Scope-Bounded Pushdown Languages

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

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

Abstract. We 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 contribute to 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 several interesting consequences, such as, the closure under complement, the decidability of universality, inclusion and equality, and a Parikh theorem. We also give a logical characterization and compare the expressiveness of the scope-bounded restriction with MVPA classes from the literature.

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 MPAs are equivalent to Turing machines already with two stacks.

In this paper, we study the formal language theory of MVPA restricted to *scoped computations* [13]: for a positive integer, a computation is k-scoped if for each stack i, each popped symbol was pushed within the last k contexts of i (a context is a continuous portion of the computation where only one stack is used). The notion of scope-bounded computations was introduced in [12] to

A.M. Shur and M.V. Volkov (Eds.): DLT 2014, LNCS 8633, pp. 116–128, 2014.

extend the analysis of MPA to unboundedly many context switches. The original notion of scope-bounded is significantly less expressive than the one used in this paper. The notion of scoped computations naturally extends to infinite words and temporal logic model checking [13,3]. Also, global reachability was solved for concurrent collapsible pushdown automata restricted to scoped computations [8].

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 scope-bounded 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 [9].

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. 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 [13], 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: In the literature several classes of MVPL have been studied: *phase* [10,11], *ordered* [6,7], and *path-tree* [14] are not determinizable and incomparable with scoped MVPL. The class of *round* MVPL [9], which is based on the notion of bounded-context switching [19], has the same properties as scoped MVPL (checking emptiness is NP-COMPLETE) but it is strictly included in it.

Parikh theorem was originally given for context-free languages in [18]. Visibility of stack operations was first introduced for input-driven pushdown automata [22] (see also [17] and references therein). A fixed-point algorithm for the reachability problem and a sequentialization are given in [15] for MPA under the restriction from [12]. The bounded context-switching restriction was proposed in [19] for under-approximate analysis of multi-threaded programs. More work on decision problems is done in [20,5] for phase MPA and ordered MPA [4].

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, for an n-stack call-return alphabet $\widetilde{\Sigma}_n$, we let $\Sigma_h = \Sigma_h^c \cup \Sigma_h^r \cup \Sigma^{int}$, $\Sigma^c = \bigcup_{h \in [n]} \Sigma_h^c$, $\Sigma^r = \bigcup_{h \in [n]} \Sigma_h^r$ and with $\Sigma = \Sigma^{int} \cup \Sigma^r \cup \Sigma^c$.

For a word $w = a_1 \ldots a_m$ over $\widetilde{\Sigma}_n$, 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* $\sim_h$ 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 (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. Fix a call-return alphabet $\widetilde{\Sigma}_n$.

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 $\alpha = \langle q, \sigma_1, \ldots, \sigma_n \rangle$, 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* $\langle q, \sigma_1, \ldots, \sigma_n \rangle \xrightarrow{a}_A \langle q', \sigma_1', \ldots, \sigma_n' \rangle$ is such that one of the following holds:

[Push] $a \in \Sigma_h^c$, $\exists \gamma \in \Gamma \setminus \{\perp\}$ 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] $a \in \Sigma_h^r$, $\exists \gamma \in \Gamma$ such that $(q, a, \gamma, q') \in \delta$, $\sigma_i' = \sigma_i$ for every $i \in ([n] \setminus \{h\})$, and either $\gamma \neq \perp$ and $\sigma_h = \gamma \cdot \sigma_h'$, or $\gamma = \sigma_h = \sigma_h' = \perp$.

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

For a word $w = a_1 \ldots a_m$ in 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 $i \in [m]$. 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 [12,13]. A *stack-h context* is a word in Σ_h^+. We say that w has at most k *maximal* contexts of stack h if $w \in \Sigma_h^* (\Sigma_{\neq h}^* \Sigma_h^*)^{k-1}$ where $\Sigma_{\neq h} = \bigcup_{h' \neq h} \Sigma_{h'}$.

For a word $w = a_1 \dots a_m \in \Sigma^*$ 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. $i \sim_h j$, $w[i,j]$ has at most k maximal contexts of stack 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 $\nu_1 = a^3\, bd^2\, c^2 a\, b^3\, c^2$. Fig. 1 illustrates its splitting into contexts and the matching relations 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.

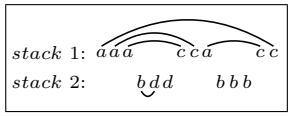

Fig. 1. A 3-scoped word

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 over $\widetilde{\Sigma}_n$ 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*, 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. $w = w_1 \dots w_d$, w_i is non-empty for $i \in [d]$ and $w_1 \dots w_i : w_{i+1} \dots w_d$ is a ($\sim_h$-*consistent*) cut for $i \in [d-1]$.

A *context-splitting* of w is a splitting $\langle w_i \rangle_{i \in [d]}$ where w_i is a stack-h_i context for $i \in [d]$. 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*.

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 $h \in [n]$, the h-projection of χ is k-bounded.

Consider a sample word $\nu_2 = a^2bd^2c^2a^2b^3c^2ad^2bc^2$. Fig. 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 $bd^2\, b^3\, d^2b$ 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.

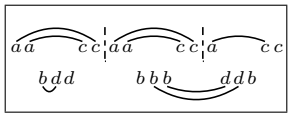

Fig. 2. k-scoped splitting

Lemma 1. *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 [9]. 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, d is omitted 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 $\sigma_0 = \perp$. 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 a 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 now define a VPA A^h_k that if the input is an encoding of a k-bounded tuple χ of stack-h contexts then it *computes* all the stack-h k-scoped SV's of A witnessed by χ. Essentially, A^h_k nondeterministically guesses any k-bounding splitting for χ and for each resulting portion, say formed by $d \leq k$ contexts, it computes a corresponding d-SV while mimicking the behavior of A^h.

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 $u = a_1 a_2 \dots a_d$, we denote with $\bar{u}$ the word $\bar{a}_1 a_2 \dots a_d$. We encode a tuple of stack-h contexts $u_1, \dots, u_m$ as $\bar{u}_1 \bar{u}_2 \dots \bar{u}_m$. The new symbols $\bar{a}$ are interpreted as a when mimicking the moves of A^h.

By assuming the input $\bar{u}_1\bar{u}_2\ldots\bar{u}_m$, 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 $\bar{a}\in\bar{\Sigma}_h$), appending a new pair to the stored SV I must not be allowed if I already contains k pairs. By Lemma 1, 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 (a guessed cut 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.

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).

Lemma 2. *I is a stack-h k-scoped switching vector of A iff $I\in\mathcal{I}^h_k(w)$ for some $w\in(\Sigma_h\cup\bar{\Sigma}_h)^*$.*

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. According to ρ, 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 $\bar{u}_6\bar{u}_8\bar{u}_{11}$. The 3-SV's computed on v_1 and v_2 are respectively S_1 and S_2 (Fig. 3).

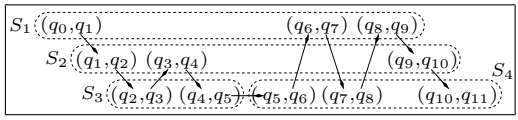

Fig. 3. Sample switching vectors and 3-scoped switching mask

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 1 and 2, the following holds:

Lemma 3. *Suppose that $A = (Q, Q_0, \Gamma, \delta, F)$ is an* MVPA *over $\widetilde{\Sigma}_n$ and $w \in Scoped(\widetilde{\Sigma}_n, k)$. Then $w \in L(A)$ 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, Sequentialization and Parikh Theorem

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

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 k-scoped splitting of w. Thus, A^D accepts w iff it constructs a switching mask as in Lemma 3, 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 $(h, \hat{q_1}, \ldots, \hat{q_n}, \mathcal{M})$, 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 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 $(h, \hat{q_1}, \ldots, \hat{q_n}, \mathcal{M})$ s.t. there is $(I_1, \ldots I_n) \in \mathcal{M}$ with $cur(I_h) \in F$.

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^h_k to define $\mathcal{I}^h_k(z)$. 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 3:

Theorem 1. *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.*

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 $\chi^h = \langle w^h_i \rangle_{i \in [x_h]}$ the h-projection of χ. Since χ is k-scoped, χ^h is k-bounded and let $\xi^h = \langle v^h_i \rangle_{y_h}$ be a k-bounding splitting for χ^h.

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

We denote with $\pi_\chi(w)$ the concatenation of all the v^h_j 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 the 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^h_k, and construct the Vpa A_{seq} starting from the cross product of A^h_k 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).

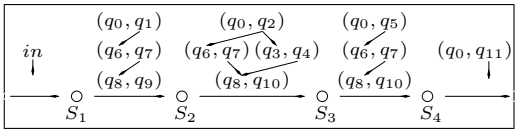

Fig. 4. A_{seq} run for the running example

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

$\Sigma^c_{seq} = \bigcup_{h\in[n]} \Sigma^c_h$, $\Sigma^r_{seq} = \bigcup_{h\in[n]} \Sigma^r_h$ and $\Sigma^{int}_{seq} = \Sigma^{int}$ (recall that the alphabets from $\widetilde{\Sigma}_n$ are pairwise disjoint). The following lemma holds:

Lemma 4. *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.

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 [18] states that for each context-free language L a regular language L' can be effectively found such that $\Phi(L) = \Phi(L')$. Lemma 4 gives an effective way to translate a k-scoped Mvpl to a Vpl, and thus we get:

Theorem 2. *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.*

5 Closure Properties, Decision Problems and Expressiveness

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 1).

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 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 [12,13]. Note that Lemma 4 reduces this problem to checking the emptiness for Vpas, and thus provides an alternative decision algorithm. Decidability of *universality, inclusion* and *equivalence* follows from the effectiveness of the closure under

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	Univ./ Equiv./Incl.
VPL [1]	Yes	Yes	Yes	Yes	PTIME-C	PTIME-C	EXPTIME-C
CFL	Yes	No	No	No	PTIME-C	PTIME-C	Undecidable
RMVPL [9]	Yes	Yes	Yes	Yes	NP	NP-C	2EXPTIME
SMVPL	**Yes**	**Yes**	**Yes**	**Yes**	**NP-c**	PSPACE-C	**2Exptime**
TMVPL [14]	Yes	Yes	Yes	No	NP-C	ETIME-C	2EXPTIME
PMVPL [10,14]	Yes	Yes	Yes	No	NP-C	2ETIME-C	3EXPTIME
OMVPL [2,14]	Yes	Yes	Yes	No	NP-C	2ETIME-C	3EXPTIME
CSL	Yes	Yes	Yes	Unknown	NLINSPACE	Undecidable	Undecidable

complementation and intersection, and the decidability of emptiness. This yields a double exponential upper bound. The best known lower bound is single exponential and comes from VPLs.

Comparisons with Known MVPL Classes. The class of SMVPL is incomparable with the main classes of MVPLs from the literature. In particular, we have compared it to RMVPL [19,9] (restricted to words with a bounded number of contexts), PMVPL [10] (restricted to words with a bounded number of phases where in each phase only the returns from one stack can occur), OMVPL [6,7,14] (restricted to words where a matched return of a stack i cannot occur between matching calls and returns of any stack j, for $j < i$), and PMVPL [14] (restricted to words with a bounded tree decomposition of the form of a stack tree).

Theorem 3. *1)* RMVPL $\subset$ SMVPL $\cap$ PMVPL. *2)* TMVPL $\supset$ PMVPL $\cup$ OMVPL. *3)* RMVPL *and* OMVPL *are incomparable. 4)* SMVPL *and* TMVPL *are incomparable. 5)* SMVPL, OMVPL, PMVPL *are pairwise incomparable.*

A Logical Characterization. We show that monadic second order logic (MSO_μ) on scoped words has the same expressiveness of scoped MVPAs. 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 [21]). We get:

Theorem 4. *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 $L_k(\varphi) = L$.*

6 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 (see Table 1 for a summary of the results on closure properties and decision problems). Moreover, the class of scoped MVPAs is determinizable and sequentializable (sequentialization is an effective technique for model-checking concurrent programs, see tools as CSeq, Microsoft Corral). We extend the results from [15,16] to a larger class of languages: there, 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. Our sequentialization construction also suggests a tree-decomposition of the multiply nested words corresponding to scoped words with bags of $O(nk)$ size. Thus, also for the more expressive definition considered here, we get that the class of graphs defined by scoped words (and thus computations of scoped MPA) has bounded tree-width.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211. ACM (2004)
2. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of multi-pushdown automata is 2ETIME-complete. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 121–133. Springer, Heidelberg (2008)
3. Atig, M.F., Bouajjani, A., Kumar, K.N., Saivasan, P.: Linear-time model-checking for multithreaded programs under scope-bounding. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 152–166. Springer, Heidelberg (2012)
4. Atig, M.F., Kumar, K.N., Saivasan, P.: Adjacent ordered multi-pushdown systems. In: Béal, M.-P., Carton, O. (eds.) DLT 2013. LNCS, vol. 7907, pp. 58–69. Springer, Heidelberg (2013)
5. Bollig, B., Kuske, D., Mennicke, R.: The complexity of model checking multi-stack systems. In: LICS, pp. 163–172. IEEE Computer Society (2013)
6. Breveglieri, L., Cherubini, A., Citrini, C., Crespi-Reghizzi, S.: Multi-push-down languages and grammars. Int. J. Found. Comput. Sci. 7(3), 253–292 (1996)
7. Carotenuto, D., Murano, A., Peron, A.: 2-visibly pushdown automata. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 132–144. Springer, Heidelberg (2007)
8. Hague, M.: Saturation of Concurrent Collapsible Pushdown Systems. In: FSTTCS. LIPIcs, vol. 24, pp. 313–325 (2013)
9. La Torre, S., Madhusudan, P., Parlato, G.: The language theory of bounded context-switching. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 96–107. Springer, Heidelberg (2010)
10. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS, pp. 161–170. IEEE Computer Society (2007)
11. La Torre, S., Madhusudan, P., Parlato, G.: An infinite automaton characterization of double exponential time. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 33–48. Springer, Heidelberg (2008)
12. La Torre, S., Napoli, M.: Reachability of multistack pushdown systems with scope-bounded matching relations. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 203–218. Springer, Heidelberg (2011)
13. La Torre, S., Napoli, M.: A temporal logic for multi-threaded programs. In: Baeten, J.C.M., Ball, T., de Boer, F.S. (eds.) TCS 2012. LNCS, vol. 7604, pp. 225–239. Springer, Heidelberg (2012)

14. La Torre, S., Napoli, M., Parlato, G.: A Unifying Approach for Multistack Pushdown Automata. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part I. LNCS, vol. 8634, pp. 373–384. Springer, Heidelberg (2014), http://users.ecs.soton.ac.uk/gp4/papers/MVPL14.pdf
15. La Torre, S., Parlato, G.: Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width. In: FSTTCS. LIPIcs, vol. 18, pp. 173–184 (2012)
16. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: POPL, pp. 283–294. ACM (2011)
17. Okhotin, A., Piao, X., Salomaa, K.: Descriptional Complexity of Input-Driven Pushdown Automata. In: Bordihn, H., Kutrib, M., Truthe, B. (eds.) Languages Alive 2012. LNCS, vol. 7300, pp. 186–206. Springer, Heidelberg (2012)
18. Parikh, R.: On context-free languages. J. ACM 13(4), 570–581 (1966)
19. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
20. Seth, A.: Global reachability in bounded phase multi-stack pushdown systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 615–628. Springer, Heidelberg (2010)
21. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, vol. 3, pp. 389–455. Springer (1997)
22. Melhorn, K.: Pebbling mountain ranges and its application to DCFL-recognition. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 422–435. Springer, Heidelberg (1980)