Models of tabulation for TAG parsing

Mark-Jan Nederhof
DFKI
Stuhlsatzenhausweg 3, D-66123, Saarbrucken, Germany
E-mail: nederhof@dfki.de

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 nondeterministic automata from a grammar. The mechanism of tabulation leads to the interpretation of these nondeterministic automata as tabular recognizers, independent of the strategy.

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

1 Introduction

Design of correct and efficient parsing algorithms for tree-adjoining grammars (TAGs) is a difficult task. A possible way to simplify 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 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 generate 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) and Vijay-Shanker and Weir (1993b), who propose 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 certain types of recognizer for TAGs, which are notational variants of existing types such as embedded pushdown automata (Schabes and Vijay-Shanker, 1990), 2-SA (Becker, 1994), and the $\mathcal{P}^2_{\text{lin}}$-automata from Weir (1994). The use of our notation simplifies the task of adapting tabulation techniques for context-free parsing to tree-adjoining languages.

One type of recognizer in this notation, which we will refer to here as *right-oriented linear indexed automata*, was discussed by Nederhof (1998b), where we also introduced its tabulation. Nederhof (1998a) used this type as a means to define a new nondeterministic LR parsing algorithm for TAGs. Using the tabulation technique introduced earlier we thereby found a modular way to define tabular LR parsers for TAGs.

A dual type of recognizer, the *left-oriented linear indexed automata*, was briefly mentioned by Nederhof (1998b), but its tabulation has until now not been defined. In this paper, we will recall tabulation for the right-oriented automata, and we will introduce tabulation for the left-oriented type.

Independently from our work, Villemonte de la Clergerie and Alonso Pardo (1998) have proposed an alternative type of automaton that captures both right and left orientation. However, the generality of these automata also causes tabulation to be very complicated. Our restriction to either left or right orientation allows significant simplification, both in terms of presentational elegance and in terms of computational complexity.

The article may be outlined as follows: In Section 2 we recall the definition of linear indexed grammars. In Section 3 we present two types of recog-

nizer, and Section 4 shows the equivalence of each of these two types to linear indexed grammars, and thereby to tree-adjoining grammars. Tabulation is discussed in Sections 5 and 6, for the respective two types of recognizer. There we will show that all computations of such recognizers can be simulated in polynomial time. Section 7 presents final conclusions.

2 Linear indexed grammars

Linear indexed grammars (LIGs) generate the same class of languages as the tree-adjoining grammars. Other kinds of grammar generating this class of languages are the head grammars and the combinatory categorial grammars (Vijay-Shanker and Weir, 1994). This class is a strict subclass of the mildly context-sensitive languages (Joshi et al., 1991).

A LIG G is 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*, $\mathcal{I}$ is a finite set of *indices* and $\mathcal{P}$ is a finite set of *productions*, each having one of the following forms:

- $A[\circ\circ\eta] \rightarrow \alpha\ B[\circ\circ\eta']\ \beta$, where $A, B \in \mathcal{N}$, $\eta, \eta' \in \mathcal{I}^*$, and α and β are lists of objects of the form $C[\eta'']$, where $C \in \mathcal{N}$ and $\eta'' \in \mathcal{I}^*$; or
- $A[\eta] \rightarrow z$, where $A \in \mathcal{N}$, $\eta \in \mathcal{I}^*$, and $z \in \Sigma \cup \{\varepsilon\}$.

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

We say a LIG is in *binary normal form*, if each production has one of the following forms:

- $A[\circ\circ\eta] \rightarrow B[\circ\circ\eta']\ C[\,]$;
- $A[\circ\circ\eta] \rightarrow B[\,]\ C[\circ\circ\eta']$; or
- $A[\,] \rightarrow z$.

where in productions of the first and second type, $\eta, \eta' \in \mathcal{I} \cup \{\varepsilon\}$ and either η or η' (or both) must be the empty string. As before, $z \in \Sigma \cup \{\varepsilon\}$.

The restriction to binary normal form helps us to avoid some of the technical problems that had to be overcome by Vijay-Shanker and Weir (1993a). It can be easily shown that any LIG can be transformed into a weakly equivalent LIG in binary normal form. Apart from the manipulation of indices, this proof is more simple than the corresponding proof for Chomsky normal form and context-free grammars (Lewis and Papadimitriou, 1981), since right-hand sides are here allowed to be empty. We will not discuss this matter any further, and assume implicitly from now on that the class of LIGs and the class of LIGs in binary normal form generate the same class of languages.

For a precise description of how LIGs generate languages, we refer to Vijay-Shanker and Weir (1994). Let it suffice here to mention that $[\circ\circ\eta]$ stands for a list. In the case that η is of the form $p \in \mathcal{I}$ then

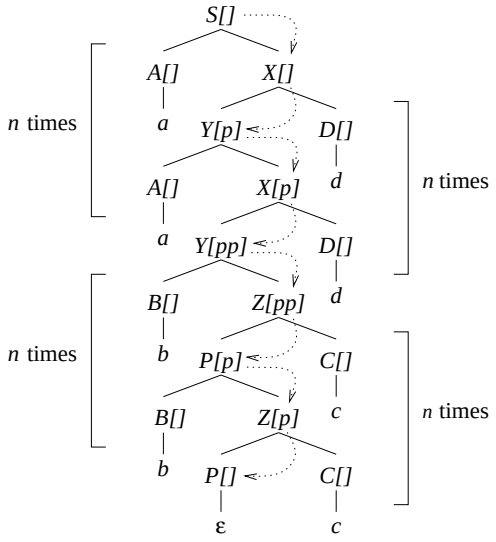

Figure 1: Parse tree for a LIG.

p is the head of that list and $\circ\circ$ is the unspecified tail. In the case that $\eta = \varepsilon$ however, $\circ\circ$ is the list itself. Both occurrences of $\circ\circ$ in a production refer to the *same* list ("consistent substitution"). The empty list is written as $[\,]$. Thus, a production $A[\circ\circ p] \rightarrow B[\circ\circ]\ C[\,]$ can be written as the following production of a definite clause grammar (Pereira and Warren, 1980):

```
'A'([p | List]) --> 'B'(List), 'C'([]).
```

As another example, a production $A[\,] \rightarrow a$ can be written as:

```
'A'([]) --> "a".
```

The start symbol is to be attached to the empty list of indices: $S[\,]$. Thus, a Prolog query for input $a_1 \cdots a_n$, would be of the form:

```
?- 'S'([], [a_1, ...,  a_n], []).
```

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

(1)	$S[\circ\circ]$	$\rightarrow$	$A[\,]\ X[\circ\circ]$	(7)	$P[\,]$	$\rightarrow$	ε
(2)	$X[\circ\circ]$	$\rightarrow$	$Y[\circ\circ p]\ D[\,]$	(8)	$A[\,]$	$\rightarrow$	a
(3)	$Y[\circ\circ]$	$\rightarrow$	$A[\,]\ X[\circ\circ]$	(9)	$B[\,]$	$\rightarrow$	b
(4)	$Y[\circ\circ]$	$\rightarrow$	$B[\,]\ Z[\circ\circ]$	(10)	$C[\,]$	$\rightarrow$	c
(5)	$Z[\circ\circ p]$	$\rightarrow$	$P[\circ\circ]\ C[\,]$	(11)	$D[\,]$	$\rightarrow$	d
(6)	$P[\circ\circ]$	$\rightarrow$	$B[\,]\ Z[\circ\circ]$				

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

3 Linear indexed automata

We discuss two 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.

The difference of the two types of automaton with (bottom-up) embedded pushdown automata (Schabes and Vijay-Shanker, 1990) is restricted to notation. However, the chosen notation will simplify the development of tabulation techniques, to be discussed in the following sections.

A *right-oriented linear indexed automaton* (R-LIA) is a 6-tuple $(\Sigma, \mathcal{Q}, I, F, \mathcal{I}, \mathcal{T})$, where Σ and $\mathcal{I}$ are as before, $\mathcal{Q}$ is a finite set of *stack symbols*, $I \in \mathcal{Q}$ is the *initial stack symbol*, $F \in \mathcal{Q}$ is the *final stack symbol* and $\mathcal{T}$ is a finite set of *transitions*. Each transition should be of one of the following forms:

- $X[\circ\circ\eta] \overset{\varepsilon}{\mapsto} Y[\circ\circ\eta']$;
- $X[\circ\circ] \overset{z}{\mapsto} Y[\circ\circ]\ Z[]$;
- $Y[]\ Z[\circ\circ] \overset{\varepsilon}{\mapsto} X[\circ\circ]$; or
- $Y[\circ\circ]\ Z[] \overset{z}{\mapsto} X[\circ\circ]$,

where $X, Y, Z \in \mathcal{Q}$, $\eta, \eta' \in \mathcal{I} \cup \{\varepsilon\}$, $z \in \Sigma \cup \{\varepsilon\}$; and for transitions of the first form, either η or η' (or both) must be the empty string.

We now represent the second type of automaton. A *left-oriented linear indexed automaton* (L-LIA) is a 6-tuple $(\Sigma, \mathcal{Q}, I, F, \mathcal{I}, \mathcal{T})$ as before. The only difference lies in the allowable transitions, each of which should be of one of the following forms:

- $X[\circ\circ\eta] \overset{\varepsilon}{\mapsto} Y[\circ\circ\eta']$;
- $X[\circ\circ] \overset{z}{\mapsto} Y[\circ\circ]\ Z[]$;
- $X[\circ\circ] \overset{\varepsilon}{\mapsto} Y[]\ Z[\circ\circ]$; or
- $Y[\circ\circ]\ Z[] \overset{z}{\mapsto} X[\circ\circ]$,

with the same restrictions on z, η and η' as before.

We define a *configuration* to be an element of $(\mathcal{Q} \times \mathcal{I}^*)^* \times \Sigma^*$. Each element from $\mathcal{Q} \times \mathcal{I}^*$ contains a stack symbol and a list of indices. A list of such elements from $\mathcal{Q} \times \mathcal{I}^*$ 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.

The finite set of transitions can conceptually be seen as the infinite set that results by consistently substituting $\circ\circ$ in the right-hand and left-hand sides of transitions by arbitrary elements from $\mathcal{I}^*$. For example, a transition $X[\circ\circ\eta] \overset{\varepsilon}{\mapsto} Y[\circ\circ\eta']$ may be seen as an infinite set of transitions of the form $X[\eta''\eta] \overset{\varepsilon}{\mapsto} Y[\eta''\eta']$, where $\eta'' \in \mathcal{I}^*$.

Let the resulting infinite set of "instantiated transitions" be called $\mathcal{T}'$. We define the binary relation $\vdash$ between configurations as: $(\alpha\beta, zw) \vdash (\alpha\gamma, w)$ if and only if $\beta \overset{z}{\mapsto} \gamma$ in $\mathcal{T}'$, for any $\alpha \in (\mathcal{Q} \times \mathcal{I}^*)^*$ and $w \in \Sigma^*$. The transitive and reflexive closure of $\vdash$ is denoted by $\vdash^*$.

Some input v is *recognized* if $(I[], v) \vdash^* (F[], \varepsilon)$. The language *accepted* by R-LIA or L-LIA M is defined to be the set of all v that are recognized.

Note that the only difference between R-LIA and L-LIA lies in the third type of transition. In the case of R-LIA, a list of indices cannot be carried to higher regions of the stack, whereas for L-LIA, they cannot be carried to lower regions. L-LIA correspond to embedded pushdown automata (Schabes and Vijay-Shanker, 1990) and 2-SA (Becker, 1994), whereas R-LIA correspond to *bottom-up* embedded pushdown automata (Schabes and Vijay-Shanker, 1990) and *bottom-up* 2-SA (de la Clergerie et al., 1998). In Section 4 we will discuss the consequences the difference between the two types of automaton has on the typical behaviour during recognition.

The two types of linear indexed automaton represent subclasses of the logical pushdown automata (Lang, 1988a). The reason these subclasses are considered in isolation is that they allow specific forms of tabulation, as will be explained later.

4 R-LIA and L-LIA versus LIG

A language is accepted by a right-oriented linear indexed automaton if and only if it is generated by a linear indexed grammar.

For the first half of the proof, we show that, for any LIG $G = (\Sigma, \mathcal{N}, S, \mathcal{I}, \mathcal{P})$ in binary normal form, we can construct an equivalent R-LIA $(\Sigma, \mathcal{Q}, I, F, \mathcal{I}, \mathcal{T})$, where $\mathcal{Q} = \mathcal{N} \cup \{I, F\} \cup \{\nabla_p \mid p \in \mathcal{P}\}$, I and F being fresh symbols. The transitions in $\mathcal{T}$ are given by the following:

- For any $p = A[\circ\circ\eta] \rightarrow B[\circ\circ\eta']\ C[]$ in $\mathcal{P}$, $\mathcal{T}$ contains:
  - $B[\circ\circ]\ C[] \overset{\varepsilon}{\mapsto} \nabla_p[\circ\circ]$; and
  - $\nabla_p[\circ\circ\eta'] \overset{\varepsilon}{\mapsto} A[\circ\circ\eta]$.
- For any $p = A[\circ\circ\eta] \rightarrow B[]\ C[\circ\circ\eta']$ in $\mathcal{P}$, $\mathcal{T}$ contains:
  - $B[]\ C[\circ\circ] \overset{\varepsilon}{\mapsto} \nabla_p[\circ\circ]$; and
  - $\nabla_p[\circ\circ\eta'] \overset{\varepsilon}{\mapsto} A[\circ\circ\eta]$.
- For any $A[] \rightarrow z$ in $\mathcal{P}$, $\mathcal{T}$ contains:
  - $R[\circ\circ] \overset{z}{\mapsto} R[\circ\circ]\ A[]$, for all $R \in \mathcal{Q}$.
- $\mathcal{T}$ contains further:
  - $I[\circ\circ]\ S[] \overset{\varepsilon}{\mapsto} F[\circ\circ]$.

The resulting automaton behaves like a pure bottom-up recognizer for the language generated by the grammar. That exactly this language is recognized will not be proven here; we refer to standard textbooks on language theory and parsing for very similar types of recognizer and their proofs of correctness (e.g. Lewis and Papadimitriou (1981)).

For the second half of the proof, we show that for any R-LIA $(\Sigma, \mathcal{Q}, I, F, \mathcal{I}, \mathcal{T})$ we can construct an equivalent LIG $G = (\Sigma, \mathcal{N}, S, \mathcal{I}, \mathcal{P})$, where $\mathcal{N} = \mathcal{Q} \times \mathcal{Q}$ and $S = (I, F)$. The productions in $\mathcal{P}$ are given by the following:

- For any $P \in \mathcal{Q}$ and $X[\circ\circ\eta] \overset{\varepsilon}{\mapsto} Y[\circ\circ\eta']$ in $\mathcal{T}$, $\mathcal{P}$ contains:
  - $(P, Y)[\circ\circ\eta'] \rightarrow (P, X)[\circ\circ\eta]$.
- For any $P \in \mathcal{Q}$, and $X[\circ\circ] \overset{z}{\mapsto} Y[\circ\circ]\ Z[]$ and $Y[\circ\circ]\ Z'[] \overset{z'}{\mapsto} X'[\circ\circ]$ in $\mathcal{T}$, $\mathcal{P}$ contains:
  - $(P, X')[\circ\circ] \rightarrow (P, X)[\circ\circ]\ z\ (Z, Z')[]\ z'$.
- For any $P \in \mathcal{Q}$, and $X[\circ\circ] \overset{z}{\mapsto} Y[\circ\circ]\ Z[]$ and $Y[]\ Z'[\circ\circ] \overset{\varepsilon}{\mapsto} X'[\circ\circ]$ in $\mathcal{T}$, $\mathcal{P}$ contains:
  - $(P, X')[\circ\circ] \rightarrow (P, X)[]\ z\ (Z, Z')[\circ\circ]$.
- For any $P \in \mathcal{Q}$, $\mathcal{P}$ contains:
  - $(P, P)[] \rightarrow \varepsilon$.

For the resulting grammar, $(X, Y)[\eta] \Rightarrow^* w$ if and only if $(X[], w) \vdash^* (Y[\eta], \varepsilon)$, for any $X, Y \in \mathcal{Q}$, $\eta \in \mathcal{I}^*$ and $w \in \Sigma^*$. (We assume the usual definition of relation $\Rightarrow^*$ for LIGs.) Again, we do not give full details of the proof, but refer to standard literature (e.g. Lewis and Papadimitriou (1981)) for similar theorems and their proofs.

A R-LIA for the grammar of the running example can be constructed as explained above. We obtain the following set of transitions, identifying productions with their number.

$$
\begin{array}{rcl@{\qquad}rcl}
A[]\ X[\circ\circ] & \overset{\varepsilon}{\mapsto} & \nabla_1[\circ\circ] & \nabla_1[\circ\circ] & \overset{\varepsilon}{\mapsto} & S[\circ\circ] \\
Y[\circ\circ]\ D[] & \overset{\varepsilon}{\mapsto} & \nabla_2[\circ\circ] & \nabla_2[\circ\circ p] & \overset{\varepsilon}{\mapsto} & X[\circ\circ] \\
A[]\ X[\circ\circ] & \overset{\varepsilon}{\mapsto} & \nabla_3[\circ\circ] & \nabla_3[\circ\circ] & \overset{\varepsilon}{\mapsto} & Y[\circ\circ] \\
B[]\ Z[\circ\circ] & \overset{\varepsilon}{\mapsto} & \nabla_4[\circ\circ] & \nabla_4[\circ\circ] & \overset{\varepsilon}{\mapsto} & Y[\circ\circ] \\
P[\circ\circ]\ C[] & \overset{\varepsilon}{\mapsto} & \nabla_5[\circ\circ] & \nabla_5[\circ\circ] & \overset{\varepsilon}{\mapsto} & Z[\circ\circ p] \\
B[]\ Z[\circ\circ] & \overset{\varepsilon}{\mapsto} & \nabla_6[\circ\circ] & \nabla_6[\circ\circ] & \overset{\varepsilon}{\mapsto} & P[\circ\circ] \\
R[\circ\circ] & \overset{\varepsilon}{\mapsto} & R[\circ\circ]\ P[], \textit{ for all } R \in \mathcal{Q} & & & \\
R[\circ\circ] & \overset{a}{\mapsto} & R[\circ\circ]\ A[], \textit{ for all } R \in \mathcal{Q} & & & \\
R[\circ\circ] & \overset{b}{\mapsto} & R[\circ\circ]\ B[], \textit{ for all } R \in \mathcal{Q} & & & \\
R[\circ\circ] & \overset{c}{\mapsto} & R[\circ\circ]\ C[], \textit{ for all } R \in \mathcal{Q} & & & \\
R[\circ\circ] & \overset{d}{\mapsto} & R[\circ\circ]\ D[], \textit{ for all } R \in \mathcal{Q} & & & \\
I[\circ\circ]\ S[] & \overset{\varepsilon}{\mapsto} & F[\circ\circ] & & &
\end{array}
$$

For each production with two elements in the right-hand side, except for productions (2) and (5), the two transitions that result can be merged to yield a smaller automaton. Figure 2 presents this smaller automaton and indicates how the input $a^2b^2c^2d^2$ is recognized.

We can further prove that a language is accepted by a left-oriented linear indexed automaton if and only if it is generated by a linear indexed grammar. The proof is very similar to the above, since L-LIAs can be seen as R-LIAs "in reverse".

For example, if we want to construct a L-LIA from a LIG, we can first "reverse" the grammar by exchanging the first and second members in all right-hand sides that possess two members. We can then construct a R-LIA as indicated above, and subsequently reverse this automaton by exchanging left-hand and right-hand sides, and by exchanging occurrences of I and F.[1] The result is a L-LIA accepting the same language as generated by the original grammar. This L-LIA exhibits a top-down behaviour, unlike the R-LIA for the reversed grammar, which exhibits a bottom-up behaviour. The resulting (simplified) L-LIA for the running example and its application on input is demonstrated in Figure 3.

The above observation is the reason I have abstained from identifying the keywords "top-down" and "bottom-up" with L-LIA and R-LIA, respectively, since we see that it is the left-to-right interpretation of the automata that make them behave in a top-down or bottom-up fashion, rather than some inherent aspect of the structure of transitions. If, say, a R-LIA showing bottom-up behaviour with respect to left-to-right processing is interpreted in reverse, by reading input from right to left, it could reveal top-down behaviour instead.

Related to the top-down or bottom-up behaviour, the two types of automaton are also distinct with respect to the correct-prefix property, assuming from now on ordinary left-to-right processing for both types. We say a recognizer 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 which 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.

This can be illustrated with regard to Figure 1. The path of dotted arrows starting from $S[]$ and ending at $P[]$ indicates how lists are passed on from mother to daughter nodes. A maximal collection of nodes that conceptually pass on lists from one to the other will be called a *spine*. The spine between $S[]$

[1] This kind of reversal for pushdown automata has been discussed before by Nederhof (1996).

$$
\begin{array}{rcl}
A[]\ X[\circ\circ] & \overset{\varepsilon}{\mapsto} & S[\circ\circ] \\
Y[\circ\circ]\ D[] & \overset{\varepsilon}{\mapsto} & \nabla_2[\circ\circ] \\
\nabla_2[\circ\circ p] & \overset{\varepsilon}{\mapsto} & X[\circ\circ] \\
A[]\ X[\circ\circ] & \overset{\varepsilon}{\mapsto} & Y[\circ\circ] \\
B[]\ Z[\circ\circ] & \overset{\varepsilon}{\mapsto} & Y[\circ\circ] \\
P[\circ\circ]\ C[] & \overset{\varepsilon}{\mapsto} & \nabla_5[\circ\circ] \\
\nabla_5[\circ\circ] & \overset{\varepsilon}{\mapsto} & Z[\circ\circ p] \\
B[]\ Z[\circ\circ] & \overset{\varepsilon}{\mapsto} & P[\circ\circ] \\
R[\circ\circ] & \overset{\varepsilon}{\mapsto} & R[\circ\circ]\ P[],\ \textit{any } R \\
R[\circ\circ] & \overset{a}{\mapsto} & R[\circ\circ]\ A[],\ \textit{any } R \\
R[\circ\circ] & \overset{b}{\mapsto} & R[\circ\circ]\ B[],\ \textit{any } R \\
R[\circ\circ] & \overset{c}{\mapsto} & R[\circ\circ]\ C[],\ \textit{any } R \\
R[\circ\circ] & \overset{d}{\mapsto} & R[\circ\circ]\ D[],\ \textit{any } R \\
I[\circ\circ]\ S[] & \overset{\varepsilon}{\mapsto} & F[\circ\circ]
\end{array}
$$

Stack	Input	Next step
$I[]$	$aabbccdd$	$I[\circ\circ] \overset{a}{\mapsto} I[\circ\circ]\ A[]$
$I[]\ A[]$	$abbccdd$	$A[\circ\circ] \overset{a}{\mapsto} A[\circ\circ]\ A[]$
$I[]\ A[]\ A[]$	$bbccdd$	$A[\circ\circ] \overset{b}{\mapsto} A[\circ\circ]\ B[]$
$I[]\ A[]\ A[]\ B[]$	$bccdd$	$B[\circ\circ] \overset{b}{\mapsto} B[\circ\circ]\ B[]$
$I[]\ A[]\ A[]\ B[]\ B[]$	$ccdd$	$B[\circ\circ] \overset{\varepsilon}{\mapsto} B[\circ\circ]\ P[]$
$I[]\ A[]\ A[]\ B[]\ B[]\ P[]$	$ccdd$	$P[\circ\circ] \overset{c}{\mapsto} P[\circ\circ]\ C[]$
$I[]\ A[]\ A[]\ B[]\ B[]\ P[]\ C[]$	cdd	$P[\circ\circ]\ C[] \overset{\varepsilon}{\mapsto} \nabla_5[\circ\circ]$
$I[]\ A[]\ A[]\ B[]\ B[]\ \nabla_5[]$	cdd	$\nabla_5[\circ\circ] \overset{\varepsilon}{\mapsto} Z[\circ\circ p]$
$I[]\ A[]\ A[]\ B[]\ B[]\ Z[p]$	cdd	$B[]\ Z[\circ\circ] \overset{\varepsilon}{\mapsto} P[\circ\circ]$
$I[]\ A[]\ A[]\ B[]\ P[p]$	cdd	$P[\circ\circ] \overset{c}{\mapsto} P[\circ\circ]\ C[]$
$I[]\ A[]\ A[]\ B[]\ P[p]\ C[]$	dd	$P[\circ\circ]\ C[] \overset{\varepsilon}{\mapsto} \nabla_5[\circ\circ]$
$I[]\ A[]\ A[]\ B[]\ \nabla_5[p]$	dd	$\nabla_5[\circ\circ] \overset{\varepsilon}{\mapsto} Z[\circ\circ p]$
$I[]\ A[]\ A[]\ B[]\ Z[pp]$	dd	$B[]\ Z[\circ\circ] \overset{\varepsilon}{\mapsto} Y[\circ\circ]$
$I[]\ A[]\ A[]\ Y[pp]$	dd	$Y[\circ\circ] \overset{d}{\mapsto} Y[\circ\circ]\ D[]$
$I[]\ A[]\ A[]\ Y[pp]\ D[]$	d	$Y[\circ\circ]\ D[] \overset{\varepsilon}{\mapsto} \nabla_2[\circ\circ]$
$I[]\ A[]\ A[]\ \nabla_2[pp]$	d	$\nabla_2[\circ\circ p] \overset{\varepsilon}{\mapsto} X[\circ\circ]$
$I[]\ A[]\ A[]\ X[p]$	d	$A[]\ X[\circ\circ] \overset{\varepsilon}{\mapsto} Y[\circ\circ]$
$I[]\ A[]\ Y[p]$	d	$Y[\circ\circ] \overset{d}{\mapsto} Y[\circ\circ]\ D[]$
$I[]\ A[]\ Y[p]\ D[]$	ε	$Y[\circ\circ]\ D[] \overset{\varepsilon}{\mapsto} \nabla_2[\circ\circ]$
$I[]\ A[]\ \nabla_2[p]$	ε	$\nabla_2[\circ\circ p] \overset{\varepsilon}{\mapsto} X[\circ\circ]$
$I[]\ A[]\ X[]$	ε	$A[]\ X[\circ\circ] \overset{\varepsilon}{\mapsto} S[\circ\circ]$
$I[]\ S[]$	ε	$I[\circ\circ]\ S[] \overset{\varepsilon}{\mapsto} F[\circ\circ]$
$F[]$	ε	recognition

Figure 2: Transitions of the simplified R-LIA, and a sequence of configurations for input *aabbccdd*.

and $P[]$ in Figure 1 is the only nontrivial spine. In general, several spines can be found in a parse tree for LIGs.

The task of a recognizer in this example is to verify that the numbers of a's, b's, c's and d's match. In the R-LIA we constructed, this was done by two cooperating mechanisms. First, the automaton stores A's and B's on the stack for all a's and b's that it reads and later matches these to the numbers of d's and c's, respectively. This means that the stack symbols by themselves, without the lists of indices, ensure that the input string is of the form $a^n b^m c^m d^n$.

The second mechanism consists of the manipulation of indices, which takes place conceptually to the right of the spine (which is why we refer to these automata as right-oriented). This mechanism is initiated only after reading the first c, as indicated in Figure 2. The indices ensure that the number of d's equals the number of c's, and only due to the first mechanism this also ensures that the number of a's equals the number of b's.

A consequence is that the automaton will detect that $a^1 b^2 c^2 d^2$ is incorrect only *after* it has read the complete input, when it finds no applicable transition for the configuration $(I[]\ X[],\ \varepsilon)$. However, the error position here is the third input position, where the second b already indicates that the numbers of a's and b's do not match. Therefore the automaton does not satisfy the correct-prefix property.

On the other hand, the L-LIA in Figure 3 "almost" satisfies the correct-prefix property. While it reads first a's and then b's, it manipulates the indices to ensure an equal number of each is read, predicting the required number of c's and d's to be read later, when the stack starts shrinking, and C's and D's are popped.

The only obstacle to the correct-prefix property is caused by the transition $P[\circ\circ] \overset{\varepsilon}{\mapsto} Z[\circ\circ]\ B[]$. If the top of stack is formed by $P[]$, the transition may replace this by $Z[]$ and $B[]$. After $B[]$ is popped, consuming one b, the top of stack is $Z[]$. To this however no transitions apply.

This situation occurs when the input starts with $a^m b^{m+1}$ $(m > 0)$. The error is detected too late, after the terminal at the error position (the b at position $2m+1$) has already been consumed. A simple patch consists in replacing the transition above by three other transitions:

$$
\begin{array}{rcl}
P[\circ\circ p] & \overset{\varepsilon}{\mapsto} & P'[\circ\circ] \\
P'[\circ\circ] & \overset{\varepsilon}{\mapsto} & P''[\circ\circ p] \\
P''[\circ\circ] & \overset{\varepsilon}{\mapsto} & Z[\circ\circ]\ B[]
\end{array}
$$

$$
\begin{array}{rcl}
S[\circ\circ] & \stackrel{\varepsilon}{\mapsto} & X[\circ\circ]\ A[] \\
X[\circ\circ] & \stackrel{\varepsilon}{\mapsto} & \nabla_2[\circ\circ p] \\
\nabla_2[\circ\circ] & \stackrel{\varepsilon}{\mapsto} & D[]\ Y[\circ\circ] \\
Y[\circ\circ] & \stackrel{\varepsilon}{\mapsto} & X[\circ\circ]\ A[] \\
Y[\circ\circ] & \stackrel{\varepsilon}{\mapsto} & Z[\circ\circ]\ B[] \\
Z[\circ\circ p] & \stackrel{\varepsilon}{\mapsto} & \nabla_5[\circ\circ] \\
\nabla_5[\circ\circ] & \stackrel{\varepsilon}{\mapsto} & C[]\ P[\circ\circ] \\
P[\circ\circ] & \stackrel{\varepsilon}{\mapsto} & Z[\circ\circ]\ B[] \\
R[\circ\circ]\ P[] & \stackrel{\varepsilon}{\mapsto} & R[\circ\circ],\ \textit{any } R \\
R[\circ\circ]\ A[] & \stackrel{a}{\mapsto} & R[\circ\circ],\ \textit{any } R \\
R[\circ\circ]\ B[] & \stackrel{b}{\mapsto} & R[\circ\circ],\ \textit{any } R \\
R[\circ\circ]\ C[] & \stackrel{c}{\mapsto} & R[\circ\circ],\ \textit{any } R \\
R[\circ\circ]\ D[] & \stackrel{d}{\mapsto} & R[\circ\circ],\ \textit{any } R \\
I[\circ\circ] & \stackrel{\varepsilon}{\mapsto} & F[\circ\circ]\ S[]
\end{array}
$$

Stack	Input	Next step
$I[]$	$aabbccdd$	$I[\circ\circ] \stackrel{\varepsilon}{\mapsto} F[\circ\circ]\ S[]$
$F[]\ S[]$	$aabbccdd$	$S[\circ\circ] \stackrel{\varepsilon}{\mapsto} X[\circ\circ]\ A[]$
$F[]\ X[]\ A[]$	$aabbccdd$	$X[\circ\circ]\ A[] \stackrel{a}{\mapsto} X[\circ\circ]$
$F[]\ X[]$	$abbccdd$	$X[\circ\circ] \stackrel{\varepsilon}{\mapsto} \nabla_2[\circ\circ p]$
$F[]\ \nabla_2[p]$	$abbccdd$	$\nabla_2[\circ\circ] \stackrel{\varepsilon}{\mapsto} D[]\ Y[\circ\circ]$
$F[]\ D[]\ Y[p]$	$abbccdd$	$Y[\circ\circ] \stackrel{\varepsilon}{\mapsto} X[\circ\circ]\ A[]$
$F[]\ D[]\ X[p]\ A[]$	$abbccdd$	$X[\circ\circ]\ A[] \stackrel{a}{\mapsto} X[\circ\circ]$
$F[]\ D[]\ X[p]$	$bbccdd$	$X[\circ\circ] \stackrel{\varepsilon}{\mapsto} \nabla_2[\circ\circ p]$
$F[]\ D[]\ \nabla_2[pp]$	$bbccdd$	$\nabla_2[\circ\circ] \stackrel{\varepsilon}{\mapsto} D[]\ Y[\circ\circ]$
$F[]\ D[]\ D[]\ Y[pp]$	$bbccdd$	$Y[\circ\circ] \stackrel{\varepsilon}{\mapsto} Z[\circ\circ]\ B[]$
$F[]\ D[]\ D[]\ Z[pp]\ B[]$	$bbccdd$	$Z[\circ\circ]\ B[] \stackrel{b}{\mapsto} Z[\circ\circ]$
$F[]\ D[]\ D[]\ Z[pp]$	$bccdd$	$Z[\circ\circ p] \stackrel{\varepsilon}{\mapsto} \nabla_5[\circ\circ]$
$F[]\ D[]\ D[]\ \nabla_5[p]$	$bccdd$	$\nabla_5[\circ\circ] \stackrel{\varepsilon}{\mapsto} C[]\ P[\circ\circ]$
$F[]\ D[]\ D[]\ C[]\ P[p]$	$bccdd$	$P[\circ\circ] \stackrel{\varepsilon}{\mapsto} Z[\circ\circ]\ B[]$
$F[]\ D[]\ D[]\ C[]\ Z[p]\ B[]$	$bccdd$	$Z[\circ\circ]\ B[] \stackrel{b}{\mapsto} Z[\circ\circ]$
$F[]\ D[]\ D[]\ C[]\ Z[p]$	$ccdd$	$Z[\circ\circ p] \stackrel{\varepsilon}{\mapsto} \nabla_5[\circ\circ]$
$F[]\ D[]\ D[]\ C[]\ \nabla_5[]$	$ccdd$	$\nabla_5[\circ\circ] \stackrel{\varepsilon}{\mapsto} C[]\ P[\circ\circ]$
$F[]\ D[]\ D[]\ C[]\ C[]\ P[]$	$ccdd$	$C[\circ\circ]\ P[] \stackrel{\varepsilon}{\mapsto} C[\circ\circ]$
$F[]\ D[]\ D[]\ C[]\ C[]$	$ccdd$	$C[\circ\circ]\ C[] \stackrel{c}{\mapsto} C[\circ\circ]$
$F[]\ D[]\ D[]\ C[]$	cdd	$D[\circ\circ]\ C[] \stackrel{c}{\mapsto} D[\circ\circ]$
$F[]\ D[]\ D[]$	dd	$D[\circ\circ]\ D[] \stackrel{d}{\mapsto} D[\circ\circ]$
$F[]\ D[]$	d	$F[\circ\circ]\ D[] \stackrel{d}{\mapsto} F[\circ\circ]$
$F[]$	ε	recognition

Figure 3: Transitions of the simplified L-LIA, and a sequence of configurations for input *aabbccdd*.

The effect of these transitions is that the list of indices is first verified to be non-empty before the next b is read. The new automaton does satisfy the correct-prefix property.

As shown in Appendix A, a L-LIA with the correct-prefix property can be constructed for any LIG.

5 Tabulation for R-LIA

The right-oriented and left-oriented linear indexed automata manipulate stacks on two levels, since the lists of indices act as stacks embedded in the other stack formed by elements from $\mathcal{Q} \times \mathcal{I}^*$. (We will further abstain from referring to lists of indices as stacks in order to avoid the obvious confusion that would ensue from this.) The consequence is that we can apply in a 2-dimensional way the tabulation technique from Lang (1974) and Billot and Lang (1989), which was originally devised for pushdown automata with only one kind of stack. The nature of the resulting tabular algorithm for R-LIA shows some similarities to the tabular, LR-like algorithm from Alonso Pardo et al. (1997). In fact, it can be seen as a generalization of this algorithm, since we will formulate tabulation independently of the parsing strategy.

The algorithm is described as follows. Given a right-oriented linear indexed automaton M and an input v, we construct a table U in polynomial time. From the presence of a certain object in U, we can effectively decide whether the input belongs to the language. The procedure can be extended so that a representation of all parse trees, the *parse forest*, is produced as side-effect of the construction of U.

The objects in the table U will be called *items* and consist of 9 fields which we pragmatically divide into two 4-tuples separated by an index: $((X, Y, i, j), p, (Z, P, k, l))$, where $X, Y, Z, P \in \mathcal{Q}$, $p \in \mathcal{I}$ and i, j, k, l are natural numbers between 0 and n, representing positions in the input $v = a_1 \cdots a_n \in \Sigma^*$.

The meaning of the first 4-tuple is almost unchanged with respect to the original tabulation method for pushdown automata with a single stack, mentioned above: by reading the input from position i to j we may replace stack symbol X by Y. The remainder of an item contains information with regard to the list of indices that is associated with Y: its head is p and its tail is a list that is associated with stack symbol P which results by replacing Z by P while reading the input from k to l. See Figure 4 for a pictorial representation.

More precisely, an element $((X, Y, i, j),\ p,$

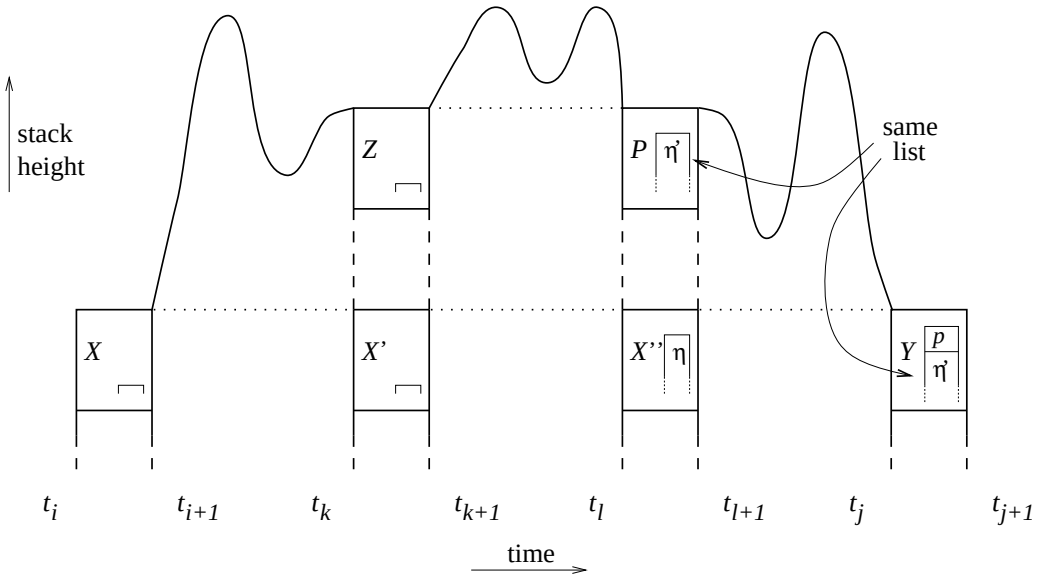

Figure 4: Meaning of an item $((X, Y, i, j), p, (Z, P, k, l))$. Time t_m is when a_m is read.

$(Z, P, k, l))$ indicates the existence of a sequence of steps of the automaton of the form $(\alpha\ X[],\ a_{i+1} \cdots a_n) \vdash^* (\alpha\ X'[]\ \beta\ Z[],\ a_{k+1} \cdots a_n) \vdash^* (\alpha\ X''[\eta]\ \beta\ P[\eta'],\ a_{l+1} \cdots a_n) \vdash^* (\alpha\ Y[\eta' p],\ a_{j+1} \cdots a_n)$, where $\alpha, \beta \in (\mathcal{Q} \times \mathcal{I}^*)^*$ and $\eta, \eta' \in \mathcal{I}^*$, and

- nowhere between $(\alpha\ X[],\ a_{i+1} \cdots a_n)$ and $(\alpha\ Y[\eta' p],\ a_{j+1} \cdots a_n)$ does the stack shrink to below the height of $\alpha\ X[]$;
- nowhere between $(\alpha\ X'[]\ \beta\ Z[],\ a_{k+1} \cdots a_n)$ and $(\alpha\ X''[\eta]\ \beta\ P[\eta'],\ a_{l+1} \cdots a_n)$ does the stack shrink to below the height of $\alpha\ X'[]\ \beta\ Z[]$; and
- the two occurrences of η' are the same list in the sense that it is passed on unaffected through the steps from $(\alpha\ X''[\eta]\ \beta\ P[\eta'],\ a_{l+1} \cdots a_n)$ to $(\alpha\ Y[\eta' p],\ a_{j+1} \cdots a_n)$. It is allowed that elements from $\mathcal{I}$ 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.

If we want to indicate that the list of indices associated with Y is empty, we fill the last 5 fields of an item with "dummy" elements: $((X, Y, i, j), \diamond, (\square, \square, 0, 0))$, where $\diamond$ is a fresh symbol representing an imaginary index in an empty list of indices, and similarly $\square$ is a "dummy" stack symbol.

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 items that need to be computed for recognition of *aabbccdd*, using the R-LIA from Figure 2, are indicated in Figure 5.

The very first item to be added to the table is due to an inference rule without antecedents or side conditions:

$$\frac{}{((I, I, 0, 0), \diamond, (\square, \square, 0, 0))}$$

The following rule is not used in the running example:

$$\frac{((X, Y, i, j), p, (Z, P, k, l))}{((X, Y', i, j), p, (Z, P, k, l))} \left\{ \begin{array}{l} Y[\circ\circ] \overset{\varepsilon}{\mapsto} Y'[\circ\circ] \end{array} \right.$$

For deriving items 8 and 12 this rule is used:

$$\frac{((X, Y, i, j), p, (Z, P, k, l))}{((X, Y', i, j), q, (X, Y, i, j))} \left\{ \begin{array}{l} Y[\circ\circ] \overset{\varepsilon}{\mapsto} Y'[\circ\circ q] \end{array} \right.$$

Items 16 and 20 are derived by:

$$\frac{\begin{array}{l}((Z, P, k, l), q, (Z', P', k', l')) \\ ((X, Y, i, j), p, (Z, P, k, l))\end{array}}{((X, Y', i, j), q, (Z', P', k', l'))} \left\{ \begin{array}{l} Y[\circ\circ p] \overset{\varepsilon}{\mapsto} Y'[\circ\circ] \end{array} \right.$$

The following is used for items 1-6, 10, 14 and 18.

$$\frac{((X, Y, i, j), p, (Z, P, k, l))}{((X', X', j', j'), \diamond, (\square, \square, 0, 0))} \left\{ \begin{array}{l} Y[\circ\circ] \overset{z}{\mapsto} R[\circ\circ]\ X'[] \\ z{=}\varepsilon \wedge j'{=}j\ \ \vee\ \ z{=}a_{j'} \wedge j'{=}j{+}1 \end{array} \right.$$

Nr.	Item in U	Derived from
0	$((I,I,0,0),\diamond,(\square,\square,0,0))$	initial item
1	$((A,A,1,1),\diamond,(\square,\square,0,0))$	$I[\circ\circ] \overset{a}{\mapsto} I[\circ\circ]\ A[\,]$ and 0
2	$((A,A,2,2),\diamond,(\square,\square,0,0))$	$A[\circ\circ] \overset{a}{\mapsto} A[\circ\circ]\ A[\,]$ and 1
3	$((B,B,3,3),\diamond,(\square,\square,0,0))$	$A[\circ\circ] \overset{b}{\mapsto} A[\circ\circ]\ B[\,]$ and 2
4	$((B,B,4,4),\diamond,(\square,\square,0,0))$	$B[\circ\circ] \overset{b}{\mapsto} B[\circ\circ]\ B[\,]$ and 3
5	$((P,P,4,4),\diamond,(\square,\square,0,0))$	$B[\circ\circ] \overset{\varepsilon}{\mapsto} B[\circ\circ]\ P[\,]$ and 4
6	$((C,C,5,5),\diamond,(\square,\square,0,0))$	$P[\circ\circ] \overset{c}{\mapsto} P[\circ\circ]\ C[\,]$ and 5
7	$((P,\nabla_5,4,5),\diamond,(\square,\square,0,0))$	$P[\circ\circ] \overset{c}{\mapsto} P[\circ\circ]\ C[\,]$ and $P[\circ\circ]\ C[\,] \overset{\varepsilon}{\mapsto} \nabla_5[\circ\circ]$ and $5+6$
8	$((P,Z,4,5),p,(P,\nabla_5,4,5))$	$\nabla_5[\circ\circ] \overset{\varepsilon}{\mapsto} Z[\circ\circ p]$ and 7
9	$((B,P,4,5),p,(P,\nabla_5,4,5))$	$B[\circ\circ] \overset{\varepsilon}{\mapsto} B[\circ\circ]\ P[\,]$ and $B[\,]\ Z[\circ\circ] \overset{\varepsilon}{\mapsto} P[\circ\circ]$ and $4+8$
10	$((C,C,6,6),\diamond,(\square,\square,0,0))$	$P[\circ\circ] \overset{c}{\mapsto} P[\circ\circ]\ C[\,]$ and 9
11	$((B,\nabla_5,4,6),p,(P,\nabla_5,4,5))$	$P[\circ\circ] \overset{c}{\mapsto} P[\circ\circ]\ C[\,]$ and $P[\circ\circ]\ C[\,] \overset{\varepsilon}{\mapsto} \nabla_5[\circ\circ]$ and $9+10$
12	$((B,Z,4,6),p,(B,\nabla_5,4,6))$	$\nabla_5[\circ\circ] \overset{\varepsilon}{\mapsto} Z[\circ\circ p]$ and 11
13	$((B,Y,3,6),p,(B,\nabla_5,4,6))$	$B[\circ\circ] \overset{b}{\mapsto} B[\circ\circ]\ B[\,]$ and $B[\,]\ Z[\circ\circ] \overset{\varepsilon}{\mapsto} Y[\circ\circ]$ and $3+12$
14	$((D,D,7,7),\diamond,(\square,\square,0,0))$	$Y[\circ\circ] \overset{d}{\mapsto} Y[\circ\circ]\ D[\,]$ and 13
15	$((B,\nabla_2,3,7),p,(B,\nabla_5,4,6))$	$Y[\circ\circ] \overset{d}{\mapsto} Y[\circ\circ]\ D[\,]$ and $Y[\circ\circ]\ D[\,] \overset{\varepsilon}{\mapsto} \nabla_2[\circ\circ]$ and $13+14$
16	$((B,X,3,7),p,(P,\nabla_5,4,5))$	$\nabla_2[\circ\circ p] \overset{\varepsilon}{\mapsto} X[\circ\circ]$ and $11+15$
17	$((A,Y,2,7),p,(P,\nabla_5,4,5))$	$A[\circ\circ] \overset{b}{\mapsto} A[\circ\circ]\ B[\,]$ and $A[\,]\ X[\circ\circ] \overset{\varepsilon}{\mapsto} Y[\circ\circ]$ and $2+16$
18	$((D,D,8,8),\diamond,(\square,\square,0,0))$	$Y[\circ\circ] \overset{d}{\mapsto} Y[\circ\circ]\ D[\,]$ and 17
19	$((A,\nabla_2,2,8),p,(P,\nabla_5,4,5))$	$Y[\circ\circ] \overset{d}{\mapsto} Y[\circ\circ]\ D[\,]$ and $Y[\circ\circ]\ D[\,] \overset{\varepsilon}{\mapsto} \nabla_2[\circ\circ]$ and $17+18$
20	$((A,X,2,8),\diamond,(\square,\square,0,0))$	$\nabla_2[\circ\circ p] \overset{\varepsilon}{\mapsto} X[\circ\circ]$ and $7+19$
21	$((A,S,1,8),\diamond,(\square,\square,0,0))$	$A[\circ\circ] \overset{a}{\mapsto} A[\circ\circ]\ A[\,]$ and $A[\,]\ X[\circ\circ] \overset{\varepsilon}{\mapsto} S[\circ\circ]$ and $1+20$
22	$((I,F,0,8),\diamond,(\square,\square,0,0))$	$I[\circ\circ] \overset{a}{\mapsto} I[\circ\circ]\ A[\,]$ and $I[\circ\circ]\ S[\,] \overset{\varepsilon}{\mapsto} F[\circ\circ]$ and $0+21$

Figure 5: Tabular recognition of the input $a_1 a_2 \cdots a_8 = aabbccdd$, using the R-LIA.

For items 9, 13, 17 and 21 we need:

$$\frac{\begin{array}{l}((X,Y,i,j),\diamond,(\square,\square,0,0))\\ ((X',X'',j',m),p,(Z,P,k,l))\end{array}}{((X,Y',i,m),p,(Z,P,k,l))} \quad \begin{cases} Y[\circ\circ] \overset{z}{\mapsto} R[\circ\circ]\ X'[\,] \\ R[\,]\ X''[\circ\circ] \overset{\varepsilon}{\mapsto} Y'[\circ\circ] \\ z{=}\varepsilon \wedge j'{=}j \ \vee\ z{=}a_{j'} \wedge j'{=}j{+}1 \end{cases}$$

Finally, items 7, 11, 15, 19 and 22 are derived by:

$$\frac{\begin{array}{l}((X,Y,i,j),p,(Z,P,k,l))\\ ((X',X'',j',m),\diamond,(\square,\square,0,0))\end{array}}{((X,Y',i,m'),p,(Z,P,k,l))} \quad \begin{cases} Y[\circ\circ] \overset{z}{\mapsto} R[\circ\circ]\ X'[\,] \\ R[\circ\circ]\ X''[\,] \overset{z'}{\mapsto} Y'[\circ\circ] \\ z{=}\varepsilon \wedge j'{=}j \ \vee\ z{=}a_{j'} \wedge j'{=}j{+}1 \\ z'{=}\varepsilon \wedge m'{=}m \ \vee\ z'{=}a_{m'} \wedge m'{=}m{+}1 \end{cases}$$

We accept the input if and only if we can derive $((I,F,0,n),\diamond,(\square,\square,0,0))$. In the running example, with input length $n=8$, this condition is fulfilled. The proof of correctness will not be discussed here.

In none of the inference rules there are more than 6 independent input positions. This implies that the algorithm can straightforwardly be implemented to have a time-complexity of $\mathcal{O}(n^6)$.

6 Tabulation for L-LIA

Tabulation for L-LIA cannot simply be obtained by reversing the tabulation algorithm from the previous section, if we assume that as before the input is to be read from left to right. We also demand the tabulation algorithm to preserve the correct-prefix property, i.e. if the L-LIA has this property then so does the tabulation algorithm.

In order to satisfy these requirements, we need to introduce an additional type of item, which is of the form $(R,\ m,\ (X,Y,i,j),\ p)$.[2] A pictorial representation of the meaning of such an item is given in Figure 6.

More precisely, an item $(R,m,(X,Y,i,j),p)$ indicates the existence of a sequence of steps of the form

[2] A very similar structure is called *CF item* by Villemonte de la Clergerie and Alonso Pardo (1998).

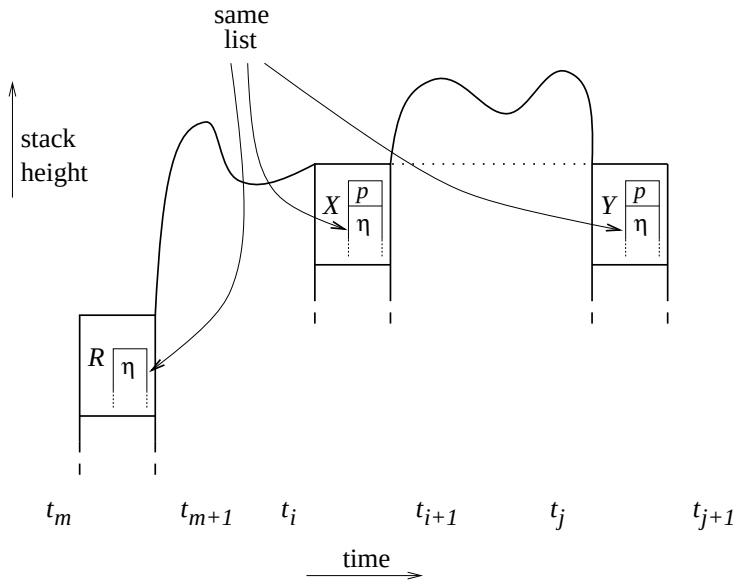

Figure 6: Meaning of an item $(R, m, (X, Y, i, j), p)$.

$(\alpha\ R[\eta],\ a_{m+1} \cdots a_n) \vdash^* (\alpha\ \beta\ X[\eta p],\ a_{i+1} \cdots a_n) \vdash^* (\alpha\ \beta\ Y[\eta p],\ a_{j+1} \cdots a_n)$, where

- the three occurrences of η are the same list in the sense that it is passed on unaffected.

The type of item we used for R-LIAs needs to be extended in order to adapt it to L-LIAs. Such an item now has the form $(R, m, (X, Y, i, j),\ p,\ (Z, P, k, l))$, and its meaning is presented by Figure 7. In such an item, the variables $X, Y, i, j, p, Z, P, k, l$ have the same function as before, except that the manipulation of the indices occurs mirrored with respect to Figure 4, due to left orientation taking the place of right orientation. R and m have the same meaning as in the items of the first form discussed above.

Below we will discuss the new inference rules, some of which are demonstrated for the running example in Figure 8.

The initial item results from:

$$\frac{}{(\Box, 0, (I, I, 0, 0), \diamond, (\Box, \Box, 0, 0))}$$

The following four rules are not used in the running example:

$$\frac{(R, m, (X, Y, i, j), p)}{(R, m, (X, Y', i, j), p)} \left\{\ Y[\circ\circ] \overset{\varepsilon}{\mapsto} Y'[\circ\circ]\right.$$

$$\frac{(R, m, (X, Y, i, j), p)}{(Y, j, (Y', Y', j, j), q)} \left\{\ Y[\circ\circ] \overset{\varepsilon}{\mapsto} Y'[\circ\circ q]\right.$$

$$\frac{\begin{array}{l}(R', m', (X', R, k, m), q)\\ (R, m, (X, Y, i, j), p)\end{array}}{(R', m', (X, Y', i, j), q)} \left\{\ Y[\circ\circ p] \overset{\varepsilon}{\mapsto} Y'[\circ\circ]\right.$$

$$\frac{(R, m, (X, Y, i, j), p, (Z, P, k, l))}{(R, m, (X, Y', i, j), p, (Z, P, k, l))} \left\{\ Y[\circ\circ] \overset{\varepsilon}{\mapsto} Y'[\circ\circ]\right.$$

The following rule is used to derive item 4:

$$\frac{(R, m, (X, Y, i, j), p, (Z, P, k, l))}{(\Box, 0, (Y', Y', j, j), q)} \left\{\ Y[\circ\circ] \overset{\varepsilon}{\mapsto} Y'[\circ\circ q]\right.$$

The following rule marks the boundary between processing the left and right sides of the spine, and is used for deriving item 8.

$$\frac{(\Box, 0, (X, Y, i, j), p)}{(\Box, 0, (X, Y', i, j), p, (\Box, \Box, 0, 0))} \left\{\ Y[\circ\circ p] \overset{\varepsilon}{\mapsto} Y'[\circ\circ]\right.$$

We derive item 12 using:

$$\frac{\begin{array}{l}(R, m, (X, Y, i, j), p, (Z, P, k, l))\\ (\Box, 0, (X', Y', j, j'), q, (\Box, \Box, 0, 0))\end{array}}{(R, m, (X, Y', i, j'), p, (Z, P, k, l))} \left\{\ Y[\circ\circ] \overset{\varepsilon}{\mapsto} X'[\circ\circ q]\right.$$

The following two rules we do not use in the running example:

$$\frac{\begin{array}{l}(R, m, (T, R', o, m'), p)\\ (R', m', (X', Y', i', i), q)\\ (R, m, (X, Y, i, j), p, (Z, P, k, l))\end{array}}{(R', m', (X', Y, i', j), q, (X, Y, i, j))} \left\{\ Y'[\circ\circ q] \overset{\varepsilon}{\mapsto} X[\circ\circ]\right.$$

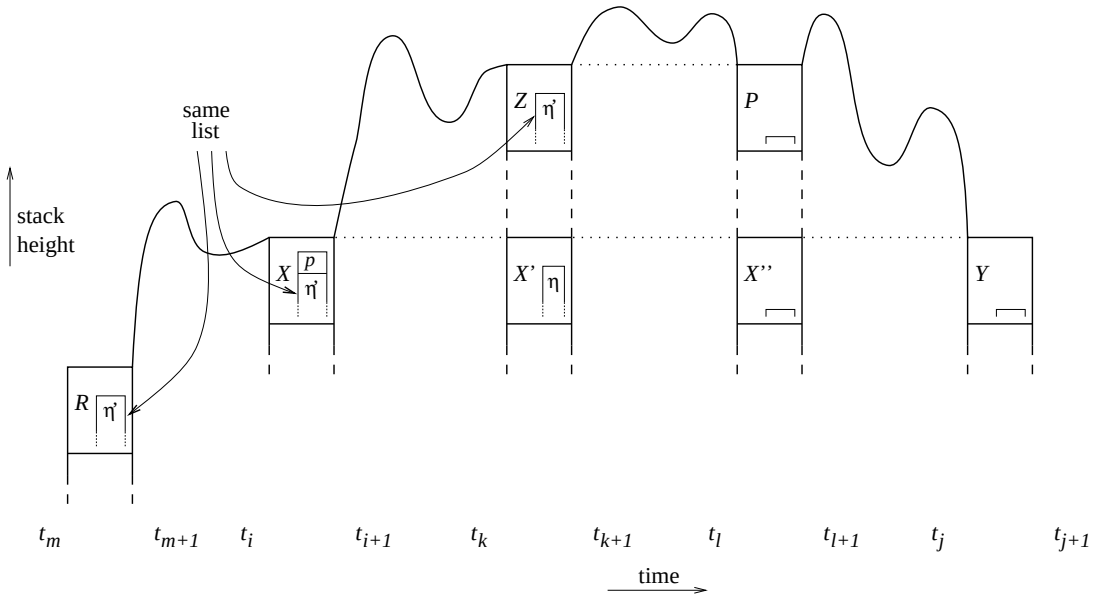

Figure 7: Meaning of an item $(R, m, (X, Y, i, j), p, (Z, P, k, l))$.

Nr.	Item in U	Derived from
0	$(\square, 0, (I, I, 0, 0) \quad , \diamond, (\square, \square, 0, 0))$	initial item
1	$(\square, 0, (S, S, 0, 0) \quad , \diamond, (\square, \square, 0, 0))$	$I[\circ\circ] \overset{\varepsilon}{\mapsto} F[\circ\circ]\ S[]$ and 0
2	$(\square, 0, (A, A, 0, 0) \quad , \diamond, (\square, \square, 0, 0))$	$S[\circ\circ] \overset{\varepsilon}{\mapsto} X[\circ\circ]\ A[]$ and 1
3	$(\square, 0, (S, X, 0, 1) \quad , \diamond, (\square, \square, 0, 0))$	$S[\circ\circ] \overset{\varepsilon}{\mapsto} X[\circ\circ]\ A[]$ and $X[\circ\circ]\ A[] \overset{a}{\mapsto} X[\circ\circ]$ and $1+2$
4	$(\square, 0, (\nabla_2, \nabla_2, 1, 1), p)$	$X[\circ\circ] \overset{\varepsilon}{\mapsto} \nabla_2[\circ\circ p]$ and 3
5	$(\square, 0, (Y, Y, 1, 1) \quad , p)$	$\nabla_2[\circ\circ] \overset{\varepsilon}{\mapsto} D[]\ Y[\circ\circ]$ and 4
6	$(\square, 0, (B, B, 1, 1) \quad , \diamond, (\square, \square, 0, 0))$	$Y[\circ\circ] \overset{\varepsilon}{\mapsto} Z[\circ\circ]\ B[]$ and 5
7	$(\square, 0, (Y, Z, 1, 2) \quad , p)$	$Y[\circ\circ] \overset{\varepsilon}{\mapsto} Z[\circ\circ]\ B[]$ and $Z[\circ\circ]\ B[] \overset{b}{\mapsto} Z[\circ\circ]$ and $5+6$
8	$(\square, 0, (Y, \nabla_5, 1, 2) \quad , p, (\square, \square, 0, 0))$	$Z[\circ\circ p] \overset{\varepsilon}{\mapsto} \nabla_5[\circ\circ]$ and 7
9	$(\square, 0, (P, P, 2, 2) \quad , \diamond, (\square, \square, 0, 0))$	$\nabla_5[\circ\circ] \overset{\varepsilon}{\mapsto} C[]\ P[\circ\circ]$ and 8
10	$(\square, 0, (Y, C, 1, 2) \quad , p, (\square, \square, 0, 0))$	$\nabla_5[\circ\circ] \overset{\varepsilon}{\mapsto} C[]\ P[\circ\circ]$ and $C[\circ\circ]\ P[] \overset{\varepsilon}{\mapsto} C[\circ\circ]$ and $8+9$
11	$(\square, 0, (\nabla_2, D, 1, 3) \quad , p, (\square, \square, 0, 0))$	$\nabla_2[\circ\circ] \overset{\varepsilon}{\mapsto} D[]\ Y[\circ\circ]$ and $D[\circ\circ]\ C[] \overset{c}{\mapsto} D[\circ\circ]$ and $4+10$
12	$(\square, 0, (S, D, 0, 3) \quad , \diamond, (\square, \square, 0, 0))$	$X[\circ\circ] \overset{\varepsilon}{\mapsto} \nabla_2[\circ\circ p]$ and $3+11$
13	$(\square, 0, (I, F, 0, 4) \quad , \diamond, (\square, \square, 0, 0))$	$I[\circ\circ] \overset{\varepsilon}{\mapsto} F[\circ\circ]\ S[]$ and $F[\circ\circ]\ D[] \overset{d}{\mapsto} F[\circ\circ]$ and $0+12$

Figure 8: Tabular recognition of the input $a_1a_2a_3a_4 = abcd$, using the L-LIA.

$$\frac{\begin{array}{l}(R, m, (X', Y', i', i), q) \\ (R, m, (Z, P, k, l), q, (Z', P', k', l')) \\ (Y', i, (X, Y, i, j), p, (Z, P, k, l))\end{array}}{(R, m, (X', Y, i', j), q, (Z', P', k', l'))} \left\{ \begin{array}{l} Y'[\circ\circ] \overset{\varepsilon}{\mapsto} X[\circ\circ p] \end{array} \right.$$

For item 6, we use:

$$\frac{(R, m, (X, Y, i, j), p)}{(\square, 0, (X', X', j', j'), \diamond, (\square, \square, 0, 0))} \left\{ \begin{array}{l} Y[\circ\circ] \overset{z}{\mapsto} Z[\circ\circ]\ X'[] \\ z{=}\varepsilon \wedge j'{=}j \ \ \vee \ \ z{=}a_{j'} \wedge j'{=}j{+}1 \end{array} \right.$$

The following is used for item 5:

$$\frac{(R, m, (X, Y, i, j), p)}{(R, m, (X', X', j, j), p)} \left\{ \begin{array}{l} Y[\circ\circ] \overset{\varepsilon}{\mapsto} Z[]\ X'[\circ\circ] \end{array} \right.$$

The following is used for items 1 and 2:

$$\frac{(R, m, (X, Y, i, j), p, (Z, P, k, l))}{(\square, 0, (X', X', j', j'), \diamond, (\square, \square, 0, 0))} \left\{ \begin{array}{l} Y[\circ\circ] \overset{z}{\mapsto} S[\circ\circ]\ X'[] \\ z{=}\varepsilon \wedge j'{=}j \ \ \vee \ \ z{=}a_{j'} \wedge j'{=}j{+}1 \end{array} \right.$$

Item 9 is derived by:

$$\frac{(R,m,(X,Y,i,j),p,(Z,P,k,l))}{(\Box,0,(X',X',j,j),\Diamond,(\Box,\Box,0,0))} \quad \{\ Y[\circ\circ] \overset{\varepsilon}{\mapsto} S[\,]\ X'[\circ\circ]$$

The following is used for item 7.

$$\frac{(R,m,(X,Y,i,j),p) \quad (\Box,0,(X',Y',j',k),\Diamond,(\Box,\Box,0,0))}{(R,m,(X,P,i,k'),p)} \begin{cases} Y[\circ\circ] \overset{z}{\mapsto} Z[\circ\circ]\ X'[\,] \\ Z[\circ\circ]\ Y'[\,] \overset{z'}{\mapsto} P[\circ\circ] \\ z{=}\varepsilon \wedge j'{=}j \ \vee\ z{=}a_{j'} \wedge j'{=}j{+}1 \\ z'{=}\varepsilon \wedge k'{=}k \ \vee\ z'{=}a_{k'} \wedge k'{=}k{+}1 \end{cases}$$

The following is used for item 11.

$$\frac{(R,m,(X,Y,i,j),p) \quad (R,m,(X',X'',j,o),p,(Z,P,k,l))}{(R,m,(X,Y',i,o'),p,(Z,P,k,l))} \begin{cases} Y[\circ\circ] \overset{\varepsilon}{\mapsto} S[\,]\ X'[\circ\circ] \\ S[\circ\circ]\ X''[\,] \overset{z}{\mapsto} Y'[\circ\circ] \\ z{=}\varepsilon \wedge o'{=}o \ \vee\ z{=}a_{o'} \wedge o'{=}o{+}1 \end{cases}$$

Items 3 and 13 are derived by:

$$\frac{(R,m,(X,Y,i,j),p,(Z,P,k,l)) \quad (\Diamond,0,(X',X'',j',o),\Diamond,(\Box,\Box,0,0))}{(R,m,(X,Y',i,o'),p,(Z,P,k,l))} \begin{cases} Y[\circ\circ] \overset{z}{\mapsto} S[\circ\circ]\ X'[\,] \\ S[\circ\circ]\ X''[\,] \overset{z'}{\mapsto} Y'[\circ\circ] \\ z{=}\varepsilon \wedge j'{=}j \ \vee\ z{=}a_{j'} \wedge j'{=}j{+}1 \\ z'{=}\varepsilon \wedge o'{=}o \ \vee\ z'{=}a_{o'} \wedge o'{=}o{+}1 \end{cases}$$

Item 10 is derived by:

$$\frac{(R,m,(X,Y,i,j),p,(Z,P,k,l)) \quad (\Diamond,0,(X',X'',j,o),\Diamond,(\Box,\Box,0,0))}{(R,m,(X,Y',i,o'),p,(Z,P,k,l))} \begin{cases} Y[\circ\circ] \overset{\varepsilon}{\mapsto} S[\,]\ X'[\circ\circ] \\ S[\circ\circ]\ X''[\,] \overset{z}{\mapsto} Y'[\circ\circ] \\ z{=}\varepsilon \wedge o'{=}o \ \vee\ z{=}a_{o'} \wedge o'{=}o{+}1 \end{cases}$$

The criterion for recognition is derivability of $(\Box,0,(I,F,0,n),\Diamond,(\Box,\Box,0,0))$.

The maximum number of input positions per inference rule is now 8, which means that the algorithm can straightforwardly be implemented to have a time-complexity of $\mathcal{O}(n^8)$. We conjecture however that using an idea from Nederhof (1997) this can be reduced to $\mathcal{O}(n^6)$. The space-complexity is $\mathcal{O}(n^5)$, since items contain 3 or 5 (i.e. maximally 5) input positions.

Our automata differ in two ways from the SD-2SA by Villemonte de la Clergerie and Alonso Pardo (1998). First, we use stacks formed by the lists of indices nested inside the main stack, whereas SD-2SA apply two separate, but cooperating, stacks. Our nested stacks allow an intuitive generalization of the tabulation technique for pushdown automata as developed by Lang (1974) and Billot and Lang (1989). This matter is purely presentational.

Second, our automata are less general: only at one side of a spine can indices be manipulated. Therefore fewer details need to be solved in the tabulation algorithm. For R-LIA, this also results in a lower time-complexity and in a lower space-complexity.

An automaton model that manipulates indices at both sides of a spine, in such a way that there is symmetry between manipulation on the two sides (a pop or push of an index on one side should be mirrored by a push or pop on the other side), can also be formulated in our notation. At this moment, the theoretical and practical advantages of such an automaton model are not clear however. This may be the subject of further study.

7 Conclusions

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 linear indexed automata, which are nondeterministic recognizers. The mechanisms of tabulation are independent of the strategy and can turn any left-oriented or right-oriented linear indexed automaton into a tabular recognizer.

Acknowledgments

I would like to thank Tilman Becker, Eric de la Clergerie and David J. Weir for interesting discussions.

This work was funded 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.

References

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

T. Becker. 1994. A new automaton model for TAGs: 2-SA. *Computational Intelligence*, 10(4):422–430.

S. Billot and B. Lang. 1989. The structure of shared forests in ambiguous parsing. In *27th*

Annual Meeting of the Association for Computational Linguistics, pages 143–151, Vancouver, British Columbia, Canada, June.

E. de la Clergerie, M.A. 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*, pages 42–45. Institute for Research in Cognitive Science, University of Pennsylvania, August.

A.K. Joshi, K. Vijay-Shanker, and D. Weir. 1991. The convergence of mildly context-sensitive grammar formalisms. In P. Sells, S.M. Shieber, and T. Wasow, editors, *Foundational Issues in Natural Language Processing*, chapter 2, pages 31–81. MIT Press.

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

B. Lang. 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, November.

B. Lang. 1988b. The systematic construction of Earley parsers: Application to the production of $\mathcal{O}(n^6)$ Earley parsers for tree adjoining grammars. Unpublished paper, December.

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

H.R. Lewis and C.H. Papadimitriou. 1981. *Elements of the Theory of Computation*. Prentice-Hall.

M.-J. Nederhof. 1996. Reversible pushdown automata and bidirectional parsing. In J. Dassow, G. Rozenberg, and A. Salomaa, editors, *Developments in Language Theory II*, pages 472–481. World Scientific, Singapore.

M.-J. Nederhof. 1997. Solving the correct-prefix property for TAGs. In T. Becker and H.-U. Krieger, editors, *Proceedings of the Fifth Meeting on Mathematics of Language*, pages 124–130. Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, August. Accepted for publication in Computational Linguistics.

M.-J. Nederhof. 1998a. An alternative LR algorithm for TAGs. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 946–952, Montreal, Quebec, Canada, August.

M.-J. Nederhof. 1998b. 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)*, pages 1–9, Paris, France, April.

F.C.N. Pereira and D.H.D. Warren. 1980. Definite clause grammars for language analysis—a survey of the formalism and a comparison with the augmented transition networks. *Artificial Intelligence*, 13:231–278.

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

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

S. Sippu and E. Soisalon-Soininen. 1990. *Parsing Theory, Vol. II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

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

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

K. Vijay-Shanker and D.J. Weir. 1993b. The use of shared forests in tree adjoining grammar parsing. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 384–393, Utrecht, The Netherlands, April.

K. Vijay-Shanker and D.J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–546.

E. Villemonte de la Clergerie 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*, pages 1333–1339, Montreal, Quebec, Canada, August.

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

A The correct-prefix property for L-LIA

A L-LIA that is constructed as outlined in Section 4 and exemplified by Figure 3 can be shown to satisfy the correct-prefix property under a certain condition that depends on the grammar. The idea behind this condition is that all objects of the form $A[\eta]$ that can be pushed on the stack may eventually be popped again, leaving no objects $B[\eta']$ for which no applicable transitions exist.

Since the constructed L-LIA exhibits top-down behaviour, the objects $A[\eta]$ that can be pushed on the stack may be expressed in terms of the grammar

by:

$$\begin{aligned} &\mathit{Outside}' \;=\; \\ &\quad \{A[\eta] \mid \; S[\,] \Rightarrow^* w\ A[\eta]\ \alpha, A \in \mathcal{N}, \eta \in \mathcal{I}^*, \\ &\qquad\quad w \in \Sigma^*, \alpha \in (\mathcal{N} \times \mathcal{I}^*)^*\} \end{aligned}$$

For the sake of convenience, we will consider a set of objects which might be slightly larger:

$$\begin{aligned} &\mathit{Outside} \;=\; \\ &\quad \{A[\eta] \mid \; C[\,] \Rightarrow^* \alpha\ A[\eta]\ \beta, \\ &\qquad\quad (\ C = S \vee \\ &\qquad\qquad D[\circ\circ\,\eta'] \to B[\circ\circ\,\eta''\,]\ C[\,] \in \mathcal{P} \vee \\ &\qquad\qquad D[\circ\circ\,\eta'] \to C[\,]\ B[\circ\circ\,\eta''\,] \in \mathcal{P}]), \\ &\qquad\quad A, B, C, D \in \mathcal{N}, \eta, \eta', \eta'' \in \mathcal{I}^*, \\ &\qquad\quad \alpha, \beta \in (\mathcal{N} \times \mathcal{I}^*)^*\} \end{aligned}$$

(In the notation above we implicitly assume $A[\eta]$ is the "distinguished descendant" of $C[\,]$ in $\alpha\ A[\eta]\ \beta$.)

The objects $A[\eta]$ which may eventually be popped without leaving a trace is given by:

$$\begin{aligned} &\mathit{Inside} \;=\; \\ &\quad \{A[\eta] \mid A[\eta] \Rightarrow^* w, A \in \mathcal{N}, \eta \in \mathcal{I}^*, w \in \Sigma^*\} \end{aligned}$$

It follows that the correct-prefix property is satisfied if $\mathit{Outside} \subseteq \mathit{Inside}$. This can be determined by considering the languages:

$$\begin{aligned} \mathit{Outside}_A &= \{\eta \mid A[\eta] \in \mathit{Outside}\} \\ \mathit{Inside}_A &= \{\eta \mid A[\eta] \in \mathit{Inside}\} \end{aligned}$$

for each $A \in \mathcal{N}$ separately, and verifying $\mathit{Outside}_A \subseteq \mathit{Inside}_A$. We will call a LIG *pure* if and only if $\mathit{Outside}_A \subseteq \mathit{Inside}_A$ for all $A \in \mathcal{N}$.

These languages of lists of indices are in general infinite, but as we will demonstrate below, we can effectively compute finite representations of these languages in terms of finite automata. Since containment for regular languages is decidable, this provides us with a decision procedure for the correct-prefix property.

For technical reasons, we will transform the grammar so that the "dummy" index $\diamond$ occurs before the actual indices, and thereby lists of indices can no longer be empty. This is done simply by replacing all occurrences of $[\,]$ in right-hand and left-hand sides of productions by $[\diamond]$. We further introduce a new start symbol $S^\dagger$ and add a production $S^\dagger[\circ\circ] \to S[\circ\circ\diamond]$.

For $\mathit{Outside}_A$ we construct triples of the form (A, p, B). Such a triple indicates that $A[\,] \Rightarrow^* \alpha\, B[p]\, \beta$; in other words, a list of indices may grow by the index p when reaching B from A, traversing the spine downwards. A fresh symbol s denotes the start of a spine; it should be seen an imaginary nonterminal for which imaginary rules of the form $s[\circ\circ] \to C[\circ\circ\diamond]$ lead to the initial creation of indices $\diamond$.

The triples will be derived by the following inference rules:

$$\frac{}{(s, \diamond, C)} \left\{ \begin{array}{l} C = S \vee \\ A[\circ\circ\eta] \to B[\circ\circ\eta']\ C[\diamond] \vee \\ A[\circ\circ\eta] \to C[\diamond]\ B[\circ\circ\eta'] \end{array} \right.$$

$$\frac{}{(A, p, B)} \left\{ \begin{array}{l} A[\circ\circ] \to B[\circ\circ p]\ C[\diamond] \vee \\ A[\circ\circ] \to C[\diamond]\ B[\circ\circ p] \end{array} \right.$$

$$\frac{(D, p, A)}{(D, p, B)} \left\{ \begin{array}{l} A[\circ\circ] \to B[\circ\circ]\ C[\diamond] \vee \\ A[\circ\circ] \to C[\diamond]\ B[\circ\circ] \end{array} \right.$$

$$\frac{\begin{array}{l}(E, q, D) \\ (D, p, A)\end{array}}{(E, q, B)} \left\{ \begin{array}{l} A[\circ\circ p] \to B[\circ\circ]\ C[\diamond] \vee \\ A[\circ\circ p] \to C[\diamond]\ B[\circ\circ] \end{array} \right.$$

A triple (A, p, B) obtained by these rules can be interpreted as a transition in a finite automaton from state A to state B with label p. A path from A to B with string η signifies existence of a derivation $A[\,] \Rightarrow^* \alpha\ B[\eta]\ \beta$. The initial state of the automaton is s. For obtaining a particular language $\mathit{Outside}_A$, we complete the automaton by making A the (only) final state.

For the "inside" languages we do something very similar, expect that we need to compute the derivations in reverse, starting from productions $A[\diamond] \to a$. We also verify that nonterminals associated with "empty" lists of indices directly next to the spine generate terminal strings. We now have:

$$\frac{}{(s, \diamond, A)} \left\{ \; A[\diamond] \to a \right.$$

$$\frac{(s, \diamond, C)}{(B, p, A)} \left\{ \begin{array}{l} A[\circ\circ p] \to B[\circ\circ]\ C[\diamond] \vee \\ A[\circ\circ p] \to C[\diamond]\ B[\circ\circ] \end{array} \right.$$

$$\frac{\begin{array}{l}(s, \diamond, C) \\ (D, p, B)\end{array}}{(D, p, A)} \left\{ \begin{array}{l} A[\circ\circ] \to B[\circ\circ]\ C[\diamond] \vee \\ A[\circ\circ] \to C[\diamond]\ B[\circ\circ] \end{array} \right.$$

$$\frac{\begin{array}{l}(s, \diamond, C) \\ (E, q, D) \\ (D, p, B)\end{array}}{(E, q, A)} \left\{ \begin{array}{l} A[\circ\circ] \to B[\circ\circ p]\ C[\diamond] \vee \\ A[\circ\circ] \to C[\diamond]\ B[\circ\circ p] \end{array} \right.$$

The required automaton for a particular language Inside_A is obtained by making A the (only) final state. This completes the definitions needed to decide if the grammar is pure, which would be sufficient to decide that the resulting L-LIA has the correct-prefix property.

If a LIG is not pure, then it can however be transformed into an equivalent and pure LIG. The idea is that each index p is replaced by a pair consisting of the original index p and a set of nonterminals K, so that when an object $A[\eta(p,K)]$ results in a top-down derivation in the transformed grammar, then a derivation $A[\eta' p] \Rightarrow^* w$, for some w, is guaranteed to exist with respect to the original grammar, where η' is the list of indices obtained from η by replacing all pairs (p', K') by p'.

The sets K are formed based on the triples (A, p, B) we derived above for the inside languages. Let us call this set of triples T.

An initial set of nonterminals is given by:

$$K_\diamond = \{A \mid (s, \diamond, A) \in T\}$$

which is to be associated with the conceptually empty list of indices $\diamond$. If a list of indices with top-element (q, K) grows, corresponding to a push of index p in the old grammar, then for the new grammar we obtain a push of (p, K'), where $K' = \{A \mid B \in K, (B, p, A) \in T\}$. We are careful to associate a list of indices with top-element (p, K) only with nonterminals A that satisfy $A \in K$.

The set of productions $\mathcal{P}'$ of the new grammar are specified by the following:

- $\mathcal{P}'$ contains:
  - $S^\dagger[\circ\circ] \rightarrow S[\circ\circ(\diamond, K_\diamond)]$.
- For $A[\circ\circ] \rightarrow B[\circ\circ]\ C[\,]$ in $\mathcal{P}$, with $C \in K_\diamond$, $\mathcal{P}'$ contains:
  - $A[\circ\circ(p,K)] \rightarrow B[\circ\circ(p,K)]\ C[(\diamond, K_\diamond)]$, for all $K \subseteq \mathcal{N}$ satisfying $A, B \in K$.
- For $A[\circ\circ] \rightarrow B[\circ\circ p]\ C[\,]$ in $\mathcal{P}$, with $C \in K_\diamond$, $\mathcal{P}'$ contains:
  - $A[\circ\circ(q,K)] \rightarrow B[\circ\circ(q,K)(p,K')]\ C[(\diamond, K_\diamond)]$, for all $K, K' \subseteq \mathcal{N}$ satisfying $A \in K, B \in K'$ and $K' = \{A' \mid B' \in K, (B', p, A') \in T\}$.
- For $A[\circ\circ p] \rightarrow B[\circ\circ]\ C[\,]$ in $\mathcal{P}$, with $C \in K_\diamond$, $\mathcal{P}'$ contains:
  - $A[\circ\circ(q,K')(p,K)] \rightarrow B[\circ\circ(q,K')]\ C[(\diamond, K_\diamond)]$, for all $K, K' \subseteq \mathcal{N}$ satisfying $A \in K, B \in K'$.
- For productions of the form $A[\circ\circ\eta] \rightarrow C[\,]\ B[\circ\circ\eta']$ in $\mathcal{P}$, the new productions in $\mathcal{P}'$ can be specified similar to the above three cases.
- For $A[\,] \rightarrow a$ in $\mathcal{P}$, with $A \in K_\diamond$, $\mathcal{P}'$ contains:
  - $A[(\diamond, K_\diamond)] \rightarrow a$.

As before, $S^\dagger$ is start symbol in the new grammar.

The resulting grammar is, strictly speaking, not in the binary normal form we presented earlier. However, a trivial grammar transformation leads to this normal form without affecting the "purity" of the grammar. Therefore, after this transformation and the creation of the L-LIA, the correct-prefix property has been satisfied.