On Multi-stack Visibly Pushdown Languages

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 contribute to the theory of formal languages of visibly multistack pushdown automata (MVPA). First, we show closure under the main operations and decidability of the main decision problems for the class of MVPA restricted to computations where a symbol can be popped from a stack S only if it was pushed within the last k contexts of S, for a given k (in each context only one stack can be pushed or popped). In particular, this class turns out to be determinizable. Second, we show the closure under complement of the class of languages accepted by ordered MVPA, where the limitation is that a stack can be popped only if all the lower indexed stacks are empty. This also gains the decidability of universality, inclusion and equivalence. As a further contribution, we compare the classes of languages accepted by different models of MVPA.

1 Introduction

Pushdown automata working with multiple stacks (*multistack pushdown automata*, MPA for short) are a natural model of concurrent programs. As an acceptor of formal languages, they are equivalent to Turing machines already with only two stacks: the input word is entirely parsed and pushed onto one stack, then by ε-moves (i.e., moves that do not consume input) any Turing machine computation can be simulated by maintaining the tape into the stacks.

A way to limit the power of MPA is to make the stack operations *visible* in the input symbols. A *multistack visibly pushdown automaton* (MVPA) has an input alphabet that is partitioned into n sets of call alphabets (Σ_c^i) and return alphabets (Σ_r^i), where n is the number of stacks, and an internal alphabet (Σ_{int}), and the automaton pushes or pops from stack i only when it reads a call from Σ_c^i or a return in Σ_r^i, respectively.

Visibility of stack operations was first introduced for standard pushdown automata [2] and on MPAs has two main consequences. First, when reading a same input the use of the stacks of any MVPA is synchronized and thus closure under intersection can be shown with the usual cross product construction (recall that standard pushdown automata are not closed under intersection). Second, the stack height is bounded by the length of the input word (ε-moves are not allowed), and thus any language accepted by an MVPA is context-sensitive (see [10]). However, main decision problems such as emptiness are still undecidable: an MVPA can accept the encodings of the runs of a given MPA.

The quest for a naturally-defined notion of a robust class of MVPA has lead to consider some restrictions on the allowed behaviors. In [13], the admissible runs are restricted to those formed by a bounded number of *phases*, where in

each phase, symbols can be pushed into all stacks but popped only from one. The corresponding class of languages is closed under all Boolean operations and has many of the properties of the class of regular languages, such as, decidability of emptiness, membership, universality, inclusion, equality, a logical characterization and a Parikh theorem. In [12], the model is restricted to runs with a bounded number of *contexts*, where in each context, only one stack can be used. The notion of context is more restrictive than phases (each context is contained into a phase and a phase can be composed of unboundedly many contexts), and thus the resulting class of languages is a subclass of the previous one. All the above results hold also for this class, further the decision problems are computationally simpler (for example, emptiness is NP-complete [19, 17] instead of 2ETIME-complete [13, 11]) and the model is determinizable, i.e., the deterministic and nondeterministic versions characterize the same class of languages. Another decidable restriction of MPA is *ordered* MPA [8], where symbols can be popped from stack i only if all the stacks from 1 to $i-1$ are empty. Visibly 2-stack MPA are studied in [6, 9].

In this paper, we contribute to the formal language theory of MVPAs in several ways. First, we study the formal languages theory of MVPA restricted to *scope-bounded computations* [14, 15]: a computation is k-scoped if for each stack i, each popped symbol was pushed within the last k contexts of i. For this class, closure under union and intersection can be shown with standard constructions, and being the reachability problem PSPACE-COMPLETE [14], we immediately get that also emptiness is PSPACE-COMPLETE. Decidability of membership is straightforward: guess a run over the input word and then check it. Our main result here is the determinization of scope-bounded MVPA. The construction is based on *switching masks* of bounded size, that are formed of *switching vectors*, one for each stack, and summarize the states of the MVPA at context-switches by considering for each stack only the top portion (i.e., the symbols that can still be popped according to the considered restriction). The resulting deterministic MVPA has size doubly exponential in both the number of stacks and the bound k. Determinization gains the closure under complement and thus, by the closure under intersection and the decidability of emptiness, we get the decidability of universality, inclusion and equality. Being this restriction more permissive than bounding the number of contexts (unboundedly many contexts are possible, even between two contexts of the same stack), to the best of our knowledge this is the largest class of formal languages with all the above properties.

Another main contribution of this paper is the proof of the closure under complement for the class of languages accepted by ordered MVPA. This result was shown via determinization in [9], however that is wrong since ordered MVPAs cannot be determinized. In fact, the language of all the words $(ab)^i c^j d^{i-j} x^j y^{i-j}$ is inherently nondeterministic for MVPAs [12] and can be accepted by a simple ordered MVPA with two stacks. In our proof, we first translate an ordered MVPA to an equivalent tree automaton, then complement it, and finally translate the resulting tree automaton back to an ordered MVPA. The overall scheme is the same used in [13] for phase-bounded MVPA, and as there, the base of our trans-

lation are the *stack trees* (where the right-child relation captures the matching calls and returns and if any, the left child gives the linear successor). The analogy with [13] ends here and note that the tree translation given there does not apply to ordered MPA. In contrast, we give a simpler and more general tree translation that applies also to classes of languages wider than ordered MVPAs. Namely, languages that can be mapped to classes of graphs admitting a tree decomposition of bounded tree-width and based on stack trees. The main difference with [13] is that we augment the stack trees (*path-trees*) with an additional labeling that plays a crucial role in reconstructing the linear order of each word. The main technical challenge is to characterize the resulting class of trees and show that can be accepted by a tree automaton. We observe that we do not need to characterize precisely in the tree automaton the restriction imposed on the MVPA, it is instead sufficient to show a bound on the size of the labels of the resulting trees. Thus, we exploit the fact that the multiply nested words corresponding to the language accepted by an ordered MVPA have bounded tree-width and the tree decomposition can be given with path-trees [18]. We further observe that this also holds for phase-bounded MVPAs and thus our approach can be used to show complementation also for this class of languages. As a consequence of the closure under complement, we get the decidability of universality, inclusion and equivalence. We complete the theory of ordered MVPA by addressing the complexity of the membership problem.

As a further contribution, we compare the expressiveness of the introduced classes of MVPAs. We show that ordered, phase-bounded and scope-bounded MVPAs define classes of languages that are pairwise incomparable; the class of the phase-bounded (resp. scope-bounded) languages strictly includes that of the context-bounded languages, while the ordered and the context-bounded languages are incomparable. We further observe that by the path-tree characterization we can show a Parikh theorem for both scope-bounded and ordered MVPA (we omit this here).

The notion of bounded context-switching was introduced in [19] to define an under-approximation method for model-checking multi-threaded programs. Scope-bounded matching relations were introduced to extend bounded-context switching to capturing unboundedly many context switches [14]. This notion naturally extends to infinite words and temporal logic model checking [15, 4]. The notion of scope-bounded matching relations used in this paper is that introduced in [15] that allows unboundedly many context-switches also between to consecutive contexts of a same stack. More work on decision problems is done in [20, 7] for phase-bounded MPA and ordered MPA [5].

2 Preliminaries

For $i, j \in \mathbb{N}$, we denote with $[i, j] = \{d \in \mathbb{N} \mid i \leq d \leq j\}$, and with $[j] = [1, j]$.
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*. Note that, by the definition it is not possible to have both unmatched calls and unmatched returns in the same word.

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 the stack when computation starts.

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 $\perp$, $\delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\perp\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{int} \times Q)$ 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, α 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* $\langle q, \sigma_1, \dots, \sigma_n \rangle \xrightarrow{a}_A \langle q', \sigma_1', \dots, \sigma_n' \rangle$ 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 $w = a_1 \dots a_m$ 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 $\widetilde{\Sigma}_n = (\Sigma, lab_{\Sigma,n})$.

A visibly pushdown automaton (VPA) [2] is an MVPA with just one stack. A *visibly pushdown language* (VPL) is an MVPL accepted by a VPA.

Bounded number of contexts/rounds. A *round* over $\widetilde{\Sigma}_n$ is a word of the form $w_1 w_2 \dots w_n$ where w_h is a stack-h context, for each $h \in [n]$. A *k-round word* over $\widetilde{\Sigma}_n$ is a word that can be obtained as the concatenation of k rounds. We denote with $\mathit{Round}(\widetilde{\Sigma}_n, k)$ the set of all the k-round words over $\widetilde{\Sigma}_n$. The notion of bounded number of rounds is strictly related to that of bunded number of contexts: each k-round word is indeed the concatenation of at most nk contexts, and a word which is the concatenation of k contexts is a k'-round word for some $k' \leq k$ (empty contexts can be used to complete rounds).

Bounded number of phases. A *phase* over $\widetilde{\Sigma}_n$ is a word in $(\Sigma_c \cup \Sigma_{int} \cup \Sigma_r^h)^*$, for a given $h \in [n]$. For an integer k, a *k-phase word* over $\widetilde{\Sigma}_n$ is a word that can be obtained as the concatenation of k phases. We denote with $\mathit{Phase}(\widetilde{\Sigma}_n, k)$ the set of all the k-phase words over $\widetilde{\Sigma}_n$.

Scope-bounded matching relations. A word $w = a_1 \dots a_m \in \Sigma^*$ is *k-scoped* over $\widetilde{\Sigma}_n$ if for each $i, j \in [m]$ such that $i \sim_h j$, for some $h \in [n]$, there are at most $2k-3$ indexes $x_1, \dots, x_{k-1}, y_1, \dots, y_{k-2} \in [i, j]$ such that $x_1 < y_1 < \dots < x_{k-2} < y_{k-2} < x_{k-1}$ and $a_x \in \bigcup_{h' \neq h} (\Sigma_c^{h'} \cup \Sigma_r^{h'})$, for each $x \in \{x_1, \dots, x_{k-1}\}$, $a_y \in \Sigma_c^h \cup \Sigma_r^h$ for each $y \in \{y_1, \dots, y_{k-2}\}$, With $\mathit{Scoped}(\widetilde{\Sigma}_n, k)$, we denote the set of all the k-scoped words over $\widetilde{\Sigma}_n$.

Ordered matching relations. A word $w = a_1 \dots a_m \in \Sigma^*$ is *ordered* over $\widetilde{\Sigma}_n$ if for each $h \in [n]$ and for each $i, j \in [m]$ such that $i \sim_h j$, it holds that for each $x < j$ such that $a_x \in \Sigma_c^{h'}$, with $h' < h$, there exists $y < j$ such that $x \sim_{h'} y$ holds (all calls of lower-index stacks preceding j are already matched at j). With $\mathit{Ordered}(\widetilde{\Sigma}_n)$, we denote the set of all the ordered words over $\widetilde{\Sigma}_n$.

Classes of languages. A language $L \subseteq \Sigma^*$ is a *bounded-round* MVPL (RMVPL) if there exist $k, n > 0$, an n-stack call-return labeling $\mathit{lab}_{\Sigma,n}$, and an MVPA A over $\widetilde{\Sigma}_n = (\Sigma, \mathit{lab}_{\Sigma,n})$ such that $L = \mathit{Round}(\widetilde{\Sigma}_n, k) \cap L(A)$. Analogously, we define *bounded-phase* MVPL (PMVPL), *scope-bounded* MVPL (SMVPL) and *ordered* MVPL (OMVPL).

3 Determinization of MVPA over scoped words

Here, we show that, when restricting to k-scoped words, deterministic and nondeterministic MVPAs are equivalent. Fix an integer $k > 0$ and an MVPA $A = (Q, Q_0, \Gamma, \delta, F)$ over $\widetilde{\Sigma}_n$.

Scope-bounded switching-vector VPA. Let $h \in [n]$ and $k > 0$. We define a VPA A_k^h over the alphabet $\Sigma^h \cup \{\flat, \sharp\}$ where the calls are $\Sigma_c^h \cup \{\sharp\}$, the returns are Σ_r^h, and the internals are $\Sigma_{int}^h \cup \{\flat\}$, with $\flat, \sharp \notin \Sigma$. In each run, A_k^h collects in its state a list of at most k pairs of states from Q (*k-switching vector* [12]): it starts from a list containing only a pair (q, q) and then updates the second component according to the transitions of A on the symbols from Σ^h. On $\flat$, A_k^h nondeterministically appends a new pair (q, q), for any $q \in Q$, to the current switching vector, if this has less than k pairs, and halts otherwise. On $\sharp$, A_k^h starts a new switching vector and pushes onto the stack a new symbol $\triangledown$. The

symbol $\triangledown$ is used to denote the bottom of the (top) stack portion that is actually generated by A_k^h to collect the new switching vector, and thus is never popped from the stack. The set of A_k^h states is $S = \bigcup_{m\in[k]}(Q \times Q)^m$, each state is also a final state and the set of initial states is $S_0 = \{(q, q)|q \in Q\}$. We omit the formal definition of the transitions. Denoting with L the language of all words of the form $w_1 \flat w_2 \dots \flat w_r$ with $r \leq k$ and $w_i \in (\Sigma^h)^*$ for $i \in [r]$, we observe that the language accepted by A_k^h is contained in $(L.\{\sharp\})^*.L$.

Fix a word z over the alphabet $\Sigma^h \cup \{\flat, \sharp\}$. With $\mathcal{I}_k^h(z)$, we denote the set of *the switching vectors* $I \in \bigcup_{r>0}(Q \times Q)^r$ s.t. there exists a run ρ of A_k^h on z and I is the concatenation of $\ell_1, \dots, \ell_d, \ell_{d+1}$ where: ℓ_{d+1} is the state of the last configuration of ρ and $\ell_1, \dots, \ell_d$ is the ordered sequence of states occurring at the configurations of ρ from which a transition on $\sharp$ is taken (in the order they appear in ρ). Note that for a switching vector $I \in \mathcal{I}_k^h(z)$, the number of pairs is the number of occurrences of $\flat$ and $\sharp$ in z plus one. For $I = (in_1, out_1) \dots (in_m, out_m)$, we denote I also with $\langle (in_i, out_i)\rangle_{i\in[m]}$, and $st(I) = in_1$ and $cur(I) = out_m$.

Switching masks. For a word w over $\tilde{\Sigma}_n$, we denote with $\langle w_i\rangle_{i\in[r]}$ a splitting of w into contexts s.t. $w = w_1 \dots w_r$ and, for each each $i \in [r]$, w_i is a stack-h_i context, for $i \in [1, r-1]$, $h_i \neq h_{i+1}$ and for $i \in [2, r]$, w_i starts with a symbol in $\Sigma_c^{h_i} \cup \Sigma_r^{h_i}$. Note that for each word w such a splitting is unique.

Fix the splitting $\langle w_i\rangle_{i\in[r]}$ of w, where each w_i is a stack-h_i context. With $cont_h(w) = (w_1^h, \dots, w_{r_h}^h)$, for $h \in [n]$, we denote the ordered sequence of all the stack-h contexts in $\langle w_i\rangle_{i\in[r]}$, i.e., there exist $i_1, \dots, i_{r_h}$ such that $1 \leq i_1 \leq \dots \leq i_{r_h} \leq r$, $w_j^h = w_{i_j}$ for $j \in [r_h]$, $h_{i_1} = \dots = h_{i_{r_j}} = h$, and $h_i \neq h$, for every $i \notin \{i_1, \dots, i_{r_h}\}$. Moreover, for each d and s such that $w_s^d = w_i$ for some $i \in [r]$, we denote with $next(d, s)$ the pair (d', s') such that $w_{s'}^{d'} = w_{i+1}$ (i.e., for the contexts in the tuples $cont_h(w)$, $h \in [n]$, *next* captures the order in which they appear in $\langle w_i\rangle_{i\in[r]}$).

A tuple $M = (I_1, \dots, I_n)$ is a *k-scoped switching mask* for w if: (1) for each $h \in [n]$, $I_h \in \mathcal{I}_h^k(w_1^h \natural_1 w_2^h \dots \natural_{r_h-1} w_{r_h}^h)$ where each $\natural_i \in \{\flat, \sharp\}$ and $cont_h(w) = (w_1^h, w_2^h, \dots, w_{r_h}^h)$, and (2) denoting $I_h = \langle in_s^h, out_s^h\rangle_{s\in[r_h]}$, $out_s^h = in_{s'}^{h'}$ for each h, s, h', s' such that $next(h, s) = (h', s')$. Moreover, we let $st(M) = st(I_{h_1})$ and $cur(M) = cur(I_{h_r})$ (recall that each w_i is a stack-h_i context). Thus:

Lemma 1. *Let $A = (Q, Q_0, \Gamma, \delta, F)$ be an* MVPA *over $\tilde{\Sigma}_n$ and $w \in Scoped(\tilde{\Sigma}_n, k)$. It holds that: $w \in L(A)$ if and only if there exists a k-scoped switching mask M for w such that $st(M) \in Q_0$ and $cur(M) \in F$.*

We recall that a switching vector from $\mathcal{I}_h^k(z)$ is called a thread interface in [16], where it is also shown that, for k-scoped words, such switching vectors can be obtained by concatenating thread interfaces formed of at most k pairs, which is exactly what is captured by the definitions of A_k^h and $\mathcal{I}_k^h(w)$. Observe that this property does not hold in general for thread interfaces of arbitrary MVPAs.

Determinization. For an MVPA A, we define a deterministic MVPA A^D that, for a k-scoped input word w, constructs the set of all switching masks according to any splitting of w into contexts. Thus, A^D accepts w iff it constructs a switching mask as in Lemma 1, and by supposing $w \in Scoped(\tilde{\Sigma}_n, k)$, iff $w \in L(A)$.

For $h \in [n]$, let $D_k^h = (S_D, S_{D,0}, \Gamma_D^h, \delta_D^h, F_D)$ be the deterministic VPA equivalent to $A_k^h = (S, S_0, \Gamma \cup \{\nabla\}, \delta^h, S)$ and obtained through the construction given in [2]. We recall that, according to that construction, the set of states S_D is $2^{S \times S} \times 2^S$, and the second component of a state is updated in a run as in the standard subset construction for finite automata. For a state $\hat{q} \in S_D$, denote with $R(\hat{q}) \subseteq S$ its second component.

We construct $A^D = (Q^D, Q_0^D, \Gamma, \delta^D, F^D)$ building on the cross product of $D_k^1, \ldots, D_k^n$; a state of A^D is $(h, \hat{q_1}, \ldots, \hat{q_n}, \mathcal{M})$, where $h > 0$ denotes the stack that is active in the current context, $h = 0$ denotes the initial state, $\hat{q_h}$ is a state of D_k^h, and $\mathcal{M}$ is a set of tuples $(I_1, \ldots I_n)$ where for $h \in [n]$, I_h is a state of A_k^h belonging to $R(\hat{q_h})$. The idea is to accumulate in the $\mathcal{M}$ component the tuples corresponding to the current switching vectors that are tracked in the states of $A_k^1, \ldots, A_k^n$ while mimicking a run of A on the input word. Therefore, within each context of stack h, we update the h-th component of all such tuples according to A_k^h transitions. On context switching from stack j to stack h, since $\flat$ and $\sharp$ are not in the input alphabet of A^D, we compute the effects of both any transition of A_k^j over such symbols and any transition of A_k^h over input a, while reading the first symbol a of the stack-h context. Namely, in each tuple $(I_1, \ldots I_n)$ in the $\mathcal{M}$ component, we update I_j according to any transition of A_k^j on $\flat$ or $\sharp$ (a different update is computed in either case), and I_h according to any transition of A_k^h on a. The components $\hat{q_1}, \ldots, \hat{q_n}$ are updated essentially by mimicking each deterministic automaton D_k^h on the stack-h contexts of the input word. Again, a transition over $\flat$ and $\sharp$ is mimicked in parallel with that of the deterministic automaton for the next context.

Formally, $Q^D = [0, n] \times (S_D)^n \times 2^{S^n}$ and $Q_0^D = \{0\} \times (S_{D,0})^n \times 2^{(S_0)^n}$. The set of final states is $F^D = \{(h, \hat{q_1}, \ldots, \hat{q_n}, \mathcal{M}) \mid h \in [n]$ and there is $(I_1, \ldots I_n) \in \mathcal{M}$ with $cur(I_h) \in F\}$.

For each push transition $t = (\hat{q}, a, \gamma, \hat{q'})$ of D_h and $I \in R(\hat{q})$, we denote with $Y_t(I)$ the set of all $I' \in R(\hat{q'})$ such that there is a push transition of A_k^h from I to I' on input a. Analogously, we can define $Y_t(I)$, when t is either a pop or an internal transition. We use Y_t to update the switching masks in our construction.

For $h > 0$ and $j \geq 0$, let $X = (j, \hat{q_1}, \ldots, \hat{q_n}, \mathcal{M})$ and $X' = (h, \hat{q_1}', \ldots, \hat{q_n}', \mathcal{M}')$ be two states from Q^D s.t. $\hat{q_i} = \hat{q_i}'$, for every $i \notin \{j, h\}$.

For $t = (\hat{q_h}, a, \hat{q_h}', \gamma)$ (resp. $t = (\hat{q_h}, a, \gamma, \hat{q_h}')$, $t = (\hat{q_h}, a, \hat{q_h}')$), if $a \in \Sigma^h$ and $t \in \delta_D^h$ then $(X, a, X', \gamma) \in \delta^D$ (resp. $(X, a, \gamma, X') \in \delta^D$, $(X, a, X') \in \delta^D$) provided that one of the following cases holds:

1. $j = 0$ and $\mathcal{M}'$ is the set of all $(I_1', \ldots, I_n')$ s.t. there exists $(I_1, \ldots, I_n) \in \mathcal{M}$, $in(I_h) \in Q_0$, $I_i = I_i'$ for $i \neq h$, and $I_h' \in Y_t(I_h)$ (*initial move*);
2. $h = j > 0$, and $\mathcal{M}'$ is the set of all $(I_1', \ldots, I_n')$ s.t. there exists $(I_1, \ldots, I_n) \in \mathcal{M}$, $I_i = I_i'$ for $i \neq h$, and $I_h' \in Y_t(I_h)$ (*move within a context*);
3. $h \neq j > 0$, $a \in \Sigma_c^h \cup \Sigma_r^h$ and $\mathcal{M}'$ is the set of all $(I_1', \ldots, I_n')$ s.t. there exists $(I_1, \ldots, I_n) \in \mathcal{M}$, $cur(I_j) = cur(I_h)$, $I_i = I_i'$ for $i \notin \{j, h\}$, $I_h' \in Y_t(I_h)$, and $I_j' \in Y_{t'}(I_j)$, where either $t' = (\hat{q_j}, \flat, \hat{q_j}') \in \delta_D^j$ or $t' = (\hat{q_j}, \sharp, \hat{q_j}', \gamma') \in \delta_D^j$, (*context-switch*).

The transition relation δ^D is the smallest one such that the above holds.

The tuples in the component $\mathcal{M}$ of A^D states of a run can be composed by concatenating the component switching vectors I_h as done for the single A_k^h to define $\mathcal{I}_k^h(z)$. Thus, for each run ρ of A^D we define a set of tuples obtained in this way. Denote this set as $\mathcal{L}_\rho$. It is possible to prove that $\mathcal{L}_\rho$ is exactly the set of all the k-scoped switching masks for the input word. From the definitions of the initial move and the set of initial states of A^D, we get that for each switching masks $M \in \mathcal{L}_\rho$, $st(M) \in Q_0$ holds. Moreover, from the definition of F^D and by the transition relation δ^D, we can show that if ρ is accepting, then there is at least a switching mask $M \in \mathcal{L}_\rho$ such that $cur(M) \in F$. Therefore, by Lemma 1:

Theorem 1. *For any n-stack call-return alphabet $\tilde{\Sigma}_n$ and any* Mvpa *A over $\tilde{\Sigma}_n$, there exists a deterministic* Mvpa *A^D over $\tilde{\Sigma}_n$ such that $Scoped(\widetilde{\Sigma}_n, k) \cap L(A^D) = Scoped(\widetilde{\Sigma}_n, k) \cap L(A)$. Moreover, the size of A^D is exponential in the number of the states of A and doubly exponential in both k and n.*

4 Closure under complement of Omvpl

Fix an ordered Mvpa A over a call-return alphabet $\widetilde{\Sigma}_n$.

Outline of the construction. We construct an ordered Mvpa $\bar{A}$ that accepts all the ordered words over $\widetilde{\Sigma}_n$ that are not in $L(A)$, i.e., the language $Ordered(\widetilde{\Sigma}_n) \setminus L(A)$, thus showing closure under complement of Omvpl. The style of the construction is to translate the Mvpa to a tree automaton, then by exploiting the closure under complementation of tree automata, construct a tree automaton for the complement, and finally, translate the resulting tree automaton back to an Mvpa. The crux is a regular-tree characterization of words such that each word w over a call-return alphabet corresponds to a labeled tree T where: each node corresponds to a position of w, the labeling encodes the linear order of w, and the right-child relation captures the matching relations. For this we introduce the concepts of T-path and path-tree.

We start recalling some notation on trees and tree automata.

Tree automata. 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.

We assume standard nondeterministic tree automata (see [21]). Also, in our construction, we use tree-walking automata [1] which are tree automata that can traverse the input tree by following a path (at any time the control is at a single node). We will use the fact that given a walking-tree automaton with r states, one can construct a language-equivalent tree automaton with $2^{O(r)}$ states [1].

Encoding paths on 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 $i \in [2, \ell]$, if v_i is a left child, i.e., $v_i = v.\swarrow$ for some $v \in T$, then v_{i-1} is the first occurrence of v 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.

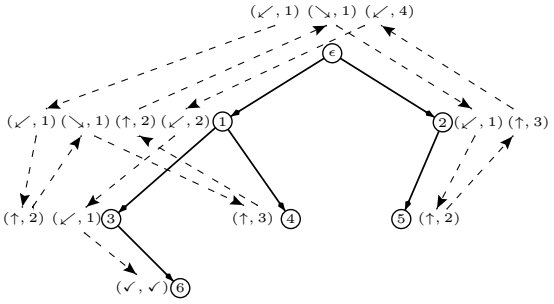

Fig. 1. A sample path-tree T_1.

We introduce the notion of *path-tree*, that is, a labeled tree (T, λ) that encodes a T-path in its labels as follows. Except for one node that is labeled with $(\checkmark, \checkmark)$, each other node is labeled with a sequence of pairs in $D \times \mathbb{N}$. 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.

Formally, denote $dir^+_\checkmark = dir^+ \cup \{(\checkmark, \checkmark)\}$ where $dir = D \times \mathbb{N}$ and $\checkmark \notin D \cup \mathbb{N}$. Also, 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, v_{j+1} is *pointed* at its i_j-th pair for $j \in [\ell - 1]$. The following lemmas hold (see Appendix for the proofs).

Lemma 2. *For any path-tree $\mathcal{T} = (T, \lambda)$, the first occurrence of each node u in $tp(\mathcal{T})$ is the one that is pointed at the first pair of u.*

Lemma 3. *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 4. *For any T-path π and path-tree Z, $tp(pt(\pi)) = \pi$ and $pt(tp(Z)) = Z$.*

Bounded-labeled path-trees are regular. We define *PathTree* as the set of all path-trees, and for any $k \in \mathbb{N}$, $PathTree_k$ as the set of all path-trees where nodes are labeled with at most k pairs.

Note that each property of the path-tree definition can be checked locally by looking just at the labels of a node and its children, and thus, by a top-down tree automaton that stores in its states the label of the parent of the current node. Since the number of different labels is exponential in k, the size of such automaton is also exponential in k.

Lemma 5. *For any $k \in \mathbb{N}$, there is an effectively constructible tree automaton accepting $PathTree_k$ whose size is exponential in k.*

From words to trees. In this subsection, we introduce a representation of words over a call-return alphabet as path-trees augmented with labels from this alphabet. Then, given an ordered MVPA A, we use this representation to construct a tree automaton $\mathcal{A}$ that accepts such a path-tree if and only if A accepts the corresponding word.

Tree encoding of words. For a word w over $\widetilde{\Sigma}_n$ we define a labeled tree $wt(w) = (T, (\lambda_{dir}, \lambda_\Sigma))$ such that (T, λ_{dir}) is a path-tree and λ_Σ labels the nodes of T s.t. (i) the right-child relation in T captures the matching relations in w (i.e., a call and its matching return label respectively a node and its right child, and unmatched calls, internals and returns do not label nodes with a right child) and (ii) w can be obtained by taking the ordered sequence of the Σ labels of the first occurrences of each vertex in $tp(T, \lambda_{dir})$.

Fix a word $w = a_1 \dots a_\ell$ over $\widetilde{\Sigma}_n$. More precisely, T, λ_{dir} and λ_Σ in the definition of $wt(w)$ are as follows.

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.

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 labeled with a_{i-1} to that labeled with a_i (first node excluded). From the definition of T-path, we get that π_w is a T-path. Thus, we let λ_{dir} be such that $(T, \lambda_{dir}) \in \mathit{PathTree}_k$, for some $k \in \mathbb{N}$, and $tp(T, \lambda_{dir}) = \pi_w$.

For $wt(w) = \mathcal{T}$, we denote with $\mathit{word}_\mathcal{T}$ the word w (note that $\mathit{word}_\mathcal{T}$ is well-defined by Lemma 4). With $\mathit{PathTree}_k(\widetilde{\Sigma}_n)$ we denote the set of all labeled trees $wt(w) = (T, (\lambda_{dir}, \lambda_\Sigma))$ such that $(T, \lambda_{dir}) \in \mathit{PathTree}_k$ (the length of the labels by λ_{dir} is bounded by k).

Regularity of $\mathit{PathTree}_k(\widetilde{\Sigma}_n)$. We construct a tree automaton for $\mathit{PathTree}_k(\widetilde{\Sigma}_n)$ as the intersection of two automata respectively checking the following two properties over the input $(T, (\lambda_{dir}, \lambda_\Sigma))$: I. $(T, \lambda_{dir}) \in \mathit{PathTree}_k$ and II. v is the right child of u if and only if $\mathit{pos}_\mathcal{T}(u) \sim_h \mathit{pos}_\mathcal{T}(v)$ in $\mathit{word}_\mathcal{T}$, for some $h \in [n]$, where $\mathit{pos}_\mathcal{T}(v)$ denotes the position of $\lambda_\Sigma(v)$ within $\mathit{word}_\mathcal{T}$.

The first automaton can be easily obtained by Lemma 5. In the following, we sketch the construction of the second automaton. The idea is to go through the negation of property II. For this, fix $\mathcal{T} = (T, (\lambda_{dir}, \lambda_\Sigma))$ and denote with $<$ the total order over the T nodes such that $u < v$ iff the first occurrence of u precedes the first occurrence of v in $tp(T, \lambda_{dir})$. By the definition of $\sim_h$, $h \in [n]$, the negation of II 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. (i) $u < v$, $\lambda_\Sigma(u) \in \Sigma_c^h$ 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 an input tree $\mathcal{T} = (T, (\lambda_{dir}, \lambda_\Sigma))$, by assuming $(T, \lambda_{dir}) \in \mathit{PathTree}_k$, we construct an automaton B_h that accepts $\mathcal{T}$ iff the right-child relation of $\mathcal{T}$ does not capture properly the matching relation $\sim_h$ of $\mathit{word}_\mathcal{T}$ (i.e., property II does not hold w.r.t. the matching relation $\sim_h$). B_h is given as the union of four automata, one for each of the above violations 1, 2, 3.*i* and 3.*ii*.

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 the violation 1. The size of this automaton is constant w.r.t. k and n.

In the other violations, the $<$ relation is used. It is simple to design a tree-walking automaton that visits the nodes of the input tree according to the sequence $\mathit{tp}(T, \lambda_{dir})$ (just follow the chain of the pairs in the λ_{dir} labeling). From Lemma 2, we get a simple criteria that can be checked locally on a node to determine the first occurrence of a vertex in a tree traversal, and thus we can check $u < v$. Thus, using this as base in our constructions, we can design the remaining tree automata quite easily. For example, to check 2.*ii*, we can assume that the automaton first guesses four nodes u, v, w, z (we can assume that such nodes are marked in the input tree and then remove the marking as in the usual projection construction) and then while visiting the tree according to $\mathit{tp}(T, \lambda_{dir})$, it checks that: if v is the right child of u, z is the right child of w, and u, w are labeled with calls of stack h, then either one of the orderings violating the nesting property holds. The size of this automaton is linear in k and thus we can construct a corresponding standard tree automaton of size $2^{O(k)}$. Similarly for the other violations we get corresponding tree automata of size $2^{O(k)}$, and thus B_h also has size $2^{O(k)}$.

For each tree $\mathcal{T}$ that is not accepted by B_h and is such that $(T, \lambda_{dir}) \in \mathit{PathTree}_k$, 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 (II) provided that the input tree $\mathcal{T}$ is such that $(T, \lambda_{dir}) \in \mathit{PathTree}_k$. Since complementation causes an exponential blow-up, the size of each $\bar{B}_h$ is $2^{2^{O(k)}}$, and the automaton resulting from their intersection has size $2^{O(n)\,2^{O(k)}}$. Therefore, by Lemma 5:

Lemma 6. *For $k \in \mathbb{N}$, there is an effectively constructible tree automaton accepting $\mathit{PathTree}_k(\widetilde{\Sigma}_n)$ of size exponential in n and doubly exponential in k.*

Complement automaton. Consider an ordered MVPA A over $\widetilde{\Sigma}_n$. By assuming that the input tree $\mathcal{T}$ belongs to $\mathit{PathTree}_k(\widetilde{\Sigma}_n)$, we can construct a tree automaton $\mathcal{A}_k$ that captures the runs of A over $\mathit{word}_\mathcal{T}$. The idea is to construct first a tree-walking automaton W_k that visits the nodes as in $\mathit{tp}(\mathcal{T})$ and mimics the A transitions on each newly visited vertex. To handle the stacks we can design W_k s.t. in the input tree the nodes labeled with calls are also labeled with the A stack symbol that has to be pushed, this way we can synchronize a push with a matching pop on this label. The size of W_k is linear in k and $|A|$. $\mathcal{A}_k$ is obtained by converting W_k to a tree automaton and then projecting out the additional labeling, therefore its size is exponential in k and $|A|$.

By intersecting $\mathcal{A}_k$ with the tree automaton P_k accepting $PathTree_k(\widetilde{\Sigma}_n)$, we get a tree automaton $\mathcal{B}_k$ accepting all the trees in $PathTree_k(\widetilde{\Sigma}_n)$ s.t. $word_{\mathcal{T}}$ is accepted by A. By [18], if we let $\kappa = (n+1)2^{n-1}+1$, any ordered word over $\widetilde{\Sigma}_n$ can be obtained as $word_{\mathcal{T}}$ for some $\mathcal{T} \in PathTree_\kappa(\widetilde{\Sigma}_n)$. Therefore, the set of ordered words $word_{\mathcal{T}}$ s.t. $\mathcal{T}$ is accepted by $\mathcal{B}_\kappa$ is exactly the set of ordered words accepted by A. Thus, we can complement $\mathcal{B}_\kappa$ and then take the intersection with P_κ, thus capturing all the trees $\mathcal{T} \in PathTree_\kappa(\widetilde{\Sigma}_n)$ s.t. the word $word_{\mathcal{T}}$ is not accepted by A (note that $word_{\mathcal{T}}$ does not need to be ordered now, but the intersection language still contains all the $\mathcal{T}$ s.t. $word_{\mathcal{T}}$ is ordered and not accepted by $\mathcal{B}_\kappa$). The size of the resulting tree automaton $\bar{\mathcal{B}}_\kappa$ is doubly exponential in $|A|$ and κ, and since $\kappa = O(n2^n)$, triply exponential in n.

From $\bar{\mathcal{B}}_\kappa$, we can construct an MVPA $\bar{A}$ that mimics $\bar{\mathcal{B}}_\kappa$ transitions as follows (see Appendix for more details): on internal symbols, $\bar{A}$ moves exactly as $\bar{\mathcal{B}}_\kappa$ (there is no right child); on call symbols, $\bar{A}$ enters the state that $\bar{\mathcal{B}}_\kappa$ would enter on the left child and pushes onto a stack the one that $\bar{\mathcal{B}}_\kappa$ 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 $\mathcal{T} \in PathTree_\kappa(\widetilde{\Sigma}_n)$, 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. If $word_{\mathcal{T}} \in Ordered(\widetilde{\Sigma}_n)$, such a stack is that with the lowest index among those with unmatched calls. In $\bar{A}$, popping the current state from the stack allows to restore properly the simulation of $\bar{\mathcal{B}}_\kappa$ from a right child. Being the size of $\bar{A}$ polynomial in $|\bar{\mathcal{B}}_\kappa|$, we get:

Theorem 2. *The class of* OMVPL*'s is closed under complement. Moreover, given an* MVPA *A, there is an effectively constructible* MVPA *$\bar{A}$ s.t. $L(\bar{A}) \cap Ordered(\widetilde{\Sigma}_n) = Ordered(\widetilde{\Sigma}_n) \setminus L(A)$, and the size of $\bar{A}$ is doubly exponential in the size of A and triply exponential in n.*

5 Other results and summary of the properties

Comparisons among the classes. All the classes of MVPL languages we consider in this paper are contained into the class of context-sensitive languages (CSL). The following languages allow us to distinguish among them:

$L_1 = \{a^i b^j c^i d^j \mid i,j > 0\}^*$, $L_2 = \{a^i b^j c^h d^j c^{i-h} \mid i > h > 0, j > 0\}$,

$L_3 = \{(ab)^i c^i d^i \mid i > 0\}$, and $L_4 = \{a^i b^j c^i d^j (ab)^h \mid i,j,h > 0\}$.

Fix $\Sigma = \{a,b,c,d\}$, $n = 2$ and $lab_{\Sigma,n}$ such that $\Sigma_c^1 = \{a\}$, $\Sigma_c^2 = \{b\}$, $\Sigma_r^1 = \{c\}$, and $\Sigma_r^2 = \{d\}$. For each $r \in [4]$, an MVPA over $\widetilde{\Sigma}_n = (\Sigma, lab_{\Sigma,n})$ exists accepting L_r. It turns out that all languages, besides L_2, contain only ordered words. Moreover, observe that words in L_1 are 2-scoped, but there is not a bound on the number of phases. Words in L_2 are 3-round. Words in L_3 are 2-phase but they are not k-scoped, for any k. Finally, words in L_4 are both 2-scoped and 2-phase but there is not a bound on the number of rounds. Thus,

	Closure properties				Decision Problems		
	$\cup$	$\cap$	Compl.	Determin.	Membership	Emptiness	Univ./ Equiv./Incl.
VPL	Yes	Yes	Yes	Yes	PTIME-C	PTIME-C	EXPTIME-C
CFL	Yes	No	No	No	PTIME-C	PTIME-C	Undecidable
RMVPL	Yes	Yes	Yes	Yes	NP	NP-C	2EXPTIME
PMVPL	Yes	Yes	Yes	No	NP-C	2ETIME-C	3EXPTIME
SMVPL	**Yes**	**Yes**	**Yes**	**Yes**	**NP-c**	PSPACE-C	**2Exptime**
OMVPL	Yes	Yes	**Yes**	No	**NP-c**	2ETIME-C	**3Exptime**
CSL	Yes	Yes	Yes	Unknown	NLINSPACE	Undecidable	Undecidable

Fig. 2. Summary of the main results on MVPLs (new results are in bold).

$L_1 \in (\text{SMVPL} \cap \text{OMVPL}) \setminus \text{PMVPL}$, $L_2 \in \text{RMVPL} \setminus \text{OMVPL}$, $L_3 \in (\text{PMVPL} \cap \text{OMVPL}) \setminus \text{SMVPL}$, and $L_4 \in (\text{PMVPL} \cap \text{SMVPL} \cap \text{OMVPL}) \setminus \text{RMVPL}$.

Theorem 3. *1)* RMVPL *is strictly contained in* SMVPL$\cap$PMVPL. *2)* RMVPL *and* OMVPL *are incomparable. 3)* SMVPL, OMVPL, PMVPL *are pairwise incomparable.*

Closure properties and decision problems. The table in Fig. 2 summarizes the closure properties and decision problems for the classes of languages we consider, and the known results for VPLs, CSLs, and CFLs (in the table, NLOG-C stands for NLOG-complete, and so on). We refer to [2] for VPLs, [13, 12] for PMVPL and RMVPL, and [10] for CSLs and CFLs. Closure under union and intersection for all classes can be shown with standard constructions (union and intersection are defined for languages over a same call-return alphabet).

Closure under complementation for SMVPL follows from determinizability which is shown in Section 3, and for OMVPL is shown in in Section 4.

The membership problem can be solved in nondeterministic polynomial time for both SMVPL and OMVPL by simply guessing the transitions on each symbol and then checking that they form an accepting run. A matching lower bound for OMVPL can be obtained with the reduction given in [13]. For SMVPL, we can give a reduction from the satisfiability of 3-CNF Boolean formulas: for a formula with k variables, we construct a k-stack MVPA that nondeterministically guesses a valuation by storing the value of each variable in a separate stack, then starts evaluating the clauses (when evaluating a literal the guessed value is popped and then pushed into the stack to be used for next evaluations); partial evaluations are kept in the finite control (each clause has just three literals and we evaluate one at each time; for the whole formula we only need to store if we have already witnessed that it is false or that all the clauses evaluated so far are all true); thus each stack is only used to store the variable evaluation, and since for each stack h, each pushed symbol is either popped in the next stack-h context or is not popped at all, the input word is 2-scoped.

Checking emptiness is known to be PSPACE-COMPLETE for SMVPL [14] and 2ETIME-COMPLETE for OMVPL [3] (2ETIME is the class of all decision problems solvable by a deterministic Turing machine in time $2^{2^{O(n)}}$). Decidability of universality, inclusion and equivalence follows from the closure under complementation and intersection, and the decidability of emptiness. This yields the

upper bounds given in the table. For the classes for which a matching lower bound is not known, the best known lower bound is derived either from that of VPLs or the emptiness problem.

References

1. Aho, A.V., Ullman, J.D.: Translations on a context-free grammar. Information and Control 19(5), 439–475 (1971)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In STOC. pp. 202–211. ACM (2004)
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. Atig, M.F., Bouajjani, A., Kumar, K.N., Saivasan, P.: Linear-time model-checking for multithreaded programs under scope-bounding. In ATVA. LNCS, vol. 7561, pp. 152–166. Springer (2012)
5. Atig, M.F., Kumar, K.N., Saivasan, P.: Adjacent ordered multi-pushdown systems. In DLT. LNCS, vol. 7907, pp. 58–69. Springer (2013)
6. Bollig, B.: On the expressive power of 2-stack visibly pushdown automata. Logical Methods in Computer Science 4(4) (2008)
7. Bollig, B., Kuske, D., Mennicke, R.: The complexity of model checking multi-stack systems. In: LICS. pp. 163–172. IEEE Computer Society (2013)
8. 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)
9. Carotenuto, D., Murano, A., Peron, A.: 2-visibly pushdown automata. In DLT. LNCS, vol. 4588, pp. 132–144. Springer (2007)
10. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
11. 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)
12. La Torre, S., Madhusudan, P., Parlato, G.: The language theory of bounded context-switching. In LATIN. LNCS, vol. 6034, pp. 96–107. Springer (2010)
13. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In LICS. pp. 161–170. IEEE Computer Society (2007)
14. La Torre, S., Napoli, M.: Reachability of multistack pushdown systems with scope-bounded matching relations. In CONCUR. LNCS, vol. 6901, pp. 203–218. (2011)
15. La Torre, S., Napoli, M.: A temporal logic for multi-threaded programs. In IFIP TCS. LNCS, vol. 7604, pp. 225–239. Springer (2012)
16. 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)
17. Lal,A., Kidd,N., Reps,T.W., Touili,T.: Interprocedural analysis of concurrent programs under a context bound. In TACAS. LNCS, vol. 4963, pp. 282–298. (2008)
18. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In POPL. pp. 283–294. ACM (2011)
19. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In TACAS. LNCS, vol. 3440, pp. 93–107. Springer (2005)
20. Seth, A.: Global reachability in bounded phase multi-stack pushdown systems. In Science, vol. 6174, pp. 615–628. Springer (2010)
21. Thomas, W.: Languages, automata, and logic. In Handbook of Formal Languages, vol. 3, pp. 389–455. Springer (1997)

A Proofs of Lemma 2 and Lemma 3

Fix a path-tree $\mathcal{T} = (T, \lambda)$. Let $v_1, \ldots, v_\ell$, and $d_1, \ldots, d_\ell$, and $i_1, \ldots, i_\ell$ be defined 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 7. *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 2.

Lemma 2. *For a path-tree $\mathcal{T} = (T, \lambda)$, the first occurrence of each node u in $tp(\mathcal{T})$ is the one that is pointed at the first pair of u.*

Proof. We observe that for the left children in $\mathcal{T}$ this is a direct consequence of property (5) of Def. 2 and the definition of $tp(\mathcal{T})$. In general, from Lemma 7, we get that all the pairs in the labeling of $\mathcal{T}$ are used to construct $tp(\mathcal{T})$. Moreover, as shown in the proof of Lemma 7, by property (6) of Def. 2, all the pairs in the labeling of any node must be visited by increasing indexes. Therefore, the first time a node v is visited along $tp(\mathcal{T})$, this is done by pointing to its first pair. □

Now, we can show Lemma 3.

Lemma 3. *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 7, 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 7 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 2 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 4

Before proving Lemma 4, we show the following property of path-trees.

Lemma 8. *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 3 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 4. *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_j^π defining $pt(\pi_j)$ is obtained from λ_{i-1}^π leaving unchanged the labels of all nodes but v_j which gets the label $\lambda_j^\pi(v_j) = \lambda_{j-1}^\pi(v_j).(d_j, i_j)$ where $v_{j+1} = v_j.d_j$. Since the concatenated pair of $\lambda_j^\pi(v_j)$ 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 3, 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 8, we can conclude that $pt(tp(\mathcal{T})) = tp(\mathcal{T})$. □

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

In this section we give a central lemma that allows to prove that ordered 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 9. *For any tree automaton $\mathcal{A}$ over Υ-labeled trees with $\mathcal{L}(\mathcal{A}) \subseteq PathTree_k(\widetilde{\Sigma}_n)$, there is an effectively constructible ordered* MVPA *A over $\widetilde{\Sigma}_n$ such that $L(A)$ is the set of all ordered 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 automaton 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.

The automaton A accepts an ordered 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 labelled 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 $|\mathcal{A}| \cdot 2^n$. □