Parsing MCS Languages with Thread Automata

Éric Villemonte de la Clergerie
INRIA

1. Introduction

Generalizing ideas presented for 2-stack automata in (Éric Villemonte de la Clergerie, 2001), we introduce *Thread Automata* [TA], a new automata formalism that may be used to describe a wide range of parsing strategies (in particular top-down prefix-valid [pv] strategies) for many Mildly-Context Sensitive [MCS] grammatical formalisms (Weir, 1988), including CFG, TAG (Joshi, 1987), Multi-Component TAG (Weir, 1988), and Linear Context-Free Rewriting Systems [LCFRS] (Weir, 1992).

As suggested by their name, the underlying idea of TA is that several lines of computation (*threads*) are followed during parsing, only one being active at any time. Threads may start sub-threads, may terminate, and may be suspended to give control to their parent or one of their direct descendants.

Intuitively, a thread may be used to recognize a constituent while new sub-threads are started to recognize its sub-constituents. Because a thread may be suspended and resumed several times, we can recognize *discontinuous* constituents with *holes* such as auxiliary trees in TAG. More generally, TA may handle complex interleaving of discontinuous constituents as shown by Fig. 4(a) for the constituents B and C. TA may also be used to parse a sub-class of Range Concatenation Grammars [RCG] (Boullier, 2000b), which covers LCFRS.

Though TA exhibit strong expressive power, they still ensure good operational and complexity properties. Indeed, we propose a simple *Dynamic Programming* [DP] interpretation for TA that ensures tabular parsing in polynomial worst-case complexity for space and time w.r.t. the length of the input string.

If we focus in this paper on top-down pv parsing strategies, it is not because we believe them to be the most important ones, but rather because we think it is important to cover the full spectrum of parsing strategies. Moreover, a tabular parser which handles pv parsing strategies may usually be easily adapted to handle other kinds of strategies, the converse being not true. For instance, there already exists a systematic non pv parsing algorithm for RCG, but we are unaware of any systematic pv parsing algorithm for them.

2. Thread Automata

Formally, a Thread Automaton 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 two distinguished initial and final non-terminals S and F;
- Θ is a finite set of transitions;
- κ denotes a partial function from $\mathcal{N}$ to some other 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 threads. It is also used by the partial function δ to *drive* computations by specifying which threads (subthreads or parent) may potentially be created or resumed at some point, δ being defined from $\mathcal{N}$ to 2^{Δ} where $\Delta = \{\bot\} \cup \mathcal{U} \cup \{u^s | u \in \mathcal{U}\}$ and $\bot \notin \mathcal{U}$. The two functions κ and δ being often consulted in conjunction, we note $\kappa\delta(A) = (\kappa(A), \delta(A))$ and $(a, d) \in \kappa\delta(A)$ if $a = \kappa(A)$ and $d \in \delta(A)$.

A *thread* is a pair $p{:}A$ where $p = u_1 \dots u_k \in \mathcal{U}^\star$ is a (possibly empty) path, and A some non-terminal symbol from $\mathcal{N}$. The empty path is denoted by ϵ. A *thread store* $\mathcal{S}$ is a finite set of threads (denoting a partial function from $\mathcal{U}$ to $\mathcal{N}$) such that its associated path set $\ker(\mathcal{S}) = \{p | p{:}A \in \mathcal{S}\}$ is closed by prefix (ie., $pu \in \ker(\mathcal{S}) \Rightarrow p \in \ker(\mathcal{S})$). We will often confuse a thread with its path.

A TA configuration is a tuple $\langle l, p, \mathcal{S} \rangle$ where l denotes the current position in the input string, p the path of the active thread, and $\mathcal{S}$ a thread store with $p{:}A \in \mathcal{S}$ for some non-terminal A. The initial configuration is

1. This function, while not essential, is useful to reduce complexity w.r.t. grammar sizes, as illustrated for TAGs (Section 3).

 Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6), pp. 193–200. Universitá di Venezia.

$c_{\text{init}} = \langle 0, \epsilon, \{\epsilon{:}S\}\rangle$ and the final one $c_{\text{final}} = \langle n, u, \{\epsilon{:}S, u{:}F\}\rangle$ where $u \in \delta(S) \cap \mathcal{U}$ 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 :

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

$$\langle l, p, \mathcal{S} \cup p{:}B\rangle \underset{\tau}{\vdash\!\!\!-} \langle l + |\alpha|, p, \mathcal{S} \cup p{:}C\rangle \qquad a_l = \alpha \text{ if } \alpha \neq \epsilon$$

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

$$\langle l, p, \mathcal{S} \cup p{:}B\rangle \underset{\tau}{\vdash\!\!\!-} \langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C\rangle \qquad (b, u) \in \kappa\delta(B) \wedge pu \notin \ker(\mathcal{S})$$

POP $[B]\, C \longmapsto D$: Terminates thread pu (if there is no existing subthread).

$$\langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C\rangle \underset{\tau}{\vdash\!\!\!-} \langle l, p, \mathcal{S} \cup p{:}C\rangle \qquad pu \notin \ker(\mathcal{S})$$

SPUSH $b\,[C] \longmapsto [b]\, D$: Resumes the subthread pu (if already created)

$$\langle l, p, \mathcal{S} \cup p{:}B \cup pu{:}C\rangle \underset{\tau}{\vdash\!\!\!-} \langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}D\rangle \qquad (b, u^s) \in \kappa\delta(B)$$

SPOP $[B]\, c \longmapsto D\,[c]$: Resumes the parent thread p of pu

$$\langle l, pu, \mathcal{S} \cup p{:}B \cup pu{:}C\rangle \underset{\tau}{\vdash\!\!\!-} \langle l, p, \mathcal{S} \cup p{:}D \cup pu{:}C\rangle \qquad (c, \bot) \in \kappa\delta(C)$$

Without restrictions, a thread may be suspended infinitely often. However, we will only consider h-TA, subclasses of TA where a thread may be suspended at most h times to return to its parent, which is a sufficient condition to ensure the termination of our tabular parsing algorithm (Section 6). Another key parameter is $d \leq |\mathcal{U}|$, the maximal number of subthreads of a thread that may be simultaneously alive. These two parameters h and d are sufficient to characterize the worst-case complexities of our tabular algorithm, namely $O(n^{2(1+h+dh)})$ for space and $O(n^{2(1+h+dh)+1})$ for time. For instance, TA with $h = 0$ and $d = 1$ are equivalent to Push-Down Automata and may be used to handle CFG, giving us optimal worst-case complexity $O(n^2)$ for space and $O(n^3)$ for time.

A finer complexity analysis (still to be confirmed) suggests that, modulo a generally satisfied condition on SWAP transitions[2], the worst-case complexities are actually $O(n^{2+h+dh+x})$ for space and $O(n^{3+h+dh+x})$ for time where $x = \min(h+dh, (l-d)(h+1))$ and l is the maximal number of subthreads created by a thread. When assuming $l = d$, we get $x = 0$ and complexities $O(n^{2+h+dh})$ for space and $O(n^{3+h+dh})$ for time.

Fig. 1 lists the transitions of a thread automaton $\mathcal{A}$ that may be used to recognize the **COUNT-3** language $a^n b^n c^n$ with a derivation of $a^3 b^3 c^3$ illustrated by Fig. 2. The characteristics of $\mathcal{A}$ are $h = 2$ and $d = 1$.

(1)	$K \overset{a}{\longmapsto} s_1$	$\kappa\delta(s_1) = (K, \{1\})$	(9)	$K \longmapsto [K]\, K$	
(2)	$[s_1]$ void $\longmapsto s_2\,[$void$]$	$\kappa\delta(s_2) = (\text{void}, \{\bot\})$	(10)	$S \longmapsto r_0$	$\kappa\delta(r_0) = (K, \{1\})$
(3)	void $[s_2] \longmapsto [$void$]\, s_3$		(11)	$[r_0]$ void $\longmapsto r_1\,[$void$]$	$\kappa\delta(r_1) = (\text{void}, \{1^s\})$
(4)	$s_3 \overset{b}{\longmapsto} s_4$	$\kappa\delta(s_1) = (\text{void}, \{1^s\})$	(12)	$[r_1]$ void $\longmapsto r_2\,[$void$]$	$\kappa\delta(r_2) = (\text{void}, \{1^s\})$
(5)	$[s_4]$ void $\longmapsto s_5\,[$void$]$	$\kappa\delta(s_5) = (\text{void}, \{\bot\})$	(13)	$[r_2]$ ret $\longmapsto$ ret	
(6)	void $[s_5] \longmapsto [$void$]\, s_6$		(14)	$K \longmapsto [K]\, t_0$	$\kappa\delta(t_0) = (\text{void}, \{\bot\})$
(7)	$s_6 \overset{c}{\longmapsto} s_7$	$\kappa\delta(s_7) = (\text{void}, \{1^s\})$	(15)	void $[t_0] \longmapsto [$void$]\, t_1$	$\kappa\delta(t_1) = (\text{void}, \{\bot\})$
(8)	$[s_7]$ ret $\longmapsto$ ret		(16)	void $[t_1] \longmapsto [$void$]$ ret	

Figure 1: TA transitions for **COUNT-3** language $a^n b^n c^n$

3. Parsing TAG

TAG parsing strategies may be encoded with TA in a straightforward way. The idea is to associate a thread to each elementary tree traversal. For instance, in Fig. 3, a thread p is started at R to traverse some elementary tree

2. The condition is that no sequence of SWAP transitions can loop from a configuration $\langle l, p, \mathcal{S} \cup p{:}A\rangle$ to a configuration $\langle r, p, \mathcal{S} \cup p{:}A\rangle$. That means for instance that we are not scanning regular expressions with Kleene stars using SWAP transitions.

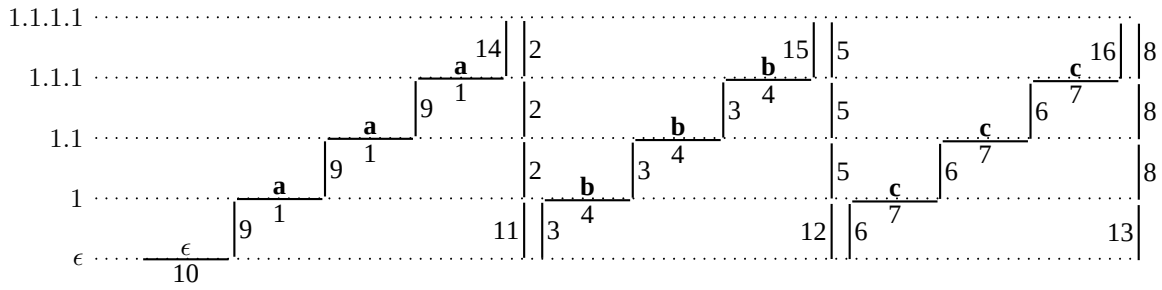
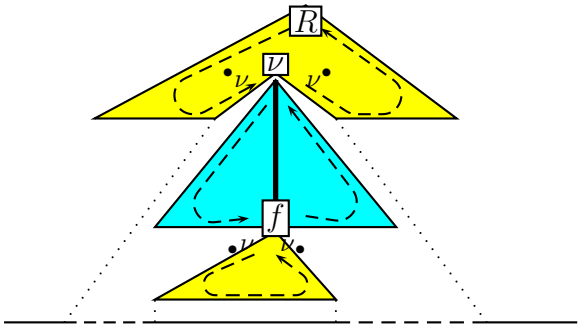


Figure 2: A derivation for $a^3b^3c^3$ starting from S

α and a subthread pu with $u = \mathrm{depth}(\nu)$ is started at node ν to handle an adjunction for non-terminal N (**Adj Call**). This subthread T selects and traverses some auxiliary tree β (**Adj Select**); is $\bot$-suspended when reaching the foot node of β to traverse the subtree $\alpha_{|\nu}$ rooted at ν (**Foot Suspend**); is resumed after the traversal of $\alpha_{|\nu}$ (**Foot Resume**); and is ended to return to its parent thread at the end of the traversal of β (**Adj Return**). [3]



(a) Traversals

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

(Adj Call)	$N^a \longmapsto [N^a]\, N^a$
(Adj Select)	$N^a \longmapsto {}^\bullet\mathrm{r}_\beta$
(Foot Suspend)	$[\,^\bullet\nu]\,\mathrm{foot} \longmapsto {}_\bullet\nu\,[\mathrm{foot}]$
(Foot Resume)	$\mathrm{adj}\,[\,_\bullet\mathrm{f}] \longmapsto [\mathrm{adj}]\;\mathrm{f}_\bullet$
(Adj Publish)	$\mathrm{r}_\beta{}^\bullet \longmapsto \mathrm{ret}$
(Adj Return)	$[\,\nu_\bullet]\,\mathrm{ret} \longmapsto \nu^\bullet$

(b) Transitions

Figure 3: Traversing trees using TA transitions

When traversing a tree τ with a thread p, the maximal number d of simultaneously alive sub-threads of p is bounded by its depth, at most one subthread being started and still alive for each node along a path of τ. A thread may only be suspended once ($h = 1$). Hence we get complexities $O(n^{4+2d})$ for space and $O(n^{5+2d})$ for time. These complexities are not optimal for TAG but correspond to those mentioned in (Éric Villemonte de la Clergerie, 2001) for a very similar tabular parsing algorithm, which has proved to be efficient in practice for linguistic grammars.

The best known worst-case complexities for TAG are $O(n^4)$ or $O(n^5)$ for space and $O(n^6)$ for time, and correspond to tabular algorithms where the traversal of the subtree $\alpha_{|\nu}$ may be shared between the different traversals of α (relying on the fact that the traversal of $\alpha_{|\nu}$ may be done independently of the adjunctions started on the path from the root of α down to ν). It is worth noting that we have very good reasons to believe that we can also achieve the $O(n^6)$ time complexity using TA and our tabular parsing algorithm (see discussion in Section 5).

4. Parsing Multi-Component TAG

Multi-Component [MC] TAG allows the adjoining or substitution of a finite set of elementary trees on nodes of a single elementary tree (*tree-local* MC-TAG) or on nodes of elementary trees of another set (*set-local* MC-

3. Note that without a triggering function κ, the **(Foot Suspend)** transition would be of the form $[\,^\bullet\nu]\;{}_\bullet\mathrm{f} \longmapsto \nu_\bullet\,[\,_\bullet\mathrm{f}]$, explicitly referring to nodes of two distinct elementary trees, and leading to complexities in $O(|G|^2)$ instead of $O(|G|)$ where $|G|$ denotes the size of the grammar.

TAG). We provide some intuitions how TA may be used to encode prefix-valid parsing strategies for both kinds of MC-TAG, restricting ourselves to sets of auxiliary trees.[4]

Tree-local MC-TAG. The idea is to assign a thread p to the traversal (in any order) of a tree set Σ and subthreads of p to the traversal of each tree in Σ.

More formally, for the traversal in any order of a tree set $\Sigma = \{\beta_1, \ldots, \beta_m\}$, we consider extended dotted points defined by

$$\Sigma{:}\rho\sigma \quad \rho \in \{ {}^{\bullet}\mathrm{i},\ {}_{\bullet}\mathrm{i},\ \mathrm{i}_{\bullet} \,|i = 1 \ldots m\}^{\star} \wedge \sigma \in \{ \mathrm{i}^{\bullet} \,|i = 1 \ldots m\}^{\star}$$

where ρ (resp. σ) is the list of trees in Σ which have been started but not yet completed (resp. completed). We note $\mathrm{ind}(\rho\sigma)$ the set of indices in $\{1, \ldots, m\}$ occurring in $\rho\sigma$. The rightmost index of ρ states which tree of Σ we are currently working on.

The non-terminal set $\mathcal{N}$ of the automaton includes these extended dotted points $\Sigma{:}\rho\sigma$, as well as the dotted nodes for each node ν and the symbols N^a and N^s for each non-terminal N of the grammar. We consider the thread label set $\mathcal{U} = \{1, \ldots, m\} \cup \{\mathrm{addr}(\tau, \nu) | \tau, \nu \text{ node of } \tau\}$ where $\mathrm{addr}(\tau, \nu)$ denotes the address of ν in τ.

We now associate to Σ the following (non exhaustive) set of transitions, where r_i denotes the root node of β_i and N_i the non-terminal label of r_i:

(Call set)	$N_i^a \longmapsto [N_i^a]\, N_i^a$	
(Start set with tree β_i)	$N_i^a \longmapsto \Sigma{:}\, {}^{\bullet}\mathrm{i}$	
(Resume set with tree β_i)	$N_i^a\, [\Sigma{:}\rho\sigma] \longmapsto [N_i^a]\, \Sigma{:}\rho\, {}^{\bullet}\mathrm{i}\, \sigma$	$i \notin \mathrm{ind}(\rho\sigma)$
(Start tree β_i)	$r_i \longmapsto [r_i]\ {}^{\bullet}\mathrm{r_i}$	$\kappa\delta(\Sigma{:}\rho\, {}^{\bullet}\mathrm{i}\, \sigma) = (r_i, \{i\})$
(Suspend β_i at foot)	$[\Sigma{:}\rho\, {}^{\bullet}\mathrm{i}\, \sigma]\, \mathrm{foot} \longmapsto \Sigma{:}\rho\, {}_{\bullet}\mathrm{i}\, \sigma\, [\mathrm{foot}]$	
(Suspend set at foot of β_i)	$[{}^{\bullet}\nu]\, \mathrm{foot} \longmapsto {}_{\bullet}\nu\, [\mathrm{foot}]$	$\kappa\delta(\Sigma{:}\rho\, {}_{\bullet}\mathrm{i}\, \sigma) = (\mathrm{foot}, \{\bot\})$
(Resume set after foot of β_i)	$\mathrm{adj}\, [\Sigma{:}\rho\, {}_{\bullet}\mathrm{i}\, \sigma] \longmapsto [\mathrm{adj}]\, \Sigma{:}\rho\, \mathrm{i}_{\bullet}\, \sigma$	
(Resume β_i after foot)	$\mathrm{adj}\, [{}_{\bullet}\mathrm{f}] \longmapsto [\mathrm{adj}]\ \mathrm{f}_{\bullet}$	$\kappa\delta(\Sigma{:}\rho\, \mathrm{i}_{\bullet}\, \sigma) = (\mathrm{adj}, \{i\})$
(End tree β_i)	$[\Sigma{:}\rho\, \mathrm{i}_{\bullet}\, \sigma]\, \mathrm{ret} \longmapsto \Sigma{:}\rho\, \mathrm{i}^{\bullet}\, \sigma$	
(Suspend set after tree β_i)	$[\nu_{\bullet}]\, \mathrm{ret} \longmapsto \nu^{\bullet}\, [\mathrm{ret}]$	$\kappa\delta(\Sigma{:}\rho\, \mathrm{i}^{\bullet}\, \sigma) = (\mathrm{ret}, \{\bot\}) \wedge \rho \neq \epsilon$
(End set)	$\Sigma{:}\sigma \longmapsto \mathrm{ret}$	$\mathrm{ind}(\sigma) = \{1, \ldots, m\}$
(Return from set)	$[\nu_{\bullet}]\, \mathrm{ret} \longmapsto \nu^{\bullet}$	

For a node ν of some elementary tree τ with non-terminal label N, we set

Adj	$\kappa\delta({}^{\bullet}\nu) = (N^a, \{\mathrm{addr}(\tau, \nu)\} \cup \{\mathrm{addr}(\tau, \mu) \| \mu \neq \nu\}^s)$ $\kappa\delta(\nu_{\bullet}) = (\mathrm{adj}, \{\mathrm{addr}(\tau, \mu)\}^s)$
Foot	$\kappa\delta({}_{\bullet}\mathrm{f}) = (\mathrm{foot}, \{\bot\})$

In terms of complexity, the number h of $\bot$-suspensions is bounded by $2m$ where m denotes the maximal number of trees per set while d is bounded by $\frac{1}{2} \max_\tau |\tau|$ where $|\tau|$ is the size of τ (number of nodes). However, in our complexity analysis, we use dh to bound the number of suspensions of a thread due to its subthreads. For tree-local MC-TAG, one can check than we can replace dh by $2 \max_\tau |\tau|$ and get much better results.

Set-local MC-TAG. We consider a single thread to traverse all trees of a set Σ (without subthreads for trees of Σ), using extended dotted nodes of the form:

$$\Sigma{:}\rho\sigma \quad \rho \in (\{ {}^{\bullet}\nu,\ {}_{\bullet}\nu,\ \nu_{\bullet},\ \nu^{\bullet} \,|\nu \text{ a node of } \beta_i\}/\{ \mathrm{r_1}^{\bullet}, \ldots, \mathrm{r_m}^{\bullet}\})^{\star} \wedge \sigma \in \{ \mathrm{r_1}^{\bullet}, \ldots, \mathrm{r_m}^{\bullet}\}^{\star}$$

where ρ (resp. σ) gives indication about each uncompleted (resp. completed) traversal of trees in Σ. We note $\mathrm{ind}(\rho\sigma)$ the set of indices $i \in \{1, \ldots, m\}$ for whose a dotted node of β_i occurs in $\rho\sigma$. The rightmost dotted node of ρ states which node of Σ we are currently working on.

The non-terminal set $\mathcal{N}$ of the automaton includes the extended dotted points $\Sigma{:}\rho\sigma$, as well as the symbols N^a and N^s for each non-terminal N of the grammar. We consider the thread label set $\mathcal{U} = \{\mathrm{addr}(\Sigma, \nu) | \Sigma, \nu \text{ node of } \Sigma\}$ where $\mathrm{addr}(\Sigma, \nu)$ denotes the unambiguous address of ν in the tree set Σ.

4. There is no special difficulty involved with substitution trees.

We associate to Σ the following set of transitions, most of them being straightforward extensions of the transitions for TAG listed in Fig. 3(b):

(Call set)	$N_i^a \longmapsto [N_i^a]\, N_i^a$	
(Start set with tree β_i)	$N_i^a \longmapsto \Sigma{:}\,^\bullet \mathrm{r_i}$	
(Add new tree β_i in set)	$N_i^a\, [\Sigma{:}\rho\sigma] \longmapsto [N_i^a]\, \Sigma{:}\rho\,^\bullet \mathrm{r_i}\, \sigma$	$i \notin \mathrm{ind}(\rho\sigma)$
(Foot suspend)	$[\Sigma{:}\rho\,^\bullet \nu\, \sigma]\, \mathrm{foot} \longmapsto \Sigma{:}\rho\,{}_\bullet \nu\, \sigma\, [\mathrm{foot}]$	
(Foot Resume)	$\mathrm{adj}\, [\Sigma{:}\rho\,{}_\bullet \mathrm{f_i}\, \sigma] \longmapsto [\mathrm{adj}]\, \Sigma{:}\rho\, \mathrm{f_i}{}_\bullet\, \sigma$	
(Suspend set after tree β_i)	$[\nu_\bullet]\, \mathrm{ret} \longmapsto \nu^\bullet\, [\mathrm{ret}]$	$\kappa\delta(\Sigma{:}\rho\, \mathrm{r_i}{}^\bullet\, \sigma) = (\mathrm{ret}, \{\bot\}) \wedge \rho \neq \epsilon$
(End set)	$\Sigma{:}\sigma \longmapsto \mathrm{ret}$	$\mathrm{ind}(\sigma) = \{1, \ldots, m\}$
(Return from set)	$[\nu_\bullet]\, \mathrm{ret} \longmapsto \nu^\bullet$	

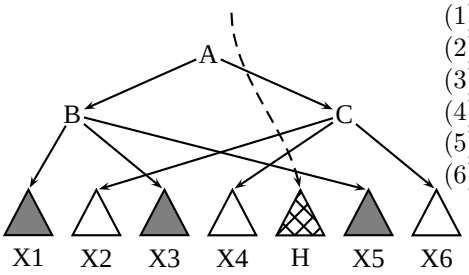
For a node ν of some elementary tree τ in set Σ and with non-terminal label N, we set

Adj	$\kappa\delta(\Sigma{:}\rho\,^\bullet \nu\, \sigma) = (N^a, \{\mathrm{addr}(\Sigma,\nu)\} \cup \{\mathrm{addr}(\Sigma,\mu) \mid \mu \neq \nu\}^s)$ $\kappa\delta(\Sigma{:}\rho\, \nu_\bullet\, \sigma) = (\mathrm{adj}, \{\mathrm{addr}(\Sigma,\mu)\}^s)$
Foot	$\kappa\delta(\Sigma{:}\rho\,{}_\bullet \mathrm{f_i}\, \sigma) = (\mathrm{foot}, \{\bot\})$

In terms of complexity, h is still bounded by $2m$ while d is now bounded by $\frac{1}{2} m \max_\tau |\tau|$ or, better, by $\frac{1}{2} \max_\Sigma |\Sigma|$ if we extend the notion of size to sets. Furthermore, as done for tree-local MC-TAG, one can easily check that we can replace dh by $2v$ in our complexity analysis where $v = \max_\Sigma |\Sigma|$. If the finer complexity analysis of Section 2 holds, we get a time complexity $O(n^{3+2(m+v)})$, to be compared with the time complexity $O(n^{2(m+v)})$ mentionned in (Boullier, 1999) for set-local MC-TAG.

5. Parsing Range Concatenation Grammars

Range Concatenation Grammars (Boullier, 2000b) are defined in terms of terminals, non-terminals, and range variables that may be instantiated by ranges of the input string. Many sub-classes of RCG may be identified and we characterize here a new one called *ordered simple RCG* [osRCG], equivalent to simple RCG which are themselves equivalent to Linear Context-Free Rewriting Systems (Weir, 1992). OsRCG are simple RCG where all ranges appearing in a literal are implicitly ordered: for instance $p(X_1X_2, X_3)$ means that range X_1 immediately precedes range X_2 which precedes X_3 (with some hole between X_2 and X_3). Fig. 4(a) shows an osRCG clause γ with two interleaved discontinuous sub-constituents B and C of clause head A, and a hole H inherited by A. Fig. 5 shows two simple osRCGs for the **COPY** language $\{ww | w \in \{a,b\}^*\}$ and for the **COUNT-**3 language $a^n b^n c^n$.



(a) Interleaved constituents

(1)	$A \longmapsto \gamma_{1.0}$	(7)	$[\gamma_{1.3}]\, \mathrm{void} \longmapsto \gamma_{1.4}\, [\mathrm{void}]$
(2)	$B \longmapsto [B]\, B$	(8)	$\mathrm{void}\, [\gamma_{1.4}] \longmapsto [\mathrm{void}]\, \gamma_{2.0}$
(3)	$[\gamma_{1.0}]\, \mathrm{void} \longmapsto \gamma_{1.1}\, [\mathrm{void}]$	(9)	$[\gamma_{2.0}]\, \mathrm{void} \longmapsto \gamma_{2.1}$
(4)	$C \longmapsto [C]\, C$	(10)	$[\gamma_{2.1}]\, \mathrm{void} \longmapsto \gamma_{2.2}$
(5)	$[\gamma_{1.1}]\, \mathrm{void} \longmapsto \gamma_{1.2}\, [\mathrm{void}]$	(11)	$\gamma_{2.2} \longmapsto \mathrm{ret}$
(6)	$[\gamma_{1.2}]\, \mathrm{void} \longmapsto \gamma_{1.3}\, [\mathrm{void}]$		

(b) TA transitions

Figure 4: osRCG clause $\gamma : \; A(X_1X_2X_3X_4, X_5X_6) \longrightarrow B(X_1, X_3, X_5)C(X_2, X_4, X_6)$

It is relatively straightforward to encode with TA a top-down pv parsing strategy for osRCG. Fig. 4(b) lists the transitions attached to γ while Fig. 6 illustrates how to apply these transitions to traverse γ where steps marked by (x) use transitions defined by other clauses. Fig. 1 also gives an idea of the transitions we get (with some simplifications) for the clauses r, s, t of the **COUNT-3** language of Fig. 5(b).

In terms of complexity, the TA associated with an osRCG G is characterized by $h = k-1$ the maximal number of holes in a constituent where k denotes the maximal arity of non-terminals of G and d the maximal number of

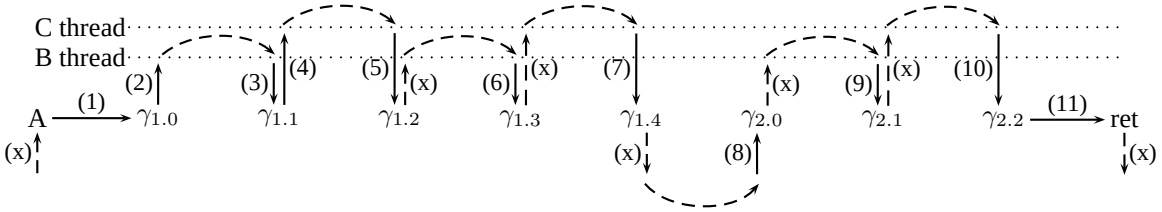
```
S(XY) -> K(X,Y).
K("","") -> .
K("a".X,"a".Y) -> K(X,Y).
K("b".X,"b".Y) -> K(X,Y).
```

(a) COPY language ww

```
S(ABC) -> K(A,B,C).                 % r
K("a".A,"b".B,"c".C) -> K(A,B,C). % s
K("","","") -> .                    % t
```

(b) COUNT-3 language $a^n b^n c^n$

Figure 5: Two simple osRCG



	$\gamma_{1.0}$	$\gamma_{1.1}$	$\gamma_{1.2}$	$\gamma_{1.3}$	$\gamma_{1.4}$	$\gamma_{2.0}$	$\gamma_{2.1}$
$\kappa\delta$	$(B, \{1\})$	$(C, \{2\})$	$(\text{void}, \{1\})$	$(\text{void}, \{2\})$	$(\text{void}, \{\bot\})$	$(\text{void}, \{1\})$	$(\text{void}, \{2\})$

Figure 6: Traversing γ

sub-constituents that are interleaved in a clause of G. [5] For instance, we get $h = d = 1$ for the **COPY** language and $h = k - 1$ and $d = 1$ for the **COUNT-k** language $\{a_1^n \dots a_k^n\}$.

The worst-case time complexity for RCG provided by (Boullier, 2000b) is $O(n^{k+v})$ where v is the maximal number of distinct range variables occurring in a clause. It is relatively difficult to compare with our results. However, if we suppose true our refined complexity analysis (Section 2) and assume $l = d$ for the maximal number l of literals in a clause, we can take $v = d(h + 1)$, which would give time complexities $O(n^{1+h+hd+d})$ for Boullier's algorithm applied on osRCG and $O(n^{3+h+hd})$ for our algorithm. Note it is possible to encode TAG as osRCG (Boullier, 2000a) in such a way that $h = 1$, $d = l = 2$ and $v = 4$, which would give the same time complexity $O(n^6)$ for both algorithms applied on TAG.

6. Dynamic Programming Interpretation

We don't provide in this extended abstract the full details of our tabular parsing algorithm, but only a simple intuition. Following a methodology presented in previous papers (Villemonte de la Clergerie and Alonso Pardo, 1998; Éric Villemonte de la Clergerie, 2001), our algorithm relies on a Dynamic Programming interpretation for TA which is based on the identification of a class of sub-derivations that may be represented by compact items and combined together to retrieve all derivations.

As shown in Fig. 7, for TA, such an elementary sub-derivation D

$$c_{\text{init}} |\frac{\star}{d_0} c_0 |\frac{\star}{d_1} \cdots c_{2i+1} |\frac{\star}{d_{2i+1}} c_{2i+2} \cdots c_{2m+1}$$

retraces the history of some thread $\pi = pu$ starting at c_0, reaching c_{2m+1} and suspended between $c_{2i+1} |\frac{\star}{d_{2i+1}} c_{2i+2}$ to return to either its parent thread p ($\bot$-suspension) or some sub-thread puv **still alive** at c_{2m+1} (v-suspension).

Using projections of configuration defined by $\overline{c} = \langle l, A\rangle$ and $\overline{c}^\kappa = \langle l, \kappa(A)\rangle$ where $c = \langle l, p, \mathcal{S} \cup p{:}A\rangle$, the essential information of D may be captured by an item $I = \overline{c_0}^\kappa / \mathcal{C} / \overline{c_{2m+1}}$ where $\mathcal{C} = v_1 : S_1 \dots v_i : S_i \dots v_m : S_m$, $S_i = \overline{c_{2i-1}}^\kappa \overline{c_{2i}}^\kappa$, and $v_i = v \in \mathcal{U}^\bot$ if $c_{2i-1} |\frac{\star}{d_{2i}} c_{2i}$ is a v-suspension. For instance, the derivation of Fig 7 gives the item $s/v : ab, \bot : cd, w : ef, v : gh/I$.

5. d may be strictly smaller than the number of sub-constituents.

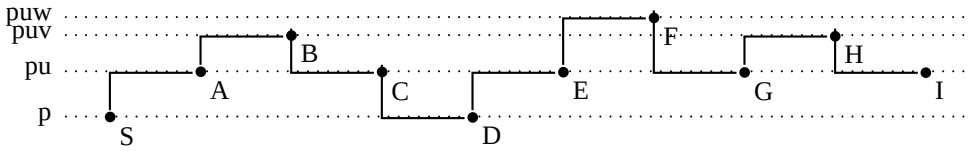


Figure 7: From a thread to item $s/v : ab, \perp : cd, w : ef, v : gh/I$

There are 5 application rules used to combine items and transitions, one for each kind of transitions. Except for the rules handling SWAP and PUSH transitions that only need an unique item, the other rules combine a transition with a *parent item* I related to some thread p and a *son item* J related to some subthread pu. These two items should *fit* together, in the sense that the holes of J should fill between the u-subparts of I and that one item should extend the other rightward. For instance, Fig. 8 shows a son item $J = a/\alpha', \perp : bc, \beta'/D$ which fits and extends a parent item $I = s/\alpha, u : ab, \beta/C$. These two items are combinable with a SPOP transition $[C]\, d \longmapsto E\, [d]$ to return an extended parent item $s/\alpha, u : ab, \beta, u : cd/E$.

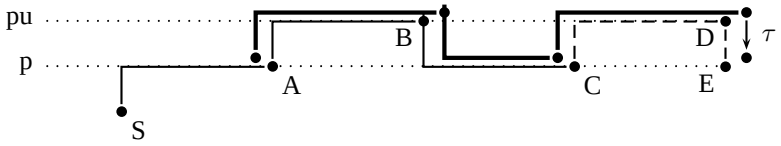


Figure 8: Combining SPOP transition $[C]\, d \longmapsto E\, [d]$ with $s/\alpha, u : ab, \beta/C$ and $a/\alpha', \perp : bc, \beta'/D$ to get item $s/\alpha, u : ab, \beta, u : cd/E$

The DP interpretation is complete and sound w.r.t. a straightforward evaluation of TA. As already mentioned, the upper-bounds for worst-case complexities are $O(n^{2(1+h+dh)})$ for space and $O(n^{2(1+h+dh)+1})$ for time, with better complexities in many cases. To provide some intuition, the space complexity is related to the maximal number h of $\perp$-suspensions and dh of v suspensions to keep trace in a item, with at most two positions in the input string per suspension and only one in many cases when the distance between the end position of a suspension and the start position of the next one belongs to some finite set independent of n. The time complexity is related to the number of distinct positions to consult when finding pairs of items that fit together to extend one of them. If we examine the case of Fig. 8, we see that we must consult all positions of parent item I to build the resulting item, check that some positions of I are also positions of the son item J, and consult the rightmost position of J to extend I. The other positions of J (in segments α' and β') may be ignored, which gives us the expected complexity.

7. Conclusion

We have introduced Thread Automata, a new automata formalism that may be used to describe a wide range of parsing strategies for a wide spectrum of MCS languages. An uniform tabular algorithm based on dynamic programming allows parsing with polynomial worst-case space and time complexities and will be soon implemented within system DyALog.

However, we still have to investigate the full spectrum of languages that may be covered by TA and explore how easy it is to describe parsing strategies for them. We would like also to explore the relationships of TA with other kinds of automata, such as (restricted) 2-Stack Automata, Embedded Push-Down Automata (Vijay-Shanker, 1988), or Tree-Walking transducers (Weir, 1992).

References

Boullier, Pierre. 1999. On Multicomponent TAG Parsing. In *TALN'99*, pages 321–326, Cargèse, Corse, France, July.

Boullier, Pierre. 2000a. On TAG Parsing. *Traitement Automatique des Langues (T.A.L.)*, 41(3):111–131. issued June 2001.

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

Joshi, Aravind K. 1987. An Introduction to Tree Adjoining Grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*. John Benjamins Publishing Co., Amsterdam/Philadelphia, pages 87–115.

Éric Villemonte de la Clergerie. 2001. Refining tabular parsers for TAGs. In *Proceedings of NAACL'01*, June.
Vijay-Shanker, K. 1988. *A Study of Tree Adjoining Grammars*. Ph.D. thesis, University of Pennsylvania, January. Available as Technical Report MS-CIS-88-03 LINC LAB 95 of the Department of Computer and Information Science, University of Pennsylvania.
Villemonte de la Clergerie, Eric and Miguel A. Alonso Pardo. 1998. A tabular interpretation of a class of 2-Stack Automata. In *Proc. of ACL/COLING'98*, August.
Weir, David. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.
Weir, David. 1992. Linear context-free rewriting systems and deterministic tree-walking transducers. In *Proc. of ACL'92*.