A tabular interpretation of a class of 2-Stack Automata

Eric Villemonte de la Clergerie
INRIA - Rocquencourt - B.P. 105
78153 Le Chesnay Cedex, FRANCE
Eric.DeLaClergerie@inria.fr

Miguel Alonso Pardo*
Universidad de La Coruña
Campus de Elviña s/n
15071 La Coruña, SPAIN
alonso@dc.fi.udc.es

Abstract
The paper presents a tabular interpretation for a kind of 2-Stack Automata. These automata may be used to describe various parsing strategies, ranging from purely top-down to purely bottom-up, for LIGs and TAGs. The tabular interpretation ensures, for all strategies, a time complexity in $O(n^6)$ and space complexity in $O(n^6)$ where $n$ is the length of the input string.

Introduction
2-Stack automata [2SA] have been identified as possible operational devices to describe parsing strategies for Linear Indexed Grammars [LIG] or Tree Adjoining Grammars [TAG] (mirroring the traditional use of Push-Down Automata [PDA] for Context-Free Grammars [CFG]). Different variants of 2SA (or not so distant Embedded Push-Down Automata) have been proposed, some to describe top-down strategies (Vijay-Shanker, 1988; Becker, 1994), some to describe bottom-up strategies (Rambow, 1994; Nederhof, 1998; Alonso Pardo et al., 1997), but none (that we know) that are able to describe both kinds of strategies.

The same dichotomy also exists in the different tabular algorithms that has been proposed for specific parsing strategies with complexity ranging from $O(n^6)$ for bottom-up strategies to $O(n^9)$ for prefix-valid top-down strategies (with the exception of a $O(n^6)$ tabular interpretation of a prefix-valid hybrid strategy (Nederhof, 1997)). It must also be noted that the different tabular algorithms may be difficult to understand and is often unclear to know if the algorithms still hold for different strategies.

This paper overcomes these problems by (a) introducing strongly-driven 2SA [SD-2SA] that may be used to describe parsing strategies for TAGs and LIGs, ranging from purely top-down to purely bottom-up, and (b) presenting a tabular interpretation of this automata in time complexity $O(n^6)$ and space complexity $O(n^6)$.

* Partially supported by Government of Spain (project HP07-223) and Xunta de Galicia (project XUGA1005F96).

The tabular interpretation follows the principles of Dynamic Programming: the derivations are broken into elementary sub-derivations that may (a) be combined in different contexts to retrieve all possible derivations and (b) be represented in a compact way by items, allowing tabulation.

The strongly-driven 2SA are introduced and motivated in Section 1. We illustrate in Sections 2 and 3 their power by describing several parsing strategies for LIGs and TAGs. Items are presented in Section 4. Section 5 lists the rules to combine items and transitions and establishes correctness theorems.

1 Strongly-driven 2-Stack Automata
2SA are natural extensions of Push-Down Automata working on a pair of stacks. However, it is known that unrestricted 2SA have the power of a Turing Machine. The remedy is to consider asymmetric stacks, one being the Master Stack MS where most of the work is done and the other being the Auxiliary Stack AS mainly used for restricted "bookkeeping".

The following remarks are intended to give an idea of the restrictions we want to enforce. The first ones are rather standard and may be found under different forms in the literature. The last one justifies the qualification of "strongly-driven" for our automata.

[Session] AS should actually be seen as a stack of session stacks, each one being associated to a session. Only the topmost session stack may be consulted or modified. This idea is closely related to the notion of Embedded Push-Down Automata (Rambow, 1994, 96–102).

[Linearity] A session starts in mode write w and switches at some point in mode erase e. In mode w (resp. e), no element can be popped from (resp. pushed to) the master stack MS. Switching back from e to w is not allowed. This requirement is related to linearity because it means that a same session stack is never used twice by "descendants" of an element in MS.

[Soft Session Exit] Exiting a session is only possible when reaching back, with an empty session...
stack and in mode erase, the MS element that
initiated the session.

[Driving] Each pushing on MS done in write mode
leaves some mark in MS about the action that
took place on the session stack. The popping
of this mark (in erase mode) will guide which
action should take place on the session stack.
In other words, we want the erasing actions to
faithfully retrace the writing actions.

Formally, a SD-2SA $A$ is specified by a tuple
$(\Sigma, M, X, S_0, S_f, \Theta)$ where $\Sigma$ denotes the finite set of
terminals, $M$ the finite set of master stack elements
and $X$ the finite set of auxiliary stack elements. The init
symbol $S_0$ and final symbol $S_f$ are distinguished elements
of $M$. $\Theta$ is a finite set of transitions.
The master stack $MS$ is a word in $(\mathcal{D}M)^*$ where
$\mathcal{D}$ denotes the set $\{\uparrow, \downarrow, \rightarrow, \leftarrow\}$ of action marks
used to remember which action (w.r.t. the auxiliary
stack $AS$) takes place when pushing the next master
stack element. The empty master stack is noted $\epsilon$
and a non-empty master stack $\delta_1 A_1 \cdots \delta_n A_n$ where
$A_n$ denotes the topmost element.

The meaning of the action marks is:

$\uparrow$ Pushing of an element on $AS$.

$\downarrow$ Popping of the topmost element of $AS$.

$\rightarrow$ No action on $AS$.

$\leftarrow$ Creation of a new session (with a new empty
session stack on $AS$).

The auxiliary stack $AS$ is a word of $(\mathcal{K}X^\ast)^*$ where
$\mathcal{K} = \{\downarrow, \uparrow, \leftarrow\}$ is a set of two elements used to
delimit session stacks in $AS$. Delimiter $\downarrow$ (resp.
$\uparrow$) is used to start a new session from a session
which is in writing (resp. erasing) mode. The empty
auxiliary stack is noted $\epsilon$.

Given some input string $x_1 \ldots x_f \in \Sigma^*$, a configuration
of $A$ is a tuple $(m, u, \Xi, \xi)$ where $m \in \{w, e\}$
denotes a mode (writing or erasing), $u$ a string position
in $[0, f]$, $\Xi$ the master stack and $\xi$ the auxiliary
stack. Modes are ordered by $w < e$ to capture the
fact that no switching from $e$ to $w$ is possible. The
initial configuration of $A$ is $(w, 0, \emptyset, \emptyset, w)$ and the
final one $(e, f, \emptyset, \emptyset, w)$.

A transition is given as a pair $(p, \Xi, \xi) \xrightarrow{\rightarrow} (q, \Theta, \theta)$
where $p, q$ are modes (or, with some abuse, variables
ranging over modes), $z$ in $\Sigma^*$, $\Xi$ and $\Theta$ suffixes of
master stacks in $M(\mathcal{D}M)^*$, and $\xi, \theta$ suffixes of auxil-
ary stacks in $X^\ast(\mathcal{K}X^\ast)^* = (X \cup \mathcal{K})^\ast$. Such a transition
applies on any configuration $(p, u, \Xi, \psi, \xi)$ such that
$x_{u+1} \ldots x_z = z$ and returns $(q, v, \Psi, \Theta, \psi)$.

We restrict the kind of allowed transitions:

SWAP $(p, A, \xi) \xrightarrow{\rightarrow} (q, B, \xi')$ with $p < q$ and either
$\xi \in \mathcal{K}$ ("session bottom check") or $\xi = \epsilon$ ("no
AS consultation").

\begin{align*}
\uparrow & \text{-WRITE} (w, A, \epsilon) \xrightarrow{\rightarrow} (w, A \triangleright B, b) \\
\rightarrow & \text{-ERASE} (e, A \triangleright B, a) \xrightarrow{\rightarrow} (e, D, \epsilon) \\
\rightarrow & \text{-WRITE} (w, A, \epsilon) \xrightarrow{\rightarrow} (w, A \rightarrow B, \epsilon) \\
\rightarrow & \text{-ERASE} (e, A \rightarrow B, \epsilon) \xrightarrow{\rightarrow} (e, C, \epsilon) \\
\leftarrow & \text{-WRITE} (m, A, \epsilon) \xrightarrow{\rightarrow} (m, C, \epsilon) \\
\leftarrow & \text{-ERASE} (e, A \uparrow B, \epsilon) \xrightarrow{\rightarrow} (e, C, \epsilon)
\end{align*}

Figure 1 graphically outlines the different kinds of
transitions using a 2D representation where the
X-axis (Y-axis) is related to the master (resp. aux-
iliary) stack. Figure 1 also shows the two forms of
derivations we encounter (during a same session).

2 Using 2SA to parse LIGs

Indexed Grammars (Aho, 1968) are an extension of
Context-free Grammars in which a stack of indices
is associated with each non-terminal symbol. Linear
Indexed Grammars (Gazdar, 1987) are a restricted
form of Indexed Grammars in which the index stack
of at most one body non-terminal (the child) is
related with the stack of the head non-terminal (the
father). The other stacks of the production must
have a bounded stack size.

Formally, a LIG $G$ is a 5-tuple $(V_T, V_N, S, V_I, P)$
where $V_T$ is a finite set of terminals, $V_N$ is a finite
set of non-terminals, $S \in V_N$ is the start symbol, $V_I$ is a finite set of indices and $P$ is a finite set of productions. Following (Gazdar, 1987) we consider productions in which at most one element can be pushed on or popped from a stack of indices:

<table>
<thead>
<tr>
<th>Production Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Terminal]</td>
<td>$A_k,0[\varepsilon] \rightarrow a_k$ where $a_k \in V_T \cup {\varepsilon}$,</td>
</tr>
<tr>
<td>[POP]</td>
<td>$A_k,0[\alpha] \rightarrow A_{k,j}[\cdot \cdot \cdot A_k,\ell[\varepsilon]]$,</td>
</tr>
<tr>
<td>[PUSH]</td>
<td>$A_k,0[\alpha] \rightarrow A_{k,j}[\cdot \cdot \cdot A_k,\ell[\alpha]]$,</td>
</tr>
<tr>
<td>[HOR]</td>
<td>$A_k,0[\alpha] \rightarrow A_{k,j}[\cdot \cdot \cdot A_k,\ell[\alpha]]$,</td>
</tr>
</tbody>
</table>

To each production $k$ of type PUSH, POP or HOR, we associate a characteristic tuple $t(k) = (d, \delta, \alpha, \beta)$ where $d$ is the position of the child and the other arguments given by the following table:

<table>
<thead>
<tr>
<th>Type</th>
<th>$\delta$</th>
<th>$\alpha$</th>
<th>$\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH</td>
<td>$\gamma$</td>
<td>$\varepsilon$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>POP</td>
<td>$\gamma$</td>
<td>$\varepsilon$</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>HOR</td>
<td>$\varepsilon$</td>
<td>$\varepsilon$</td>
<td>$\varepsilon$</td>
</tr>
</tbody>
</table>

We introduce symbols $\nabla_{k,i}$ as a shortcut for dotted productions $\{A_{k,0} \rightarrow A_{k,1}, \ldots A_{k,j}, \ldots A_{k,n_k}\}$.

In order to design a broad class of parsing strategies ranging from pure top-down to pure bottom-up, we parameterize the automaton to be presented by a call projection $\rightarrow$ from $V$ to $V^{call}$ and a return projection $\leftarrow$ from $V$ to $V^{call}$ where $V = V_N \cup V_I$ and $V^{call}$ and $V^{return}$ are two sets of elements. We require $V^{call} \cap V^{return} = \emptyset$ and $(\rightarrow, \leftarrow)$ to be invertible, i.e. $V_X, Y \in V$, $(X, X) = (Y, Y) \Rightarrow X = Y$.

The projections extend to sequences by taking $X_1, \ldots, X_n = X_1'$, $X_n'$ and $X \leftarrow$ (similarly for $\rightarrow$).

Given a LIG $G$ and a choice of productions, we define the 2SA $A(G, \rightarrow, \leftarrow) = (V_T, \mathcal{M}, X, X', S, \Theta)$ where $\mathcal{M} = \{ \nabla_{k,i} \} \cup \mathcal{W} \cup \mathcal{F}$, $X' = \mathcal{W} \cup \mathcal{F}$, and whose transitions are built using the following rules:

- **Call/Return of a non child**
  - $\text{CALL} : (\varepsilon, \nabla_{k,i}[\alpha]) \rightarrow (\varepsilon, \nabla_{k,i}[\alpha], \varepsilon)$
  - $\text{RET} : (e, \nabla_{k,i}[\alpha], \varepsilon) \rightarrow (m, \nabla_{k,i}[\alpha], \varepsilon)$

- **Call/Return of a child for $t(k) = (i + 1, \delta, \alpha, \beta)$**
  - $\text{CALL}(\delta) : (\varepsilon, \nabla_{k,i}[\alpha]) \rightarrow (\varepsilon, \nabla_{k,i}[\alpha], \varepsilon)$
  - $\text{RET}(\delta) : (e, \nabla_{k,i}[\alpha], \varepsilon) \rightarrow (e, \nabla_{k,i}[\alpha], \varepsilon)$

- **Production Selection**
  - $\text{SEL} : (\varepsilon, A_{k,0}[\alpha]) \rightarrow (\varepsilon, \nabla_{k,i}[\alpha], \varepsilon)$

- **Production Publishing**
  - $\text{PUB} : (e, \nabla_{k,i}[\alpha], \varepsilon) \rightarrow (e, \nabla_{k,i}[\alpha], \varepsilon)$

- **Scanning (for terminal productions)**
  - $\text{SCAN} : (\varepsilon, A_{k,0}[\alpha], \varepsilon) \rightarrow (e, \nabla_{k,i}[\alpha], \varepsilon)$

The reader may easily check that $A(G, \rightarrow, \leftarrow)$ recognizes $L(G)$. The choice of the call and return elements for the MS $(A_{k,i}, \nabla_{k,i})$ and the AS $(\nabla_{k,i})$ and $(\nabla_{k,i})$ defines a parsing strategy, by controlling how information flow between the phases of prediction and propagation. The following table lists the choices corresponding to the main parsing strategies (but others are definable).

<table>
<thead>
<tr>
<th>Strategy</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-Down</td>
<td>$\downarrow$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>Bottom-Up</td>
<td>$\downarrow$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
</tr>
<tr>
<td>Earley</td>
<td>$\downarrow$</td>
<td>$\uparrow$</td>
<td>$\downarrow$</td>
</tr>
</tbody>
</table>

It is also worth to note that the description of $A(G, \rightarrow, \leftarrow)$ could be simplified. Indeed, for every configuration $(m, e, \Xi, \eta)$ derivable with $A(G, \rightarrow, \leftarrow)$, we can show that $\Xi = \{ \nabla_{k_1}, \ldots, \nabla_{k_n} \}$ and that $\delta$ only depends on $\nabla_{k_i}$. That means that we could use a master stack without action marks, these marks being implicitly given by the elements $\nabla_{k_i}$.

### 3 Using 2SA to parse TAGs

Tree Adjoining Grammars are a extension of CFG introduced by Joshi in (Joshi, 1987) that use trees instead of productions as primary representing structure. Formally, a TAG is a 5-tuple $G = (V_X, V_T, S, I, A)$, where $V_X$ is a finite set of non-terminal symbols, $V_T$ a finite set of terminal symbols, $S$ the axiom of the grammar, $I$ a finite set of initial trees and $A$ a finite set of auxiliary trees. $IUA$ is the set of elementary trees. Internal nodes are labeled by non-terminals and leaf nodes by terminals or $\varepsilon$, except for exactly one leaf per auxiliary tree (the foot) which is labeled by the same non-terminal used as label of its root node.

New trees are derived by adjoining: let be $\alpha$ a tree containing a node $\nu$ labeled by $A$ and let be $\beta$ an auxiliary tree whose root and foot nodes are also labeled by $A$. Then, the adjoining of $\beta$ at the adjunction node $\nu$ is obtained by excising the subtree $\alpha_{\nu}$ of $\alpha$ with root $\nu$, attaching $\beta$ to $\nu$ and attaching the excised subtree to the foot of $\beta$ (See Fig. 2).

![Figure 2: Traversal of an adjunction](image-url)
• \(\nu_{k,0} \rightarrow \nu_{k,1} \ldots \nu_{k,n_k}\), where \(\nu_{k,0}\) denotes some non-leaf node \(k\) of \(\alpha\) and \(\nu_{k,i}\) the \(i^{th}\) son of \(k\).

• \(\nu_{k,0} \rightarrow a_k\), where \(\nu_{k,0}\) denotes some leaf node \(k\) of \(\alpha\) with terminal label \(a_k\).

As done for LGs, we introduce symbols \(\nabla_{k,i}\) to denote dotted productions and consider projections \(\rightarrow\) and \(\leftarrow\) to define the parameterized 2SA \(A(G, \rightarrow, \leftarrow) = (V_r, M, \nu_0, \theta)\) where \(M = \{\nabla_{k,i}\} \cup \{\nu_{k,i}\} \cup \{\nu_{k,i}\} \cup \{\nu_{k,i}\}\). The transitions are given by the following rules (and illustrated in Figure 2).

• Call / Return for a node not on a spine. The call starts a new session, exited at return.

\[
\text{CALL : } (m, \nu_{k,i+1}\epsilon) \rightarrow (w, \nu_{k,i+1}\epsilon, |w|) \\
\text{RET : } (e, \nu_{k,i+1}\epsilon) \rightarrow (m, \nu_{k,i+1}\epsilon)
\]

• Call / Return for a node \(\nu_{k,i+1}\) on a spine.

The adjunction stack is propagated un-modified along the spine.

\[
\text{SCALL : } (w, \nu_{k,i+1}\epsilon) \rightarrow (w, \nu_{k,i+1}\epsilon, |w|) \\
\text{SRET : } (e, \nu_{k,i+1}\epsilon) \rightarrow (e, \nu_{k,i+1}\epsilon)
\]

• Call / Return for an adjunction on node \(\nu_{k,0}\).

The computation is diverted to parse some acceptable auxiliary tree \(\beta\) (with root node \(r_\beta\)), and a continuation point is stored on the auxiliary stack.

\[
\text{ACALL : } (w, \nu_{k,0}\epsilon) \rightarrow (w, \nu_{k,0}\epsilon, \beta) \\
\text{ARET : } (e, \nu_{k,0}\epsilon) \rightarrow (e, \nu_{k,0}\epsilon)
\]

• Call / Return for a foot node \(f_\beta\). The continuation stored by the adjunction is used to parse the excised subtree.

\[
\text{FCALL : } (w, f_\beta, A) \rightarrow (w, f_\beta, A, \epsilon) \\
\text{FRET : } (e, f_\beta, A) \rightarrow (e, f_\beta, A)
\]

Note: These two transitions use a variable \(A\) over \(M\). This is a slight extension of 2SA that preserves correctness and complexity.

• Production Selection

\[
\text{SEL : } (w, \nu_{k,0}\epsilon) \rightarrow (w, \nu_{k,0}\epsilon)
\]

• Production Publishing

\[
\text{PUB : } (m, \nu_{k,0}\epsilon) \rightarrow (e, \nu_{k,0}\epsilon)
\]

• Scanning

\[
\text{SCAN : } (w, \nu_{k,0}\epsilon, |w|) \rightarrow (e, \nu_{k,0}\epsilon, |w|)
\]

Different parsing strategies can be obtained by choosing the call \((\nu_{k,i})\) and return \((\nu_{k,i})\) elements:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>(\nu)</th>
<th>(\bar{\nu})</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix-valid Top-Down</td>
<td>(\nu)</td>
<td>(\bar{\nu})</td>
</tr>
<tr>
<td>Bottom-Up</td>
<td>(\nu)</td>
<td>(\bar{\nu})</td>
</tr>
<tr>
<td>prefix-valid Earley</td>
<td>(\nu)</td>
<td>(\bar{\nu})</td>
</tr>
</tbody>
</table>

Non-prefix-valid variants of the top-down and Earley-like strategies can also be defined, by taking \(\beta = \bot\) and \(\beta = r_\beta\) for every root node \(r_\beta\) of an auxiliary tree \(\beta\) (the projections being unmodified on the other elements). In other words, we get a full prediction on the context-free backbone of \(G\) but no prediction on the adjunctions.

4 Items

We identify two kinds of elementary derivations, namely Context-Free (CF) and escaped Context-Free (xCF) derivations, respectively represented by CF and xCF items. An item keeps the pertinent information relative to a derivation, which allows to apply the sequence of transitions associated with the derivation in different contexts.

Before presenting these items, we introduce the following classification about derivations.

A derivation \((p, u, \Xi A, \xi) \rightarrow (q, v, \Theta, \theta)\) is said **rightward** if no element of \(\xi\) is crossed (even for consultation) during the derivation and if \(\Lambda\) is only consulted. Then \(\Xi A\) is a prefix of \(\Theta\).

Similarly, a derivation \((p, u, \Xi, \xi) \rightarrow (q, v, \Theta, \theta)\) is said **upward** if no element of \(\xi\) is crossed (even for consultation). Then \(\xi\) is a prefix of \(\theta\).

We also note \(w[p/p']\) the prefix substitution of \(p\) by \(q\) for all words \(w,p,q\) on some vocabulary such that \(p\) is prefix of \(w\).

4.1 Context-Free Derivations

A **Context-Free** (CF) derivation only depends on the topmost element \(A\) of the initial stack MS. That means that no element of the initial AS and no element of MS below element \(A\) is needed:

\[
(o, u, \Xi A, \xi) \rightarrow_{d_1} (w, v, \Theta B, \theta) \rightarrow_{d_2} (m, w, \Theta B \delta C, \zeta c)
\]

where

- \(d_1\) and \(d_2\) are both rightward and upward.
- \(d_2\) is rightward.
- either \((\delta \neq \epsilon, o = w, \text{ and } c \in \mathcal{X})\) or \((\delta = \epsilon, \text{ and } c = \epsilon)\).

For such a derivation, we have:

**Proposition 4.1** For all prefix stacks \(\Xi', \xi'\),

\[
(o, u, \Xi' A, \xi') \rightarrow_{d_1} (w, v, \Theta' B, \theta') \rightarrow_{d_2} (m, w, \Theta' B \delta C, \zeta c)
\]

where \(\Theta' = \Theta[\Xi'/\Xi]\) and \(\theta' = \theta[\xi'/\xi]\).

The proposition suggests representing the CF derivation by a CF item of the form

\[
AB \delta \tilde{C}m
\]

where \(A = (u, A)\) and \(B = (v, B)\) are micro configurations and \(\tilde{C} = (w, C, c)\) a mini configuration.
4.2 Escaped Context-Free Derivations

An escaped Context-Free [xCF] derivation is almost a CF derivation, except for an escape sub-derivation that accesses deep elements of AS.

\[
\begin{align*}
(w, u, [\Xi A, \xi]) & \vdash^{*}_{d_1} (w, v, [\Xi B, \theta]) \\
\vdash^{*}_{d_2} (w, s, \Phi D, \xi d) & \\
\vdash^{*}_{d_3} (e, t, \Phi D \setminus E, \phi) & \\
\vdash^{*}_{d_4} (e, w, \Theta B \delta C, \phi c)
\end{align*}
\]

where

- \( d_1 \) and \( d_2 \) are both rightward and upward.
- \( d_2 \) and \( dx \) are rightward.
- \( d_3 \) is upward.
- \( \delta \neq \Xi \) and \( d, c \in X' \).

**Proposition 4.2** For all prefix stacks \( \Xi' \) and \( \xi' \), stack \( \phi' \), and rightward derivation

\[
(w, s, \Phi D, \xi' d) \vdash^{*}_{dx} (e, t, \Phi D \setminus E, \phi')
\]

where \( \Phi' = \Phi[\Xi'/\Xi] \), we have

\[
\begin{align*}
(w, u, [\Xi' A, \xi']) & \vdash^{*}_{d_1} (w, v, [\Xi' B, \theta'][\Xi'/\xi]) \\
\vdash^{*}_{d_2} (w, s, \Phi[\Xi'/\Xi] D, \xi' d) & \\
\vdash^{*}_{d_3} (e, t, \Phi[\Xi'/\Xi] D \setminus E, \phi') & \\
\vdash^{*}_{d_4} (e, w, \Theta [\Xi'/\Xi] B \delta C, \phi' c)
\end{align*}
\]

The proposition suggests representing the xCF derivation by an xCF item of the form

\[
AB\delta[\bar{D} E]Ce
\]

where \( A = \langle u, A \rangle \), \( B = \langle v, B \rangle \), \( \bar{D} = \langle s, D, d \rangle \), \( E = \langle t, E \rangle \) and \( \bar{C} = \langle w, C, c \rangle \).

In order to homogenize notations, we also use the alternate notation \( AB\delta[\bar{D} E]Ce \) to represent CF item \( AB\delta Cm \), introducing a dummy symbol \( \epsilon \).

The specific forms taken by CF and xCF items for the different actions \( \delta \) are outlined in Figure 3.

5 Combining items and transitions

We provide the rules to combine items and transitions in order to retrieve all possible SA derivations.

These rules do not explicitly fold the scanning constraints and suppose that the string \( z \) may be read between positions \( w \) and \( k \) of the input string. They use holes * to denote slots that do not need be consulted. For any mini configuration \( \bar{A} = \langle u, A, a \rangle \), we note \( \bar{A} = \langle u, A \rangle \) its micro projection.

\[
[-\text{WRITE}] \quad \tau = (w, C, e) \xrightarrow{\epsilon} (w, C \Rightarrow F, \epsilon)
\]

\[
A**[\infty] \bar{C}w \Rightarrow A\bar{C}^\epsilon \Rightarrow [\infty] \bar{F}w
\]

where \( \bar{C} = \langle w, C, c \rangle \), and \( \bar{F} = \langle k, F, c \rangle \).

\[
[\Rightarrow-\text{WRITE}] \quad \tau = (w, C) \xrightarrow{\epsilon} (w, C \Rightarrow F, \epsilon)
\]

\[
A**[\infty] \bar{C}w \Rightarrow A\bar{C}^\epsilon \Rightarrow [\infty] \bar{F}w
\]

where \( \bar{C} = \langle w, C, c \rangle \), and \( \bar{F} = \langle k, F, c \rangle \).

\[
[=\text{WRITE}] \quad \tau = (w, C) \xrightarrow{\epsilon} (w, C \Rightarrow F, \epsilon)
\]

\[
A**[\infty] \bar{C}m \Rightarrow A\bar{C}^\epsilon \Rightarrow [\infty] \bar{F}w
\]

where \( \bar{C} = \langle w, C, c \rangle \), and \( \bar{F} = \langle k, F, c \rangle \).

\[
[\Rightarrow=\text{WRITE}] \quad \tau = (w, C, c) \xrightarrow{\epsilon} (w, C \Rightarrow= F, \epsilon)
\]

\[
\bar{A}^\epsilon \Rightarrow \bar{M} \Rightarrow [\infty] \bar{F}w
\]

where \( \bar{C} = \langle w, C, c \rangle \), \( \bar{B} = \langle v, B, b \rangle \), \( \bar{F} = \langle k, F, c \rangle \), and (when \( \bar{D} \neq \epsilon \)) \( \bar{D} = \langle s, D, b \rangle \).
\[
\tau = (e, B, C, \epsilon) \xrightarrow{\Delta} (e, F, f)
\]
\[
\begin{align*}
\tilde{A} \tilde{B} \triangleright \tilde{D} \tilde{E} \tilde{C} e \\
\tilde{A} \times \lambda (\infty) \tilde{M} \tilde{w} & \xrightarrow{\epsilon} \tilde{M} \times O \tilde{D} \tilde{E} \tilde{C} \tilde{F} e
\end{align*}
\]
(6)
where \( \tilde{C} = \langle w, C, \vert = m \rangle \), \( \tilde{B} = \langle v, B, b \rangle \), \( \tilde{M} = \langle l, M, m \rangle \), \( \tilde{F} = \langle k, F, f \rangle \), and (when \( \tilde{D} = \epsilon \)) \( \tilde{D} = \langle \ast, \ast, m \rangle \).

\[\models \text{ERASE} \quad \tau = (e, B, C, \vert = m) \xrightarrow{\epsilon} (m, F, e)\]
\[\tilde{B}^{\ast} \triangleright \tilde{D} \tilde{E} \times \tilde{C} e \quad \tilde{M} \lambda (\infty) \tilde{D} \tilde{B} \tilde{w} \xrightarrow{\epsilon} \tilde{M} \lambda \tilde{D} \tilde{E} \tilde{B} \tilde{m}
\]
(7)

\[ \tilde{C} = \langle w, C, \vert = m \rangle, \quad \tilde{B} = \langle v, B, b \rangle, \quad \tilde{F} = \langle k, F, f \rangle \]

\[\models \text{ERASE} \quad \tau = (e, B, C, \vert = m) \xrightarrow{\ast} (e, F, e)\]
\[\tilde{B} \tilde{B}^{\ast} \times \tilde{D} \tilde{E} \tilde{C} e \quad \tilde{M} \lambda (\infty) \tilde{D} \tilde{B} \tilde{w} \xrightarrow{\ast} \tilde{M} \lambda \tilde{D} \tilde{E} \tilde{B} \tilde{w}
\]
(8)

\[ \tilde{C} = \langle w, C, \vert = m \rangle, \quad \tilde{B} = \langle v, B, b \rangle, \quad \tilde{F} = \langle k, F, f \rangle \]

\[\models \text{SWAP} \quad \tau = (p, C, \xi) \xrightarrow{\epsilon} (q, F, \xi)\]
\[\tilde{A} \tilde{B} \tilde{D} \tilde{E} \tilde{C} \tilde{m} \xrightarrow{\ast} \tilde{A} \tilde{B} \tilde{C} \tilde{D} \tilde{E} \tilde{F} \tilde{m}
\]
(10)

where \( \tilde{C} = \langle w, C, \vert = m \rangle \), \( \tilde{F} = \langle k, F, c \rangle \), and either \( \xi = - \frac{\partial}{\partial} \text{ or } \epsilon = \frac{\partial}{\partial} \).

The best way to apprehend these rules is to visualize them graphically as done for the two most complex ones (Rules 6 and 9) in Figures 4 and 5.

\[\text{Figure 4: Application of Rule 6}\]

\[\text{Figure 5: Application of Rule 9}\]

5.1 Reducing the complexity
An analysis of the time complexity to apply each rule gives us polynomial complexities \( O(n^u) \) with \( u \leq 6 \) except for Rule 9 where \( u = 8 \). However, by adopting an idea from (Nederhof, 1997), we replace Rule 9 by the alternate and equivalent Rule 11.

\[\begin{align*}
\tilde{B}^{\ast} \times \tilde{D} \tilde{E} \times \tilde{C} e \\
\tilde{M} \lambda (\infty) \tilde{D} \tilde{B} \tilde{w} \xrightarrow{\epsilon} \tilde{M} \lambda \tilde{D} \tilde{E} \times \tilde{C} e
\end{align*}\]
(11)

where \( \tilde{C} = \langle w, C, \vert = m \rangle, \quad \tilde{B} = \langle v, B, b \rangle, \quad \tilde{F} = \langle k, F, f \rangle, \quad \text{and (when } \tilde{D} = \epsilon \text{) } \tilde{D} = \langle l, O, b \rangle.\]

Rule 11 has same complexity than Rule 9, but may actually be split into two rules of lesser complexity \( O(n^6) \), introducing an intermediary pseudo-item \( BB^{\ast} \tilde{D} \tilde{E} \times \tilde{C} \tilde{e} \) (intuitively assimilable to a “deeply escaped” \( \tilde{C} \) derivation).

Rule 12 collects these pseudo-items (independently from any transition) while Rule 13 combines them with items (given a \( \text{ERASE} \) transition \( \tau \)).

\[\begin{align*}
BB^{\ast} \times \tilde{D} \tilde{E} \times \tilde{C} e \\
\tilde{M} \lambda (\infty) \tilde{D} \tilde{B} \tilde{w} \xrightarrow{\ast} \tilde{M} \lambda \tilde{D} \tilde{E} \tilde{B} \tilde{w}
\end{align*}\]
(12)

\[\begin{align*}
BB^{\ast} \times \tilde{D} \tilde{E} \times \tilde{C} e \\
\tilde{M} \lambda (\infty) \tilde{D} \tilde{B} \tilde{w} \xrightarrow{\ast} \tilde{M} \lambda \tilde{D} \tilde{E} \tilde{B} \tilde{w}
\end{align*}\]
(13)

where \( \tilde{C} = \langle w, C, \vert = m \rangle, \quad \tilde{B} = \langle v, B, b \rangle, \quad \tilde{F} = \langle k, F, f \rangle, \quad \text{and (when } \tilde{D} = \epsilon \text{) } \tilde{D} = \langle l, O, b \rangle.\]

Theorem 5.1 The worst time complexity of the application rules \( 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13 \) is \( O(n^6) \) where \( n \) is the length of the input string. The worst space complexity is \( O(n^6) \).

5.2 Correctness results
Two main theorems establish the correctness of derivable items w.r.t. derivable configurations.

A derivable item is either the initial item or an item resulting from the application of a combination rules on derivable items. The initial item \( \langle 0, \epsilon \rangle \langle 0, \epsilon \rangle | (\infty) \langle 0, 0, \vert = w \rangle \) stands for the virtual derivation step \( (w, 0, \epsilon, \epsilon) \langle (w, 0, \vert = w, 0, \vert = w) \).

Theorem 5.2 (Soundness) For every derivable item \( T = AB \delta \tilde{D} \tilde{E} \tilde{C} \tilde{m} \), there exists a derivation on configurations

\[\langle 0, \epsilon \rangle \xrightarrow{\delta} \langle 0, \epsilon \rangle \]

such that \( U(T) \times V \) is a \( CF \) or \( zCF \) derivation representable by \( I \).

Proof: By induction on the item derivation length and case analysis.
Theorem 5.3 (Completeness) For all derivable configuration \((m, w, \Xi C, \xi c)\), there exists a derivable item \(AB\subseteq D_E \subseteq C m\) such that \(C = (w, C, c)\).

Proof: By induction on the configuration derivation length and by case analysis of the different application rules. We also need the following "Extraction Lemma". 

Proposition 5.1 From any derivation 
\[
\langle 0, c \rangle \xrightarrow{\epsilon} (m, w, \Xi C, \xi c)
\]
may be extracted a suffix CF or CF sub-derivation \(\mathcal{U} \subseteq (m, w, \Xi C, \xi c)\) for some configuration \(\mathcal{U}\).

5.3 Illustration
In the context of TAG parsing (Sect. 3), we can provide some intuition of the items that are built with \(A(G, \rightarrow^+, \rightarrow^*)\), using some characteristic points encountered during the traversal of an adjunction (Fig. 6).

<table>
<thead>
<tr>
<th>after CALL</th>
<th>before RET</th>
</tr>
</thead>
<tbody>
<tr>
<td>on ADJ</td>
<td>(A_1 A_1 \rightarrow^* F_i w)</td>
</tr>
<tr>
<td>on SPINE</td>
<td>(A_1 S_1 \rightarrow^* F_i w)</td>
</tr>
<tr>
<td>on FOOT</td>
<td>(A_1 F_1 \rightarrow^* A_3 w)</td>
</tr>
</tbody>
</table>

![Figure 6: Adjunction and Items](image)

6 Conclusion
This paper unifies different results about TAGs and LGs in an uniform setting and illustrates the advantages of a clear distinction between the use of an operational device and the evaluation of this device. The operational device (here SD-2SA) helps us to focus on the description of parsing strategies (for LGs and TAGs), while, independently, we design an efficient evaluation mechanism for this device (here tabular interpretation with complexity \(O(n^6)\)).

Besides illustrating a methodology, we believe our approach also opens new axes of research.

For instance, even if the tabular interpretation we have presented has (we believe) the best possible complexity, it is still possible (using techniques outside the scope of this paper, (Barthélemy and Villemonte de la Clergie, 1996)) to improve its efficiency by refining what information should be kept in each kind of items (hence increasing computation sharing and reducing the number of items).

To handle TAGs or LGs with attributes, we also plan to extend SD-2SA to deal with first-order terms (rather than just symbols) using unification to apply transitions and subsumption to check items.

References


