International Journal of Foundations of Computer Science
Vol. 27, No. 2 (2016) 215–233

DOI: 10.1142/S0129054116400074

World Scientific
www.worldscientific.com

Scope-Bounded Pushdown Languages

Salvatore La Torre* and Margherita Napoli†
Dipartimento di Informatica, Università degli Studi di Salerno
Via Giovanni Paolo II 132, 84084 Fisciano, Italy
**slatorre@unisa.it*
†*napoli@unisa.it*

Gennaro Parlato
School of Electronics & Computer Science, University of Southampton
Highfield, Southampton S0171BJ, UK
gennaro@ecs.soton.ac.uk

Received 27 July 2015
Accepted 5 December 2015
Communicated by Arseny Shur

We study the formal language theory of multistack pushdown automata (Mpa) restricted to computations where a symbol can be popped from a stack S only if it was pushed within a bounded number of contexts of S (*scoped* Mpa). We show that scoped Mpa are indeed a robust model of computation, by focusing on the corresponding theory of *visibly* Mpa (Mvpa). We prove the equivalence of the deterministic and nondeterministic versions and show that scope-bounded computations of an n-stack Mvpa can be simulated, rearranging the input word, by using only one stack. These results have some interesting consequences, such as, the closure under complement, the decidability of universality, inclusion and equality, and the effective semilinearity of the Parikh image (Parikh's theorem). As a further contribution, we give a logical characterization and compare the expressiveness of the scope-bounded restriction with other Mvpa classes from the literature. To the best of our knowledge, scoped Mvpa languages form the largest class of formal languages accepted by Mpa that enjoys all the above nice properties.

Keywords: Language theory and automata; models of concurrent systems; multistack pushdown automata; visibly pushdown automata.

1. Introduction

Pushdown automata working with multiple stacks (*multistack pushdown automata*, Mpa for short) are the automata-theoretic model of concurrent programs with recursion and shared memory. Within the domain of formal verification of programs, program executions are analyzed against correctness properties, that may refer to the stack operations in the model such as for stack inspection properties and Hoare-like pre/post conditions. Such visibility of stack operations is captured in the formal languages by the notion of *visibly pushdown language* [1].

The class of *multistack visibly pushdown languages* (MVPL) is defined via the model of *multistack visibly pushdown automaton* (MVPA), that is a MPA where the push and pop operations on each stack are made visible in the input symbols, by a partition of the input alphabet into calls, returns and internals. Though visibility allows to synchronize the stack usage in the constructions, thus gaining interesting properties such as the closure under intersection, in general, it does not limit the expressiveness up to gaining decidability: the language of the executions (i.e., the sequence of transitions) of a MPA is a MVPL, and MPA are equivalent to Turing machines already with two stacks.

In this paper, we study the formal language theory of MVPA restricted to *scoped computations* [2]: for a positive integer k, a computation is k-scoped if for each stack i, each popped symbol was pushed within the last k contexts of i (where a context is a continuous portion of the computation in which only one stack is used).

The notion of scoped computations was first introduced in [3] to extend the bounded-context switching analysis of MPA [4] to unboundedly many contexts. In this original formulation, though the overall number of contexts allowed in a computation is unbounded, only a bounded number of them can occur between a matching pair of push and pop actions of a stack i. The notion of scoped computations given above, that we will use in this paper, instead allows for unboundedly many contexts of stacks $j \neq i$ and thus significantly increases expressiveness.

Our first main contribution is to prove that deterministic and nondeterministic scoped MVPA are language equivalent. The main notion used in our construction is the *switching mask*. A switching mask summarizes the states of a MVPA at context-switches. We show that for scoped computations also the switching masks are bounded. The resulting deterministic MVPA has size doubly exponential in both the number of stacks and the bound k. By this construction we gain the closure under complement, and by the effectiveness of closure under intersection and the decidability of emptiness, we also get the decidability of universality, inclusion, and equality. In general, MVPA and most of the already studied classes of MVPA are not determinizable [5].

As a second main contribution, we show a sequentialization construction for scoped MVPA. Namely, we give a mapping π that rearranges the contexts in a scoped word w s.t. it can be read by using only one stack (all the calls and returns of the starting alphabet are interpreted as calls and returns of the only available stack). We show a construction that starting from a MVPA A builds a visibly pushdown automaton A_{seq} that accepts all the scoped words in $\pi(L(A))$. Sequentialization of concurrent programs is nowadays one of the emerging techniques for building model-checkers for concurrent programs (see [6–9]). As a corollary of this result, we can show a Parikh theorem for scoped MVPL.

Closure under union and intersection can be shown via standard constructions, and since the reachability problem is PSPACE-COMPLETE [2], we also get that emptiness is PSPACE-COMPLETE. Decidability of membership is straightforward: guess

and check a run over the input word. We also give an MSO characterization of scoped MVPL. To the best of our knowledge this class is the largest subclass of MVPL with all the above properties.

As a further result we compare scoped MVPL with the main MVPL classes from the literature and show that it is incomparable with the most expressive ones, and strictly subsumes the others.

Related work: A preliminary version of this paper has appeared as [10]. A fixed-point algorithm for the reachability problem and a sequentialization are given in [11] for MPA under the scoped restriction given [3]. The notion of scoped computations naturally extends to infinite words, and the related model checking problem is addressed in [2,12] for LTL-like temporal logics, and in [13] for MSO-definable temporal logic using the notion of split-width [14]. Split-width is also used to show the decidability of the MSO theory of the graphs corresponding to scoped computations [14]. Global reachability for concurrent collapsible pushdown automata restricted to scoped computations was solved in [15].

In the literature several other classes of MVPL have been studied. The classes of *phase* MVPL [16, 17], *ordered* MVPL [18, 19], and *path-tree* MVPL [20] are not determinizable and incomparable with scoped MVPL. The class of *round* MVPL [5], which is based on the notion of bounded-context switching [4], has the same properties as scoped MVPL (though checking emptiness is NP-COMPLETE) but it is strictly included in it. More work on decision problems on MPA with restrictions is done in [21–26].

The bounded context-switching restriction was proposed in [4] for under-approximate analysis of multi-threaded programs and was motivated by the fact that many concurrency errors manifest themselves already after only a few context switches. This has given rise to a variety of *context-bounded analysis* methods [6–9, 27–31].

MPA with asynchronous shared memory [32] do not allow to execute atomically a read followed by a write in the shared memory. For this model, reachability turns out to be PSPACE-COMPLETE for configurations of one leader pushdown thread and unboundedly many instances of a contributor pushdown thread [32,33]. Decidability is extended to any class of threads that are effectively closed under synchronized product with finite automata, and for the leader, also have an effective downward closure or an effective semilinear Parikh image [34]. Our results on scoped MPA makes this class a suitable candidate both for the leader and the contributor threads.

Visibility of stack operations was first introduced for input-driven pushdown automata [35] (see also [36] and references therein). Parikh theorem was originally given for context-free languages in [37].

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 an integer $n > 0$, an n-stack call-return alphabet $\widetilde{\Sigma}_n$ is $(\Sigma^{int}, \langle \Sigma_h^c, \Sigma_h^r \rangle_{h \in [n]})$, where $\Sigma^{int}, \Sigma_1^c, \Sigma_1^r, \ldots, \Sigma_n^c, \Sigma_n^r$ are

pairwise disjoint finite alphabets. Σ^{int} is the set of *internals*, and for $h \in [n]$, Σ_h^r is the set of *stack-h returns* and Σ_h^c is the set of *stack-h calls*.

In the following, fix an n-stack call-return alphabet $\widetilde{\Sigma}_n$, and $\Sigma_h = \Sigma_h^c \cup \Sigma_h^r \cup \Sigma^{int}$, $\Sigma^c = \bigcup_{h\in[n]} \Sigma_h^c$, $\Sigma^r = \bigcup_{h\in[n]} \Sigma_h^r$ and $\Sigma = \Sigma^{int} \cup \Sigma^r \cup \Sigma^c$.

For a word $w = a_1 \dots a_m \in \widetilde{\Sigma}_n$, denoting $C_h = \{i \in [m] \mid a_i \in \Sigma_h^c\}$ and $R_h = \{i \in [m] \mid a_i \in \Sigma_h^r\}$, the *matching relation* defined by w is $\sim_h \subseteq C_h \times R_h$ s.t.:

(1) if $i \sim_h j$ then $i < j$,
(2) 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
(3) 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. $j \sim_h i$).

It is easy to see that the above definition uniquely identify a binary relation. Assume, by contradiction, that there are two different relations $\sim_h, \sim'_h$ satisfying the items above and there are $i, j \in [m]$ such that: $i \sim_h j$, $i \not\sim'_h j$ and, for every $i', j' \in [m]$ such that $j' - i' < j - i$, $i' \sim_h j'$ if and only if $i' \sim'_h j'$. Since $i \sim_h j$, we have that $i \in C_h$, $j \in R_h$, and from (1) above that $i < j$. Since $\sim'_h$ satisfies (2), there is an $i' \in [i, j]$ s.t. either $i' \sim'_h j$ or $i \sim'_h i'$. Suppose that $i' \sim'_h j$ (the other case is similar). Being $j - i' < j - i$, by hypothesis we get $i' \sim_h j$, and then, from (3) $i' = i$ must hold. Therefore, $i \sim'_h j$, as well, that contradicts our hypothesis.

When $i \sim_h j$, we say that positions i and j *match* in w (they are *matching call and return* 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 pushes a symbol on stack h when it reads a stack-h call, and pops a symbol from stack h when it reads a stack-h return. Moreover, it just changes its state, without reading or 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* $|\{(q, a, q') \in \delta \mid q' \in Q\} \cup \{(q, a, q', \gamma') \in \delta \mid q' \in Q, \gamma' \in \Gamma\} \cup \{(q, a, \gamma, q') \in \delta \mid q' \in Q\}| \leq 1$*, 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:

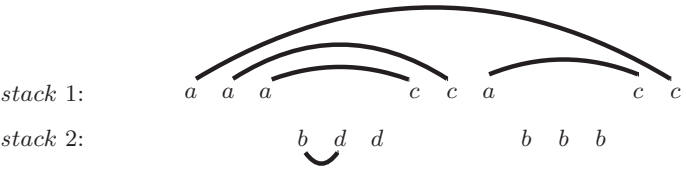

Fig. 1. A sample 3-scoped word.

[Push] $a \in \Sigma_h^c$, $\exists \gamma \in \Gamma \setminus \{\bot\}$ 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] $a \in \Sigma_h^r$, $\exists \gamma \in \Gamma$ 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 $w = a_1 \dots a_m \in \Sigma^*$, 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 is accepted by 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 is a *multi-stack visibly pushdown language* (MVPL) if it is accepted by an MVPA over a call-return alphabet $\widetilde{\Sigma}_n$.

A visibly pushdown automaton (VPA) [1] is an MVPA with just one stack, and a *visibly pushdown language* (VPL) is an MVPL accepted by a VPA.

Scope-bounded matching relations [2, 3]. A *stack-h context* is a word in Σ_h^+. For a word $w = a_1 \dots a_m$, we denote with $w[i, j]$ the subword $a_i \dots a_j$.

A word w is *k-scoped* (according to $\widetilde{\Sigma}_n$) if for each $h \in [n]$ and $i, j \in [m]$ s.t. $i \sim_h j$, $w[i, j] \in \Sigma_h^* \, (\Sigma_{\neq h}^* \, \Sigma_h^*)^{k-1}$ where $\Sigma_{\neq h} = \bigcup_{h' \neq h} \Sigma_{h'}$ i.e., each matching call and return of stack h occur within at most k stack-h maximal contexts.

In all the examples, we assume $\Sigma_1^c = \{a\}$, $\Sigma_2^c = \{b\}$, $\Sigma_1^r = \{c\}$, and $\Sigma_2^r = \{d\}$. Consider a sample word $\nu_1 = a^3 \, bd^2 \, c^2 a \, b^3 \, c^2$. Figure 1 illustrates its splitting into contexts and the matching relations, denotated with edges. Note that the only pair of matching b's and d's is in the same stack-2 context. Moreover, the first a occurs in the first stack-1 context and is matched by the last c which occurs in the third stack-1 context. Any other matching pair of a's and c's occur within two stack-1 contexts. Therefore, ν_1 is k-scoped for any $k \geq 3$ but it is not 2-scoped.

With $Scoped(\widetilde{\Sigma}_n, k)$, we denote the set of all the k-scoped words over $\widetilde{\Sigma}_n$. A language $L \subseteq \Sigma^*$ is a *scoped* MVPL (SMVPL) if $L = Scoped(\widetilde{\Sigma}_n, k) \cap L(A)$ for some MVPA A over the call-return alphabet $\widetilde{\Sigma}_n$.

3. Properties of MVPA Runs Over Scoped Words

Fix an integer $k > 0$ and an MVPA $A = (Q, Q_0, \Gamma, \delta, F)$ over $\widetilde{\Sigma}_n$.

***k*-scoped splitting.** For a word $w \in \Sigma^*$ and $h \in [n]$, a *cut* of w is $w_1 : w_2$ s.t. $w = w_1 w_2$. Such a cut is *consistent* with the matching relation $\sim_h$ ($\sim_h$-*consistent*,

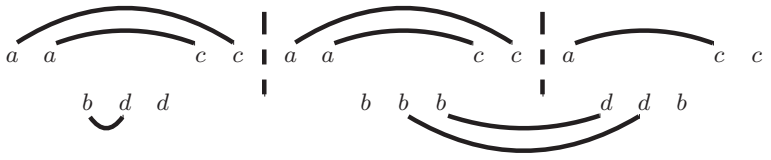

Fig. 2. A sample 2-scoped splitting.

for short) if in w no call of stack h occurring in the prefix w_1 is matched with a return occurring in the suffix w_2.

A ($\sim_h$-*consistent*) *splitting* of w is defined by a set of ($\sim_h$-*consistent*) cuts of w, that is, it is an ordered tuple $\langle w_i \rangle_{i \in [d]}$ s.t. $w = w_1 \dots w_d$ and $w_1 \dots w_i : w_{i+1} \dots w_d$ is a ($\sim_h$-*consistent*) cut of w, for each $i \in [d-1]$.

A *context-splitting* of w is a splitting $\langle w_i \rangle_{i \in [d]}$ of w, where w_i is a stack-h_i context for $h_i \in [n]$, and $i \in [d]$. The *canonical* context-splitting of w is the only context-splitting $\langle w_i \rangle_{i \in [d]}$ s.t., for each $i \in [2, d]$, stack-h_i context w_i starts with a call or a return, and $h_{i-1} \neq h_i$. For example, Fig. 1 gives the canonical context-splitting η of ν_1 that splits ν_1 into: *aaa*, *bdd*, *cca*, *bbb*, and *cc*.

For all $h \in [n]$, the *h-projection* of a context-splitting $\chi = \langle w_i \rangle_{i \in [d]}$ is obtained from χ by deleting all the w_i that are not stack-h contexts. For example, the 2-projection of η is: *bdd*, *bbb*. Note that a h-projection is trivially a context-splitting.

An ordered tuple $\chi = \langle w_i \rangle_{i \in [d]}$ of stack-h contexts is *k-bounded* if there is a $\sim_h$-consistent splitting $\xi = \langle v_i \rangle_{i \in [m]}$ of $w_1 \dots w_d$ s.t. each v_i is the concatenation of at most k consecutive contexts of χ. In the following, we refer to such a ξ as a *k-bounding* splitting for χ and will denote with χ_{v_i} the ordered tuple of the contexts from χ that form v_i, for $i \in [m]$.

A *k-scoped splitting* χ of w is the canonical context-splitting of w refined with additional cuts s.t. for all $h \in [n]$, the h-projection of χ is k-bounded.

Consider a sample word $\nu_2 = a^2\, bd^2\, c^2 a^2\, b^3\, c^2 a\, d^2 b\, c^2$. Figure 2 illustrates a 2-scoped splitting χ that refines the canonical context-splitting of ν_2 by further cutting it at the dashed vertical lines. Thus, χ splits ν_2 into: *aa*, *bdd*, *cc*, *aa*, *bbb*, *cc*, *a*, *ddb*, *cc*. We observe that the dashed lines define a $\sim_1$-consistent splitting of word $a^2\, c^2\, a^2\, c^2\, a\, c^2$ where each portion is the concatenation of two contexts of the 1-projection of χ. Moreover, by cutting the word $bd^2\, b^3\, d^2 b$ at the first dashed line, we get a $\sim_2$-consistent splitting where each portion has at most two contexts of the 2-projection of χ.

Given a k-scoped word w, for each stack h each matching call and return of stack h in w occur within at most k stack-h maximal contexts. Thus, for each stack h, it is possible to split w with cuts that are consistent with $\sim_h$ and such that each portion contains at most k stack-h maximal contexts. Moreover, we can pick these cuts such that they split w either between two consecutive maximal contexts or within a maximal context of stack h. Thus, we can obtain a k-scoped splitting of w by taking the canonical splitting of w and then refining it with the above

cuts for each stack h. Vice-versa, a k-scoped splitting of w gives evidence that each matching call and return of stack h in w occur within at most k stack-h maximal contexts, and thus, that w is k-scoped. Therefore, we get the following lemma:

Lemma 2. *A word w is k-scoped iff there is a k-scoped splitting of w.*

Scope-bounded switching-vector VPA. Fix $h \in [n]$. We start by recalling the definition of switching vector [5]. Intuitively, a switching vector summarizes the computations of an MVPA across several consecutive stack-h contexts.

Let A^h be the VPA over Σ_h obtained by restricting A to use only stack h. For $d > 0$, a tuple $I = \langle in_i, out_i \rangle_{i \in [d]}$ is a stack-h *d-switching vector* (d-SV, for short, or simply SV when we do not need to refer to its size) if there is an ordered tuple $\langle w_i \rangle_{i \in [d]}$ of stack-h contexts such that, for $i \in [d]$, $\langle in_i, \sigma_{i-1} \rangle \xrightarrow{w_i}_{A^h} \langle out_i, \sigma_i \rangle$ where $\sigma_0 = \perp$ (i.e., there is a sequence of d runs of A^h where the first run starts from a stack containing only the bottom-of-stack symbol and each other run starts with the stack content that is left at the end of the previous run in the sequence). We also define $st(I) = in_1$ and $cur(I) = out_d$, and say that $\langle w_i \rangle_{i \in [d]}$ *witnesses* I.

A stack-h *k-scoped* switching vector is an SV I that can be witnessed by a k-bounded ordered tuple of stack-h contexts.

Let χ be a k-bounded ordered tuple of stack-h contexts and $\xi = \langle v_i \rangle_{i \in [m]}$ be a k-bounding splitting for χ. Denote with I a stack-h k-scoped SV witnessed by χ. From the definition, I is given by the concatenation $I_1 \dots I_m$ where each I_i is a stack-h d_i-SV witnessed by χ_{v_i} and $d_i \in [k]$ is the number of contexts of χ_{v_i}. Note that not all the concatenations of SV's with at most k pairs form a k-scoped SV. In fact, by concatenating two witnesses a call from one could match a return from the other, thus the resulting tuple could not be k-bounded.

We encode a tuple of stack-h contexts by marking the first symbol of each context. Namely, for each $a \in \Sigma$, we add a fresh symbol $\bar{a}$ that is a call (resp. return, internal) if a is a call (resp. return, internal). Let $\bar{\Sigma}_h$ denote the set of all such new symbols. For a word $u = a_1 a_2 \dots a_r$, we denote with $\bar{u}$ the word $\bar{a}_1 a_2 \dots a_r$. We encode a tuple of stack-h contexts $u_1, \dots, u_m$ as $\bar{u}_1 \bar{u}_2 \dots \bar{u}_m$.

We now define a VPA A^h_k that ensures the following: if the input word is an encoding of a tuple χ of stack-h contexts that can be refined with additional cuts into a k-bounded tuple χ', then A^h_k *computes* all the stack-h k-scoped SV's of A witnessed by χ'. Essentially, A^h_k nondeterministically guesses any refinement χ' of χ and any k-bounding splitting for χ', and moreover, for each resulting portion, say formed by $d \leq k$ contexts, it computes a corresponding d-SV while mimicking the behavior of A^h (the new symbols $\bar{a}$ are interpreted as a by A^h_k when mimicking the moves of A^h).

By assuming the input $\bar{u}_1 \bar{u}_2 \dots \bar{u}_m$, in a typical run, A^h_k starts from any $(p, p) \in Q^2$ and on reading the first symbol of $\bar{u}_1$, it updates the second component in this pair according to an A^h move. Now, assume a stored pair (p, p'). On any other symbol of $\bar{u}_1$, for any move of A^h from p' to p'' there are two nondeterministic moves of A^h_k: one updating p' to p'' in the stored pair (as before), the other starting

a new SV by storing (p', p'') and thus guessing a cut. On the first symbol of $\bar{u}_2$, for any $q \in Q$ and for any move of A^h from q to q', again there are two nondeterministic moves as before: one updating the stored pair to $(p, p')(q, q')$, the other starting a new SV by replacing the stored pair with (q, q'). Then, the run continues similarly on the rest of the input.

There are two more aspects that A^h_k needs to take care of.

First, we only store d-SV's for $d \leq k$: when context-switching (i.e., reading a symbol $\bar{a} \in \bar{\Sigma}_h$), the nondeterministic move of appending a new pair to the stored SV I must not be allowed if I already contains k pairs. By Lemma 2, this is sufficient for our purposes.

Second, we need to ensure that A^h_k uses only the portion of the stack that has been pushed since the computation of the current SV started; moreover, if it attempts to pop a symbol that was pushed when computing the previous SV, then the guessed splitting is clearly wrong (one of the guessed cuts is not consistent with $\sim_h$) and the computation should halt. To ensure this, we store a bit e^s in the states of A^h_k and maintain the invariant: $e^s = 1$ iff the stack does not contain symbols pushed after the last guessed cut. Also, since pop transitions on an empty stack are allowed in VPAs, even if the portion of the stack currently in use is empty, we should allow them only if the whole stack is also empty. Thus, we store another bit e^g and maintain the invariant: $e^g = 1$ iff the stack is empty.

To maintain these two invariants, for each stack symbol γ of A^h we use two additional stack symbols γ^g and γ^s respectively to mark the bottom of the whole stack and the bottom of the currently used stack portion.

A state of A^h_k is thus (e^g, e^s, I) where $e^g, e^s \in \{0, 1\}$ and $I \in (Q \times Q)^m$, $m \in [k]$. All the states are final and all the states of the form $(1, 1, (q, q))$ for $q \in Q$ are initial.

We leave to the reader the formal definition of the transitions.

Let w be a word over the alphabet $\Sigma_h \cup \bar{\Sigma}_h$. With $\mathcal{I}^h_k(w)$, we denote the set of the SV's $I \in \bigcup_{d>0}(Q \times Q)^d$ s.t. there exists a run ρ of A^h_k on w and I is the concatenation of $I_1, \ldots, I_j, I_{j+1}$ where: I_{j+1} is the SV stored in the state of the last configuration of ρ and $I_1, \ldots, I_j$ is the sequence of the SV's of all the states occurring at the configurations of ρ from which a transition that starts a new SV is taken (in the order they appear in ρ).

From the above construction, we get:

Lemma 3. *I is a stack-h k-scoped switching vector of A iff $I \in \mathcal{I}^h_k(w)$ for some $w \in (\Sigma_h \cup \bar{\Sigma}_h)^*$.*

Consider the following example. Let ρ be a run of an MVPA A over a 3-stack call-return alphabet given as $\langle q_{i-1}, \bar{\sigma}_{i-1}\rangle \xrightarrow{u_i} \langle q_i, \bar{\sigma}_i\rangle$ with $i \in [11]$ and contexts u_i. Let $v_1 = \bar{u}_1\bar{u}_7\bar{u}_9$ be accepted by A^1_3, $v_2 = \bar{u}_2\bar{u}_4\bar{u}_{10}$ by A^2_3 and $v_3 = \bar{u}_3\bar{u}_5\bar{u}_6\bar{u}_8\bar{u}_{11}$ by A^3_3. The 3-SV's computed on v_1 and v_2 according to ρ are respectively S_1 and S_2 A^3_3 computes on v_3 the concatenation of the 2-SV S_3 over $\bar{u}_3\bar{u}_5$ and the 3-SV S_4 over $\bar{u}_6\bar{u}_8\bar{u}_{11}$. Figure 3 shows these switching vectors.

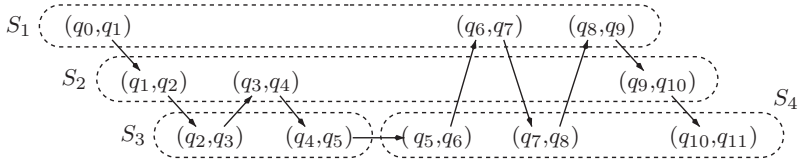

Fig. 3. Sample switching vectors and 3-scoped switching mask.

Switching masks. We use the SV's to summarize the runs of an MVPA over scoped words. For a k-scoped splitting χ of a word w over $\widetilde{\Sigma}_n$ and $h \in [n]$, denote with d_h the number of contexts in the h-projection χ^h of χ. Moreover, for $h, h' \in [n]$, $j \in [d_h]$ and $j' \in [d_{h'}]$, we define $next_\chi(h, j) = (h', j')$ s.t. the j'-th context of $\chi^{h'}$ is the context following in w the j-th context of χ^h.

For a word w over $\widetilde{\Sigma}_n$, a tuple $M = (I_1, \ldots, I_n)$ is a *k-scoped switching mask* for w if there is a k-scoped splitting χ of w s.t. for $h \in [n]$: (1) $I_h = \langle in_y^h, out_y^h \rangle_{y \in [x_h]}$ is a stack-h k-scoped SV of A and (2) $out_y^h = in_{y'}^{h'}$ for each h, y, h', y' for which $next_\chi(h, y) = (h', y')$. Moreover, we let $st(M) = st(I_{h_1})$ and $cur(M) = cur(I_{h_d})$, where each w_i in χ is a stack-h_i context.

In Fig. 3, we give the 3-scoped switching mask according to the sample run ρ given above. The edges denote the mapping $next_\chi$.

Thus, by the given definitions and Lemmas 2 and 3, the following holds:

Lemma 4. *Let $A = (Q, Q_0, \Gamma, \delta, F)$ be an* MVPA *over $\widetilde{\Sigma}_n$ and $w \in Scoped(\widetilde{\Sigma}_n, k)$. 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 $cur(M) \in F$.*

4. Determinization, Closure Properties and Decision Problems

Determinization. We show that, when restricting to k-scoped words, deterministic and nondeterministic MVPA are equivalent.

Let A be an MVPA over a call return alphabet $\widetilde{\Sigma}_n$. 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 k-scoped splitting of w. Thus, A^D accepts w iff it constructs a switching mask as in Lemma 4, and by supposing $w \in Scoped(\widetilde{\Sigma}_n, k)$, 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 $A_k^h = (S, S_0, \Gamma, \delta^h, S)$ and obtained through the construction given in [1]. 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 $\hat{q} \in S_D$, denote with $R(\hat{q})$ the set of SV's contained in the A_k^h states stored as the second component of $\hat{q}$.

We construct $A^D = (Q^D, Q_0^D, \Gamma^D, \delta^D, F^D)$ building on the cross product of $D_k^1, \ldots, D_k^n$; a state of A^D is $(i, \hat{q_1}, \ldots, \hat{q_n}, \mathcal{M})$, where $i > 0$ denotes the stack that is active in the current context, $i = 0$ 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 from $R(\hat{q_h})$. The idea

is to accumulate in the $\mathcal{M}$ component the tuples corresponding to the current SV's 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, in each tuple $(I_1, \ldots I_n)$ in the $\mathcal{M}$ component, on reading input a, we update I_h according to any transition of A_k^h on a if this is not the first symbol of the context, and on $\bar{a}$, otherwise (when context-switching into a stack-h context). 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 by dealing with the first symbol of each context as before. The accepting states are of the form $(i, \hat{q_1}, \ldots, \hat{q_n}, \mathcal{M})$ s.t. there is $(I_1, \ldots I_n) \in \mathcal{M}$ with $cur(I_i) \in F$.

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

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 $i \notin \{j, h\}$. Let $a \in \Sigma_h$.

For $t = (\hat{q_h}, a, \hat{q_h}', \gamma)$ (resp. $t = (\hat{q_h}, a, \gamma, \hat{q_h}')$, $t = (\hat{q_h}, a, \hat{q_h}')$), we add (X, a, X', γ) (resp. (X, a, γ, X'), (X, a, X')) to δ^D provided that $t \in \delta_D^h$, $h = j > 0$, 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*).

For $t = (\hat{q_h}, \bar{a}, \hat{q_h}', \gamma)$ (resp. $t = (\hat{q_h}, \bar{a}, \gamma, \hat{q_h}')$, $t = (\hat{q_h}, \bar{a}, \hat{q_h}')$), if $t \in \delta_D^h$ then $(X, a, X', \gamma) \in \delta^D$ (resp. $(X, a, \gamma, X') \in \delta^D$, $(X, a, X') \in \delta^D$) provided that:

(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*), or
(2) $h \neq j > 0$, $a \in \Sigma_h^c \cup \Sigma_h^r$ 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 $i \neq h$, and $I_h' \in Y_t(I_h)$ (*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 $\mathcal{L}_\rho$ of tuples obtained in this way. We can show that $\mathcal{L}_\rho$ is exactly the set of all the k-scoped switching masks for the input word. Also, from the above description, we get that for each switching mask $M \in \mathcal{L}_\rho$, $st(M) \in Q_0$ holds, and 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 4:

Theorem 5. *For any n-stack call-return alphabet $\widetilde{\Sigma}_n$ and any* MVPA *A over $\widetilde{\Sigma}_n$, there exists a deterministic* MVPA *A^D over $\widetilde{\Sigma}_n$ 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 k and n.*

Closure properties and decision problems. Language union and intersection are defined for languages over a same call-return alphabet. The closure under these set operations can be shown with standard constructions and by exploiting that the stacks are synchronized over the input symbols. Complementation is defined w.r.t. the set $\mathit{Scoped}(\widetilde{\Sigma}_n, k)$ for a call-return alphabet $\widetilde{\Sigma}_n$, that is the complement of L is $\mathit{Scoped}(\widetilde{\Sigma}_n, k) \setminus L$. The closure under complementation follows from determinizability (Theorem 5).

The *membership problem* can be solved in nondeterministic polynomial time by simply guessing the transitions on each symbol and then checking that they form an accepting run. A matching lower bound can be given by 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 clause 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.

Decidability of checking *universality, inclusion* and *equivalence* follows from the effectiveness of the closure under complementation and intersection, and the decidability of emptiness, which is known to be PSPACE-COMPLETE [2, 3]. This yields a double exponential upper bound. The best known lower bound is single exponential and comes from VPLs.

The following theorem summarizes the above results.

Theorem 6. *The class of k-scoped* MVPL *over $\widetilde{\Sigma}_n$ is closed under language union, intersection and complementation with respect to $\mathit{Scoped}(\widetilde{\Sigma}_n, k)$. The Membership Problem for k-scoped* MVPL *is* NP-COMPLETE. *The Universality, Inclusion and Equivalence problems can be decided in* 2EXPTIME.

5. Sequentialization and Parikh's Theorem

Sequentialization. We show that when restricting to k-scoped words, we can mimic the computations of an n-stack MVPA A using only one stack (*sequentialization*). We start by describing how the input word is rearranged.

Fix a k-scoped word w over $\widetilde{\Sigma}_n$, and let $\chi = \langle w_i \rangle_{i \in [d]}$ be a k-scoped splitting of w. For $h \in [n]$, denote with $\chi^h = \langle w_i^h \rangle_{i \in [x_h]}$ the h-projection of χ. Since χ is k-scoped, χ^h is k-bounded and let $\xi^h = \langle v_i^h \rangle_{y_h}$ be a k-bounding splitting for χ^h.

We define a total order $\preceq_w$ over all the v_j^h according to the position of their first symbol in w, that is, $v_j^h \preceq_w v_{j'}^{h'}$ iff $r \leq s$ where r is the position in w of the first symbol of v_j^h and s is that of the first symbol of $v_{j'}^{h'}$.

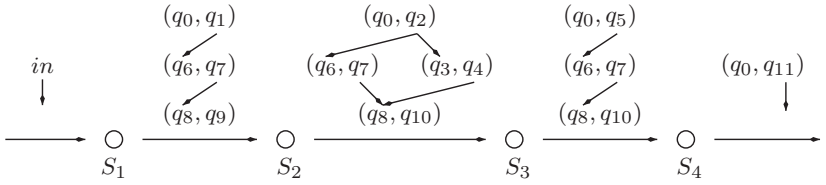

Fig. 4. A_{seq} run for the running example.

We denote with $\pi_\chi(w)$ the concatenation of all the v_j^h in the ordering given by $\preceq_w$. For example, consider the word u and the k-scoped splitting ξ resulting from the example of Fig. 3. The word $\pi_\xi(u)$ is $u_1u_7u_9.u_2u_4u_{10}.u_3u_5.u_6u_8u_{11}$.

We define $\pi(w)$ as the set of all words $\pi_\chi(w)$ for any possible k-scoped splitting χ of w. We extend π to languages in the usual way.

We show that L is a k-scoped MVPL iff $\pi(L)$ is VPL (all calls and returns are interpreted as calls and returns of a same unique stack). In fact, since ξ^h is k-bounding for χ^h, we get to process consecutively each set of (at most k) contexts that share the same stack content. Thus, when entering the next portion, we can start as the stack were empty (all that is left in the stack is not needed any more). Moreover, all the stack-h contexts, for a given h, occur in the same order as in w. Thus, we can process them by using A_k^h, and construct the VPA A_{seq} starting from the cross product of A_k^h for $h \in [n]$.

A second main feature of π is that when reading an input word $v \in \pi(w)$, we can reconstruct w by using only bounded memory: at any time, we keep a summary of each already processed portion of w (i.e., starting and ending states of corresponding portions of an A run) and a partial order of all such portions.

Observe that while parsing v, we know neither w nor a run on it. We reconstruct them on-the-fly by making nondeterministic guesses and ruling out the wrong guesses as soon as we realize it. For simplicity, we illustrate our idea on our running example by assuming that we know instead the run and the word u. We refer to in Fig. 4 and for $i \in [4]$, S_i is as in Fig. 3. The input word to A_{seq} is $u_1u_7u_9.u_2u_4u_{10}.u_3u_5.u_6u_8u_{11} \in \pi(u)$. After parsing $u_1u_7u_9$, we compute S_1 according to the considered run, and store the partial order shown on the edge from S_1 to S_2. Now after parsing $v_2 = u_2u_4u_{10}$, we compute S_2. Since u_2 follows u_1 and u_{10} follows u_9, by the ordering in v_2 and the fact that u_7 and u_4 are not consecutive, we get the partial order labeling the edge from S_2 to S_3, and so on.

We succeed in reconstructing w iff in the end the maintained partial order collapses to just one summary (i.e., all the portions get connected). To keep the size of the stored partial order small, when the computation of a stack-h d-SV I starts, we ensure that all the previously computed stack-h SV's are entirely *hidden* in the summaries (i.e., each pair of such SV's has been glued on both sides to other pairs) except for at most the second component of the last pair. In this case, we impose that the first pair of I starts with such a second component (as for S_3 and S_4 in the running example).

This is indeed sufficient to accept all the words in $\pi(w)$ for a k-scoped word w. In fact, assume as input $\pi_\chi(w)$ for a k-scoped splitting χ, and also the notation given in the beginning of this subsection. By definition of π_χ, the v_i^h's forming the k-bounding splittings ξ^h, for $h \in [n]$ and $i \in [y_h]$, are ordered according to their first contexts. Thus, when processing a v_i^h all the $v_{i'}^{h'} \preceq_w v_i^h$ have been already read by A_{seq} and hence, the first contexts of all such $v_{i'}^{h'}$ belong to a prefix of w that has been already processed. Therefore, the computed partial orders can be restricted to those that have a unique pair that precedes all the others. Moreover, for each $v_{i'}^h \preceq_w v_i^h$, since ξ^h is a splitting of the concatenation of the stack-h contexts of w (in the order they appear in w), also all the contexts of $v_{i'}^h$ must be in the already processed prefix of w. Hence, the number of pairs in the considered class of partial orders is bounded by $(n-1)(k-1)+1$.

Intuitively, A_{seq} mimics the cross product of $A_k^1, \ldots, A_k^n$ and maintains the partial orders of the summaries (pairs of control states) of the starting MVPA A as observed above. The partial orders are updated at any context switch by using nondeterminism to guess how the next context is related to the summaries in the partial order. The nondeterminism of each A_k^h is reduced by ruling out all the moves that are not consistent with the stored partial order. The accepting states of A_{seq} are those with a partial order that is a single pair.

We omit the formal definition of A_{seq}. We only observe further that since the input of each A_k^h is over $\Sigma_h \cup \bar{\Sigma}_h$, we first need to transform them into corresponding VPAs B_k^h over Σ_h. This is done by modifying A_k^h such that the starting symbol of each context is now guessed nondeterministically (which is quite standard). Thus, denoting as B_k^h the resulting VPAs, we get that $\bar{w}_1 \ldots \bar{w}_d \in L(A_k^h)$ iff $w_1 \ldots w_d \in L(B_k^h)$. Also, the call-return alphabet of A_{seq} is $\widetilde{\Sigma}_{seq}$ where $\Sigma_{seq}^c = \bigcup_{h\in[n]} \Sigma_h^c$, $\Sigma_{seq}^r = \bigcup_{h\in[n]} \Sigma_h^r$ and $\Sigma_{seq}^{int} = \Sigma^{int}$ (recall that the alphabets from $\widetilde{\Sigma}_n$ are pairwise disjoint). The following lemma holds:

Lemma 7. *For an* MVPA A *and a* k*-scope word* w *over* $\widetilde{\Sigma}_n$, $\pi(L(A)) = L(A_{seq})$. *The size of* A_{seq} *is exponential in* k *and* n, *and polynomial in the size of* A.

Note that the above lemma reduces the emptiness problem for SMVPL to checking the emptiness for VPAs, and thus provides an alternative decision algorithm for this problem.

Parikh's theorem. The Parikh mapping associates a word with the vector of the numbers of the occurrences of each symbol in the word. Formally, the Parikh image of a word w, over the alphabet $\{a_1, \ldots, a_\ell\}$, is $\Phi(w) = (\#a_1, \ldots, \#a_\ell)$ where $\#a_i$ is the number of occurrences of a_i in w. This mapping extends to languages in the natural way: $\Phi(L) = \{\Phi(w) | w \in L\}$.

Parikh's theorem [37] states that for each context-free language L a regular language L' can be effectively found such that $\Phi(L) = \Phi(L')$. Lemma 7 gives an effective way to translate a k-scoped MVPL to a VPL, and thus we get:

Theorem 8. *For every k-scoped* MVPL L *over* $\widetilde{\Sigma}_n$*, there is a regular language* L' *over* Σ *such that* $\Phi(L') = \Phi(L)$*. Moreover,* L' *can be effectively computed.*

6. Expressiveness

Comparisons among classes of MVPLs. We start recalling the main classes of MVPLs from the literature, and then compare them with the class of SMVPL.

In the following definitions, let $\widetilde{\Sigma}_n$ be a call return alphabet.

Bounded number of contexts/rounds [4,5]. A *round* over $\widetilde{\Sigma}_n$ is a word of the form $w_1 w_2 \dots w_n$ where, for each $h \in [n]$, w_h is either a stack-h context or the empty word (*empty context*). A *k-round word* over $\widetilde{\Sigma}_n$ is a word that can be obtained as the concatenation of k rounds. We denote with $Round(\widetilde{\Sigma}_n, k)$ 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 [16]. 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.

Ordered matching relations [18]. 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).

Bounded path-trees [20]. $Ptree(\widetilde{\Sigma}_n, k)$ be the set of words w over $\widetilde{\Sigma}_n$ that can be encoded into a *stack tree* (a binary tree obtained by labeling the root with the first symbol of w, and then the successor in w labels the left child, unless it is a matched return, and in this case it labels the right child of the matching call) and starting from the root all the nodes can be visited s.t. the first occurrences of each node matches the linear order of w and each node is not visited more than k times.

Classes of languages. A language L is a *round* MVPL (RMVPL) if there exist $k, n > 0$, an n-stack call-return alphabet $\widetilde{\Sigma}_n$, and an MVPA A over $\widetilde{\Sigma}_n$ such that $L = Round(\widetilde{\Sigma}_n, k) \cap L(A)$. Analogously, we define *phase* MVPL (PMVPL), *ordered* MVPL (OMVPL), and *path-tree* MVPL (TMVPL).

Comparisons. 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:

$$L_1 = \{a^i b^j c^i d^j (ab)^h \mid i, j, h > 0\}, \qquad L_2 = \{(a^i b^j c^i d^j)^h \mid i, j, h > 0\},$$
$$L_3 = \{a^i b^j c^h d^j c^{i-h} \mid i > h > 0, j > 0\}, \quad L_4 = \{(ab)^i c^i d^i \mid i > 0\},$$
$$L_5 = \{a^i b^j c^i d^j ab\,(ca\,db)^h \mid i > h > 0, j > 0\}.$$

For all the above languages, first assume the call-return alphabet $\widetilde{\Sigma}_2$, where $\Sigma^c_1 = \{a\}$, $\Sigma^c_2 = \{b\}$, $\Sigma^r_1 = \{c\}$, $\Sigma^r_2 = \{d\}$, and $\Sigma^{int} = \emptyset$.

For each $i \in [5]$, we can easily construct an MVPA over $\widetilde{\Sigma}_2$ that accepts L_i (the MVPA must just use the stacks to match calls and returns).

We observe the following: words in L_1 are 2-scoped, 2-phase and ordered, thus $L_1 \in$ PMVPL $\cap$ SMVPL $\cap$ OMVPL; those in L_2 are 2-scoped and ordered, thus $L_2 \in$ SMVPL $\cap$ OMVPL; those in L_3 are the concatenation of three rounds, thus $L_3 \in$ RMVPL; those in L_4 are 2-phase and ordered, thus $L_4 \in$ PMVPL $\cap$ OMVPL; finally, those in L_5 are 2-scoped, thus $L_5 \in$ SMVPL.

Now, observe that, for each $i \in [5]$, an MVPA accepting L_i must have at least two stacks and a call-return alphabet in which symbols a and c are respectively call and return of the same stack, while b and d are respectively call and return of another stack. It is easy to see that if this is not the case, an MVPA could not recognize L_i, for any $i \in [5]$.

For any call-return alphabet $\widetilde{\Sigma}_n$ that is consistent with the above observation, we get that L_1 is not contained into $\mathit{Round}(\widetilde{\Sigma}_n, k)$, for any fixed k (because of the suffix $(ab)^h$). Thus, $L_1 \notin$ RMVPL. Analogously, we get that $L_2 \notin$ PMVPL since there is no bound on the number of phases of L_2 words. Words in L_3 are not ordered since after reading the prefixes $a^i b^j$ and $a^i b^j c^h$ both stacks are not empty, therefore for any stack ordering in one of the two cases we cannot pop on the next return. Thus $L_3 \notin$ OMVPL, and we get also that the union of L_2 and L_3, which is a language in $\mathit{Ptree}(\widetilde{\Sigma}_n, k)$ (by the closure under union of TMVPL [20]), is neither ordered nor k-phase for any k. Words in L_4 are not in $\mathit{Scoped}(\widetilde{\Sigma}_n, k)$, for any k, in fact the prefix $(ab)^i$ determines that the input has at least i context switches (recall that a and b must be symbols of different stacks) and hence $L_4 \notin$ SMVPL. Finally, we can argue that $L_5 \notin$ TMVPL. For this, we observe that from the definition of stack tree, in the stack tree of a word $w = a^i b^j c^i d^j ab\,(ca\,db)^h \in L_5$, there is a subtree corresponding to the suffix $ab\,(ca\,db)^h$ that has exactly two paths: one labeled with $ab(db)^h$ (a is the label of the root of this subtree) and the other with $a(ca)^h$. Thus in the visit of all the nodes according to the ordering in w, we need to visit the root of such subtree exactly $2h + 1$ times, and therefore there is no bound on the number of visits on the stack trees for L_5.

Therefore, since PMVPL $\cup$ OMVPL $\subseteq$TMVPL holds [20], we get:

Theorem 9.
(1) *R*MVPL $\subset$ *S*MVPL $\cap$ *P*MVPL.
(2) *T*MVPL $\supset$ *P*MVPL $\cup$ *O*MVPL.
(3) *R*MVPL *and O*MVPL *are incomparable.*
(4) *S*MVPL *and T*MVPL *are incomparable.*
(5) *S*MVPL*, O*MVPL*, P*MVPL *are pairwise incomparable.*

A logical characterization of SMVPL. We show that Monadic Second Order Logic (MSO_μ) on scoped words has the same expressiveness as scoped MVPA. Here a word $w \in \Sigma^*$ is a structure over the universe $\{1, \ldots, |w|\}$. The logic has in its signature a predicate P_a for each $a \in \Sigma$ where $P_a(i)$ is true if the i-th symbol of w is a, and n predicates μ_h with $h \in [n]$, such that $\mu_h(i, j)$ holds true iff $i \sim_h j$.

We convert MSO sentences to automata using standard techniques that rely on the closure under Boolean operations and projection (see [38]).

More formally, let us fix a countable infinite set of first-order variables $x, y, \ldots$ and a countable infinite set of monadic second-order (set) variables $X, Y, \ldots$. The *monadic second-order logic* (MSO_μ) over $\widetilde{\Sigma}_n$ is defined by the following grammar:

$$\varphi := P_a(x) \mid x \in X \mid x \leq y \mid \mu_h(x, y) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $h \in [n]$, $a \in \Sigma$, x, y are a first-order variables and X is a set variable.

The models are words over Σ with the natural semantics on the structure for words. Each first order variable is associated with a position of w, and the second-order variables range over sets of positions. A *sentence* is a formula with no free variables. We define $L_k(\varphi)$ to be the set of all words of $\mathit{Scoped}(\widetilde{\Sigma}_n, k)$ that satisfy a sentence φ.

We exploit the standard technique to convert MSO sentences to automata (see [38]) to prove the following theorem. We want to remark that this construction heavily relays on the fact that the automata are closed under Boolean operations and projection.

We get the following:

Theorem 10. *Let k, n be two positive integers, $\widetilde{\Sigma}_n$ be a call-return alphabet, and $L \subseteq \mathit{Scoped}(\widetilde{\Sigma}_n, k)$. L is k-scoped* Mvpl *iff there is an MSO_μ sentence φ over $\widetilde{\Sigma}_n$ with $L_k(\varphi) = L$.*

Proof. The proof from MSO_μ to scoped Mvpa is done by structural induction on φ. Although φ is a sentence, its sub formulas may contain free variables. For a sub formula ψ of φ, we denote it with $\psi(x_1, \ldots, x_t, X_1, \ldots, X_s)$ where $V = \{x_1, \ldots, x_t, X_1, \ldots, X_s\}$ is the set of all free variables of ψ. We consider the extended alphabet Σ^V that consists of all pairs (a, Z) such that $a \in \Sigma$ and Z is a map from V to $\{0, 1\}$ used to associate free variables with positions. We assume that words in $(\Sigma^V)^*$ have the property that first order variable are associated with exactly one position in the word. Given ψ, we define a Mvpa A_ψ that accepts all words in $(\Sigma^V)^*$ such that (1) the underlying word w is in $\mathit{Scoped}(\widetilde{\Sigma}_n, k)$, and (2) the free variables associated with the corresponding positions make ψ true.

The property that first order variables appear exactly once along the word can be imposed using a finite state Mvpa. Similarly, we can easily design Mvpa for $P_a(x)$, $x \in X$ and $x \leq y$. For sub formula $\mu_h(x, y)$ we design a Mvpa that stores the input symbol (a, Z) on the stack h whenever a is a call of stack h. Thus, when reading a symbol (a', Z') with $Z'(y) = 1$ we check that the symbol on stack h, say (a'', Z''), is such that $Z''(x) = 1$.

Disjunction and negation are easily treated as scoped Mvpa are closed under union and negation. Similarly, existential quantifiers can be treated by projecting out that variables from the valuation function.

For the opposite direction of the proof, we again apply a standard technique. Let $\{q_1, \ldots, q_\ell\}$ be the set of all states of the Mvpa. We design a formula of the

Table 1. Summary of the main results on MVPLs (new results are in bold). In the table, NP-C stands for NP-COMPLETE, and so on.

	Closure properties				Decision Problems		
	$\cup$	$\cap$	Compl.	Determin.	Membership	Emptiness	Un./ Eq./Incl.
VPL [1]	Yes	Yes	Yes	Yes	PTIME-C	PTIME-C	EXPTIME-C
CFL	Yes	No	No	No	PTIME-C	PTIME-C	Undecidable
RMVPL [5]	Yes	Yes	Yes	Yes	NP	NP-C	2EXPTIME
SMVPL	**Yes**	**Yes**	**Yes**	**Yes**	**NP-c**	PSPACE-C	**2Exptime**
TMVPL [20]	Yes	Yes	Yes	No	NP-C	ETIME-C	2EXPTIME
PMVPL [16, 20]	Yes	Yes	Yes	No	NP-C	2ETIME-C	3EXPTIME
OMVPL [20, 39]	Yes	Yes	Yes	No	NP-C	2ETIME-C	3EXPTIME
CSL	Yes	Yes	Yes	Unknown	NLINSPACE	Undecidable	Undecidable

form $\exists X_1, \ldots, X_\ell.\eta$, where X_i is the set of all positions where the run is in state q_i, and η ensures that this labelling indeed forms a run, by imposing the initial and final condition and making sure that states associated with consecutive positions are related by a transition of the automaton. □

7. Conclusions

We have shown that the class of SMVPL is closed under all the Boolean operations, it has a logical characterization, the Parikh theorem holds and the main decision problems are decidable. Table 1 summarizes the results on the closure properties and decision problems. Moreover, the class of scoped MVPA is determinizable and sequentializable (sequentialization has shown to be an effective technique for model-checking concurrent programs and is implemented in tools as CSeq, Microsoft Corral).

Our sequentialization construction allows us to extend to a larger class of languages the results from [11,22] on sequentialization and tree decompositions. In [11] the definition of scoped words from [3] is used: only computations under a bounded number of round-robin scheduling are allowed and thus only scoped words with a bounded number of contexts between any two consecutive contexts of a same stack can be captured. For example, consider the language $L = x(yz)^*t$ for the call-return alphabet $\widetilde{\Sigma}_3$ where $\Sigma_1^c = \{x\}$, $\Sigma_2^c = \{y\}$, $\Sigma_3^c = \{z\}$, $\Sigma_1^r = \{t\}$, and the remaining alphabets are empty. Then, L is SMVPL (precisely 2-scoped) but it is not captured by the definition given in [3] (unboundedly many rounds would be needed). The sequentialization construction also suggests a tree-decomposition of the multiply nested words corresponding to scoped words with bags of $O(nk)$ size. Thus showing that also for the larger class of languages considered here, the class of graphs defined by scoped words (and thus computations of scoped MPA) has bounded tree-width. Therefore, along with our MSO characterization, we get that the MSO theory of scoped words is decidable.

References

[1] R. Alur and P. Madhusudan, "Visibly pushdown languages," in *STOC*, pp. 202–211, ACM, 2004.

[2] S. La Torre and M. Napoli, "A temporal logic for multi-threaded programs," in *IFIP TCS*, vol. 7604 of *LNCS*, pp. 225–239, Springer, 2012.
[3] S. La Torre and M. Napoli, "Reachability of multistack pushdown systems with scope-bounded matching relations," in *CONCUR*, vol. 6901 of *LNCS*, pp. 203–218, Springer, 2011.
[4] S. Qadeer and J. Rehof, "Context-bounded model checking of concurrent software," in *TACAS*, vol. 3440 of *LNCS*, pp. 93–107, Springer, 2005.
[5] S. La Torre, P. Madhusudan, and G. Parlato, "The language theory of bounded context-switching," in *LATIN*, vol. 6034 of *LNCS*, pp. 96–107, Springer, 2010.
[6] A. Lal and T. W. Reps, "Reducing concurrent analysis under a context bound to sequential analysis," *Formal Methods in Syst. Design*, vol. 35, no. 1, pp. 73–97, 2009.
[7] M. Emmi, S. Qadeer, and Z. Rakamaric, "Delay-bounded scheduling," in *POPL* [40], pp. 411–422.
[8] O. Inverso, E. Tomasco, B. Fischer, S. La Torre, and G. Parlato, "Bounded model checking of multi-threaded C programs via lazy sequentialization," in *CAV*, vol. 8559 of *LNCS*, pp. 585–602, Springer, 2014.
[9] S. Chaki, A. Gurfinkel, and O. Strichman, "Time-bounded analysis of real-time systems," in *FMCAD*, pp. 72–80, FMCAD Inc., 2011.
[10] S. La Torre, M. Napoli, and G. Parlato, "Scope-bounded pushdown languages," in *DLT*, vol. 8633 of *LNCS*, pp. 116–128, Springer, 2014.
[11] S. La Torre and G. Parlato, "Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width," in *FSTTCS*, vol. 18 of *LIPIcs*, pp. 173–184, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
[12] M. F. Atig, A. Bouajjani, K. N. Kumar, and P. Saivasan, "Linear-time model-checking for multithreaded programs under scope-bounding," in *ATVA*, vol. 7561 of *LNCS*, pp. 152–166, Springer, 2012.
[13] R. Mennicke, "Model checking concurrent recursive programs using temporal logics," in *MFCS* [41], pp. 438–450.
[14] A. Cyriac, P. Gastin, and K. N. Kumar, "MSO decidability of multi-pushdown systems via split-width," in *CONCUR*, vol. 7454 of *LNCS*, pp. 547–561, Springer, 2012.
[15] M. Hague, "Saturation of concurrent collapsible pushdown systems," in *FSTTCS*, vol. 24 of *LIPIcs*, pp. 313–325, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
[16] S. La Torre, P. Madhusudan, and G. Parlato, "A robust class of context-sensitive languages," in *LICS*, pp. 161–170, IEEE Computer Society, 2007.
[17] S. La Torre, P. Madhusudan, and G. Parlato, "An infinite automaton characterization of double exponential time," in *CSL*, vol. 5213 of *LNCS*, pp. 33–48, Springer, 2008.
[18] L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghizzi, "Multi-pushdown languages and grammars," *Int. J. Found. Comput. Sci.*, vol. 7(3), pp. 253–292, 1996.
[19] D. Carotenuto, A. Murano, and A. Peron, "2-visibly pushdown automata," in *DLT*, vol. 4588 of *LNCS*, pp. 132–144, Springer, 2007.
[20] S. La Torre, M. Napoli, and G. Parlato, "A unifying approach for multistack pushdown automata," in *MFCS* [41], pp. 377–389.
[21] A. Seth, "Global reachability in bounded phase multi-stack pushdown systems," in *CAV*, vol. 6174 of *LNCS*, pp. 615–628, Springer, 2010.
[22] P. Madhusudan and G. Parlato, "The tree width of auxiliary storage," in *POPL* [40], pp. 283–294.
[23] B. Bollig, D. Kuske, and R. Mennicke, "The complexity of model checking multi-stack systems," in *LICS*, pp. 163–172, IEEE Computer Society, 2013.

[24] M. F. Atig, K. N. Kumar, and P. Saivasan, "Adjacent ordered multi-pushdown systems," in *Developments in Language Theory*, vol. 7907 of *LNCS*, pp. 58–69, Springer, 2013.

[25] P. A. Abdulla, M. F. Atig, O. Rezine, and J. Stenman, "Budget-bounded model-checking pushdown systems," *Formal Methods in Syst. Design*, vol. 45, no. 2, pp. 273–301, 2014.

[26] M. F. Atig, K. N. Kumar, and P. Saivasan, "Adjacent ordered multi-pushdown systems," *Int. J. Found. Comput. Sci.*, vol. 25, no. 8, pp. 1083–1096, 2014.

[27] S. La Torre, P. Madhusudan, and G. Parlato, "Reducing context-bounded concurrent reachability to sequential reachability," in *CAV*, vol. 5643 of *LNCS*, pp. 477–492, Springer, 2009.

[28] A. Bouajjani, M. Emmi, and G. Parlato, "On sequentializing concurrent programs," in *SAS*, vol. 6887 of *LNCS*, pp. 129–145, Springer, 2011.

[29] T. L. Nguyen, B. Fischer, S. La Torre, and G. Parlato, "Unbounded lazy-cseq: A lazy sequentialization tool for C programs with unbounded context switches - (competition contribution)," in *TACAS* [42], pp. 461–463.

[30] E. Tomasco, O. Inverso, B. Fischer, S. La Torre, and G. Parlato, "Verifying concurrent programs by memory unwinding," in *TACAS* [42], pp. 551–565.

[31] O. Inverso, T. L. Nguyen, B. Fischer, S. La Torre, and G. Parlato, "Lazy-CSeq: A context-bounded model checking tool for multi-threaded c-programs," in *ASE*, 2015.

[32] M. Hague, "Parameterised pushdown systems with non-atomic writes," in *FSTTCS*, vol. 13 of *LIPIcs*, pp. 457–468, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[33] J. Esparza, P. Ganty, and R. Majumdar, "Parameterized verification of asynchronous shared-memory systems," in *CAV*, vol. 8044 of *LNCS*, pp. 124–140, Springer, 2013.

[34] S. La Torre, A. Muscholl, and I. Walukiewicz, "Safety of parametrized asynchronous shared-memory systems is almost always decidable," in *CONCUR*, LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[35] K. Mehlhorn, "Pebbling moutain ranges and its application of DCFL-recognition," in *ICALP*, vol. 85 of *LNCS*, pp. 422–435, Springer, 1980.

[36] A. Okhotin, X. Piao, and K. Salomaa, "Descriptional complexity of input-driven pushdown automata," in *Languages Alive* (H. Bordihn, M. Kutrib, and B. Truthe, eds.), vol. 7300 of *LNCS*, pp. 186–206, Springer, 2012.

[37] R. Parikh, "On context-free languages," *J. ACM*, vol. 13, no. 4, pp. 570–581, 1966.

[38] W. Thomas, "Languages, automata, and logic," in *Handbook of Formal Languages*, vol. 3, pp. 389–455, Springer-Verlag New York, Inc., 1997.

[39] M. F. Atig, B. Bollig, and P. Habermehl, "Emptiness of multi-pushdown automata is 2Etime-complete," in *DLT*, vol. 5257 of *LNCS*, pp. 121–133, Springer, 2008.

[40] *Proceedings of the* 38*th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL* 2011, ACM, 2011.

[41] *Proceedings of* 39*th International Symposium on Mathematical Foundations of Computer Science, MFCS* 2014, vol. 8634 of *LNCS*, Springer, 2014.

[42] *Proceedings of* 21*st International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS* 2015, vol. 9035 of *LNCS*, Springer, 2015.