

This means that the address of a distinguished child is always greater than the address of its secondary object companion (if any), whether this object lays to its left or to its right. We agree that at each derivation step the address η of the object to be derived can be associated with the production $r_i()$ (i.e. $\Gamma_1 A_1(\alpha_1)x_1 \underset{L}{\overset{r_i(),\eta}{\Rightarrow}} \Gamma_1\Gamma_2 x_1$). Note that in a derivation such addresses are all differents.

The sequence of pairs $(r_1(), \eta_1 = \varepsilon), \ldots, (r_i(), \eta_i), \ldots, (r_j(), \eta_j), \ldots, (r_n(), \eta_n)$ associated with any rightmost derivation, can be reordered in such a way that any two consecutive pair $(r_k(), \eta_k), (r_l(), \eta_l)$ verify $\eta_k \leq \eta_l$.

Let $(r'_1() = r_1(), \eta'_1 = \varepsilon), \ldots, (r'_i(), \eta'_i), \ldots, (r'_j(), \eta'_j), \ldots, (r'_n(), \eta'_n)$ be this reordered sequence. The derivation $A_0(\alpha_0) \underset{L}{\overset{r'_1()}{\Rightarrow}} \ldots \underset{L}{\overset{r'_i()}{\Rightarrow}} \ldots \underset{L}{\overset{r'_j()}{\Rightarrow}} \ldots \underset{L}{\overset{r'_n()}{\Rightarrow}} \ldots$ implied by this reordered sequence in which at each step the object to be derived is the one with the smallest address is called *linear derivation*[3] and is denoted by $\underset{\ell,L}{\Rightarrow}$. Moreover, the way it is defined shows that for each rightmost derivation there exists a unique linear derivation. Since for each word x in $\mathcal{L}(L)$ there is at least a rightmost $S()/x$-derivation, there is also a linear $S()/x$-derivation.

For a derivation the reflexive transitive closure of the distinguished child relation is the *distinguished descendant* relation. Since the address of a distinguished child is always greater than the address of its secondary object companion, in a linear derivation, it will be derived after it (and after the descendants of that companion). Therefore, if we have $A(\alpha) \underset{\ell,L}{\overset{+}{\Rightarrow}} x_1 A'(\alpha')x_3$, $A'(\alpha')$ is a distinguished descendant of $A(\alpha)$.

The sequence of objects $A_1(\alpha_1)\ldots A_i(\alpha_i)A_{i+1}(\alpha_{i+1})\ldots A_p(\alpha_p)$ is called a *spine* if, there is a derivation in which each object $A_{i+1}(\alpha_{i+1})$ is the distinguished child of $A_i(\alpha_i)$ (and the distinguished descendant of $A_j(\alpha_j), 1 \leq j \leq i$). Equivalently, if η and $\eta.2^k$ are two addresses in a derivation, the sequence of objects whose addresses are $\eta, \eta.2, \ldots, \eta.2^h, \ldots, \eta.2^k$ where $0 \leq h \leq k$ is a spine. This means that for a derivation and an object $A(\alpha)$ at a given address, we can talk, when no confusion can arise, of the spine of $A(\alpha)$.

Definition 4 *For a given LIG L, a production $r()$ is useful iff it occurs in some $S()/x$-derivation*

$$S() \underset{L}{\overset{*}{\Rightarrow}} \Gamma_1 \underset{L}{\overset{r()}{\Rightarrow}} \Gamma_2 \underset{L}{\overset{*}{\Rightarrow}} x$$

[3]linear reminds us that we are in a LIG and relies upon a linear (total) order over object occurrences in a derivation.

Definition 5 *A LIG is reduced iff all its productions are useful.*

We note that if a LIG L is reduced and if its production set P_L is non-empty, we have $\mathcal{L}(L) \neq \emptyset$.

However, contrary to the CF case, when L is reduced:

- an accessible object $A(\alpha)$ (i.e. $S() \overset{*}{\underset{L}{\Rightarrow}} \Gamma_1 A(\alpha)\Gamma_2$) can be non productive (i.e. $\nexists A(\alpha) \overset{*}{\underset{L}{\Rightarrow}} x$);
- a productive object can be inaccessible;
- a new production built with constituents occurring in useful productions is not necessarily useful.

4 Linear Derivation Grammar

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

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

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 called *linear derivation grammar* (LDG). Such a LDG is of importance since it defines all valid derivations w.r.t. L (and only these ones). We will see how it can be used to define a parser (and a recognizer) for LIGs.

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)

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

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, $\underset{1}{\overset{\gamma}{\prec}}$ the pushing of γ, and $\underset{1}{\overset{\gamma}{\succ}}$ 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.

If the relations $\overset{\gamma}{\underset{+}{\succ}}$ and $\approx$ are defined as

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

we can easily 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[4] 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. Of course, the maximum size of these relations is $\mathcal{O}(|V_N|^2)$.

Definition 6 *Let $L = (V_N, V_T, V_I, P_L, S)$ be a LIG, its linear derivation grammar (LDG) is the CFG D_L (or D when L is understood) $D = (V_N^D, V_T^D, P^D, S^D)$ where*

- *The set of non-terminal symbols is $V_N^D = \{[A] \mid A \in V_N\} \cup \{[A\rho B] \mid A, B \in V_N \wedge \rho \in \mathcal{R}\}$, where $\mathcal{R}$ is the set of relations $\{\overset{\gamma}{\underset{1}{\prec}}, \underset{1}{\diamond}, \overset{\gamma}{\underset{1}{\succ}}, \underset{+}{\diamond}, \approx, \overset{\gamma}{\underset{+}{\succ}}\}$. In fact we will only use valid non-terminals $[A\rho B]$ for which the relation ρ holds between A and B.*
- *The terminal symbols of D are the productions of L: $V_T^D = P_L$*
- *$S^D = [S]$*
- *Below, $[\Gamma_1\Gamma_2]$ denotes either the non-terminal symbol $[X]$ when $\Gamma_1\Gamma_2 = X()$ or the empty string ε when $\Gamma_1\Gamma_2 \in V_T^*$.*

$$\begin{aligned} P^D = \; & \{[A] \to r() \mid r() = A() \to w \in P_L\} \cup & (1) \\ & \{[A] \to r()[A \underset{+}{\diamond} B] \mid r() = B() \to w \in P_L\} \cup & (2) \\ & \{[A \underset{+}{\diamond} C] \to [\Gamma_1\Gamma_2]r() \mid r() = A(..) \to \Gamma_1 C(..)\Gamma_2 \in P_L\} \cup & (3) \\ & \{[A \underset{+}{\diamond} C] \to [A \approx C]\} \cup & (4) \end{aligned}$$

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

$$\{[A \underset{+}{\diamond} C] \rightarrow [B \underset{+}{\diamond} C][\Gamma_1\Gamma_2]r() \mid r() = A(..) \rightarrow \Gamma_1 B(..)\Gamma_2 \in P_L\} \cup \quad (5)$$

$$\{[A \underset{+}{\diamond} C] \rightarrow [B \underset{+}{\diamond} C][A \approx B]\} \cup \quad (6)$$

$$\{[A \approx C] \rightarrow [B \overset{\gamma}{\underset{+}{\succ}} C][\Gamma_1\Gamma_2]r() \mid r() = A(..) \rightarrow \Gamma_1 B(..\gamma)\Gamma_2 \in P_L\} \cup \quad (7)$$

$$\{[A \overset{\gamma}{\underset{+}{\succ}} C] \rightarrow [\Gamma_1\Gamma_2]r() \mid r() = A(..\gamma) \rightarrow \Gamma_1 C(..)\Gamma_2 \in P_L\} \cup \quad (8)$$

$$\{[A \overset{\gamma}{\underset{+}{\succ}} C] \rightarrow [\Gamma_1\Gamma_2]r()[A \underset{+}{\diamond} B] \mid r() = B(..\gamma) \rightarrow \Gamma_1 C(..)\Gamma_2 \in P_L\} \quad (9)$$

The productions in P^D define all the ways linear derivations can be composed from linear sub-derivations. 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.

Theorem 1 *Let $L = (V_N, V_T, V_I, P_L, S)$ be a LIG and $D = (V_N^D, V_T^D, P^D, S^D)$ its LDG, we have*

$$\mathcal{L}(D) \quad = \quad \{r_1() \ldots r_n() \mid S() \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \ldots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} x \wedge x \in \mathcal{L}(L)\}$$

Proof: In this proof we will only consider two types of linear derivations. The set $\mathcal{B}$ of linear *balanced* derivations

$$\mathcal{B} \quad = \quad \{A() \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \ldots \underset{\ell,L}{\overset{r_i()}{\Rightarrow}} \ldots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} x_1 B() x_3 \mid B() \text{ is a distinguished descendant of } A()\}$$

and the set $\mathcal{C}$ of linear *closed* derivations

$$\mathcal{C} \quad = \quad \{A() \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \ldots \underset{\ell,L}{\overset{r_i()}{\Rightarrow}} \ldots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} x\}$$

Note that any linear closed derivation is the composition of a linear balanced $A()/x_1B()x_3$-derivation and a trivial linear closed derivation $B() \underset{\ell,L}{\overset{B()\rightarrow x_2}{\Rightarrow}} x_2$. Let $A() \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \ldots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} x_1 B() x_3 \in \mathcal{B}$, and $C() \underset{\ell,L}{\overset{*}{\Rightarrow}} y_1 A(\alpha) y_3$ and $B(\alpha) \underset{\ell,L}{\overset{*}{\Rightarrow}} z_1 D() z_3$ be two linear derivations, then $C() \underset{\ell,L}{\overset{*}{\Rightarrow}} y_1 A(\alpha) y_3 \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \ldots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} y_1 x_1 B(\alpha) x_3 y_3 \underset{\ell,L}{\overset{*}{\Rightarrow}} y_1 x_1 z_1 D() z_3 x_3 y_3$ is a linear balanced derivation.

Recall that the language of a non-terminal A is defined by $\mathcal{L}(A) = \{x \mid A \overset{+}{\Rightarrow} x\}$. Our theorem results from the proof that the languages of non-terminal symbols in V_N^D can

be defined in terms of linear balanced or closed derivations in L in the following way

$$\begin{aligned}\mathcal{L}([A_{n+1} \underset{+}{\diamond\!\!-} A_1]) &= \{r_1()\dots r_n() \mid A_{n+1}() \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \dots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} x_1 A_1() x_3 \in \mathcal{B}\} \\ \mathcal{L}([A])) &= \{r_1()\dots r_n() \mid A() \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \dots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} x \in \mathcal{C}\}\end{aligned}$$

The proof that if $r_1()\dots r_n() \in \mathcal{L}([A \underset{+}{\diamond\!\!-} B])$, the derivation $A() \underset{L}{\overset{r_n()}{\Rightarrow}} \dots \underset{L}{\overset{r_1()}{\Rightarrow}} x_1 B() x_3$ is a linear balanced derivation or if $r_1()\dots r_n() \in \mathcal{L}([A])$ the derivation $A() \underset{L}{\overset{r_n()}{\Rightarrow}} \dots \underset{L}{\overset{r_1()}{\Rightarrow}} x$ is a linear closed derivation is performed by induction on the length of the strings in $\mathcal{L}([A \underset{+}{\diamond\!\!-} B])$ or $\mathcal{L}([A])$.

The proof that every derivation $A() \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \dots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} x_1 B() x_3 \in \mathcal{B}$ (resp. $A() \underset{\ell,L}{\overset{r_n()}{\Rightarrow}} \dots \underset{\ell,L}{\overset{r_1()}{\Rightarrow}} x \in \mathcal{C}$) is such that $r_1()\dots r_n()$ is a string in $\mathcal{L}([A \underset{+}{\diamond\!\!-} B])$ (resp. $\mathcal{L}([A])$) is performed by induction on the length of these derivations.

Finally the theorem results from the fact that $\mathcal{L}(D) = \mathcal{L}([S])$.

$\diamond$

Moreover, we can show that LDGs are non ambiguous. This non ambiguity results from the following property. Let $\sigma \in P_L^*$ and $X \in V_N^D$ s.t. $\sigma \in \mathcal{L}(X)$: there is a unique production $X \to YZ \in P^D$ such that $X \underset{D}{\overset{X \to YZ}{\Rightarrow}} YZ \underset{D}{\overset{*}{\Rightarrow}} Y\sigma_2 \underset{D}{\overset{*}{\Rightarrow}} \sigma_1\sigma_2 = \sigma$. In fact we can prove a much stronger result: LDGs are SLR(1). The proof of that theorem strongly relies upon the property that the sets FOLLOW($[A]$) and FIRST($[B \underset{+}{\diamond\!\!-} C]$) are disjoint for all $[A]$ and $[B \underset{+}{\diamond\!\!-} C]$ in V_N^D.

If D is generated top-down: we first produce the $[S]$-productions and then an $[X]$-production is produced iff the non-terminal $[X]$ occurs in the RHS of some already produced production, we are sure that all symbols in D are accessible. However, some symbols can be non productive. This is due to the fact that when a non-terminal, say $[A \underset{+}{\diamond\!\!-} C]$ is generated in some RHS, though we are sure that there is at least one spine between A and C, the relation $\underset{+}{\diamond\!\!-}$ does not guaranty the existence of linear closed (sub-)derivations starting on secondary objects generated during the walk along the main spine.

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. This shows that the LIG $L' = (V_N, V_T, V_I, V_T^{D'}, S)$ is equivalent to L and is reduced. By the way, the construction of D' solve the emptiness problem for LIGs: a LIG specify the empty set iff the set $V_T^{D'}$ is empty[5].

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

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) Boullier gives 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 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 (see (Vijay-Shanker, 1987)) that derivations in TAGs are context-free 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 7 *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[6]: 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

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

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[7] 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 $\mathcal{L}(D_{L^x})$[8]. 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 6). The first form is production (6) ($[A \underset{+}{\diamond\!\!-} C] \rightarrow [B \underset{+}{\diamond\!\!-} C][A \approx B]$), 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 $X_i \overset{*}{\underset{D}{\Rightarrow}} \sigma_i$,

[7] See (Kasami, 1965) and (Younger, 1967) for the Cocke-Kasami-Younger method, (Earley, 1968) for the Earley algorithm, and (Lang, 1974), (Tomita, 1987) and (Rekers, 1992) for generalized LR parsing.

[8] 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 $r_p()$).

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 6. 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 $\underset{+}{\lessdot}$, $\overset{\gamma}{\underset{+}{\gtrdot}}$ 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 $[A \underset{+}{\lessdot} B]$ is useless in the LDG D_L associated with the initial LIG L, we know that any non-terminal s.t. $[[A]_i^j \underset{+}{\lessdot} [B]_k^l]$ is also useless in D_{L^x}. Therefore, the static computation of a reduced LDG for the initial LIG L (and the corresponding $\underset{+}{\lessdot}$, $\overset{\gamma}{\underset{+}{\gtrdot}}$ and $\approx$ relations) can be used to direct the parsing process and decrease the parsing time (see section 6). Up to our knowledge, it is the first time that a static computation on LIGs can be used to possibly decrease the parsing time.

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:

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

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

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

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 $T \to c$, 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:

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

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

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

The numbers (i) refer to definition 6. 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:

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

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:

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

For this LIGed forest the relations are:

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

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:

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

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 $S() \overset{r_3()}{\underset{\ell,L}{\Rightarrow}} S(\gamma_c)c \overset{r_4()}{\underset{\ell,L}{\Rightarrow}} T(\gamma_c)c \overset{r_7()}{\underset{\ell,L}{\Rightarrow}} cT()c \overset{r_8()}{\underset{\ell,L}{\Rightarrow}} ccc$.

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:

$$r_1() = A(..) \rightarrow A(..\gamma_a) \quad r_2() = A(..) \rightarrow B(..) \quad r_3() = B(..\gamma_a) \rightarrow B(..) \quad r_4() = B() \rightarrow a$$

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:

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

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

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

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:

$$r_1^1() = [A]_0^1(..) \rightarrow [A]_0^1(..\gamma_a) \quad r_2^2() = [A]_0^1(..) \rightarrow [B]_0^1(..)$$
$$r_3^3() = [B]_0^1(..\gamma_a) \rightarrow [B]_0^1(..) \quad r_4^4() = [B]_0^1() \rightarrow a$$

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

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

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:

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

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 $[B]_0^1 \overset{\gamma_a}{\underset{+}{\succ}} [B]_0^1$ 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. This means that 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 a 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 are expected 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 (linear) 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. To appear in *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, Santa Cruz, CA.

Jay C. Earley. 1968. An efficient context-free parsing algorithm. *PhD thesis*, Carnegie-Mellon University, Pittsburgh, PA.

T. Kasami. 1965. An efficient recognition and syntax algorithm for context-free languages. In *Technical Report AF-CRL-65-758*, Air Force Cambridge Research Laboratory, Bedford, MA.

Bernard Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In *Automata, Languages and Programming, 2nd Colloquium*, Lectures Notes in Computer Science, Springer-Verlag, Vol. 14, pages 255–269.

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.

Jan Rekers. 1992. Parser generation for interactive environments. *PhD thesis*, University of Amsterdam.

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

M. Tomita. 1987. An efficient augmented context-free parsing algorithm. In *Computational Linguistics*, Vol. 13, pages 31–46.

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.

D. H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. In *Information and Control*, Vol. 10, No. 2, pages 189–208.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399