Parsing Mildly Context-Sensitive Languages with Thread Automata

Éric Villemonte de la Clergerie
INRIA, Rocquencourt, B.P. 105, 78153 Le Chesnay (France)
`Eric.De_La_Clergerie@inria.fr`

Abstract

We introduce simple but powerful automata, called *Thread Automata*, to describe a wide range of parsing strategies for Mildly Context-Sensitive languages. Thread Automata are completed by a Dynamic Programming interpretation ensuring that tabular parsing may be performed with polynomial worst-case complexity.

1 Introduction

Mildly Context-Sensitive [MCS] formalisms form an (informal) class of formalisms presenting interesting linguistic and algorithmic properties (Weir, 1988). For instance, they include Tree Adjoining Grammars [TAG] (Joshi, 1987), and local Multi-Component TAG [MC-TAG] (Weir, 1988) where parse trees may be derived by combining elementary parse trees. MCS also include simple Range Concatenation Grammars [sRCG] (Boullier, 2000) where constraints on ranges of the input string specify linguistic constituents. A common characteristic of all these grammars is their ability to handle interleaved and discontinuous constituents, which seems important in order to handle linguistic phenomena such as topicalization, deep extraction, clitic movements or scrambling.

MCS are also interesting from an algorithmic point of view because their associated languages may be parsed with polynomial worst-case complexities in both space and time. However, there is no unified framework to express and compare parsing algorithms for MCS. Furthermore, a parsing algorithm is often expressed for some underlying parsing strategy (generally a bottom-up strategy), making it difficult to adapt the algorithm for some other strategy.

We introduce, in Section 2, a new kind of automata, namely *thread automata* [TA], that may be used to describe a wide range of parsing strategies for many MCS formalisms, including top-down prefix-valid [pv] strategies. As suggested by their name, the underlying intuition of TA is that several threads may be followed during parsing, with only one thread active at any time. A thread may start subthreads, may terminate, and may be suspended to return control either to its parent or to one of its direct descendants. Because a thread may be suspended and resumed several times, we can recognize interleaved discontinuous constituents by assigning a thread to a constituent. We illustrate these properties by explaining how to build TA encoding top-down pv strategies for TAG (Section 3) and ordered sRCG (Section 4).

The interest of TA is not only descriptive but also algorithmic. Indeed, we present, in Section 5, a Dynamic Programming interpretation of TA that ensures polynomial worst-case complexities in time and space, w.r.t. the length n of the input string.

2 Thread Automata

Formally, a Thread Automaton A is a tuple $(\mathcal{N}, \Sigma, S, F, \kappa, \mathcal{K}, \delta, \mathcal{U}, \Theta)$ where

- Σ (resp. $\mathcal{N}$) denotes a finite set of terminal (resp. non-terminal) symbols, with initial and final non-terminals S and F being two distinguished non-terminals;
- The *triggering* function κ denotes a partial function from $\mathcal{N}$ to some finite set $\mathcal{K}$ and is used to capture the amount of information consulted in a thread to trigger the application of some kinds of transitions; [1]
- $\mathcal{U}$ is a finite set of labels used to identify

[1]This function is not essential but still useful to reduce complexity w.r.t. the grammar size, as illustrated for TAGs (Section 3).

threads. $\mathcal{U}$ is also used by the partial function δ to *drive* computations by specifying the threads (subthreads or parents) that may be created or resumed at some point. δ is defined from $\mathcal{N}$ to $\mathcal{U}^{\perp} = \mathcal{U} \cup \{\perp\}$ where $\perp \notin \mathcal{U}$;

- Θ is a finite set of transitions.

A *thread* is a pair $p{:}A$ where $p = u_1 \ldots u_n \in \mathcal{U}^*$ is a (possibly empty) *thread path*, and A some non-terminal symbol from $\mathcal{N}$. The empty path is denoted by ϵ. We will usually confuse a thread with its path, speaking of thread p.

A *thread store* $\mathcal{S}$ is a finite set of threads, representing a function from $\mathrm{dom}(\mathcal{S})$ to $\mathcal{N}$ where $\mathrm{dom}(\mathcal{S}) \subset \mathcal{U}^*$ is closed by prefix (i.e., $pu \in \mathrm{dom}(\mathcal{S}) \Rightarrow p \in \mathrm{dom}(\mathcal{S})$).

A TA configuration is a tuple $\langle l, p, \mathcal{S} \rangle$ where l denotes the current position in the input string, p the active thread, and $\mathcal{S}$ a thread store with $p \in \mathrm{dom}(\mathcal{S})$. The initial configuration is $c_{\mathrm{init}} = \langle 0, \epsilon, \{\epsilon{:}S\} \rangle$ and the final one $c_{\mathrm{final}} = \langle n, u, \{\epsilon{:}S, u{:}F\} \rangle$ where $u = \delta(S)$ and n denotes the length of the input string.

A derivation step $c \underset{\tau}{\vdash} c'$ is performed using a transition $\tau \in \Theta$ of the following kind, where bracketed (resp. non-bracketed) parts denote contents of non active (resp. active) threads and $\kappa\delta(A)$ being a shortcut for $(\kappa(A), \delta(A))$:

SWAP $B \overset{\alpha}{\longmapsto} C$: Changes the contents of the active thread, possibly scanning a terminal on the input string.

$$\langle l, p, \mathcal{S} \cup p{:}B \rangle \underset{\tau}{\vdash} \langle l + |\alpha|, p, \mathcal{S} \cup p{:}C \rangle$$

where $a_l = \alpha$ if $\alpha \neq \epsilon$

PUSH $b \longmapsto [b]C$: Creates a new subthread, suspending its parent.

$$\langle l, p, \mathcal{S} \cup p{:}B \rangle \underset{\tau}{\vdash} \langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle$$

where $\kappa\delta(B) = (b, u)$ and $pu \notin \mathrm{dom}(\mathcal{S})$

POP $[B]C \longmapsto D$: Ends the active thread, returning control to its parent.

$$\langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \underset{\tau}{\vdash} \langle l, p, \mathcal{S} \cup p{:}C \rangle$$

where $\delta(C) = \perp$ and $pu \notin \mathrm{dom}(\mathcal{S})$

SPUSH $b[C] \longmapsto [b]D$: Resumes a suspended subthread of the active thread.

$$\langle l, p, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \underset{\tau}{\vdash} \langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}D \rangle$$

where $\kappa\delta(B) = (b, u)$

SPOP $[B]c \longmapsto D[c]$: Resumes the parent thread of the active thread.

$$\langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C \rangle \underset{\tau}{\vdash} \langle l, p, \mathcal{S} \cup p{:}D \cup pu{:}C \rangle$$

where $\kappa\delta(C) = (c, \perp)$

One may prove that these definitions imply $\delta(B) = u$ for POP and SPOP transitions, and $\delta(C) = \perp$ for SPUSH transitions.

The reflexive transitive closure of $\vdash$ is noted $\overset{\star}{\vdash}$ and the transitive closure $\overset{\pm}{\vdash}$.

Without restriction, a thread may be suspended infinitely many often, to return to its parent ($\perp$-suspension) or to a subthread v (v-suspension). We consider TA subclasses, namely h-TA, where a thread can be $\perp$-suspended at most h times. A sufficient condition to ensure that A is an h-TA is to exhibit some mapping $\lambda : \mathcal{N} \mapsto \{0, \ldots, h\}$ such that the following equations hold:

$$\begin{aligned}
\forall b[C] \longmapsto [b]D \in \Theta, &\quad \lambda(C) < \lambda(D) \\
\forall [B]c \longmapsto D[c] \in \Theta, &\quad \lambda(B) \leq \lambda(D) \\
\forall [B]c \longmapsto D \in \Theta, &\quad \lambda(B) \leq \lambda(D) \\
\forall B \overset{\alpha}{\longmapsto} C \in \Theta, &\quad \lambda(B) \leq \lambda(C)
\end{aligned}$$

0-TA are actually equivalent to Push-Down Automata, because SPUSH and SPOP transitions are no longer used and because we can assume $|\mathcal{U}| = 1$, without any loss of generality.

Figure 1 sketches how a Thread Automaton recognize $a^2b^2c^2$ using threads 1 and 1.1 to recognize two crossing constituents abc (and thread 1.1.1 for the empty constituent $\epsilon\epsilon\epsilon$).

3 TAGs and TA

A Tree Adjoining Grammar $G = (\mathcal{N}, \Sigma, \mathcal{I}, \mathcal{A}, S)$ (Joshi, 1987) is essentially characterized by elementary trees in $\mathcal{T} = \mathcal{I} \cup \mathcal{A}$ which correspond to partial parse trees. The label $l(\nu)$ of a node ν is a non-terminal in $\mathcal{N}$ or a terminal in Σ (only on leaf nodes). New trees may be derived by *substituting* some *initial tree* $\alpha \in \mathcal{I}$ at a leaf node ν, or

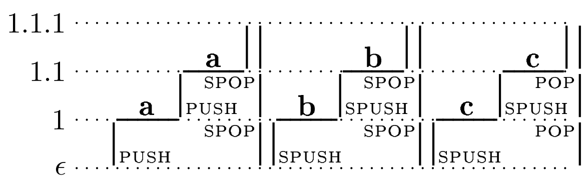

Figure 1: A derivation for $a^2b^2c^2$

by *adjoining* some *auxiliary tree* $\beta \in \mathcal{A}$ at node ν. Figure 2 shows an example of adjoining where β is inserted at node ν with the subtree rooted at ν inserted on the distinguished *foot node* f of β. To simplify this presentation and without generality loss, we assume that all nodes are either marked as non-adjoinable [NA] or as obligatory adjoinable [OA], leaf nodes being marked as NA. We suppose the reader familiar with the details about TAGs.

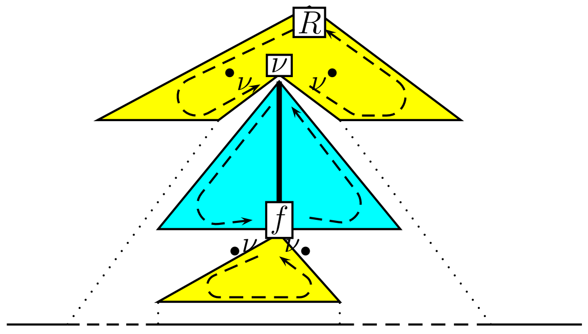

Figure 2: Tree traversals

As suggested by Figure 2, parsing TAG may be seen as traversing elementary trees (from left to right), using standard *dotted nodes* $^\bullet\nu$, $_\bullet\nu$, $\nu_\bullet$, and $\nu^\bullet$ to specify which part of a tree has already been recognized. Using TA, we associate a thread to each tree traversal. When a substitution or an adjunction starts, a new thread is started (with PUSH transitions **[SCALL]** or **[ACALL]**). When a substitution or an adjunction is completed, the associated thread is removed (with POP transitions **[SRET]** or **[ARET]**). More interestingly, the thread attached to an auxiliary tree is $\perp$-suspended to return to the adjunction node when reaching the foot node (with SPOP transition **[FCALL]**) and resumed when coming back from the adjunction node to the foot node (with SPUSH transition **[FRET]**). Intuitively, a thread may only be $\perp$-suspended once (to handle a foot node) and at most d direct subthreads of a thread may be alive at any time, where d denotes the maximal depth of elementary trees.

More formally, we build the Thread Automaton $A_G = (\mathcal{N}_G, \Sigma, S, \mathrm{ret}, \kappa, \mathcal{K}, \delta, \mathcal{U}, \Theta)$:

- $\mathcal{N}_G = \{N^a, N^s | N \in \mathcal{N}\} \cup \mathcal{D}_G$ where $\mathcal{D}_G$ denotes the minimal set of all dotted nodes for G modulo the equivalence relation identifying dotted nodes that actually correspond to a same computation point.[2]
- $\mathcal{K} = \{N^a, N^s | N \in \mathcal{N}\} \cup \{\mathrm{adj}, \mathrm{foot}\}$ and $\mathcal{U} = \{0, \ldots, d\}$ where $d = \max_{t \in \mathcal{T}} \mathrm{depth}(t)$
- κ and δ are defined as follows, where $N = l(\nu)$ and $u = \mathrm{depth}(\nu)$:

Adj	$\kappa\delta(^\bullet\nu) = (N^a, u)$ $\kappa\delta(\nu_\bullet) = (\mathrm{adj}, u)$
Subst	$\kappa\delta(_\bullet\nu) = (N^s, u)$
foot	$\kappa\delta(_\bullet\mathrm{f}) = (\mathrm{foot}, \perp)$

- Θ includes the following transitions,[3] to handle
  - a terminal node ν with $l(\nu) = a$

(SCAN)	$_\bullet\nu \xmapsto{a} \nu_\bullet$

  - a substitution node ν and an initial tree α with $l(\nu) = l(r_\alpha) = N$

(SCALL)	$N^s \longmapsto [N^s]N^s$
(SSEL)	$N^s \longmapsto {}^\bullet\mathrm{r}_\alpha$
(SPUB)	$\mathrm{r}_\alpha{}^\bullet \longmapsto \mathrm{ret}$
(SRET)	$[_\bullet\nu]\mathrm{ret} \longmapsto \nu_\bullet$

  - an adjunction node ν and an auxiliary tree β with $l(\nu) = l(r_\beta) = N$

(ACALL)	$N^a \longmapsto [N^a]N^a$
(ASEL)	$N^a \longmapsto {}^\bullet\mathrm{r}_\beta$
(APUB)	$\mathrm{r}_\beta{}^\bullet \longmapsto \mathrm{ret}$
(ARET)	$[\nu_\bullet]\mathrm{ret} \longmapsto \nu^\bullet$

[2]for instance, $\nu^\bullet \equiv {}^\bullet\mu$ if μ is the rightmost brother of ν and $_\bullet\nu \equiv {}^\bullet\mu$ if μ is the leftmost son of ν.

[3]These transitions also illustrate the role of the trigger function κ. For instance, without κ, [FCALL] becomes $[^\bullet\nu]_\bullet\mathrm{f} \longmapsto {}_\bullet\nu[_\bullet\mathrm{f}]$ referring to nodes from two different trees and leading to complexity in $O(|G|^2)$, instead of $O(|G|)$ otherwise. We express with κ that the triggering info for [FCALL] is to reach a foot node, its name being not pertinent.

– a foot node f and an adjunction node ν

$$\begin{array}{ll}\text{(FCALL)} & [{}^\bullet\nu]\text{foot} \longmapsto {}_\bullet\nu[\text{foot}] \\ \text{(FRET)} & \text{adj}[{}_\bullet\text{f}] \longmapsto [\text{adj}]\,\text{f}_\bullet\end{array}$$

The automaton A_G encodes a top-down pv parsing strategy for G. It also belongs to the 1-TA subclass (at most one $\perp$-suspension), because of the mapping λ defined by $\lambda({}^\bullet\nu) = \lambda(\nu^\bullet) = \lambda(\nu_\bullet) = \lambda(\nu^\bullet) = 0$ if ν is strictly on the left of a spine (or in an initial tree); $\lambda({}^\bullet\nu) = \lambda(\nu^\bullet) = \lambda(\nu_\bullet) = \lambda(\nu^\bullet) = 1$ if ν is strictly on the right of a spine; $\lambda({}^\bullet\nu) = \lambda(\nu^\bullet) = 0$ and $\lambda(\nu_\bullet) = \lambda(\nu^\bullet) = 1$ if ν is on a spine; $\lambda(N^a) = \lambda(N^s) = 0$ if N is a non-terminal; and $\lambda(\text{ret}) = 1$.

By assigning a thread to a set of elementary trees and slightly extending the definition of δ, one can describe parsing strategies for tree-local or set-local Multi-Component TAG (Villemonte de la Clergerie, 2002).

4 Ordered simple RCG

Figure 3 illustrates the *ordered simple RCG* [osRCG] clause $\gamma : A(X_1X_2X_3X_4, X_5X_6) \longrightarrow B(X_1, X_3, X_5)C(X_2, X_4, X_6)$ where range variables X_i are instantiated by ranges of the input string. A discontinuous constituent A is built from two discontinuous constituents B and C that interleave their ranges. The two parts of A are separated by a *hole* range H, inherited from some ancestor or sibling of A. OsRCG may typically be used to describe such phenomena of complex constituent interleaving. Actually, osRCG, like simple RCG, are equivalent to *Linear Context-Free Rewriting systems* (Weir, 1992), a very general class of MCS.

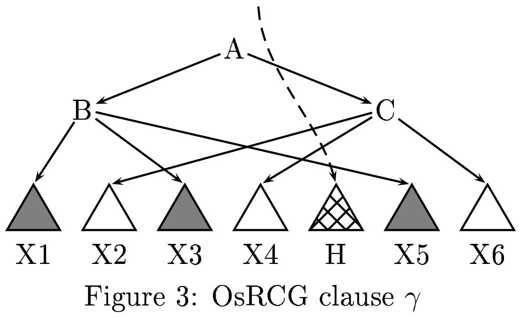

Figure 3: OsRCG clause γ

More formally, a osRCG $G = (\mathcal{N}, \Sigma, \mathcal{V}, \mathcal{C}, S)$ is a RCG (Boullier, 2000) such that each clause $\gamma \in \mathcal{C}$ is of the form $A_0 \longrightarrow A_1, \ldots, A_m$ with :

1. The head literal $A_0 = N_0(\overrightarrow{\alpha_1}, \ldots, \overrightarrow{\alpha_m})$ where N_0 is a non-terminal in $\mathcal{N}$, $\overrightarrow{\alpha_k} = \alpha_{k.1} \ldots \alpha_{k.f_k}$, and $\alpha_{k.i}$ is a terminal in Σ or a range variable in $\mathcal{V}$. We note ranges$(\gamma) = \{k.i | k = 1, \ldots, m \wedge i = 1, \ldots, f_i\}$, lexically ordered.
2. The body literal $A_j = N_j(Y_{j.1}, \ldots, Y_{j.m_j})$ where N_j is a non-terminal and $Y_{j.l}$ is a range variable.
3. Each range variable X of γ has exactly two occurrences, one in the head literal ($X = \alpha_{k.i}$) and one in some body literal ($X = Y_{j.l}$). We note $k.i \rhd_\gamma j.l$.
4. If a range variable X precedes a range variable Y (as argument) in some body literal, then X also precedes Y (as argument) in the head literal (*total range ordering*), i.e.
$$r \rhd_\gamma j.l \wedge r' \rhd_\gamma j.l + 1 \Rightarrow r < r'$$

We assume that each non-terminal $N \in \mathcal{N}$ has a fixed arity $a(N)$. An instantiation σ from range variables X to ranges $\sigma(X) = [l, r[$ of the input string must satisfy (a) a concatenation constraint $\alpha = \alpha_1\alpha_2$ with $\sigma(\alpha) = [l, s]$ if $\sigma(\alpha_1) = [l, r[$ and $\sigma(\alpha_2) = [r, s[$; and (b) a literal $A = N(\alpha_1, \ldots, \alpha_{a(N)})$ with $\sigma(A) = (\sigma(\alpha_1), \ldots, \sigma(\alpha_{a(N)}))$ if its arguments α_i are all satisfied. A literal A is derivable for σ if there exists a clause $\gamma : A_0 \longrightarrow A_1, \ldots, A_m$ such that $\sigma(A) = \sigma(A_0)$, σ satisfies A_i $i \geq 0$, and all A_i are derivable.

We define the number $\sharp_\gamma j$ of uncompleted constituents at A_j, i.e. started before A_j but not yet finished, by $|\{j' | \exists r_1 < r_2 < r_3,\ r_1 \rhd j'.1 \wedge r_2 \rhd j.1 \wedge r_3 \rhd j'.m_{j'}\}|$.

Because the ranges occurring in a clause are ordered, it is relatively easy to describe a TA encoding a top-down left-to-right pv parsing strategy by traversing each range $k.i$ in order, executing the action attached at $k.i$: create or resume a thread for some sub-constituent A_j, or resume the parent thread at holes. We build the Tread Automaton $A_G = (\mathcal{N}_G, \Sigma, S, \text{ret}, \kappa, \mathcal{K}, \delta, \mathcal{U}, \Theta)$ such that:

- $\mathcal{N}_G = \mathcal{N} \cup \{\text{ret}\} \cup \{\gamma_{k.0}, \gamma_{k.i} | \gamma \in \mathcal{C}, k.i \in \text{ranges}(\gamma)\}$, where $\gamma_{k.i}$ are new symbols, denoting computation points and similar to dotted rules.

- $\mathcal{K} = \mathcal{N} \cup \{\text{void}\}$ and $\mathcal{U} = \{0, \ldots, d\}$, where $d = \max_{\gamma,j} \sharp_\gamma j$ denotes the maximal number of uncompleted body literals at some point in a clause.
- $\kappa\delta(\gamma_{k.i}) = (N_j, \sharp_\gamma j)$ if $k.i + 1 \rhd_\gamma j.1$; $\kappa\delta(\gamma_{k.i}) = (\text{void}, \sharp_\gamma j)$ if $k.i + 1 \rhd_\gamma j.l$ and $l > 1$; $\kappa\delta(\gamma_{k.i}) = (void, \perp)$ if $k.i$ is a $\perp$-*point* (i.e., $k.j \in \text{ranges}(\gamma) \Rightarrow j \leq i$).

For each clause γ, Θ includes the following transitions:

(SEL) γ: $N_0 \longmapsto \gamma_{1.0}$

(PUB) $\gamma_{k.i} \longmapsto \text{ret}$ if $k.i$ is the last range of γ.

(SCAN) $\gamma_{k.i} \overset{\alpha_{k.i+1}}{\longmapsto} \gamma_{k.i+1}$ if $\alpha_{k.i+1}$ is a terminal.

(CALL) $N_j \longmapsto [N_j]N_j$ for all non-terminals N_j, started at $\gamma_{k.i}$, i.e. when $k.i + 1 \rhd_\gamma j.1$.

(RET) $[\gamma_{k.i}]\text{ret} \longmapsto \gamma_{k.i+1}$ if $\alpha_{k.i+1}$ is the last argument of some body literal in γ.

(HOLE) $\text{void}[\gamma_{k.i}] \longmapsto [\text{void}]\gamma_{k+1.0}$ if $k.i$ is a $\perp$-point, but not the last one.

(COMP) $[\gamma_{k.i}]\text{void} \longmapsto \gamma_{k.i+1}[\text{void}]$ if $\alpha_{k.i+1}$ is a range variable, not occurring as the last argument of a body literal.

A_G is an h-TA with $h = a - 1$ with $a = \max_{N \in \mathcal{N}} a(N)$ denoting the maximal arity of a non terminal.

5 Dynamic Programming interpretation

Directly applying transitions upon configurations would lead to exponential time complexity and even looping in many cases. Instead, we design a Dynamic Programming [DP] interpretation of TA that allows us to build tabular parsers running in polynomial complexity. We first identify a class of derivations, called *escaped Context-Free derivations*, that may be represented in a compact way using *items*. Then we show how these items may be combined together and with transitions to retrieve all possible derivations.

5.1 Escaped Context-Free derivations

An escaped Context-Free [xCF] derivation D resumes all information relative to the active thread π of a given configuration, namely its starting point, its current ending point, and when it was suspended to return to its parent thread ($\perp$-suspension) or to some uncompleted subthread πv (v-suspension). For instance, Figure 4 shows an xCF derivation with $\pi = pu$ created at S, with a $\perp$-suspension between C and D, two v-suspensions (AB and GH), and a w-suspension (EF).

Formally, D is characterized by $(d_i)_{i=0\ldots2m+1}$ where d_i are sequences of transitions leading to configurations $c_i = \langle l_i, p_i, \mathcal{S}_i \rangle$, such that:

$$c_{\text{init}} \overset{\star}{\underset{d_0}{\vdash}} c_0 \overset{+}{\underset{d_1}{\vdash}} c_1 \ldots c_{2m} \overset{+}{\underset{d_{2m+1}}{\vdash}} c_{2m+1}$$

and

1. A new subthread $\pi = p_0 u$ of the active thread p_0 is created at c_0 ($\pi \notin \text{dom}(\mathcal{S}_0)$), active at c_{2m+1} ($p_{2m+1} = \pi$), and alive between c_0 and c_{2m+1} ($c_0 \overset{+}{\vdash} \langle l, p, \mathcal{S} \rangle \overset{\star}{\vdash} c_{2m+1} \Rightarrow \pi \in \text{dom}(\mathcal{S})$).
2. Each derivation $c_{2i} \overset{+}{\vdash} \langle l, p, \mathcal{S} \rangle \overset{\star}{\vdash} c_{2i+1}$, $i = 0 \ldots m$, is performed on thread $p = \pi$ or on subthreads $p = \pi q$ no longer alive at c_{m+1} ($p \notin \text{dom}(\mathcal{S}_{2m+1})$).
3. Each derivation $c_{2i-1} \overset{+}{\vdash} \langle l, p, \mathcal{S} \rangle \overset{\star}{\vdash} c_{2i}$, $i = 1 \ldots m$, suspends $p_{2i-1} = \pi$ and gives control to either ancestors ($\perp$-suspend) or descendants alive at c_{2m+1} (v-suspend).

   ($\perp$-susp) $\qquad \pi$ not prefix of p

   (v-susp) $\qquad p = \pi v q \wedge \pi v \in \text{dom}(\mathcal{S}_{2m+1})$

5.2 Items

An item is a trace of a xCF derivation where we remove as much as possible useless information in order to increase computation sharing. In particular, we do not keep full configurations $c = \langle l, p, \mathcal{S} \cup p{:}A \rangle$ but only *micro-configuration* $\overline{c} = \langle l, A \rangle$ or κ*-micro-configuration* $\overline{c}^\kappa = \langle l, a \rangle^\kappa$ where $a = \kappa(A)$. We will often simplify $\langle l, a \rangle^\kappa$ into $\langle l, a \rangle$.

Given an xCF derivation D, its projection $\overline{D}$ is defined by an *item* of the form

$$\overline{c_0}^\kappa / \mathcal{C} / \overline{c_{2m+1}}$$

where $\mathcal{C} = v_1 : S_1 \ldots v_i : S_i \ldots v_m : S_m$, $S_i = \overline{c_{2i-1}}^\kappa \overline{c_{2i}}^\kappa$, and $v_i = v \in \mathcal{U}^\perp$ if $c_{2i-1} \overset{\star}{\underset{d_{2i}}{\vdash}} c_{2i}$ is a v-suspension.

For instance, the item $s/v : ab, \perp : cd, w : ef, v : gh/I$ is associated to the xCF derivation of Fig. 4.

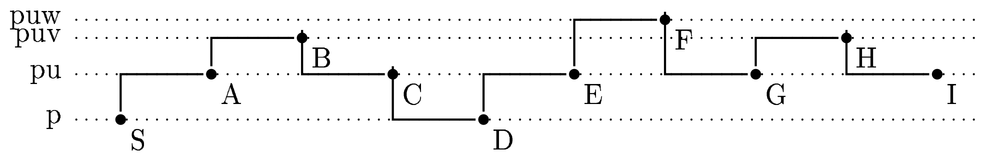

Figure 4: From a xCF derivation to item $s/v : ab, \perp : cd, w : ef, v : gh/I$

5.3 Application rules

Combining transitions and items to build new items will generally involve two items I and J, I being related to a parent thread and J to some sub-thread u. Furthermore, either I prepares an extension of its son J by filling its $\perp$-suspensions (*down extension*, Figure 5) or J prepares an extension of its parent I by filling its u-suspensions (*up extension*, Figure 6). To define formally these extensions, we need some auxiliary definitions.

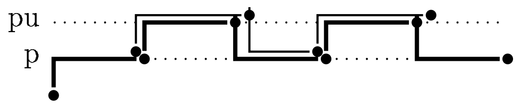

Figure 5: down extension $I \searrow_{\mathrm{u}} J$

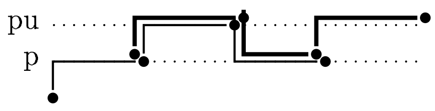

Figure 6: up extension $J \nearrow^{\mathrm{u}} I$

Given an item $I = \langle l, a \rangle \, / \overrightarrow{u_i{:}S_i} / \, \langle r, B \rangle$, we define $\mathrm{I}^\bullet = \langle r, B \rangle$, $\delta(I) = \delta(B)$, and its **index set** $\mathrm{ind}(I) = \{u \in \mathcal{U}^\perp | u = \delta(I) \wedge \exists i, u = u_i\}$. For $u \in \mathrm{ind}(I)$, we define its u-**complement** $I_{/u} = u_{i_1}{:}S_{i_1} \dots u_{i_k}{:}S_{i_k}$ where $\{i_1 < \dots < i_k\} = \{i | u_i \neq u\}$ and $i_j < i_{j+1}$; it u-**subsegment** $I_{|u} = S_{i_1} \dots S_{i_k}$ where $\{i_1 < \dots < i_k\} = \{i | u_i = u\}$ and $i_j < i_{j+1}$; and its u-**line** $I_{\to u}$ by

$$\begin{array}{ll} I_{\to u} = I_{|u} \cdot \langle r, \kappa B \rangle & u \in \mathcal{U} \\ I_{\to \perp} = \langle l, a \rangle \, . I_{|\perp} \cdot \langle r, \kappa B \rangle & \text{otherwise} \end{array}$$

We can now define the **up and down extension** predicates when $\delta(I) = u$ and $\delta(J) = \perp$:

$$J \nearrow^{\mathrm{u}} I \Longleftrightarrow_{\mathrm{def}} J_{\to \perp} = I_{\to u} \, \mathrm{J}^\bullet$$
$$I \searrow_{\mathrm{u}} J \Longleftrightarrow_{\mathrm{def}} I_{\to u} = J_{\to \perp} \, \mathrm{I}^\bullet$$

Using these predicates, we easily specify how to combine transitions and items, as shown in Figure 7. The case of a SPOP transition is illustrated in Figure 8: combining $I = s/u : ab/C$, $J = a/\perp : bc/d$, and $\tau : [C]d \longmapsto E[d]$ returns the item $s/u : ab, u : cd/E$ that extends I.

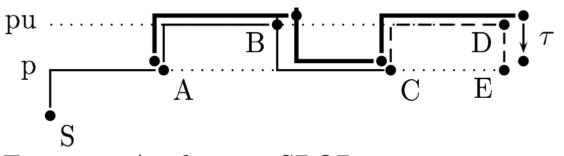

Figure 8: Applying a SPOP transition τ

5.4 Some results

The DP interpretation is sound (for each derivable item I, there exists a derivable configuration c such that $\overline{c} = \mathrm{I}^\bullet$)[4] and complete (for each derivable configuration c, there exists a derivable item I such that $\overline{c} = \mathrm{I}^\bullet$).

The termination of the DP interpretation is ensured for h-TA, with naive worst-case complexities $O(n^{2+2s})$ for space and $O(n^{3+2s})$ for time, where s, bounded by $h + dh$, denotes the maximal number of suspensions occurring in items. Indeed, the number of $\perp$-suspensions and of subthreads being bounded, the number of distinct items is also bounded. Space complexity comes from the number of possible distinct positions occurring in items. Time complexity is given by the number of distinct positions that are consulted when applying the application rules.

For $h = 0$, we retrieve the worst-case complexities for CFG, namely $O(n^2)$ in space and $O(n^3)$ in time. For TAG ($h = 1$), we get $O(n^{4+2d})$ in space and $O(n^{5+2d})$ in time, where d denotes the maximal depth of elementary trees. These complexities are not optimal but correspond to those mentioned in (Éric Villemonte de la Clergerie, 2001) for a very similar parsing algorithm for TAG, based on 2-Stack Automata [2SA], and which seems to be efficient in practice for linguistic grammars.

[4]assigning the initial item $\langle 0, \perp \rangle \, // \, \langle 0, S \rangle$ to c_{init}.

$$\text{(SWAP)}\qquad \frac{B \overset{\alpha}{\longmapsto} C \quad \langle l,a\rangle\,/\mathcal{C}/\,\langle r,B\rangle}{\langle l,a\rangle\,/\mathcal{C}/\,\langle r+|\alpha|,C\rangle} \qquad a_r = \alpha \text{ if } \alpha \neq \epsilon$$

$$\text{(PUSH)}\qquad \frac{b \longmapsto [b]C \quad I}{\langle r,b\rangle\,//\,\langle r,C\rangle} \qquad \begin{cases} \mathrm{I}^\bullet = \langle r,B\rangle \wedge \kappa\delta(B) = (b,u) \\ u \notin \mathrm{ind}(I) \end{cases}$$

$$\text{(POP)}\qquad \frac{[B]C \longmapsto D \quad \begin{matrix} J \\ \langle l,a\rangle\,/\mathcal{C}/\,\langle r,B\rangle^I \end{matrix}}{\langle l,a\rangle\,/I_{/u}/\,\langle s,D\rangle} \qquad \begin{cases} J \nearrow^{\mathrm{u}} I \\ \kappa\delta(B) = (b,u) \\ \mathrm{J}^\bullet = \langle s,C\rangle \wedge \mathrm{ind}(J) \subset \{\bot\} \end{cases}$$

$$\text{(SPUSH)}\qquad \frac{b[C] \longmapsto [b]D \quad \begin{matrix} \langle l,a\rangle\,/\mathcal{C}/\,\langle s,C\rangle^J \\ I \end{matrix}}{\langle l,a\rangle\,/\mathcal{C},\bot : \langle s,c\rangle\,\langle r,b\rangle\,/\,\langle r,D\rangle} \qquad \begin{cases} I \searrow_{\mathrm{u}} J \\ \mathrm{I}^\bullet = \langle r,B\rangle \wedge \kappa\delta(B) = (b,u) \\ \kappa\delta(C) = (c,\bot) \end{cases}$$

$$\text{(SPOP)}\qquad \frac{[B]c \longmapsto D[c] \quad \begin{matrix} J \\ \langle l,a\rangle\,/\mathcal{C}/\,\langle r,B\rangle^I \end{matrix}}{\langle l,a\rangle\,/\mathcal{C},u : \langle r,b\rangle\,\langle s,c\rangle\,/\,\langle s,D\rangle} \qquad \begin{cases} J \nearrow^{\mathrm{u}} I \\ \kappa\delta(B) = (b,u) \\ \mathrm{J}^\bullet = \langle s,C\rangle \wedge \kappa\delta(C) = (c,\bot) \end{cases}$$

Figure 7: Application rules

These worst-case complexities are upper estimations. A finer analysis gives, in many cases, complexities as low as $O(n^{2+s})$ for space and $O(n^{3+s})$ for time (Villemonte de la Clergerie, 2002).

6 Conclusion

The Thread Automata presented in this paper generalize and simplify various kinds of automata that have been previously proposed to parse MCS languages such as 2SA for TAGs. They may be used for a wide range of languages and parsing strategies. Their DP interpretation provides a uniform method to build tabular parsers for these languages, running in polynomial time, independently of the underlying parsing strategies.

We should implement the DP interpretation for TA in a near future, extending TA to handle logic arguments.

We would also like (a) to compare the formal power of TA w.r.t. other automata formalisms such as Tree-Walking Transducers (Weir, 1992) and (b) to investigate the descriptive power of TA for other linguistic formalisms. For instance, it does not seem possible to use TA in their current form to recognize the **MIX** language of strings having an equal number of each letter of some alphabet. The MIX language is an extreme case of scrambling but is conjectured not to be a MCS language.

References

Pierre Boullier. 2000. Range concatenation grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT2000)*, pages 53–64, Trento, Italy, February.

Aravind K. Joshi. 1987. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins Publishing Co., Amsterdam/Philadelphia.

Éric Villemonte de la Clergerie. 2001. Refining tabular parsers for TAGs. In *Proceedings of NAACL'01*, June.

Éric Villemonte de la Clergerie. 2002. Parsing MCS languages with thread automata. In *Proc. of TAG+6*, May.

David Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* Ph.D. thesis, University of Pennsylvania.

David Weir. 1992. Linear context-free rewriting systems and deterministic tree-walking transducers. In *Proc. of ACL'92*.