A Unifying Approach for Multistack Pushdown Automata (Track B)

Salvatore La Torre[1], Margherita Napoli[1], 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, that also shows that this problem is fixed parameter tractable in the number of phases.

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 [2–5, 7, 8, 12–14, 16]).

Formal language theories are a valuable source of tools for applications in other domains. In this context 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 the intersection of two languages, pattern-matching problems are often rephrased as membership queries. In a recent paper [9], 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 relies on both a restriction on the admitted behaviours 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). In particular,

visibility gains the closure under intersection (that does not hold also for a single stack pushdown automaton) since it has the effect of synchronizing the stack operations on a same stack. It is not a severe restriction for applications, the sequence of locations visited in the executions of programs being indeed visible. Imposing a restriction on the behaviors of an MPA gains instead the decidability of the emptiness problem (note that this happens independently of visibility for the above observation). Showing the decidability of emptiness for meaningful classes of MPA is often challenging, so that showing the closure under complement when the expressiveness of the deterministic and nondeterministic versions of the considered class of MPA's is different (as it is often the case). The closure under union is instead simple since MPA's are nondeterministic.

In the literature, the results on MPA's have been shown with different techniques for the different restrictions that have been considered. In this paper, we introduce a unifying approach to show the two main technical challenges in proving such theories robust: emptiness decidability and closure under complement. For this 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 including the *ordered* MVPA [6, 7] and the *bounded-phase* MVPA [12].

A visibly path-tree is essentially a tree that encodes a visibly multi-stack word such that: (1) its right-child relation captures the relations among matching *calls* (each causing a push transition on a specified stack i) and *returns* (each causing a pop transition on a specified stack i) and if any, the left child of a node is its linear successor (*stack tree* [12]), and (2) it has an additional labeling that encodes a traversal of the tree that reconstructs the linear order of the corresponding word. This additional 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 over the given alphabet 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 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. We show the closure under complement by first taking the intersection of P_k and the complement of $\mathcal{A}_k$, and then by using the fact that the accepted trees are k-path-trees (and in particular, stack trees), we construct the MVPA $\bar{A}$ 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. For the effectiveness of these two proofs, we also get the decidability of containment, equivalence and universality.

Our approach is general, in the sense that it 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 captures the restriction R on the k-path-trees.

By the tree decompositions given in [15], we get a bound $k = (n+1)2^{n-1}+1$ for ordered MVPA where n is the number of stacks, and $k = 2^d+2^{d-1}+1$ for bounded-phase MVPA, where d is the bound on the number of phases. Moreover, from [12], the bounded-phase restriction is captured on k-path-trees by a tree automaton of size doubly exponential in d. Therefore, as corollaries of our results we show the closure under complement for the class of ordered MVPA that was open[3] and an algorithm in time $2^{(n+\log|A|)2^{O(d)}}$ to check the emptiness for bounded-phase MVPA, improving the $2^{|A|\,2^{O(d)}}$ bound shown in [10], and thus proving that this problem is fixed parameter tractable in the number of phases (being $n = O(d)$).

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 Σ and an integer $n > 0$, an n-stack call-return labeling is a mapping $lab_{\Sigma,n} : \Sigma \to (\{ret, call\} \times [n]) \cup \{int\}$, and an n-stack call-return alphabet is a pair $\widetilde{\Sigma}_n = (\Sigma, lab_{\Sigma,n})$. 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*) and $\Sigma_c^h = \{a \in \Sigma \mid lab_{\Sigma,n}(a) = (call, h)\}$ (*set of calls*). Moreover, $\Sigma_{int} = \{a \in \Sigma \mid lab_{\Sigma,n}(a) = int\}$ (*set of internals*), $\Sigma^h = \Sigma_c^h \cup \Sigma_r^h \cup \Sigma_{int}$, $\Sigma_c = \bigcup_{h=1}^n \Sigma_c^h$ and $\Sigma_r = \bigcup_{h=1}^n \Sigma_r^h$.

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.

[3] A proof via determinization was given in [7], 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 [11].

Definition 1. (Multi-stack visibly pushdown automaton) *A* multi-stack visibly pushdown automaton *(*Mvpa*) over* $\widetilde{\Sigma}_n$*, is a tuple* $A = (Q, Q_I, \Gamma, \delta, Q_F)$ *where* Q *is a finite set of states,* $Q_I \subseteq Q$ *is the set of initial states,* Γ *is a finite stack alphabet containing the symbol* $\bot$*,* $\delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\bot\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{int} \times Q)$ *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 $\alpha = \langle q, \sigma_1, \ldots, \sigma_n \rangle$, where $q \in Q$ and each $\sigma_h \in (\Gamma \setminus \{\bot\})^*.\{\bot\}$ is a *stack content*. Moreover, α is *initial* if $q \in Q_I$ and $\sigma_h = \bot$ 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] $a \in \Sigma_c^h$, $\exists \gamma \in \Gamma \setminus \{\bot\}$ 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 $\gamma \neq \bot$ and $\sigma_h = \gamma \cdot \sigma'_h$, or $\gamma = \sigma_h = \sigma'_h = \bot$.

[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 Σ^*, a *run* of A on w from α_0 to α_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 $w \in \Sigma^*$ is accepted by an Mvpa A if there is an initial configuration α and an accepting configuration α' such that $\alpha \xrightarrow{w}_A \alpha'$. The language accepted by A is denoted with $L(A)$.

A language $L \subseteq \Sigma^*$ is called a *multi-stack visibly pushdown language* (Mvpl) if there exist $n > 0$ and an n-stack call-return labeling $lab_{\Sigma,n}$ such that L is accepted by an Mvpa over $\widetilde{\Sigma}_n = (\Sigma, lab_{\Sigma,n})$.

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 ε 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 Υ, a Υ*-labeled tree* is a pair (T, λ) where T is a tree, and $\lambda : T \to \Upsilon$ is a labeling map.

Path-trees. For a tree T, a T-path π is a sequence that contains at least one occurrence of each node of T, and corresponds to a visit of T starting from the root and ending at a node that appears only once in π. Namely, a T*-path* 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 $d_i \in D$ (π corresponds to a traversal of T), (3) for $i \in [\ell - 1]$, $v_\ell \neq v_i$ (the last node occurs once in π), (4) π 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 π (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 π_1 or concatenating more occurrences, the resulting sequence would not satisfy one of the above properties.

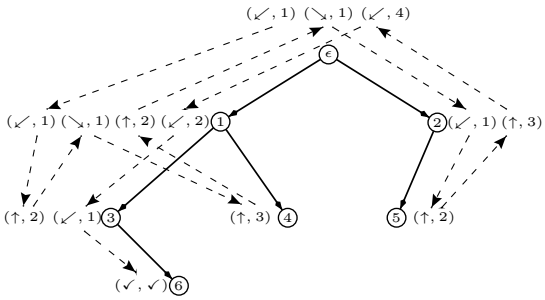

Fig. 1. A sample path-tree T_1.

We introduce the notion of *path-tree*, that is, a labeled tree (T, λ) 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 ending at $(\checkmark, \checkmark)$ by appending to a (d, i) labeling a node u, as the next pair in the chain, 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 λ in T. To ensure that the defined path is a T-path, we require some additional properties on λ which are detailed in the formal definition 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 π_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 $\rho[j]$ the pair (d_j, i_j), for $j \in [h]$.

Definition 2. *A $dir^+_{\checkmark}$-labeled tree (T, λ) is a* path-tree *if:*

1. *there is exactly one node labeled with $(\checkmark, \checkmark)$; and*

for every node v of T with $\lambda(v) = (d_1, i_1)(d_2, i_2) \dots (d_h, i_h)$, and $j \in [h]$, the following holds:

2. *if $i_j \neq \checkmark$ then $v.d_j$ is a node of T and $i_j \leq |\lambda(v.d_j)|$* (existence of the pointed pair)*;*
3. *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)*;*
4. *if $v \neq \varepsilon$ then there exists $i \in [|\lambda(v.\uparrow)|]$ s.t. $\lambda(v.\uparrow)[i] = (d, 1)$, $d \in \{\swarrow, \searrow\}$, and $v.\uparrow.d = v$* (except for the root the first pair in a label is always pointed from the parent)*;*
5. *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)*;*
6. *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 $(\swarrow, j+1)$ (resp. $(\searrow, j+1)$), if $d_j = \uparrow$ and v is a left (resp. right) child* (if a pair of u points to a pair β of an adjacent node v, the next pair of u is pointed from a pair β' that follows β in the v 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, λ) into a corresponding sequence of T nodes, and show that indeed $tp(T, \lambda)$ is a T-path. Let $\pi = v_1, \ldots, v_\ell$, and $d_1, \ldots, d_\ell$, and $i_1, \ldots, i_\ell$ 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 π. Also, we say that, in π, 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 lemmas hold (see Appendix for the proofs).

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.*

Lemma 2. *For any path-tree $\mathcal{T} = (T, \lambda)$, $tp(\mathcal{T})$ is a T-path.*

From T-paths to path-trees. We introduce a function pt that maps a T-path π into a corresponding path-tree (T, λ), and show that pt and tp are each the inverse function of the other.

For a T-path $\pi = v_1, \ldots, v_\ell$, we define the tree $pt(\pi)$ such that its labeling map defines exactly π. For this, we iteratively construct a sequence of labeling maps λ_i^π for $i \in [\ell]$, by concatenating a suitable pair at each iteration. Formally, denote $dir_\checkmark^* = dir^* \cup \{(\checkmark, \checkmark)\}$. For a T-path $\pi = v_1, v_2, \ldots, v_\ell$ and $i \in [\ell]$, let $\lambda_i^\pi : T \to dir_\checkmark^*$ 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 $v \in T \setminus \{v_\ell\}$.

We define $pt(\pi)$ as (T, λ_ℓ^π).

From the definitions we get (see Appendix for a proof):

Lemma 3. *For any T-path π 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, λ_{dir}) is a path-tree and λ_Σ maps each node of T to a symbol from $\widetilde{\Sigma}_n$. With $word_\mathcal{T}$ we denote the word $\lambda_\Sigma(v_1) \ldots \lambda_\Sigma(v_h)$ where $v_1 \ldots v_h$ 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 from a call-return alphabet 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, λ_{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 over $\widetilde{\Sigma}_n$ to a visibly path-tree $wt(w) = (T, (\lambda_{dir}, \lambda_\Sigma))$ as follows.

Let $w = a_1 \dots a_\ell$ over $\widetilde{\Sigma}_n$. The labeled tree (T, λ_Σ) 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. (Note that this corresponds to the notion of *stack tree* introduced in [12].)

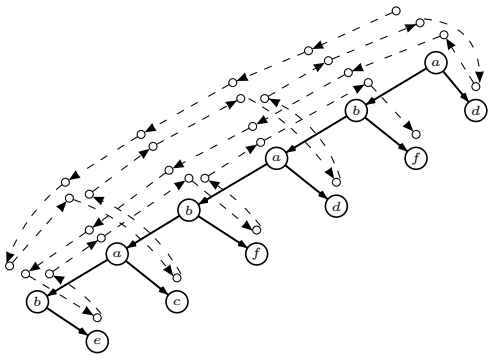

Fig. 2. The visibly path-tree $wt(w)$ for $w = (ab)^3\, cd^2\, ef^2$.

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 $i \in [2, \ell]$, π_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 π_w is a T-path. Thus, we define λ_{dir} to encode π_w, i.e., such that $tp(T, \lambda_{dir}) = \pi_w$.

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 exaple 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 $PTW_k(\widetilde{\Sigma}_n)$ 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 [17]).

Regularity of $PT_k(\widetilde{\Sigma}_n)$. Consider an input tree $\mathcal{T} = (T, (\lambda_{dir}, \lambda_\Sigma))$. We construct the tree automaton P_k accepting $PT_k(\widetilde{\Sigma}_n)$ as the intersection of two automata P and R, where P checks that (T, λ_{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. 2 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, λ_{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 size 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 (along $tp(\mathcal{T})$ nodes are visited starting from the root and then moving to neighbor nodes), 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. (See Appendix for more details on this construction.)

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 of size polynomial in k. However, 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 of size $O(k)$, and thus B_h also has size $O(k)$.

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, λ_{dir}) is a path-tree. Since complementation causes an exponential blow-up, 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:

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 4. *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.** With $\mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n)$ we denote a set of words parameterized over a possibly empty tuple of integer-valued parameters $\bar{p}$ and a call-return alphabet $\widetilde{\Sigma}_n$. For example, we can define $C_b(\widetilde{\Sigma}_n)$ as 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 $i \in [1, b]$ (*bounded context-switching* [16]). Also, denote with $\mathcal{C}$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 $L = L(A) \cap \mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n)$. With TMVPL, we denote such a class for $\mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n) = PTW_k(\widetilde{\Sigma}_n)$.

A class $\mathcal{C}$MVPL is *PT-covered* if for each $n > 0$ and $\bar{p}$, there exists a $k > 0$ such that $\mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n) \subseteq PTW_k(\widetilde{\Sigma}_n)$. We refer to such k as the *PT-parameter*. A class $\mathcal{C}$MVPL is *PT-definable* if it is PT-covered and there is an automaton $\mathcal{A}_{\mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n)}$ that accepts the set of all trees $wt(w)$ s.t. $w \in \mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n)$. 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 [11], we get:

Lemma 5. *The class of* MVPA*'s that captures* TMVPL *is not determinizable.*

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

Despite of Lemma 5, 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. the word $word_{\mathcal{T}}$ 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. In the following, we construct an MVPA $\bar{A}$ that accepts all words w such that $wt(w) \in L(\bar{\mathcal{B}}_k)$, that concludes the proof.

Constructing the MVPA $\bar{A}$*.* From $\bar{\mathcal{B}}_k$, we can construct an MVPA $\bar{A}$ that mimics $\bar{\mathcal{B}}_k$ transitions as follows (see Appendix for more details): on internal symbols, $\bar{A}$ moves exactly as $\bar{\mathcal{B}}_k$ (there is no right child); on call symbols, $\bar{A}$ enters the state that $\bar{\mathcal{B}}_k$ would enter on the left child and pushes onto a stack the one that $\bar{\mathcal{B}}_k$ would enter on the right child; on return symbols, $\bar{A}$ acts as if the current state is the one popped from the stack. (We recall that the stack is uniquely determined by the input symbol.) 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. In $\bar{A}$, popping the current state from the stack allows to restore properly the simulation of $\bar{\mathcal{B}}_k$ from a right child. Being the size of $\bar{A}$ polynomial in $|\bar{\mathcal{B}}_k|$ and exponential in n, we get (notice that since $\mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n) \subseteq PTW_k(\widetilde{\Sigma}_n)$, the words in $L(\bar{A})$ that are not in $\mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n)$ are ruled out by the intersection with this set):

Theorem 2. *Any PT-covered class* $\mathcal{C}$MVPL *is closed under complement. Also, for an* MVPA A*, there is an effectively constructible* MVPA $\bar{A}$ *s.t.* $L(\bar{A}) \cap \mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n) = \mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n) \setminus L(A)$*, 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 [6]. In fact, by the tree-decompositions given in [15] 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 $\mathcal{C}_{\text{MVPL}}$, we reduce the emptiness problem to the emptiness for tree automata by constructing a tree automaton given as intersection of $\mathcal{A}_{\mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n)}$, P_k and $\mathcal{A}_k$, where k is the PT-parameter.

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

The size of $\mathcal{A}_{\mathcal{C}_{\bar{p}}(\widetilde{\Sigma}_n)}$ 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, 15]), 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 DLT, LNCS vol. 5257, pp. 121–133. Springer (2008)
4. Bansal, K., Demri, S.: Model-checking bounded multi-pushdown systems. In CSR, LNCS vol. 7913, pp. 405–417. Springer (2013)
5. Bollig, B., Kuske, D., Mennicke, R.: The complexity of model checking multi-stack systems. In: LICS. pp. 163–172. IEEE Computer Society (2013)
6. 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)
7. Carotenuto, D., Murano, A., Peron, A.: 2-visibly pushdown automata. In DLT, LNCS vol. 4588, pp. 132–144. Springer (2007)
8. Cyriac, A., Gastin, P., Kumar, K.N.: MSO decidability of multi-pushdown systems via split-width. In CONCUR, LNCS vol. 7454, pp. 547–561. Springer (2012)
9. Esparza, J., Ganty, P., Majumdar, R.: A perfect model for bounded verification. In: LICS. pp. 285–294. IEEE (2012)
10. La Torre, S., Madhusudan, P., Parlato, G.: An infinite automaton characterization of double exponential time. In CSL. LNCS, vol. 5213, pp. 33–48. Springer (2008)
11. La Torre, S., Madhusudan, P., Parlato, G.: The language theory of bounded context-switching. In LATIN, LNCS vol. 6034, pp. 96–107. Springer (2010)
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., Napoli, M.: Reachability of multistack pushdown systems with scope-bounded matching relations. In CONCUR, LNCS vol. 6901, pp. 203–218. (2011)
14. Lal, A., Touili, T., Kidd, N., Reps, T.W.: Interprocedural analysis of concurrent programs under a context bound. In TACAS, LNCS vol. 4963, pp. 282–298. Springer (2008)
15. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In POPL pp. 283–294. ACM (2011)
16. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In TACAS, LNCS vol. 3440, pp. 93–107. Springer (2005)
17. Thomas, W.: Languages, automata, and logic. In Handbook of Formal Languages, Volume 3, pp. 389–455. Springer (1997)

A Proofs of Lemma 1 and Lemma 2

Fix a path-tree $\mathcal{T} = (T, \lambda)$. In this section, we assume the notation $v_1, \ldots, v_\ell$, and $d_1, \ldots, d_\ell$, and $i_1, \ldots, i_\ell$ as in the definition of $tp(\mathcal{T})$, that is, as 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 $\lambda(v_j)[i_{j-1}] = (d_j, i_j)$.

Denote with $labels(\mathcal{T})$ the sequence $(v_1, d_1, i_1), \ldots, (v_\ell, d_\ell, i_\ell)$. In the following lemma, we show that for each node v of $\mathcal{T}$ and for each $(d, i) \in \lambda_{dir}(v)$, there is exactly a $j \in [\ell]$ such that $(v, d, i) = (v_j, d_j, i_j)$, that is, along $tp(\mathcal{T})$, the pairs in the labels of $\mathcal{T}$ form a chain that starts from the first pair of the root, and ends at $(\checkmark, \checkmark)$, after visiting exactly once each pair in the labeling of each node of $\mathcal{T}$.

Lemma 6. *For any path-tree $\mathcal{T} = (T, \lambda)$, $v \in \mathcal{T}$ and $(d, i) \in \lambda_{dir}(v)$, by denoting $labels(\mathcal{T}) = (v_1, d_1, i_1), \ldots, (v_\ell, d_\ell, i_\ell)$, there is exactly a $j \in [\ell]$ such that $(v, d, i) = (v_j, d_j, i_j)$. Moreover, v_1 is the root of $\mathcal{T}$, (d_1, i_1) is the first pair of the root, and $(d_\ell, i_\ell) = (\checkmark, \checkmark)$.*

Proof. Let V be the set of all triples (v, d, i) where v is a node of T and (d, i) is a pair in the label $\lambda(v)$. Let $t = (v, d, i)$ and $t' = (v', d', i')$ be two triples in V. We define a successor predicate *succ* on the pairs (t, t') in which $succ(t, t')$ holds true if $v' = v.d$ and (d', i') is the i-th pair in $\lambda(v')$.

Denote $t_j = (v_j, d_j, i_j)$ for $j \in [\ell]$. Note that from the definition of $labels(\mathcal{T})$, we have that $succ(t_j, t_{j+1})$ holds for any $j \in [\ell - 1]$.

From properties 2 and 3 of Def. 2, every triple in V has a unique successor and a unique predecessor, except for (1) (ϵ, d, i) where (d, i) is the first pair in $\lambda(\epsilon)$ which does not have a predecessor, and (2) $(v, \checkmark, \checkmark)$ where v is the unique node of T (the existence of such a node is assured by property 1 of Def. 2) with $\lambda(v) = (\checkmark, \checkmark)$ which does not have a successor.

Therefore, all the triples in V form one chain, and possibly one or more disjoint loops. Now, denote with $t'_1, \ldots, t'_N$ any sequence of triples such that $succ(t'_j, t'_{j+1}$ for $j \in [N - 1]$. Denote $t'_j = (v'_j, d'_j, i'_j)$ for $j \in [N]$. By property 6 of Def. 2, for each selection of two triples t'_r and t'_s s.t. $r < s$, $v'_r = v'_s = v$, and $v'_j \neq v$ for $j \in [r + 1, s - 1]$, we get that $i'_s = i'_r + 1$. Therefore none of the triples can repeat in any such sequence, and thus all the triples of V form exactly one chain that is $labels(\mathcal{T})$ and the lemma is shown. □

We prove first Lemma 1.

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.*

Proof. Denote with $labels(\mathcal{T})_j$ the sequence $(v_1, d_1, i_1), \ldots, (v_j, d_j, i_j)$ for $j \leq \ell$. We show by induction on j, that for all $h \in [1, j]$ and $i \geq 1$, if $v_h = u$ is the i-th occurrence of u in $labels(\mathcal{T})_j$, then either $h = 1$ and $i = 1$, or $h > 1$ and $i_{h-1} = i$. This clearly implies the lemma.

The base case $j = 1$ is trivial. In fact, $labels(\mathcal{T})_1 = (v_1, d_1, i_1)$, and since v_1 is the first occurrence of the root, then the property clearly holds.

Now assume by induction that the property holds for $1 \leq j < \ell$. Thus, for $h \in [1, j]$ and $i \geq 1$, if $v_h = u$ is the i-th occurrence of u in $labels(\mathcal{T})_j$, then either $h = 1$ and $i = 1$, or $h > 1$ and $i_{h-1} = i$. Therefore, we need to show this property only for $h = j + 1$, in order to prove the induction step.

Suppose that v_{j+1} is an occurrence of a node v of $\mathcal{T}$, and that v_r is the last occurrence of such v in $labels(\mathcal{T})_j$. By induction we know that v_r is the i_{r-1}-th occurrence of v, thus v_{j+1} is the $(i_{r-1} + 1)$-th such occurrence. From Lemma 6, we get that all the pairs in the labeling of $\mathcal{T}$ are used to construct $tp(\mathcal{T})$, and thus $labels(\mathcal{T})$. Moreover, as shown in the proof of Lemma 6, by property (6) of Def. 2, all the pairs in the labeling of any node must be visited by increasing indexes. Therefore, i_j must be $(i_{r-1} + 1)$, and thus the property holds also for $h = j + 1$. □

Now, we can show Lemma 2.

Lemma 2. *For any path-tree $\mathcal{T} = (T, \lambda)$, $tp(\mathcal{T})$ is a T-path.*

Proof. Denote $tp(\mathcal{T}) = v_1, v_2, \ldots, v_\ell$. Directly from the definition of $tp(\mathcal{T})$ we get that v_1 is the root and v_{j+1} is adjacent to v_j (i.e., it is either the parent or a child of v_j) for $j \in [\ell - 1]$. Thus, properties (1) and (2) of the definition of T-path hold. Directly from Lemma 6, we get that the last node of $tp(\mathcal{T})$ is labeled with only the pair $(\checkmark, \checkmark)$ and hence property (3) of the T-path definition holds. Since each node of $\mathcal{T}$ has at least a pair labeling it, again from Lemma 6 we get that $tp(\mathcal{T})$ contains at least one occurrence of each node in T, that entails property (4) of the T-path definition. By Lemma 1 and property (5) of Def. 2, we get that also property (5) of the T-path definition holds for $tp(\mathcal{T})$, that ends the proof. □

B Proof of Lemma 3

Before proving Lemma 3, we show the following property of path-trees.

Lemma 7. *For any two distinct path-trees $\mathcal{T}_1 = (T, \lambda_1)$ and $\mathcal{T}_2 = (T, \lambda_2)$, $tp(\mathcal{T}_1) \neq tp(\mathcal{T}_2)$.*

Proof. Let V_1 and V_2 be the sets of triples as defined in the proof of Lemma 2 for $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. Similarly, we define the successor relations $succ_1$ and $succ_2$ fir V_1 and V_2. We now prove that if λ_1 and λ_2 are different, it must be the case that $\pi_1 = tp(\mathcal{T}_1) \neq tp(\mathcal{T}_2) = \pi_2$.

Let $t_1^1 t_2^1 \ldots t_{\ell_1}^1$ be the sequence of all triples in V_1 such that $succ_1(t_j^1, t_{j+1}^1)$ holds, for any $j \in [\ell_1 - 1]$. Similarly, we define $t_1^2 t_2^2 \ldots t_{\ell_2}^2$ for the set V_2. If $\lambda_1 \neq \lambda_2$ then either $\ell_1 \neq \ell_2$, hence $\pi_1 \neq \pi_2$, or $\ell_1 = \ell_2 = \ell$ and the sequences $t_1^1 t_2^1 \ldots t_\ell^1$ and $t_1^2 t_2^2 \ldots t_\ell^2$ are different. Let $\ell_1 = \ell_2$ and j be the least index in which the two sequences differ, with $t_j^1 = (v_j^1, d_j^1, i_j^1)$ and $t_j^2 = (v_j^2, d_j^2, i_j^2)$. Note that, v_j^1 and v_j^2 must necessarily be the same otherwise j would not be the least index. Instead, d_j^1 and d_j^2 must necessarily be distinct. In fact, if $d_j^1 = d_j^2$ it must be the case that $i_j^1 = i_j^2$. Thus, we have that $v_{j+1}^1 \neq v_{j+1}^2$ which make π_1 different from π_2 at position $j + 1$. □

Lemma 3. *For any T-path π and path-tree $\mathcal{T}$, $tp(pt(\pi)) = \pi$ and $pt(tp(\mathcal{T})) = \mathcal{T}$.*

Proof. We first show that if π is a T-path then $tp(pt(\pi)) = \pi$. Let $\pi = v_1, v_2, \ldots, v_\ell$ and $\pi_j = v_1, v_2, \ldots, v_j$, for every $j \in [\ell]$. We prove by induction on $j \in [\ell]$ that $tp(pt(\pi_j)) = \pi_j$ where $v_1, \ldots, v_j$, $d_1, \ldots, d_j$ and $i_1, \ldots, i_j$ are the witnessing sequences of $tp(pt(\pi_j))$.

The case for $i = 1$ is straightforward. Consider $i \in [2, \ell]$. The labelling map λ^π_j defining $pt(\pi_j)$ is obtained from λ^π_{i-1} leaving unchanged the labels of all nodes but v_j which gets the label $\lambda^\pi_j(v_j) = \lambda^\pi_{j-1}(v_j).(d_j, i_j)$ where $v_{j+1} = v_j.d_j$. Since the concatenated pair of $\lambda^\pi_j(v_j)$ is at position i_{j-1}, and $v_1, \ldots, v_{j-1}$, $d_1, \ldots, d_{j-1}$ and $i_1, \ldots, i_{j-1}$ are the witnessing sequences of $tp(pt(\pi_{j-1}))$ (by inductive hypothesis), we can straightforwardly derive that $v_1, \ldots, v_j$, $d_1, \ldots, d_j$ and $i_1, \ldots, i_j$ are the only maximal sequences defining $tp(pt(\pi_j))$.

We conclude the proof of the first statement by noticing that $tp(pt(\pi)) = tp(pt(\pi_\ell)) = \pi_\ell = \pi$.

Let $\mathcal{T}$ be a path-tree. We now prove that $pt(tp(\mathcal{T})) = \mathcal{T}$. From Lemma 2, we know that $tp(\mathcal{T})$ is a unique T-path, say π. Since $tp(pt(\pi)) = \pi$ we have that $tp(pt(tp(\mathcal{T}))) = tp(\mathcal{T})$. From Lemma 7, we can conclude that $pt(tp(\mathcal{T})) = tp(\mathcal{T})$. □

C Checking $u < v$: more details on the tree automaton S

The tree automaton S has $3 + 2\,k$ states:

- an initial state that is also used to store that w has not being guessed yet,
- an acceptance state and a rejection state,
- states of the form (i, x) where $i \in [1, k]$ and $x \in \{u, v\}$, that are used to store at a node z the request of visiting the node x starting from the i-th pair of z and before returning to the $(i + 1)$-th pair of z.

While in the initial state, the automaton S nondeterministically guesses a path to a node w. Thus, at each node z, nondeterministically it identifies z as the node w, and propagates the search to a child of z.

In the first case, the tree automaton S nondeterministically guesses the node w, its child w_u whose subtree contains u and its child w_v whose subtree contains v. Thus, denoting $\lambda_{dir}(z) = (d_1, i_1) \ldots (d_\ell, i_\ell)$, it also 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). Then, this guess is passed onto the two children by entering the states (i_r, u) on the child w_u and (i_s, v) on the child w_v.

In the second case, the initial state is passed onto one of the children of z and the acceptance state is passed on the other (if any). If z has no children, then the rejection state is entered.

The rejection and the acceptance states are sink states (i.e., once the automaton enters on of them, it never leaves it)

From a state (i, x), $x \in \{u, v\}$, if the current node z is x and $i = 1$ then S enters the acceptance state, otherwise it propagates the request onto one of the children of z, if any, and enters the rejection state otherwise. Namely, denoting $\lambda_{dir}(w) = (d_1, i_1) \dots (d_\ell, i_\ell)$, a pair (d_r, i_r) with $r \in [i, s-1]$ is guessed where $z.d_r$ is a child of z and s is the smallest $j > i$ such that $d_j = \uparrow$ (the s-th pair points back to its parent); then the state (i_r, x) is passed onto $z.d_r$. If z has another child, then the acceptance state is entered on it.

The correctness of this automaton strongly relies on Lemma 1. In fact, we use the nondeterminism to select the least common ancestor of u and v, and then converge to $x \in \{u, v\}$ by shortcutting in each visited node w the portions of $tp(\mathcal{T})$ that return on w before visiting x. Note that when the request is propagated to a child, we do not allow to select any index past the first pair that points back to the parent. This ensures that, when accepted, the first occurrence of u occurs before the first occurrence of v.

D Construction of the automaton $\bar{A}$

In this section we give a central lemma that allows to prove that bounded path-tree MVPAs are closed under complement.

We first introduce some definitions. A (top-down) *tree-automaton* on Υ-labeled trees is a tuple $\mathcal{A} = (P, P_I, \Delta)$ where P is a finite set of states, $P_I \subseteq P$ is the set of initial states, and $\Delta = \langle \Delta_{\{\swarrow,\searrow\}}, \Delta_{\{\swarrow\}}, \Delta_{\{\searrow\}}, \Delta_\emptyset \rangle$ is a set of four transition relations, with:

- $\Delta_{\{\swarrow,\searrow\}} \subseteq P \times \Upsilon \times P \times P$;
- for $d \in \{\swarrow, \searrow\}$, $\Delta_{\{d\}} \subseteq P \times \Upsilon \times P$;
- $\Delta_\emptyset \subseteq P \times \Upsilon$.

A *run* of $\mathcal{A}$ over a Υ-labeled tree (T, λ) is a P-labeled tree (T', λ') where $\lambda'(\epsilon) \in P_I$, and for every node $v \in T$:

- if v has both children, then $(\lambda'(v), \lambda(v), \lambda'(v.\swarrow), \lambda'(v.\searrow)) \in \Delta_{\{\swarrow,\searrow\}}$;
- if v has only the d-child, with $d \in \{\swarrow, \searrow\}$, then $(\lambda'(v), \lambda(v), \lambda'(v.d)) \in \Delta_{\{d\}}$;
- if v is a leaf, then $(\lambda'(v), \lambda(v)) \in \Delta_\emptyset$.

Tree automata are usually defined using a set of final states; this has been absorbed into the $\Delta_\emptyset$ component of the transition relation. A labelled tree (T, λ) is accepted by a tree automaton $\mathcal{A}$ iff there exists a run of $\mathcal{A}$ over T. The set of trees accepted by $\mathcal{A}$ is the language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$.

Now we prove the main result of this section. The lemma below completes the proof of Theorem 2.

Lemma 8. *For any tree automaton $\mathcal{A}$ over Υ-labeled trees with $\mathcal{L}(\mathcal{A}) \subseteq PT_k(\widetilde{\Sigma}_n)$, there is an effectively constructible* MVPA *A over $\widetilde{\Sigma}_n$ such that $L(A)$ is the set of all the k-path-tree words $word_\mathcal{T}$ such that $\mathcal{T} \in \mathcal{L}(\mathcal{A})$. Moreover, the size of A is polynomial in the size of $\mathcal{A}$ and exponential in n.*

Proof. Let $\mathcal{A} = (P, P_I, \Delta)$ and $\Delta = \langle \Delta_{\{\swarrow,\searrow\}}, \Delta_{\{\swarrow\}}, \Delta_{\{\searrow\}}, \Delta_{\emptyset} \rangle$.

The main idea of the construction is the following. While A reads a word w it mimics a run of $\mathcal{A}$ on $wt(w)$, and w is accepted iff $wt(w)$ is accepted by $\mathcal{A}$. We recall that there is a 1-to-1 map between the positions in w and the nodes of $wt(w)$. Thus, reading w all nodes of $wt(w)$ are visited exactly once. From the definition of wt, it is easy to see that every node is discovered only after its parent has already been visited.

The MVPA A is defined such that the following invariant is maintained during the simulation. For each unmatched call symbol that has been read so far there is a symbol in the appropriate stack. This property derives from the visibility of the alphabet. On the other hand, each element in the stacks corresponds to a distinct unmatched position in w in the part of w that has been read so far. For each of these unmatched positions, say i, and denoting with u the right child of the node of $wt(w)$ corresponding to position i, the symbol stored in the stack is either (1) the state assigned to u by the tree automaton $\mathcal{A}$, or (2) a special symbol $*$ in case u does not exist. Furthermore, if v is the left child of the node associated with the last read position in w, then A stores in its control the $\mathcal{A}$ state q_v assigned to v, otherwise it stores the special symbol $*$ in its control meaning that v does not exist.

We now define the moves of A, and we show by induction on the length of w (as we go defining them) that the above property is maintained.

The initial state of A has an initial state of P_I stored in its control which corresponds to the state associated to the root of $wt(w)$. Let σ be the first unread symbol, and v be the node of $wt(w)$ associated to this occurrence of σ. We distinguish the following cases:

Internal: If $\sigma \in \Sigma_{int}$, then the state associated with v is stored in the control of A (by inductive hypothesis). Now, A guesses the label of v and whether v has a left child or not. Notice that, because $\sigma \in \Sigma_{int}$, the node v cannot have a right child (by definition of wt). If v does not have a left child, A will pick a move from $\Delta_{\emptyset}$ to be simulated on v. If an $\mathcal{A}$ move exists, A will store $*$ in its control. Instead, if v has a left child, it will nondeterministically pick a move from $\Delta_{\{\swarrow\}}$ that will be simulated at v and stores in its control the state assigned to the left child of v. It is clear that in this case the above property is maintained.

Call: If $\sigma \in \Sigma_c^i$, similarly to the previous case, the state associated to v is stored in the control of A. Now, A guesses the label of v and whether v has or not a left and a right child, respectively. Based on this, A picks nondeterministically a move from Δ. If v has a left child, the state associated with it through the tree automaton move will be stored in the control of A, otherwise it stores $*$. If v has a right child the state associated with it will be stored on the top of stack i, otherwise $*$ is pushed onto stack i. Thus, also in this case the invariant is maintained.

Return: If $\sigma \in \Sigma_r^i$, the node associated with the position right before the current read position will correspond to a node that does not have a left child, and thus from the invariant, $*$ is stored in the control state of A. Now,

A recovers the state associated to v by popping the state stored on the top of stack i, and it will proceed in the same way it handles an internal symbol of the alphabet. Again this shows that this maintains the invariant.

A accepts a k-tpath-tree word if all stored elements in the stacks are $*$ symbols and also the symbol maintained in the control is a $*$. This reflects the fact that no more nodes in $wt(w)$ exists and all those nodes have been correctly labeled by the run and all leaves are accepting. To implement this mechanism A will maintain in its control also a tuple of bits, one for each stack, to remember whether each stack has still an $\mathcal{A}$ state in its content. To update those bits correctly it will also store in each position of the stack an additional bit that tells whether below that position in the stack there is an $\mathcal{A}$ state stored. It easy to see that this information can be easily maintained during the execution of A moves.

The size of A is thus polynomial in $|\mathcal{A}| \cdot 2^n$. □