BIDIRECTIONAL AUTOMATA FOR TREE ADJOINING GRAMMARS

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

Víctor J. Díaz
Depto. de Lenguajes y Sist. Informáticos
Universidad de Sevilla
Avda. Reina Mercedes s/n
41012 Sevilla (Spain)
vjdiaz@lsi.us.es

Abstract

We define a new model of automata for the description of bidirectional parsing strategies for tree adjoining grammars and a tabulation mechanism that allow them to be executed in polynomial time. This new model of automata provides a modular way of describing bidirectional parsing strategies for TAG, separating the description of a strategy from its execution.

1 Introduction

Bidirectional parsing strategies for Tree Adjoining Grammars (TAG) show a good performance, particularly when applied to linguistically motivated grammars [5]. These strategies are usually implemented as tabular parsers [8, 14, 6], sharing as common characteristics the use of items to represent traces of derivations and the use of inference rules to derive new items from existing ones. Regrettably, for most parsers it is difficult to separate the tabulation strategy from the parsing strategy.

A promising way of simplifying the task of designing correct and efficient parsing algorithms is to apply well-known techniques from the realm of context-free parsing, which allow tabulation to be seen separately from the parsing strategy: the actual parsing strategy can be described by means of the construction of a non-deterministic pushdown automaton, and tabulation is introduced by means of some generic mechanism such as memoization. The construction of parsers in this way allows more straightforward proofs of correctness and makes parsing strategies easier to understand and to implement.

This approach has been successfully applied to the design of parsing algorithms for TAG that read the input string left-to-right [1, 3, 10, 11]. In this paper, we define new models of automata which can start reading the input string in any position, spanning to the left and to the right to include substrings which were themselves read in the same bidirectional way. Tabulation techniques are provided in order to execute efficiently these automata.

This paper is outlined as follows. The rest of the introduction is a presentation of tree adjoining grammars. Section 2 introduces bidirectional push-down automata, showing how they can be used to define bidirectional parsers for context-free grammars. Section 3 extends the model of automata developed in the previous section to define a new model of automata suitable for defining bidirectional parsing strategies for TAG. Section 4 presents final conclusions.

1.1 Tree Adjoining Grammars

A Context-Free Grammar (CFG) is a tuple (V_N, V_T, S, P), where V_N is a finite set of non-terminal symbols, V_T a finite set of terminal symbols, $S \in V_N$ is the axiom of the grammar, and P is a finite set of productions (rewriting rules) of the form $A \to \delta$ with $A \in V_N$ and $\delta \in (V_T \cup V_N)^*$. Tree Adjoining Grammars [7] are an extension of context-free grammars that use trees instead of productions as the primary representing structure. Formally, a TAG is a tuple $(V_N, V_T, S, \boldsymbol{I}, \boldsymbol{A})$, where V_N, V_T and $S \in V_N$ are defined as for CFG, $\boldsymbol{I}$ a finite set of *initial trees* and $\boldsymbol{A}$ is a finite disjoint set of *auxiliary trees*. $\boldsymbol{I} \cup \boldsymbol{A}$ is the set of *elementary trees*. Internal nodes are labeled by non-terminals and leaf nodes by terminals or the empty string ϵ, except for just one leaf per auxiliary tree (the *foot*) which is labeled by the same non-terminal used as the label of its root node. The path in an auxiliary tree from the root node to the foot node is called the *spine* of the tree.

New trees are derived by *adjoining*: let γ be a tree containing a node N^γ labeled by A and let β be an auxiliary tree whose root and foot nodes are also labeled by A. Then, the adjoining of β at the *adjunction node* N^γ is obtained by excising the subtree of γ with root N^γ, attaching β to N^γ and attaching the excised subtree to the foot of β.

The operation of *substitution* does not increase the generative power of the formalism but it is usually considered when we are dealing with lexicalized tree adjoining grammars. In this case, non-terminals can also label leaf nodes (called substitution nodes) of elementary trees. An initial tree can be substituted at a substitution node if its root is labeled by the same non-terminal that labels the substitution node.

2 Bidirectional Push-Down Automata

Push-Down Automata (PDA) are the operational devices for parsing context-free grammars. Following [4], we define a PDA as a tuple $(V_T, V_S, \Theta, \$_0, \$_f)$ where V_T is a finite set of terminal symbols, V_S is a finite set of stack symbols, $\$_0 \in V_S$ is the initial stack symbol, $\$_f \in V_S$ is the final stack symbol and Θ is a finite set of SWAP, PUSH and POP transitions. A configuration of a PDA is usually defined as a pair $(\xi, a_l \dots a_n)$, where $\xi \in V_S^*$ is the stack attained and $a_l \dots a_n$ the part of the input string $a_1 \dots a_n$ to be read. We consider an alternative and equivalent definition in which the position l is stored in the top element of ξ. Thus, a configuration is given by the contents of $\hat{\xi}$, a stack of pairs in $V_S \times \mathbb{N}$.[1] The initial configuration is $(\$_0, 0)$. Other configurations are attained by applying transitions as follows:

- The application of a SWAP transition of the form $C \overset{a}{\longmapsto} F$ to a configuration $\hat{\xi}(C, l)$ yields a configuration $\hat{\xi}(F, l + |a|)$ as a result of replacing C by F and scanning the terminal $a = a_{l+1}$ or the empty string $a = \epsilon$.

- The application of a PUSH transition of the form $C \longmapsto C\,F$ to a configuration $\hat{\xi}(C, l)$ yields a configuration $\hat{\xi}(C, l)(F, l)$ as a result of pushing F onto C.

- The application of a POP transition of the form $CF \longmapsto G$ to a configuration $\hat{\xi}(C, l)(F, m)$ yields a configuration $\hat{\xi}(G, m)$ as a result of popping C and F, which are replaced by G.

[1] This pairs are called *modes* in [2].

where $C, F, G \in V_S$ and $a \in V_T \cup \{\epsilon\}$. An input string $a_1 \dots a_n$ is successfully recognized by a PDA if the final configuration $(\$_0, 0)(\$_f, n)$ is attained. Only SWAP transitions can scan elements from the input string. This is not a limitation, as a scanning push transition $C \stackrel{a}{\longmapsto} C\,F$ could be emulated by the consecutive application of two transitions $C \longmapsto C\,F'$ and $F' \stackrel{a}{\longmapsto} F$, while a scanning pop transition $C\ F \stackrel{a}{\longmapsto} G$ could be emulated by $C\ F \longmapsto G'$ and $G' \stackrel{a}{\longmapsto} G$, where F' and G' are fresh stack symbols.

We call transitions SWAP, PUSH and POP *r-transitions* as they can only read the input string from the left to the right. Thus, push-down automata can only be used to implement unidirectional parsing strategies that read the input string in the same way.

Bidirectional parsing strategies can start computations at any position of the input string and can span to the right and to the left to include substrings which were scanned in a bidirectional way by some subcomputations. As a first step towards the definition of a Bidirectional Push-Down Automata (BPDA), we must adapt configurations in order to be able to represent the discontinuous recognition of the input string. Thus, configurations of a BPDA will be given by the contents of Ξ, a stack of triples in $V_S \times \mathbb{N} \times \mathbb{N}$. The initial configuration is $(\$_0, 0, 0)$. Other configurations are attained by applying transitions as follows:

- The application of a SWAP_R transition of the form $C \stackrel{a}{\longmapsto}_R F$ to a configuration $\Xi(C, k, l)$ yields a configuration $\Xi(F, k, l + |a|)$ as a result of replacing C by F and scanning the terminal $a = a_{l+1}$ or the empty string $a = \epsilon$ to the right of the substring spanned by C.

- The application of a SWAP_L transition of the form $C \stackrel{a}{\longmapsto}_L F$ to a configuration $\Xi(C, k, l)$ yields a configuration $\Xi(F, k - |a|, l)$ as a result of replacing C by F and scanning the terminal $a = a_k$ or the empty string $a = \epsilon$ to the left of the substring spanned by C.

- The application of a PUSH_R transition of the form $C \longmapsto_R C\,F$ to a configuration $\Xi(C, k, j)$ yields a configuration $\Xi(C, k, l)(F, l, l)$. It is expected that F will span a substring inmediatly to the right of the substring spanned by C.

- The application of a PUSH_L transition of the form $C \longmapsto_L C\,F$ to a configuration $\Xi(C, k, j)$ yields a configuration $\Xi(C, k, l)(F, k, k)$. It is expected that F will span a substring inmediatly to the left of the substring spanned by C.

- The application of a PUSH_U transition of the form $C \stackrel{a}{\longmapsto}_U C\,F$ to a configuration $\Xi(C, k, j)$ yields a configuration $\Xi(C, k, l)(F, m, m + |a|)$ as a result of pushing F onto C and scanning the terminal $a = a_{m+1}$ or the empty string $a = \epsilon$. PUSH_U transitions are *undirected* in the sense that the substring spanned by F is not necessarily adjacent to the substring spanned by C.

- The application of a POP_R transition of the form $CF \longmapsto_R G$ to a configuration $\Xi(C, k, l)(F, l, m)$ yields a configuration $\Xi(G, k, m)$. The substring spanned by F is adjacent to the right of the substring spanned by C.

- The application of a POP_L transition of the form $CF \longmapsto_L G$ to a configuration $\Xi(C, k, l)(F, m, k)$ yields a configuration $\Xi(G, m, l)$. The substring spanned by F is adjacent to the left of the substring spanned by C.

[INIT]	$\$_0 \vdash\!\!\rightarrow_R \$_0 \ (S \rightarrow \bullet \bullet \delta)$
[SCAN$_a$]	$(A \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3) \stackrel{a}{\vdash\!\!\rightarrow}_U (A \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3) \ (B \rightarrow \nu \bullet a \bullet \omega)$
[SCAN$_\epsilon$]	$(A \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3) \stackrel{\epsilon}{\vdash\!\!\rightarrow}_U (A \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3) \ (B \rightarrow \bullet\bullet)$
[CONC-R]	$(A \rightarrow \nu \bullet \delta_1 \bullet \delta_2 \omega) \ (A \rightarrow \nu\delta_1 \bullet \delta_2 \bullet \omega) \vdash\!\!\rightarrow_R (A \rightarrow \nu \bullet \delta_1\delta_2 \bullet \omega)$
[CONC-L]	$(A \rightarrow \nu\delta_1 \bullet \delta_2 \bullet \omega) \ (A \rightarrow \nu \bullet \delta_1 \bullet \delta_2 \omega) \vdash\!\!\rightarrow_L (A \rightarrow \nu \bullet \delta_1\delta_2 \bullet \omega)$
[INC]	$(B \rightarrow \bullet\delta\bullet) \stackrel{\epsilon}{\vdash\!\!\rightarrow}_R (A \rightarrow \nu \bullet B \bullet \omega)$

Table 1: Compilation schema for a bottom-up bidirectional strategy

An input string $a_1 \ldots a_n$ is successfully recognized by a PDA if the final configuration $(\$_0, 0, 0)(\$_f, 0, n)$ is attained. SWAP_R, PUSH_R and POP_R transitions are the r-transitions corresponding to unidirectional PDA. SWAP_L, PUSH_L and POP_L transitions are *l-transitions* that advance "to the left" in the reading of the input string. However, the union of r-transitions and l-transitions is not sufficient to implement bidirectional parsers, we need PUSH_U transitions of the form $C \stackrel{a}{\vdash\!\!\rightarrow}_U C\ F$ to start subcomputations at any position of the input string. In any computation recognizing the input string, we guarantee that each terminal in the input string is read only once by means of the definition of SWAP_R and SWAP_L transitions (they can not re-read elements which are in the span of the top element of the stack) and the definition of POP_R and POP_L transitions (they can not pop stack elements spanning overlapping substrings). Thus, BPDA accept exactly the class of context-free languages.

We define a Bidirectional Push-Down Automata (BPDA) as a tuple $(V_T, V_S, \Theta_{\text{BPDA}}, \$_0, \$_f)$ with Θ_{BPDA} containing SWAP_R, SWAP_L, PUSH_R, PUSH_L, POP_R, POP_L and PUSH_U transitions. As an example of the kind of parsers that can be implemented using BPDA, a compilation schema[2] of a context-free grammar into a bidirectional push-down automaton implementing a bottom-up bidirectional parsing strategy is derived. In the resulting automaton, V_T is equal to the set of terminals of the source grammar, V_S is the union of $\{\$_0, \$_f\}$ and a set of dotted productions,[3] the initial element $\$_0$ is used to start computations, the final element $\$_f$ is $(S \rightarrow \bullet\delta\bullet)$ and Θ_{BPDA} contains the set of transitions derived by the compilation rules shown in table 1. [**SCAN**$_a$] and [**SCAN**$_\epsilon$] transitions are in charge of recognizing terminals and epsilon productions, respectively. [**CONC-R**] and [**CONC-L**] transitions concatenate a new part of a production to the right and to the left of the part already recognized, respectively. Once a production having B on its left-hand side has been completely recognized, a [**INC**] transition continues with the recognition of a production having B on its right-hand side

The direct execution of a BPDA may be exponential with respect to the length of the input string and may even loop. To get polynomial complexity, we must avoid duplicating computations by tabulating traces of configurations called *items*. To determine the right amount of information to keep in an item is the crucial point to get efficient executions. From [4] we know that extensions of push-down automata can be tabulated using S^2 items that store the two elements on the top of the

[2] A compilation schema is a set of rules indicating how to construct an automaton according to a given grammar and parsing strategy.

[3] Dotted productions $A \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3$ are used to indicate that the part δ_2 has been recognized.

configuration stack. Therefore, we can use S^2 items $[B, i, j; C, k, l]$, indicating the part of the input string $a_{k+1} \ldots a_l$ recognized by the top element C and the part $a_{i+1} \ldots a_j$ recognized by the element B placed immediately under C, to design a tabular framework for BPDA.[4]

New items are derived from existing items by means of inference rules of the form $\frac{antecedents}{consequent}$ *conditions* similar to those used in grammatical deduction systems [12], meaning that if all antecedents are present and conditions are satisfied then the consequent item should be generated. Conditions usually refer to transitions of the automaton and to terminals from the input string. The set of inference rules for S^2 items is the following:

$$\frac{[B, i, j; C, k, l]}{[B, i, j; F, k, l + |a|]} \quad C \overset{a}{\longmapsto}_R F \quad a = a_{l+1} \text{ or } a = \epsilon$$

$$\frac{[B, i, j; C, k, l]}{[C, k, l; F, m, m + |a|]} \quad C \overset{a}{\longmapsto}_U C\ F \quad a = a_{m+1} \text{ or } a = \epsilon$$

$$\frac{[B, i, j; C, k, l]}{[B, i, j; F, k - |a|, l]} \quad C \overset{a}{\longmapsto}_L F \quad a = a_k \text{ or } a = \epsilon$$

$$\frac{[C, k, l; F, l, m] \quad [B, i, j; C, k, l]}{[B, i, j; G, k, m]} \quad C\ F \longmapsto_R G$$

$$\frac{[B, i, j; C, k, l]}{[C, k, l; F, l, l]} \quad C \longmapsto_R C\ F$$

$$\frac{[C, k, l; F, m, k] \quad [B, i, j; C, k, l]}{[B, i, j; G, m, l]} \quad C\ F \longmapsto_L G$$

$$\frac{[B, i, j; C, k, l]}{[C, k, l; F, k, k]} \quad C \longmapsto_L C\ F$$

The worst case complexity with respect to the length n of the input string is $\mathcal{O}(n^5)$.[5] This complexity can be reduced by considering more compact kinds of items. From [4] we know that if the results of the non-deterministic computation are constrained only by bottom-up propagation of computed facts (e.g. bottom-up and Earley strategies, but not pure top-down strategies) S^1 items storing only the top element of the configuration stack can be used to derive a sound and complete tabular interpretation. In the case of BPDA, S^1 items are of the form $[C, k, l]$. The set of inference rules for S^1 items is derived from the previous set, obtaining the following one:

$$\frac{[C, k, l]}{[F, k, l + |a|]} \quad C \overset{a}{\longmapsto}_R F \quad a = a_{l+1} \text{ or } a = \epsilon$$

$$\frac{[C, k, l]}{[F, m, m + |a|]} \quad C \overset{a}{\longmapsto}_U C\ F \quad a = a_{m+1} \text{ or } a = \epsilon$$

$$\frac{[C, k, l]}{[F, k - |a|, l]} \quad C \overset{a}{\longmapsto}_L F \quad a = a_k \text{ or } a = \epsilon$$

$$\frac{[F, l, m] \quad [C, k, l]}{[G, k, m]} \quad C\ F \longmapsto_R G$$

$$\frac{[C, k, l]}{[F, l, l]} \quad C \longmapsto_R C\ F$$

$$\frac{[F, m, k] \quad [C, k, l]}{[G, m, l]} \quad C\ F \longmapsto_L G$$

$$\frac{[C, k, l]}{[F, k, k]} \quad C \longmapsto_L C\ F$$

[4] For unidirectional push-down automata working with r-transitions, only positions j and l need to be considered. For unidirectional PDA working with l-transitions only positions i and k are relevant. For BPDA the four positions i, j, k and l are needed.

[5] This is the complexity of some tabular predictive bidirectional algorithms. See for example the predictive head-corner parsing algorithm for context-free grammars described in [13, chapter 11].

This set of inference rules, when used to interpret an automaton resulting from the compilation schema shown in table 1, is equivalent to the set of deduction steps of the parsing schema[6] **dVH1** defined in [13], corresponding to a bidirectional parsing strategy:

$$[\mathbf{SCAN}_a]\quad \frac{[A \to \delta_1 \bullet \delta_2 \bullet \delta_3, k, l] \quad [a, m, m+1]}{[B \to \nu \bullet a \bullet \omega, m, m+1]} \qquad [\mathbf{SCAN}_\epsilon]\quad \frac{[A \to \delta_1 \bullet \delta_2 \bullet \delta_3, k, l]}{[B \to \bullet\bullet, m, m]}$$

$$[\mathbf{CONC\text{-}R}]\quad \frac{[A \to \nu\delta_1 \bullet \delta_2 \bullet \omega, l, m] \quad [A \to \nu \bullet \delta_1 \bullet \delta_2\omega, k, l]}{[A \to \nu \bullet \delta_1\delta_2 \bullet \omega, k, m]} \qquad [\mathbf{CONC\text{-}L}]\quad \frac{[A \to \nu \bullet \delta_1 \bullet \delta_2\omega, m, k] \quad [A \to \nu\delta_1 \bullet \delta_2 \bullet \omega, k, l]}{[A \to \nu \bullet \delta_1\delta_2 \bullet \omega, m, l]}$$

$$[\mathbf{INC}]\quad \frac{[B \to \bullet\delta\bullet, k, l]}{[A \to \nu \bullet B \bullet \omega, k, l]}$$

considering that in the parsing schemata framework [13], the antecedent $[A \to \delta_1 \bullet \delta_2 \bullet \delta_3, k, l]$ in $[\mathbf{SCAN}_a]$ and $[\mathbf{SCAN}_\epsilon]$ steps can be filtered out as it does not restrict their application, and that steps **[CONC-L]** and **[CONC-R]** can be collapsed into a single one, as order of antecedents is not relevant in a parsing schema.

3 Bidirectional Linear Indexed Automata

Linear Indexed Automata (LIA) [1, 11] are an extension of push-down automata in which each stack symbol has been associated with a list of indices. Right-oriented Linear Indexed Automata (R-LIA) [10] are a subclass of linear indexed automata that can be used to implement parsing strategies for TAG in which adjunctions are recognized in a bottom-up way. As bidirectional parsing strategies for TAG described in literature recognize adjunctions bottom-up [8, 14, 6], we take R-LIA as the starting point for the development of a bidirectional automata for TAG.

We define Bidirectional Right-oriented Linear Indexed Automata (BR–LIA) as an extension of BPDA in which each stack symbol has been associated with a list of indices. Formally, a BR-LIA is a tuple $(V_T, V_S, V_I, \Theta_{\mathrm{BR-LIA}}, \$_0, \$_f)$ where V_T is a finite set of terminal symbols, V_S is a finite set of stack symbols, V_I is a finite set of indices, $\Theta_{\mathrm{BR-LIA}}$ is a finite set of transitions, $\$_0 \in V_S$ is the initial stack symbol, and $\$_f \in V_S$ is the final stack symbol. A configuration of a BR–LIA is given by the contents of Υ, a stack of triples in $V_S[V_I^*] \times \mathbb{N} \times \mathbb{N}$. The initial configuration is $(\$_0[\,], 0, 0)$. Other configurations are attained by applying the following set of transitions in $\Theta_{\mathrm{BR-LIA}}$ as follows:

- The application of a $\mathrm{SWAP_R}[\circ\circ]$ transition of the form $C[\circ\circ p] \overset{a}{\longmapsto}_R F[\circ\circ q]$ to a configuration $\Upsilon\,(C[\eta\, p], k, l)$ yields a configuration $\Upsilon\,(F[\eta\, q], k, l + |a|)$ as a result of replacing $C[\eta\, p]$ by $F[\eta\, q]$ and scanning the terminal $a = a_{l+1}$ or the empty string $a = \epsilon$ to the right of the substring spanned by C.

[6] In brief, a parsing schema is a deductive parsing system where inference rules are called *deduction steps* and conditions on the existence of a given terminal a_{m+1} are represented by means of special antecedent items of the form $[a, m, m+1]$ called *hypothesis*.

- The application of a $\mathrm{SWAP_L}[\circ\circ]$ transition of the form $C[\circ\circ p] \overset{a}{\longmapsto}_L F[\circ\circ q]$ to a configuration $\Upsilon\,(C[\eta\,p], k, l)$ yields a configuration $\Upsilon\,(F[\eta\,q], k-|a|, l)$ as a result of replacing $C[\eta\,p]$ by $F[\eta\,q]$ and scanning the terminal $a = a_{l+1}$ or the empty string $a = \epsilon$ to the left of the substring spanned by C.

- The application of a $\mathrm{PUSH_R}[\ ]$ transition of the form $C[\circ\circ] \longmapsto_R C[\circ\circ]\ F[\ ]$ to a configuration $\Upsilon\,(C[\eta], k, l)$ yields a configuration $\Upsilon\,(C[\eta], k, l)\,(F[\,], l, l)$ as a result of pushing $F[\,]$ onto $C[\eta]$. It is expected that $F[\,]$ will span a substring inmediatly to the right of the substring spanned by $C[\eta]$.

- The application of a $\mathrm{PUSH_L}[\ ]$ transition of the form $C[\circ\circ] \longmapsto_L C[\circ\circ]\ F[\ ]$ to a configuration $\Upsilon\,(C[\eta], k, l)$ yields a configuration $\Upsilon\,(C[\eta], k, l)\,(F[\ ], k, k)$. It is expected that $F[\ ]$ will span a substring inmediatly to the left of the substring spanned by $C[\eta]$.

- The application of a $\mathrm{PUSH}_U[\ ]$ transition of the form $C[\circ\circ] \overset{a}{\longmapsto}_U C[\circ\circ]\ F[\ ]$ to a configuration $\Upsilon\,(C[\eta], k, l)$ yields a configuration $\Upsilon\,(C[\eta], k, l)\,(F[\,], m, m+|a|)$ as a result of pushing $F[\,]$ onto $C[\eta]$ and scanning the terminal $a = a_{m+1}$ or the empty string $a = \epsilon$. PUSH_U transitions are *undirected* in the sense that the substring spanned by $F[\,]$ is not necessarily adjacent to the string spanned by $C[\eta]$.

- The application of a $\mathrm{POP_R}[\ ]$ transition of the form $C[\circ\circ]\ F[\ ] \longmapsto_R G[\circ\circ]$ to a configuration $\Upsilon\,(C[\eta], k, l)\,(F[\ ], l, m)$ yields a configuration $\Upsilon\,(G[\eta], k, m)$ as a result of popping $C[\eta]$ and $F[\ ]$, which are replaced by $G[\eta]$. The substring spanned by $F[\,]$ is adjacent to the right of the substring spanned by $C[\eta]$.

- The application of a $\mathrm{POP_L}[\ ]$ transition of the form $C[\circ\circ]\ F[\ ] \longmapsto_L G[\circ\circ]$ to a configuration $\Upsilon\,(C[\eta], k, l)\,(F[\ ], m, k)$ yields a configuration $\Upsilon\,(G[\eta], m, l)$. The substring spanned by $F[\ ]$ is adjacent to the left of the substring spanned by $C[\eta]$.

- The application of a $\mathrm{POP_R}[\circ\circ]$ transition of the form $C[\ ]\ F[\circ\circ] \longmapsto_R G[\circ\circ]$ to a configuration $\Upsilon\,(C[\ ], k, l)\,(F[\eta], l, m)$ yields a configuration $\Upsilon\,(G[\eta], k, m)$ as a result of popping $C[\ ]$ and $F[\eta]$, which are replaced by G. The substring spanned by $F[\eta]$ is adjacent to the right of the substring spanned by $C[\,]$.

- The application of a $\mathrm{POP_L}[\circ\circ]$ transition of the form $C[\ ]\ F[\circ\circ] \longmapsto_L G[\circ\circ]$ to a configuration $\Upsilon\,(C[\ ], k, l)\,(F[\eta], m, k)$ yields a configuration $\Upsilon\,(G[\eta], m, l)$. The substring spanned by $F[\eta]$ is adjacent to the left of the substring spanned by $C[\,]$.

where $C, F, G \in V_S$, $a \in V_T \cup \{\epsilon\}$, $\eta \in V_I^*$ and $\circ\circ$ represent any list of indices. In the case of $\mathrm{SWAP_R}[\circ\circ]$ and $\mathrm{SWAP_L}[\circ\circ]$ transitions, $p, q \in V_I \cup \{\epsilon\}$ and either p or q, or both, must be the empty string. An input string $a_1 \ldots a_n$ is successfully recognized by a BR–LIA if the final configuration $(\$_0[\,], 0, 0)(\$_f[\,], 0, n)$ is attained.

As an example of the kind of parsers that can be implemented using BR-LIA, a compilation schema of a tree adjoining grammar into a bidirectional right-oriented linear indexed automata implementing a bottom-up bidirectional parsing strategy is derived. We consider each elementary tree γ of a TAG as formed by a set of context-free productions $\mathcal{P}(\gamma)$: a node N^γ and its g children $N_1^\gamma \ldots N_g^\gamma$ are represented by a production $N^\gamma \rightarrow N_1^\gamma \ldots N_g^\gamma$. The elements of the productions are the nodes of the tree, except for the case of elements belonging to $V_T \cup \{\varepsilon\}$ in the right-hand side of a production. Those elements may not have children and are not candidates to be adjunction nodes, so we identify

[INIT]	$\$_0[\circ\circ] \longmapsto_R \$_0[\circ\circ]\ (\top \rightarrow \bullet\bullet \mathbf{R}^\alpha)[\,]$ $\alpha \in \boldsymbol{I},\ S = \text{label}(\mathbf{R}^\alpha)$
[SCAN$_a$]	$(N^\gamma \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3)[\circ\circ] \overset{a}{\longmapsto}_U (N^\gamma \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3)[\circ\circ]\ (M^{\gamma'} \rightarrow \nu \bullet a \bullet \omega)[\,]$
[SCAN$_\epsilon$]	$(N^\gamma \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3)[\circ\circ] \overset{\epsilon}{\longmapsto}_U (N^\gamma \rightarrow \delta_1 \bullet \delta_2 \bullet \delta_3)[\circ\circ]\ (M^{\gamma'} \rightarrow \bullet\bullet)[\,]$
[CONC-R]	$(N^\gamma \rightarrow \nu \bullet \delta_1 \bullet \delta_2\omega)[\circ\circ]\ (N^\gamma \rightarrow \nu\delta_1 \bullet \delta_2 \bullet \omega)[\,] \longmapsto_R (N^\gamma \rightarrow \nu \bullet \delta_1\delta_2 \bullet \omega)[\circ\circ]$ $\forall M^\gamma \in \delta_2,\ M^\gamma \notin \text{spine}(\gamma)$
[CONC-L]	$(N^\gamma \rightarrow \nu\delta_1 \bullet \delta_2 \bullet \omega)[\circ\circ]\ (N^\gamma \rightarrow \nu \bullet \delta_1 \bullet \delta_2\omega)[\,] \longmapsto_L (N^\gamma \rightarrow \nu \bullet \delta_1\delta_2 \bullet \omega)[\circ\circ]$ $\forall M^\gamma \in \delta_1,\ M^\gamma \notin \text{spine}(\gamma)$
[SCONC-R]	$(N^\gamma \rightarrow \nu \bullet \delta_1 \bullet \delta_2\omega)[\,]\ (N^\gamma \rightarrow \nu\delta_1 \bullet \delta_2 \bullet \omega)[\circ\circ] \longmapsto_R (N^\gamma \rightarrow \nu \bullet \delta_1\delta_2 \bullet \omega)[\circ\circ]$ $\exists M^\gamma \in \delta_2,\ M^\gamma \in \text{spine}(\gamma)$
[SCONC-L]	$(N^\gamma \rightarrow \nu\delta_1 \bullet \delta_2 \bullet \omega)[\,]\ (N^\gamma \rightarrow \nu \bullet \delta_1 \bullet \delta_2\omega)[\circ\circ] \longmapsto_L (N^\gamma \rightarrow \nu \bullet \delta_1\delta_2 \bullet \omega)[\circ\circ]$ $\exists M^\gamma \in \delta_1,\ M^\gamma \in \text{spine}(\gamma)$
[INC]	$(M^\gamma \rightarrow \bullet\delta\bullet)[\circ\circ] \overset{\epsilon}{\longmapsto}_R (N^\gamma \rightarrow \nu \bullet M^\gamma \bullet \omega)[\circ\circ]$ $\mathbf{nil} \in \text{adj}(M^\gamma)$
[FOOT]	$(M^\gamma \rightarrow \bullet\delta\bullet)[\circ\circ] \overset{\epsilon}{\longmapsto}_R (\mathbf{F}^\beta \rightarrow \bullet\bot\bullet)[\circ\circ M^\gamma]$ $\beta \in \text{adj}(M^\gamma)$
[ADJ]	$(\top \rightarrow \bullet\mathbf{R}^\beta\bullet)[\circ\circ M^\gamma] \overset{\epsilon}{\longmapsto}_R (N^\gamma \rightarrow \nu \bullet M^\gamma \bullet \omega)[\circ\circ]$ $\beta \in \text{adj}(M^\gamma)$
[SUBS]	$(\top \rightarrow \bullet\mathbf{R}^\alpha\bullet)[\circ\circ] \overset{\epsilon}{\longmapsto}_R (N^\gamma \rightarrow \nu \bullet M^\gamma \bullet \omega)[\circ\circ]$ $\alpha \in \text{subs}(M^\gamma)$

Table 2: Compilation schema for TAG

such nodes labeled by a terminal with that terminal. We use $\beta \in \text{adj}(N^\gamma)$ to denote that a tree $\beta \in \boldsymbol{A}$ may be adjoined at node N^γ. If adjunction is not mandatory at N^γ, then $\mathbf{nil} \in \text{adj}(N^\gamma)$. If a tree $\alpha \in \boldsymbol{I}$ may be substituted at node N^γ, then $\alpha \in \text{subs}(N^\gamma)$. We consider the additional productions $\top \rightarrow \mathbf{R}^\alpha$, $\top \rightarrow \mathbf{R}^\beta$ and $\mathbf{F}^\beta \rightarrow \bot$ for each initial tree α and each auxiliary tree β, where $\mathbf{R}^\alpha$ is the root node of α and $\mathbf{R}^\beta$ and $\mathbf{F}^\beta$ are the root node and foot node of β, respectively.

In the resulting automata, V_T is equal to the set of terminals of the source grammar, V_S is the union of $\{\$_0, \$_f\}$ and a set of dotted productions, V_I is the set of adjunction nodes, the initial element $\$_0$ starts computations, the final element $\$_f$ is $(\top \rightarrow \bullet\mathbf{R}^\alpha\bullet)$ such that $\alpha \in \boldsymbol{I}$ and $S = \text{label}(\mathbf{R}^\alpha)$, and $\Theta_{\text{BR}-\text{LIA}}$ contains the set of transitions derived by the compilation rules shown in table 2. The function of the transitions produced by the compilation rules is as follows. A **[INIT]** transition is used to start the computation; **[SCAN$_a$]** and **[SCAN$_\epsilon$]** transitions start the recognition of terminals and epsilon productions, respectively; **[CONC-R]** and **[CONC-L]** transitions concatenate a new part of a production to the right and to the left of the part already recognized, respectively, such that the new part does not include nodes in the spine; **[SCONC-R]** and **[SCONC-L]** transitions deal with nodes in the spine; **[INC]** transitions continue the bottom-up traversal of an elementary tree once a subtree has been completely recognized; **[FOOT]** transitions start the bottom-up traversal of an elementary

tree at the foot node; [**ADJ**] transitions finish an adjunction when the bottom-up traversal of an elementary tree has been completed; [**SUBS**] transitions deal with completed substitutions.

As in the case of BPDA, the direct execution of a BR-LIA may be exponential and even loop. To get polynomial complexity, we can extend the tabular interpretations based on S^2 and S^1 items developed for BPDA. Towards this aim, we need to extend items to store information about the indices lists. Instead of storing a list in each item, we obtain a better sharing by storing only the top element and a logical pointer to other items [10]. Following the chains of pointers it is possible to retrieve the entire lists of indices. Therefore, S^2 items are of the form $[B, i, j; C, k, l \mid p \mid D, r, s; E, t, u]$, where p is the top element of the list of indices associated with C, and $(D, r, s; E, t, u)$ is the logical pointer. In the case that C is associated with an empty list of indices, we let p be the dummy index $\Diamond$, $D = E$ the dummy stack symbol $\Box$ and $r = s = t = u$ the dummy input position $-$.

To the best of our knowledge, all bidirectional parsing algorithms for TAG presented in the literature perform a bottom-up traversal of trees [14, 6, 9], which can be combined with an Earley-like traversal of some parts of a tree [8]. Therefore, a tabular interpretation based on S^1 items of the form $[C, k, l \mid p \mid E, t, u]$ is sound and complete for these strategies. The set of inference rules is the following one:

$$\frac{[C,k,l \mid p \mid E,t,u]}{[F,k,l+|a| \mid p \mid E,t,u]} \quad C[\circ\circ] \overset{a}{\longmapsto}_R F[\circ\circ] \quad a = a_{l+1} \text{ or } a = \epsilon$$

$$\frac{[C,k,l \mid p \mid E,t,u]}{[F,k,k \mid \Diamond \mid \Box,-,-]} \quad C[\circ\circ] \longmapsto_L C[\circ\circ]\, F[\,]$$

$$\frac{[C,k,l \mid p \mid E,t,u]}{[F,k-|a|,l \mid p \mid E,t,u]} \quad C[\circ\circ] \overset{a}{\longmapsto}_L F[\circ\circ] \quad a = a_k \text{ or } a = \epsilon$$

$$\frac{[C,k,l \mid p \mid E,t,u]}{[F,m,m+|a| \mid \Diamond \mid \Box,-,-]} \quad C[\circ\circ] \overset{a}{\longmapsto}_U C[\circ\circ]\, F[\,] \quad a = a_{m+1} \text{ or } a = \epsilon$$

$$\frac{[C,k,l \mid p \mid E,t,u]}{[F,k,l+|a| \mid q \mid C,k,l]} \quad C[\circ\circ] \overset{a}{\longmapsto}_R F[\circ\circ q] \quad a = a_{l+1} \text{ or } a = \epsilon$$

$$\frac{[F,l,m \mid \Diamond \mid \Box,-,-] \quad [C,k,l \mid p \mid E,t,u]}{[G,k,m \mid p \mid E,t,u]} \quad C[\circ\circ]\, F[\,] \longmapsto_R G[\circ\circ]$$

$$\frac{[C,k,l \mid p \mid E,t,u]}{[F,k-|a|,l \mid q \mid C,k,l]} \quad C[\circ\circ] \overset{a}{\longmapsto}_L F[\circ\circ q] \quad a = a_k \text{ or } a = \epsilon$$

$$\frac{[F,m,k \mid \Diamond \mid \Box,-,-] \quad [C,k,l \mid p \mid E,t,u]}{[G,m,l \mid p \mid E,t,u]} \quad C[\circ\circ]\, F[\,] \longmapsto_L G[\circ\circ]$$

$$\frac{[C,k,l \mid p \mid E,t,u] \quad [E,t,u \mid q \mid E',t',u']}{[F,k,l+|a| \mid q \mid E',t',u']} \quad C[\circ\circ p] \overset{a}{\longmapsto}_R F[\circ\circ] \quad a = a_{l+1} \text{ or } a = \epsilon$$

$$\frac{[F,l,m \mid p \mid E,t,u] \quad [C,k,l \mid \Diamond \mid \Box,-,-]}{[G,k,m \mid p \mid E,t,u]} \quad C[\,]\, F[\circ\circ] \longmapsto_R G[\circ\circ]$$

$$\frac{[C,k,l \mid p \mid E,t,u] \quad [E,t,u \mid q \mid E',t',u']}{[F,k-|a|,l \mid q \mid E',t',u']} \quad C[\circ\circ p] \overset{a}{\longmapsto}_L F[\circ\circ] \quad a = a_k \text{ or } a = \epsilon$$

$$\frac{[F,m,k \mid p \mid E,t,u] \quad [C,k,l \mid \Diamond \mid \Box,-,-]}{[G,m,l \mid p \mid E,t,u]} \quad C[\,]\, F[\circ\circ] \longmapsto_L G[\circ\circ]$$

$$\frac{[C,k,l \mid p \mid E,t,u]}{[F,l,l \mid \Diamond \mid \Box,-,-]} \quad C[\circ\circ] \longmapsto_R C[\circ\circ]\, F[\,]$$

The worst-case time complexity with respect to the length n of the input string is the standard $\mathcal{O}(n^6)$ complexity for TAG parsing.[7] This set of inference rules, when applied to the set of transitions

[7] Using S^2 items, the worst-case complexity increases to $\mathcal{O}(n^{12})$, which should be the complexity of a hypothetical

described by the previous compilation schema, gives as a result the parsing schema **dVH** for TAG defined in [6], corresponding to a bottom-up bidirectional parsing strategy:

$$[\mathbf{SCAN}_a] \quad \frac{[N^\gamma \to \delta_1 \bullet \delta_2 \bullet \delta_3, k, l \mid p \mid E, t, u] \quad [a, m, m+1]}{[M^{\gamma'} \to \nu \bullet a \bullet \omega, m, m+1 \mid \Diamond \mid \Box, -, -]}$$

$$[\mathbf{SCAN}_\epsilon] \quad \frac{[N^\gamma \to \delta_1 \bullet \delta_2 \bullet \delta_3, k, l \mid p \mid E, t, u]}{[M^{\gamma'} \to \bullet\bullet, m, m \mid \Diamond \mid \Box, -, -]}$$

$$[\mathbf{CONC\text{-}R}] \quad \frac{[N^\gamma \to \nu\delta_1 \bullet \delta_2 \bullet \omega, l, m \mid \Diamond \mid \Box, -, -] \quad [N^\gamma \to \nu \bullet \delta_1 \bullet \delta_2\omega, k, l \mid p \mid E, t, u]}{[N^\gamma \to \nu \bullet \delta_1\delta_2 \bullet \omega, k, m \mid p \mid E, t, u]}$$

$$[\mathbf{CONC\text{-}L}] \quad \frac{[N^\gamma \to \nu \bullet \delta_1 \bullet \delta_2\omega, m, k \mid \Diamond \mid \Box, -, -] \quad [N^\gamma \to \nu\delta_1 \bullet \delta_2 \bullet \omega, k, l \mid p \mid E, t, u]}{[N^\gamma \to \nu \bullet \delta_1\delta_2 \bullet \omega, m, l \mid p \mid E, t, u]}$$

$$[\mathbf{SCONC\text{-}R}] \quad \frac{[N^\gamma \to \nu\delta_1 \bullet \delta_2 \bullet \omega, l, m \mid p \mid E, t, u] \quad [N^\gamma \to \nu \bullet \delta_1 \bullet \delta_2\omega, k, l \mid \Diamond \mid \Box, -, -]}{[N^\gamma \to \nu \bullet \delta_1\delta_2 \bullet \omega, k, m \mid p \mid E, t, u]}$$

$$[\mathbf{SCONC\text{-}L}] \quad \frac{[N^\gamma \to \nu \bullet \delta_1 \bullet \delta_2\omega, m, k \mid p \mid E, t, u] \quad [N^\gamma \to \nu\delta_1 \bullet \delta_2 \bullet \omega, k, l \mid \Diamond \mid \Box, -, -]}{[N^\gamma \to \nu \bullet \delta_1\delta_2 \bullet \omega, m, l \mid p \mid E, t, u]}$$

$$[\mathbf{INC}] \quad \frac{[M^\gamma \to \bullet\delta\bullet, k, l \mid p \mid E, t, u]}{[N^\gamma \to \nu \bullet M^\gamma \bullet \omega, k, l \mid p \mid E, t, u]}$$

$$[\mathbf{FOOT}] \quad \frac{[M^\gamma \to \bullet\delta\bullet, k, l \mid p \mid E, t, u]}{[\mathbf{F}^\beta \to \bullet\bot\bullet, k, l \mid M^\gamma \mid M^\gamma \to \bullet\delta\bullet, k, l]}$$

$$[\mathbf{ADJ}] \quad \frac{[\top \to \bullet\mathbf{R}^\beta\bullet, k, l \mid M^\gamma \mid M^\gamma \to \bullet\delta\bullet, t, u] \quad [M^\gamma \to \bullet\delta\bullet, t, u \mid q \mid E', t', u']}{[N^\gamma \to \nu \bullet M^\gamma \bullet \omega, k, l \mid q \mid E', t', u']}$$

$$[\mathbf{SUBS}] \quad \frac{[\top \to \bullet\mathbf{R}^\alpha\bullet, k, l \mid p \mid E, t, u]}{[N^\gamma \to \nu \bullet M^\gamma \bullet \omega, k, l \mid p \mid E, t, u]}$$

Antecedent $[N^\gamma \to \delta_1 \bullet \delta_2 \bullet \delta_3, k, l \mid p \mid E, t, u]$ in $[\mathbf{SCAN}_a]$ and $[\mathbf{SCAN}_\epsilon]$ can be filtered out as it does not restrict the application of these steps. Steps $[\mathbf{CONC\text{-}R}]$ and $[\mathbf{SCONC\text{-}L}]$ can be collapsed into a single step, and $[\mathbf{CONC\text{-}L}]$ and $[\mathbf{SCONC\text{-}R}]$ can also be collapsed into a single step, as the

predictive head-corner parsing algorithm for TAG, such as an extension of the predictive head-corner algorithm proposed in [13, chapter 11] for context-free grammars.

order of antecedents is not important in a parsing schema. In the step [**SUBS**] it can be shown that $p = \Diamond$, $E = \Box$ and $t = u = -$ since α is an initial tree.

It is interesting to remark that some components of items are redundant. This fact is clear when we observe the [**ADJ**] step. The element $M^\gamma \rightarrow \bullet\delta\bullet$ of the item $[\top \rightarrow \bullet\mathbf{R}^\beta\bullet, k, l \mid M^\gamma \mid M^\gamma \rightarrow \bullet\delta\bullet, t, u]$ is redundant, because it is the production associated to M^γ, already present in the item. The element M^γ is itself redundant, as the item is valid for all node M^γ of an elementary tree such that β can be adjoined at M^γ. With this refinement of items, we obtain exactly the schema **dVH** for TAG proposed in [6].

4 Conclusions

In order to provide a common framework for the description of bidirectional parsing algorithms for TAG, we have defined a new class of bidirectional automata which works in polynomial time and we have shown how tabular parsing algorithms can be derived from the automaton describing the parsing strategy and the tabulation technique associated to the automata model. As illustration, we have considered the case of the bottom-up bidirectional strategy for TAG proposed in [6] but the approach can be applied to the other bidirectional strategies defined in the literature. A study of the performance of this strategy and a comparison with other uni- and bidirectional strategies for TAG can be found in [5]. The use of bidirectional automata to define parsers allowed us to concentrate on the parsing strategy itself, abstracting for details of implementation such as the input positions spanned by the elements in a production.

Acknowledgments

This research has been partially supported by Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica (Grant TIC2000-0370-C02-01), FEDER of EU (Grant 1FD97-0047-C04-02) and Xunta de Galicia (Grants PGIDT99XI10502B and PGIDT01PXI10506PN).

References

[1] Miguel A. Alonso, Mark-Jan Nederhof, and Eric de la Clergerie. Tabulation of Automata for Tree Adjoining Languages. *Grammars*, 3(2/3):89–110, 2000.

[2] Sylvie Billot and Bernard Lang. The structure of shared forest in ambiguous parsing. In *Proc. of the 27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151, Vancouver, British Columbia, Canada, June 1989. ACL.

[3] Eric de la Clergerie and Miguel A. Alonso. A tabular interpretation of a class of 2-Stack Automata. In *COLING-ACL'98, 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Proceedings of the Conference*, volume II, pages 1333–1339, Montreal, Quebec, Canada, August 1998. ACL.

[4] Eric de la Clergerie and Bernard Lang. LPDA: Another look at tabulation in logic programming. In Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming (ICLP'94)*, pages 470–486. MIT Press, June 1994.

[5] Víctor J. Díaz and Miguel A. Alonso. Comparing tabular parsers for Tree Adjoining Grammars. In David S. Warren, Manuel Vilares, Leandro Rodríguez Liñares, and Miguel A. Alonso, editors, *Proc. of Second International Workshop on Tabulation in Parsing and Deduction (TAPD 2000)*, pages 91–100, Vigo, Spain, September 2000.

[6] Víctor J. Díaz, Miguel A. Alonso, and Vicente Carrillo. Bidirectional parsing of TAG without heads. In *Proc. of 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 67–72, Paris, France, May 2000.

[7] Aravind K. Joshi and Yves 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.

[8] Alberto Lavelli and Giorgio Satta. Bidirectional parsing of lexicalized tree adjoining grammars. In *Proceedings of the 5th Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, Berlin, Germany, April 1991. ACL.

[9] Patrice Lopez. Extended partial parsing for lexicalized tree grammars. In *Proc. of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 159–170, Trento, Italy, February 2000.

[10] Mark-Jan Nederhof. Linear indexed automata and tabulation of TAG parsing. In *Proc. of First Workshop on Tabulation in Parsing and Deduction (TAPD'98)*, pages 1–9, Paris, France, April 1998.

[11] Mark-Jan Nederhof. Models of tabulation for TAG parsing. In *Proc. of the Sixth Meeting on Mathematics of Language (MOL 6)*, pages 143–158, Orlando, Florida, USA, July 1999.

[12] Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36, July-August 1995.

[13] Klaas Sikkel. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms.* Texts in Theoretical Computer Science — An EATCS Series. Springer-Verlag, Berlin/Heidelberg/New York, 1997.

[14] Gertjan van Noord. Head-corner parsing for TAG. *Computational Intelligence*, 10(4):525–534, 1994.