A formal definition of Bottom-up Embedded Push-Down Automata and their tabulation technique

Miguel A. Alonso, Eric de la Clergerie and Manuel Vilares

Abstract

We provide a formal definition of Bottom-up Embedded Push-Down Automata. By removing the finite-state control, we obtain an alternative definition which is adequate to define a tabulation framework for this class of automata and to show the equivalence with respect to other models of automata accepting tree adjoining languages.

Keywords: Automata, parsing, tabulation, tree adjoining grammars.

1 Introduction

Tree adjoining grammars (TAG) [7] are an extension of context-free grammars that use trees instead of productions as the primary representing structure. Formally, a TAG is a 5-tuple $\mathcal{G} = (V_N, V_T, S, \boldsymbol{I}, \boldsymbol{A})$, 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, $\boldsymbol{I}$ a finite set of *initial trees* and $\boldsymbol{A}$ a finite 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 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 elementary tree from the root node to the foot node is called the *spine* of the tree.

New trees are derived by *adjoining*: let α be a tree containing a node N^α 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^α is obtained by excising the

Miguel A. Alonso Pardo, Departamento de Computación, Facultad de Informática, Universidad de La Coruña, Campus de Elviña s/n, 15071 La Coruña (Spain), http://www.dc.fi.udc.es/~alonso/, alonso@dc.fi.udc.es

Eric Villemonte de la Clergerie, Institut National de Recherche en Informatique et en Automatique, Domaine de Voluceau, Rocquecourt, B.P. 105, 78153 Le Chesnay Cedex (France), http://atoll.inria.fr/~clerger/, Eric.De_La_Clergerie@inria.fr

Manuel Vilares Ferro, Departamento de Computación, Facultad de Informática, Universidad de La Coruña, Campus de Elviña s/n, 15071 La Coruña (Spain), http://www.dc.fi.udc.es/~vilares/, vilares@dc.fi.udc.es

This research has been partially supported by the FEDER of EU (Grant 1FD97-0047-C04-02) and Xunta de Galicia (Grant PGIDT99XI10502B).

subtree of α with root N^{α}, attaching β to N^{α} 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.

Bottom-up embedded push-down automata (BEPDA) have been described in [11] as an extension of push-down automata adequate to implement parsing strategies for TAG in which adjunctions are recognized in a bottom-up way. In fact, BEPDA are the dual version of embedded push-down automata (EPDA) [12, 2], a model of automata in which adjunctions must be recognized in a top-down way. A less informal presentation of BEPDA has been shown in [10], with some inconsistencies between the set of allowed transitions and the set of configurations attainable.

Right-oriented linear indexed automata [9] and bottom-up 2–stack automata [5] can be used to implement parsing strategies similiar to those of BEPDA. Both models of automata have associated a tabulation framework allowing their execution in polynomial time with respect to the size of the input string. In this paper we provide a formal definition for BEPDA and then we propose an alternative definition in order to provide a tabulation framework for this class of automata.

2 A formal definition of BEPDA

A bottom-up embedded push-down automata consists of a finite-state control, an input tape and a stack (that we call the main stack) made up of non-empty stacks containing stack symbols. Formally, a BEPDA is defined by a tuple $(Q, V_T, V_S, \delta, q_0, Q_F, \$_f)$ where:

- Q is a finite set of states.
- V_T is a finite set of terminal symbols.
- V_S is a finite set of stack symbols.
- $q_0 \in Q$ is the initial state.
- $Q_F \subseteq Q$ is the set of final states.
- $\$_f \in V_S$ is the final stack symbol.
- δ is a mapping from $Q \times V_T \cup \{\epsilon\} \times (\lceil V_S^+)^* \times V_S^* \times (\lceil V_S^+)^*$ into finite subsets of $Q \times V_S \cup \{\lceil V_S\}$, where $\lceil \notin V_S$ is a new symbol used as stack separator.

An *instantaneous configuration* is a triple (q, Υ, w), where q is the current state, $\Upsilon \in (\lceil V_S^+)^*$ represents the contents of the automaton stack and w is the part of the input string that is yet to be read. It is important to remark that every individual stack contained in the main stack must store at least one stack symbol. The main stack will be empty only in the initial configuration (q_0, ϵ, w).

Transitions in δ allow an automaton to derive a new configuration (q', Υ', w) from a configuration (q, Υ, aw), which is denoted $(q, \Upsilon, aw) \vdash (q', \Upsilon', w)$. The reflexive and transitive closure of $\vdash$ is denoted by $\overset{*}{\vdash}$. There exist two different types of transitions:

1. Transitions of the first type are of the form

$$(q', \lceil Z) \in \delta(q, a, \epsilon, \epsilon, \epsilon)$$

and they can be applied to a configuration (q, Υ, aw) to yield a configuration $(q', \Upsilon \lceil Z, w)$.

2. Transitions of the second type are of the form

$$(q', Z) \in \delta(q,\ a,\ \lceil\alpha_k \ldots \lceil\alpha_{i+1},\ Z_m \ldots Z_1,\ \lceil\alpha_i \ldots \lceil\alpha_1)$$

where $m \geq 0$ and $k \geq i \geq 0$. They can be applied to a configuration

$$(q,\ \Upsilon \lceil\alpha_k \ldots \lceil\alpha_{i+1} \lceil\alpha Z_m \ldots Z_1 \lceil\alpha_i \ldots \lceil\alpha_1,\ aw)$$

to yield a configuration

$$(q', \Upsilon \lceil\alpha Z, w)$$

The *language accepted by final state* by a BEPDA is the set $w \in V_T^*$ of input strings such that $(q_0, \epsilon, w) \vdash {}^*(p, \Upsilon, \epsilon)$, where $p \in Q_F$ and $\Upsilon \in (\lceil V_S^+)^*$.

The *language accepted by empty stack* by a BEPDA is the set $w \in V_T^*$ of input strings such that $(q_0, \epsilon, w) \vdash {}^*(q, \lceil\$_f, \epsilon)$ for any $q \in Q$. At this point it is interesting to observe the duality with respect to EPDA: EPDA computations start with a stack $\lceil\$_0$ to finish with an empty stack while BEPDA computations start with an empty stack to finish with a stack $\lceil\$_f$.

It can be proved that for any BEPDA accepting a language L by final state there exists a BEPDA accepting the same language by empty stack and vice versa.

Example 1 The bottom-up embedded push-down automaton defined by the tuple $(\{q_0, q_1, q_2, q_3\}, \{a, b, c, d\}, \{B, C, D\}, \delta, q_0, \emptyset, \$_f)$, with δ containing the transitions shown in Fig. 1 (left box), accepts the language $\{a^n b^n c^n d^n \mid n \geq 0\}$ by empty stack. The sequence of configurations for the recognition of the input string *aabbccdd* is shown in Fig. 1 (right box), where the first column shows the transition applied, the second one the current state, the third one the contents of the stack and the fourth column shows the part of the input string to be read. □

3 BEPDA without states

Finite-state control is not a fundamental component of push-down automata, as the current state in a configuration can be stored in the top element of the stack of the automaton [8]. Finite-state control can also be eliminated from bottom-up embedded push-down automata, obtaining a new definition that considers a BEPDA 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 six types of transition:

(a)	$(q_0, \lceil D) \in \delta(q_0, a, \epsilon, \epsilon, \epsilon)$
(b)	$(q_1, \lceil C) \in \delta(q_0, b, \epsilon, \epsilon, \epsilon)$
(c)	$(q_1, \lceil C) \in \delta(q_1, b, \epsilon, \epsilon, \epsilon)$
(d)	$(q_2, B) \in \delta(q_1, c, \epsilon, C, \epsilon)$
(e)	$(q_2, B) \in \delta(q_2, c, \lceil C, \epsilon, \epsilon)$
(f)	$(q_3, B) \in \delta(q_2, d, \lceil D, BB, \epsilon)$
(g)	$(q_3, B) \in \delta(q_3, d, \lceil D, BB, \epsilon)$
(h)	$(q_3, \$_f) \in \delta(q_3, d, \lceil D, B, \epsilon)$
(i)	$(q_0, \lceil \$_f) \in \delta(q_0, a, \epsilon, \epsilon, \epsilon)$

	q_0		$aabbccdd$
(a)	q_0	$\lceil D$	$abbccdd$
(a)	q_0	$\lceil D \lceil D$	$bbccdd$
(b)	q_1	$\lceil D \lceil D \lceil C$	$bccdd$
(c)	q_1	$\lceil D \lceil D \lceil C \lceil C$	$ccdd$
(d)	q_2	$\lceil D \lceil D \lceil C \lceil B$	cdd
(e)	q_2	$\lceil D \lceil D \lceil BB$	dd
(f)	q_3	$\lceil D \lceil B$	d
(h)	q_3	$\lceil \$_f$	

Figure 1: BEPDA accepting $\{a^n b^n c^n d^n \mid n > 0\}$ and configurations for $aabbccdd$

SWAP: Transitions of the form $C \stackrel{a}{\longmapsto} F$ that replace the top element of the top stack while scanning a. The application of such a transition on a stack $\Upsilon \lceil \alpha C$ returns the stack $\Upsilon \lceil \alpha F$.

PUSH: Transitions of the form $C \stackrel{a}{\longmapsto} C\ F$ that push F onto C. The application of such a transition on a stack $\Upsilon \lceil \alpha C$ returns the stack $\Upsilon \lceil \alpha C F$.

POP: Transitions of the form $C\ F \stackrel{a}{\longmapsto} G$ that replace C and F by G. The application of such a transition on $\Upsilon \lceil \alpha C F$ returns the stack $\Upsilon \lceil \alpha G$.

WRAP: Transitions of the form $C \stackrel{a}{\longmapsto} C, \lceil F$ that push a new stack $\lceil F$ on the top of the main stack. The application of such a transition on a stack $\Upsilon \lceil \alpha C$ returns the stack $\Upsilon \lceil \alpha C \lceil F$.

UNWRAP-A: Transitions *unwrap-above* of the form $C, \lceil F \stackrel{a}{\longmapsto} G$ that delete the top stack $\lceil F$ and replace the new top element by G. The application of such a transition on a stack $\Upsilon \lceil \alpha C \lceil F$ returns the stack $\Upsilon \lceil \alpha G$.

UNWRAP-B: Transitions *unwrap-below* of the form $\lceil C, F \stackrel{a}{\longmapsto} G$ that delete the stack $\lceil C$ placed just below the top stack and replace the top element by G. The application of such a transition on a stack $\Upsilon \lceil C \lceil \alpha F$ returns the stack $\Upsilon \lceil \alpha G$.

(a)	$\$_0 \overset{a}{\longmapsto} \$_0, \lceil D$
(b)	$D \overset{a}{\longmapsto} D, \lceil D$
(c)	$D \longmapsto D, \lceil C$
(d)	$C \overset{b}{\longmapsto} C, \lceil C$
(e)	$C \longmapsto B$
(f)	$B \longmapsto BE$
(g)	$\lceil C, E \overset{c}{\longmapsto} C$
(h)	$BC \longmapsto B$
(i)	$\lceil D, B \overset{d}{\longmapsto} D$
(j)	$BD \longmapsto B$
(k)	$\$_0, \lceil D \longmapsto \$_f$

	$\lceil \$_0$	$aabbccdd$
(a)	$\lceil \$_0 \lceil D$	$aabbccdd$
(b)	$\lceil \$_0 \lceil D \lceil D$	$abbccdd$
(c)	$\lceil \$_0 \lceil D \lceil D \lceil C$	$bbccdd$
(d)	$\lceil \$_0 \lceil D \lceil D \lceil C \lceil C$	$bccdd$
(d)	$\lceil \$_0 \lceil D \lceil D \lceil C \lceil C \lceil C$	$ccdd$
(e)	$\lceil \$_0 \lceil D \lceil D \lceil C \lceil C \lceil B$	$ccdd$
(f)	$\lceil \$_0 \lceil D \lceil D \lceil C \lceil C \lceil BE$	$ccdd$
(g)	$\lceil \$_0 \lceil D \lceil D \lceil C \lceil BC$	cdd
(e)	$\lceil \$_0 \lceil D \lceil D \lceil C \lceil BB$	cdd
(f)	$\lceil \$_0 \lceil D \lceil D \lceil C \lceil BBE$	cdd
(g)	$\lceil \$_0 \lceil D \lceil D \lceil BBC$	dd
(h)	$\lceil \$_0 \lceil D \lceil D \lceil BB$	dd
(i)	$\lceil \$_0 \lceil D \lceil BD$	d
(j)	$\lceil \$_0 \lceil D \lceil B$	d
(i)	$\lceil \$_0 \lceil D$	
(k)	$\lceil \$_f$	

Figure 2: BEPDA without finite-state control for $\{a^n b^n c^n d^n \mid n > 0\}$

where $C, F, G \in V_S$, $\Upsilon \in (\lceil V_S^*)^*$, $\alpha \in V_S^*$ and $a \in V_T \cup \{\epsilon\}$. It can be proved that transitions of a BEPDA with states can be emulated by transitions in Θ and vice versa.

An *instantaneous configuration* is a pair (Υ, w), where Υ represents the contents of the automaton stack and w is the part of the input string that is yet to be read. A configuration (Υ, aw) derives a configuration (Υ', w), denoted $(\Upsilon, aw) \vdash (\Upsilon', w)$, if and only if there exist a transition that applied to Υ gives Υ' and scans a from the input string. We use $\vdash^*$ to denote the reflexive and transitive closure of $\vdash$. An input string is accepted by an BEPDA if $(\lceil \$_0, w) \vdash^* (\lceil \$_f, \epsilon)$. The language accepted by an BEPDA is the set of $w \in V_T^*$ such that $(\lceil \$_0, w) \vdash^* (\lceil \$_f, \epsilon)$.

Example 2 The bottom-up embedded push-down automaton without states defined by the tuple $(\{a, b, c, d\}, \{B, C, D, E, F, \$_0, \$_f\}, \Theta, \$_0, \$_f)$, with Θ containing the transitions shown in Fig. 2 (left box), accepts the language $\{a^n b^n c^n d^n \mid n \geq 0\}$. The sequence of configurations to recognize the input string *aabbccdd* is also shown in Fig. 2 (right-box), where the first column shows the transition applied in each step, the second one shows the contents of the stack and the third column shows the part of the input string to be read. □

[INIT]	$\$_0 \longmapsto \$_0 \; [\overrightarrow{\top^\alpha}$	$\alpha \in \boldsymbol{I},\ S = \text{label}(\mathbf{R}^\alpha)$
[FINAL]	$\$_0 \; [\overleftarrow{\top^\alpha} \longmapsto \$_f$	$\alpha \in \boldsymbol{I},\ S = \text{label}(\mathbf{R}^\alpha)$
[CALL]	$\nabla^\gamma_{r,s} \longmapsto \nabla^\gamma_{r,s},\ [\overrightarrow{N^\gamma_{r,s+1}}$	$N^\gamma_{r,s+1} \notin \text{spine}(\gamma),\ \mathbf{nil} \in \text{adj}(N_{r,s+1})$
[SCALL]	$\nabla^\beta_{r,s} \longmapsto \nabla^\beta_{r,s} \; [\overrightarrow{N^\beta_{r,s+1}}$	$N^\beta_{r,s+1} \in \text{spine}(\beta),\ \mathbf{nil} \in \text{adj}(N^\beta_{r,s+1})$
[SEL]	$\overrightarrow{N^\gamma_{r,0}} \longmapsto \nabla^\gamma_{r,0}$	
[PUB]	$\nabla^\gamma_{r,n_r} \longmapsto \overleftarrow{N^\gamma_{r,0}}$	
[RET]	$\nabla^\gamma_{r,s},\ [\overleftarrow{N^\gamma_{r,s+1}} \longmapsto \nabla^\gamma_{r,s+1}$	$N^\gamma_{r,s+1} \notin \text{spine}(\gamma),\ \mathbf{nil} \in \text{adj}(N_{r,s+1})$
[SRET]	$[\nabla^\beta_{r,s},\ \overleftarrow{N^\beta_{r,s+1}} \longmapsto \nabla^\beta_{r,s+1}$	$N^\beta_{r,s+1} \in \text{spine}(\beta),\ \mathbf{nil} \in \text{adj}(N^\beta_{r,s+1})$
[SCAN]	$\overrightarrow{N^\gamma_{r,0}} \overset{a}{\longmapsto} \overleftarrow{N^\gamma_{r,0}}$	$N^\gamma_{r,0} \to a$
[AdjCALL]	$\nabla^\gamma_{r,s} \longmapsto \nabla^\gamma_{r,s},\ [\overrightarrow{\top^\beta}$	$\beta \in \text{adj}(N^\gamma_{r,s+1})$
[AdjRET-a]	$[\nabla^\gamma_{r,s},\ \overleftarrow{\top^\beta} \longmapsto \top$	$\beta \in \text{adj}(N^\gamma_{r,s+1})$
[AdjRET-b]	$\Delta^\gamma_{r,s} \ \top \longmapsto \nabla^\gamma_{r,s+1}$	
[FootCALL]	$\nabla^\beta_{f,0} \longmapsto \nabla^\beta_{f,0},\ [\overrightarrow{N^\gamma_{r,s+1}}$	$N^\beta_{f,0} = \mathbf{F}^\beta,\quad \beta \in \text{adj}(N^\gamma_{r,s+1})$
[FootRET-a]	$[\nabla^\beta_{f,0},\ \overleftarrow{N^\gamma_{r,s+1}} \longmapsto \Delta^\gamma_{r,s}$	$N^\beta_{f,0} = \mathbf{F}^\beta,\quad \beta \in \text{adj}(N^\gamma_{r,s+1})$
[FootRET-b]	$\Delta^\gamma_{r,s} \longmapsto \Delta^\gamma_{r,s} \; \nabla^\beta_{f,1}$	$N^\beta_{f,0} = \mathbf{F}^\beta,\quad \beta \in \text{adj}(N^\gamma_{r,s+1})$
[SubsCALL]	$\nabla^\gamma_{r,s} \longmapsto \nabla^\gamma_{r,s},\ [\overrightarrow{\top^\alpha}$	$\alpha \in \text{subs}(N_{r,s+1})$
[SubsRET]	$\nabla^\gamma_{r,s},\ [\overleftarrow{\top^\alpha} \longmapsto \nabla^\gamma_{r,s+1}$	$\alpha \in \text{subs}(N_{r,s+1})$

Figure 3: Generic compilation schema for TAG

4 Compiling TAG into BEPDA

Automata are interesting for parsing because they allow us to separate two different problems that arise during the definition of parsing algorithms: the description of the parsing strategy and the execution of the parser. By means of automata, a parsing strategy for a given grammar can be translated into a set of transitions defining a (possibly non deterministic) automaton and then the automaton can be executed using some standard technique.

In this section we define a generic compilation schema for tree adjoining grammars

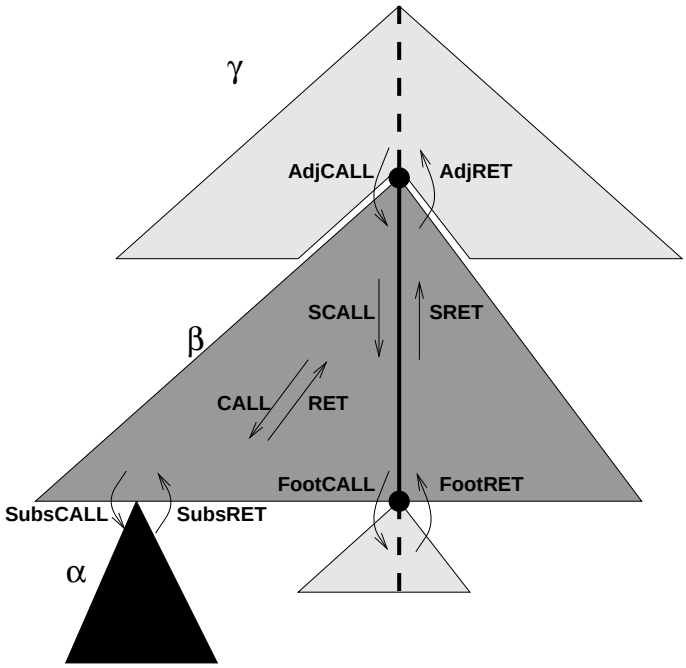

Figure 4: Meaning of compilation rules

based on a call/return model [6]. 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 $N_1^\gamma \dots N_g^\gamma$ are represented by a production $N^\gamma \rightarrow N_1^\gamma \dots N_g^\gamma$. 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 production. Those elements may not have children and are not candidates to be adjunction nodes, so we identify 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 $\top^\alpha \rightarrow \mathbf{R}^\alpha$, $\top^\beta \rightarrow \mathbf{R}^\beta$ 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.

Fig. 3 shows the compilation rules from TAG to BEPDA, where symbols $\nabla_{r,s}^\gamma$ has been introduced to denote dotted productions. The meaning of each compilation rule is graphically shown in Fig. 4. This schema is parameterized by $\overrightarrow{N^\gamma}$, the information propagated top-down with respect to the node N^γ, and by $\overleftarrow{N^\gamma}$, the information propagated bottom-up. When the schema is used to implement a top-down traversal of elementary trees $\overrightarrow{N^\gamma} = N^\gamma$ and $\overleftarrow{N^\gamma} = \Box$, where $\Box$ is a fresh stack symbol. A bottom-up traversal requires $\overrightarrow{N^\gamma} = \Box$ and $\overleftarrow{N^\gamma} = N^\gamma$. For a mixed traversal of elementary trees, $\overrightarrow{N^\gamma} = \overline{N^\gamma}$ and $\overleftarrow{N^\gamma} = \overline{\overline{N^\gamma}}$, where $\overline{N^\gamma}$ and $\overline{\overline{N^\gamma}}$ are used to distinguish the top-down prediction from the bottom-up propagation of a node.

With respect to adjunctions, we can observe in Fig. 3 that each stack stores pending adjunctions with respect to the node placed on the top of the stack in a bottom-up treatment of adjunctions: when a foot node is reached, the adjunction node is stored on the top of the stack ([**FootCALL-a**]); the traversal of the elemen-

Transition	BEPDA	R–LIA
SWAP	$C \overset{a}{\longmapsto} F$	$C[\circ\circ] \overset{a}{\longmapsto} F[\circ\circ]$
PUSH	$C \overset{a}{\longmapsto} CF$	$C[\circ\circ] \overset{a}{\longmapsto} F[\circ\circ C]$
POP	$CF \overset{a}{\longmapsto} G$	$F[\circ\circ C] \overset{a}{\longmapsto} G[\circ\circ]$
UNWRAP-A	$C, [F \overset{a}{\longmapsto} G$	$C[\circ\circ]\ F[\,] \overset{a}{\longmapsto} G[\circ\circ]$
UNWRAP-B	$[C, F \overset{a}{\longmapsto} G$	$C[\,]\ F[\circ\circ] \overset{a}{\longmapsto} G[\circ\circ]$
WRAP	$C \overset{a}{\longmapsto} C, [F$	$C[\circ\circ] \overset{a}{\longmapsto} C[\circ\circ]\ F[\,]$

Figure 5: Equivalence between BEPDA and L-LIA

tary tree is suspended to continue with the traversal of the adjoined auxiliary tree ([**FootCALL-b**]); the adjunction stack is propagated through the spine ([**SRET**]) up to the root node ([**AdjRET-a**]); and then the stack element corresponding to the auxiliary tree is eliminated to resume the traversal of the elementary tree ([**AdjRET-b**]). To avoid confusion, we store $\Delta^{\gamma}_{r,s}$ instead of $\nabla^{\gamma}_{r,s}$ to indicate that an adjunction was started at node $N^{\gamma}_{r,s+1}$. A symbol Δ can be seen as a symbol ∇ waiting an adjunction to be completed.

Substitution is managed trough transitions generated by compilation rules [**SubsCALL**], which start the traversal of the substituted trees, and [**SubsRET**], which resume the traversal of the tree containing the substitution node once the substituted tree has been completely traversed.

5 BEPDA and other automata for TAG

5.1 Right-oriented linear indexed automata

Linear indexed automata (LIA) [3] are an extension of push-down automata in which each stack symbol has been associated to a list of indices. Right-oriented linear indexed automata (R-LIA) 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. BEPDA and R–LIA are equivalent classes of automata. Given a BEPDA, the equivalent R-LIA is obtained by means of a simple change in the notation: the top element of a stack is considered a stack symbol, and the rest of the stack is considered the indices list associated to it, as is shown in Fig. 5. The same procedure also serves to obtain the BEPDA equivalent to a given R–LIA.

5.2 Bottom-up 2–stack automata

Strongly-driven 2-stack automata (SD–2SA) [4] are an extension of push-down automata working on a pair of asymmetric stacks, a master stack and an auxiliary

BEPDA transition		**BU–2SA transition**	
SWAP	$C \overset{a}{\longmapsto} F$	$(m, C, \epsilon) \overset{a}{\longmapsto} (m, F, \epsilon)$	**SWAP1**
	$\lceil C \overset{a}{\longmapsto} \lceil F$	$(\mathbf{w}, C, \models^m) \overset{a}{\longmapsto} (\mathbf{e}, F, \models^m)$	**SWAP2**
WRAP	$C \overset{a}{\longmapsto} C, \lceil F$	$(m, C, \epsilon) \overset{a}{\longmapsto} (\mathbf{w}, C \models^m F, \models^m)$	$\models$**WRITE**
	$\lceil C \overset{a}{\longmapsto} \lceil C, \lceil F$	$(\mathbf{w}, C, \epsilon) \overset{a}{\longmapsto} (\mathbf{w}, C \rhd F, \epsilon)$	$\rhd$**WRITE**
UNWRAP-A	$C, \lceil F \overset{a}{\longmapsto} G$	$(\mathbf{e}, C \models^m F, \models^m) \overset{a}{\longmapsto} (m, G, \epsilon)$	$\models$**ERASE**
UNWRAP-B	$\lceil C, F \overset{a}{\longmapsto} G$	$(\mathbf{e}, C \rhd F, \epsilon) \overset{a}{\longmapsto} (\mathbf{e}, G, \epsilon)$	$\rightarrow$**ERASE**
	$\lceil C, XF \overset{a}{\longmapsto} G$	$(\mathbf{e}, C \rhd F, X) \overset{a}{\longmapsto} (\mathbf{e}, G, \epsilon)$	$\nearrow$**ERASE**
	$\lceil C, F \overset{a}{\longmapsto} XG$	$(\mathbf{e}, C \rhd F, \epsilon) \overset{a}{\longmapsto} (\mathbf{e}, G, X)$	$\searrow$**ERASE**

Figure 6: Correspondence between BEPDA and BU–2SA

stack. These stacks are partitioned into sessions. Computations in each session are performed in one of two modes *write* and *erase*. A session starts in mode write and switches at some point to mode erase. In mode write (resp. erase), no element can be popped from (resp. pushed to) the master stack. Switching back from erase to write mode is not allowed. Bottom-up 2–stack automata (BU–2SA) [5] are a projection of SD–2SA requiring the emptiness of the auxiliary stack during computations in mode write. When a new session is created in a BU–2SA, a mark $\models$ is left on the master stack, other movements performed in write mode leaving a mark $\rhd$. These marks are popped in erase mode.

The full set of BU–2SA transitions is shown in Fig. 6. Transitions of type **SWAP2** are equivalent to $\lceil C \overset{a}{\longmapsto} \lceil F$ in BEPDA, compound transitions obtained from the consecutive application of $C \longmapsto C, \lceil F'$ and $\lceil C, F' \overset{a}{\longmapsto} F$, where F' is a fresh stack symbol. In a similar way, transitions of type $\nearrow$**ERASE** are translated into compound transitions formed by an UNWRAP-B and a POP transition, and transitions of type $\searrow$**ERASE** are translated into the composition of UNWRAP-B and PUSH transitions. Slightly different is the case for transitions of type $\rhd$**WRITE**, equivalent to $\lceil C \overset{a}{\longmapsto} \lceil C, \lceil F$ transitions in BEPDA, which are obtained as the consecutive application of $\lceil C \overset{a}{\longmapsto} \lceil C'$ and $C' \longmapsto C', \lceil F$, an additional transition $C', \lceil G \overset{b}{\longmapsto} K$ for each transition $C, \lceil G \overset{b}{\longmapsto} K$ in the automaton, and an additional transition $\lceil C', G \overset{b}{\longmapsto} K$ for each transition $\lceil C, G \overset{b}{\longmapsto} K$, where C' is a fresh stack symbol.

As a consequence, it is possible to build a BEPDA for any given BU–2SA. However, the reverse is not always true: PUSH and POP transitions can only be translated into BU–2SA if they are merged with an UNWRAP-B transition. So, a BEPDA implementing a shift-reduce strategy (requiring the use of PUSH and POP transitions in combination with UNWRAP-A transitions[1]) can not be translated into a BU–2SA.

[1] A linear indexed automata implementing a LR-like strategy for linear indexed grammars using this kind of transitions is described in [1].

6 Tabulation

The direct execution of BEPDA 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*. The amount of information to keep in an item is the crucial point to determine to get efficient executions.

In order to define items and attending to the form of the transitions, we can classify derivations of BEPDA into the following two types:

Call derivations. Correspond to the placement of an unitary stack onto the top of the main stack:

$$(\Upsilon \ [B, a_{i+1} \dots a_n) \vdash^* (\Upsilon \ [C, a_{j+1} \dots a_n)$$

where $B, C \in V_S$ and $\Upsilon \in ([V_S^*)^*$.These derivations are independent of Υ and so they can be represented by items

$$[B, i, C, j, - \mid -, -, -, -]$$

Return derivations. Correspond to the placement of a non-unitary stack onto the top of the main stack:

$$\begin{aligned}
&(\Upsilon \ [B, a_{i+1} \dots a_n) \\
&\vdash^* (\Upsilon \ [B \ \ \Upsilon_1 \ [D, a_{p+1} \dots a_n) \\
&\vdash^* (\Upsilon \ [B \ \ \Upsilon_1 \ [\alpha E, a_{q+1} \dots a_n) \\
&\vdash^* (\Upsilon \ [\alpha X C, a_{j+1} \dots a_n)
\end{aligned}$$

where $B, C, D, E, X \in V_S$, $\alpha \in V_S^*$, $\Upsilon, \Upsilon_1 \in ([V_S^*)^*$. The two occurrences of α denote the same stack in the sense that α is neither consulted nor modified through derivation. These derivations are independent of Υ but not with respect to the subderivation

$$([D, a_{p+1} \dots a_n) \vdash^* ([\alpha E, a_{q+1} \dots a_n)$$

so they can be represented in compact form by items

$$[B, i, C, j, X \mid D, p, E, q]$$

To combine items, we use the following set of inference rules. Each rule is of the form $\frac{\eta_1, \dots, \eta_k}{\eta'}$ *trans*, meaning that if all antecedents η_i are present and there exists the transition *trans*, then the consequent item η' should be generated.

$$\frac{[B, i, C, j, X \mid D, p, E, q]}{[B, i, F, k, X \mid D, p, E, q]} \ C \overset{a}{\longmapsto} F$$

$$\frac{[B,i,C,j,X \mid D,p,E,q]}{[B,i,F,k,C \mid B,i,C,j]} \quad C \overset{a}{\longmapsto} CF$$

$$\frac{\begin{array}{c}[B,i,F,j,C \mid D,p,E,q] \\ [D,p,E,q,X' \mid O,u,P,v]\end{array}}{[B,i,G,k,X' \mid O,u,P,v]} \quad CF \overset{a}{\longmapsto} G$$

$$\frac{[B,i,C,j,X \mid D,p,E,q]}{[F,k,F,k,- \mid -,-,-,-]} \quad C \overset{a}{\longmapsto} C, \lceil F$$

$$\frac{\begin{array}{c}[F,k,F',k',- \mid -,-,-,-] \\ [B,i,C,j,X \mid D,p,E,q]\end{array}}{[B,i,G,l,X \mid D,p,E,q]} \quad \begin{array}{l} C \overset{a}{\longmapsto} C, \lceil F \\ C, \lceil F' \overset{b}{\longmapsto} G \end{array}$$

$$\frac{\begin{array}{c}[F,k,F',k',X \mid D,p,E,q] \\ [B,i,C,j,- \mid -,-,-,-]\end{array}}{[B,i,G,l,X \mid D,p,E,q]} \quad \begin{array}{l} C \overset{a}{\longmapsto} C, \lceil F \\ \lceil C, F' \overset{b}{\longmapsto} G \end{array}$$

where $k = j$ if $a = \epsilon$, $k = j+1$ if $a = a_{j+1}$, $l = k'$ if $b = \epsilon$ and $l = k'+1$ if $b = a_{k'+1}$.

Computation start with the initial item $[\$_0, 0, \$_0, 0, - \mid -,-,-,-]$. An input string $a_1 \ldots a_n$ has been recognized if the final item $[\$_0, 0, \$_f, n, - \mid -,-,-,-]$ is present. It can be proved that handling items with the inference rules is equivalent to applying the transitions on the whole stacks.

The space complexity of the proposed tabulation technique with respect to the length of the input string is $\mathcal{O}(n^4)$, due to every item stores four positions of the input string. The worst case time complexity is $\mathcal{O}(n^6)$.

7 Conclusion

We have provided a formal definition of bottom-up embedded push-down automata. We have also shown that finite-state control can be eliminated, obtaining a new definition in which transitions are in a form useful to describe compilation schemata for TAG and suitable for tabulation. The resulting definition has been shown to be equivalent to right-oriented linear indexed automata and a superset of bottom-up 2–stack automata with respect to the parsing strategies that can be described in both models of automata.

References

[1] Miguel A. Alonso, Eric de la Clergerie, and Manuel Vilares. Automata-based parsing in dynamic programming for Linear Indexed Grammars. In A. S. Narin'yani, editor, *Proc. of DIALOGUE'97 Computational Linguistics and its Applications International Workshop*, pages 22–27, Moscow, Russia, June 1997.

[2] Miguel A. Alonso, Eric de la Clergerie, and Manuel Vilares. A redefinition of Embedded Push-Down Automata. In *Proc. of 5th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+5)*, pages 19–26, Paris, France, May 2000.

[3] Miguel A. Alonso, Mark-Jan Nederhof, and Eric de la Clergerie. Tabulation of automata for tree adjoining languages. *Grammars*, Forthcoming.

[4] 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.

[5] Eric de la Clergerie, Miguel A. Alonso, and David Cabrero Souto. A tabular interpretation of bottom-up automata for TAG. In *Proc. of Fourth International Workshop on Tree-Adjoining Grammars and Related Frameworks (TAG+4)*, pages 42–45, Philadelphia, PA, USA, August 1998.

[6] Eric de la Clergerie and François Barthélemy. Information flow in tabular interpretations for generalized Push-Down Automata. *Theoretical Computer Science*, 199(1–2):167–198, 1998.

[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] Bernard Lang. Towards a uniform formal framework for parsing. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, pages 153–171. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[9] 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.

[10] Owen Rambow. *Formal and Computational Aspects of Natural Language Syntax*. PhD thesis, University of Pennsylvania, 1994. Available as IRCS Report 94-08 of the Institute of Research in Cognitive Science, University of Pennsylvania.

[11] Yves Schabes and K. Vijay-Shanker. Deterministic left to right parsing of tree adjoining languages. In *Proc. of 28th Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Oittsburgh, Pennsylvania, USA, June 1990. ACL.

[12] K. Vijay-Shanker. *A Study of Tree Adjoining Grammars*. PhD thesis, University of Pennsylvania, January 1988. Available as Technical Report MS-CIS-88-03 LINC LAB 95 of the Department of Computer and Information Science, University of Pennsylvania.