NEW TABULAR ALGORITHMS FOR LIG PARSING

Miguel A. Alonso
Jorge Graña
Manuel Vilares
Departamento de Computación
Universidad de La Coruña
Campus de Elviña s/n
15071 La Coruña (Spain)
{alonso,grana,vilares}@dc.fi.udc.es

Eric de la Clergerie
INRIA
Domaine de Voluceau
Rocquecourt, B.P. 105
78153 Le Chesnay (France)
Eric.De_La_Clergerie@inria.fr

Abstract

We develop a set of new tabular parsing algorithms for Linear Indexed Grammars, including bottom-up algorithms and Earley-like algorithms with and without the valid prefix property, creating a continuum in which one algorithm can in turn be derived from another. The output of these algorithms is a shared forest in the form of a context-free grammar that encodes all possible derivations for a given input string.

1 Introduction

Tree Adjoining Grammars (TAG) [8] and Linear Indexed Grammars (LIG) [7] are extensions of Context Free Grammars (CFG). Tree adjoining grammars use trees instead of productions as primary representing structure and seems to be adequate to describe syntactic phenomena occurring in natural language, due to their extended domain of locality and to their ability for factoring recursion from the domain of dependencies. Linear indexed grammars associate a stack of indices with each non-terminal symbol, with the restriction that the indices stack of the head non-terminal of each production (the *father*) can be inherited by at most one body non-terminal (the *dependent child*) while the other stacks must have a bounded stack size.

Several parsing algorithms have been proposed for TAG, ranging from simple bottom-up algorithms, like CYK [17], to sophisticated extensions of the Earley's algorithm [9]. In order to improve efficiency, it is usual to translate the source tree adjoining grammar into a linear indexed grammar [16, 12, 13, 17]. However, in some cases is not possible to translate the parsing strategy from TAG to LIG, as there are parsing strategies for TAG which are not incorporated in any parsing algorithm for LIG. To eliminate this drawback, we present in this paper several parsing algorithms for LIG which mimic the most popular parsing strategies for TAG [1].

1.1 Linear Indexed Grammars

A linear indexed grammar is a tuple (V_T, V_N, V_I, P, S), where V_T is a finite set of terminals, V_N a finite set of non-terminals, V_I is a finite set of indices, $S \in V_N$ is the start symbol and P is a finite set of productions. Following [7] we consider productions in which at most one element can be pushed on

or popped from a stack of indices:

$$A_0[\circ\circ\gamma] \rightarrow A_1[\,]\ldots A_{i-1}[\,]\ A_i[\circ\circ\gamma']\ A_{i-1}[\,]\ \ldots A_m[\,]$$

$$A_0[\,] \rightarrow a$$

where m is the length of the production, $A_j \in V_N$ for each $0 \leq j \leq m$, A_i is the dependent child, $\circ\circ$ is the part of the indices stack transmitted from the father to the dependent child, $\gamma, \gamma' \in V_I \cup \{\epsilon\}$ and for each production either γ or γ' or both must be ϵ and $a \in V_T \cup \{\epsilon\}$.

The derivation relation $\Rightarrow$ is defined for LIG as $\Upsilon \Rightarrow \Upsilon'$

- if $\Upsilon = \Upsilon_1 A[\alpha\gamma]\ \Upsilon_4$ and there exists a production $A[\circ\circ\gamma] \rightarrow \Upsilon_2 A'[\circ\circ\gamma']\ \Upsilon_3$ such that $\Upsilon' = \Upsilon_1\Upsilon_2 A'[\alpha\gamma']\ \Upsilon_3\Upsilon_4$
- or else if $\Upsilon = \Upsilon_1 A[\,]\ \Upsilon_4$ and there exists a production $A[\,] \rightarrow a$ such that $\Upsilon' = \Upsilon_1\ a\ \Upsilon_4$

where $A \in V_N$, $\alpha \in V_I^*$ and $\gamma, \gamma' \in V_I \cup \{\epsilon\}$. The reflexive and transitive closure of $\Rightarrow$ is denoted by $\stackrel{*}{\Rightarrow}$. The language defined by a LIG is the set of strings $w \in V_T^*$ such that $S[\,] \stackrel{*}{\Rightarrow} w$.

To parse this type of grammars, tabulation techniques with polynomial complexity can be designed based on a property defined in [17], that we call *context-freeness property of LIG*, establishing that if $A[\gamma] \stackrel{*}{\Rightarrow} uB[\,]w$ where $u, w \in V_T^*$, $A, B \in V_N$, $\gamma \in V_I \cup \{\epsilon\}$ and $B[\,]$ is a dependent descendant of $A[\gamma]$, then for each $\Upsilon_1, \Upsilon_2 \in (V_N[V_I^*] \cup V_T)^*$and $\beta \in V_I^*$ we have $\Upsilon_1 A[\beta\gamma]\Upsilon_2 \stackrel{*}{\Rightarrow} \Upsilon_1 uB[\beta]w\Upsilon_2$. Also, if $B[\gamma]$ is a dependent descendant of $A[\,]$ and $A[\,] \stackrel{*}{\Rightarrow} uB[\gamma]w$ then $\Upsilon_1 A[\beta]\Upsilon_2 \stackrel{*}{\Rightarrow} \Upsilon_1 uB[\beta\gamma]w\Upsilon_2$.

1.2 Parsing Schemata

We will describe parsing algorithms using *Parsing Schemata*, a framework for high-level description of parsing algorithms [15]. An interesting application of this framework is the analysis of the relations between different parsing algorithms by studying the formal relations between their underlying parsing schemata.

A *parsing system* for a grammar G and string $a_1 \ldots a_n$ is a triple $\langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$, with $\mathcal{I}$ a set of *items* which represent intermediate parse results, $\mathcal{H}$ an initial set of items called *hypothesis* that encodes the sentence to be parsed, and $\mathcal{D}$ a set of *deduction steps* that allow new items to be derived from already known items. Deduction steps are of the form $\frac{\eta_1,\ldots,\eta_k}{\xi}$ *cond*, meaning that if all antecedents η_i of a deduction step are present and the conditions *cond* are satisfied, then the consequent ξ should be generated by the parser. A set $\mathcal{F} \subseteq \mathcal{I}$ of *final items* represent the recognition of a sentence. A *parsing schema* is a parsing system parameterized by a grammar and a sentence.

Parsing schemata are closely related to *grammatical deduction systems* [14], where items are called *formula schemata*, deduction steps are *inference rules*, hypothesis are *axioms* and final items are *goal formulas*.

2 A CYK-like Algorithm

We have chosen the CYK-like algorithm for LIG described in [16] as our starting point. Due to the intrinsic limitations of this pure bottom-up algorithm, the grammars it can deal with are restricted to those having two elements, or one element which must be a terminal, in the right-hand side of each production. This restriction could be considered as the transposition of the Chomsky normal form to linear indexed grammars.

The algorithm works by recognizing in a bottom-up way the part of the input string spanned by each grammar element. The items used in the tabular interpretation of this algorithm are of the form $[A, \gamma, i, j \mid B, p, q]$ and represent one of the following types of derivation:

- $A[\gamma] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p \; B[\,] \; a_{q+1} \ldots a_j$ if and only if $(B, p, q) \neq (-, -, -)$, $B[\,]$ is a dependent descendent of $A[\gamma]$ and $(p, q) \leq (i, j)$.
- $A[\,] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j$ if and only if $\gamma = -$ and $(B, p, q) = (-, -, -)$.

where $-$ means that the value of a component is not bound and $(p, q) \leq (i, j)$ is used to represent that $i \leq p \leq q \leq j$ when p and q are bound.

These items are like those proposed for the tabulation of linear indexed automata [10] and for the tabulation of bottom-up 2–stack automata [6]. They are slightly different from the items of the form $[A, \gamma, i, j \mid B, \eta, p, q]$ proposed by Vijay-Shanker and Weir in [16] for their CYK-like algorithm, where the element $\eta \in V_I$ is redundant: due to the context-freeness property of LIG we have that if $A[\gamma] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p \; B[\,] \; a_{q+1} \ldots a_j$ then for any β we have that $A[\beta\gamma] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p \; B[\beta] \; a_{q+1} \ldots a_j$.

Schema 1 *The parsing system $\mathbb{P}_{\text{CYK}}$ corresponding to the CYK-like parsing algorithm for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \ldots a_n$ is defined as follows:*

$$\mathcal{I}_{\text{CYK}} = \{ \; [A, \gamma, i, j \mid B, p, q] \mid A, B \in V_N, \; \gamma \in V_I, \; 0 \leq i \leq j \, , \;\; (p, q) \leq (i, j) \; \}$$

$$\mathcal{H}_{\text{CYK}} = \{ \; [a, i-1, i] \mid a = a_i, \; 1 \leq i \leq n \; \}$$

$$\mathcal{D}^{\text{Scan}}_{\text{CYK}} = \frac{[a, j, j+1]}{[A, -, j, j+1 \mid -, -, -]} \quad A[\,] \rightarrow a \in P$$

$$\mathcal{D}^{[\circ\circ\gamma][\,][\circ\circ]}_{\text{CYK}} = \frac{\begin{array}{c}[B, -, i, k \mid -, -, -], \\ [C, \eta, k, j \mid D, p, q]\end{array}}{[A, \gamma, i, j \mid C, k, j]} \quad A[\circ\circ\gamma] \rightarrow B[\,] \; C[\circ\circ] \in P$$

$$\mathcal{D}^{[\circ\circ\gamma][\circ\circ][\,]}_{\text{CYK}} = \frac{\begin{array}{c}[B, \eta, i, k \mid D, p, q], \\ [C, -, k, j \mid -, -, -]\end{array}}{[A, \gamma, i, j \mid B, i, k]} \quad A[\circ\circ\gamma] \rightarrow B[\circ\circ] \; C[\,] \in P$$

$$\mathcal{D}^{[\circ\circ][\,][\circ\circ]}_{\text{CYK}} = \frac{\begin{array}{c}[B, -, i, k \mid -, -, -], \\ [C, \eta, k, j \mid D, p, q]\end{array}}{[A, \eta, i, j \mid D, p, q]} \quad A[\circ\circ] \rightarrow B[\,] \; C[\circ\circ] \in P$$

$$\mathcal{D}^{[\circ\circ][\circ\circ][\,]}_{\text{CYK}} = \frac{\begin{array}{c}[B, \eta, i, k \mid D, p, q], \\ [C, -, k, j \mid -, -, -]\end{array}}{[A, \eta, i, j \mid D, p, q]} \quad A[\circ\circ] \rightarrow B[\circ\circ] \; C[\,] \in P$$

$$\mathcal{D}^{[\circ\circ][\,][\circ\circ\gamma]}_{\text{CYK}} = \frac{\begin{array}{c}[B, -, i, k \mid -, -, -], \\ [C, \gamma, k, j \mid D, p, q], \\ [D, \eta, p, q \mid E, r, s]\end{array}}{[A, \eta, i, j \mid E, r, s]} \quad A[\circ\circ] \rightarrow B[\,] \; C[\circ\circ\gamma] \in P$$

$$\mathcal{D}^{[\circ\circ][\circ\circ\gamma][\,]}_{\text{CYK}} = \frac{\begin{array}{c}[B, \gamma, i, k \mid D, p, q], \\ [C, -, k, j \mid -, -, -], \\ [D, \eta, p, q \mid E, r, s]\end{array}}{[A, \eta, i, j \mid E, r, s]} \quad A[\circ\circ] \rightarrow B[\circ\circ\gamma] \; C[\,] \in P$$

$$\mathcal{D}_{\text{CYK}} = \mathcal{D}^{\text{Scan}}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ\gamma][\,][\circ\circ]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ\gamma][\circ\circ][\,]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ][\,][\circ\circ]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ][\circ\circ][\,]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ][\,][\circ\circ\gamma]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ][\circ\circ\gamma][\,]}_{\text{CYK}}$$

$$\mathcal{F}_{\text{CYK}} = \{ \; [S, -, 0, n \mid -, -, -] \; \}$$

The hypotheses defined for this parsing system are the standard ones and therefore they will be omitted in the remaining parsing systems described in this paper.

Steps in the set $\mathcal{D}_{\text{CYK}}^{\text{Scan}}$ are in charge of starting the parsing process. The other steps are in charge of combining the items corresponding to the elements in the right-hand side of a production in order to generate the item corresponding to the left-hand side element, propagating bottom-up the information about the indices stack.

The space complexity of the algorithm with respect to the length n of the input string is $\mathcal{O}(n^4)$, as each item stores four positions of the input string. The time complexity is $\mathcal{O}(n^6)$ and it is given by the deduction steps in $\mathcal{D}_{\text{CYK}}^{[\circ\circ\gamma]\text{right}}$ and $\mathcal{D}_{\text{CYK}}^{[\circ\circ\gamma]\text{left}}$. Although these steps involve 7 positions of the input string, by partial application each step can be decomposed in a set of deduction steps involving at most 6 positions.

A CYK-like algorithm generalized for linear indexed grammar with productions manipulating more than one symbol at the top of the indices stacks is described in [17]. The same kind of generalization could be incorporated into the algorithm presented here.

3 A Bottom-up Earley-like Algorithm

The CYK-like algorithm has an important limitation: it can only be applied to linear indexed grammars having at most two children in the right-hand side. To overcome this limitation we use dotted production into items instead single nodes. Thus, we can distinguish the part of a production already processed from the part not yet processed. With respect to notation, we use $\boldsymbol{A}$ to denote a grammar element having the non-terminal A when the associated indices stack is not relevant in that context.

The items of the new parsing system $\mathbb{P}_{\text{buE}}$ are of the form $[\boldsymbol{A} \rightarrow \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B, p, q]$ and can be obtained by refining the items in $\mathbb{P}_{\text{CYK}}$. They represent one of the following two cases:

- $A[\gamma] \Rightarrow \Upsilon_1\Upsilon_2 \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p\ B[\,]\ a_{q+1} \ldots a_j\ \Upsilon_2$ if and only if $(B,p,q) \neq (-,-,-)$, $B[\,]$ is a dependent descendant of $A[\gamma]$ and $(p,q) \leq (i,j)$.
- $\Upsilon_1 \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j$ if and only if $\gamma = -$ and $(B,p,q) = (-,-,-)$. If the dependent child is in Υ_1 then the indices stack associated to $\boldsymbol{A}$ and to the dependent child must be empty.

The set of deduction steps of $\mathbb{P}_{\text{buE}}$ is obtained by refining the steps in $\mathbb{P}_{\text{CYK}}$: steps $\mathcal{D}_{\text{CYK}}^{[\circ\circ\gamma][\,][\circ\circ]}$, $\mathcal{D}_{\text{CYK}}^{[\circ\circ\gamma][\circ\circ][\,]}$, $\mathcal{D}_{\text{CYK}}^{[\circ\circ][\,][\circ\circ]}$, $\mathcal{D}_{\text{CYK}}^{[\circ\circ][\circ\circ][\,]}$, $\mathcal{D}_{\text{CYK}}^{[\circ\circ][\,][\circ\circ\gamma]}$ and $\mathcal{D}_{\text{CYK}}^{[\circ\circ][\circ\circ\gamma][\,]}$ are separated into different steps Init and Comp. Finally, the domain is extended to deal with linear indexed grammars having productions of arbitrary length.

Schema 2 *The parsing system $\mathbb{P}_{\text{buE}}$ corresponding to the bottom-up Earley-like parsing algorithm for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \ldots a_n$ is defined as follows:*

$$\mathcal{I}_{\text{buE}} = \left\{ [\boldsymbol{A} \rightarrow \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B, p, q] \mid \begin{array}{l} \boldsymbol{A} \rightarrow \Upsilon_1\Upsilon_2 \in P,\ B \in V_N,\ \gamma \in V_I, \\ 0 \leq i \leq j\ ,\ (p,q) \leq (i,j) \end{array} \right\}$$

$$\mathcal{D}_{\text{buE}}^{\text{Init}} = \frac{}{[\boldsymbol{A} \rightarrow \bullet\Upsilon, -, i, i \mid -, -, -]}$$

$$\mathcal{D}_{\text{buE}}^{\text{Scan}} = \frac{[A[\,] \rightarrow \bullet a, -, j, j \mid -, -, -],\ [a, j, j+1]}{[A[\,] \rightarrow a\bullet, -, j, j+1 \mid -, -, -]}$$

$$\mathcal{D}_{\text{buE}}^{\text{Comp}[\,]} = \frac{[\boldsymbol{A} \to \Upsilon_1 \bullet B[\,]\,\Upsilon_2, \gamma, i, k \mid C, p, q],\ [\boldsymbol{B} \to \Upsilon_3 \bullet, -, k, j \mid -, -, -]}{[\boldsymbol{A} \to \Upsilon_1\, B[\,] \bullet \Upsilon_2, \gamma, i, j \mid C, p, q]}$$

$$\mathcal{D}_{\text{buE}}^{\text{Comp}[\circ\circ\gamma][\circ\circ]} = \frac{[A[\circ\circ\gamma] \to \Upsilon_1 \bullet B[\circ\circ]\,\Upsilon_2, -, i, k \mid -, -, -],\ [\boldsymbol{B} \to \Upsilon_3 \bullet, \eta, k, j \mid C, p, q]}{[A[\circ\circ\gamma] \to \Upsilon_1\, B[\circ\circ] \bullet \Upsilon_2, \gamma, i, j \mid B, k, j]}$$

$$\mathcal{D}_{\text{buE}}^{\text{Comp}[\circ\circ][\circ\circ]} = \frac{[A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ]\,\Upsilon_2, -, i, k \mid -, -, -],\ [\boldsymbol{B} \to \Upsilon_3 \bullet, \eta, k, j \mid C, p, q]}{[A[\circ\circ] \to \Upsilon_1\, B[\circ\circ] \bullet \Upsilon_2, \eta, i, j \mid C, p, q]}$$

$$\mathcal{D}_{\text{buE}}^{\text{Comp}[\circ\circ][\circ\circ\gamma]} = \frac{[A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ\gamma]\,\Upsilon_2\,, -, i, k \mid -, -, -],\ [\boldsymbol{B} \to \Upsilon_3 \bullet\,, \gamma, k, j \mid C, p, q],\ [\boldsymbol{C} \to \Upsilon_4 \bullet, \eta, p, q \mid D, r, s]}{[A[\circ\circ] \to \Upsilon_1\, B[\circ\circ\gamma] \bullet \Upsilon_2, \eta, i, j \mid D, r, s]}$$

$$\mathcal{D}_{\text{buE}} = \mathcal{D}_{\text{buE}}^{\text{Init}} \cup \mathcal{D}_{\text{buE}}^{\text{Scan}} \cup \mathcal{D}_{\text{buE}}^{\text{Comp}[\,]} \cup \mathcal{D}_{\text{buE}}^{\text{Comp}[\circ\circ\gamma][\circ\circ]} \cup \mathcal{D}_{\text{buE}}^{\text{Comp}[\circ\circ][\circ\circ]} \cup \mathcal{D}_{\text{buE}}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}$$

$$\mathcal{F}_{\text{buE}} = \{\ [\boldsymbol{S} \to \Upsilon \bullet\,, -, 0, n \mid -, -, -]\ \}$$

The space complexity with respect to the input string is $\mathcal{O}(n^4)$ because each item stores four positions. The time complexity with respect to the input string is $\mathcal{O}(n^6)$ and it is given by deduction steps in $\mathcal{D}_{\text{buE}}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}$.

4 An Earley-like Algorithm

The algorithm described by the parsing system $\mathbb{P}_{\text{buE}}$ does not take into account whether the part of the input string recognized by each grammar element can be derived from S, the axiom of the grammar. Earley-like algorithms limit the number of deduction steps that can be applied in each moment by predicting productions which are candidates to be part of a derivation having as its starting point the axiom of the grammar.

As a first approach, we consider prediction is performed taking into account the context-free skeleton only. The parsing system so obtained is denoted $\mathbb{P}_{\text{E}}$ and can be derived from $\mathbb{P}_{\text{buE}}$ by applying the following filters:

- Init deduction steps only consider productions with S as left-hand side.
- Instead of generating items of the form $[\boldsymbol{A} \to \bullet\Upsilon, -, i, i \mid -, -, -]$ for each possible production $\boldsymbol{A} \to \Upsilon \in P$ and positions i and j, a set of Pred deduction steps generate only those items involving productions with a relevant context-free skeleton.

Schema 3 *The parsing system $\mathbb{P}_{\text{E}}$ corresponding to the Earley-like parsing algorithm for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \ldots a_n$ is defined as follows:*

$$\mathcal{I}_{\text{E}} = \mathcal{I}_{\text{buE}}$$

$$\mathcal{D}_{\mathrm{E}}^{\mathrm{Init}} = \frac{}{[\boldsymbol{S} \rightarrow \bullet \Upsilon, -, 0, 0 \mid -, -, -]}$$

$$\mathcal{D}_{\mathrm{E}}^{\mathrm{Pred}} = \frac{[\boldsymbol{A} \rightarrow \Upsilon_1 \bullet \boldsymbol{B}\ \Upsilon_2, \gamma, i, j \mid C, p, q]}{[\boldsymbol{B} \rightarrow \bullet \Upsilon_3, -, j, j \mid -, -, -]}$$

$$\mathcal{D}_{\mathrm{E}} = \mathcal{D}_{\mathrm{E}}^{\mathrm{Init}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Scan}} \cup \mathcal{D}_{\mathrm{E}}^{\mathrm{Pred}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp}[\,]} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp}[\circ\circ\gamma][\circ\circ]} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp}[\circ\circ][\circ\circ]} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp}[\circ\circ][\circ\circ\gamma]}$$

$$\mathcal{F}_{\mathrm{E}} = \mathcal{F}_{\mathrm{buE}}$$

This algorithm, which has a space complexity $\mathcal{O}(n^4)$ and a time complexity $\mathcal{O}(n^6)$, is very close to the Earley-like algorithm described by Schabes and Shieber in [13] although the latter can only be applied to a specific class of linear indexed grammars obtained from tree adjoining grammars. However, both algorithms share an important feature: they are weakly predictive as they do not consider the contents of the indices stacks when predictive steps are applied. In appearance, the algorithm proposed by Schabes and Shieber in [13] consults the element on the top of the indices stack at prediction, but a deeper study of the behavior of the algorithm makes it clear that this is not really true, as the authors store the context-free skeleton of the elementary trees of a TAG into the indices stacks, reducing the non-terminal set of the resulting LIG to $\{t, b\}$. Indeed, a production $b[\circ\circ\eta] \rightarrow t[\eta_1] \ldots t[\circ\circ\eta_s] \ldots t[\eta_n]$ is equivalent to $\eta^b[\circ\circ] \rightarrow \eta_1^t[\,] \ldots \eta_s^t[\circ\circ] \ldots \eta_n^t[\,]$ and a production $b[\circ\circ\eta] \rightarrow t[\eta_1] \ldots t[\eta_n]$ is equivalent to $\eta^b[\,] \rightarrow \eta_1^t[\,] \ldots \eta_n^t[\,]$.

5 An Earley-like Algorithm Preserving the VPP

Parsers satisfying the *valid prefix property* (VPP) guarantee that, as they read the input string from left to right, the substrings read so far are valid prefixes of the language defined by the grammar. More formally, a parser satisfies the valid prefix property if, for any substring $a_1 \ldots a_k$ read from the input string $a_1 \ldots a_k a_{k+1} \ldots a_n$, it guarantees that there is a string of tokens $b_1 \ldots b_m$, where b_i need not be part of the input string, such that $a_1 \ldots a_k b_1 \ldots b_m$ is a valid string of the language.

In the case of LIG, preserving the valid prefix property requires checking if each predicted production $A[\circ\circ\gamma] \rightarrow \bullet\Upsilon$ satisfies $S[\,] \stackrel{*}{\Rightarrow} w\ A[\alpha\gamma]\ \Upsilon$ where $\gamma \in V_I \cup \{\epsilon\}$. Therefore, to obtain an Earley-like parsing algorithm for LIG preserving this property we need to modify the Pred steps of the parsing system $\mathbb{P}_{\mathrm{E}}$ in order to predict information about the indices stacks. As a consequence, items must be also modified, introducing a new element that allows us to track the contents of the predicted indices stacks. The items are now of the form $[E, h \mid \boldsymbol{A} \rightarrow \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B, p, q]$ and they represent one of the following types of derivation:

- $S[\quad] \stackrel{*}{\Rightarrow} a_1 \ldots a_h\, E[\alpha]\, \Upsilon_4 \stackrel{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i\, A[\alpha\gamma]\, \Upsilon_3 \Upsilon_4 \stackrel{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i \ldots a_p\, B[\alpha]\, a_{q+1} \ldots a_j\, \Upsilon_2 \Upsilon_3 \Upsilon_4$ if and only if $(B, p, q) \neq (-, -, -)$, $A[\alpha\gamma]$ is a dependent descendent of $E[\alpha]$ and $B[\alpha]$ is a dependent descendent of $A[\alpha\gamma]$. This type of derivation corresponds to the completer of the dependent child of a production having the non-terminal A as left-hand side. The indices stack associated to that non-terminal is not empty.

- $S[\,] \overset{*}{\Rightarrow} a_1 \ldots a_h\, E[\alpha]\, \Upsilon_4 \overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i\, A[\alpha\gamma]\, \Upsilon_3 \Upsilon_4 \overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i \ldots a_j\, \Upsilon_2 \Upsilon_3 \Upsilon_4$ if and only if $(E,h) \neq (-,-)$, $(B,p,q) = (-,-,-)$, $A[\alpha\gamma]$ is a dependent descendant of $E[\alpha]$, Υ_1 does not contain the descendent child of $A[\alpha\gamma]$ and $(p,q) \leq (i,j)$. This type of derivation refers to a prediction of the non-terminal A with a non-empty indices stack.

- $S[\,] \overset{*}{\Rightarrow} a_1 \ldots a_i A[\,]\Upsilon_4 \overset{*}{\Rightarrow} a_1 \ldots a_i \ldots a_j\, \Upsilon_2 \Upsilon_4$ if and only if $(E,h) = (-,-)$, $\gamma = -$ and $(B,p,q) = (-,-,-)$. If Υ_1 includes the dependent child of $A[\,]$ then the indices stack associated to that dependent child is empty. This type of derivation refers to the prediction or completer of a non-terminal A with an empty indices stack.

The new set of items so defined is a refinement of the items in the parsing system $\mathbb{P}_{\mathrm{E}}$: the element γ is used to store the top of the predicted indices stacks (in the parsing system $\mathbb{P}_{\mathrm{E}}$, $\gamma = -$ for items resulting of a prediction) and the pair (E,h) allows us to track the item involved in the prediction.

With respect to the deduction steps, the completer steps must be adapted to the new form of the items in order to manipulate the new components E and h and the predicted steps must be refined taking into account the different types of productions.

Schema 4 *The parsing system $\mathbb{P}_{\mathrm{Earley_1}}$ corresponding to the Earley-like parsing algorithm preserving the valid-prefix property for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \ldots a_n$ is defined as follows:*

$$\mathcal{I}_{\mathrm{Earley_1}} = \left\{ [E,h \mid \boldsymbol{A} \rightarrow \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B,p,q] \mid \begin{array}{l} \boldsymbol{A} \rightarrow \Upsilon_1 \Upsilon_2 \in P,\ B,C \in V_N,\ \gamma \in V_I, \\ 0 \leq h \leq i \leq j\ ,\ \ (p,q) \leq (i,j) \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{Earley_1}} = \frac{}{[-,- \mid \boldsymbol{S} \rightarrow \bullet \Upsilon, -, 0, 0 \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Scan}}_{\mathrm{Earley_1}} = \frac{[-,- \mid A[\,] \rightarrow \bullet a, -, j, j \mid -,-,-],\ [a,j,j+1]}{[-,- \mid A[\,] \rightarrow a \bullet, -, j, j+1 \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Pred}[\,]}_{\mathrm{Earley_1}} = \frac{[E,h \mid \boldsymbol{A} \rightarrow \Upsilon_1 \bullet B[\,]\, \Upsilon_2, \gamma, i, j \mid C,p,q]}{[-,- \mid \boldsymbol{B} \rightarrow \bullet \Upsilon_3, -, j, j \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Pred}[\circ\circ\gamma][\circ\circ]}_{\mathrm{Earley_1}} = \frac{[E,h \mid A[\circ\circ\gamma] \rightarrow \Upsilon_1 \bullet B[\circ\circ]\, \Upsilon_2, \gamma, i, j \mid -,-,-],\ [M,m \mid \boldsymbol{E} \rightarrow \bullet \Upsilon_3, \gamma', h, h \mid -,-,-]}{[M,m \mid \boldsymbol{B} \rightarrow \bullet \Upsilon_4, \gamma', j, j \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Pred}[\circ\circ][\circ\circ]}_{\mathrm{Earley_1}} = \frac{[E,h \mid A[\circ\circ] \rightarrow \Upsilon_1 \bullet B[\circ\circ]\, \Upsilon_2, \gamma, i, j \mid -,-,-]}{[E,h \mid \boldsymbol{B} \rightarrow \bullet \Upsilon_3, \gamma, j, j \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Pred}[\circ\circ][\circ\circ\gamma]}_{\mathrm{Earley_1}} = \frac{[E,h \mid A[\circ\circ] \rightarrow \Upsilon_1 \bullet B[\circ\circ\gamma]\, \Upsilon_2, \gamma', i, j \mid -,-,-]}{[A,i \mid \boldsymbol{B} \rightarrow \bullet \Upsilon_3, \gamma, j, j \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Comp}[\,]}_{\mathrm{Earley_1}} = \frac{[E,h \mid \boldsymbol{A} \rightarrow \Upsilon_1 \bullet B[\,]\, \Upsilon_2, \gamma, i, j \mid C,p,q],\ [-,- \mid \boldsymbol{B} \rightarrow \Upsilon_3 \bullet, -, j, k \mid -,-,-]}{[E,h \mid \boldsymbol{A} \rightarrow \Upsilon_1\, B[\,] \bullet \Upsilon_2, \gamma, i, k \mid C,p,q]}$$

$$\mathcal{D}^{\text{Comp}[\circ\circ\gamma][\circ\circ]}_{\text{Earley}_1} = \frac{\begin{array}{l}[E,h \mid A[\circ\circ\gamma] \rightarrow \Upsilon_1 \bullet B[\circ\circ]\ \Upsilon_2, \gamma, i, j \mid -,-,-],\\ [M,m \mid \boldsymbol{B} \rightarrow \Upsilon_4 \bullet, \gamma', j, k \mid C,p,q]\end{array}}{[E,h \mid A[\circ\circ\gamma] \rightarrow \Upsilon_1\ B[\circ\circ] \bullet \Upsilon_2, \gamma, i, k \mid B,j,k]}$$

$$\mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ]}_{\text{Earley}_1} = \frac{\begin{array}{l}[E,h \mid A[\circ\circ] \rightarrow \Upsilon_1 \bullet B[\circ\circ]\ \Upsilon_2, \gamma, i, j \mid -,-,-],\\ [E,h \mid \boldsymbol{B} \rightarrow \Upsilon_3 \bullet, \gamma, j, k \mid C,p,q]\end{array}}{[E,h \mid A[\circ\circ] \rightarrow \Upsilon_1\ B[\circ\circ] \bullet \Upsilon_2, \gamma, i, k \mid C,p,q]}$$

$$\mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}_{\text{Earley}_1} = \frac{\begin{array}{l}[E,h \mid A[\circ\circ] \rightarrow \Upsilon_1 \bullet B[\circ\circ\gamma]\ \Upsilon_2\ , \gamma', i, j \mid -,-,-],\\ [A,i \mid \boldsymbol{B} \rightarrow \Upsilon_3 \bullet\ , \gamma, j, k \mid C,p,q],\\ [E,h \mid \boldsymbol{C} \rightarrow \Upsilon_4 \bullet, \gamma', p, q \mid D,r,s]\end{array}}{[E,h \mid A[\circ\circ] \rightarrow \Upsilon_1\ B[\circ\circ\gamma] \bullet \Upsilon_2, \gamma', i, k \mid D,r,s]}$$

$$\begin{aligned}\mathcal{D}_{\text{Earley}_1} = \ & \mathcal{D}^{\text{Init}}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Scan}}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\,]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ\gamma][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ][\circ\circ\gamma]}_{\text{Earley}_1} \cup \\ & \mathcal{D}^{\text{Comp}[\,]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ\gamma][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}_{\text{Earley}_1}\end{aligned}$$

$$\mathcal{F}_{\text{Earley}_1} = \{\ [-,- \mid \boldsymbol{S} \rightarrow \Upsilon \bullet\ , -, 0, n \mid -,-,-]\ \}$$

The space complexity of the algorithm with respect to the length n of the input string is $\mathcal{O}(n^5)$, due to the five positions of the input string stored in each item. The time complexity is $\mathcal{O}(n^7)$ due to deduction steps in the set $\mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}_{\text{Earley}_1}$. To reduce the time complexity we will use a technique similar to that used in [5, 2] to reduce the complexity of the tabular interpretations of automata for tree adjoining languages. In this case, we split each deduction step in $\mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}_{\text{Earley}_1}$ into two different steps such that their final complexity is at most $\mathcal{O}(n^6)$. The resulting parsing schema is defined by the following parsing system.

Schema 5 *The parsing system $\mathbb{P}_{\text{Earley}}$ corresponding to the Earley-like parsing algorithm preserving the valid-prefix property working with a time complexity $\mathcal{O}(n^6)$ for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \ldots a_n$ is defined as follows:*

$$\mathcal{I}_{\text{Earley}^{(1)}} = \left\{ [E,h \mid \boldsymbol{A} \rightarrow \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B,p,q] \mid \begin{array}{l}\boldsymbol{A} \rightarrow \Upsilon_1\Upsilon_2 \in P,\ B,C \in V_N,\ \gamma \in V_I,\\ 0 \leq h \leq i \leq j\ ,\ \ (p,q) \leq (i,j)\end{array} \right\}$$

$$\mathcal{I}_{\text{Earley}^{(2)}} = \left\{ [[\boldsymbol{A} \rightarrow \Upsilon \bullet, \gamma, i, j \mid B,p,q]] \mid \begin{array}{l}\boldsymbol{A} \rightarrow \Upsilon \in P,\ B \in V_N,\\ \gamma \in V_I,\ i \leq j\ ,\ \ (p,q) \leq (i,j)\end{array} \right\}$$

$$\mathcal{I}_{\text{Earley}} = \mathcal{I}_{\text{Earley}^{(1)}} \cup \mathcal{I}_{\text{Earley}^{(2)}}$$

$$\mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]^0}_{\text{Earley}} = \frac{\begin{array}{l}[A,i \mid \boldsymbol{B} \rightarrow \Upsilon_3 \bullet\ , \gamma, j, k \mid C,p,q],\\ [E,h \mid \boldsymbol{C} \rightarrow \Upsilon_4 \bullet, \gamma', p, q \mid D,r,s]\end{array}}{[[\boldsymbol{B} \rightarrow \Upsilon_3 \bullet\ , \gamma, j, k \mid D,r,s]]}$$

$$\mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]^1}_{\text{Earley}} = \frac{\begin{array}{l}[[\boldsymbol{B} \rightarrow \Upsilon_3 \bullet\ , \gamma, j, k \mid D,r,s]],\\ [E,h \mid A[\circ\circ] \rightarrow \Upsilon_1 \bullet B[\circ\circ\gamma]\ \Upsilon_2\ , \gamma', i, j \mid -,-,-],\\ [E,h \mid \boldsymbol{C} \rightarrow \Upsilon_4 \bullet, \gamma', p, q \mid D,r,s]\end{array}}{[E,h \mid A[\circ\circ] \rightarrow \Upsilon_1\ B[\circ\circ\gamma] \bullet \Upsilon_2, \gamma', i, k \mid D,r,s]}$$

$$\mathcal{D}_{\text{Earley}} = \mathcal{D}^{\text{Init}}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Scan}}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\,]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ\gamma][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ][\circ\circ\gamma]}_{\text{Earley}_1} \cup$$
$$\mathcal{D}^{\text{Comp}[\,]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ\gamma][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]^0}_{\text{Earley}} \cup \mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]^1}_{\text{Earley}}$$

$$\mathcal{F}_{\text{Earley}} = \mathcal{F}_{\text{Earley}_1}$$

Deduction steps Comp$[\circ\circ][\circ\circ\gamma]^0$ generate an intermediate item of the form $[[\boldsymbol{B} \rightarrow \Upsilon_3\bullet\ , \gamma, j, k \mid D, r, s]]$ that will be taken as antecedent for steps Comp$[\circ\circ][\circ\circ\gamma]^1$ and that represents a derivation

$$B[\gamma'\gamma] \overset{*}{\Rightarrow} a_{j+1} \ldots a_p\, C[\gamma']\, a_{s+1} \ldots a_q \overset{*}{\Rightarrow} a_{j+1} \ldots a_p \ldots a_r\, D[\,]\, a_{s+1} \ldots a_q \ldots a_k$$

for some γ', p and q. Deduction steps Comp$[\circ\circ][\circ\circ\gamma]^1$ combine this pseudo-item with an item $[E, h \mid A[\circ\circ] \rightarrow \Upsilon_1 \bullet B[\circ\circ\gamma]\ \Upsilon_2\ , \gamma', i, j \mid -, -, -]$ that represents a derivation

$$S[\,] \overset{*}{\Rightarrow} a_1 \ldots a_h\, E[\alpha]\, \Upsilon_5 \overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i\, A[\alpha\gamma']\, \Upsilon_3\Upsilon_5 \overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i \ldots a_j\, B[\alpha\gamma'\gamma]\, \Upsilon_2\Upsilon_3\Upsilon_5$$

and with an item $[E, h \mid \boldsymbol{C} \rightarrow \Upsilon_4\bullet, \gamma', p, q \mid D, r, s]$ representing a derivation

$$S[\,] \overset{*}{\Rightarrow} a_1 \ldots a_h\, E[\alpha]\, \Upsilon_5 \overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_p\, C[\alpha\gamma']\, \Upsilon_4\Upsilon_5 \overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_p \ldots a_r\, D[\alpha]\, a_{s+1} \ldots a_q\, \Upsilon_4\Upsilon_5$$

and a new item of the form $[E, h \mid A[\circ\circ] \rightarrow \Upsilon_1\ B[\circ\circ\gamma] \bullet \Upsilon_2, \gamma', i, k \mid D, r, s]$ is generated. This last item represent the existence of a derivation

$$\begin{aligned}
S[\,] &\overset{*}{\Rightarrow} a_1 \ldots a_h\, E[\alpha]\, \Upsilon_5 \\
&\overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i\, A[\alpha\gamma']\, \Upsilon_3\Upsilon_5 \\
&\overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i \ldots a_j\, B[\alpha\gamma'\gamma]\, \Upsilon_2\Upsilon_3\Upsilon_5 \\
&\overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i \ldots a_j \ldots a_p\, C[\alpha\gamma']\, a_{q+1} \ldots a_k\, \Upsilon_2\Upsilon_3\Upsilon_5 \\
&\overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i \ldots a_j \ldots a_p \ldots a_r\, D[\alpha]\, a_{s+1} \ldots a_{q+1} \ldots a_k\, \Upsilon_2\Upsilon_3\Upsilon_5
\end{aligned}$$

6 The Shared Forest

The algorithms described so far are just recognizers. They do not build a representation of the derived trees. However, we can modify them to build these trees as a shared forest satisfying the following properties: it must store in a compact form all parse trees and it must be possible to retrieve every individual parse tree in linear time with respect to the size of the forest.

Billot and Lang [3] define the shared forest for a context-free grammar $\mathcal{G} = (V_T, V_N, P, S)$ and an input string $a_1 \ldots a_n$ as a context free grammar in which the non-terminals are of the form $\langle A, i, j\rangle$, where $A \in V_N$ and $i, j \in 0..n$, and productions are of the form

$$\langle A_0, j_0, j_m\rangle \rightarrow w_0\ \langle A_1, j_0, j_1\rangle\ w_1\ \langle A_2, j_1, j_2\rangle\ \ldots\ w_{m-1}\ \langle A_m, j_{m-1}, j_m\rangle\ w_m$$

where $A_0 \rightarrow w_0A_1w_1A_2 \ldots w_{m-1}A_mw_m \in P$ and $w_i \in V_T^*$, meaning that A_0 recognizes the part $a_{j_0+1} \ldots a_{j_m}$ of the input string by applying the production $A_0 \rightarrow w_0A_1w_1A_2 \ldots w_{m-1}A_mw_m$ such that A_i recognizes the part $a_{j_{i-1}+1} \ldots a_{j_i}$ of the input string.

We can extend the concept of shared forest for CFG to define the concept of *LIGed forest* [18]. Given the shared forest of the context-free skeleton of a LIG, when a LIG production $A_0[\circ\circ\gamma] \rightarrow A_1[\,] \ldots A_d[\circ\circ\gamma'] \ldots A_m[\,]$ is involved in a derivation, a production

$$\langle A_0, j_0, j_m\rangle[\circ\circ\gamma] \rightarrow \langle A_1, j_0, j_1\rangle\ \ldots\ \langle A_d, j_{d-1}, j_d\rangle[\circ\circ\gamma']\ \ldots\ \langle A_m, j_{m-1}, j_m\rangle$$

is added to the LIGed forest meaning that A_0 recognizes the part $a_{j_0+1} \ldots a_{j_m}$ of the input string by applying the production $A_0[\circ\circ\gamma] \rightarrow A_1 \ldots A_d[\circ\circ\gamma'] \ldots A_m$ such that A_i recognizes the part $a_{j_{i-1}+1} \ldots a_{j_i}$ of the input string and the indices stack is passed from A_0 to A_d replacing the top index γ for γ'. The LIG so generated does not satisfy our definition of shared forest because single parse trees can not be extracted in linear time. Vijay-Shanker and Weir [18] try to solve this problem by defining a non-deterministic finite state automaton that determines if a given LIGed forest symbol $\langle A, i, j\rangle[\alpha]$ derives a string of terminals. A similar finite-state automata is also defined by Nederhof in [11].

As an alternative approach, Boullier [4] defines the shared forest for a LIG $\mathcal{G} = (V_T, V_N, V_I, P, S)$ and an input string w by means of a *linear derivation grammar*, a context-free grammar recognizing the language defined by the sequences of LIG productions of $\mathcal{G}$ that could be used to derive w. Previously to the construction of the linear derivation grammar, we must compute the transitive closure for a set of relations on $V_N \times V_N$.

To avoid the use of additional data structures, such as finite automata or precomputed relations, we have been inspired by the use of context-free grammars to represent the parse forest of tree adjoining grammars [18] in order to capture the context-freeness of production application in the case of LIG. Given a linear indexed grammar $\mathcal{G} = (V_T, V_N, V_I, P, S)$ and an input string $w = a_1 \ldots a_n$, the shared forest for $\mathcal{G}$ and w is a context-free grammar $\mathcal{G}^w = (V_T, V_N^w, P^w, S^w)$. Elements in V_N^w have the form $\langle A, \gamma, i, j, B, p, q\rangle$, where $A, B \in V_N$, $\gamma \in V_I$ and $i, j, p, q \in 0 \ldots n$. The axiom S^w is the non-terminal $\langle S, -, 0, n, -, -, -, \rangle$. Productions in P^w are of the form:

Case 1a: If $A[\,] \Rightarrow a_j$ then add the production $\langle A, -, j-1, j, -, -, -\rangle \rightarrow a_j$

Case 1b: If $A[\,] \Rightarrow \epsilon$ then add the production $\langle A, -, j, j, -, -, -\rangle \rightarrow \epsilon$

Case 2a: If $A[\circ\circ\gamma] \rightarrow B[\,]\ C[\circ\circ] \in P$, $B[\,] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_k$ and $C[\alpha\eta] \stackrel{*}{\Rightarrow} a_{k+1} \ldots a_p\, D[\alpha]\, a_{q+1} \ldots a_j$ then add the production $\langle A, \gamma, i, j, C, k, j\rangle \rightarrow \langle B, -, i, k, -, -, -\rangle\ \langle C, \eta, k, j, D, p, q\rangle$

Case 2b: If $A[\circ\circ\gamma] \rightarrow B[\circ\circ]\ C[\,] \in P$, $B[\alpha\eta] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p\, D[\alpha]\, a_{q+1} \ldots a_k$ and $C[\,] \stackrel{*}{\Rightarrow} a_{k+1} \ldots a_j$ then add the production $\langle A, \gamma, i, j, B, i, k\rangle \rightarrow \langle B, \eta, i, k, D, p, q\rangle\ \langle C, -, k, j, -, -, -\rangle$

Case 3a: If $A[\circ\circ] \rightarrow B[\,]\ C[\circ\circ] \in P$, $B[\,] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_k$ and $C[\alpha\eta] \stackrel{*}{\Rightarrow} a_{k+1} \ldots a_p\, D[\alpha]\, a_{q+1} \ldots a_j$ then add the production $\langle A, \eta, i, j, D, p, q\rangle \rightarrow \langle B, -, i, k, -, -, -\rangle\ \langle C, \eta, k, j, D, p, q\rangle$

Case 3b: If $A[\circ\circ] \rightarrow B[\circ\circ]\ C[\,] \in P$, $B[\alpha\eta] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p\, D[\alpha]\, a_{q+1} \ldots a_k$ and $C[\,] \stackrel{*}{\Rightarrow} a_{k+1} \ldots a_j$ then add the production $\langle A, \eta, i, j, D, p, q\rangle \rightarrow \langle B, \eta, i, k, D, p, q\rangle\ \langle C, -, k, j, -, -, -\rangle$

Case 4a: If $A[\circ\circ] \rightarrow B[\,]\ C[\circ\circ\gamma] \in P$, $B[\,] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_k$ and $C[\alpha\eta\gamma] \stackrel{*}{\Rightarrow} a_{k+1} \ldots a_p\, D[\alpha\eta]\, a_{q+1} \ldots a_j$ and $D[\alpha\eta] \stackrel{*}{\Rightarrow} a_{p+1} \ldots a_r\, E[\alpha]\, a_{s+1} \ldots a_q$ then add the production $\langle A, \eta, i, j, E, r, s\rangle \rightarrow \langle B, -, i, k, -, -, -\rangle\ \langle C, \gamma, k, j, D, p, q\rangle\ \langle D, \eta, p, q, E, r, s\rangle$

Case 4b: If $A[\circ\circ] \rightarrow B[\circ\circ\gamma]\ C[\,] \in P$, $B[\alpha\eta\gamma] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p\, D[\alpha\eta]\, a_{q+1} \ldots a_k$ and $C[\,] \stackrel{*}{\Rightarrow} a_{k+1} \ldots a_j$ and $D[\alpha\eta] \stackrel{*}{\Rightarrow} a_{p+1} \ldots a_r\, E[\alpha]\, a_{s+1} \ldots a_q$ then add the production $\langle A, \eta, i, j, E, r, s\rangle \rightarrow \langle B, \gamma, i, k, D, p, q\rangle\ \langle C, -, k, j, -, -, -\rangle\ \langle D, \eta, p, q, E, r, s\rangle$

Case 5: If $A[\circ\circ\gamma] \rightarrow B[\circ\circ] \in P$ and $B[\alpha\eta] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p\, D[\alpha]\, a_{q+1} \ldots a_j$ then add the production $\langle A, \gamma, i, j, B, i, j\rangle \rightarrow \langle B, \eta, i, j, D, p, q\rangle$

Case 6: If $A[\circ\circ] \rightarrow B[\circ\circ] \in P$ and $B[\alpha\gamma] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p\, D[\alpha]\, a_{q+1} \ldots a_j$ then add the production $\langle A, \gamma, i, j, D, p, q\rangle \rightarrow \langle B, \gamma, i, j, D, p, q\rangle$

Case 7: If $A[\circ\circ] \rightarrow B[\circ\circ\gamma] \in P$ and $B[\alpha\eta\gamma] \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_p\, D[\alpha\eta]\, a_{q+1} \ldots a_j$ and $D[\alpha\eta] \stackrel{*}{\Rightarrow} a_{p+1} \ldots a_r\, E[\alpha]\, a_{s+1} \ldots a_q$ then add the production $\langle A,\eta,i,j,E,r,s\rangle \rightarrow \langle B,\gamma,i,j,D,p,q\rangle\ \langle D,\eta,p,q,E,r,s\rangle$

In cases 4a, 4b and 7, derivations starting at $\langle D,\eta,p,q,E,r,s\rangle$ allow us to retrieve the rest of the indices stack corresponding to A. Note that we are assuming a grammar with productions having at most two children. Any production $A_0[\circ\circ\gamma] \rightarrow A_1[\,] \ldots A_d[\circ\circ\gamma'] \ldots A_m[\,]$ can be translated into

$$
\begin{array}{ll}
\nabla_0[\,] \rightarrow \epsilon & \nabla_{d+1}[\circ\circ] \rightarrow \nabla_d[\circ\circ]\ A_{d+1}[\,] \\
\nabla_1[\circ\circ] \rightarrow \nabla_0[\circ\circ]\ A_1[\,] & \vdots \\
\vdots & \nabla_m[\circ\circ] \rightarrow \nabla_{m-1}[\circ\circ]\ A_m[\,] \\
\nabla_{d-1}[\circ\circ] \rightarrow \nabla_{d-2}[\circ\circ]\ A_{d-1}[\,] & A_0[\circ\circ] \rightarrow \nabla_m[\circ\circ] \\
\nabla_d[\circ\circ\gamma] \rightarrow \nabla_{d-1}[\,]\ A_d[\circ\circ\gamma'] &
\end{array}
$$

where the ∇_i are fresh symbols that represent partial recognition of the original production. In fact, a ∇_i symbol is equivalent to a dotted production with the dot just before the non-terminal A_{i+1} or with the dot at the end of the right-hand side in the case of ∇_m.

It is interesting to remark that the set of non-terminals is a subset of the set of items for CYK-like and bottom-up Earley-like algorithms, and Earley-like algorithms without the VPP. The case of the Earley-like algorithm preserving the valid prefix property is slightly different, as a non-terminal $\langle A,\gamma,i,j,B,p,q\rangle$ represent the class of items $[E,h \mid A,\gamma,i,j \mid D,p,q]$ for any value of E and h.

Like context-free grammars used as shared forest in the case of TAG [18], the derivations in $\mathcal{G}^w$ encode derivations of the string w by $\mathcal{G}$ but the specific set of terminal strings that is generated by $\mathcal{G}^w$ is not important. We do however have the language generated by $\mathcal{G}^w$ is not empty if and only if w belongs to the language generated by $\mathcal{G}$. We can prune $\mathcal{G}^w$ by retaining only production with useful symbols to guarantee that every non-terminal can derive a terminal string. In this case, derivations of w in the original grammar can be read off by simple reading off of derivations in $\mathcal{G}^w$.

The number of possible productions in $\mathcal{G}^w$ is $\mathcal{O}(n^7)$. The complexity can be reduced to $\mathcal{O}(n^6)$ by transforming productions of the form $A[\circ\circ] \rightarrow B[\,]\ C[\circ\circ\gamma]$ into two productions $A[\circ\circ] \rightarrow B[\,]\ X[\circ\circ]$ and $X[\circ\circ] \rightarrow C[\circ\circ\gamma]$ where X is a fresh non-terminal. A similar transformation must be applied to productions $A[\circ\circ] \rightarrow B[\circ\circ\gamma]\ C[\,]$.

7 Conclusion

We have described a set of algorithms for LIG parsing, creating a continuum which has the CYK-like parsing algorithm by Vijay-Shanker and Weir [16] as its starting point and a new Earley-like algorithm which preserves the valid prefix property as its goal. In the middle, a new bottom-up Earley-like algorithm and a new Earley-like algorithm have been described. The time complexity for all these algorithms with respect to the length of the input string is $\mathcal{O}(n^6)$. Other algorithms could also have been included in the continuum, but for reasons of space we have chosen to show only the algorithms we consider milestones in the development of parsing algorithms for LIG.

Acknowledgements

We would like to thank Pierre Boullier, Patrice Lopez and Mark-Jan Nederhof for fruitful discussions. This work was partially supported by the FEDER of EU (project 1FD97-0047-C04-02) and Xunta de

Galicia (projects PGIDT99XI10502B and XUGA20402B97).

References

[1] Alonso, M. A., D. Cabrero, E. de la Clergerie, and M. Vilares. 1999 Tabular algorithms for TAG parsing. *Proc. of EACL'99*, pages 150–157, Bergen, Norway.

[2] Alonso, M. A., E. de la Clergerie, and D. Cabrero. 1999. Tabulation of automata for tree adjoining languages. *Proc. of MOL-6*, pages 127–141, Orlando, Florida.

[3] Billot, S. and B. Lang. 1989. The structure of shared forest in ambiguous parsing. *Proc. of ACL'89*, pages 143–151, Vancouver, British Columbia, Canada.

[4] Boullier, P. 1996. Another facet of LIG parsing. *Proc. of ACL'96*, Santa Cruz, CA.

[5] De la Clergerie, E. and M. A. Alonso. 1998. A tabular interpretation of a class of 2-Stack Automata. *Proc. of COLING-ACL'98*, volume II, pages 1333–1339, Montreal, Canada.

[6] De la Clergerie, E., M. A. Alonso, and D. Cabrero. 1998. A tabular interpretation of bottom-up automata for TAG. *Proc. of TAG+4*, pages 42–45, Philadelphia.

[7] Gazdar, G. 1987. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel Publishing Company.

[8] Joshi, A. K. and Y. Schabes. Tree-adjoining grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages. Vol 3: Beyond Words*, chapter 2, pages 69–123. Springer-Verlag, Berlin/Heidelberg/New York, 1997.

[9] Nederhof, M-J. 1997 Solving the correct-prefix property for TAGs. In T. Becker and H.-V. Krieger, editors, *Proc. of MOL-5*, pages 124–130, Schloss Dagstuhl, Saarbruecken, Germany.

[10] Nederhof, M-J. 1998. Linear indexed automata and tabulation of TAG parsing. *Proc. of TAPD'98*, pages 1–9, Paris, France.

[11] Nederhof, M-J. 1999. Models of tabulation for TAG parsing. *Proc. of MOL-6*, pages 143–158, Orlando, Florida.

[12] Schabes, Y. 1992. Stochastic lexicalized tree-adjoining grammars. *Proc. of COLING'92*, pages 426–432, Nantes, France.

[13] Schabes, Y. and S. M. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

[14] Shieber, S. M., Y. Schabes, and F. C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1&2):3–36.

[15] Sikkel, K. 1997. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms.* Springer-Verlag, Berlin/Heidelberg/New York.

[16] Vijay-Shanker, K. and D. J. Weir. 1991. Polynomial parsing of extensions of context-free grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, chapter 13, pages 191–206. Kluwer Academic Publishers, Norwell, MA.

[17] Vijay-Shanker, K. and D. J. Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

[18] Vijay-Shanker, K. and D. J. Weir. 1993. The use of shared forest in tree adjoining grammar parsing. *Proc. of EACL'93*, pages 384–393.