

The Language Theory of Bounded Context-Switching

Salvatore La Torre¹, P. Madhusudan², and Gennaro Parlato^{1,2}

¹ Università degli Studi di Salerno, Italy

² University of Illinois at Urbana-Champaign, USA

Abstract. Concurrent compositions of recursive programs with finite data are a natural abstraction model for concurrent programs. Since reachability is undecidable for this class, a restricted form of reachability has become popular, where the set of states reached within k context-switches, for a fixed small constant k , is explored.

In this paper, we consider the language theory of these models. More precisely, we consider concurrent recursive programs with finite variable data domains that communicate using shared memory and work within k round-robin rounds of context-switches, and where further the stack operations are made visible (as in visibly pushdown automata). We show that the corresponding class of languages, for any fixed k , forms a robust subclass of context-sensitive languages, closed under all the Boolean operations. Also, surprisingly, the deterministic and nondeterministic versions of the automata-theoretic model define the same class of languages.

1 Introduction

Concurrent threads with recursive procedures that communicate using shared memory is a natural and attractive model of computation, as it captures most existing forms of concurrent imperative programs. While message-passing is more common in the distributed computing world where processes run on different machines, the advent of multi-core computing and the increase of event-driven and task-driven programming (where procedures on a single computer get activated on events) has led to an increased interest in shared-memory programs.

In the methodology of model-checking for program verification, a common paradigm is to analyze programs by verifying a model of it, where the model is obtained by abstracting or simplifying the *data-domain* used by a program, but preserving the control flows accurately. Many program analysis frameworks, such as data-flow analysis or predicate abstraction, fall under this category. Hence concurrent recursive programs where variables range over a finite data-domain is an attractive model of study.

The automata-theoretic model of a concurrent program with recursion and shared-memory communication is simply an automaton with multiple stacks. Since Turing machines can be simulated by an automaton with two stacks, the *emptiness* problem for these automata, and hence the model-checking problem, is undecidable.

In order to overcome this barrier, a recent proposal is to search only the space reached by these automata using a bounded number of context-switches. In other words, we view the computation as occurring in k consecutive stages (for a fixed

constant k), where in each stage only one of the concurrent threads is active. This restriction in the automaton model translates to restricting the computation to k stages, where in each stage, only one of the stacks is manipulated. It turns out that in this model the reachability problem is decidable (for any fixed k) [15].

The idea of checking and testing concurrent programs under a context-switching bound has gained attention in recent years due to several reasons. First, there is an intuitive appeal that one expects most concurrency errors to manifest themselves within a few context-switches. This has been argued fairly effectively in recent experimental studies (see [13]). Second, the model-checking problem for bounded context-switching is decidable [15], thus yielding exact algorithmic methods to solve the reachability problem. And third, checking concurrent programs under a context-bound can be done *compositionally*— we can search the state space by avoiding to build explicitly the product of local states of all automata, and instead work with only the space defined by a single thread and k copies of shared variables. The last aspect is in fact a very appealing aspect of bounded context-switching (and the least articulated) which has been exploited in recent work: model-checking tools for concurrent Boolean programs have been developed [7, 11, 16], and translations of concurrent programs to sequential programs have been developed that reduce bounded context-switching reachability to sequential reachability, even for general C-programs [8, 11]. In recent work, the above translation has even been used to verify concurrent programs under a context-bound using deductive verification tools for sequential software [10].

In this paper, we undertake a language- and automata-theoretic study of the concurrent programs with recursion under a context-switching bound. While research has so far concentrated on the computability of reachability states, we instead look at the *class of languages* accepted by these automata. In doing so, we first make the calls and returns to procedures in the concurrent program *visible*— this for sequential programs yields the class of *visibly pushdown automata* [1], which has been shown to define a robust class of context-free languages, and has led to a flurry of research (see [18] for a list of papers in this area).

We consider automata with n stacks, where an execution goes through k *round-robin schedules* of computation, i.e., a round is a sequence of exactly n contexts one for each stack and in each round the stacks alternates according to a fixed ordering (say from stack 1 through n). In a *context* for a stack i , for $i = 1, \dots, n$, the automaton can only read letters pertaining to stack i and manipulate stack i . The visibility of actions on the stack immediately implies that the class of languages is closed under union and intersection. Surprisingly, we show that the nondeterministic and deterministic versions of these automata are equivalent. The determinization construction is the key technical theorem in this paper, and crucially uses the compositional reasoning of the automata using interfaces of tuples of global states that we alluded to earlier. Determinizability gives us closure under complement as well, and hence shows that the class of automata with k -rounds of round-robin scheduling is a robust class closed under all Boolean operations.

We make several other observations regarding these automata. First, when the automata are generalized to have no bound on the number of round-robins of schedule, they are *not* determinizable. Second, it is well-known now that the emptiness of these automata are decidable, and in fact is NP-complete. Thus, for the closure under boolean operations, also universality and inclusion are decidable. Third, since these

automata define a subclass of bounded-phase multi-stack pushdown automata [9], it follows that the Parikh theorem holds for this class. Further, we show that the monadic second-order logic on n -nested words with k rounds, where the logic has n binary relations corresponding to the n nesting relations on the word, corresponds exactly to the class of languages introduced in this paper.

Finally, we can relax round-robin schedules to executions where threads are scheduled in arbitrary ways but where there are at most k context-switches. The corresponding class of language also is closed under all Boolean operations, admits determinization, and admits a decidable emptiness problem. The class of languages accepted by a multi-stack pushdown automaton with a bounded number of context-switches turns out is exactly the class accepted by multi-stack pushdown automata with a bounded number of round-robin rounds.

In conclusion, the contribution of this paper is to show that the class of multi-stack visibly pushdown automata, with either k round-robin or k context-switching schedules, forms a robust subclass of context-sensitive languages closed under all Boolean operations and is determinizable.

Related Work: The classes of multi-stack automata studied in this paper are proper subclasses of the multi-stack automata which work in a bounded number of phases, where in each phase, symbols can be pushed into all stacks but popped only from one stack [9]. Though the class of languages accepted by bounded phase multi-stack automata are closed under all Boolean operations as well, they are *not determinizable*, while the class of automata we consider here are. To the best of our knowledge, the class of automata we have introduced in this paper is the first extension of visibly pushdown automata with multiple stacks which is determinizable. [3] give a wrong determinization construction for 2-stack visibly pushdown automata, which are indeed not determinizable (even if an ordering on the usage of stacks is placed) as shown in [5]. Also it is not known if the classes of automata considered there are still closed under complement (the proof given in that paper relied on the determinizability of the model). Besides the papers we have already cited, bounded context-switching has also been exploited for systems with heaps [2], systems communicating using queues [6], and weighted pushdown systems [12].

2 Multi-Stack Pushdown Automata

In this section we give the notation and definitions to introduce the model of automata we will use in the rest of the paper.

Given two positive integers i and j , $i \leq j$, we denote with $[i, j]$ the set of integers k with $i \leq k \leq j$, and with $[j]$ the set $[1, j]$.

An n -stack call-return alphabet is a tuple $\tilde{\Sigma}_n = \langle \Sigma_c^i, \Sigma_r^i, \Sigma_{int}^i \rangle_{i=1}^n$ of pairwise disjoint finite alphabets. For any $i \in [n]$, Σ_c^i is a finite set of *calls of stack i* , Σ_r^i is a finite set of *returns of stack i* , and Σ_{int}^i is a finite set of *internal actions of stack i* . For any such $\tilde{\Sigma}_n$, let

- $\Sigma^i = \Sigma_c^i \cup \Sigma_r^i \cup \Sigma_{int}^i$, for every $i \in [n]$;
- $\Sigma_c = \bigcup_{i=1}^n \Sigma_c^i$, $\Sigma_r = \bigcup_{i=1}^n \Sigma_r^i$, and $\Sigma_{int} = \bigcup_{i=1}^n \Sigma_{int}^i$;
- $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$.

A multi-stack visibly pushdown automaton over such an alphabet must push on stack i exactly one symbol when it reads a call of the i -th alphabet, and pop exactly one symbol from stack i when it reads a return of the i -th alphabet. Also, it cannot touch any stack when reading an internal symbol.

Definition 1. (MULTI-STACK VISIBLY PUSHDOWN AUTOMATON) A multi-stack visibly pushdown automaton (MVPA) over the n -stack call-return alphabet $\tilde{\Sigma}_n = \langle \Sigma_c^i, \Sigma_r^i, \Sigma_{int}^i \rangle_{i=1}^n$, is a tuple $M = (Q, Q_I, \Gamma, \delta, Q_F)$ where Q is a finite set of states, $Q_I \subseteq Q$ is the set of initial states, Γ is a finite stack alphabet that contains a special bottom-of-stack symbol \perp , $\delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\perp\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{int} \times Q)$, and $Q_F \subseteq Q$ is the set of final states. Moreover, M is deterministic if $|Q_I| = 1$, and $|\{(q, a, q') \in \delta\} \cup \{(q, a, q', \gamma) \in \delta\} \cup \{(q, a, \gamma, q') \in \delta\}| = 1$, for every $q \in Q$ and $a \in \Sigma$.

Let us fix an n -stack alphabet $\tilde{\Sigma}_n$ for the rest of the paper.

A transition (q, a, q', γ) , for $a \in \Sigma_c^i$ and $\gamma \neq \perp$, is a push-transition where on input a , γ is pushed onto stack i and the control changes state from q to q' . Similarly, (q, a, γ, q') for $a \in \Sigma_r^i$ is a pop-transition where on input a , γ is read from the top of stack i and popped (except for $\gamma = \perp$, which is never popped), and the control changes from q to q' . A transition (q, a, q') , for $a \in \Sigma_{int}$, is an internal transition where on input a the control changes from q to q' .

A *stack content* σ is a nonempty finite sequence over Γ where the bottom-of-stack symbol \perp appears always in the end, i.e., $\sigma \in (\Gamma \setminus \{\perp\})^* \cdot \{\perp\}$. A *configuration* of an MVPA M over $\tilde{\Sigma}_n$ is a tuple $C = \langle q, \sigma_1, \dots, \sigma_n \rangle$, where $q \in Q$ and each σ_i is a stack content. Moreover, C is *initial* if $q \in Q_I$ and $\sigma_i = \perp$ for every $i \in [n]$, and *accepting* if $q \in Q_F$. *Transitions* between configurations are defined as follows: $\langle q, \sigma_1, \dots, \sigma_n \rangle \xrightarrow{a}_M \langle q', \sigma'_1, \dots, \sigma'_n \rangle$ if one of the following holds (M is omitted whenever it is clear from the context):

- [Push]** If $a \in \Sigma_c^i$ (i.e., a is a call of stack i), then $\exists \gamma \in \Gamma$ such that $(q, a, q', \gamma) \in \delta$, $\sigma'_i = \gamma \cdot \sigma_i$, and $\sigma'_h = \sigma_h$ for every $h \in ([n] \setminus \{i\})$.
- [Pop]** If $a \in \Sigma_r^i$ (i.e., a is a return of stack i), then $\exists \gamma \in \Gamma$ such that $(q, a, \gamma, q') \in \delta$, $\sigma'_h = \sigma_h$ for every $h \in ([n] \setminus \{i\})$, and either $\gamma \neq \perp$ and $\sigma_i = \gamma \cdot \sigma'_i$, or $\sigma_i = \sigma'_i = \perp$.
- [Internal]** If $a \in \Sigma_{int}$, then $(q, a, q') \in \delta$, and $\sigma'_h = \sigma_h$ for every $h \in [n]$.

For a word $w = a_1 \dots a_m$ in Σ^* , a *run* of M on w from C_0 to C_m , denoted $C_0 \xrightarrow{w}_M C_m$, is a sequence of transitions $C_{i-1} \xrightarrow{a_i} C_i$ for $i \in [m]$ where each C_j is a configuration. A word $w \in \Sigma^*$ is accepted by an MVPA M if there is an initial configuration C and an accepting configuration C' such that $C \xrightarrow{w}_M C'$. The set of all words accepted by M is denoted with $L(M)$.

A visibly pushdown automaton [1] is an MVPA with just one stack.

Definition 2. (VISIBLY PUSHDOWN AUTOMATON) A visibly pushdown automaton, denoted VPA, is an MVPA over $\tilde{\Sigma}_n$ with $n = 1$. A language over Σ accepted by a VPA is a visibly pushdown language. With VPL we denote the class of visibly pushdown languages.

2.1 Restricting to a bounded number of rounds

Let w be a word over $\tilde{\Sigma}_n$. A *context* in w over Σ^i , with $i \in [n]$, is a word $w' \in (\Sigma^i)^*$ such that w can be factorized as either $uaw'bv$ or uaw' or $w'bv$ where $a, b \notin \Sigma^i$. Note that the empty word ε is a context over any stack alphabet.

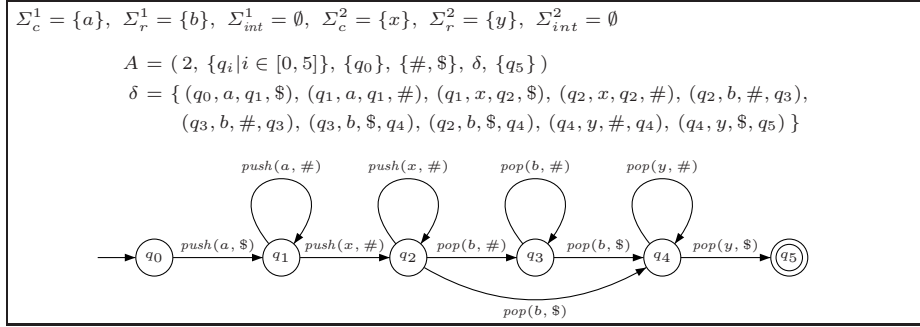


Fig. 1. A 2-round MVPA recognizing the language $\{a^t x^s b^t y^s \mid t, s \geq 1\}$.

A *round* over $\tilde{\Sigma}_n$ is a word w of Σ^* of the form $w_1 w_2 \dots w_n$ where for each $i \in [n]$, w_i is a context over Σ^i . A *k-round word* over $\tilde{\Sigma}_n$ is a word of Σ^* that can be obtained as the concatenation of k rounds over $\tilde{\Sigma}_n$. Let $Round(\tilde{\Sigma}_n, k)$ denote the set of all the k -round words over $\tilde{\Sigma}_n$.

Definition 3. (MULTI-STACK VISIBLY PUSHDOWN LANGUAGES WITH k -ROUNDS) For any k , a k -round multi-stack visibly pushdown automaton (k -round MVPA) over $\tilde{\Sigma}_n$ is a tuple $A = (k, Q, Q_I, \Gamma, \delta, Q_F)$ where $M = (Q, Q_I, \Gamma, \delta, Q_F)$ is an MVPA over $\tilde{\Sigma}_n$. Moreover, A is deterministic iff M is deterministic. The language accepted by A is $L(A) = L(M) \cap Round(\tilde{\Sigma}_n, k)$ and is called a k -round multi-stack visibly pushdown language. The class of k -round multi-stack visibly pushdown languages is denoted with k -RVPL. The set $\bigcup_{k \geq 1} k$ -RVPL is denoted with RVPL (the class of multi-stack visibly pushdown languages with a bounded number of rounds).

Example 1. Figure 1 gives a formal definition of a 2-round MVPA A over $\tilde{\Sigma}_2$ that accepts the language $\{a^t x^s b^t y^s \mid t, s \geq 1\}$. The automaton A checks whether the input word has the form $a^+ x^+ b^+ y^+$ using its control states. A starts in the control state q_0 . When it reads the first call symbol a it pushes the symbol $\$$ onto the stack S_1 ; for all the remaining a 's A pushes the symbol $\#$ onto S_1 . Stack S_1 will contain as many symbols as the number of read a 's. When the first call symbol x of stack 2 is read a $\$$ symbol is pushed onto stack S_2 , for the remaining b 's the symbol $\#$ is pushed onto stack S_2 . As in the previous case, stack S_2 will contain as many symbols as the b 's which are read. Stack S_1 is then popped for each return symbol b until S_1 is empty (read the symbol $\#$). Then only return symbols y can be read. Stack S_2 is popped for each read y until it gets empty (popping the symbol $\$$). After that A moves into the accepting state q_5 . \square

The main result on MVPAs with a bounded number of rounds, that we show in this paper, is that the class of languages accepted by the deterministic and the nondeterministic models coincide. Notice that the boundedness of the number of rounds is crucial in our proof. In fact, determinizability does not hold in general for MVPAs. To see this consider the language $L = \{(ab)^i c^j d^{i-j} x^j y^{i-j} \mid i \in \mathbb{N}, j \in [i]\}$ over $\tilde{\Sigma}_2 = (\Sigma_c^1, \Sigma_r^1, \Sigma_c^2, \Sigma_r^2, \emptyset)$, with $\Sigma_c^1 = \{a\}, \Sigma_r^1 = \{c, d\}, \Sigma_c^2 = \{b\}, \Sigma_r^2 = \{x, y\}$. First, observe that since the number of occurrences of ab is unbounded in L , this language contains words with an unbounded number of rounds and thus cannot be

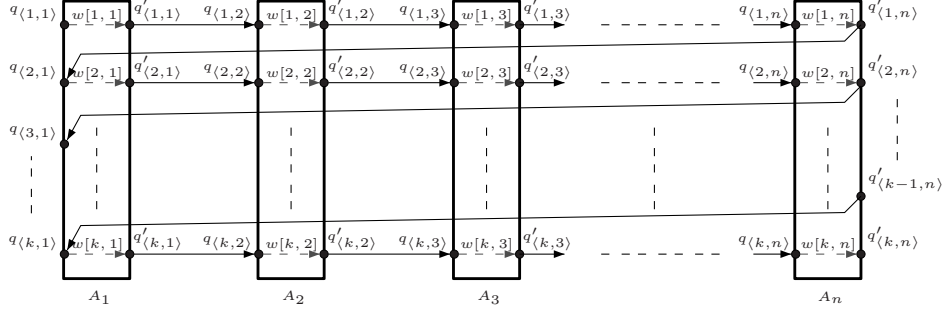


Fig. 2. Decomposition of a k -round MVPA.

accepted by a k -round MVPA for any fixed k . Further, this language is accepted by a nondeterministic MVPA which guesses nondeterministically the index j when pushing symbols on reading a and b . However, it is not accepted by any deterministic MVPA, since a deterministic MVPA would need an unbounded number of control states to store the index j (see [9]). Therefore, we have:

Theorem 1. *The class of MVPA is not closed under determinization.*

3 Determinization of k -round MVPAs

In this section we prove the main result of this paper: if A is a k -round MVPA over $\tilde{\Sigma}_n$, then there exists a deterministic k -round MVPA A^D over $\tilde{\Sigma}_n$ such that $L(A) = L(A^D)$.

For ease presentation, we assume that each context of a word in $Round(\tilde{\Sigma}_n, k)$ has at least a symbol.

Fix a k -round MVPA A and a word $w \in Round(\tilde{\Sigma}_n, k)$. The main idea behind our construction of A^D is to look at the executions of A on w as shown in Fig. 2, where $w[i, j]$ denotes the j -th context of the i -th round in w . The automaton A is seen as composed of individual blocks A_j for $j \in [n]$, each A_j working as a VPA whose stack is the j -th stack of A .

Initially A is in control state $q_{(1,1)}$. Then, it starts the computation by passing $q_{(1,1)}$ to A_1 . A_1 reads $w[1, 1]$ and reaches a control state $q'_{(1,1)}$ with stack content $\sigma_{(1,1)}$. At this point, A stops A_1 and passes $q_{(1,2)} = q'_{(1,1)}$ to A_2 . A_2 reads $w[1, 2]$ and reaches a state $q'_{(1,2)}$ with stack content $\sigma_{(1,2)}$. And so on, from A_3 through A_n , until $q'_{(1,n)}$ is reached. Now, A passes $q_{(2,1)} = q'_{(1,n)}$ to A_1 . Since this is the first time A_1 is re-activated after reading $w[1, 1]$, its stack (i.e., the first one) has not changed in the meantime. Thus A_1 starts now from control state $q_{(2,1)}$ and stack content $\sigma_{(1,1)}$. Then, again by reading $w[2, 1]$, A_1 reaches a control state $q'_{(2,1)}$ and A_2 is started from control state $q_{(2,2)} = q'_{(2,1)}$ and stack content $\sigma_{(1,2)}$, and so on, until completion of the whole run.

The salient aspect in the above description is that each run of A on w can be computed by running each A_j individually on $w[1, j], \dots, w[k, j]$, provided that $q_{(1,j)}, \dots, q_{(k,j)}$ are fed. Also, note that A_j computes a relation of state pairs $\langle q_{(i,j)}, q'_{(i,j)} \rangle$ which are connected by a run of A_j over words $w[i, j]$ for $i \in [k]$, and thus, a relation of tuples $\langle q_{(i,j)}, q'_{(i,j)} \rangle_{i=1}^k$ corresponding to words $\langle w[i, j] \rangle_{i=1}^k$. We call such relation tuples *switching vectors*. It is worth noticing that the switching vectors store all the

information we need to stitch together the local runs of all A_j 's in order to build a global run of A .

This suggests the following scheme to construct A^D . First, for each A_j , construct a VPA A'_j that computes the switching vectors (by storing them in the accepting states) corresponding to $\langle w[i, j] \rangle_{i=1}^k$ when reading $w[1, j] \# \dots \# w[k, j] \#$, where $\#$ is a fresh internal symbol. Construct the deterministic VPAs A_j^D corresponding to each A'_j . Then, construct A^D by composing the A_j^D 's, for $j \in [n]$, such that: (1) the states of A^D are the cross product of the states of the A_j^D 's; (2) in each context over Σ^j , except for the first symbol, only A_j^D is executed, and on reading the first symbol a of a context $j + 1$ of a round i , A_j^D is executed on input $\#$ and A_{j+1}^D is executed on input a ; (3) a word w is accepted if the computed switching vectors for each A_j^D can be composed according to a scheme such as in Fig. 2, i.e., they form a *sequence of compatible tuples*.

The above sketched construction is formally addressed in the rest of this section (more details are available in the Appendix). We start by defining the switching vectors, and then construct the VPA computing the switching vectors for a given VPA. We then define the concept of compatible tuples and prove that acceptance of A can be checked by verifying that there is a sequence of switching vectors of A_1, \dots, A_n which are compatible (this is used in the argument for showing soundness and correctness of the construction). Finally, we construct A^D by composing the VPAs computing the switching vectors of A_1, \dots, A_n and argue its soundness and correctness.

3.1 Visibly pushdown automata computing switching-vectors

Definition 4. (SWITCHING VECTORS) *Let M be a VPA over $\tilde{\Sigma}_1$ with set of control states Q , and $u = \langle u_i \rangle_{i=1}^k$ be a tuple of k words in Σ^* . The tuple $V = \langle (q_i, q'_i) \rangle_{i=1}^k \in (Q \times Q)^k$, is a switching-vector of M with respect to u if there exist k pairs of M configurations $\langle (C_i, C'_i) \rangle_{i=1}^k$, with $C_i = \langle q_i, \sigma_i \rangle$ and $C'_i = \langle q'_i, \sigma'_i \rangle$, such that (1) $\sigma_1 = \perp$, (2) $\sigma'_i = \sigma_{i+1}$, for every $i \in [k - 1]$ (3) $C_i \xrightarrow{u_i}_M C'_i$, for every $i \in [k]$.*

In the next lemma we prove the existence of a VPA T , called *switching automaton*, that computes switching-vectors of a given VPA.

Lemma 1. (SWITCHING AUTOMATA) *Let M be a VPA over $\langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$, and let $\#$ be a fresh symbol not in Σ . Then, there exists a nondeterministic VPA T over $\langle \Sigma_c, \Sigma_r, \Sigma_{int} \cup \{\#\} \rangle$ such that V is a switching-vector of M with respect to $\langle u_i \rangle_{i=1}^k \in \Sigma^k$ iff the state $\langle V \rangle$ is reachable in T reading the word $u_1 \# u_2 \# \dots \# u_k \#$.*

Sketch of the Proof. Let $M = (Q, Q_I, \Gamma, \delta, Q_F)$ be a VPA over $\langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$, and $V = \langle (q_i, q'_i) \rangle_{i=1}^k \in (Q \times Q)^k$. We assume that the symbol $\#$ does not belong to Σ .

The idea behind the construction of the VPA T that recognized the switching-vectors of M with respect to an input word is the following. T guesses, in its initial state, the switching-vector V that will compute on the input word. It simulates M on all the symbols in Σ . Whenever T reads the symbol $\#$ it changes the control state of M according to the guessed V : if T reads the i -th $\#$ symbol, and q_i is the state of M before reading $\#$, then T changes the state of M from q'_i to q_{i+1} , with an internal move on $\#$. At the end, when T reads the last $\#$ (the k -th) and the state of M is actually q'_k , then T moves into the state containing only V . \square

3.2 Compatible tuples

Definition 5. (COMPATIBLE TUPLES) *A sequence of compatible tuples V_1, V_2, \dots, V_n , with $V_j = \langle (q_{\langle i,j \rangle}, q'_{\langle i,j \rangle}) \rangle_{i=1}^k$, is such that*

- $q'_{\langle i,j \rangle} = q_{\langle i,j+1 \rangle}$, for every $i \in [k], j \in [n-1]$, and
- $q_{\langle i,n \rangle} = q_{\langle i+1,1 \rangle}$, for every $i \in [k-1]$.

The following lemma allows us to check acceptance of A by the existence of a sequence of compatible switching vectors. The lemma is used in the next section to argue soundness and correctness of the determinization construction.

Lemma 2. *Let $w \in \text{Round}(\tilde{\Sigma}_n, k)$, $A = (k, Q, Q_I, \Gamma, \delta, Q_F)$ be a k -round MVPA over $\tilde{\Sigma}_n$, and $w_j = \langle w[i, j] \rangle_{i=1}^k$, for $j \in [n]$. The word $w \in L(A)$ iff for each $j \in [n]$, there exists a switching-vector $V_j = \langle (q_{\langle i,j \rangle}, q'_{\langle i,j \rangle}) \rangle_{i=1}^k$ of the VPA A_j with respect to w_j such that V_1, V_2, \dots, V_n is a sequence of compatible tuples, $q_{\langle 1,1 \rangle} \in Q_I$, and $q_{\langle k,n \rangle} \in Q_F$.*

3.3 Determinization of k -round MVPAS

Theorem 2. (DETERMINIZABILITY) *If A is a k -round MVPA over $\tilde{\Sigma}_n$, then there exists a deterministic k -round MVPA A^D over $\tilde{\Sigma}_n$ such that $L(A) = L(A^D)$. Moreover, the size of A^D is doubly exponential in the number of rounds, and singly exponential in the number of stacks and the number of states of A .*

Proof. Let $A = (k, Q, Q_I, \Gamma, \delta, Q_F)$ be a k -round MVPA over $\tilde{\Sigma}_n = \langle \Sigma_c^j, \Sigma_r^j, \Sigma_{int}^j \rangle_{j=1}^n$. Define A_j , for $j \in [n]$, the VPA $(Q, Q_I, \Gamma, \delta', Q_F)$ over $\langle \Sigma_c^j, \Sigma_r^j, \Sigma_{int}^j \rangle$ where $\delta' \subseteq \delta$ is the set of all moves of δ on symbols of Σ^j .

For $j \in [n]$, let T_j be a switching automaton which accepts the same language as A_j constructed according to Lemma 1 and S_j be the deterministic VPA such that $L(S_j) = L(T_j)$ which is obtained via the construction given in [1]. Thus, a state of S_j is of the form (S, R) , where R is the set of reachable control states of T_j . From the definition of T_j , it is easy to see that the R -component of the S_j state reached on input $u_1 \# u_2 \# \dots \# u_k \#$ is the set of all the switching-vectors of A_j with respect to $\langle u_i \rangle_{i=1}^k \in \Sigma^k$.

The idea behind the construction of A^D is the following. A^D simulates each S_j in parallel, by keeping track of the control state of every S_j . The entire simulation mimics the schema shown in Fig. 2. After the entire input word w is read, A^D has reached a state storing the set of all switching-vectors of each A_j . Thus, from Lemma 2, A^D accepts the input word w if and only if A accepts it.

An issue that has to be addressed in the simulation of all S_j is the following. Let w be the input word of A . S_j needs to read a symbol $\#$ when a context-switch happens, i.e., at the the end of each context j of each round. Thus, the idea is to simulate the move on $\#$ meanwhile A^D processes the first symbol of the next context. This solves the problem for all the occurrences of $\#$ but the last one (no other context follows). Thus, for the simulation of S_n in the last round, we keep a pair $(q, q_\#)$ of S_n states where q is the current state of S_n in the simulation, and $q_\#$ is the state computed from q by applying the transition on $\#$. Thus, q is used to simulate the moves of S_n , and $q_\#$ is considered only for acceptance. More details on this construction are reported in the Appendix. \square

4 Closure Properties and Decision Problems

Let A_1, A_2 be two k -round MVPAS such that $L(A_1) = L_1$ and $L(A_2) = L_2$. To prove that $L_1 \cup L_2$ is a k -RVPLS use the standard nondeterministic construction. $L_1 \cap L_2$ can be accepted by an MVPA A that has as its set of states the product of the states of A_1 and A_2 , and as its stack alphabet the product of the stack alphabets of A_1 and A_2 . When reading a call of stack i , if A_1 pushes γ_1 and A_2 pushes γ_2 , respectively on their i -th stack, then A pushes (γ_1, γ_2) onto its i -th stack. The set of initial (final) states is the product of the initial (final) states of A_1 and A_2 . Moreover, from Theorem 2 we can construct a deterministic k -round MVPA A accepting L_1 . A k -round MVPA accepting $\overline{L_1}$ by complementing the final states of A . Thus, we have:

Theorem 3. (CLOSURE PROPERTIES) *Let L_1 and L_2 be two k -RVPLS over $\tilde{\Sigma}_n$. Then, $L_1 \cup L_2$, $L_1 \cap L_2$ and $\overline{L_1}$ are k -RVPLS over $\tilde{\Sigma}_n$.*

The *membership problem* for k -round MVPAS is to check, for any fixed k -round MVPA A over $\tilde{\Sigma}_n$, whether a given word $w \in \Sigma^*$ is accepted by A . From Theorem 2, we have:

Theorem 4. (MEMBERSHIP) *The membership problem for k -RVPLS is decidable in linear time.*

The *emptiness problem* for k -RVPLS is to check whether a given k -round MVPA accepts at least a word of $\text{Round}(\tilde{\Sigma}_n, k)$. From [12, 15] we have:

Theorem 5. (EMPTINESS) *The emptiness problem for k -RVPLS is NP-complete.*

The *universality problem* for k -RVPLS is to check whether a given k -round MVPA accepts all the strings of $\text{Round}(\tilde{\Sigma}_n, k)$. The *inclusion problem* is to find whether, given two k -round MVPAS A_1 and A_2 over $\tilde{\Sigma}_n$, $L(A_1) \subseteq L(A_2)$. From the closure under Boolean operations (Theorem 3) and decidability of emptiness (Theorem 5), we have:

Theorem 6. (UNIVERSALITY, INCLUSION) *The universality and the inclusion problems are decidable.*

5 Discussion

In this section, we start by discussing two models that can be obtained by bounding the number of contexts or the number of *phases* instead of the number of rounds in MVPAS. Then, we compare the class of languages defined by all these models in terms of recognized languages, closure properties and decision problems. Finally, we provide a monadic second-order logic characterization of the introduced classes of languages, and conclude with few remarks.

MVPA with bounded context-switches. Fix a $k \geq 0$. A k -context word w is a word of Σ^* such that $w = w_1 \dots w_k$ and for each $i \in [k]$, w_i is a context. We denote with $\text{Context}(\tilde{\Sigma}_n, k)$ the set of all k -context words over $\tilde{\Sigma}_n$. A k -context multi-stack visibly pushdown automaton (k -context MVPA) over $\tilde{\Sigma}_n$ is a tuple $A = (k, Q, Q_I, \Gamma, \delta, Q_F)$

where $M = (Q, Q_I, \Gamma, \delta, Q_F)$ is an MVPA over $\tilde{\Sigma}_n$. The language accepted by A is $L(A) = L(M) \cap \text{Context}(\tilde{\Sigma}_n, k)$. The class of languages accepted by k -context MVPAs is denoted with k -CVPL, and the set $\bigcup_{k>0} k$ -CVPL is denoted with CVPL (the class of all languages accepted by a k -context MVPA for any k).

The relation between k -RVPL and k -CVPL is stated in the following lemma.

Lemma 3. *Given a k -round MVPA R there is an nk -context MVPA C such that $L(C) = L(R)$. Moreover, if R is deterministic then also C is deterministic.*

Vice-versa, given a k -context MVPA C there is an k -round MVPA R such that $L(R) = L(C)$. Moreover, if C is deterministic then also R is deterministic.

From the above results, we can derive for k -context MVPAs all the results we have shown for k -round MVPAs in the previous sections. Also we have that the classes RVPL and CVPL coincide.

MVPA with a bounded number of phases [9]. Given a word $w \in \Sigma^*$, we denote with $\text{Ret}(w)$ the set of all returns in w . A word w is a *phase* if $\text{Ret}(w) \subseteq \Sigma_r^i$, for some $i \in [n]$. For any k , a k -phase word is a word $w \in \Sigma^*$ such that w can be factorized as $w = w_1 w_2 \dots w_{k'}$ where $k' \leq k$ and w_h is a phase, for every $h \in [k']$. With $\text{Phases}(\tilde{\Sigma}_n, k)$ we denote the set of all k -phase words over $\tilde{\Sigma}_n$.

For any k , a k -phase multi-stack visibly pushdown automaton (k -phase MVPA) over $\tilde{\Sigma}_n$ is a tuple $A = (k, Q, Q_I, \Gamma, \delta, Q_F)$ where $M = (Q, Q_I, \Gamma, \delta, Q_F)$ is an MVPA over $\tilde{\Sigma}_n$. The language accepted by A is $L(A) = L(M) \cap \text{Phases}(\tilde{\Sigma}_n, k)$. The class of languages accepted by a k -MVPA is denoted with k -PVPL, and the set $\bigcup_{k>0} k$ -PVPL is denoted with PVPL (the class of all languages accepted by a k -phase MVPA for any k).

k -PVPL is not closed under complement (the example given to argue Theorem 1 also shows nondeterminizability of k -phase MVPAs [9]). We denote with DPVPL the class of languages accepted by deterministic k -phase MVPAs for any k .

The notion of phase is less restrictive than the notion of context, i.e., a context can be entirely contained in a phase, while a phase may be composed of unboundedly many contexts. For example, consider the word $w = (ab)^m c^m d^m$ where a is call of stack 1 and c is its matching return, and b is a call of stack 2 and d is its matching return. From the definitions, we have that w is a 2-phase word for any m , and is a $(2m + 2)$ -context word but not a $2m + 1$ context word. In general, the language $\{(ab)^i c^i d^i \mid i \geq 0\}$ is accepted by a deterministic 2-phase MVPA but is not accepted by any k -context MVPA for any k , and thus by not any k -round MVPA for any k . Therefore, by Lemma 3 we have the following theorem.

Theorem 7. $\text{RVPL} \subset \text{DPVPL}$.

The Parikh mapping $\Phi(w)$, introduced by Parikh [14], associates a word with the vector of natural numbers that reflect the number of occurrences of the symbols in the word. This mapping extends to languages in the natural way. Since a Parikh theorem holds for k -phase MVPAs [9], from Theorem 7 we get:

Corollary 1. *For every RVPL (resp. CVPL) L over $\tilde{\Sigma}_n$, there exists a regular language L' over Σ such that $\Phi(L') = \Phi(L)$. Moreover, L' can be effectively computed.*

	Closure properties				Decision Problems	
	U	∩	Compl.	Determ.	Emptiness	Univ./ Equiv./Incl.
Reg.	Yes	Yes	Yes	Yes	NLOG	PSPACE
VPL	Yes	Yes	Yes	Yes	PTIME	EXPTIME
CFL	Yes	No	No	No	PTIME	Undecidable
Rvpl	Yes	Yes	Yes	Yes	NP	In 2Exptime
PVPL	Yes	Yes	Yes	No	2ETIME	In 3EXPTIME
CSL	Yes	Yes	Yes	Unknown	Undecidable	2EXPTIME-HARD Undecidable

Fig. 3. Summary of main closure properties and decision problems.

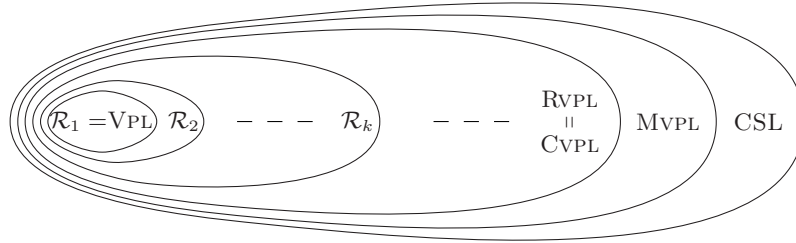


Fig. 4. Relationships among the considered classes of languages and the context-sensitive languages. With \mathcal{R}_k we denote the class k -RVPL.

Comparisons. The table in Figure 3 summarize and compare the closure properties and the decision problems for CSLs, CFLs, VPLs, PVPLs, RVPLs, and regular languages (see [4]). In the table, NLOG stands for NLOG-complete, and so on for the other complexity classes.

We also observe that the class 1-RVPL coincides exactly with VPL. In fact, to accept a language in 1-RVPL we really cannot exploit the multiple stacks since each stack is used just in one context. Therefore, we can simulate the n stacks just with one. Given a 2-stack alphabet as in Example 1, the language $L_h = \{a^{t_1}x^{s_1} \dots a^{t_h}x^{s_h}b^t y^s \mid t = \sum_{i=1}^h, s = \sum_{i=1}^h\}$ can be used to show a strict hierarchy on the class of languages accepted by k -round MVPA for different values of k . In particular, L_k is accepted by a $(k+1)$ -round MVPA but not by any k -round MVPA. Moreover, by Corollary 1, the language $\{a^{2^m} \mid m \geq 0\}$ is context sensitive but is not accepted by any k -round MVPA.

In Fig. 4, we summarize the relations among the considered classes of languages.

A Logical Characterization. Consider the *monadic second-order logic* (MSO_μ) over $\tilde{\Sigma}_n$ defined by:

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

where $j \in [n]$, $a \in \Sigma$, x, y are a first-order variables and X is a set variable [9].

The models are words over Σ . The first order variables are interpreted over the positions of w , and the second-order variables range over sets of positions. Each of the n binary relations μ_j ($j \in [n]$) is interpreted as the nested matching relation of calls and returns of Σ^j . We denote with $R_k(\varphi)$ (resp. $C_k(\varphi)$) the set of all words of $\text{Round}(\tilde{\Sigma}_n, k)$ (resp. $\text{Round}(\tilde{\Sigma}_n, k)$) that satisfy a sentence φ . By standard techniques to convert MSO to automata (given that the automata are closed under boolean operations and projection), we get (see [17]):

Theorem 8. *A language L is a k -RVPL (resp. k -CVPL) over $\tilde{\Sigma}_n$ iff there is an MSO_μ sentence φ over $\tilde{\Sigma}_n$ with $R_k(\varphi) = L$ (resp. $C_k(\varphi)$).*

Conclusions. We have studied a robust class of languages and automata which is closed under all the Boolean operations and has decidable problems, and strictly generalizes the class of visibly pushdown languages. Notably the class of multi-stack visibly pushdown automata working within a bounded number of rounds (or contexts) is closed under determinization, differently from other previously introduced models with multiple pushdown stores.

The models we have considered have been already successfully used in the practice of the verification of concurrent programs. In this paper, we have shown that such models indeed also form an interesting class from a theoretical point of view.

References

1. R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC*, pag. 202–211. ACM, 2004.
2. A. Bouajjani, S. Fratani, and S. Qadeer. Context-bounded analysis of multithreaded programs with dynamic linked structures. In *CAV*, vol. 4590 of *LNCS*, pag. 207–220. Springer, 2007.
3. D. Carotenuto, A. Murano, and A. Peron. 2-visibly pushdown automata. In *DLT*, vol. 4588 of *LNCS*, pag. 132–144. Springer, 2007.
4. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
5. S. La Torre, P. Madhusudan, and G. Parlato. 2-VPAs are not determinizable. <http://www.cs.uiuc.edu/homes/~madhu/vpa/wrong-proof-CMP07.html>, 2007.
6. S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS, LNCS 4963*, pag. 299–314. Springer, 2008.
7. S. La Torre, P. Madhusudan, and G. Parlato. Analyzing recursive programs using fixed-point calculus. In *PLDI*, 2009.
8. S. La Torre, P. Madhusudan, and G. Parlato. Reducing context-bounded concurrent reachability to sequential reachability. In *CAV*, 2009.
9. S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *LICS*, pag. 161–170. IEEE Computer Society, 2007.
10. S. Lahiri, S. Qadeer, and Z. Rakamaric. Static and precise detection of concurrency errors in systems code using SMT solvers. In *CAV*, 2009.
11. A. Lal and T. W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. In *CAV*, vol. 5123 of *LNCS*, pag. 37–51. Springer, 2008.
12. A. Lal, T. Touili, N. Kidd, and T. W. Reps. Interprocedural analysis of concurrent programs under a context bound. In *TACAS*, vol. 4963 of *LNCS*, pag. 282–298. Springer, 2008.
13. M. Musuvathi and S. Qadeer. Iterative context bounding for systematic testing of multithreaded programs. In *PLDI*, pag. 446–455. ACM, 2007.
14. R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.
15. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, vol. 3440 of *LNCS*, pag. 93–107. Springer, 2005.
16. D. Suwimonteerabuth, J. Esparza, and S. Schwoon. Symbolic context-bounded analysis of multithreaded java programs. In *SPIN*, vol. 5156 of *LNCS*, pag. 270–287. Springer, 2008.
17. W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages, Volume 3*, pag. 389–455. 1997.
18. VPL webpage. <http://www.cs.uiuc.edu/homes/~madhu/vpa/>.

A Construction of switching automaton (Lemma 1)

The idea behind the construction of the VPA T that recognized the switching-vectors of M with respect to an input word is the following. T guesses, in its initial state, the switching-vector V that will compute on the input word. It simulates M on all the symbols in Σ . Whenever T reads the symbol $\#$ it changes the control state of M according to the guessed V : if T reads the i -th $\#$ symbol, and q_i is the state of M before reading $\#$, then T changes the state of M from q'_i to q_{i+1} , with an internal move on $\#$. At the end, when T reads the last $\#$ (the k -th) and the state of M is actually q'_k , then T moves into the state containing only V .

A control state of T is of the form $\langle i, q, V \rangle$, where i is a counter to take track of the index of the word u_i that T is reading, q is meant to keep the state of M , and V is the switching-vector guessed in the initial configuration. Formally, $T = (Q^T, Q_I^T, \Gamma, \delta^T, \emptyset)$, where $Q^T = ([k] \times Q \times (Q \times Q)^k) \cup (Q \times Q)^k$, $Q_I^T = \{ (1, q_1, V) \mid V \in (Q \times Q)^k \}$, and δ^T is the minimal set containing the following transitions:

- [Int] $(\langle i, q, V \rangle, a, \langle i, q', V \rangle) \in \delta^T$, if $(q, a, q') \in \delta$;
- [Push] $(\langle i, q, V \rangle, a, \langle i, q', V \rangle, \gamma) \in \delta^T$, if $(q, a, q', \gamma) \in \delta$;
- [Pop] $(\langle i, q, V \rangle, a, \gamma, \langle i, q', V \rangle) \in \delta^T$, if $(q, a, \gamma, q') \in \delta$;
- [Context-Switch] $(\langle i, q'_i, V \rangle, \#, \langle i+1, q_{i+1}, V \rangle) \in \delta^T$.
- [Last #] $(\langle k, q'_k, V \rangle, \#, V) \in \delta^T$.

B Detailed proof of Theorem 2

Theorem 2. (DETERMINIZABILITY) *If A is a k -round MVPA over $\tilde{\Sigma}_n$, then there exists a deterministic k -round MVPA A^D over $\tilde{\Sigma}_n$ such that $L(A) = L(A^D)$. Moreover, the size of A^D is doubly exponential in the number of rounds, and singly exponential in the number of stacks and the number of states of A .*

Proof. Let $A = (k, Q, Q_I, \Gamma, \delta, Q_F)$ be a k -round MVPA over $\tilde{\Sigma}_n$. Define A_j , for $j \in [n]$, the VPA $(Q, Q_I, \Gamma, \delta', Q_F)$ over $\langle \Sigma_c^j, \Sigma_r^j, \Sigma_{int}^j \rangle$ where $\delta' \subseteq \delta$ is the set of all moves of δ on symbols of Σ^j and $\tilde{\Sigma}_n = \langle \Sigma_c^j, \Sigma_r^j, \Sigma_{int}^j \rangle_{j=1}^n$.

For $j \in [n]$, let T_j be a switching automaton which accepts the same language as A_j constructed according to Lemma 1 and S_j be the deterministic VPA such that $L(S_j) = L(T_j)$ which is obtained via the construction given in [1]. Thus, a state of S_j is of the form (S, R) , where R is the set of reachable control states of T_j . From the definition of T_j , it is easy to see that the R -component of the S_j state reached on input $u_1\#u_2\#\dots\#u_k\#$ is the set of all the switching-vectors of A_j with respect to $\langle u_i \rangle_{i=1}^k \in \Sigma^k$.

The idea behind the construction of A^D is the following. A^D simulates each S_j in parallel, by keeping track of the control state of every S_j . The entire simulation mimics the schema shown in Fig. 2. After the entire input word w is read, A^D has reached a state storing the set of all switching-vectors of each A_j . Thus, from Lemma 2, A^D accepts the input word w if and only if A accepts it.

An issue that has to be addressed in the simulation of all S_j is the following. Let w be the input word of A . S_j needs to read a symbol $\#$ when a context-switch happens, i.e., at the the end of each context j of each round. Thus, the idea is to simulate the

move on $\#$ meanwhile A^D processes the first symbol of the next context. This solves the problem for all the occurrences of $\#$ but the last one (no other context follows). Thus, for the simulation of S_n in the last round, we keep a pair $(q, q_\#)$ of S_n states where q is the current state of S_n in the simulation, and $q_\#$ is the state computed from q by applying the transition on $\#$. Thus, q is used to simulate the moves of S_n , and $q_\#$ is considered only for acceptance.

A state of A^D has the form $(q_1, q_2, \dots, q_n, q_{n+1}, i, j)$. The meaning of such tuple is that q_h is the current control state of each S_h in the simulation, for $h \in [n]$, q_{n+1} is as the state $q_\#$ described above, meanwhile i, j keep track of the current round and context, respectively.

Here we formally define A^D . Let $S_j = (Q_j, q_I^j, \Gamma^D, \emptyset)$, for every $j \in [n]$. Then, $A^D = (k, Q^D, q_I^D, \Gamma^D, \delta^D, Q_F^D)$, where $Q^D = Q_1 \times Q_2 \times \dots \times Q_n \times Q_n \times [k] \times [n]$, $q_I^D = (q_I^1, q_I^2, \dots, q_I^n, q_I^n, 1, 1)$, and δ^D is the minimal set of transitions defined by the following rules. For two tuples $\{q_i\}_{i=1}^{n+1}$ and $\{q'_i\}_{i=1}^{n+1}$, if not specified explicitly we assume that $q_n = q'_n$, for every $i \in [n+1]$.

Simulation of S_j when $i = k, j = n$ does not hold.

- [Int] $(\langle \{q_t\}_{t=1}^{n+1}, i, j \rangle, a, \langle \{q'_t\}_{t=1}^{n+1}, i, j \rangle) \in \delta^D$, if $(q_j, a, q'_j) \in \delta_j$;
- [Push] $(\langle \{q_t\}_{t=1}^{n+1}, i, j \rangle, a, \langle \{q'_t\}_{t=1}^{n+1}, i, j \rangle, \gamma) \in \delta^D$, if $(q_j, a, q'_j, \gamma) \in \delta_j$;
- [Pop] $(\langle \{q_t\}_{t=1}^{n+1}, i, j \rangle, a, \gamma, \langle \{q'_t\}_{t=1}^{n+1}, i, j \rangle) \in \delta^D$, if $(q_j, a, \gamma, q'_j) \in \delta_j$;

Change context on the same round. Let $j \in [2, n]$, and $(q_{j-1}, \#, q'_{j-1}) \in \delta_{j-1}$.

- [Int] $(\langle \{q_t\}_{t=1}^{n+1}, i, j-1 \rangle, a, \langle \{q'_t\}_{t=1}^{n+1}, j, j \rangle) \in \delta^D$, if $(q_j, a, q'_j) \in \delta_j$;
- [Push] $(\langle \{q_t\}_{t=1}^{n+1}, i, j-1 \rangle, a, \langle \{q'_t\}_{t=1}^{n+1}, i, j \rangle, \gamma) \in \delta^D$, if $(q_j, a, q'_j, \gamma) \in \delta_j$;
- [Pop] $(\langle \{q_t\}_{t=1}^{n+1}, i, j-1 \rangle, a, \gamma, \langle \{q'_t\}_{t=1}^{n+1}, i, j \rangle) \in \delta^D$, if $(q_j, a, \gamma, q'_j) \in \delta_j$;

Change context on consecutive rounds. Let $i \in [2, k]$, and $(q_n, \#, q'_n) \in \delta_n$.

- [Int] $(\langle \{q_t\}_{t=1}^{n+1}, i-1, n \rangle, a, \langle \{q'_t\}_{t=1}^{n+1}, i, 1 \rangle) \in \delta^D$, if $(q_1, a, q'_1) \in \delta_1$;
- [Push] $(\langle \{q_t\}_{t=1}^{n+1}, i-1, n \rangle, a, \langle \{q'_t\}_{t=1}^{n+1}, i, 1 \rangle, \gamma) \in \delta^D$, if $(q_1, a, q'_1, \gamma) \in \delta_1$;
- [Pop] $(\langle \{q_t\}_{t=1}^{n+1}, i-1, n \rangle, a, \gamma, \langle \{q'_t\}_{t=1}^{n+1}, i, 1 \rangle) \in \delta^D$, if $(q_1, a, \gamma, q'_1) \in \delta_1$;

Simulation of S_n on round k . Let and $(q'_n, \#, q'_{n+1}) \in \delta_j$.

- [Int] $(\langle \{q_t\}_{t=1}^{n+1}, k, n \rangle, a, \langle \{q'_t\}_{t=1}^{n+1}, k, n \rangle) \in \delta^D$, if $(q_n, a, q'_n) \in \delta_n$;
- [Push] $(\langle \{q_t\}_{t=1}^{n+1}, k, n \rangle, a, \langle \{q'_t\}_{t=1}^{n+1}, k, n \rangle, \gamma) \in \delta^D$, if $(q_n, a, q'_n, \gamma) \in \delta_n$;
- [Pop] $(\langle \{q_t\}_{t=1}^{n+1}, k, n \rangle, a, \gamma, \langle \{q'_t\}_{t=1}^{n+1}, k, n \rangle) \in \delta^D$, if $(q_n, a, \gamma, q'_n) \in \delta_n$;

Finally, the final states of A^D are all the states of the form $(q_1, q_2, \dots, q_n, q_{n+1}, k, n)$ such that there exist a sequence of n switching-vectors V_1, V_2, \dots, V_n , with $V_j \in R(q_j)$ for every $j \in [n-1]$, $V_n \in R(q_{n+1})$ such that Lemma 2 holds. Here $R(q)$ denotes the R -component of the state q .

Turn now to the count of the number of states of A^D . From Lemma 1, the number of states of each T_j is $t = O(k \cdot m^{2k+1})$, where $m = |Q|$. Instead S_j has $s = O(2^{t^2})$ states (see [1]). Finally, A^D has $O(s^{n+1} \cdot k \cdot n)$ states. Therefore, the overall number of states of A^D is $O(2^{k^2 m^{4k+2}(n+1)} \cdot k \cdot n)$, where $m = |Q|$. \square