Yet Another $\mathcal{O}(n^6)$ Recognition Algorithm for Mildly Context-Sensitive Languages

Pierre Boullier

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Yet Another $\mathcal{O}(n^6)$ Recognition Algorithm for Mildly Context-Sensitive Languages

Pierre Boullier

N° 2730

Novembre 1995

PROGRAMME 3

ISSN 0249-6399

INRIA
ROCQUENCOURT

Yet Another $\mathcal{O}(n^6)$ Recognition Algorithm for Mildly Context-Sensitive Languages

Pierre Boullier*

Programme 3 — Intelligence artificielle, systèmes cognitifs et interaction homme-machine
Projet Atoll

Rapport de recherche n ° 2730 — Novembre 1995 — 22 pages

Abstract: Vijay-Shanker and Weir have shown in [19] that Tree Adjoining Grammars and Combinatory Categorial Grammars can be transformed into equivalent Linear Indexed Grammars (LIGs) which can be recognized in $\mathcal{O}(n^6)$ time using a Cocke-Kasami-Younger style algorithm. This paper exhibits another recognition algorithm for LIGs, with the same upper-bound complexity, but whose average case behaves much better. This algorithm works in two steps: first a general context-free parsing algorithm (using the underlying context-free grammar) builds a shared parse forest, and second, the LIG properties are checked on this forest. This check is based upon the composition of simple relations and does not require any computation of symbol stacks.

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

(Résumé : tsvp)

This research report is an extended version of [3].It benefits from discussions, especially with David Weir, and amends the original algorithm.

*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

Un autre algorithme de reconnaissance en $\mathcal{O}(n^6)$ pour les langages modérément contextuels

Résumé : Vijay-Shanker et Weir ont montré dans [19] que les Grammaires d'Arbres Adjoints et les Grammaires Catégorielles Combinatoires peuvent être transformées en Grammaires Indexées Linéaires (LIG) équivalentes qui peuvent être reconnues en temps $\mathcal{O}(n^6)$ en utilisant un algorithme à la Cocke-Kasami-Younger. Cet article propose un autre algorithme de reconnaissance pour les LIG, ayant la même complexité maximale, mais dont le comportement moyen est bien meilleur. Cet algorithme travaille en deux étapes: tout d'abord un analyseur non contextuel général (qui utilise la grammaire non contextuelle sous-jacente) construit une forêt d'analyse partagée sur laquelle, dans une deuxième phase, les propriétes des LIG sont vérifiées. Cette vérification repose sur la composition de relations simples et ne nécessite aucun calcul de piles de symboles.

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

1 Introduction

It is well known that natural language processing cannot be described by purely context-free grammars (CFGs). On the other hand, general context-sensitive formalisms are powerful enough but cannot be parsed in reasonable time. Therefore, various intermediate frameworks have been investigated, the trade-off being between expressiveness power and computational tractability. One of these formalism classes is the so-called mildly context-sensitive languages which can be described by several equivalent grammar types. Among these types, Tree Adjoining Grammars (TAGs) are attractive because they can express some natural language phenomena (see Abeillé and Schabes [1]) and many systems are based upon this framework (see for example [11] and [7]). Formal properties of TAGs have been studied (see Vijay-Shanker and Joshi [17], and Vijay-Shanker [16]) and a recognizer for TAGs (see [19]), based upon a Cocke-Kasami-Younger method ([5] and [20]), works in $\mathcal{O}(n^6)$ worst time. Unfortunately, with such algorithms, this complexity is always reached. More practical methods, which are usually based upon the Earley parsing algorithm [4], have also been investigated (see for example Schabes [14] and Poller [12]). Though the $\mathcal{O}(n^6)$ worst-case time is not improved, for some inputs, the actual complexity may be much better. However, the design of a better worst-case recognizer remains an open problem.

In [19], Vijay-Shanker and Weir have shown that mildly context-sensitive grammars have the same formal power and that TAGs and Combinatory Categorial Grammars (CCGs) can be transformed into equivalent Linear Indexed Grammars (LIGs).

An Indexed Grammar [2] is a CFG in which each object is a non-terminal associated with a stack of symbols. The productions of this grammar class define on the one hand a derived relation in the usual sense and, on the other hand, the way symbols are pushed or popped on top of the stacks which are associated with each non-terminal. A restricted form of Indexed Grammar called LIG allows only for the stack associated with the left-hand side (LHS) non-terminal of a given production to be associated with at most one non-terminal in the right-hand side (RHS).

This paper presents a new recognition algorithm for LIGs. It works in two main steps:

1. a CF-parsing algorithm, working on the underlying CF-grammar, builds a shared parse forest;

2. the LIG conditions (see Section 3) are checked on that forest.

Since that second step does not depend on the way the shared parse forest is built, any general CF-parsing algorithm can be used in the first step. As previously mentioned, CKY or Earley parsing algorithms are candidates but others too. In particular, generalized LR parsing methods (see Lang [8], Tomita [15], and Rekers [13]) are good challengers. In fact, the CF-parsing algorithm of our prototype system upon which this paper ideas have been tested, implements a non-deterministic LR (or LC at will) parsing algorithm using a graph-structured stack. Under certain conditions, for a given input string of length n, the CF-parsing takes a time $\mathcal{O}(n^3)$ in the worst case and moreover the output shared parse forest of size $\mathcal{O}(n^3)$ (in the worst case) is such that each elementary tree can be seen as an occurrence (after some non-terminal renaming) of a production in the underlying CFG. Following Lang [9], in Section 2, we stress this analogy by defining a shared parse forest as a CFG.

Obviously, the checking of the LIG properties can be performed on a forest by computing the stacks of symbols along paths, and in associating with each (shared) node a set of such stacks. As in [19], in devising an elaborate sub-stack sharing mechanism, this check could be

performed in $\mathcal{O}(n^6)$ time. In Section 4, we take a different approach: this check is performed without the computation of any stack of symbols (and hence without having to design any sub-stack sharing structure). Given a single tree of the (unfolded) shared parse forest, we identify *spine*s as being paths along which individual stack of symbols are evaluated. The origin of such a spine corresponds to the birth of a stack which evolves according the LIG stack schemas and which finally vanishes at the end of the spine. The checking of LIG conditions relies on the simple observation that, for a given spine, the stack actions must be bracketed. Each time a push or pop occurs at a node, there is a twin node where the opposite action, acting on the same symbol at the same stack level, should take place. In a shared parse forest, different spines may share nodes. In particular, a given couple of twin nodes may be shared among several spines, with the corresponding check being done only once. In Section 5 we show that this check sharing, expressed as relations between twin nodes, results in a worst case $\mathcal{O}(n^6)$-time LIG recognition. Our algorithm is illustrated by an example in Section 6. Sections 7 and 8 show that cyclic grammars are handled too. Since TAGs or CCGs can be transformed into equivalent LIGs [19], this complexity extends over mildly context-sensitive languages.

2 Parse Tree and Shared Parse Forest

The goal of this section is to set up the vocabulary and to define our vision of shared parse forests.

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 $A \rightarrow \sigma$ or by $r_p, 1 \leq p \leq |P|$; 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, and β, σ denote elements of V^*.

On V^* we define $|P|$ disjoint binary relations named *right*[1] *derive by* $B \rightarrow \beta$ and denoted by $\underset{G}{\overset{B\rightarrow\beta}{\Rightarrow}}$ (or simply $\overset{B\rightarrow\beta}{\Rightarrow}$ when G is understood) as the set $\{(\sigma Bx, \sigma\beta x) \mid B \rightarrow \beta \in P\}$.

The relation *derive* denoted by $\Rightarrow$ is defined by:

$$\Rightarrow \;=\; \bigcup_{B\rightarrow\beta\in P} \overset{B\rightarrow\beta}{\Rightarrow}$$

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

[1] In the sequel the qualifier *right* will disappear since only right derivations, right sentential forms, etc ... are introduced.

$\overset{r_q}{\Rightarrow}$ are disjoint when p and q are different, between any two consecutive strings σ_i and σ_{i+1} in a derivation, the relation $\overset{r_p}{\Rightarrow}$ whose (σ_i, σ_{i+1}) is an element is uniquely known.

A *σ-derivation* is a derivation starting with 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 an S/x-derivation, to accurately define the contribution of any symbol occurrence X (its X-sentence) to the sentence x, we will define its range.

Let $d^x = (\sigma_1, \ldots, \sigma_k, \ldots, \sigma_l, \ldots, \sigma_m, \ldots, \sigma_p)$ be an S/x-derivation (i.e. $\sigma_1 = S$, $\sigma_p = x = a_1 \ldots a_n$, and $1 \leq k \leq l \leq m \leq p$). We define the *range* of (this occurrence of) X in σ_k as being the couple (i, j) with $(0 \leq i \leq j \leq n)$ such that:

$$\begin{array}{lcl} \sigma_k & = & \beta X \beta' x_3'' \\ \sigma_l & = & \beta X x_3' x_3'' \\ \sigma_m & = & \beta x_2 x_3' x_3'' \\ \sigma_p & = & x_1 x_2 x_3' x_3'' \end{array}$$

with

$$\begin{array}{lcl} x & = & x_1 x_2 x_3' x_3'' \\ x_1 & = & a_1 \ldots a_i \\ x_2 & = & a_{i+1} \ldots a_j \\ x_3' x_3'' & = & a_{j+1} \ldots a_n \end{array}$$

It follows that in any S/x-derivation, each occurrence of any symbol X can be decorated by its range (i, j) and becomes a *ranged*-symbol denoted by $[X]_i^j$ (or even $[X]$, when the explicit values of the range are not necessary). Moreover, the production used at each derivation step, may be transformed into a *ranged*-production such that each non-terminal is changed by its ranged counterpart.

The set of these ranged-productions defined (our vision of) a parse tree.

Definition 1 *Let $G = (V_N, V_T, P, S)$ be a CFG, $x = a_1 \ldots a_n$ a sentence in $\mathcal{L}(G)$, and d^x an S/x-derivation. We call parse tree (w.r.t. G and d^x) the CFG $G^{d^x} = (V_N^x, V_T^x, P^{d^x}, S^x)$ where:*

- $V_N^x = \{[B]_i^j \mid B \in V_N \wedge 0 \leq i \leq j \leq n\}$.
- $V_T^x = \{a_i \mid a_i \in V_T \wedge x = x_1 a_i x_3\}$.
- $S^x = [S]_0^n$.
- *P^{d^x} is the set of ranged-productions. $[B]_i^j \rightarrow Y_1 \ldots Y_k \ldots Y_p$ is an element of P^{d^x} iff we have the following conditions: there exist two consecutive strings $\sigma_l = \beta B x_3$ and $\sigma_{l+1} = \beta X_1 \ldots X_k \ldots X_p x_3$ in d^x such that (i, j) is the range of B in σ_l and for every $1 \leq k \leq p$ we have if $X_k \in V_T$ then $Y_k = X_k$, otherwise $Y_k = [X_k]_{i_k}^{j_k}$ if the range of X_k in σ_{l+1} is (i_k, j_k).*

The main property of this parse tree grammar is that for each non-terminal symbol $[X]_i^j$ in V_N^x there exists exactly one string w in V_T^{x*} such that $[X]_i^j \overset{+}{\underset{G^{d^x}}{\Rightarrow}} w$ and moreover this string is $a_{i+1} \dots a_j$. Applied to the start symbol $[S]_0^n$, we have $\mathcal{L}(G^{d^x}) = \{x\}$.

If $d^x = (\sigma_1 = S, \dots, \sigma_k, \dots, \sigma_p = x)$ is an S/x-derivation, the sequence of strings $([d^x] = [\sigma_1] = [S]_0^n, \dots, [\sigma_k], \dots, [\sigma_p] = x)$ such that every $[\sigma_k]$ is the string σ_k where each non-terminal occurrence is changed into its ranged counterpart is called ranged-derivation. Of course $[d^x]$ is an $[S]_0^n/x$-derivation in G^{d^x}. If G is not cyclic (i.e. $\nexists A, A \overset{+}{\underset{G}{\Rightarrow}} A$) it is the only one. Conversely, if G is cyclic, G^{d^x} can also be cyclic. If there is a cycle, our definition denotes, by a single parse tree (grammar), the ambiguities showed by the unbounded number of (usual) trees when this cycle is taken 1, 2, ... times.

The whole notion of ambiguity will be captured by the following definition of shared parse forest.

Definition 2 *Let $G = (V_N, V_T, P, S)$ be a CFG, and x a sentence in $\mathcal{L}(G)$. The shared parse forest for x (w.r.t. G) is the CFG, $G^x = (V_N^x, V_T^x, P^x, S^x)$ where:*

- $V_N^x = \{[B]_i^j \mid X \in V_N \wedge 0 \leq i \leq j \leq n\}$.
- $V_T^x = \{a_i \mid a_i \in V_T \wedge x = x_1 a_i x_3\}$.
- $S^x = [S]_0^n$.
- $P^x = \bigcup_{d^x \in D^x} P^{d^x}$ *where D^x is the set of all S/x-derivations, and P^{d^x} is the production set of the parse tree $G^{d^x} = (V_N^x, V_T^x, P^{d^x}, S^x)$ associated with any derivation d^x in D^x.*

Of course we have

$$\mathcal{L}(G^x) = \{x\}$$

Any production $r_p = [B] \rightarrow [X_1] \dots [X_q]$ in P^x is mapped by the unary operator $^-$ to its associated production $\overline{r_p} = B \rightarrow X_1 \dots X_q$ in P.

Note that, for a given sentence x of length n, the number of non-terminal symbols in V_N^x is $\mathcal{O}(n^2)$, and therefore the set P^x is bounded, though D^x can be unbounded when G is cyclic. Without any restriction on G, the size of P^x is $\mathcal{O}(n^{l+1})$ where l is the maximum number of non-terminal symbols in the RHS of any production in P. If G is unambiguous, (or if the parsing of x does not exhibit any ambiguity,) this size is linear in n.

This vision of a set of parse trees as a CFG has several formal and practical advantages (thanks to [10], see also [18]). It exhibits a particular case of a general result: the intersection of CF-languages (defined by G) and regular languages (the input string x) are CF-languages (the resulting shared parse forest G^x). G^x can also be seen as a specialization of G (productions in G^x are productions in G, up to some renaming), which only defines (in all the same possible ways as G) the string x. This CFG allows to define an unbounded number of derivations (when G is cyclic) in a finite way. A *context sharing* occurs when there are several occurrences of the same non-terminal in RHSs, while a *sub-tree sharing* occurs when there are several occurrences of the same non-terminal in LHSs. This sharing may even be considered as optimal if we impose (as done here) that productions (elementary trees) in a parse tree, have the same structure as their corresponding production in G.

Given a sequential input $x = a_1 \dots a_n$, this regular language is defined by a non-deterministic finite state automaton (FSA) $\mathcal{M} = (Q, \Sigma, q_0, \delta, F)$ where:

- $Q = \{i \mid 0 \leq i \leq n\}$
- $\Sigma = \{a_i \mid x = x_1 a_i x_3\}$
- $q_0 = 0$
- $F = \{n\}$
- $\delta(i-1, a_i) = \{i\}$ for each $0 < i \leq n$ and $\delta(i, \varepsilon) = \{i\}$ for each $0 \leq i \leq n$.

The general case, (i.e. context-free languages are closed under intersection with regular languages,) is also of interest in natural language processing since regular languages can express, not only linear sentences, but also different phenomena like ill-formed or ambiguous inputs. Below, we define a shared parse forest when the input is a FSA.

Definition 3 *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 extended grammar, and $\mathcal{M} = (Q, \Sigma, q_0, \delta, F)$ be a 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:*

- $V_N^{\mathcal{M}} = \{[S']\} \bigcup \{[X]_i^j \mid X \in V_N \wedge i, j \in Q\}$.
- $V_T^{\mathcal{M}} = \{a \mid a \in V_T \cap \Sigma\}$.
- $P^{\mathcal{M}} = \{[S'] \rightarrow [S]_{q_0}^{q_f} \mid q_f \in F\} \bigcup \{[X_0]_i^j \rightarrow \varepsilon \mid X_0 \rightarrow \varepsilon \in P \wedge j \in \delta(i, \varepsilon)\} \bigcup \{[X_0]_{r_0}^{r_p} \rightarrow Y_1 \ldots Y_k \ldots Y_p \mid X_0 \rightarrow X_1 \ldots X_k \ldots X_p \in P \wedge r_0 \in Q \wedge \forall k, 1 \leq k \leq p \exists r_k \in Q s.\ t. Y_k = [X_k]_{r_{k-1}}^{r_k} \wedge X_k \in V_N \vee Y_k = X_k \wedge X_k \in V_T \cap \Sigma \wedge \delta(r_{k-1}, X_k) = r_k\}$.

Of course this grammar described the intersection of G and $\mathcal{M}$:

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

and for each production $r_p \in P^{\mathcal{M}}$, there is an associated production $\overline{r_p}$ in $P \cup \{S' \rightarrow S\}$.

We call *canonical* shared parse forest the CFG whose production set $P_c^{\mathcal{M}}$ is the subset of $P^{\mathcal{M}}$ where all productions containing useless symbols[2] 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 unreachable and non-productive[3] symbols can be produced.

The bottom-up version, where a production is produced 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 unreachable.

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

[2] A symbol X is useless if it does not appear in any sentential form.

[3] A symbol X is unreachable (from the start symbol) if there is no $S/\beta X w$-derivation, it is non-productive if there is no X/x-derivation.

already been generated leads to a top-down parser. In this case, in the generated forest, all symbols are reachable but some of them can be non-productive.

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

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)

An indexed grammar is a CFG in which stack of symbols are associated with non-terminals. The derive relation, in addition to its usual meaning, handles these stacks of symbols. LIGs are a restricted form of indexed grammars in which the stack associated with the non-terminal in the LHS of any production is associated with at most one non-terminal in the RHS. Other non-terminals are associated with stacks of bounded size.

In fact, in a production, it is not a stack which is associated with a non-terminal, but rather a stack schema expressing a way to compute a stack. Let V_l denotes a finite set of (stack) symbols, a stack is an element of V_l^*. A stack schema is an element of $V_b \times V_l^*$ where $V_b = \{\varepsilon, ..\}$. We adopt the convention that α will denote members of V_l^*, π elements of V_b, and γ elements of V_l. The stack schema $(..\alpha)$ matches all the stacks whose prefix (bottom) part is left unspecified and whose suffix (top) part is α. A stack may be considered as a stack schema whose first component (the element of V_b) is ε.

A triple (A, ε, α) in $V_N \times V_b \times V_l^*$ is called a *secondary object* and is denoted by $A(\alpha)$ while a triple $(A, .., \alpha)$ is called a *primary object* and is denoted by $A(..\alpha)$. The disjoint sets of primary and secondary objects are respectively denoted by V_O^P and V_O^S. The set of *objects* denoted V_O is $V_O^P \cup V_O^S$. The object $A(\pi\alpha)$, whose non-terminal component part is A, is called an A-object. We use Γ to denote strings in $(V_O \cup V_T)^*$. $A(..\alpha)$ denotes an object whose stack suffix (stack top) is α and with an arbitrary prefix (stack bottom). $A()$ denotes that an empty stack schema is associated with the non-terminal A. $A(\alpha)$ denotes that the stack α is associated with the non-terminal A.

Following [19], we formally defined a LIG as follows:

Definition 4 *A LIG, L is denoted by (V_N, V_T, V_l, 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_l is a finite set of stack symbols.*
- *P_L, the production set, is a finite subset of $V_O \times (V_O \cup V_T)^*$.*
- *$S \in V_N$ is the start symbol.*

Each production in P_L is denoted by $A(\pi\alpha) \rightarrow \Gamma$ or $r_p()$[4] where $1 \leq p \leq |P_L|$.
The general form of a production in a LIG is:

[4]The parentheses reminds us that we are in a LIG!

$$r_p() = A(\pi\alpha) \rightarrow w_1 A_1(\alpha_1) \dots w_{i-1} A_{i-1}(\alpha_{i-1}) w_i A_i(\pi\alpha_i) w_{i+1} A_{i+1}(\alpha_{i+1}) \dots w_p A_p(\alpha_p) w_{p+1}$$

If the LHS object $A(\pi\alpha)$ is secondary (i.e. $\pi = \varepsilon$), we observe that all the objects (if any) in the RHS should also be secondary, while if $A(\pi\alpha)$ is primary (i.e. $\pi = ..$), there must be exactly one primary object (here $A_i(\pi\alpha_i)$) in the RHS.

The above production is called an A-production. If this production is used, for any $\Gamma_1, \Gamma_2 \in ((V_N \times V_l^*) \cup V_T)^*$ and $\alpha' \in V_l^*$, we define the binary relation *derive by* $r_p()$ on LIGs by:

$$\Gamma_1 A(\alpha'\alpha)\Gamma_2 \underset{L}{\overset{r_p()}{\Rightarrow}} \Gamma_1 w_1 A_1(\alpha_1) \dots w_{i-1} A_{i-1}(\alpha_{i-1}) w_i A_i(\alpha'\alpha_i) w_{i+1} A_{i+1}(\alpha_{i+1}) \dots w_p A_p(\alpha_p) w_{p+1} \Gamma_2$$

when $\pi = .. \vee \alpha' = \varepsilon$.

We observe that the stack $\alpha'\alpha$ associated with the non-terminal A in the LHS and the stack $\alpha'\alpha_i$ associated with the non-terminal A_i in the RHS have the same prefix α'.

The language defined by L is the set:

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

We define the *CF-backbone* of a LIG as being its underlying CFG.

Definition 5 *Let $L = (V_N, V_T, V_l, P_L, S)$ be a LIG, its CF-backbone is the CFG, $G_L = (V_N, V_T, P_G, S)$, or simply G when L is understood, where:*

$$P_G = \{A \rightarrow w_1 A_1 \dots w_{i-1} A_{i-1} w_i A_i w_{i+1} A_{i+1} \dots w_p A_p w_{p+1} \mid A(\pi\alpha) \rightarrow w_1 A_1(\alpha_1) \dots w_{i-1} A_{i-1}(\alpha_{i-1}) w_i A_i(\pi\alpha_i) w_{i+1} A_{i+1}(\alpha_{i+1}) \dots w_p A_p(\alpha_p) w_{p+1} \in P_L\}.$$

If there is a one to one mapping between P_L and P_G the LIG is said to be *fair*. It is not very difficult to find an algorithm which transforms any LIG into an equivalent fair LIG. In the sequel we will only consider fair LIGs.

Due to the one to one mapping between (fair) LIGs and their CF-backbones we assume that iff $r_p()$ is a production in P_L, then r_p, with the same index p, denotes the corresponding production in its CF-backbone P_G.

Definition 6 *Let $L = (V_N, V_T, V_l, P_L, S)$ 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 $L^x = (V_N^x, V_T^x, V_l, P_L^x, S^x)$ s.t. G^x is its CF-backbone and each stack schema $(\pi_k\alpha_k)$ associated with the non-terminal $[A_k]$, occurring at position k in production $r_p() \in P_L^x$ is the stack schema of the object at position k in $\overline{r_p}() \in P_L$. More formally we have:*

$$P_L^x = \{r_p() = [A_0](\pi_0\alpha_0) \rightarrow [w_1][A_1](\pi_1\alpha_1) \dots [w_k][A_k](\pi_k\alpha_k) \dots [w_{m+1}] \mid r_p = [A_0] \rightarrow [w_1][A_1] \dots [w_k][A_k] \dots [w_{m+1}] \in P_G^x \wedge \overline{r_p}() = A_0(\pi_0\alpha_0) \rightarrow w_1 A_1(\pi_1\alpha_1) \dots w_k A_k(\pi_k\alpha_k) \dots w_{m+1} \in P_L\}$$

By construction, any LIGed forest is fair and between a LIG L and its LIGed forest L^x for x, the following property holds:

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

An object is said *initial* (resp. *final*) if it is secondary and occurs in the RHS (resp. LHS) of a production. V_O^I (resp. V_O^F) denotes the set of initial (resp. final) objects.

Definition 7 *For a given LIGed forest for x, we call spine, any sequence of $2p$ $(1 \leq p)$ objects $(o_1, o_2, \ldots, o_{2i-1}, o_{2i}, o_{2i+1}, \ldots, o_{2p})$ such that:*

- *o_1 (resp. o_{2p}) is an initial (resp. final) object.*
- *Inside objects o_j (if any) $(\forall j, 1 < j < 2p)$ are primary.*
- *$\forall i, 1 \leq i \leq p$, two consecutive objects $o_{2i-1} = X_1(\pi_1\alpha_1)$, and $o_{2i} = X(\pi\alpha)$ are such that $X_1 = X$, and o_{2i-1} (resp. o_{2i}) occurs in the RHS (resp. LHS) of a P_L^x production.*

This notion of spine is fundamental in LIG theory since it represents a path upon which stacks of symbols are evaluated. For example, followed in the direct way (top-down), the spine $(o_1 = X_1(\alpha_1), o_2 = X_1(..\alpha'_1), \ldots, o_{2i-1} = X_i(..\alpha_i), o_{2i} = X_i(..\alpha'_i), \ldots, o_{2p} = X_p(\alpha'_p))$ indicates that:

- a stack s is created and initialized with α_1 on the initial object o_1;
- if α'_1 is a suffix of s, then α'_1 is popped from s on object o_2;

$\vdots$

- the string of symbols α_i is pushed on s on object o_{2i-1};
- if α'_i is a suffix of s, then α'_i is popped from s on object o_{2i};

$\vdots$

- on the final object o_{2p}, if α'_p is a suffix of s, then α'_p is popped from s and the stack s is checked for emptiness.

A spine is said to be *valid* if each check sketched above succeeds[5].
More formally, these (valid) spines can be viewed as languages.

Definition 8 *Let $L^x = (V_N^x, V_T^x, V_l, P_L^x, S^x)$ be a LIGed forest for x. We define a valid spines grammar as being the LIG $L^{L^x} = (V_N^{L^x}, V_T^{L^x}, V_l, P^{L^x}, \langle Z \rangle)$ where:*

- *$\langle Z \rangle$ is the start symbol.*
- $V_N^{L^x} = \{\langle [B] \rangle \mid [B] \in V_N^x\} \cup \{\langle Z \rangle\}$
- *$V_T^{L^x} = V_O$, V_O is the set of objects.*
- $P^{L^x} = \{\langle [A] \rangle(..\alpha_1) \rightarrow [B](..\alpha_2)\langle [B] \rangle(..\alpha_2) \mid [A](..\alpha_1) \rightarrow \Gamma_1[B](..\alpha_2)\Gamma_2 \in P_L^x\} \cup \{\langle Z \rangle() \rightarrow [B](\alpha_2)\langle [B] \rangle(\alpha_2) \mid [A](\pi_1\alpha_1) \rightarrow \Gamma_1[B](\alpha_2)\Gamma_2 \in P_L^x\} \cup \{\langle [A] \rangle(\alpha_1) \rightarrow [A](\alpha_1) \mid [A](\alpha_1) \rightarrow \Gamma \in P_L^x\} \cup \{\langle Z \rangle() \rightarrow S^x()\langle S^x \rangle()\}$

The elements of the LIG language $\mathcal{L}(L^{L^x})$ are valid spines while the elements of its CF-backbone are merely spines. Note that its CF-backbone is a regular grammar.

[5]Of course it is possible to adopt the dual vision and to evaluate stacks along spines in the opposite (bottom-up) way. A stack is created and initialized with α'_p on the final object o_{2p}. Elements are pushed on LHS objects while they are checked and popped on RHS objects, and finally α_1 is popped on o_1 and the stack is checked for emptiness.

4 Our LIG Recognition Algorithm

In this paper we restrict our attention to LIGs with the following characteristics:

1. the RHS of a production contains at most two symbols;
2. the stack schema $(\pi\alpha)$ of any object (primary or secondary) is such that $0 \leq |\alpha| \leq 1$.

Recall that our recognition algorithm works on shared parse forests. Therefore, it is assumed that such a forest has been built by any general CF-parsing algorithm, working on the associated CF-backbone grammar, with a string x as input.

The reason why we allow at most two symbols in the RHS of the CF-backbone is to build the forest in time $\mathcal{O}(n^3)$. Moreover, in such a case, the parameters of the shared parse forest are kept within some suitable upper bounds: in particular the number of productions is $\mathcal{O}(n^3)$, the number of non-terminal symbols is $\mathcal{O}(n^2)$, the number of X-productions for any given $X = [A]_i^j$ is $\mathcal{O}(n)$ and the number of occurrences of such a non-terminal symbol X in the RHSs is also $\mathcal{O}(n)$.

The restriction on stack schemas, have been chosen only for pedagogic facilities. This restriction does not change neither our algorithm principle nor its upper bound complexity. Moreover, it is easy to see that this form of LIG constitutes a normal form.

In a first time, we will restrict our attention to non-cyclic CF-backbones. This restriction will guarantee that in any parse (sub-)tree, internal nodes are different from the root node. This restriction will be relaxed in Section 7.

Contrary to the previous section where we saw that a stack of symbols can be evaluated along spines, we choose not to compute stacks explicitly. The idea of our algorithm is based upon the remark that each time a symbol γ is pushed on a stack at a given place, this very symbol should be popped at some other place. The converse should also be true. The following will exhibit a mean by which this property could be checked without explicitly computing neither stacks nor spines.

We could remark that we are not interested in finding all the valid spines between any pair of objects (o_1, o_2), but only if there is at least one such valid spine. As a first consequence we will only consider *abridged spine*s (*a-spine* for short) $(o_1, o_2, o_4, \ldots, o_{2i}, o_{2i+2}, \ldots, o_{2p})$ which summarize all the spines $(o_1, o_2, o_3, o_4, \ldots, o_{2i}, o_{2i+1}, o_{2i+2}, \ldots, o_{2p})$ where the RHSs (odd) objects (except the initial one) have been erased.

The first purpose of our algorithm is to compute the relation *valid spine* which is the set of all couples (o_1, o_2) s.t. o_1 is an initial object, o_2 is a final object, and there is at least one valid spine between o_1 and o_2.

In order to reach this goal, for a given LIGed forest for x, we define on its objects V_O, $2|V_l|+1$ binary relations noted (for some γ in V_l) $\overset{\gamma}{\prec}$, $\overset{\gamma}{\succ}$, and $\prec\!\!\succ$. These relations between objects indicate the evolution of an imaginary stack between the first and the second object.

The element (o_1, o_2) of $\overset{\gamma}{\prec}$ (resp. $\overset{\gamma}{\succ}$) means that the stack associated with o_2 is built by pushing γ (resp. popping γ if possible) on top of the stack associated with o_1. The element (o_1, o_2) of $\prec\!\!\succ$ means that the stacks associated with o_1 and o_2 are identical.

Let $[X_1](\pi_1\alpha_1) \rightarrow \Gamma_1[X_2'](\pi_2'\alpha_2')\Gamma_2$ and $[X_2](\pi_2\alpha_2) \rightarrow \Gamma$ be two productions in P_L^x with $[X_2'] = [X_2]$. Moreover, assume that o_1, o_2', and o_2 respectively denotes the objects $[X_1](\pi_1\alpha_1)$, $[X_2'](\pi_2'\alpha_2')$, and $[X_2](\pi_2\alpha_2)$. The Table 1 indicates precisely the way these relations are defined. All other couples of objects are non comparable.

π_1	π'_2	π_2	Conditions	Relations
any	ε	any	$\alpha'_2 = \alpha_2$	$o'_2 \multimap o_2$
any	ε	..	$\alpha'_2 = \gamma \wedge \alpha_2 = \varepsilon$	$o'_2 \overset{\gamma}{\prec} o_2$
..	..	any	$\alpha'_2 = \alpha_2$	$o_1 \multimap o_2$
..	..	any	$\alpha'_2 = \gamma \wedge \alpha_2 = \varepsilon$	$o_1 \overset{\gamma}{\prec} o_2$
..	..	any	$\alpha'_2 = \varepsilon \wedge \alpha_2 = \gamma$	$o_1 \overset{\gamma}{\succ} o_2$

Table 1: $\overset{\gamma}{\prec}$, $\overset{\gamma}{\succ}$, and $\multimap$ definitions.

Our algorithm will simply compose the previous relations in order to relate an object where a symbol is pushed to the object(s) where this very symbol is popped in order to finally answer the question: is there at least one valid spine between o_1 and o_2 where o_1 is initial and o_2 is final?

Formally the valid spine relation, denoted by $\bowtie$, is defined by

$$\bowtie \;=\; \{(o_1, o_2) \mid o_1 \in V_O^I \wedge o_2 \in V_O^F \wedge o_1 \overset{+}{\approx} o_2\}$$

where $\approx$, the *valid sub-spine* relation, is the smallest solution of the following recursive equation

$$\approx \;=\; \multimap \cup \overset{\gamma}{\prec} \overset{*}{\approx} \overset{\gamma}{\succ}$$

We will implement this computation as the limit of the composition of the $\overset{\gamma}{\prec}$, $\multimap$, and $\overset{\gamma}{\succ}$ relations and we will show that this algorithm has an $\mathcal{O}(n^6)$-time upper bound complexity.

The laws governing this composition are shown in Table 2 where o_1 and o_3 are any objects and o_2 always designates a primary object[6].

These composition rules are applied until no more new element can be added to any of these relations[7].

If an initial object o_1 and a final object o_2 are such that $o_1 \bowtie o_2$, this means that there is (at least) one valid spine between these objects. Conversely if there are initial objects with no corresponding final object (in $\bowtie$), or final objects with no initial object, this means that there is no valid spine starting (or ending) at that object and that the productions where these

[6]If unrestricted stack schemas have been used, for example, the composition of $\overset{\alpha_1}{\prec}$ and $\overset{\alpha_2}{\succ}$ would have led to three possibilities, depending upon the stack suffixes α_1 and α_2, namely $\multimap$ if $\alpha_1 = \alpha_2$, $\overset{\alpha'_2}{\prec}$ if $\alpha_1 = \alpha'_2\alpha_2$, and $\overset{\alpha'_1}{\succ}$ if $\alpha'_1\alpha_1 = \alpha_2$.

[7]The algorithm in [3] computes all valid sub-spines of length k (k-spines) from $k-1$-spines and the initial relations. This is wrong since the set of k-spines can be empty though the set of h-spines with $h > k$, can be non empty. Moreover, k-spines can not always be composed from $k-1$-spines and 1-spines (or 1-spines and $k-1$-spines).

$o_1 \overset{\gamma}{\prec} o_2$	and	$o_2 \prec\!\succ o_3$	$\Longrightarrow$	$o_1 \overset{\gamma}{\prec} o_3$
$o_1 \overset{\gamma}{\prec} o_2$	and	$o_2 \overset{\gamma}{\succ} o_3$	$\Longrightarrow$	$o_1 \prec\!\succ o_3$
$o_1 \prec\!\succ o_2$	and	$o_2 \overset{\gamma}{\prec} o_3$	$\Longrightarrow$	$o_1 \overset{\gamma}{\prec} o_3$
$o_1 \prec\!\succ o_2$	and	$o_2 \prec\!\succ o_3$	$\Longrightarrow$	$o_1 \prec\!\succ o_3$
$o_1 \prec\!\succ o_2$	and	$o_2 \overset{\gamma}{\succ} o_3$	$\Longrightarrow$	$o_1 \overset{\gamma}{\succ} o_3$
$o_1 \overset{\gamma}{\succ} o_2$	and	$o_2 \prec\!\succ o_3$	$\Longrightarrow$	$o_1 \overset{\gamma}{\succ} o_3$

Table 2: Valid Composition of relations.

objects occur are invalid w.r.t. the LIG conditions and therefore should be erased. This erasing of productions in the LIGed forest L^x for x, creates a new LIG say $\underline{L^x}$.

The string x is an element of the initial LIG L iff the language of the CF-backbone for $\underline{L^x}$ is non empty.

The procedure in Table 3 implements the definition of the initial relations given in Table 1.

```
(1)   procedure init-relations ()
(2)     R = Φ
(3)     V_O^LHS = {o | o → Γ ∈ P_L^x}
(4)     for each o' = [X](π'α') in V_O^LHS do
(5)       for each o → Γ1[X](π2α2)Γ2 in P_L^x ∪ {S'() → S^x()} do
(6)         if π2 = ε then o = [X](π2α2) end if
(7)         if α2 = α' then R = R ∪ {(o, ≺≻, o')}
(8)         else if α2 = γ and α' = ε then R = R ∪ {(o, ≺^γ, o')}
(9)         else if α2 = ε and α' = γ then R = R ∪ {(o, ≻^γ, o')}
(10)        end if
(11)      end do
(12)    end do
(13)  end procedure
```

Table 3: The initial relations $\mathcal{R} = \overset{\gamma}{\prec} \cup \overset{\gamma}{\succ} \cup \prec\!\succ$.

At line (2) , $\mathcal{R}$,which will hold all the initial relations, is initialized to the empty set. Line (3) collects in V_O^{LHS} the LHS objects. The loop at lines (4–12) examines each such LHS object o' which is supposed to be an $[X]$-object. The embedded loop at lines (5–11) selects the productions with an $[X]$-object in RHS. Note that we have added a new production $S'() \rightarrow S^x()$ which introduces a new initial object $S^x()$ called the *start* object. This augmented LIG and its start object allow us to handle spines whose initial object non-terminal symbol is the LIG start symbol. The first member of a relation is a LHS object o, except when the RHS object $[X](\pi_2\alpha_2)$ is secondary (and therefore initial), this case is processed at line (6). The choice of the

relations is governed by the relative values of the stack schemas $(\pi_2\alpha_2)$ and $(\pi'\alpha')$. Lines (7–10) select the appropriate initial relation. The case where $\alpha = \gamma$, $\alpha' = \gamma'$, and $\gamma \neq \gamma'$ (push of γ immediately followed by a pop of γ') is erroneous.

An other view is to consider $\mathcal{R}$ as defining a FSA $(Q, \Sigma, q_0, \delta, F)$ in the following way:

- $Q = V_O \cup \{X^x(), S'()\}$.
- $\Sigma = Q \times \mathcal{S}$ where $\mathcal{S}$ is the set of $2|V_l| + 1$ relation symbols.
- $q_0 = S'()$.
- $F = V^F$
- $(o, r, o') \in \mathcal{R} \Longrightarrow o' \in \delta(o, (o', r))$ and $o \in V_o^I \cup \{S^x()\} \Longrightarrow o \in \delta(q_0, (o', \prec\!\!\succ))$

We can easily see that the strings in its language $(o_1, r_1) \ldots (o_i, r_i) \ldots (o_p, r_p)$ are such that $o_1 \ldots o_i \ldots o_p$ are the spines in definition 8 and that they are valid when the sequence of relation symbols $r_1 \ldots r_i \ldots r_p$ can be composed into the only relation $\prec\!\!\succ$. This FSA can be related with the one denoted M_{G_w} and defined in [18].

The procedure in Table 4 describes the way these initial relations are augmented in using the composition rules of Table 2. The parameters o_2, r and o_3 of this procedure are such that r is any relation symbol $\overset{\gamma}{\prec}$, $\overset{\gamma}{\succ}$, or $\prec\!\!\succ$ and that (o_2, r, o_3) is an element of $\mathcal{R}$. $\mathcal{U}$ is the subset of $\mathcal{R}$ whose elements have not yet been composed. This procedure tries to compose (o_2, r, o_3) to its left (resp. right) at lines (2–7) (resp. lines (8–13)) with an element (o_1, r_1, o_2) (resp. (o_3, r_1, o_4)) of $\mathcal{R}$ when this composition $r_1 \circ r$ (resp. $r \circ r_1$) is valid w.r.t. the rules of Table 2. If the resulting element (o_1, r', o_3) (resp. (o_2, r', o_4)) is new, it is added to both $\mathcal{R}$ (it's a new element) and $\mathcal{U}$ (it should be composed latter on). We see at lines (2) and (8) that both a left and a right compositions are tried. These left and right compositions are mandatories since the laws in Table 2 are not associative. For example look at the composition of $o_1 \overset{\gamma_a}{\prec} o_2 \overset{\gamma_b}{\prec} o_3 \overset{\gamma_b}{\succ} o_4$.

```
(1)    procedure compose (o2, r, o3)
(2)        for each (o1, r1, o2) in R do
(3)            if r1 ∘ r = r' ∧ (o1, r', o3) ∉ R then
(4)                R = R ∪ {(o1, r', o3)}
(5)                U = U ∪ {(o1, r', o3)}
(6)            end if
(7)        end do
(8)        for each (o3, r1, o4) in R do
(9)            if r ∘ r1 = r' ∧ (o2, r', o4) ∉ R then
(10)               R = R ∪ {(o2, r', o4)}
(11)               U = U ∪ {(o2, r', o4)}
(12)           end if
(13)       end do
(14)   end procedure
```

Table 4: The compose procedure.

```
(1)   function recognize (L, x) return boolean
(2)       let L = (V_N, V_T, V_l, P_L, S)
(3)       create G = (V_N, V_T, P_G, S) /* its CF-backbone */
(4)       create G^x = (V_N^x, V_T^x, P_G^x, S^x) /* its shared parse forest for x */
(5)       if L(G^x) = ∅ then return false end if
(6)       create L^x = (V_N^x, V_T^x, V_l, P_L^x, S^x) /* its LIGed forest */

(7)       call init-relations ()
(8)       U = R
(9)       for each (o_1, r, o_2) in U do
(10)          U = U − {(o_1, r, o_2)}
(11)          call compose (o_1, r, o_2)
(12)      end do
(13)      ⋈= {(o_1, o_2) | (o_1, -<>-, o_2) ∈ R ∧ o_1 ∈ V_O^I ∧ o_2 ∈ V_O^F}
(14)      I = {o_1 | (o_1, o_2) ∈⋈}
(15)      F = {o_2 | (o_1, o_2) ∈⋈}

(16)      if S^x() not in I then return false end if
(17)      for each r_p() = o_0 → Γ_1 o_h Γ_2 in P_L^x do
(18)          if o_0 in V_O^S and o_0 not in F or
(19)              o_h in V_O^S and o_h not in I then
(20)              erase r_p in P_G^x
(21)          end if
(22)      end do
(23)      return useless-symbol-elimination(P_G^x) ≠ ∅
(24)  end function
```

Table 5: The Recognition Algorithm.

The main function which describes our recognizing algorithm is in Table 5.

Its parameters are a LIG L and an input string x. At line (3), G denotes its CF-backbone. The shared parse forest G^x at line (4) is supposed to have been computed by any general CF-parsing algorithm. If $x \notin \mathcal{L}(G)$, it will not be in $\mathcal{L}(L)$ either (line (5)). At line (6), L^x denotes the corresponding LIGed forest. The set $\mathcal{U}$, whose initial value is set at line (8), holds at each time the subset of $\mathcal{R}$ which has not yet been composed. The loop at lines (9–12) tries to compose each new element. The ultimate goal is the computation of the valid spine relation $\bowtie$ at line (13). The set I line (14) (resp. F line (15)) contains all initial (resp. final) objects starting (resp. ending) a valid spine. When the start object $S^x()$ is not an element of I, this means that there is no valid spine starting at the root and therefore the recognizer failed (line (16)).

Since G^x is the CF-backbone of L^x, each time a production $r_p()$ in P_L^x contains a non valid initial or final object, its corresponding production r_p in P_G^x is erased (see lines (17–22)). At line (23) we assume that a classical algorithm eliminates from P_G^x all useless symbols. If the resulting production set is not empty, it contains a production of the form $[S]_0^n \rightarrow \ldots$ which

shows that x is a sentence of that reduced production set and therefore that x is an element of L^x and hence an element of L.

5 Its Complexity

Recall that the complexity of this algorithm is evaluated on binary form grammars (i.e. the length of their RHSs is at most two). The influence of this restriction is discussed at the end of this section.

Objects in LIGed forests are of the form $[A]_i^j(\pi\alpha)$. The maximum number of ranged-symbols $[A]_i^j$ is $\mathcal{O}(n^2)$ where n is the length of the input string. All other parameters (non-terminals and stack schemas) are constant for a given LIG L. Therefore, the size of any set which contains objects has an $\mathcal{O}(n^2)$ upper bound, especially I, F, and V_O^{LHS} while the maximum size of the relations (sets of couples) is $\mathcal{O}(n^4)$. Moreover, we assume that it takes a constant time to test or to add any element in a set.

5.1 Complexity of the *init-relations* procedure

In Table 3, we have:

line (3) A single pass over P_L^x computes V_O^{LHS}, whose size is $\mathcal{O}(n^2)$, in time $\mathcal{O}(n^3)$.

lines (6–10) Each activation of this body is performed in constant time.

lines (5–11) For a given ranged-non-terminal $[X]$ there are at most $\mathcal{O}(n)$ occurrences of $[X]$ in the RHSs of P_L^x. Therefore, each activation of this loop takes $\mathcal{O}(n)$ time.

lines (4–12) That loop is executed $\mathcal{O}(n^2)$ times so it takes $\mathcal{O}(n^3)$ time.

lines (1–13) At the end the time complexity of *init-relations* is $\mathcal{O}(n^3)$.

Since the body part (lines (7–10)) where the initial relations are computed is executed at most $\mathcal{O}(n^3)$ times, the initial size of $\mathcal{R}$ is $\mathcal{O}(n^3)$. When the grammar is in binary form, the number of $[X]_i^j$-productions and the number of productions where the non-terminal $[X]_i^j$ occurs in RHS is $\mathcal{O}(n)$. This means that the number of triples (o, r, o') or (o', r, o) in $\mathcal{R}$ where a given object o occurs as first or last member is at most $\mathcal{O}(n)$.

5.2 Complexity of the *compose* procedure

In Table 4, we have:

lines (3–6) and (9–12) Each of these loop bodies takes a constant time.

lines (2–7) (resp. (8–13)) For any object o_2 (resp. o_3) this loop is executed $\mathcal{O}(n^2)$ times.

Therefore, in the worst case, the time complexity of each call of the *compose* procedure is $\mathcal{O}(n^2)$.

5.3 Complexity of the Recognition Algorithm

In Table 5, we have:

line (4) Can take $\mathcal{O}(n^3)$ with the appropriate CF-parsing algorithm since the length of the longest RHS is two.

line (6) The LIGed forest is almost simply a copy of the shared parse forest and therefore takes $\mathcal{O}(n^3)$.

line (7) Takes $\mathcal{O}(n^3)$ (see 5.1).

lines (9–12) The cardinality of $\mathcal{U}$ is $\mathcal{O}(n^4)$ and each element is examined only once, this loop is executed $\mathcal{O}(n^4)$ times and since each call to *compose* takes $\mathcal{O}(n^2)$ (see 5.2), the overall time is at most $\mathcal{O}(n^6)$.

lines (13–15) These sets could be computed as a by product of the *init-relations* or *compose* procedures, here they are extracted in $\mathcal{O}(n^4)$ time.

lines (17–22) Takes $\mathcal{O}(n^3)$.

line (23) A classical algorithm for the elimination of useless symbol is performed in time linear with the size of the grammar, so in our case it will take $\mathcal{O}(n^3)$.

Therefore, in the worst case, for a non cyclic grammar in binary form, the time complexity of our recognition algorithm is $\mathcal{O}(n^6)$. Its space complexity is $\mathcal{O}(n^4)$ since it essentially handles on one side a LIGed forest whose size is $\mathcal{O}(n^3)$ and on the other side relations whose size is $\mathcal{O}(n^4)$ since they are mainly couples of objects.

We can wonder whether a lower upper bound can be reached for some sub-classes of LIGs. When the relations are such that their cardinality is in $\mathcal{O}(n^2)$ and each object could be related with at most $\mathcal{O}(n)$ other objects, it is not difficult to see that our recognizer has an $\mathcal{O}(n^3)$ worst time bound, but unfortunately we are not aware of any grammatical characterization of such a sub-class!

Of course, this recognizer should only be considered as a principle algorithm which must be improved in practical implementations. One of its defaults is shown by the following example. Assume that five objects are related by $o_1 \prec\!\!\diamond\!\!\succ o_2 \overset{\gamma}{\prec} o_3 \prec\!\!\diamond\!\!\succ o_4 \overset{\gamma}{\succ} o_5$, we easily see that the element $o_1 \prec\!\!\diamond\!\!\succ o_5$ can be got after only three compositions though our algorithm will perform fourteen compositions! It can be modified to introduce a composition order to avoid the construction of elements in multiple ways. Though the worst upper bound is not modified, this improved algorithm can for example check the LIG conditions in linear time when the CF-backbone is unambiguous. In such a case we know that the shared parse forest is a simple (parse) tree whose size is $\mathcal{O}(n)$ which can be built in time $\mathcal{O}(n^2)$ by an Earley or generalized LR parsing algorithm (see [6]). Therefore, for unambiguous grammars, a total recognition time of $\mathcal{O}(n^2)$ can be reached.

This shows that our algorithm closely depends upon the actual number of elements in the relations, and that the complexity decreases with these values. Since the $)(n^4)$ maximum size seems seldom found in practice, the average behavior of our recognizer is better than its worst case.

It should be pointed out that our algorithm is valid, even without restricting the maximum length l of the RHSs. The only consequence is that the recognizing time can be increased since the CF-parsing time (and the size of the shared parse forest) can be of the order $\mathcal{O}(n^{l+1})$. With this hypothesis, the time taken by the *init-relations* procedure becomes $\mathcal{O}(n^{l+1})$. However, the cardinalities of the $\mathcal{R}$ and $\mathcal{U}$ relations stay in $\mathcal{O}(n^4)$ and therefore the checking of the LIG conditions stays in time $\mathcal{O}(n^6)$. Finally, without restriction, a fair non cyclic LIG can be recognized by our algorithm in time $\max(\mathcal{O}(n^{l+1}), \mathcal{O}(n^6))$ and space $\max(\mathcal{O}(n^{l+1}), \mathcal{O}(n^4))$.

6 An 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:

$$\begin{array}{llll} S(..) \to S(..\gamma_a)a & S(..) \to S(..\gamma_b)b & S(..) \to S(..\gamma_c)c & S(..) \to T(..) \\ T(..\gamma_a) \to aT(..) & T(..\gamma_b) \to bT(..) & T(..\gamma_c) \to cT(..) & T() \to c \end{array}$$

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 the last production $T() \to c$, 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).

Let $x = ccc$ be an input string, we wish to know whether x is an element of $\mathcal{L}(L)$.

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:

$$\begin{array}{lll} [S]_0^3 \to [S]_0^2 c & [S]_0^3 \to [T]_0^3 & [S]_0^2 \to [S]_0^1 c \\ [S]_0^2 \to [T]_0^2 & [S]_0^1 \to [T]_0^1 & [T]_0^3 \to c[T]_1^3 \\ [T]_1^3 \to c[T]_2^3 & [T]_2^3 \to c & [T]_0^2 \to c[T]_1^2 \\ [T]_1^2 \to c & [T]_0^1 \to c & \end{array}$$

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:

$$\begin{array}{lll} [S]_0^3(..) \to [S]_0^2(..\gamma_c)c & [S]_0^3(..) \to [T]_0^3(..) & [S]_0^2(..) \to [S]_0^1(..\gamma_c)c \\ [S]_0^2(..) \to [T]_0^2(..) & [S]_0^1(..) \to [T]_0^1(..) & [T]_0^3(..\gamma_c) \to c[T]_1^3(..) \\ [T]_1^3(..\gamma_c) \to c[T]_2^3(..) & [T]_2^3() \to c & [T]_0^2(..\gamma_c) \to c[T]_1^2(..) \\ [T]_1^2() \to c & [T]_0^1() \to c & \end{array}$$

In this LIGed forest there are three a-spines which are shown below with their objects separated by the appropriate initial relations:

$$[S]_0^3() \leadsto [S]_0^3(..) \overset{\gamma_c}{\prec} [S]_0^2(..) \overset{\gamma_c}{\prec} [S]_0^1(..) \leadsto [T]_0^1()$$
$$[S]_0^3() \leadsto [S]_0^3(..) \overset{\gamma_c}{\prec} [S]_0^2(..) \overset{\gamma_c}{\succ} [T]_0^2(..\gamma_c) \leadsto [T]_1^2()$$
$$[S]_0^3() \leadsto [S]_0^3(..) \overset{\gamma_c}{\succ} [T]_0^3(..\gamma_c) \overset{\gamma_c}{\succ} [T]_1^3(..\gamma_c) \leadsto [T]_2^3()$$

Though these a-spines are not computed by our algorithm, it is easier to see what happens directly on them. In particular we can see that the first and last a-spine are not valid since there is $\overset{\gamma_c}{\prec}$ (or $\overset{\gamma_c}{\succ}$) without corresponding $\overset{\gamma_c}{\succ}$ (or $\overset{\gamma_c}{\prec}$) and that only the middle a-spine is valid. In fact the *init-relations* procedure computes the initial relations (shown within the a-spines), and the calls to *compose* add to $\mathcal{R}$ the following elements:

$$[S]_0^3() \overset{\gamma_c}{\prec} [S]_0^2(..) \qquad [S]_0^2(..) \overset{\gamma_c}{\prec} [T]_0^1() \qquad [S]_0^3(..) \leadsto [T]_0^2(..\gamma_c)$$
$$[S]_0^2(..) \overset{\gamma_c}{\succ} [T]_1^2() \qquad [S]_0^3() \overset{\gamma_c}{\succ} [T]_0^3(..\gamma_c) \quad [T]_0^3(..\gamma_c) \overset{\gamma_c}{\succ} [T]_2^3()$$
$$[S]_0^3() \leadsto [T]_0^2(..\gamma_c) \quad [S]_0^3(..) \leadsto [T]_1^2() \quad [S]_0^3() \leadsto [T]_1^2()$$

The valid-spine relation $\bowtie$ is $\{([S]_0^3(), [T]_1^2())\}$ and we have $I = \{[S]_0^3()\}$ and $F = \{[T]_1^2()\}$.

Since the start object $[S]_0^3()$ is in I, the execution of lines (17–22) in Table 5 leads to erase the productions $[T]_0^1 \to c$, and $[T]_2^3 \to c$ in P_G^x. The *useless-symbol-elimination* function called at line (23) returns the following (non empty) production set:

$$[S]_0^3 \to [S]_0^2 c \quad [S]_0^2 \to [T]_0^2 \quad [T]_0^2 \to c[T]_1^2 \quad [T]_1^2 \to c$$

which shows that $ccc \in \mathcal{L}(L)$.

We can remark that, with that example, our recognition algorithm is in fact a parsing algorithm (i.e. all resulting a-spines are valid). This is not always the case. Assume a LIGed forest with the following four a-spines: $s_1 = (o_1, \ldots, o_3)$, $s_2 = (o_1, \ldots, o_4)$, $s_3 = (o_2, \ldots, o_3)$, and $s_4 = (o_2, \ldots, o_4)$. Moreover assume that the only valid a-spines are s_1 and s_4, therefore, the algorithm will consider that o_1 and o_2 are valid initial objects and that o_3 and o_4 are valid final objects and that no production elimination should take place. Therefore, the LIGed forest is left unchanged but could not be considered as a representation of the shared parse forest for the initial LIG since there are a-spines s_2 and s_3 which are not valid.

7 The Case of Cyclic Grammars

In this section, we examine the case of cyclic grammars.

When the CF-backbone of a LIG is cyclic, two objects o_i and o_j can be identical in some a-spines $(o_1, \ldots, o_i, \ldots, o_j, \ldots, o_{2p})$. In fact, if such a cyclic a-spine exists, there is an unbounded number of a-spines in which the cycle $(o_i, \ldots, o_j)$ is repeated 1, 2, ... times. The question being, whether there is among these a-spines, at least one along which a stack of symbols may be correctly evaluated (i.e. is there a valid a-spine)?

We could check that our recognizer is valid even when the grammar is cyclic, except that it is possible to avoid the storing of elements like $(o, \leadsto, o)$. If an object o is such that $o \leadsto o$, this means that if a stack of symbols is α when reaching the object o, this stack is still α when leaving o, after being modified along the loop summarized by $o \leadsto o$. Therefore, as long as LIG

conditions are considered, this case does not need to be registered. We note that an element $o \prec\!\!\!\prec o$ will appear in $\mathcal{R}$ only if there is a loop around o along which the composition of the relations leads to r. Therefore, the space and time complexity is not changed.

This shows that our recognition algorithm has a time complexity of $\mathcal{O}(n^6)$ and a space complexity of $\mathcal{O}(n^4)$ in all cases, even when the grammar is cyclic.

8 A Cyclic LIG Example

The following LIG, where A is the start symbol:

$$A(..) \rightarrow A(..\gamma_a) \quad A(..) \rightarrow B(..) \quad B(..\gamma_a) \rightarrow B(..) \quad B() \rightarrow a$$

is cyclic (we have $A \stackrel{+}{\Rightarrow} A$ and $B \stackrel{+}{\Rightarrow} B$ in its CF-backbone), and the stack schemas in production $A(..) \rightarrow A(..\gamma_a)$ indicate that an unbounded number of push γ_a actions could take place, while production $B(..\gamma_a) \rightarrow B(..)$ indicates an unbounded number of pops. Its CF-backbone is unbounded ambiguous and therefore, the number of parse trees for the input string $x = a$ is not bounded. However its shared parse forest (which is itself a cyclic CFG) and the corresponding LIGed forest can be computed. Let o_1–o_6 be denotations for the following objects:

$$\begin{array}{lll} o_1 = [A]_0^1() & o_2 = [A]_0^1(..) & o_3 = [A]_0^1(..\gamma_a) \\ o_4 = [B]_0^1(..) & o_5 = [B]_0^1(..\gamma_a) & o_6 = [B]_0^1() \end{array}$$

For $x = a$, the start object of its LIGed forest is o_1 and its production set is:

$$o_2 \rightarrow o_3 \quad o_2 \rightarrow o_4 \quad o_5 \rightarrow o_4 \quad o_6 \rightarrow a$$

The initial relations extracted from this LIG by the algorithm in Table 3 are:

$$\begin{array}{rcl} \mathcal{R} & = & \{(o_1, \prec\!\!\!\prec, o_2), (o_2, \prec\!\!\!\prec, o_6), (o_5, \prec\!\!\!\prec, o_6)\} \\ & \bigcup & \{(o_2, \overset{\gamma_a}{\prec}, o_2)\} \\ & \bigcup & \{(o_2, \overset{\gamma_a}{\succ}, o_5), (o_5, \overset{\gamma_a}{\succ}, o_5)\} \end{array}$$

Below we find the elements added by the call to *compose*.

$$\begin{array}{lllll} (o_1, \prec\!\!\!\prec, o_5) & (o_1, \prec\!\!\!\prec, o_6) & (o_2, \prec\!\!\!\prec, o_5) & & \\ (o_1, \overset{\gamma_a}{\prec}, o_2) & (o_1, \overset{\gamma_a}{\prec}, o_5) & (o_1, \overset{\gamma_a}{\prec}, o_6) & (o_2, \overset{\gamma_a}{\prec}, o_5) & (o_2, \overset{\gamma_a}{\prec}, o_6) \\ (o_1, \overset{\gamma_a}{\succ}, o_5) & (o_1, \overset{\gamma_a}{\succ}, o_6) & (o_2, \overset{\gamma_a}{\succ}, o_6) & (o_5, \overset{\gamma_a}{\succ}, o_6) & \end{array}$$

The extraction of the valid-spine relation and of the sets I and F gives:

$$\begin{array}{rcl} \bowtie & = & \{(o_1, o_6)\} \\ I & = & \{o_1\} \\ F & = & \{o_6\} \end{array}$$

Since the only initial object is o_1, and the only final object is o_6, no production is erased at lines (17–22) in Table 5 and therefore the output of *useless-symbol-elimination* is the initial shared parse forest itself. This shows that the string a is an element of the given cyclic LIG.

9 Conclusion

In this paper we have presented a new recognition algorithm which works for the class of mildly context-sensitive languages. Though its worst case complexity does not improve over previous ones (i.e. a $\mathcal{O}(n^6)$ time and $\mathcal{O}(n^4)$ space are achieved), the recognizer behaves in practice much faster than its worst case.

The advantages of this algorithm can mainly be summarized as follows:

- parsing of the input string with the underlying CFG and checking of the LIG conditions are split into separate phases;
- LIG conditions checking relies upon a very simple principle which can be expressed by binary relations;
- the recognition test is simply performed by composition of the previous relations;
- therefore, no symbol stack computation is needed;
- it can be applied to unrestricted fair LIGs (though the $\mathcal{O}(n^6)$ limit can then be exceeded).

We can wonder whether the first point is really an advantage since it can be retorted that illegal paths should be aborted as soon as possible. Our argument is that it wastes time to compute symbol stacks in $\mathcal{O}(n^6)$ along paths which can be discovered as syntactically illegal in $\mathcal{O}(n^3)$.

This algorithm is implemented in a prototype system which is part of an ongoing effort to get a set of parsers for various NL formalisms.

References

[1] Abeillé, A., and Schabes, Y. 1989. Parsing idioms in lexicalized TAGs. *Proceedings of the fourth conference of the ACL.*

[2] Aho, A. V. 1968. Indexed grammars—An extension to context free grammars. *J. ACM*, Vol. 15, pp. 647-671.

[3] Boullier, P. 1995. Yet another $\mathcal{O}(n^6)$ recognition algorithm for mildly context-sensitive languages. *Proceedings of the fourth international workshop on parsing technologies*, pp 34-47.

[4] Earley, Jay C. 1968. An efficient context-free parsing algorithm. *Ph.D. thesis*, Carnegie-Mellon University, Pittsburgh, PA.

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

[6] Kipps, J. R. 1989. Analysis of Tomita's algorithm for general context-free parsing. *International Parsing Workshop'89*, pp. 193-202.

[7] Kilger, A., and Finkler, W. 1993. TAG-based incremental generation. *German Research Center for Artificial Intelligence (DFKI), Technical Report*, Saarbrücken (Germany).

[8] LANG, B. 1974. Deterministic techniques for efficient non-deterministic parsers. *Automata, Languages and Programming, 2nd Colloquium*, Lectures Notes in Computer Science, Springer-Verlag, Vol. 14, pp. 255-269.

[9] LANG, B. 1991. Towards a uniform formal framework for parsing. *Current Issues in Parsing Technology*, edited by M. Tomita, Kluwer Academic Publishers, pp. 153-171.

[10] LANG, B. 1994. Recognition can be harder than parsing. *Computational Intelligence*, Vol. 10, No. 4, pp. 486-494.

[11] PAROUBEK, P., SCHABES, Y., and JOSHI, A. K. 1992. XTAG-a graphical workbench for developing tree-adjoining grammars. *Third Conference on Applied Natural Language Processing*, Trento (Italy).

[12] POLLER, P. 1994. Incremental parsing with LD/TLP-TAGs. *Computational Intelligence*, Vol. 10, No. 4, pp. 549-562.

[13] REKERS, J. 1992. Parser generation for interactive environments. *Ph.D. thesis*, University of Amsterdam.

[14] SCHABES, Y. 1994. Left to right parsing of lexicalized tree-adjoining grammars. *Computational Intelligence*, Vol. 10, No. 4, pp. 506-524.

[15] TOMITA, M. 1987. An efficient augmented context-free parsing algorithm. *Computational Linguistics*, Vol. 13, pp. 31-46.

[16] VIJAY-SHANKER, K. 1987. A study of tree adjoining grammars. *PhD. thesis*, University of Pennsylvania.

[17] VIJAY-SHANKER, K., and JOSHI, A. K. 1985. Some computational properties of tree adjoining grammars. *23rd Meeting of the Association for Computational Linguistics*, Chicago, pp. 82-93.

[18] VIJAY-SHANKER, K., and WEIR D. J. 1993. The Used of Shared Forests in Tree Adjoining Grammar Parsing. *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL'93)*, Utrecht, The Netherlands, pp. 384-393.

[19] VIJAY-SHANKER, K., and WEIR D. J. 1994. Parsing some constrained grammar formalisms. *ACL Computational Linguistics*, Vol. 19, No. 4, pp. 591-636.

[20] YOUNGER, D. H. 1965. Recognition and parsing of context-free languages in time n^3. *Information and Control*, Vol. 10, No. 2, pp. 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