Tabulation of Automata for Tree-Adjoining Languages

Miguel A. Alonso Pardo
Universidade da Coruña (alonso@dc.fi.udc.es)

Mark-Jan Nederhof
DFKI (nederhof@dfki.de)

Eric Villemonte de la Clergerie
INRIA (eric.de_la_clergerie@inria.fr)

Abstract. We propose a modular design of tabular parsing algorithms for tree-adjoining languages. The modularity is made possible by a separation of the parsing strategy from the mechanism of tabulation. The parsing strategy is expressed in terms of the construction of a nondeterministic automaton from a grammar; three distinct types of automaton will be discussed. The mechanism of tabulation leads to the simulation of these nondeterministic automata in polynomial time, independent of the parsing strategy.

The proposed application of this work is the design of efficient parsing algorithms for tree-adjoining grammars and related formalisms.

Keywords: automata, linear indexed grammars, parsing, tabulation, tree-adjoining grammars

1. Introduction

Design of correct and efficient parsing algorithms for tree-adjoining grammars (TAGs) and equivalent formalisms such as linear indexed grammars (LIGs) is a difficult task. A promising way of simplifying this task is to apply well-known techniques from the realm of context-free parsing and logical programming, 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 (nondeterministic) pushdown automaton or a set of Horn clauses, and tabulation is introduced by means of some generic mechanism such as memoization. For example, if we choose the parsing strategy to be LR parsing (Sippu and Soisalon-Soininen, 1990) and construct a nondeterministic LR parser in the form of a pushdown automaton, then we may construct a *tabular* LR parser by applying the generic technique from (Lang, 1974) and (Billot and Lang, 1989), which allows tabulation of any pushdown automaton.

This modular way of constructing tabular algorithms has obvious advantages over direct constructions, as exemplified for tabular LR parsing by (Tomita, 1986). For example, it allows more straightforward proofs of correctness, is easier to understand and cheaper to implement.

The first modular approach to TAG parsing was proposed by (Lang, 1988b): a TAG is compiled into a logical pushdown automaton, which is interpreted by means of dynamic programming. However, it turns out that the chosen dynamic programming technique is too general for this particular task, and therefore requires fine-tuning in order to obtain an appropriate TAG parsing algorithm.

For further relevant work we refer to (Lang, 1994), who proposes to separate the parsing problem into the intersection of the grammar with an input and reduction of the resulting grammar.

The approach chosen in this paper relies on tabulation as originally devised for context-free parsing. We use three types of recognizer for tree-adjoining languages (TALs), which are notational variants of existing types such as (bottom-up) embedded pushdown automata (Schabes and Vijay-Shanker, 1990), 2-SA (Becker, 1994) and *bottom-up* 2-SA (de la Clergerie et al., 1998), SD-2SA (Villemonte de la Clergerie and Alonso Pardo, 1998), and $\mathcal{P}^2_{\mathrm{lin}}$-automata (Weir, 1994). The use of our notation simplifies the task of adapting tabulation techniques for context-free parsing to TALs.

We present the three types of recognizer as being closely related, and we show that together they allow a wide spectrum of different parsing strategies, including bottom-up, top-down and Earley strategies on two distinct levels. We do this by specifying schemata to compile LIGs into automata; altogether, 15 different schemata, representing 9 different parsing strategies, are warranted by our definitions.

We present the tabulation algorithms for each of the three types of automaton, and show that these algorithms are independent of the parsing strategy that was used to produce an automaton from a grammar.

The article may be outlined as follows: In Section 2 we recall the definition of linear indexed grammars. In Section 3 we present three types of recognizer, and Section 4 presents a number of ways to compile grammars into such recognizers. Tabulation of the recognizers and the correctness thereof are discussed in Sections 5 and 6. Section 7 presents final conclusions.

2. Linear indexed grammars

Tree-adjoining languages are by definition generated by tree-adjoining grammars (Joshi, 1987). The same class of languages is generated by linear indexed grammars (Gazdar, 1987), by head grammars, and by combinatory categorial grammars (Vijay-Shanker and Weir, 1994). This class is a strict subclass of mildly context-sensitive languages (Joshi et al., 1991).

The present paper concentrates on LIGs as representations of tree-adjoining languages, because they allow the simplest explanation of the principles underlying tabulation for tree-adjoining languages. Our results are however equally valid for any of the other representations mentioned above. (Note in particular that it has been demonstrated by (Vijay-Shanker and Weir, 1993b) that parse trees for a TAG can be computed through the computation of a parse forest for a LIG.)

We define a *linear indexed grammar* as a 5-tuple $(\Sigma, \mathcal{N}, S, \mathcal{I}, \mathcal{P})$, where Σ is a finite set of *terminals*, $\mathcal{N}$ is a finite set of *nonterminals*, $S \in \mathcal{N}$ is the *start symbol*, and $\mathcal{I}$ is a finite set of *indices*. We assume the sets Σ, $\mathcal{N}$ and $\mathcal{I}$ are pairwise disjoint, and none of them contains the symbols "[" or "]". Further, $\mathcal{P}$ is a finite set of *productions*, each being of one of the following forms:

- $A[\circ\circ\,\eta] \rightarrow \alpha\ B[\circ\circ\,\eta']\ \beta$, where $A, B \in \mathcal{N}$, $\eta, \eta' \in \mathcal{I} \cup \{\epsilon\}$ and either η or η' (or both) are ϵ, and α and β are lists of objects of the form $C[\,]$, where $C \in \mathcal{N}$; or

- $A[\,] \rightarrow z$, where $A \in \mathcal{N}$, and $z \in \Sigma \cup \{\epsilon\}$.

(In this paper, the empty string is denoted by ϵ.)

We now define the binary relation $\Rightarrow$ on elements from $(\Sigma \cup \mathcal{N} \cup \mathcal{I} \cup \{[,]\})^*$ as the least relation such that:

- $v\,A[\eta''\eta]\,w \ \Rightarrow\ v\,\alpha B[\eta''\eta']\beta\,w$, for any production $A[\circ\circ\eta] \rightarrow \alpha\ B[\circ\circ\eta']\ \beta$, any $v, w \in (\Sigma \cup \mathcal{N} \cup \mathcal{I} \cup \{[,]\})^*$, and any $\eta'' \in \mathcal{I}^*$; and

- $v\,A[\,]\,w \Rightarrow v\,z\,w$, for any production $A[\,] \rightarrow z$, and any $v, w \in (\Sigma \cup \mathcal{N} \cup \mathcal{I} \cup \{[,]\})^*$.

The transitive and reflexive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. The language generated by a LIG is now defined to be the set of all strings $w \in \Sigma^*$ such that $S[\,] \Rightarrow^* w$.

An example of a LIG is given by the following set of productions:

$$
\begin{array}{rrcl@{\qquad}rrcl}
(1) & S[\circ\circ] & \rightarrow & A[\,]\ S[\circ\circ p]\ D[\,] & (5) & A[\,] & \rightarrow & a \\
(2) & S[\circ\circ] & \rightarrow & Q[\circ\circ] & (6) & B[\,] & \rightarrow & b \\
(3) & Q[\circ\circ p] & \rightarrow & B[\,]\ Q[\circ\circ]\ C[\,] & (7) & C[\,] & \rightarrow & c \\
(4) & Q[\,] & \rightarrow & \epsilon & (8) & D[\,] & \rightarrow & d
\end{array}
$$

The language it generates is $\{a^n b^n c^n d^n \mid n \geq 0\}$. A parse tree for the string *aabbccdd* is given in Figure 1. The dotted arrows indicate how

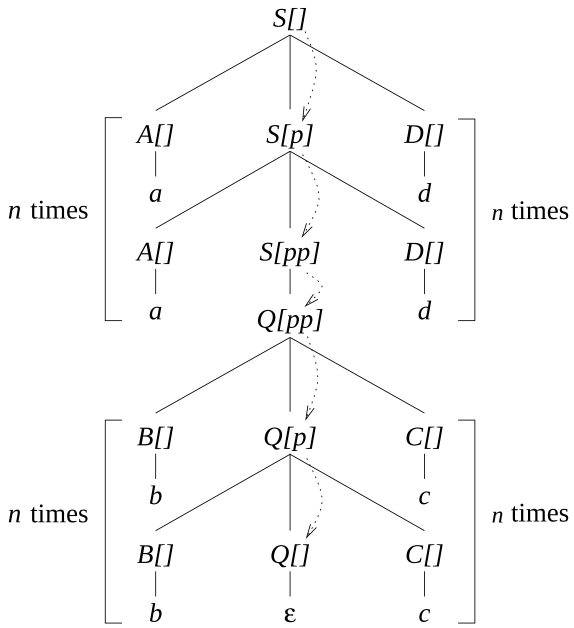

Figure 1. Parse tree for a LIG, with a spine along the dotted arrows.

lists of indices are passed from one node to the next. (The daughter node to which a list of indices is passed is called the *distinguished child* in (Vijay-Shanker and Weir, 1993).) A path along such dotted arrows will be called a *spine*. In general, there may be several such spines in a parse tree for a LIG.

In the running example, we may decide whether a given string of a's, b's, c's and d's, in this order, is in the language if we verify that the numbers of a's, b's, c's and d's are equal. This may be done by dividing the problem into smaller tasks as follows. First we may verify the following condition: 1) for each a there is a corresponding d and vice versa, and for each b there is a corresponding c and vice versa. In addition, we should verify either 2) for each a there is a corresponding b and vice versa, or 2') for each c there is a corresponding d and vice versa. By virtue of the first condition, condition 2) ensures that also c's and d's occur in equal numbers, and conversely, condition 2') ensures that also a's and b's occur in equal numbers. More precisely, 1) and 2) together imply 2'), and 1) and 2') together imply 2).

The first condition can be checked by a traditional pushdown automaton, which would push symbols on its stack for the a's and b's it finds on the left side of the spine, and which would then pop the corresponding c's and d's it may find on the right side of the spine. The three types of automaton we will discuss in the sequel would all verify the second condition by means of a refinement of the pushdown model used for verifying the first condition. In this refinement, lists of indices are manipulated as stacks embedded within the main stack. (We will mostly abstain from referring to lists of indices as stacks because of the obvious confusion ensuing from this.)

Intuitively, the respective types of automaton differ in the choice whether condition 2) is checked, on the left side of the spine, or whether condition 2') is checked, on the right side, or whether both 2) and 2') are checked. Although the latter case in this simple example entails redundant computation, in general the corresponding type of automaton allows parsing strategies that cannot be expressed by any of the two other types.

3. Linear indexed automata

We discuss three types of recognizer that are equivalent to linear indexed grammars and that are based on the well-known pushdown automata, which are equivalent to context-free grammars. The rationale is that LIGs are nothing more than context-free grammars extended with parameters in the form of lists of indices, and therefore pushdown automata extended with the same kind of parameter suffice to built recognizers for languages generated by LIGs. All types of automaton we will discuss here can be seen as restricted forms of indexed pushdown automata (Parchmann et al., 1980), which are themselves restricted forms of logical pushdown automata (Lang, 1988a).

Each automaton of one of the types is a 7-tuple $(\Sigma, \Gamma_L, \Gamma_R, I, J, \mathcal{I}, \mathcal{T})$, where Σ is a finite set of terminals as before, Γ_L and Γ_R are two disjoin finite sets of *stack symbols*, $I \in \Gamma_L$ is the *initial stack symbol*, $J \in \Gamma_R$ is the *final stack symbol*, $\mathcal{I}$ is a finite set of indices as before, and $\mathcal{T}$ is a finite set of *transitions*. We denote the set of all stack symbols by $\Gamma = \Gamma_L \cup \Gamma_R$.

A transition has the form $\alpha \overset{z}{\mapsto} \beta$, where $z \in \Sigma \cup \{\epsilon\}$, and α and β are lists of objects of the form $A[\,]$ or $A[\circ\circ\,\eta]$, where $A \in \Gamma$ and $\eta \in \mathcal{I} \cup \{\epsilon\}$. The three types of automaton differ in the allowable kinds of transition.

We define a *configuration* to be an element of $(\Gamma \times \mathcal{I}^*)^* \times \Sigma^*$. Each element from $\Gamma \times \mathcal{I}^*$ contains a stack symbol and a list of indices. A list of such elements from $\Gamma \times \mathcal{I}^*$ represents the stack of the automaton. The stack is constructed from left to right, i.e. the bottom element will be represented as the leftmost element. An element from Σ^* represents the remaining suffix of the input v after a certain number of symbols at the left end have been consumed.

We define the binary relation $\vdash$ between configurations as follows. We have $(\alpha\beta, zw) \vdash (\alpha\gamma, w)$ if and only if there exists some transition $\beta' \overset{z}{\mapsto} \gamma'$ in $\mathcal{T}$ and a list of indices $\eta \in \mathcal{I}^*$ so that, by consistently substituting all occurrences of $\circ\circ$ in the transition by η, we create β and γ out of β' and γ', respectively. The transitive and reflexive closure of $\vdash$ is denoted by $\vdash^*$.

Some input v is *recognized* if $(I[\,], v) \vdash^* (J[\,], \epsilon)$. The language *accepted* by an automaton M is defined to be the set of all v that are recognized.

Conceptually, the first type of automaton we define manipulates the indices both on the left and right sides of a spine. It has no left-to-right or right-to-left bias, and, in this sense, is "universal", as opposed to two other types of automaton to be discussed later that are specialized to one or the other direction.

Thus, a *universal linear indexed automaton* (U-LIA) is a 7-tuple as explained above, and each transition should be of one of the following forms:

- $A[\,] \overset{z}{\mapsto} B[\,]$, where $A \in \Gamma_L$ and $B \in \Gamma_R$;
- $A[\circ\circ\,\eta] \overset{\epsilon}{\mapsto} B[\,]\ C[\circ\circ\,\eta']$, where $A, C \in \Gamma_L$;
- $B[\,]\ C[\circ\circ\,\eta] \overset{\epsilon}{\mapsto} A[\circ\circ\,\eta']$, where $C, A \in \Gamma_R$;
- $A[\circ\circ] \overset{\epsilon}{\mapsto} B[\circ\circ]\ C[\,]$, where $A, B \in \Gamma_d$, for some $d \in \{L, R\}$, and $C \in \Gamma_L$; or
- $B[\circ\circ]\ C[\,] \overset{\epsilon}{\mapsto} A[\circ\circ]$, where $B, A \in \Gamma_d$, for some $d \in \{L, R\}$, and $C \in \Gamma_R$,

and for transitions of the second and third forms, either η or η' (or both) must be the empty string. We further impose the following restriction: For each pair of transitions of the form $A_1[X_1] \overset{\epsilon}{\mapsto} B[Y_1]\ C_1[Z_1]$ and $B[Y_2]\ C_2[Z_2] \overset{\epsilon}{\mapsto} A_2[X_2]$ from $\mathcal{T}$, one of these two conditions must hold:

1. $Y_1 = Y_2 = \epsilon$, $|X_1| = |X_2|$, and $|Z_1| = |Z_2|$, or
2. $X_1 = Y_1 = Y_2 = X_2 = \circ\circ$, $Z_1 = Z_2 = \epsilon$.

This restriction ensures that a pushing transition of the second type can only be "coupled" to a popping transition of the third type (case 1 in the restriction), and similarly, that a transition of the fourth type can only be coupled to a transition of the fifth type (case 2). More precisely, if the stack height has increased by one element by means of a certain type of pushing transition, then we know which type of popping transition will be used later on to decrease it again to the former height.

Furthermore, the restriction in the first case ensures that the length of the list of indices associated to symbol A_1 is identical to that associated to A_2, by virtue of the same restriction imposed recursively on C_1 and C_2, and due to the fact that the change of the length of the list of indices caused by a choice of X_1 and Z_1 is the reverse of the change caused by a choice of Z_2 and X_2, since $|X_1| = |X_2|$ and $|Z_1| = |Z_2|$.

Note that here, as in the two other types of automaton to be discussed in the sequel, the stack symbols in Γ_L allow lists of indices to be carried to higher regions of the stack, whereas those in Γ_R take lists of indices to lower regions of the stack. Typically, when we build a U-LIA from a LIG, this automaton will use symbols from Γ_L at the left side of a spine and symbols from Γ_R at the right side.

For comparison with existing types of automaton, we introduce two variants of U-LIA, called R-LIA and L-LIA. Conceptually, an R-LIA manipulates lists of indices only on the right side of a spine, and an L-LIA manipulates them only on the left side. On a technical level, either the second type of transition of U-LIA is changed so that all lists of indices are empty, which yields R-LIA, or the third type of transition is changed in the same way, which yields L-LIA.

Thus, a *right-oriented linear indexed automaton* (R-LIA) is a 7-tuple as before, where each transition is of one of the following forms:

- $A[\,] \overset{z}{\mapsto} B[\,]$, where $A \in \Gamma_L$ and $B \in \Gamma_R$;
- $A[\,] \overset{\epsilon}{\mapsto} B[\,]\ C[\,]$, where $A, C \in \Gamma_L$;
- $B[\,]\ C[\circ\circ\,\eta] \overset{\epsilon}{\mapsto} A[\circ\circ\,\eta']$, where $C, A \in \Gamma_R$;
- $A[\circ\circ] \overset{\epsilon}{\mapsto} B[\circ\circ]\ C[\,]$, where $A, B \in \Gamma_d$, for some $d \in \{L, R\}$, and $C \in \Gamma_L$; or
- $B[\circ\circ]\ C[\,] \overset{\epsilon}{\mapsto} A[\circ\circ]$, where $B, A \in \Gamma_d$, for some $d \in \{L, R\}$, and $C \in \Gamma_R$,

with the same restrictions as before on η and η'. We further impose the following restriction: For each pair of transitions of the form $A_1[X_1] \overset{\epsilon}{\mapsto} B[Y_1]\ C_1[Z_1]$ and $B[Y_2]\ C_2[Z_2] \overset{\epsilon}{\mapsto} A_2[X_2]$ from $\mathcal{T}$, one of these two conditions must hold:

1. $Y_1 = Y_2 = \epsilon$, or
2. $X_1 = Y_1 = Y_2 = X_2 = \circ\circ$ and $Z_1 = Z_2 = \epsilon$.

This restriction on transitions is not essential, but reduces the number of cases to be considered for tabulation, in Section 5. Actually, transitions of the second type are redundant (see (Nederhof, 1998) for an alternative definition of R-LIA without them). Furthermore, the distinction between Γ_L and Γ_R is unnecessary. The definition of R-LIA in this paper however is chosen to stress the relation to U-LIA.

Note that, in an R-LIA, a (non-empty) list of indices cannot be carried to higher regions of the stack. The converse holds for the dual type of automaton: in an L-LIA, lists of indices cannot be carried to lower regions.

Thus, a *left-oriented linear indexed automaton* (L-LIA) is a 7-tuple as before, where each transition is of one of the following forms:

- $A[\,] \overset{z}{\longmapsto} B[\,]$, where $A \in \Gamma_L$ and $B \in \Gamma_R$;
- $A[\circ\circ\,\eta] \overset{\epsilon}{\longmapsto} B[\,]\ C[\circ\circ\,\eta']$, where $A, C \in \Gamma_L$;
- $B[\,]\ C[\,] \overset{\epsilon}{\longmapsto} A[\,]$, where $C, A \in \Gamma_R$;
- $A[\circ\circ] \overset{\epsilon}{\longmapsto} B[\circ\circ]\ C[\,]$, where $A, B \in \Gamma_d$, for some $d \in \{L, R\}$, and $C \in \Gamma_L$; or
- $B[\circ\circ]\ C[\,] \overset{\epsilon}{\longmapsto} A[\circ\circ]$, where $B, A \in \Gamma_d$, for some $d \in \{L, R\}$, and $C \in \Gamma_R$,

with the same restrictions as before on η and η', and the same restriction on the "coupling" of pairs of transitions as for R-LIA.

U-LIA correspond to SD-2SA (Villemonte de la Clergerie and Alonso Pardo, 1998), in which modes and marks have the same role as our two disjoint set Γ_L and Γ_R and the restrictions on the set of transitions; in both cases the motivation is to reduce the class of languages accepted by the automaton models to precisely the class of TALs. L-LIAs correspond to embedded pushdown automata (Schabes and Vijay-Shanker, 1990) and 2-SA (Becker, 1994), whereas R-LIAs correspond to *bottom-up* embedded pushdown automata (Schabes and Vijay-Shanker, 1990) and *bottom-up* 2-SA (de la Clergerie et al., 1998). The difference between our types of automaton and those presented in existing literature is restricted to notation. However, the chosen notation will simplify the development of tabulation techniques, to be discussed in Section 5.

Table I. Generic compilation schema

Rule	Transition	Production
[INIT]	$I[\circ\circ] \overset{\epsilon}{\longmapsto} I[\,]\ \overrightarrow{S}[\circ\circ]$	
[CALL]	$\nabla_{r,s}[\circ\circ] \overset{\epsilon}{\longmapsto} \nabla_{r,s}[\circ\circ]\ \overrightarrow{A_{r,s+1}}[\,]$	$X_{r,s+1}{=}A_{r,s+1}[]$
[SCALL]	$\nabla_{r,s}[\overrightarrow{\circ\circ\,\eta}] \overset{\epsilon}{\longmapsto} \nabla_{r,s}[\,]\ \overrightarrow{A_{r,s+1}}[\overrightarrow{\circ\circ\,\eta'}]$	$X_{r,0}{=}A_{r,0}[\circ\circ\,\eta]$,$X_{r,s+1}{=}A_{r,s+1}[\circ\circ\,\eta']$
[SEL]	$\overrightarrow{A_{r,0}}[\circ\circ] \overset{\epsilon}{\longmapsto} \overrightarrow{A_{r,0}}[\,]\ \nabla_{r,0}[\circ\circ]$	
[PUB]	$\overrightarrow{A_{r,0}}[\,]\ \nabla_{r,n_r}[\circ\circ] \overset{\epsilon}{\longmapsto} \overleftarrow{A_{r,0}}[\circ\circ]$	
[RET]	$\nabla_{r,s}[\circ\circ]\ \ \overleftarrow{A_{r,s+1}}[\,] \overset{\epsilon}{\longmapsto} \nabla_{r,s+1}[\circ\circ]$	$X_{r,s+1}{=}A_{r,s+1}[]$
[SRET]	$\nabla_{r,s}[\,]\ \overleftarrow{A_{r,s+1}}[\overleftarrow{\circ\circ\,\eta'}] \overset{\epsilon}{\longmapsto} \nabla_{r,s+1}[\overleftarrow{\circ\circ\,\eta}]$	$X_{r,0}{=}A_{r,0}[\circ\circ\,\eta]$,$X_{r,s+1}{=}A_{r,s+1}[\circ\circ\,\eta']$
[SCAN]	$\nabla_{r,0}[\,] \overset{z}{\longmapsto}\ \nabla_{r,n_r}[\,]$	$X_{r,1}{=}z$
[FINAL]	$I[\,]\ \overleftarrow{S}[\circ\circ] \overset{\epsilon}{\longmapsto} J[\circ\circ]$	

4. Compilation schemata for LIGs

A compilation schema translates the set of productions of a source grammar into a set of transitions of a given class of automata. Table I shows a generic compilation schema transforming a linear indexed grammar into a set of transitions of a linear indexed automaton, where $\overrightarrow{X}$ and $\overleftarrow{X}$ denote the information propagated top-down and bottom-up, respectively, with respect to X, which can be a nonterminal or an expression pertaining to a list of indices. Each row presents one compilation rule: the first column contains its name, the second column is the resulting transition and the third contains conditions on the form that productions must have to be considered by a rule.

For a production r with nonterminals in the right-hand side, we define n_r as the number of those nonterminals, and let $X_{r,0}, \ldots, X_{n_r}$ and $A_{r,0}, \ldots, A_{n_r} \in \mathcal{N}$ to be such that production r can be written as $X_{r,0} \to X_{r,1} \cdots X_{r,n_r}$, where each $X_{r,i}$ $(0 \leq i \leq n_r)$ is of the form $A_{r,i}[]$ or $A_{r,i}[\circ\circ\,\eta]$. For a production r without nonterminals in the right-hand side, we define $n_r = 1$ and let $X_{r,0}, X_{r,1}$ and $A_{r,0}$ be such that production r can be written as $X_{r,0}[] \to X_{r,1}$, where $X_{r,0} = A_{r,0}[]$ and $X_{r,1} \in \Sigma \cup \{\epsilon\}$. For each production, the automaton contains the symbols $\nabla_{r,0}, \ldots, \nabla_{r,n_r}$. A symbol $\nabla_{r,j}$ is used to indicate that the part $X_{r,1} \cdots X_{r,j}$ of a production r has been recognized.

The function of the transitions produced by the compilation rules is as follows. The **[INIT]** transition is used to start the computation; **[CALL]** transitions are used to call a nonterminal that does

Table II. Parameters for the context-free strategy

Context-free strategy	$\overrightarrow{A_{r,s+1}}$	$\overleftarrow{A_{r,s+1}}$
Bottom-up	$\Box$	$A_{r,s+1}$
Top-down	$A_{r,s+1}$	$\Box$
Earley	$\overline{A_{r,s+1}}$	$\overline{\overline{A_{r,s+1}}}$

Table III. Parameters for the indices strategy

	R-LIA		L-LIA		U-LIA	
indices strategy	$\overrightarrow{\circ\circ\eta}$	$\overleftarrow{\circ\circ\eta}$	$\overrightarrow{\circ\circ\eta}$	$\overleftarrow{\circ\circ\eta}$	$\overrightarrow{\circ\circ\eta}$	$\overleftarrow{\circ\circ\eta}$
Bottom-up	ϵ	$\circ\circ\eta$			$\circ\circ\Diamond$	$\circ\circ\eta$
Top-down			$\circ\circ\eta$	ϵ	$\circ\circ\eta$	$\circ\circ\Diamond$
Earley					$\circ\circ\eta$	$\circ\circ\eta$

not represent a distinguished child; [**SCALL**] (*spine call*) transitions call a nonterminal representing a distinguished child; [**SEL**] transitions select a production; [**PUB**] transitions finish the parsing of a production; [**RET**] transitions continue the parsing process after a non-distinguished child has been processed; [**SRET**] (*spine return*) transitions continue the parsing process after a distinguished child has been processed; [**SCAN**] transitions are used to match a terminal z in a rule r of the form $A[] \to z$ to a symbol from the input (or to merely recognize the empty string when $z = \epsilon$); and the [**FINAL**] transition is used to finish the computation.

Parsing strategies may be specified using a pair of strategies controlling the flow of information, the first one (*context-free strategy*) dealing with nonterminals, while the other (*indices strategy*) deals with the indices. Table II shows the context-free strategies obtained by different instantiations of $\overrightarrow{A}$ and $\overleftarrow{A}$ in the generic compilation schema above, where $\Box$ is a fresh stack symbol. If a nonterminal A is propagated both top-down and bottom-up, we distinguish the two cases by the stack symbols $\overline{A}$ and $\overline{\overline{A}}$, respectively. The different indices strategies obtained by instantiating $\overrightarrow{\circ\circ\eta}$ and $\overleftarrow{\circ\circ\eta}$ are shown in Table III, where $\Diamond$ is a fresh index.

All 9 combinations of the 3 context-free parsing strategies from Table II and the 3 indices strategies from Table III can be implemented in U-LIA, as this class of automata does not restrict the information that can be propagated in either direction. The stack symbols are divided

Table IV. Compilation schema for R-LIA

Rule	Transition	Production
[INIT]	$I[\circ\circ] \overset{\epsilon}{\mapsto} I[\,]\ \overrightarrow{S}[\circ\circ]$	
[CALL]	$\nabla_{r,s}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{r,s}[\circ\circ]\ \overrightarrow{A_{r,s+1}}[\,]$	$X_{r,s+1}{=}A_{r,s+1}[]$
[SCALL]	$\nabla_{r,s}[\,] \overset{\epsilon}{\mapsto} \nabla_{r,s}[\,]\ \overrightarrow{A_{r,s+1}}[\,]$	$X_{r,0}{=}A_{r,0}[\circ\circ\,\eta], X_{r,s+1}{=}A_{r,s+1}[\circ\circ\,\eta']$
[SEL]	$\overrightarrow{A_{r,0}}[\circ\circ] \overset{\epsilon}{\mapsto} \overrightarrow{A_{r,0}}[\,]\ \nabla_{r,0}[\circ\circ]$	
[PUB]	$\overrightarrow{A_{r,0}}[\,]\ \nabla_{r,n_r}[\circ\circ] \overset{\epsilon}{\mapsto} \overleftarrow{A_{r,0}}[\circ\circ]$	
[RET]	$\nabla_{r,s}[\circ\circ]\ \overleftarrow{A_{r,s+1}}[\,] \overset{\epsilon}{\mapsto} \nabla_{r,s+1}[\circ\circ]$	$X_{r,s+1}{=}A_{r,s+1}[]$
[SRET]	$\nabla_{r,s}[\,]\ \overleftarrow{A_{r,s+1}}[\circ\circ\,\eta'] \overset{\epsilon}{\mapsto} \nabla_{r,s+1}[\circ\circ\,\eta]$	$X_{r,0}{=}A_{r,0}[\circ\circ\,\eta], X_{r,s+1}{=}A_{r,s+1}[\circ\circ\,\eta']$
[SCAN]	$\nabla_{r,0}[\,] \overset{z}{\mapsto} \nabla_{r,n_r}[\,]$	$X_{r,1}{=}z$
[FINAL]	$I[\,]\ \overleftarrow{S}[\circ\circ] \overset{\epsilon}{\mapsto} J[\circ\circ]$	

in two sets as follows. For every nonterminal A, we have $\overrightarrow{A} \in \Gamma_L$ and $\overleftarrow{A} \in \Gamma_R$, and for each production

$$A_{r,0}[\circ\circ\,\eta] \to A_{r,1}[\,] \cdots A_{r,i-1}[\,]\ A_{r,i}[\circ\circ\,\eta']\ A_{r,i+1}[\,] \cdots A_{r,n_r}[\,]$$

we have $\nabla_{r,j} \in \Gamma_L$ for all $j < i$ and $\nabla_{r,j} \in \Gamma_R$ for all $j \geq i$. and for each rule r of the form $A[] \to z$ we have $\nabla_{r,0} \in \Gamma_L$ and $\nabla_{r,1} \in \Gamma_R$. It can easily be shown that the requirements imposed on the set of transitions for U-LIA in Section 3 are then satisfied.

In the case of a R-LIA, top-down propagation of indices is not allowed, and therefore the generic compilation schema of Table I is constrained to parsing strategies incorporating a bottom-up indices strategy. With this limitation, we can compile a LIG into a R-LIA according to Table IV, where Γ_L and Γ_R are as before. This generic compilation schema still allows a choice of one of the three context-free strategies.

Conversely, L-LIA does not allow bottom-up propagation of indices. With this restriction, Table I can be specialized to L-LIA, with a top-down indices strategy, resulting in Table V.

To illustrate the compilation schemata, we show the compilation of the linear indexed grammar presented in Section 2 into a U-LIA, by an Earley strategy both on the context-free level and on the level of indices. Table VI presents the set of transitions. Table VII shows a sequence of steps of the automaton in the recognition of the input string *abcd*. The first column lists the transitions that were applied to

Table V. Compilation schema for L-LIA

Rule	Transition	Production
[INIT]	$I[\circ\circ] \overset{\epsilon}{\mapsto} I[\,]\ \overrightarrow{S}[\circ\circ]$	
[CALL]	$\nabla_{r,s}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{r,s}[\circ\circ]\ \overrightarrow{A_{r,s+1}}[\,]$	$X_{r,s+1}{=}A_{r,s+1}[]$
[SCALL]	$\nabla_{r,s}[\circ\circ\,\eta] \overset{\epsilon}{\mapsto} \nabla_{r,s}[\,]\ \overrightarrow{A_{r,s+1}}[\circ\circ\,\eta']$	$X_{r,0}{=}A_{r,0}[\circ\circ\,\eta], X_{r,s+1}{=}A_{r,s+1}[\circ\circ\,\eta']$
[SEL]	$\overrightarrow{A_{r,0}}[\circ\circ] \overset{\epsilon}{\mapsto} \overrightarrow{A_{r,0}}[\,]\ \nabla_{r,0}[\circ\circ]$	
[PUB]	$\overrightarrow{A_{r,0}}[\,]\ \nabla_{r,n_r}[\circ\circ] \overset{\epsilon}{\mapsto} \overleftarrow{A_{r,0}}[\circ\circ]$	
[RET]	$\nabla_{r,s}[\circ\circ]\ \overleftarrow{A_{r,s+1}}[\,] \overset{\epsilon}{\mapsto} \nabla_{r,s+1}[\circ\circ]$	$X_{r,s+1}{=}A_{r,s+1}[]$
[SRET]	$\nabla_{r,s}[\,]\ \overleftarrow{A_{r,s+1}}[\,] \overset{\epsilon}{\mapsto} \nabla_{r,s+1}[\,]$	$X_{r,0}{=}A_{r,0}[\circ\circ\,\eta], X_{r,s+1}{=}A_{r,s+1}[\circ\circ\,\eta']$
[SCAN]	$\nabla_{r,0}[\,] \overset{z}{\mapsto} \nabla_{r,n_r}[\,]$	$X_{r,1}{=}z$
[FINAL]	$I[\,]\ \overleftarrow{S}[\circ\circ] \overset{\epsilon}{\mapsto} J[\circ\circ]$	

obtain the stacks in the second column, with the remainders of the input string in the third column.

5. Tabulation

The automata we have defined in Section 3 manipulate stacks on two levels, since the lists of indices act as stacks embedded within the main stack. This observation allows us to apply, in a generalized way, a tabulation technique originally devised for pushdown automata with only one kind of stack, as presented in (Lang, 1974; Billot and Lang, 1989).

The essence of the tabulation algorithm is described as follows. Given a linear indexed automaton and an input $v = a_1 \cdots a_n \in \Sigma^*$, for some $n \geq 0$, we construct a table U in polynomial time. From the presence of a certain object in U, we can effectively decide whether the input is a string in the language. The procedure can be extended so that a representation of all parse trees, the *parse forest*, is produced as a side-effect of the construction of U. In addition, the procedure can be extended to handle feature structures, analogously to the extension of tabular context-free parsing to handle logic terms (Villemonte de la Clergerie and Barthélemy, 1998).

The objects in the table U will be called *items*, and indicate the existence of certain computations of the automata that may be performed between certain positions in the input v and that involve local

Table VI. Transitions of the example U-LIA

Rule	Transition
(0a) **[INIT]**	$I[\circ\circ] \mapsto I[\,]\ \overline{S}[\circ\circ]$
(1a) **[SEL]**	$\overline{S}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{S}[\,]\ \nabla_{1,0}[\circ\circ]$
(1b) **[CALL]**	$\nabla_{1,0}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{1,0}[\circ\circ]\ \overline{A}[\,]$
(1c) **[RET]**	$\nabla_{1,0}[\circ\circ]\ \overline{\overline{A}}[\,] \overset{\epsilon}{\mapsto} \nabla_{1,1}[\circ\circ]$
(1d) **[SCALL]**	$\nabla_{1,1}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{1,1}[\,]\ \overline{S}[\circ\circ p]$
(1e) **[SRET]**	$\nabla_{1,1}[\,]\ \overline{\overline{S}}[\circ\circ p] \overset{\epsilon}{\mapsto} \nabla_{1,2}[\circ\circ]$
(1f) **[CALL]**	$\nabla_{1,2}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{1,2}[\circ\circ]\ \overline{D}[\,]$
(1g) **[RET]**	$\nabla_{1,2}[\circ\circ]\ \overline{D}[\,] \overset{\epsilon}{\mapsto} \nabla_{1,3}[\circ\circ]$
(1h) **[PUB]**	$\overline{S}[\,]\ \nabla_{1,3}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{\overline{S}}[\circ\circ]$
(2a) **[SEL]**	$\overline{S}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{S}[\,]\ \nabla_{2,0}[\circ\circ]$
(2b) **[SCALL]**	$\nabla_{2,0}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{2,0}[\,]\ \overline{Q}[\circ\circ]$
(2c) **[SRET]**	$\nabla_{2,0}[\,]\ \overline{\overline{Q}}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{2,1}[\circ\circ]$
(2d) **[PUB]**	$\overline{S}[\,]\ \nabla_{2,1}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{\overline{S}}[\circ\circ]$
(3a) **[SEL]**	$\overline{Q}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{Q}[\,]\ \nabla_{3,0}[\circ\circ]$
(3b) **[CALL]**	$\nabla_{3,0}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{3,0}[\circ\circ]\ \overline{B}[\,]$
(3c) **[RET]**	$\nabla_{3,0}[\circ\circ]\ \overline{\overline{B}}[\,] \overset{\epsilon}{\mapsto} \nabla_{3,1}[\circ\circ]$
(3d) **[SCALL]**	$\nabla_{3,1}[\circ\circ p] \overset{\epsilon}{\mapsto} \nabla_{3,1}[\,]\ \overline{Q}[\circ\circ]$
(3e) **[SRET]**	$\nabla_{3,1}[\,]\ \overline{\overline{Q}}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{3,2}[\circ\circ p]$
(3f) **[CALL]**	$\nabla_{3,2}[\circ\circ] \overset{\epsilon}{\mapsto} \nabla_{3,2}[\circ\circ]\ \overline{C}[\,]$
(3g) **[RET]**	$\nabla_{3,2}[\circ\circ]\ \overline{\overline{C}}[\,] \overset{\epsilon}{\mapsto} \nabla_{3,3}[\circ\circ]$
(3h) **[PUB]**	$\overline{Q}[\,]\ \nabla_{3,3}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{\overline{Q}}[\circ\circ]$
(4a) **[SEL]**	$\overline{Q}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{Q}[\,]\ \nabla_{4,0}[\circ\circ]$
(4b) **[SCAN]**	$\nabla_{4,0}[\,] \overset{\epsilon}{\mapsto} \nabla_{4,1}[\,]$
(4c) **[PUB]**	$\overline{Q}[\,]\ \nabla_{4,1}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{\overline{Q}}[\circ\circ]$
(5a) **[SEL]**	$\overline{A}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{A}[\,]\ \nabla_{5,0}[\circ\circ]$
(5b) **[SCAN]**	$\nabla_{5,0}[\,] \overset{a}{\mapsto} \nabla_{5,1}[\,]$
(5c) **[PUB]**	$\overline{A}[\,]\ \nabla_{5,1}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{\overline{A}}[\circ\circ]$
(6a) **[SEL]**	$\overline{B}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{B}[\,]\ \nabla_{6,0}[\circ\circ]$
(6b) **[SCAN]**	$\nabla_{6,0}[\,] \overset{b}{\mapsto} \nabla_{6,1}[\,]$
(6c) **[PUB]**	$\overline{B}[\,]\ \nabla_{6,1}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{\overline{B}}[\circ\circ]$
(7a) **[SEL]**	$\overline{C}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{C}[\,]\ \nabla_{7,0}[\circ\circ]$
(7b) **[SCAN]**	$\nabla_{7,0}[\,] \overset{c}{\mapsto} \nabla_{7,1}[\,]$
(7c) **[PUB]**	$\overline{C}[\,]\ \nabla_{7,1}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{\overline{C}}[\circ\circ]$
(8a) **[SEL]**	$\overline{D}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{D}[\,]\ \nabla_{8,0}[\circ\circ]$
(8b) **[SCAN]**	$\nabla_{8,0}[\,] \overset{d}{\mapsto} \nabla_{8,1}[\,]$
(8c) **[PUB]**	$\overline{D}[\,]\ \nabla_{8,1}[\circ\circ] \overset{\epsilon}{\mapsto} \overline{\overline{D}}[\circ\circ]$
(0b) **[FINAL]**	$I[\,]\ \overline{\overline{S}}[\circ\circ] \overset{\epsilon}{\mapsto} J[\circ\circ]$

changes in the (main) stack and in the lists of indices. The form of items is slightly different for the different types of automaton.

5.1. Tabulation for U-LIA

For the tabulation of U-LIA, we need two types of item. The simpler of the two has the form $[H, h \mid A, B, i, j \mid p]$, where $H, A, B \in \Gamma_L$, $h, i, j \in \{0, 1, \ldots, n\}$ and $p \in \mathcal{I}$, and will be referred to as a *call item*. We need this item to keep track of a list of indices while the main stack grows in size and the list of indices is manipulated and carried to higher regions of the stack; see Figure 2 for a pictorial representation.

Table VII. Configurations during the recognition of *abcd*

trans.	stack	string
	$I[\,]$	*abcd*
(0a)	$I[\,]\ \overline{S}_{0,0}[\,]$	*abcd*
(1a)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,0}[\,]$	*abcd*
(1b)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,0}[\,]\ \overline{A}[\,]$	*abcd*
(5a)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,0}[\,]\ \overline{A}[\,]\ \nabla_{5,0}[\,]$	*abcd*
(5b)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,0}[\,]\ \overline{A}[\,]\ \nabla_{5,1}[\,]$	*bcd*
(5c)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,0}[\,]\ \overline{\overline{A}}[\,]$	*bcd*
(1c)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]$	*bcd*
(1d)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[p]$	*bcd*
(2a)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[p]$	*bcd*
(2b)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[p]$	*bcd*
(3a)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,0}[p]$	*bcd*
(3b)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,0}[p]\ \overline{B}[\,]$	*bcd*
(6a)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,0}[p]\ \overline{B}[\,]\ \nabla_{6,0}[\,]$	*bcd*
(6b)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,0}[p]\ \overline{B}[\,]\ \nabla_{6,1}[\,]$	*cd*
(6c)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,0}[p]\ \overline{\overline{B}}[\,]$	*cd*
(3c)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,1}[p]$	*cd*
(3d)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,1}[\,]\ \overline{Q}[\,]$	*cd*
(4a)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,1}[\,]\ \overline{Q}[\,]\ \nabla_{4,0}[\,]$	*cd*
(4b)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,1}[\,]\ \overline{Q}[\,]\ \nabla_{4,1}[\,]$	*cd*
(4c)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,1}[\,]\ \overline{\overline{Q}}[\,]$	*cd*
(3e)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,2}[p]$	*cd*
(3f)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,2}[p]\ \overline{C}[\,]$	*cd*
(7a)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,2}[p]\ \overline{C}[\,]\ \nabla_{7,0}[\,]$	*cd*
(7b)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,2}[p]\ \overline{C}[\,]\ \nabla_{7,1}[\,]$	*d*
(7c)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,2}[p]\ \overline{\overline{C}}[\,]$	*d*
(3g)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{Q}[\,]\ \nabla_{3,3}[p]$	*d*
(3h)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,0}[\,]\ \overline{\overline{Q}}[p]$	*d*
(2c)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{S}[\,]\ \nabla_{2,1}[p]$	*d*
(2d)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,1}[\,]\ \overline{\overline{S}}[p]$	*d*
(1e)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,2}[\,]$	*d*
(1f)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,2}[\,]\ \overline{D}[\,]$	*d*
(8a)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,2}[\,]\ \overline{D}[\,]\ \nabla_{8,0}[\,]$	*d*
(8b)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,2}[\,]\ \overline{D}[\,]\ \nabla_{8,1}[\,]$	
(8c)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,2}[\,]\ \overline{\overline{D}}[\,]$	
(1g)	$I[\,]\ \overline{S}_{0,0}[\,]\ \nabla_{1,3}[\,]$	
(1h)	$I[\,]\ \overline{\overline{S}}_{0,0}[\,]$	
(0b)	$J[\,]$	

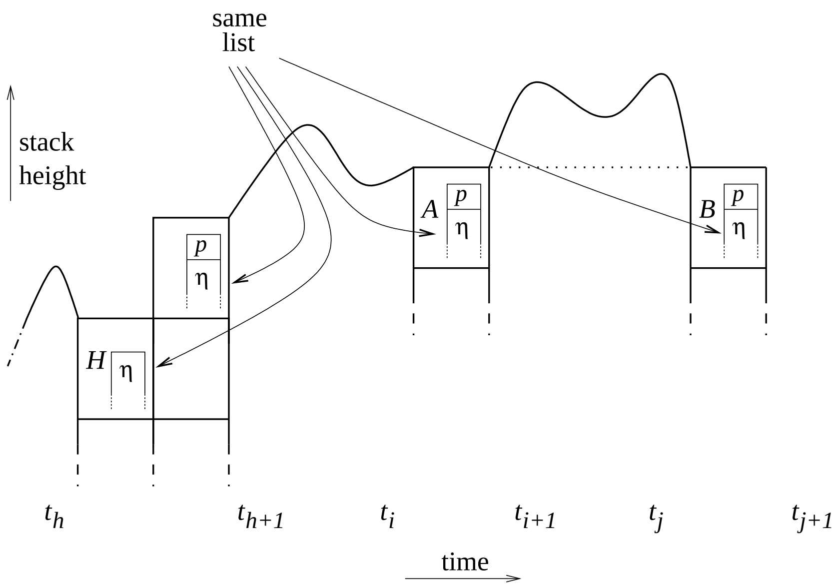

Figure 2. Meaning of an item $[H, h \mid A, B, i, j \mid p]$. Time t_k is when symbol a_k is read from the input.

Formally, an item $[H, h \mid A, B, i, j \mid p]$ represents the existence of a sequence of steps of the automaton of the form

$$
\begin{array}{llrcl}
(0) & (I[\,], & a_0 & \cdots & a_n) \vdash^* \\
(1) & (w\ H[\eta], & a_{h+1} & \cdots & a_n) \vdash \\
(2) & (w\ H'[\,]\ G[\eta\, p], & a_{h+1} & \cdots & a_n) \vdash^* \\
(3) & (w\ H'[\,]\ w'\ A[\eta\, p], & a_{i+1} & \cdots & a_n) \vdash^* \\
(4) & (w\ H'[\,]\ w'\ B[\eta\, p], & a_{j+1} & \cdots & a_n)
\end{array}
$$

such that nowhere between configurations (3) and (4) does the stack shrink to access stack symbols in w', and all occurrences of η denote the same list, in the sense that it is passed on unaffected through this sequence of steps. It is allowed that indices are pushed on η and then popped again, but it is not allowed that elements from η are popped and then other elements are pushed to accidentally result in the same list. Also, p is not popped between (3) and (4).

In the case that A and B are connected to the empty list of indices, we let p, H, h be the dummy index $\Diamond$, dummy stack symbol $\Box$, and dummy input position $-$, respectively.

The second type of item is of the form $[H, h \mid A, B, i, j \mid p, q \mid E, F, l, m]$, where $H, A, E \in \Gamma_L$ and $B, F \in \Gamma_R$. It is used when the stack shrinks again, as represented in Figure 3. Formally, such an item indicates the existence of a sequence of steps of the automaton of the

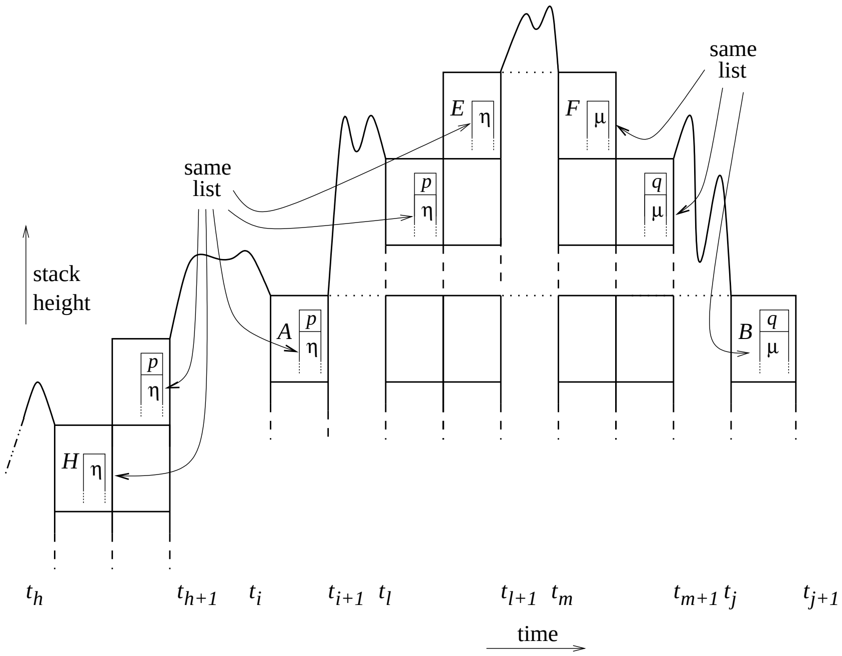

Figure 3. Meaning of an item $[H, h \mid A, B, i, j \mid p, q \mid E, F, l, m]$.

form

$$
\begin{array}{lllr}
(0) & (I[\,], & a_0 \quad \cdots \quad a_n) & \vdash^* \\
(1) & (w\ H[\eta], & a_{h+1} \quad \cdots \quad a_n) & \vdash \\
(2) & (w\ H'[\,]\ G[\eta\, p], & a_{h+1} \quad \cdots \quad a_n) & \vdash^* \\
(3) & (w\ H'[\,]\ w'\ A[\eta\, p], & a_{i+1} \quad \cdots \quad a_n) & \vdash^* \\
(4) & (w\ H'[\,]\ w'\ w''\ E'[\eta\, p], & a_{l+1} \quad \cdots \quad a_n) & \vdash \\
(5) & (w\ H'[\,]\ w'\ w''\ D[\,]\ E[\eta], & a_{l+1} \quad \cdots \quad a_n) & \vdash^* \\
(6) & (w\ H'[\,]\ w'\ w''\ D[\,]\ F[\mu], & a_{m+1} \cdots a_n) & \vdash \\
(7) & (w\ H'[\,]\ w'\ w''\ F'[\mu\, q], & a_{m+1} \cdots a_n) & \vdash^* \\
(8) & (w\ H'[\,]\ w'\ B[\mu\, q], & a_{j+1} \cdots a_n) &
\end{array}
$$

such that nowhere between configurations (3) and (8) does the stack shrink to access stack symbols in w', and nowhere between configurations (5) and (6) does the stack shrink to access $D[\,]$, all occurrences of η denote the same list (in the same sense as before) and p is not popped between (3) and (4), and all occurrence of μ are the same list and q is not popped between (7) and (8).

The constraints imposed on U-LIA in Section 3 ensure that the manipulations affecting the length of the list of indices while the (main) stack shrinks are the reverse of the changes that took place while the stack was growing. This means that η and μ must have equal length.

How items are derived from other items will be specified by means of inference rules. Each such rule consists of a list of antecedents, a consequent and a list of side conditions. The antecedents and the consequent are items. The side conditions refer to transitions of the automaton and to terminals from the input string. We omit further discussion of the meaning of inference rules and assume the reader is familiar with deductive parsing (Shieber et al., 1995).

The first rule has no antecedents and no side conditions. It corresponds to the initial configuration of the automaton.

$$\frac{}{[\Box, - \mid I, I, 0, 0 \mid \Diamond]}$$

The following two rules correspond to the creation of a fresh list of indices. In terms of parse trees, these steps typically correspond to the beginning of a new spine; in the first rule, this spine "branches off" from another spine at the left, in the second rule, it branches off at the right.

$$\frac{[H, h \mid D, A, i, j \mid p]}{[\Box, - \mid C, C, j, j \mid \Diamond]} \left\{ A[\circ\circ] \overset{\epsilon}{\mapsto} B[\circ\circ]\ C[\,] \right.$$

$$\frac{[H, h \mid D, A, i, j \mid p, q \mid E, F, l, m]}{[\Box, - \mid C, C, j, j \mid \Diamond]} \left\{ A[\circ\circ] \overset{\epsilon}{\mapsto} B[\circ\circ]\ C[\,] \right.$$

The transitions that increase the height of the stack, while lists of indices are manipulated and carried to higher regions of the stack, are tabulated by the following rules.

$$\frac{[H, h \mid D, A, i, j \mid p]}{[H, h \mid C, C, j, j \mid p]} \left\{ A[\circ\circ] \overset{\epsilon}{\mapsto} B[\,]\ C[\circ\circ] \right.$$

$$\frac{[H, h \mid D, A, i, j \mid p]}{[A, j \mid C, C, j, j \mid q]} \left\{ A[\circ\circ] \overset{\epsilon}{\mapsto} B[\,]\ C[\circ\circ\, q] \right.$$

$$\frac{\begin{array}{c}[F, m \mid E, H, k, h \mid q] \\ [H, h \mid D, A, i, j \mid p]\end{array}}{[F, m \mid C, C, j, j \mid q]} \left\{ A[\circ\circ\, p] \overset{\epsilon}{\mapsto} B[\,]\ C[\circ\circ] \right.$$

When a certain list of indices has been carried to the highest region in the stack that it will occupy, we stop using call items and switch to return items. Since at this point the list of indices is empty, both items contain many dummy symbols. In terms of parse trees, this rule is used at the lowest nodes in a spine:

$$\frac{[\Box, - \mid D, A, i, j \mid \Diamond]}{[\Box, - \mid D, B, i, k \mid \Diamond, \Diamond \mid \Box, \Box, -, -]} \left\{ \begin{array}{l} A[\,] \overset{z}{\mapsto} B[\,] \\ (z = \epsilon \wedge k = j) \vee \\ \quad (z = a_{j+1} \wedge k = j+1) \end{array} \right.$$

While the stack shrinks, the manipulation of the list of indices is matched to the corresponding manipulation that occurred while the stack was still growing. We distinguish between three kinds of manipulation of indices, leading to the following three rules.

$$\frac{\begin{array}{l} [H, h \mid D, A_1, i, j \mid p] \\ [H, h \mid C_1, C_2, j, k \mid p, q \mid E, F, l, m] \end{array}}{[H, h \mid D, A_2, i, k \mid p, q \mid E, F, l, m]} \left\{ \begin{array}{l} A_1[\circ\circ] \overset{\epsilon}{\mapsto} B[\,]\ C_1[\circ\circ] \\ B[\,]\ C_2[\circ\circ] \overset{\epsilon}{\mapsto} A_2[\circ\circ] \end{array} \right.$$

$$\frac{\begin{array}{l} [H, h \mid D, A_1, i, j \mid p'] \\ [H, h \mid E, F, l, m \mid p', q' \mid E', F', l', m'] \\ [A_1, j \mid C_1, C_2, j, k \mid p, q \mid E, F, l, m] \end{array}}{[H, h \mid D, A_2, i, k \mid p', q' \mid E', F', l', m']} \left\{ \begin{array}{l} A_1[\circ\circ] \overset{\epsilon}{\mapsto} B[\,]\ C_1[\circ\circ\, p] \\ B[\,]\ C_2[\circ\circ\, q] \overset{\epsilon}{\mapsto} A_2[\circ\circ] \end{array} \right.$$

$$\frac{\begin{array}{l} [H', h' \mid D', H, h'', h \mid p'] \\ [H, h \mid D, A_1, i, j \mid p] \\ [H', h' \mid C_1, C_2, j, k \mid p', q' \mid E, F, l, m] \end{array}}{[H, h \mid D, A_2, i, k \mid p, q \mid C_1, C_2, j, k]} \left\{ \begin{array}{l} A_1[\circ\circ\, p] \overset{\epsilon}{\mapsto} B[\,]\ C_1[\circ\circ] \\ B[\,]\ C_2[\circ\circ] \overset{\epsilon}{\mapsto} A_2[\circ\circ\, q] \end{array} \right.$$

In terms of the parse tree, when a spine has been completely treated, then we resume the treatment of the older spine from which it had branched off. The newer spine may have branched off at the left or at the right of the older spine, and these respective cases are treated by the following two rules.

$$\frac{\begin{array}{l} [H, h \mid D, A_1, i, j \mid p] \\ [\Box, - \mid C_1, C_2, j, k \mid \Diamond, \Diamond \mid \Box, \Box, -, -] \end{array}}{[H, h \mid D, A_2, i, k \mid p]} \left\{ \begin{array}{l} A_1[\circ\circ] \overset{\epsilon}{\mapsto} B[\circ\circ]\ C_1[\,] \\ B[\circ\circ]\ C_2[\,] \overset{\epsilon}{\mapsto} A_2[\circ\circ] \end{array} \right.$$

$$\frac{\begin{array}{l} [H, h \mid D, A_1, i, j \mid p, q \mid E, F, l, m] \\ [\Box, - \mid C_1, C_2, j, k \mid \Diamond, \Diamond \mid \Box, \Box, -, -] \end{array}}{[H, h \mid D, A_2, i, k \mid p, q \mid E, F, l, m]} \left\{ \begin{array}{l} A_1[\circ\circ] \overset{\epsilon}{\mapsto} B[\circ\circ]\ C_1[\,] \\ B[\circ\circ]\ C_2[\,] \overset{\epsilon}{\mapsto} A_2[\circ\circ] \end{array} \right.$$

After applying the inference rules until no more new items can be added to table U, the existence of an item

$[\Box, - \mid I, J, 0, n \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$ in U indicates that the input is recognized, i.e. that the string $a_1 \cdots a_n$ is in the language.

Table VIII shows the tabular simulation of the U-LIA from Table VI for the input string *abcd*. Each item is assigned a number. The third column refers to such a number where the item is used in the antecedent of an inference rule deriving another item. We have omitted items that can be derived but do not contribute to recognition of the input. By comparing Tables VII and VIII we can see that every configuration in the former corresponds to an item in the latter.

5.2. Tabulation for L-LIA

Tabulation for L-LIA is very similar to that for U-LIA above. The difference in behaviour of such automata is that while the stack shrinks with symbols from Γ_R on top, the list of indices is empty. This means that return items can be somewhat simplified by omitting the second index, so that we obtain items of the form $[H, h \mid A, B, i, j \mid p \mid E, F, l, m]$. The same number of inference rules is needed as before, but they can be slightly simplified with respect to U-LIA.

5.3. Tabulation for R-LIA

Although R-LIA is the dual of L-LIA, tabulation for R-LIA is much simpler than that for L-LIA or U-LIA. This paradox is resolved by the observation that all automata are assumed to read from left to right, and the tabular simulation of the automata adheres to the same order, which creates the asymmetry between right-oriented automata simulated left-to-right and left-oriented automata simulated left-to-right as well.

Tabulation of R-LIA was defined before in (Nederhof, 1998), generalizing the tabular, LR-like algorithm from (Alonso Pardo et al., 1997). The first observation leading to a simplification with respect to U-LIA is that the lists of indices are empty while the stack grows, which means that the call items are of the form $[\Box, - \mid A, B, i, j \mid \Diamond]$, which we simplify to $[A, B, i, j \mid \Diamond]$. Similarly, we may omit the components H, h and p from return items, so that we obtain $[A, B, i, j \mid q \mid E, F, l, m]$. By extending the meaning of return items to include the case $A, B \in \Gamma_L$, we may write call items as $[A, B, i, j \mid \Diamond \mid \Box, \Box, -, -]$ so that we obtain a uniform structure for both kinds of item. The consequence is that now only one rule is needed for a transition $A[\circ\circ] \overset{\epsilon}{\mapsto} B[\circ\circ]\ C[\,]$, whereas we needed two for U-LIA, and only one rule is needed for a pair of transitions $A_1[\circ\circ] \overset{\epsilon}{\mapsto} B[\circ\circ]\ C_1[\,]$ and $B[\circ\circ]\ C_2[\,] \overset{\epsilon}{\mapsto} A_2[\circ\circ]$, where we needed two inference rules for U-LIA. The other inference rules from

Table VIII. Items generated during the recognition of *abcd*

Nr.	Item in U	Derived from
0	$[\Box, - \mid I, I, 0, 0 \mid \Diamond]$	
1	$[\Box, - \mid \overline{S}, \overline{S}, 0, 0 \mid \Diamond]$	(0a) and 0
2	$[\Box, - \mid \nabla_{1,0}, \nabla_{1,0}, 0, 0 \mid \Diamond]$	(1a) and 1
3	$[\Box, - \mid \overline{A}, \overline{A}, 0, 0 \mid \Diamond]$	(1b) and 2
4	$[\Box, - \mid \nabla_{5,0}, \nabla_{5,0}, 0, 0 \mid \Diamond]$	(5a) and 3
5	$[\Box, - \mid \nabla_{5,1}, \nabla_{5,1}, 0, 1 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(5b) and 4
6	$[\Box, - \mid \overline{A}, \overline{\overline{A}}, 0, 1 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(5a), (5c), 3 and 5
7	$[\Box, - \mid \nabla_{1,0}, \nabla_{1,1}, 0, 1 \mid \Diamond]$	(1b), (1c), 2 and 6
8	$[\nabla_{1,1}, 1 \mid \overline{S}, \overline{S}, 1, 1 \mid p]$	(1d) and 7
9	$[\nabla_{1,1}, 1 \mid \nabla_{2,0}, \nabla_{2,0}, 1, 1 \mid p]$	(2a) and 8
10	$[\nabla_{1,1}, 1 \mid \overline{Q}, \overline{Q}, 1, 1 \mid p]$	(2b) and 9
11	$[\nabla_{1,1}, 1 \mid \nabla_{3,0}, \nabla_{3,0}, 1, 1 \mid p]$	(3a) and 10
12	$[\Box, - \mid \overline{B}, \overline{B}, 1, 1 \mid \Diamond]$	(3b) and 11
13	$[\Box, - \mid \nabla_{6,0}, \nabla_{6,0}, 1, 1 \mid \Diamond]$	(6a) and 12
14	$[\Box, - \mid \nabla_{6,0}, \nabla_{6,1}, 1, 2 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(6b) and 13
15	$[\Box, - \mid \overline{B}, \overline{\overline{B}}, 1, 2 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(6a), (6c), 12 and 14
16	$[\nabla_{1,1}, 1 \mid \nabla_{3,0}, \nabla_{3,1}, 1, 2 \mid p]$	(3b), (3c), 11 and 15
17	$[\Box, - \mid \overline{Q}, \overline{Q}, 2, 2 \mid \Diamond]$	(3d) and 16
18	$[\Box, - \mid \nabla_{4,0}, \nabla_{4,0}, 2, 2 \mid \Diamond]$	(4a) and 17
19	$[\Box, - \mid \nabla_{4,0}, \nabla_{4,1}, 2, 2 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(4b) and 18
20	$[\Box, - \mid \overline{Q}, \overline{\overline{Q}}, 2, 2 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(4a), (4c), 17 and 19
21	$[\nabla_{1,1}, 1 \mid \nabla_{3,0}, \nabla_{3,2}, 1, 2 \mid p, p \mid \overline{Q}, \overline{\overline{Q}}, 2, 2]$	(3d), (3e), 7, 16 and 20
22	$[\Box, - \mid \overline{C}, \overline{C}, 2, 2 \mid \Diamond]$	(3f) and 21
23	$[\Box, - \mid \nabla_{7,0}, \nabla_{7,0}, 2, 2 \mid \Diamond]$	(7a) and 22
24	$[\Box, - \mid \nabla_{7,0}, \nabla_{7,1}, 2, 3 \mid \Diamond]$	(7b) and 23
25	$[\Box, - \mid \overline{C}, \overline{\overline{C}}, 2, 3 \mid \Diamond]$	(7a), (7c), 22 and 24
26	$[\nabla_{1,1}, 1 \mid \nabla_{3,0}, \nabla_{3,3}, 1, 3 \mid p, p \mid \overline{Q}, \overline{\overline{Q}}, 2, 2]$	(3f), (3g), 21 and 25
27	$[\nabla_{1,1}, 1 \mid \overline{Q}, \overline{\overline{Q}}, 1, 3 \mid p, p \mid \overline{Q}, \overline{\overline{Q}}, 2, 2]$	(3a), (3h), 10 and 26
28	$[\nabla_{1,1}, 1 \mid \nabla_{2,0}, \nabla_{2,1}, 1, 3 \mid p, p \mid \overline{Q}, \overline{\overline{Q}}, 2, 2]$	(2b), (2c), 9 and 27
29	$[\nabla_{1,1}, 1 \mid \overline{s}, \overline{\overline{S}}, 1, 3 \mid p, p \mid \overline{Q}, \overline{\overline{Q}}, 2, 2]$	(2a), (2d), 8 and 28
30	$[\Box, - \mid \nabla_{1,0}, \nabla_{1,2}, 0, 3 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(1d), (1e), 7, 20 and 29
31	$[\Box, - \mid \overline{D}, \overline{D}, 3, 3 \mid \Diamond]$	(1f) and 30
32	$[\Box, - \mid \nabla_{8,0}, \nabla_{8,0}, 3, 3 \mid \Diamond]$	(8a) and 31
33	$[\Box, - \mid \nabla_{8,0}, \nabla_{8,1}, 3, 4 \mid \Diamond]$	(8b) and 32
34	$[\Box, - \mid \overline{D}, \overline{\overline{D}}, 3, 4 \mid \Diamond]$	(8a), (8c), 31 and 33
35	$[\Box, - \mid \nabla_{1,0}, \nabla_{1,3}, 0, 4 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(1f), (1g), 30 and 34
36	$[\Box, - \mid \overline{S}, \overline{\overline{S}}, 0, 4 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(1a), (1h), 1 and 35
37	$[\Box, - \mid I, J, 0, 0 \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$	(0a), (0b), 0 and 36

U-LIA can also be much simplified when they are adapted to R-LIA, especially where they involve call items.

6. Correctness

We sketch the proof of a number of properties of the tabulation of U-LIA. First, we can show that a string is recognized by the tabular algorithm if and only if it is recognized by the automaton. In one direction of the proof, we can show that if items in the antecedents of inference rules satisfy the characterization of items (see Figures 2 and 3 and the accompanying running text), then also the items in the consequents satisfy the characterization. It then follows that the presence of an item $[\Box, - \mid I, J, 0, n \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$ in U implies the recognition of the input by the automaton by means of a sequence $(I[\,], a_1 \cdots a_n) \vdash^* (J[\,], \epsilon)$.

For the other direction of the proof, we can show that any sequence of steps of the automaton that can be represented by an item can be broken up into smaller sequences of steps that can also be represented by items, in such a way that conceptually some inference rule is applied backwards, creating a number of antecedents from the consequent. Formally, this is a proof by induction on the length of sequences of steps, which is tedious but straightforward, and shows that recognition of input by the automaton, via a sequence of steps of the form $(I[\,], a_1 \cdots a_n) \vdash^* (J[\,], \epsilon)$, can be mimicked by application of inference rules, which eventually results in the item $[\Box, - \mid I, J, 0, n \mid \Diamond, \Diamond \mid \Box, \Box, -, -]$ being added to U.

The soundness and completeness of items as established in the proofs outlined above also lead to a proof of a second property of tabulation: the j-th input symbol is read in a sequence of steps $(I[\,], a_1 \cdots a_n) \vdash^* (w, a_j a_{j+1} \cdots a_n) \vdash (w', a_{j+1} \cdots a_n)$ of the automaton if and only if some item of the form $[H, h \mid A, B, i, j \mid p]$ or of the form $[H, h \mid A, B, i, j \mid p, q \mid E, F, l, m]$ is added to the table U by the tabular algorithm. This implies that if the input is not in the language, the automaton and the tabulated simulation of the automaton consult an identical prefix of the input, or informally, they both get stuck at the same input position. A consequence is that if the automaton satisfies the *correct-prefix property*, then this property is preserved in the tabulation.[1]

The above can also be proved to hold for L-LIA and R-LIA. In general however, the preservation of the correct-prefix property for R-LIAs by tabulation is irrelevant, since usually R-LIAs constructed from LIGs, e.g. following the compilation schemata from Section 4, do not satisfy the correct-prefix property. An informal explanation based on the example from Figure 1 is that if the string contains unequal numbers of a's and b's, then this is detected only indirectly, in the second half of the string, by matching a's and b's to d's and c's, respectively, and by comparing the numbers of c's and d's; recall that an R-LIA manipulates indices at the right side of a spine. Therefore, a longer prefix of the input must be consulted by the R-LIA than is strictly necessary to detect that the input cannot be in the language.

Generally, U-LIAs and L-LIAs do satisfy the correct-prefix property if they are constructed from a LIG by means of a top-down or Earley context-free strategy, and a top-down or Earley indices strategy, except if the grammar warrants a derivation $S \Rightarrow^* v\,A[\eta]\,w$, for some $A \in \mathcal{N}, \eta \in \mathcal{I}^*, v, w \in (\Sigma \cup \mathcal{N} \cup \mathcal{I} \cup \{[,]\})^*$, but no derivation $A[\eta] \Rightarrow^* x$ for any $x \in \Sigma^*$. Any grammar can however be transformed into one that generates the same language, but for which this problem does not occur (Nederhof, 1999b, Appendix A).

We can further show that the tabulation for U-LIA, L-LIA and R-LIA can be implemented to have a time complexity of $\mathcal{O}(n^6)$. This requires breaking up some inference rules, in the same way as this time complexity was achieved in (Nederhof, 1999a), for a parsing algorithm for TALs with the correct-prefix property.

7. Conclusion

We have proposed a modular design of tabular parsing algorithms for a class of languages represented by, amongst others, tree-adjoining grammars and linear indexed grammars. This modular design relies on a separation of the parsing strategy from the mechanism of tabulation. The parsing strategy is expressed in terms of the construction of U-LIAs, L-LIAs or R-LIAs from linear indexed grammars. We have shown that these three types of automaton together allow a wide range of different parsing strategies.

Although each parsing strategy can only be realized using one of the three types of automaton, the mechanism of tabulation for such a type of automaton is independent of the chosen parsing strategy. This modularity allows simple proofs of correctness and straightforward implementation of tabular parsing algorithms for tree-adjoining languages.

Acknowledgements

We would like to thank Tilman Becker, David Cabrero, Manuel Vilares, and David Weir for fruitful discussions. This work has been partially supported by FEDER of European Union under Grant 1FD97-0047-C04-02, by Xunta de Galicia under Grant PGIDT99XI10502B, and by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the VERBMOBIL Project under Grant 01 IV 701 V0. The second author was employed at AT&T Shannon Laboratory during a part of the period this paper was written.

Notes

[1] We say a recognizer for language L satisfies the *correct-prefix property* if it does not read past the position of the first syntax error in the input. This position can be defined as the rightmost symbol of the shortest prefix of the input that cannot be extended to be a correct sentence in the language L. In formal notation, this prefix for a given erroneous input $v \notin L$ is defined as the string wa, where $v = wax$, for some x, such that $wy \in L$, for some y, but $waz \notin L$, for any z. The occurrence of a in v indicates the error position. The term "correct-prefix property" is used in (Sippu and Soisalon-Soininen, 1988), but in other publications we also find "prefix property", "valid prefix property" or "viable prefix property".

References

Alonso Pardo, M., E. de la Clergerie, and M. Vilares Ferro: 1997, 'Automata-based parsing in dynamic programming for Linear Indexed Grammars'. In: A. Narin'yani (ed.): *Computational Linguistics and its Applications, proceedings.* Moscow, Russia, pp. 22–27.

Becker, T.: 1994, 'A new automaton model for TAGs: 2-SA'. *Computational Intelligence* **10**(4), 422–430.

Billot, S. and B. Lang: 1989, 'The Structure of Shared Forests in Ambiguous Parsing'. In: *27th Annual Meeting of the ACL.* Vancouver, British Columbia, Canada, pp. 143–151.

de la Clergerie, E., M. Alonso Pardo, and D. Cabrero Souto: 1998, 'A Tabular Interpretation of Bottom-up Automata for TAG'. In: *Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks.* pp. 42–45.

Gazdar, G.: 1987, 'Applicability of Indexed Grammars to Natural Languages'. In: U. Reyle and C. Rohrer (eds.): *Natural Language Parsing and Linguistic Theories.* D. Reidel Publishing Company, pp. 69–94.

Joshi, A.: 1987, 'An introduction to tree adjoining grammars'. In: A. Manaster-Ramer (ed.): *Mathematics of Language.* Amsterdam: John Benjamins Publishing Company, pp. 87–114.

Joshi, A., K. Vijay-Shanker, and D. Weir: 1991, 'The Convergence of Mildly Context-Sensitive Grammar Formalisms'. In: P. Sells, S. Shieber, and T. Wasow (eds.): *Foundational Issues in Natural Language Processing.* MIT Press, Chapt. 2, pp. 31–81.

Lang, B.: 1974, 'Deterministic techniques for efficient non-deterministic parsers'. In: *Automata, Languages and Programming, 2nd Colloquium*, Vol. 14 of *Lecture Notes in Computer Science.* Saarbrücken, pp. 255–269.

Lang, B.: 1988a, 'Complete Evaluation of Horn Clauses: An Automata Theoretic Approach'. Rapport de Recherche 913, Institut National de Recherche en Informatique et en Automatique, Rocquencourt, France.

Lang, B.: 1988b, 'The Systematic Construction of Earley Parsers: Application to the Production of $\mathcal{O}(n^6)$ Earley Parsers for Tree Adjoining Grammars'. Unpublished paper.

Lang, B.: 1994, 'Recognition can be harder than parsing'. *Computational Intelligence* **10**(4), 486–494.

Nederhof, M.-J.: 1998, 'Linear indexed automata and tabulation of TAG parsing'. In: *Actes des premières journées sur la Tabulation en Analyse Syntaxique et Déduction (Tabulation in Parsing and Deduction).* Paris, France, pp. 1–9.

Nederhof, M.-J.: 1999a, 'The Computational Complexity of the Correct-Prefix Property for TAGs'. *Computational Linguistics* **25**(3), 345–360.

Nederhof, M.-J.: 1999b, 'Models of tabulation for TAG parsing'. In: *Sixth Meeting on Mathematics of Language.* Orlando, Florida USA, pp. 143–158.

Parchmann, R., J. Duske, and J. Specht: 1980, 'On Deterministic Indexed Languages'. *Information and Control* **45**, 48–67.

Schabes, Y. and K. Vijay-Shanker: 1990, 'Deterministic left to right parsing of tree adjoining languages'. In: *28th Annual Meeting of the ACL.* Pittsburgh, Pennsylvania, USA, pp. 276–283.

Shieber, S., Y. Schabes, and F. Pereira: 1995, 'Principles and implementation of deductive parsing'. *Journal of Logic Programming* **24**, 3–36.

Sippu, S. and E. Soisalon-Soininen: 1988, *Parsing Theory, Vol. I: Languages and Parsing.* Springer-Verlag.

Sippu, S. and E. Soisalon-Soininen: 1990, *Parsing Theory, Vol. II: LR(k) and LL(k) Parsing.* Springer-Verlag.

Tomita, M.: 1986, *Efficient Parsing for Natural Language.* Kluwer Academic Publishers.

Vijay-Shanker, K. and D. Weir: 1993, 'Parsing Some Constrained Grammar Formalisms'. *Computational Linguistics* **19**(4), 591–636.

Vijay-Shanker, K. and D. Weir: 1993b, 'The Use of Shared Forests in Tree Adjoining Grammar Parsing'. In: *Sixth Conference of the European Chapter of the ACL.* Utrecht, The Netherlands, pp. 384–393.

Vijay-Shanker, K. and D. Weir: 1994, 'The Equivalence of Four Extensions of Context-Free Grammars'. *Mathematical Systems Theory* **27**, 511–546.

Villemonte de la Clergerie, E. and M. Alonso Pardo: 1998, 'A tabular interpretation of a class of 2-Stack Automata'. In: *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, Vol. 2. Montreal, Quebec, Canada, pp. 1333–1339.

Villemonte de la Clergerie, E. and F. Barthélemy: 1998, 'Information flow in Tabular Interpretations for generalized Push-Down Automata'. *Theoretical Computer Science* **199**, 167–198.

Weir, D.: 1994, 'Linear iterated pushdowns'. *Computational Intelligence* **10**(4), 431–439.