# A Unifying Approach for Multistack Pushdown Automata*

Salvatore La Torre[1], Margherita Napoli[1], and Gennaro Parlato[2]

[1] Dipartimento di Informatica, Università degli Studi di Salerno, Italy
[2] School of Electronics and Computer Science, University of Southampton, UK

**Abstract.** We give a general approach to show the closure under complement and decide the emptiness for many classes of multistack visibly pushdown automata (MVPA). A central notion in our approach is the *visibly path-tree*, i.e., a stack tree with the encoding of a path that denotes a linear ordering of the nodes. We show that the set of all such trees with a bounded size labeling is regular, and path-trees allow us to design simple conversions between tree automata and MVPA's. As corollaries of our results we get the closure under complement of ordered MVPA that was an open problem, and a better upper bound on the algorithm to check the emptiness of bounded-phase MVPA's.

## 1 Introduction

Pushdown automata working with multiple stacks (*multistack pushdown automata*, MPA for short) are a natural model of the control flow of shared-memory multithreaded programs. They are as much expressive as Turing machines already when only two stacks are used (the two stacks can act as the two halves of the tape portion in use). Therefore, the research on MPA's related to the development of formal methods for the analysis of multithreaded programs has mainly focused on decidable restricted versions of these models (as a sample of recent research see [2–6, 8, 9, 12, 14–18, 20]).

Formal language theories are a valuable source of tools for applications in other domains. Robust definitions, i.e., classes with decidable decision problems and closed under the main language operations (among all the Boolean operations), are particularly appealing. For instance, in the automata-theoretic approach to model-checking linear-time properties, the verification problem can be rephrased as a language inclusion or checking the emptiness of a language intersection, pattern-matching problems are often rephrased as membership queries. In a recent paper [10], the authors define a notion of *perfect* class of languages as a class that is closed under the Boolean operations and with a decidable emptiness problem, and investigate perfect classes modulo *bounded languages*.

Robust theories of MPA's introduced in the literature rely on both a restriction on the admitted behaviours [12–14] and the *visibility* [1] of stack operations (each symbol of the input alphabet explicitly identifies if a push onto stack $i$, or a pop from stack $i$, or no stack operation must happen on reading it).

* Partially supported by the FARB grants 2011-2013, Università degli Studi di Salerno.

E. Csuhaj-Varjú et al. (Eds.): MFCS 2014, Part I, LNCS 8634, pp. 377–389, 2014.

The restriction is imposed to gain the decidability of the emptiness problem. Visibility instead gains the closure under intersection, which does not hold also for a single stack pushdown automaton (with visibility, stack operations on a same stack synchronize). It is not a severe restriction for applications, the sequence of locations visited in the executions of programs being indeed visible.

In the literature, the results on MPA's are shown with different techniques for the different restrictions. Here, we introduce a unifying approach to show the two main technical challenges in proving robustness: emptiness decidability and closure under complement. We introduce the notion of *visibly path-tree*, that incidentally also allows us to define a robust class of *multistack visibly pushdown automata* (MVPA) that subsumes the main classes indentified in the literature.

A visibly path-tree is essentially a tree that encodes a visibly multi-stack word such that: (1) the left child of a node is its linear successor and the right-child relation captures the relations among matching *calls* (each causing a push transition on a specified stack) and *returns* (each causing a pop transition on a specified stack) and (2) it has an additional labeling that encodes a traversal of the tree that reconstructs the corresponding word. This labeling is formed by an ordered sequence of pairs, each composed of a direction (pointing to a neighbor in the tree) and an index (denoting the position of the pair to follow in the pointed neighbor). We define the class TMVPA by restricting MVPA to languages that for a given $k > 0$, contain only words that can be encoded into a visibly $k$-path-tree, i.e., a path-tree with at most $k$ pairs in the labeling of each node.

Our first result is to construct, for an MVPA $A$ over an $n$-stack alphabet, two tree automata $P_k$ and $\mathcal{A}_k$. $P_k$ accepts the set $PT_k$ of all visibly $k$-path-trees and has size $2^{O(nk)}$. If the input is a $k$-path-tree, $\mathcal{A}_k$ accepts it iff it encodes a word accepted by $A$. $\mathcal{A}_k$ has size $O(|A|^k)$. Thus, we reduce the emptiness problem for TMVPA to checking the emptiness of the intersection of $P_k$ and $\mathcal{A}_k$, that yields a $2^{O(k(n+\log|A|))}$ time solution. To show the closure under complement we first take the tree automaton for the intersection of $P_k$ and the complement of $\mathcal{A}_k$, and then, from this, we construct $\bar{A}$ that accepts the complement of the language accepted by $A$. The size of $\bar{A}$ is exponential in the size of $A$ and doubly exponential in $k$. From the effectiveness of these two proofs, we also get the decidability of containment, equivalence and universality problems.

Our approach is general, in the sense that it indeed works for each class of MVPA's that is defined by a restriction $R$ that *refines* the bounded path-tree restriction used to define TMVPA, i.e., such that there is a bound $k$ that suffices to encode in $k$-path-trees all the words satisfying $R$. This is sufficient for the complement since the actual restriction is captured by the resulting MVPA in the end. Instead for the emptiness, we also need to construct an additional tree automaton that exactly captures the restriction $R$ on the $k$-path-trees.

As corollaries of our results, we show the closure under complement for ordered MVPA that was open[1] and an algorithm in time $2^{(n+\log|A|)2^{O(d)}}$ to check the

[1] A proof via determinization was given in [8], but that is wrong since the language of all the words $(ab)^i c^j d^{i-j} x^j y^{i-j}$ is both accepted by a 2-stack OMVPA and inherently nondeterministic for MVPA's [13].

emptiness for bounded-phase MVPA, improving the $2^{|A|\,2^{O(d)}}$ bound shown in [11, 12] and matching the bound that can be derived from the results of [9].

Our path-tree representation has been inspired by the tree decomposition of bounded-phase and ordered words given in [19]. Concerning the closure under complement, our construction is structured as that from [12] but differs from it for the tree encoding of runs (path-tree) and thus in the tree automata constructions. Further, the approach from [12] does not apply directly to the ordered restriction and a different encoding would be needed. We give (i) a unifying approach for bounded-phase and ordered restrictions by a less restrictive limitation, and (ii) sufficient conditions for applying our constructions to new restrictions that can be captured by bounded path-trees.

## 2 Preliminaries

For $i, j \in \mathbb{N}$, we let $[i, j] = \{d \in \mathbb{N} \mid i \leq d \leq j\}$ and $[j] = [1, j]$.

**Words over call-return alphabets.** Given a finite alphabet $\Sigma$ and an integer $n > 0$, an $n$-stack call-return labeling is a $lab_{\Sigma,n} : \Sigma \to (\{ret, call\} \times [n]) \cup \{int\}$, and an $n$-stack call-return alphabet is $\widetilde{\Sigma}_n = (\Sigma, lab_{\Sigma,n})$. We fix the $n$-stack call-return alphabet $\widetilde{\Sigma}_n = (\Sigma, lab_{\Sigma,n})$ for the rest of the paper.

For $h \in [n]$, we denote $\Sigma_r^h = \{a \in \Sigma \mid lab_{\Sigma,n}(a) = (ret, h)\}$ (*set of returns*), $\Sigma_c^h = \{a \in \Sigma \mid lab_{\Sigma,n}(a) = (call, h)\}$ (*set of calls*), and $\Sigma_{int} = \{a \in \Sigma \mid lab_{\Sigma,n}(a) = int\}$ (*set of internals*). Moreover, $\Sigma_c = \bigcup_{h=1}^{n} \Sigma_c^h$, $\Sigma_r = \bigcup_{h=1}^{n} \Sigma_r^h$ and $\Sigma^h = \Sigma_c^h \cup \Sigma_r^h \cup \Sigma_{int}$.

A *stack-h context* is a word in $(\Sigma^h)^*$. For a word $w = a_1 \dots a_m$ over $\widetilde{\Sigma}_n$, denoting $C_h = \{i \in [m] \mid a_i \in \Sigma_c^h\}$ and $R_h = \{i \in [m] \mid a_i \in \Sigma_r^h\}$, the *matching relation* $\sim_h$ defined by $w$ is such that (1) $\sim_h \subseteq C_h \times R_h$, (2) if $i \sim_h j$ then $i < j$, (3) for each $i \in C_h$ and $j \in R_h$ s.t. $i < j$, there is an $i' \in [i, j]$ s.t. either $i' \sim_h j$ or $i \sim_h i'$, and (4) for each $i \in C_h$ (resp. $i \in R_h$) there is at most one $j \in [m]$ s.t. $i \sim_h j$ (resp. $j \sim_h i$). When $i \sim_h j$, we say that positions $i$ and $j$ *match* in $w$. If $i \in C_h$ and $i \not\sim_h j$ for any $j \in R_h$, then $i$ is an *unmatched call*. Analogously, if $i \in R_h$ and $j \not\sim_h i$ for any $j \in C_h$, then $i$ is an *unmatched return*.

**Multi-stack visibly pushdown languages.** A multi-stack visibly pushdown automaton over an $n$-stack call-return alphabet pushes a symbol on stack $h$ when it reads a call of the stack $h$, and pops a symbol from stack $h$ when it reads a return of the stack $h$. Moreover, it just changes its state, without modifying any stack, when reading an internal symbol. A special bottom-of-stack symbol $\perp$ is used: it is never pushed or popped, and is in each stack when computation starts. A *multi-stack visibly pushdown automaton* (MVPA) $A$ over $\widetilde{\Sigma}_n$ is $(Q, Q_I, \Gamma, \delta, Q_F)$ where $Q$ is a finite set of states, $Q_I \subseteq Q$ is the set of initial states, $\Gamma$ is a finite stack alphabet containing $\perp$, $\delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\perp\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{int} \times Q)$ is the transition function, and $Q_F \subseteq Q$ is the set of final states. Moreover, $A$ is *deterministic* if $|Q_I| = 1$, and $|\{(q, a, q') \in \delta\} \cup \{(q, a, q', \gamma') \in \delta\} \cup \{(q, a, \gamma, q') \in \delta\}| \leq 1$, for each $q \in Q$, $a \in \Sigma$ and $\gamma \in \Gamma$.

A *configuration* of an MVPA $A$ over $\widetilde{\Sigma}_n$ is a tuple $\alpha = \langle q, \sigma_1, \dots, \sigma_n \rangle$, where $q \in Q$ and each $\sigma_h \in (\Gamma \setminus \{\perp\})^*.\{\perp\}$ is a *stack content*. Moreover, $\alpha$ is *initial*

if $q \in Q_I$ and $\sigma_h = \perp$ for every $h \in [n]$, and *accepting* if $q \in Q_F$. A *transition* $\langle q, \sigma_1, \ldots, \sigma_n \rangle \xrightarrow{a}_A \langle q', \sigma'_1, \ldots, \sigma'_n \rangle$ is such that one of the following holds:

**[Push]** $a \in \Sigma_c^h$, $\exists \gamma \in \Gamma \setminus \{\perp\}$ such that $(q, a, q', \gamma) \in \delta$, $\sigma'_h = \gamma \cdot \sigma_h$, and $\sigma'_i = \sigma_i$ for every $i \in ([n] \setminus \{h\})$.

**[Pop]** $a \in \Sigma_r^h$, $\exists \gamma \in \Gamma$ such that $(q, a, \gamma, q') \in \delta$, $\sigma'_i = \sigma_i$ for every $i \in ([n] \setminus \{h\})$, and either $\gamma \neq \perp$ and $\sigma_h = \gamma \cdot \sigma'_h$, or $\gamma = \sigma_h = \sigma'_h = \perp$.

**[Internal]** $a \in \Sigma_{int}$, $(q, a, q') \in \delta$, and $\sigma'_h = \sigma_h$ for every $h \in [n]$.

For a word $w = a_1 \ldots a_m$ in $\Sigma^*$, a *run* of $A$ on $w$ from $\alpha_0$ to $\alpha_m$, denoted $\alpha_0 \xrightarrow{w}_A \alpha_m$, is a sequence of transitions $\alpha_{i-1} \xrightarrow{a_i}_A \alpha_i$, for $i \in [m]$. A word $w$ is accepted by $A$ if there exist an initial configuration $\alpha$ and an accepting configuration $\alpha'$ such that $\alpha \xrightarrow{w}_A \alpha'$. The language accepted by $A$ is denoted with $L(A)$. A language $L \subseteq \Sigma^*$ is a *multi-stack visibly pushdown language* (MVPL) if there exist an MVPA $A$ over $\widetilde{\Sigma}_n = (\Sigma, lab_{\Sigma,n})$ such that $L = L(A)$.

## 3 Visibly Path-Trees

**Trees.** A (binary) *tree* $T$ is any finite prefix-closed subset of $\{\swarrow, \searrow\}^*$. A *node* is any $x \in T$, the *root* is $\varepsilon$ and the edge-relation is implicit: *edges* are pairs of the form $(v, v.d)$ with $v, v.d \in T$ and $d \in \{\swarrow, \searrow\}$; for a node $v$, $v.\swarrow$ is its *left-child* and $v.\searrow$ is its *right-child*. We also denote with $v.\uparrow$ the *parent* of $v$, and with $D = \{\uparrow, \swarrow, \searrow\}$ the set of directions. For a finite alphabet $\Upsilon$, a $\Upsilon$*-labeled tree* is a pair $(T, \lambda)$ where $T$ is a tree, and $\lambda : T \to \Upsilon$ is a labeling map.

**Path-trees.** For a tree $T$, a $T$-path $\pi$ is any sequence $\pi = v_1, v_2, \ldots, v_\ell$ of $T$ nodes s.t. (1) $v_1$ is the root of $T$, (2) for $i \in [\ell - 1]$, $v_{i+1}$ is $v_i.d_i$ for some $d_i \in D$ ($\pi$ corresponds to a traversal of $T$), (3) for $i \in [\ell - 1]$, $v_\ell \neq v_i$ (the last node occurs once in $\pi$), (4) $\pi$ contains at least one occurrence of each node in $T$, and (5) for $i \in [1, \ell - 1]$, if $v_i$ is the first occurrence of a node $v \in T$ that has a left child, i.e., $v.\swarrow \in T$, then $v_{i+1}$ is the first occurrence of $v.\swarrow$ in $\pi$ (in the $T$ traversal, we first visit the left child of any newly discovered node).

For the tree $T_1$ in Fig. 1, $\pi_1 = \varepsilon, 1, 3, 1, 4, 1, \varepsilon, 2, 5, 2, \varepsilon, 1, 3, 6$ is a $T_1$-path. By deleting exactly one occurrence of any node in $\pi_1$ or concatenating more occurrences, the resulting sequence would not satisfy one of the above properties.

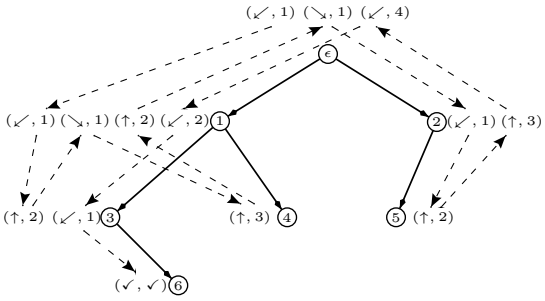


**Fig. 1.** A sample path-tree $T_1$

We introduce the notion of *path-tree*, that is, a labeled tree $(T, \lambda)$ that encodes a $T$-path in its labels as follows. Denote $dir^+_\checkmark = dir^+ \cup \{(\checkmark, \checkmark)\}$ where $dir = D \times \mathbb{N}$ and $\checkmark \notin D \cup \mathbb{N}$. Except for one node that is labeled with $(\checkmark, \checkmark)$, each other node has a label in $dir^+$. The labeling is such that by starting from the first pair of the root, we can build a chain of

pairs ending at $(\checkmark, \checkmark)$. In such a chain, a pair $(d, i)$ labeling a node $u$ is followed by the $i$-th pair labeling $u.d$ (i.e., a child or the parent of $u$, depending on $d$). For example, a pair $(\swarrow, 2)$ at a node $u$ denotes that the next pair in the chain is the second pair labeling its left child. The sequence of nodes visited by following such a chain is the path defined by $\lambda$ in $T$. To ensure that the defined path is a $T$-path, we require some additional properties on $\lambda$ which are detailed in the formal below. In Fig. 1, we give a path-tree $T_1$ and emphasize the chain defined by the labels of $T_1$ by linking the pairs with dashed arrows. The path defined by the labeling of $T_1$ is the path $\pi_1$ above, which is a $T_1$-path.

In the following, for a sequence $\rho = (d_1, i_1) \dots (d_h, i_h) \in dir^+_{\checkmark}$, we let $|\rho| = h$ and denote with $\rho[j] = (d_j, i_j)$, for $j \in [h]$.

**Definition 1.** *A $dir^+_{\checkmark}$-labeled tree $(T, \lambda)$ is a* path-tree *if there is exactly one node labeled with $(\checkmark, \checkmark)$ and, for every node $v$ of $T$ with $\lambda(v) = (d_1, i_1) \dots (d_h, i_h)$, and for every $j \in [h]$, the following holds:*

1. *if $i_j \neq \checkmark$ then $v.d_j$ is a node of $T$ and $i_j \leq |\lambda(v.d_j)|$* (pointed pair exists)*;*
2. *if $v \neq \varepsilon$ or $j > 1$, then there are exactly one node $u$ and one index $i \leq |\lambda(u)|$ s.t. $\lambda(u)[i] = (d, j)$ and $u.d = v$* (except for the first pair labeling the root, every pair is pointed exactly from one adjacent node)*;*
3. *if $v = u.d$, for a node $u$ of $T$ and $d \in \{\swarrow, \searrow\}$, then there exists $i \leq |\lambda(u)|$ s.t. $\lambda(u)[i] = (d, 1)$* (except for the root the first pair in a label is always pointed from the parent)*;*
4. *if $v.\swarrow \in T$ then $\lambda(v)[1] = (\swarrow, 1)$* (the first pair in a label always points to the first pair of the left child, if any)*;*
5. *if $j < h$ there is a $i > i_j$ s.t. $\lambda(v.d_j)[i]$ is $(\uparrow, j+1)$, if $d_j \in \{\swarrow, \searrow\}$, and $\lambda(v.d_j)[i]$ is $(\swarrow, j+1)$ (resp. $(\searrow, j+1)$), if $d_j = \uparrow$ and $v$ is a left (resp. right) child* (if a pair of $v$ points to a pair $\beta$ of an adjacent node $u = v.d_j$, the next pair of $v$ is pointed from a pair $\beta'$ of $u$ that follows $\beta$ in the $u$ labeling)*; moreover, for all $\ell \in [i_j + 1, i - 1]$, $\lambda(v.d_j)[\ell]$ does not point to a pair of $v$.*

*Path-trees define $T$-paths.* We define a function $tp$ that maps each path-tree $(T, \lambda)$ into a corresponding sequence of $T$ nodes, and show that indeed $tp(T, \lambda)$ is a $T$-path. Let $\pi = v_1, \dots, v_\ell$, and $d_1, \dots, d_\ell$, and $i_1, \dots, i_\ell$ be the maximal sequences such that (1) $v_1$ is the root and $\lambda(v_1)[1] = (d_1, i_1)$, and (2) for $j \in [2, \ell]$, $v_j = v_{j-1}.d_{j-1}$ and $\lambda(v_j)[i_{j-1}] = (d_j, i_j)$. We define $tp(T, \lambda)$ as the sequence $\pi$. Also, we say that, in $\pi$, the occurrence $v_1$ *corresponds to* the first pair of the root and the occurrence $v_{j+1}$ of a node $v \in T$ *corresponds to* the $i_j$-th pair of $v$, for $j \in [\ell - 1]$. The following lemma holds:

**Lemma 1.** *For each path-tree $\mathcal{T} = (T, \lambda)$, node $u$ and $i \leq |\lambda(u)|$, the $i$-th occurrence of $u$ in $tp(\mathcal{T})$ corresponds to the $i$-th pair of $u$, and $tp(\mathcal{T})$ is a $T$-path.*

*From $T$-paths to path-trees.* For a $T$-path $\pi$, we define $pt(\pi)$ as the tree whose labeling defines exactly $\pi$. We iteratively construct a sequence of labeling maps by concatenating a pair at each iteration.

Denote $dir^*_{\checkmark} = dir^* \cup \{(\checkmark, \checkmark)\}$. For a $T$-path $\pi = v_1, \dots, v_\ell$ and $i \in [\ell]$, let $\lambda^\pi_i : T \to dir^*_{\checkmark}$ be the mapping defined as follows:

- $\lambda_1^\pi(v_1) = (d_1, 1)$, $v_2 = v_1.d_1$ and $\lambda_1^\pi(v) = \varepsilon$ for every $v \in T \setminus \{v_1\}$;
- for $i \in [2, \ell - 1]$, $\lambda_i^\pi(v_i) = \lambda_{i-1}^\pi(v_i).(d_i, j + 1)$ where $j = |\lambda_{i-1}^\pi(v_{i+1})|$, $v_{i+1}$ is $v_i.d_i$ and for every $v \in T \setminus \{v_i\}$, $\lambda_i^\pi(v) = \lambda_{i-1}^\pi(v)$;
- $\lambda_\ell^\pi(v_\ell) = (\checkmark, \checkmark)$, and $\lambda_\ell^\pi(v) = \lambda_{\ell-1}^\pi(v)$ for every $v \in T \setminus \{v_\ell\}$.

Define $pt(\pi)$ as $(T, \lambda_\ell^\pi)$. From the definitions we get:

**Lemma 2.** *For any $T$-path $\pi$ and path-tree $Z$, $tp(pt(\pi)) = \pi$ and $pt(tp(Z)) = Z$.*

**Visibly path-trees.** Let $\mathcal{T} = (T, (\lambda_{dir}, \lambda_\Sigma))$ be such that $(T, \lambda_{dir})$ is a path-tree and $\lambda_\Sigma$ maps each node of $T$ to a symbol from $\widetilde{\Sigma}_n$. With $word_\mathcal{T}$ we denote the word $\lambda_\Sigma(v_1) \dots \lambda_\Sigma(v_h)$ where $v_1 \dots v_h$ is obtained from $tp(T, \lambda_{dir})$ by retaining only the first occurrences of each $T$ node. Also, for a node $z$ of $T$, we set $pos_\mathcal{T}(z) = i$ if $z = v_i$, that is, $pos_\mathcal{T}$ denotes the position corresponding to $z$ within $word_\mathcal{T}$.

Intuitively, a visibly path-tree is a path-tree with an additional labeling such that the right child relation captures exactly the matching relations defined by the word corresponding to the encoded $T$-path. Formally, a *visibly path-tree* $\mathcal{T}$ over $\widetilde{\Sigma}_n$ is a labeled tree $(T, (\lambda_{dir}, \lambda_\Sigma))$ such that (1) $(T, \lambda_{dir})$ is a path-tree and (2) $v$ is the right child of $u$ if and only if $pos_\mathcal{T}(u) \sim_h pos_\mathcal{T}(v)$ in $word_\mathcal{T}$, for some $h \in [n]$ (*right-child relation corresponds to the matching relations of $word_\mathcal{T}$*).

For $k > 0$, a *visibly $k$-path-tree* is a visibly path-tree $\mathcal{T} = (T, (\lambda_{dir}, \lambda_\Sigma))$ such that each $\lambda_{dir}(v)$ contains at most $k$ pairs.

*Tree encoding of words.* We can map each word $w = a_1 \dots a_\ell$ over $\widetilde{\Sigma}_n$ to a visibly path-tree $wt(w) = (T, (\lambda_{dir}, \lambda_\Sigma))$ as follows. The labeled tree $(T, \lambda_\Sigma)$ is such that $|T| = \ell$, $a_1$ labels the root of $T$ and for $i \in [2, \ell]$: $a_i$ labels the right child of the node labeled with $a_j$, $j < i$, if $j \sim_h i$ for some $h \in [n]$, and labels the left child of the node labeled with $a_{i-1}$, otherwise.

Define a path $\pi_w = v_1 \pi_2 \dots \pi_\ell$ of $T$ such that $v_1$ is the root of $T$ and for $i \in [2, \ell]$, $\pi_i$ is the ordered sequence of nodes that are visited on the shortest path in $T$ from the node corresponding to $a_{i-1}$ to that corresponding to $a_i$ (first node excluded). It is simple to verify that indeed $\pi_w$ is a $T$-path. Thus, we define $\lambda_{dir}$ to encode $\pi_w$, i.e., such that $tp(T, \lambda_{dir}) = \pi_w$.

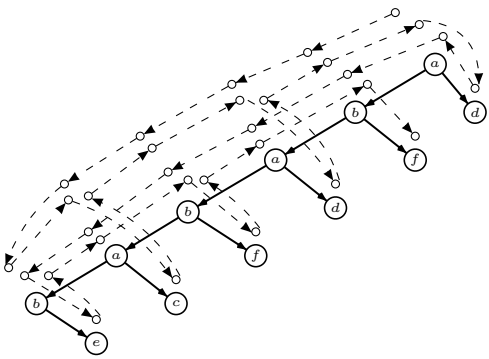


**Fig. 2.** The visibly path-tree $wt(w)$ for $w = (ab)^3\, cd^2\, ef^2$

A *$k$-path-tree word* $w$ over $\widetilde{\Sigma}_n$ is s.t. $wt(w)$ is a visibly $k$-path-tree over $\widetilde{\Sigma}_n$. Fig. 2 gives an example of the visibly 5-path-tree corresponding to the word $(ab)^3\, cd^2\, ef^2$ with call-return alphabet where $a$ is a call and $c, d$ are returns of stack 1, and $b$ is a call and $e, f$ are returns of stack 2.

In the following, we denote with $T_k(\widetilde{\Sigma}_n)$ the set of all $k$-path-tree words and with $PT_k(\widetilde{\Sigma}_n)$ the set of all the visibly $k$-path trees, over $\widetilde{\Sigma}_n$.

## 4 Two Base Constructions Used in Our Approach

We assume that the reader is familiar with the standard notion of nondeterministic tree automata (see [21]).

*Regularity of* $PT_k(\widetilde{\Sigma}_n)$. We construct a tree automaton $P_k$ accepting $PT_k(\widetilde{\Sigma}_n)$ as the intersection of two automata $P$ and $R$, where, for an input tree $\mathcal{T} = (T, (\lambda_{dir}, \lambda_\Sigma))$, $P$ checks that $(T, \lambda_{dir})$ is indeed a path-tree and $R$ checks that the right-child relation of $\mathcal{T}$ corresponds to the matching relations of $word_\mathcal{T}$.

Note that each property stated in Def. 1 is local to each node and its children. Thus, $P$ can check them by storing in its states the label of the parent of the current node. Assuming a bound $k$ on the number of pairs labeling each node, the size of $P$ is thus exponential in $k$.

To construct $R$, we first construct an automaton for the negation of property (2) of the definition of visibly path-tree and then complement it.

Fix $\mathcal{T} = (T, (\lambda_{dir}, \lambda_\Sigma))$ and for any two nodes $u, v$ of $T$, define $<$ s.t. $u < v$ holds iff the first occurrence of $u$ precedes the first occurrence of $v$ in $tp(T, \lambda_{dir})$.

We recall property (2): "a node $v$ is the right child of $u$ in $T$ if and only if $pos_\mathcal{T}(u) \sim_h pos_\mathcal{T}(v)$ in $word_\mathcal{T}$, for some $h \in [n]$". By the definition of $\sim_h$, $h \in [n]$, the negation of property (2) holds iff either:

1. there are $u, v \in T$ s.t. $v$ is the right child of $u$, $\lambda_\Sigma(u) \in \Sigma_c^h$ (call of stack $h$) and $\lambda_\Sigma(v) \notin \Sigma_r^h$ (not a return of stack $h$); or
2. there are $u, v \in T$ s.t. $u < v$, and (i) $\lambda_\Sigma(u) \in \Sigma_c^h$ and $u$ has no right child, and (ii) $\lambda_\Sigma(v) \in \Sigma_r^h$ and $v$ is not a right child (i.e., by the right-child relation, there are a call and a return of stack $h$ that are both unmatched); or
3. there are $u, v \in T$ s.t. $v$ is the right child of $u$, $\lambda_\Sigma(u) \in \Sigma_c^h$, and either:
   *i.* there is a $w \in T$ s.t. $u < w < v$ and either (a) $\lambda_\Sigma(w) \in \Sigma_c^h$ and $w$ has no right child, or (b) $\lambda_\Sigma(w) \in \Sigma_r^h$ and $w$ is not a right child (i.e., the right-child relation leaves unmatched either a call or a return occurring between a matched pair of the same stack $h$); or
   *ii.* there are $w, z \in T$ s.t. $z$ is the right child of $w$, $\lambda_\Sigma(w) \in \Sigma_c^h$, and either $w < u < z < v$ or $u < w < v < z$ (i.e., the right-child relation restricted to stack $h$ is not nested).

For $h \in [n]$ and assuming $(T, \lambda_{dir})$ is a path tree s.t. $|\lambda_{dir}(u)| \leq k$ for each $u \in T$, we construct an automaton $B_h$ as the union of four automata, one for each of the above violations 1, 2, 3.*i* and 3.*ii*. Thus, $B_h$ accepts $\mathcal{T}$ iff the right-child relation of $\mathcal{T}$ does not capture properly the matching relation $\sim_h$ of $word_\mathcal{T}$ (i.e., property (2) does not hold w.r.t. the matching relation $\sim_h$).

The first automaton nondeterministically guesses a node $u$ and then accepts iff $u$ has a right child, say $v$, and the labels of $u$ and $v$ witness violation 1. The number of states of this automaton is constant w.r.t. $k$ and $n$.

In the other violations, the $<$ relation is used. We now describe an efficient construction to capture this relation by a tree automaton on $k$-path-trees and then conclude the discussion on the remaining violations.

*Checking* $u < v$. We first assume that the input tree has two marked nodes $u$ and $v$. Observe that $u < v$ holds iff either (a) $v$ is in the subtree rooted at $u$, or (b) there are a node $w$ with two children and $i \leq |\lambda_{dir}(w)|$ s.t. $u$ and $v$ are in two different subtrees rooted at the children of $w$, and in $tp(\mathcal{T})$ the $i$-th occurrence of $w$ occurs in between the first occurrence of $u$ and the first occurrence of $v$.

Property (a) can be easily checked by a top-down tree automaton with a constant number of states. For property (b), we construct a tree automaton $S$ that nondeterministically guesses the node $w$, its child $w_u$ whose subtree contains $u$ and its child $w_v$ whose subtree contains $v$. Then, denoting $\lambda_{dir}(w) = (d_1, i_1) \dots (d_\ell, i_\ell)$, it guesses two pairs $(d_r, i_r), (d_s, i_s)$ such that $r < s$, $w.d_r = w_u$ and $w.d_s = w_v$, with the meaning that: the first occurrence of $u$ must be in between the $r$-th and the $(r+1)$-th occurrence of $w$, and the first occurrence of $v$ must be in between the $s$-th and the (s+1)-th occurrence of $w$ (if any). By Lemma 1, this is ensured by checking that $u$ is visited on its first pair by starting from the $i_r$-th pair of $w$ and before reaching the $i_{r+1}$-th pair of $w$, and similarly $v$ w.r.t. the $i_s$-th and $i_{s+1}$-th pairs of $w$. The guessed $i_r$ and the request of searching for the first occurrence of $u$ are passed onto $w_u$, analogously $i_s$ and $v$ are passed onto $w_v$. Each such request is then passed along a nondeterministically guessed path in the respective subtrees, updating the indices according to the given intuition. $S$ rejects the tree if it can visit the requested node but not on its first pair, or it reaches a leaf, and either (i) it has not guessed the node $w$ yet or (ii) is on a selected path and the requested node was not found. In all the other cases it accepts.

Overall, we can construct $S$ with an initial state (that is used also to store that $w$ has not being guessed yet), an acceptance state, a rejection state and states of the form $(i, x)$ where $i \in [1, k]$ and $x \in \{u, v\}$ (storing the request after $w$ is guessed). Thus, in total, it has $3 + 2\,k$ states.

$<$ *relations over many nodes.* To check Boolean combinations of constraints of the form $u < v$, we can of course use the standard construction with unions and intersections of copies of $S$, that will yield an automaton with polynomially many states. A more efficient construction that is linear in $k$ can be obtained by generalizing the approach used for $S$ to capture the wished relation directly.

*Handling the remaining violations.* By using an automaton as above to check a proper ordering of the guessed nodes, we can design the tree automata for the rest of the violations quite easily. For example, consider the violation 3.*ii*. Denote with $S'$ the automaton that checks $w < u < z < v$ and with $S''$ the automaton that checks $u < w < v < z$. Assuming that $u, v, w, z$ are marked in the input tree, the properties $v$ is the right child of $u$, $z$ is the right child of $w$, and $u, w$ are labeled with calls of stack $h$ are local to the nodes $u, w$ and their right children, thus can be easily checked by a tree automaton $M$ with a constant number of states. Thus, we construct a tree automaton, that captures the intersection of $M$ with the union of $S'$ and $S''$. This automaton, by a direct construction of the automaton equivalent to the union of of $S'$ and $S''$ as observed above, can be built with a number of states linear in $k$. From it, the automaton $V_{3ii}$ checking for violation 3.*ii* can be obtained by removing the assumption on the marking

of $u, v, w, z$ as in the usual projection construction. Thus it can be constructed with a number of states that is linear in $k$.

Similarly for the other violations we get corresponding tree automata with $O(k)$ states, and thus also $B_h$ also has $O(k)$ states.

For each tree $\mathcal{T} \in L(P)$ that is not accepted by $B_h$, we get that its right-child relation does not violate the $\sim_h$ relation. Thus, denoting with $\bar{B}_h$ the automaton obtained by complementing $B_h$, if we take the intersection of all $\bar{B}_h$ for $h \in [n]$, we get an automaton checking property (2) of the definition of visibly $k$-path-tree provided that the input tree $\mathcal{T} \in L(P)$, i.e., $(T, \lambda_{dir})$ is a path-tree. Since complementation causes an exponential blow-up in the number of states, the size of each $\bar{B}_h$ is $2^{O(k)}$, and the automaton resulting from their intersection has size $2^{O(n\,k)}$. Therefore, we get:

**Theorem 1.** *For $k \in \mathbb{N}$, there is an effectively constructible tree automaton accepting $PT_k(\widetilde{\Sigma}_n)$ of size exponential in $n$ and $k$.*

*Tree automaton for an* MVPA. By assuming $\mathcal{T} \in PT_k(\widetilde{\Sigma}_n)$, we can construct a tree automaton $\mathcal{A}_k$ that captures the runs of $A$ over $word_{\mathcal{T}}$.

Assume that our tree automaton can read the input tree $\mathcal{T}$ by moving along the path $tp(\mathcal{T})$ (not just top-down but as a tree-walking automaton that moves by following the encoded path). Also assume that each node labeled with a call is also labeled with a stack symbol (that is used to match push and pop transitions). We can then simulate any run of $A$ by mimicking its moves at each node $u$ when it is visited for the first time (from Lemma 1 this happens when a node is visited on its first pair). To construct a corresponding top-down tree automaton we use the fact that on each run of the above automaton we cross a node at most $k$ times and according to the directions annotated in its labels. Thus we can use as states ordered tuples of at most $k$ states of $A$, and design the transitions as in the standard construction from two-way to one way finite automata, moving top-down in the tree and matching the state of a node with the states of its children according to the directions in the labels. When such a matching is not possible, the automaton halts rejecting the input. On the only node labeled with $(\checkmark, \checkmark)$, it accepts iff $A$ accepts. The tree automaton $\mathcal{A}_k$ is then obtained from this automaton by projecting out the stack symbols from the input, and therefore, its size is $O(|A|^k)$.

**Lemma 3.** *For an* MVPA *$A$ and a $k$-path-tree $\mathcal{T}$, $\mathcal{A}_k$ accepts $\mathcal{T}$ iff $word_{\mathcal{T}} \in L(A)$. The size of $\mathcal{A}_k$ is $O(|A|^k)$.*

## 5 A General Approach for Complement and Emptiness

We first introduce two properties for classes of MVPL languages by using the notion of $k$-path-trees. Given an MVPL class $\mathcal{L}$, the first property requires that there is a $k$ s.t. each word in a language of $\mathcal{L}$ can be encoded as a visibly $k$-path-tree. We show that each class that has such a property is closed under complement. The second property requires in addition that the language of all

$k$-path-trees corresponding to words in the languages of $\mathcal{L}$ is regular. We show that for each such class the emptiness problem is decidable.

MVPL **classes and properties.** We consider here a generic notation $\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n)$ for sets of words over a call-return alphabet $\widetilde{\Sigma}_n$, each set being characterized by a particular restriction, captured by the symbol $\mathcal{X}$, and parameterized over a possibly empty tuple of integer-valued parameters $\bar{p}$. For example, we have already defined the set $T_k(\widetilde{\Sigma}_n)$ of the $k$-path-tree words. Another example is $C_b(\widetilde{\Sigma}_n)$ that denotes the set of all words $w \in \Sigma^*$ that can be split into $w_1 \dots w_b$, where $w_i$ contains calls and returns of at most one stack, for $i \in [b]$ (*bounded context-switching* [20]).

We denote with $\mathcal{X}$MVPL the class of all the languages $L \subseteq \Sigma^*$ such that there exist an $n$-stack call-return labeling $lab_{\Sigma,n}$, a valuation of the parameters $\bar{p}$, and an MVPA $A$ over $\widetilde{\Sigma}_n = (\Sigma, lab_{\Sigma,n})$ for which $L = L(A) \cap \mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n)$. For example, TMVPL denotes such a class for $\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n) = T_k(\widetilde{\Sigma}_n)$.

A class $\mathcal{X}$MVPL is *PT-covered* if for each $n > 0$ and $\bar{p}$, there exists a $k > 0$ such that $\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n) \subseteq T_k(\widetilde{\Sigma}_n)$. We refer to such $k$ as the *PT-parameter*. A class $\mathcal{X}$MVPL is *PT-definable* if it is PT-covered and there is an automaton $\mathcal{A}_{\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n)}$ that accepts the set of all trees $wt(w)$ s.t. $w \in \mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n)$. Clearly, TMVPL is PT-definable.

**Deterministic** MVPA**'s do not capture all** TMVPL**.** Let $L_1$ be the language $\{(ab)^i\, c^{i-j} d^j\, e^{i-j} f^j \mid i \geq j > 0\}$ and assume the call-return alphabet as in the example of Fig. 2. An MVPA $A$ accepting $L_1$ just needs to guess the value of $j$ (by nondeterministically switching to a different symbol to push onto both stacks) after reading a prefix $(ab)^j$ and then check exact matching with the returns. Also, notice that for each $w \in L_1$, $w$ is also 5-path-tree (see Fig. 2), and since $L_1$ is inherently nondeterministic for MVPAs [13], we get:

**Lemma 4.** *The class of* MVPA*'s that captures* TMVPL *is not determinizable.*

**Complement.** Consider a $\mathcal{X}$MVPL language $L$ over a call-return alphabet $\widetilde{\Sigma}_n$. The *complement* of $L$ in $\mathcal{X}$MVPL is $\bar{L} = \mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n) \setminus L$.

Despite of Lemma 4, we show that TMVPL, and in general any PT-covered MVPL class, are all closed under complement. For this, consider an MVPA $A$, and denote with $P_k$ the tree automaton accepting $PT_k(\widetilde{\Sigma}_n)$ and $\mathcal{A}_k$, as in Section 4. We can complement $\mathcal{A}_k$ and then take the intersection with $P_k$, thus capturing all the trees $\mathcal{T} \in PT_k(\widetilde{\Sigma}_n)$ s.t. $word_{\mathcal{T}}$ is not accepted by $A$. The size of the resulting tree automaton $\bar{\mathcal{B}}_k$ is exponential in $|A|$ and doubly exponential in $k$.

The following lemma concludes the proof.

**Lemma 5.** *For any tree automaton $\mathcal{H}$ with $\mathcal{L}(\mathcal{H}) \subseteq PT_k(\widetilde{\Sigma}_n)$, there is an effectively constructible* MVPA *$H$ over $\widetilde{\Sigma}_n$ such that $L(H)$ is the set of all the $k$-path-tree words $word_{\mathcal{T}}$ such that $\mathcal{T} \in \mathcal{L}(\mathcal{H})$. Moreover, the size of $H$ is polynomial in the size of $\mathcal{H}$ and exponential in $n$.*

Intuitively, from $\mathcal{H}$, we can construct an MVPA $H$ that mimics $\mathcal{H}$ transitions as follows: on internal symbols, $H$ moves exactly as $\mathcal{H}$ (there is no right child);

on call symbols, $H$ enters the state that $\mathcal{H}$ would enter on the left child and pushes onto a stack the one that $\mathcal{H}$ would enter on the right child; on return symbols, $H$ acts as if the current state is the one popped from the stack. The correctness of this construction relies on the fact that for each tree $\mathcal{T} \in PT_k(\widetilde{\Sigma}_n)$, the successor position in $word_{\mathcal{T}}$ corresponds to the left child in $\mathcal{T}$, if any, or else, to a uniquely determined node (a right child) labeled with a return matching the most recent still unmatched call of the stack.

From the above lemma we can construct an MVPA $\bar{A}$ that accepts all words $w$ such that $wt(w) \in L(\vec{\mathcal{B}}_k)$. Being the size of $\bar{A}$ polynomial in $|\vec{\mathcal{B}}_k|$ and exponential in $n$, we get (notice that since $\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n) \subseteq T_k(\widetilde{\Sigma}_n)$, the words in $L(\bar{A})$ that are not in $\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n)$ are ruled out by the intersection with this set):

**Theorem 2.** *Any PT-covered class $\mathcal{X}$*MVPL *is closed under complement. Also, for an* MVPA *$A$, there is an effectively constructible* MVPA *$\bar{A}$ s.t. $L(\bar{A}) \cap \mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n) = \mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n) \setminus L(A)$, and its size is exponential in $|A|$ and doubly exponential in the PT-parameter.*

As corollaries of the above theorem, we get the closure under complement of two well-known classes of MVPL: PMVPL defined by the sets $P_d(\widetilde{\Sigma}_n)$ of all words with a number of *phases* bounded by $d$ [12], and OMVPL defined by the sets $O(\widetilde{\Sigma}_n)$ of all words for which when a pop transition happens it is always from the least indexed non-empty stack [7]. In fact, by the tree-decompositions given in [19] we get that OMVPL is PT-covered with $k = (n+1)2^{n-1}+1$, where $n$ is the number of stacks, and PMVPL is PT-covered for $k = 2^d + 2^{d-1} + 1$, where $d$ is the bound on the number of phases.

**Corollary 1.** OMVPL *(resp.* PMVPL*) is closed under complement. Moreover, for an* MVPA *$A$, there is an effectively constructible* MVPA *$\bar{A}$ s.t. $L(\bar{A}) \cap O(\widetilde{\Sigma}_n) = O(\widetilde{\Sigma}_n) \setminus L(A)$ (resp. $L(\bar{A}) \cap P_d(\widetilde{\Sigma}_n) = P_d(\widetilde{\Sigma}_n) \setminus L(A)$), and its size is exponential in the size of $A$ and triply exponential in $n$ (resp. $d$, where $d$ is the bound on the number of phases).*

**Emtpiness.** For PT-definable classes $\mathcal{X}$MVPL, we reduce the emptiness problem to the emptiness for tree automata by constructing a tree automaton given as intersection of $\mathcal{A}_{\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n)}$, $P_k$ and $\mathcal{A}_k$, where $k$ is the PT-parameter.

**Theorem 3.** *The emptiness problem for any PT-definable class $\mathcal{X}$*MVPL *is decidable in $|\mathcal{A}_{\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n)}|\,|A|^k\, 2^{O(nk)}$ time, where $A$ is the starting* MVPA *and $n$ is the number of stacks.*

The size of $\mathcal{A}_{\mathcal{X}_{\bar{p}}(\widetilde{\Sigma}_n)}$ is bounded by the size of $P_k$ in both OMVPL and PMVPL (we can construct such automata from simple MSO formulas capturing the restrictions and using the linear successor relation, see [12, 19]), thus we get:

**Corollary 2.** *The emptiness problem for* PMVPL *(resp.* OMVPL*) is decidable in $2^{(n+\log|A|)2^{O(d)}}$ (resp. $|A|^{2^{O(n \log n)}}$) time, where $A$ is the starting* MVPA*, $n$ is the number of stacks and $d$ is the bound on the number of phases.*

## References


1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211. ACM (2004)
2. Atig, M.F.: Model-checking of ordered multi-pushdown automata. Logical Methods in Computer Science 8(3) (2012)
3. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of multi-pushdown automata is 2Etime-complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
4. Atig, M.F., Bouajjani, A., Narayan Kumar, K., Saivasan, P.: Linear-time model-checking for multithreaded programs under scope-bounding. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 152–166. Springer, Heidelberg (2012)
5. Bansal, K., Demri, S.: Model-checking bounded multi-pushdown systems. In: Bulatov, A.A., Shur, A.M. (eds.) CSR 2013. LNCS, vol. 7913, pp. 405–417. Springer, Heidelberg (2013)
6. Bollig, B., Kuske, D., Mennicke, R.: The complexity of model checking multi-stack systems. In: LICS, pp. 163–172. IEEE Computer Society (2013)
7. Breveglieri, L., Cherubini, A., Citrini, C., Crespi-Reghizzi, S.: Multi-push-down languages and grammars. Int. J. Found. Comput. Sci. 7(3), 253–292 (1996)
8. Carotenuto, D., Murano, A., Peron, A.: 2-visibly pushdown automata. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 132–144. Springer, Heidelberg (2007)
9. Cyriac, A., Gastin, P., Kumar, K.N.: MSO decidability of multi-pushdown systems via split-width. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 547–561. Springer, Heidelberg (2012)
10. Esparza, J., Ganty, P., Majumdar, R.: A perfect model for bounded verification. In: LICS, pp. 285–294. IEEE (2012)
11. La Torre, S., Madhusudan, P., Parlato, G.: An infinite automaton characterization of double exponential time. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 33–48. Springer, Heidelberg (2008)
12. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS, pp. 161–170. IEEE Computer Society (2007)
13. La Torre, S., Madhusudan, P., Parlato, G.: The language theory of bounded context-switching. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 96–107. Springer, Heidelberg (2010)
14. La Torre, S., Napoli, M., Parlato, G.: Scope-Bounded Pushdown Languages. In: DLT. LNCS. Springer (2014)
15. La Torre, S., Napoli, M.: Reachability of multistack pushdown systems with scope-bounded matching relations. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 203–218. Springer, Heidelberg (2011)
16. La Torre, S., Napoli, M.: A temporal logic for multi-threaded programs. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) TCS 2012. LNCS, vol. 7604, pp. 225–239. Springer, Heidelberg (2012)
17. La Torre, S., Parlato, G.: Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width. In: FSTTCS. LIPIcs, vol. 18, pp. 173–184. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
18. Lal, A., Touili, T., Kidd, N., Reps, T.: Interprocedural analysis of concurrent programs under a context bound. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 282–298. Springer, Heidelberg (2008)

19. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: POPL, pp. 283–294. ACM (2011)
20. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
21. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, vol. 3, pp. 389–455. Springer (1997)