Reachability of Multistack Pushdown Systems with Scope-Bounded Matching Relations[⋆]

Salvatore La Torre and Margherita Napoli

Dipartimento di Informatica
Università degli Studi di Salerno, Italy

Abstract. A multi-stack pushdown system is a natural model of concurrent programs. The basic verification problems are in general undecidable (two stacks suffice to encode a Turing machine), and in the last years, there have been some successful approaches based on under-approximating the system behaviors. In this paper, we propose a restriction of the semantics of the general model such that a symbol that is pushed onto a stack can be popped only within a bounded number of context-switches. Note that, we allow runs to be formed of unboundedly many execution contexts, we just bound the scope (in terms of number of contexts) of matching push and pop transitions. We call the resulting model a *multi-stack pushdown system with scope-bounded matching relations* (SMPDS). We show that the configuration reachability and the location reachability problems for SMPDS are both PSPACE-complete, and that the set of the reachable configurations is regular, in the sense that there exists a multi-tape finite automaton that accepts it.

1 Introduction

Multi-stack pushdown systems are a natural and well-established model of programs with both concurrency and recursive procedure calls, which is suitable to capture accurately the flow of control. A multi-stack pushdown system is essentially a finite control equipped with one or more pushdown stores. Each store encodes a recursive thread of the program and the communication between the different threads is modelled with the shared states of the finite control.

The class of multi-stack pushdown systems is very expressive. It is well known that two stacks can simulate an unbounded read/write tape, and therefore, a push-down system with two stacks suffices to mimic the behaviour of an arbitrary Turing machine. In the standard encoding, it is crucial for the automaton to move an arbitrary number of symbols from one stack to another and repeat this for arbitrarily many times. To achieve decidability it is thus necessary to break this capability by placing some limitations on the model.

In the last years, the analysis of multi-stack pushdown systems within a bounded number of execution contexts (in each context only one stack is used)

[⋆] This work was partially funded by the MIUR grants FARB 2009-2010 Università degli Studi di Salerno, Italy.

has been proposed as an effective method for finding bugs in concurrent programs [14]. This approach is justified in practice by the general idea that most of the bugs of a concurrent program are likely to manifest themselves already within few execution contexts (which has also been argued empirically in [13]). Though bounding the number of context-switching in the explored runs does not bound the depth of the search of the state space (the length of each context is unbounded), it has the immediate effect of bounding the interaction among different threads and thus the exchanged information. In fact, the reachability problem with this limitation becomes decidable and is NP-complete [11,14].

In this paper, we propose a decidable notion of multistack pushdown system that does not bound the number of interactions among the different stacks, and thus looks more suitable for a faithful modeling of programs with an intensive interaction between threads. We impose a restriction which is technically an extension of what is done in the bounded context-switching approach but is indeed conceptually very different. We allow an execution to go through an unbounded number of contexts, however recursive calls that are returned can only span for a bounded number of contexts. In other words, we bound the *scope* of the matching push and pop operations in terms of the number of context switches allowed in the between. Whenever a symbol is pushed onto a stack, it is either popped within a bounded number of context switches or never popped. This has the effect that in an execution of the system, from each stack configuration which is reached, at most a finite amount of information can be moved into the other stacks, thus breaking the ability of the multistack pushdown system of simulating a Turing machine. We call the resulting model a *multistack pushdown system with scope-bounded matching relations* (SMPDS).

Bounding the scope of the matching relations to a given number of context-switches instead of the whole computation has some appealing aspects. First, for a same bound k, it covers a reachable space which is at least that covered by the bounded context-switching analysis, and in some cases can be significantly larger. In fact, there are systems such that the whole space of configurations can be explored with a constant bound on the scope while it will require a large number of context switches or even unboundedly many of them (see for example Figure 1). Thus, for systems where the procedure calls do not need to hang for many contexts before returning, the bounded scope analysis covers the behavior explored with the context bounded analysis with smaller values of the bound k, which is a critical parameter for the complexity of the decision algorithms (time is exponential in k in both settings). It is known that looking at the computations as nested words with multiple stack relations, when restricting to k-context computations, the corresponding set has a bounded tree-width (see [12]). Moreover, a bounded context-switch multistack pushdown system can be simulated by a standard pushdown system with just one stack (see [7,10]). For SMPDS instead, any of these results does not seem to work or at least requires a more complex encoding. Finally, SMPDS have a natural and meaningful semantics for infinite computations which allows to observe also infinitely many interactions between the different threads.

In this paper, we tackle the reachability problem for SMPDS, and show that it is PSPACE-complete. Our decision algorithm is an elaborated adaptation of the saturation procedure which is used for the analysis of pushdown sytems [15] and reused in [14] for solving the reachability within a bounded number of context switches. As in [14] we grow the set of reachable configurations as tuples of automata accepting configurations of each stack, and construct the automata by iterating the saturation algorithm from [15] into layers, each for execution context. However, in [14] the construction has a natural limit in the allowed number of contexts which is bounded, while in our setting, we appeal to the bound on the scope of the matching relations and use the automata to represent not all the stack contents but only the portions corresponding to the last k execution contexts.

We assume that the executions of an SMPDS go through rounds of schedule, where in each round all stacks are scheduled exactly once and always according to the same order. To prove our results, we construct a finite graph, that we call the *reachability graph*, whose nodes are n-tuples of k-layered automata along with the n-states of the finite control at which the context-switches have happened in the last round. The edges of the graph link a tuple u to any other tuple v such that executing a whole round, by context-switching at the states listed in u, the stack content of the last k rounds is captured by the tuple of k-layered automata in v. We show that the reachability graph reduces the location reachability for SMPDS to standard reachability in finite graphs. Since its size is exponential in the number of locations, the number of stacks and the number of rounds in a scope, this problem can be decided in polynomial space. The reachability graph can be also used as a general framework where the k-layered automata are combined to obtain an n-tape finite automaton that accept exactly the reachable configurations.

The configuration reachability problem is stated as the problem of deciding if a configuration within a set of target configurations is reachable. The target set is given as a Cartesian product of regular languages over the stack alphabets. The n-tape finite automaton accepting the reachable configurations can be modified to compute the intersection with the target set (this can be done componentwise since the target set is a Cartesian product) and thus the configuration reachability problem can be reduced to check emptiness of an n-tape finite automaton of exponential size, and thus again this can be done in polynomial space. PSPACE-hardness of the considered problems can be shown with standard constructions from the membership of linear bounded automata.

Related Work. In [9], the notion of context is relaxed and the behaviours of multistack pushdown systems are considered within a bounded number of phases, where in each phase only one stack is allowed to execute pop transitions but all the stacks can do push transitions. The reachability problem in this model turns out to be 2ETIME-complete [9,5]. We observe that in each phase an unbounded amount of information can pass from one stack to any other, but still this can be done only a bounded number of times. Thus, in some sense this extension is orthogonal to that proposed in this paper. Moreover, it is simple to verify that the extension of SMPDS where contexts are replaced with phases in the rounds is as powerful as Turing machines.

The notion of bounded-context switching has been successfully used in recent work: model-checking tools for concurrent Boolean programs [6,10,16]; translations of concurrent programs to sequential programs reducing bounded context-switching reachability to sequential reachability [7,10]; model-checking tools for Boolean abstractions of parameterized programs (concurrent programs with unboundedly many threads each running one of finitely many codes) [8]; sequentialization algorithms for parameterized programs [3]; model-checking of programs with dynamic creation of threads [1]; analysis of systems with systems heaps [2], systems communicating using queues [4], and weighted pushdown systems [11].

2 Multistack Pushdown Systems

In this section we introduce the notations and definitions 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 $[1, j]$.

A multi-stack pushdown system consists of a finite control along with one or more pushdown stores. There are three kinds of transitions that can be executed: the system can push a symbol on any of its stacks, or pop a symbol from any of them, or just change its control location by maintaining unchanged the stack contents. For the ease of presentation and without loss of generality we assume that the the symbols used in each stack are disjoint from each other. Therefore, a multi-stack pushdown system is coupled with an n-stack alphabet $\widetilde{\Gamma}_n$ defined as the union of n pairwise disjoint finite alphabets $\Gamma_1, \ldots, \Gamma_n$. Formally:

Definition 1. (MULTI-STACK PUSHDOWN SYSTEM) *A* multi-stack pushdown system *(*MPDS*) with n stacks is a tuple $M = (Q, Q_I, \widetilde{\Gamma}_n, \delta)$ where Q is a finite set of states, $Q_I \subseteq Q$ is the set of initial states, $\widetilde{\Gamma}_n$ is an n-stack alphabet, and $\delta \subseteq (Q \times Q) \cup (Q \times Q \times \Gamma) \cup (Q \times \Gamma \times Q)$ is the transition relation.*

We fix an n-stack alphabet $\widetilde{\Gamma}_n = \cup_{i=1}^{n} \Gamma_i$ for the rest of the paper. A transition (q, q') is an internal transition where the control changes from q to q' and the stack contents stay unchanged. A transition (q, q', γ) for $\gamma \in \Gamma_i$ is a push-transition where the symbol γ is pushed onto stack i and the control changes from q to q'. Similarly, (q, γ, q') for $\gamma \in \Gamma_i$ is a pop-transition where γ is read from the top of stack i and popped, and the control changes from q to q'. A *stack content* w is a possibly empty finite sequence over Γ. A *configuration* of an MPDS M is a tuple $C = \langle q, w_1, \ldots, w_n \rangle$, where $q \in Q$ and each w_i is a stack content. Moreover, C is *initial* if $q \in Q_I$ and $w_i = \varepsilon$ for every $i \in [n]$. Transitions between configurations are defined as follows: $\langle q, w_1, \ldots, w_n \rangle \rightarrow_M \langle q', w'_1, \ldots, w'_n \rangle$ if one of the following holds (M is omitted whenever it is clear from the context):

[Push] there is a transition $(q, q', \gamma) \in \delta$ such that $\gamma \in \Gamma_i$, $w'_i = \gamma \cdot w_i$, and $w'_h = w_h$ for every $h \in ([n] \setminus \{i\})$.

[Pop] there is a transition $(q, \gamma, q') \in \delta$ such that $w_i = \gamma \cdot w'_i$ and $w'_h = w_h$ for every $h \in ([n] \setminus \{i\})$.

[Internal] there is a transition $(q, q') \in \delta$, and $w'_h = w_h$ for every $h \in [n]$.

A *run* of M from C_0 to C_m, with $m \geq 0$, denoted $C_0 \rightsquigarrow_M C_m$, is a possibly empty sequence of transitions $C_{i-1} \rightarrow_M C_i$ for $i \in [m]$ where each C_i is a configuration. A pushdown system (PDS) is an MPDS with just one stack.

Reachability. A *target set of configurations* for M is $S \times \mathcal{L}(A_1) \times \ldots \times \mathcal{L}(A_n)$ such that $S \subseteq Q$ and for $i \in [n]$, $\mathcal{L}(A_i) \subseteq \varGamma_i^*$ is the language accepted by a finite automaton A_i. Given an MPDS $M = (Q, Q_I, \widetilde{\varGamma}_n, \delta)$ and a target set of configurations T, the *reachability problem* for M with respect to target T asks to determine whether there is a run of M from C_0 to C such that C_0 is an initial configuration of M and $C \in T$. The *location reachability* problem for MPDS is defined as the reachability problem with respect to a target set of the form $S \times \varGamma_1^* \times \ldots \times \varGamma_n^*$.

It is well known that the reachability problem for multi-stack pushdown systems is undecidable already when only two stacks are used (two stacks suffice to encode the behavior of a Turing machine) and is decidable in polynomial time (namely, cubic time) when only one stack is used (pushdown systems).

Theorem 1. *The (location) reachability problem is undecidable for* MPDS *and is decidable in cubic time for* PDS.

Execution Contexts and Rounds. We fix an MPDS $M = (Q, Q_I, \widetilde{\varGamma}_n, \delta)$. A *context* of M is a run of M where the pop and push transitions are all over the same stack (the only active stack in the context), i.e., a run $C_0 \rightarrow_M C_1 \rightarrow_M \ldots \rightarrow_M C_m$ over the transition rules $\delta \cap ((Q \times Q) \cup (Q \times \varGamma_i \times Q) \cup (Q \times Q \times \varGamma_i))$ for some $i \in [n]$. The *concatenation* of two runs $C \rightsquigarrow_M C'$ and $C' \rightsquigarrow_M C''$ is the run $C \rightsquigarrow_M C' \rightsquigarrow_M C''$. A *round* of M is the concatenation of n contexts $C_{i-1} \rightsquigarrow C_i$ for $i \in [n]$, where stack i is the active stack of $C_{i-1} \rightsquigarrow C_i$. Note that a run without push and pop transitions (and in particular, a run formed of a single configuration) is a context where the active stack can be any of the stacks. Thus, each run of M can be seen as the concatenation of many rounds (and contexts).

Multi-stack Pushdown Systems with Scope-bounded Matching Relations. In the standard semantics of MPDS a pop transition (q, γ, q') can be always executed from q when γ is at the top of the stack. We introduce a semantics that restricts this, in the sense that the pop transitions are allowed to execute only when the symbol at the top of the stack was pushed within the last k rounds, for a fixed k. Thus, each symbol pushed onto the stack can be effectively used only for boundedly many rounds (*scope of the push/pop matching*). We call the resulting model an MPDS *with scope-bounded matching relations* (SMPDS, for short). We use the notation k-SMPDS when we need to stress the actual bound k on the number of rounds of the scope. Formally:

Definition 2. (Mpds with scope-bounded matching relations) *A multi-stack pushdown system (k-*SMpds*) with n stacks and k-round scope is a tuple* $M = (k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$ *where* $k \in \mathbb{N}$ *and* $(Q, Q_I, \widetilde{\Gamma}_n, \delta)$ *is as for* Mpds.

For an Mpds $M = (Q, Q_I, \widetilde{\Gamma}_n, \delta)$ we often denote the corresponding SMpds $(k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$ with (k, M).

To describe the behavior of such systems, we extend the notion of configuration. We assume that when a symbol is pushed onto the stack, it is annotated with the number of the current round. Also we keep track of the current round number and the currently active stack. An *extended configuration* is of the form $\langle r, s, q, w_1, \ldots, w_n\rangle$ $r \in \mathbb{N}$, $s \in [n]$, $q \in Q$ and $w_i \in (\Gamma \times \mathbb{N})^*$. Note that by removing the components r and s, and the round annotation from the stacks, from each extended configuration we obtain a standard configuration. We define a mapping *conf* that maps each extended configuration to the corresponding configuration, that is, $conf(\langle r, s, q, w_1, \ldots, w_n\rangle)$ is the configuration $\langle q, \pi(w_1), \ldots, \pi(w_n)\rangle$, where for each $w_i = (\gamma_1, i_1) \ldots (\gamma_m, i_m)$, with $\pi(w_i)$ we denote the stack content $\gamma_1 \ldots \gamma_m$. An extended initial configuration is an extended configuration $E = \langle r, s, q, w_1, \ldots, w_n\rangle$ such that $r = s = 1$ and $conf(E)$ is an initial configuration.

Transitions between extended configurations are formally defined as follows: $\langle r, s, q, w_1, \ldots, w_n\rangle \mapsto_M \langle r', s', q', w'_1, \ldots, w'_n\rangle$ if one of the following holds (M is omitted whenever it is clear from the context):

[Push] $r' = r$, $s' = s$, $w'_h = w_h$ for every $h \in ([n] \setminus \{s\})$, and there is a transition $(q, q', \gamma) \in \delta$ such that $\gamma \in \Gamma_s$ and $w'_s = (\gamma, r) \cdot w_s$.

[Pop] $r' = r$, $s' = s$, $w'_h = w_h$ for every $h \in ([n] \setminus \{s\})$, and there is a transition $(q, \gamma, q') \in \delta$ such that $w_s = (\gamma, h) \cdot w'_s$ and $h > r - k$.

[Internal] $r' = r$, $s' = s$, and there is a transition $(q, q') \in \delta$ $w'_h = w_h$ for every $h \in [n]$.

[Context-switch] $w'_h = w_h$ for every $h \in [n]$, and if $s = n$ then $r' = r + 1$ and $s' = 1$, otherwise $r' = r$ and $s' = s + 1$.

An *extended run* of a k-SMpds M from E_0 to E_m, with $m \geq 0$, denoted $E_0 \hookrightarrow_M E_m$, is a possibly empty sequence of transitions $E_{i-1} \mapsto_M E_i$ for $i \in [m]$ where each E_i is an extended configuration. A *run* of a k-SMpds M from C to C', denoted $C \leadsto_M C'$, is the projection through *conf* of an extended run $E \hookrightarrow_M E'$ such that $C = conf(E)$ and $C' = conf(E')$. Using this notion of run, we define the reachability and the location reachability problems for SMpds as for Mpds. Given a run $\rho = C' \leadsto_M C$ where $C = \langle q, w_1, \ldots, w_n\rangle$, and an integer $m \geq 0$, with $Last_\rho(C, m)$ we denote the tuple $(y_1, \ldots, y_n)$ where for $i \in [n]$, y_i a prefix of w_i and contains the symbols pushed onto stack i in the last m rounds of ρ and not yet popped.

Example 1. Figure 1 gives a 2-stack Mpds M with stack alphabets $\Gamma_1 = \{a\}$ and $\Gamma_2 = \{b, c\}$. The starting location of M is q_0. A typical execution of the system M from the initial location q_0 starts pushing a on stack 1, and then b on stack 2. Thus, M iteratively pushes a on stack 1 and c on stack 2, until it starts popping

$M = (\{q_i | i \in [0,5]\}, \{q_0\}, \Gamma_1 \cup \Gamma_2, \delta) \quad \Gamma_2 = \{a\}, \ \Gamma_2 = \{b, c\}$

$\delta = \{ (q_0, q_1, a), (q_1, q_2, b), (q_2, q_3, a), (q_3, q_2, c), (q_2, c, q_4), (q_2, b, q_5), (q_4, c, q_4), (q_4, b, q_5) \}$

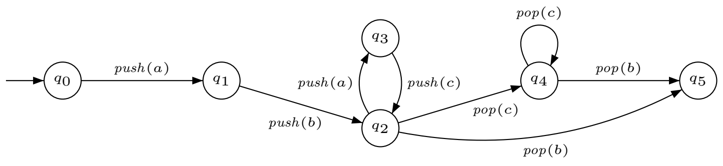

Fig. 1. The MPDS M from Example 1 and its graphical representation

c from stack 2. When all c's are popped out, it can also pop the b and finally reach location q_5 with stack 2 empty. Observe that each iteration of pushes of a's and c's (resp. b's) is a round, and assuming k of such iterations, a run ρ as described above can be split into at least k-rounds. Thus, it cannot be captured by any run of the SMPDS (h, M) for any $h < k$. However, denoting with ρ' the prefix of ρ up to the first pop transition, clearly ρ' is a run of $(1, M)$. □

3 Reachability within a Fixed Number of Contexts and Phases

Bounded-context Switching Reachability. A *k-round run* of M is the concatenation of k rounds. The *reachability problem within k rounds* is the restriction of the reachability problem to the sole k-round runs of M. It is defined as the problem of determining whether there is a k-round run of M starting from an initial configuration and reaching a target configuration in T.

Theorem 2. *[11,14] The (location) reachability problem within k rounds for* MPDS *is NP-complete.*

Bounded-phase Reachability. A *phase* of M is a run of M where the pop transitions are all from the same stack (pushes onto any of the stacks are allowed within the same phase). Exploring all the runs of a system obtained as the concatenation of k phases ensures a better coverage of the state space compared to k-contexts reachability. In fact, a k-phase run can be formed of an arbitrary number of contexts (for example, a run that iterates k times a push onto stack 1 and a push onto stack 2 is a $2k$-context one, while it uses only one phase). On the other side, the resulting reachability problem has higher complexity.

Theorem 3. *[9,5] The location reachability problem within k phases for* MPDS *is* 2ETIME-*complete.*

Coverage of the State Space in the Different Notions of Reachability. We compare the different reachability problems we have introduced in terms of coverage

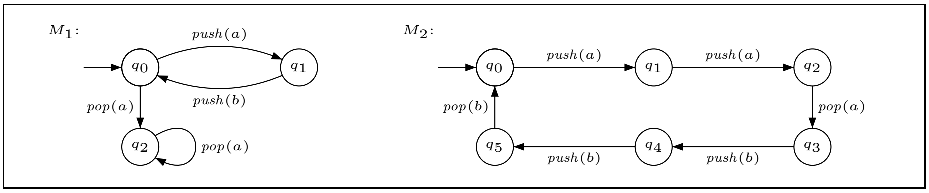

Fig. 2. Graphical representation of the MPDS M_1 and M_2

of the reachable state space of a multi-stack pushdown system. We start observing that for the MPDS M from Example 1, for each $k \geq 2$, the configuration $\langle q_4, a^k, c^{k-1}b\rangle$ is reachable in M within at least k rounds and is also reachable in the SMPDS $(1, M)$. Moreover, for each $k \geq 1$, the configuration $\langle q_5, a^k, \varepsilon\rangle$ is reachable in the SMPDS (k, M) and is not reachable in any of the SMPDS (h, M) for $h < k$. Therefore, the reachable set of a k-SMPDS strictly covers the set of configurations reachable within k-rounds, and thus provides a more careful approximation of the reachable configurations of MPDS. Formally, the following result holds.

Lemma 1. *Let $M = (Q, Q_I, \widetilde{\Gamma}_n, \delta)$ be an* MPDS.

If a configuration C' is reachable from C in (k, M), then C' is also reachable from C in M. Vice-versa, there is an MPDS *M' such that for each $k \in \mathbb{N}$ there is a reachable configuration C that is not reachable in the* SMPDS *(k, M).*

If a configuration C' is reachable from C in M within k rounds, then C' is also reachable from C in (k, M). Vice-versa, there is an MPDS *M' such that for each $k \in \mathbb{N}$ there is a reachable configuration C of (k, M') that is not reachable within k rounds.*

Now, fix $\Gamma_1 = \{a\}$, $\Gamma_2 = \{b\}$. Let M_1 be the 2-stack MPDS from Figure 2. Since the only pop transitions are from the same stack, any run of M_1 is 1-phase. It is simple to see that a configuration $C_k = \langle q_2, \varepsilon, b^k\rangle$ for $k \in \mathbb{N}$, is only reachable with a run of at least k rounds and moreover in the last round all the a's pushed onto stack 1 should be readable in order to pop all them out. Thus, C_k is not reachable in the SMPDS (h, M) for any $h < k$. Moreover, let M_2 be the 2-stack MPDS from Figure 2. It is simple to see that any run of M_2 is also a run of $(1, M_2)$. However, a configuration $C_k = \langle q_0, a^k, b^k\rangle$ for $k \in \mathbb{N}$ is not reachable with a run with less than $2k$ phases.

Therefore, from the above arguments and the given definitions, we get that the notion of reachability for SMPDS is not comparable with the notion of reachability up to k-phases. More precisely, we get the following result.

Lemma 2. *There is an* MPDS *M such that any reachable configuration can be reached within one phase, and for each $k \in \mathbb{N}$ there is a configuration C that is not reachable in the* SMPDS *(k, M).*

There is an MPDS *M such that any reachable configuration can be reached also in $(1, M)$, and for any $k \in \mathbb{N}$ there is a configuration C that is not reachable within k phases.*

4 Location Reachability for SMPDS

In this section, we present a decision algorithm that solves the location reachability problem, and address its computational complexity. The main step in our algorithm consists of constructing a finite graph such that solving the considered decision problem equals to solving a standard reachability problem on this graph, that we thus call the reachability graph. The nodes of the reachability graph are tuples of n automata, one for each stack and each accepting the words formed by the symbols pushed onto the corresponding stack in the last k rounds of a run of the k-SMPDS. Any such component automaton is structured into layers, which are added incrementally one on the top of the lower ones essentially by applying the saturation procedure from [15]. We use the reachability graph in Section 5 as a main component in the construction of a multi-tape finite automaton accepting the reachable configurations of a given SMPDS.

For the rest of this section we fix a k-SMPDS $M = (k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$ and $\Gamma = \cup_{i=1}^{n} \Gamma_i$. We start defining k-layered automata. We assume that the reader is familiar with the basic concepts on finite automata and graphs.

k-layered Automata. A k-layered automaton A of M is essentially a finite automaton structured into $(k+1)$ layers whose set of states contains k copies of each $q \in Q$ (each copy belonging to a different layer from 1 through k) along with a new state q_F which is the sole final state and the sole state of layer 0. The input alphabet is Γ_h for some h, and the set of transitions contains only transitions of the form (s, γ, s') where the layer of s' is not larger than the layer of s. Moreover, there are no transitions leaving from q_F and every state is either isolated or connected to q_F. Formally, we have:

Definition 3. (k-LAYERED AUTOMATON) *A k-layered finite automaton A of M over Γ_h is a finite automaton $(S, \Gamma_h, \delta, S_0)$, where $h \in [n]$, Γ_h is the input alphabet and:*

- *$S = \cup_{i=0}^{k} S_i$ is the set of states where $S_0 = \{q_F\}$ and $S_i = \{\langle q, i\rangle \mid q \in Q\}$, for $i \in [k]$ (for $i \in [0,k]$, S_i denotes the layer i of A);*
- *$\delta \subseteq S \times (\Gamma_h \cup \{\varepsilon\}) \times S$ is the transition relation and contains transitions of the form (s, γ, s') such that $s \in S_i$, $s' \in S_j$, $i > 0$ and $i \geq j$;*
- *for each state $s \in S$, either there is a run from s to q_F or s is isolated (i.e., there are no transitions involving s).*

For $t \in [0,k]$, S_t is called the top layer *if t is the largest i such that there is at least a state of layer i which is connected to q_F (t is referred to as the* top-layer index*) and A is* full *if its top layer is S_k. The language accepted by A from a top-layer state $\langle q, t\rangle$, for $t > 0$, is denoted $\mathcal{L}(A, q)$. Moreover, $\mathcal{L}(A, q_F)$ denotes the language $\{\epsilon\}$ accepted A from q_F when the top-layer index is* 0 .

Note that two k-layered automata over the same alphabet Γ_h may differ only on the set of transitions and the only k-layered automaton over the alphabet Γ_h of top-layer index 0 is the one having no transitions. In the following, we often refer to a state $\langle q, i\rangle$ of a layered automaton as the copy of q in layer i .

We define the following transformations on k-layered automata. Let A be a k-layered automaton A with alphabet Γ_h and top-layer index t.

With *DownShift*(A) we denote the k-layered automaton A' obtained from A by shifting the transitions one layer down, i.e., the set of transitions of A' is the smallest set such that if $(\langle q,j\rangle,\gamma,s)$ is a transition of A with $j>1$, then $(\langle q,j-1\rangle,\gamma,s')$ is a transition of A' where s' is q_F, if $s\in S_0\cup S_1$, and is $\langle q',i-1\rangle$, if $s=\langle q',i\rangle$ for some $i\in[2,k]$ (note that if S_t is the top layer of A, with $t>1$, then S_{t-1} is the top layer of A').

With *Add*(A,s,s') we denote the k-layered automaton A' obtained from A by adding the transition (s,ε,s').

With *Saturate*(A) we denote the k-layered automaton A' obtained by applying to A the saturation procedure from [15] with respect to the internal transitions and the push and pop transitions involving stack h, and such that the new transitions that are added are all leaving from the top-layer states. Namely, let $t>0$ be the top layer index (if $t=0$, *Saturate* does nothing), the saturation procedure consists of repeating the following steps until no more transitions can be added (we let $\gamma\in\Gamma_h$ in the following):

– for an internal transition $(q,q')\in\delta$: $(\langle q',t\rangle,\varepsilon,\langle q,t\rangle)$ is added to set of transitions provided that $\langle q,t\rangle$ is connected to q_F;
– for a push-transition $(q,q',\gamma)\in\delta$: $(\langle q',t\rangle,\gamma,\langle q,t\rangle)$ is added to set of transitions provided that $\langle q,t\rangle$ is connected to q_F;
– for a pop-transition $(q,\gamma,q')\in\delta$: $(\langle q',t\rangle,\varepsilon,\langle q'',i\rangle)$, with $i\le t$, is added to the set of transitions provided that there is a run of A' from $\langle q,t\rangle$ to $\langle q'',i\rangle$ over γ (note that such a run may contain an arbitrary number of ε-transitions; also, $\langle q'',i\rangle$ is not isolated, and thus, connected to q_F by definition).

Note that all the transitions which cross layers, that get added in the above saturation procedure, are ε-transitions. Moreover, we recall that a similar procedure is given in [15] for constructing a finite automaton accepting all the configurations of a Pds P which are reachable starting from those accepted by a given finite automaton R. The iterated application of the saturation procedure over the execution contexts of an Mpds is already exploited in [14]. However, only runs with a constant bounded number of context switches are considered there and thus it is sufficient to iterate k times. Here, we need to iterate more. The essence of our algorithm is captured by the following definition.

Let A be a k-layered automaton with top layer index t and q be a state of M, define *Successor*(A,q,q') as follows, where q' is a state of M, if $t>0$, and $q'=q_F$, if $t=0$. When A is full, *Successor*(A,q,q') denotes the automaton that is obtained from A by first shifting the transitions one layer down, then adding to the resulting automaton an ε-transition from $\langle q,k\rangle$ to $\langle q',k-1\rangle$ and then applying the saturation procedure. Formally: *Successor*$(A,q,q')=$ *Saturate*(*Add*(*DownShift*$(A),\langle q,k\rangle,\langle q',k-1\rangle$)). Moreover, if $\langle q',k-1\rangle$ is connected to q_F then *Successor*(A,q,q') is a full k-layered automaton.

In the other case, i.e., if A is not full, *Successor*(A,q,q') denotes the automaton obtained as before except that no shifting is needed now. Formally, if $t>0$ then *Successor*$(A,q,q')=$ *Saturate*(*Add*$(A,\langle q,t+1\rangle,\langle q',t\rangle)$). Moreover, if $\langle q',t\rangle$ is connected to q_F then $Successor(A, q, q')$ is a k-layered automaton with top-layer index $(t+1)$. Finally, if $t = 0$, $Successor(A, q, q_F) = Saturate(Add(A, \langle q, 1\rangle, q_F))$ and is a k-layered automaton of top-layer index 1.

The Reachability Graph. In this section, we construct a finite graph that summarizes the computations of an SMPDS. In particular, each vertex stores, as tuples of k-layered automata, the portions of the stack contents that have been pushed onto the stacks in the last k rounds of any run leading to it. In each vertex is also stored the sequence of the states entered when the context switches occur in the last round. Each edge then summarizes the effects of the execution of an entire round on the information stored in the vertices.

The *reachability graph* $\mathcal{G}_M = (V_M, E_M)$ of M is defined as follows. The set of vertices V_M contains tuples of the form $(q_0, A_1, q_1, \ldots, A_n, q_n)$ where $q_0 \in Q$, each A_i is a k-layered automaton on alphabet Γ_i and for each $i \in [n]$, if the top-layer index of A_i is 0, then $q_i = q_F$, otherwise, $q_i \in Q$ is such that the copy of q_i in the top layer of A_i is not isolated (and thus connected to q_F). The set of edges E_M contains an edge from $(q_0, A_1, q_1, \ldots, A_n, q_n)$ to $(q'_n, A'_1, q'_1, \ldots, A'_n, q'_n)$ if $A'_i = Successor(A_i, q'_{i-1}, q_i)$, for $i \in [n]$ and denoting $q'_0 = q_0$ (note that in each vertex that has in-going edges, the first and the last components coincide).

Let A^0 be the k-layered automaton without any transition. We refer to each vertex of the form $(q, A^0, q_F, \ldots, A^0, q_F)$ such that $q \in Q_I$ as an *initial vertex* of $\mathcal{G}_M$ and denote it with in_q. Observe that since $q_F \notin Q$, $q \neq q_F$ always holds in the above tuple and thus each initial vertex has no in-going edges.

The following lemma relates the reachability within an SMPDS M with the reachability within $\mathcal{G}_M$.

Lemma 3. *Let M be a k-SMPDS. There is a path of m edges in $\mathcal{G}_M$ starting from an initial vertex in_q and ending at a vertex $v = (q_0, A_1, q_1, \ldots, A_n, q_n)$ if and only if there is a run ρ of m rounds in M starting from the initial configuration $\langle q, \varepsilon, \ldots, \varepsilon\rangle$ and ending at a configuration $C = \langle q_0, w_1, \ldots, w_n\rangle$ such that for $i \in [n]$, $y_i \in \mathcal{L}(A_i, q_i)$ where $(y_1, \ldots, y_n) = \mathrm{Last}_\rho(C, \min\{k, m\})$.*

Proof. We only sketch the proof for the "only if" part, the "if" part being similar. The proof is by induction on the length m of the path in $\mathcal{G}_M$. The base of the induction ($m = 0$) follows from the fact that there is a one-to-one correspondence between an initial vertex in_q of $\mathcal{G}$ and an initial configuration $\langle q, \varepsilon, \ldots, \varepsilon\rangle$ of M. Now, assume by induction hypothesis that the statement holds for $m \geq 0$. Thus, for any path of m edges from in_q to $v = (q_0, A_1, q_1, \ldots, A_n, q_n)$, there is a corresponding m-rounds run ρ of M from $\langle q, \varepsilon, \ldots, \varepsilon\rangle$ to $C = \langle q_0, w_1, \ldots, w_n\rangle$ such that for $i \in [n]$, $y_i \in \mathcal{L}(A_i, q_i)$ where $(y_1, \ldots, y_n) = Last_\rho(C, \min\{k, m\})$. Let $w_i = z'_i z''_i$ for $i \in [n]$ where $(z'_1, \ldots, z'_n) = Last_\rho(C, \min\{k-1, m\})$ (clearly, $z''_i = \varepsilon$ if $m < k$). Consider now an edge of $\mathcal{G}_M$ from v to $v' = (q'_0, A'_1, q'_1, \ldots, A'_n, q'_n)$. From the definition of $\mathcal{G}_M$ we get that $q'_0 = q'_n$, $A'_1 = Successor(A_1, q_0, q_1)$, and $A'_i = Successor(A_i, q'_{i-1}, q_i)$ for $i \in [2, n]$. Now, by a standard proof by induction on the number of applications of the rules of the saturation procedure, it is possible to show that, denoting $C_0 = C$ and for $i \in [n]$, there are $y'_i \in \mathcal{L}(A'_i, q'_i)$ and contexts $C_{i-1} \rightsquigarrow_M C_i$ where the stack i is active, and such that $y'_i z''_i$ is the content of stack i in C_i and q'_i is the location of C_i. Thus, $C \rightsquigarrow_M C_1 \ldots \rightsquigarrow_M C_n$ is a round and $C_n = \langle q'_n, y'_1 z''_1, \ldots, y'_n z''_n \rangle$ (recall that in each context $C_{i-1} \rightsquigarrow_M C_i$ only stack i can be updated). Therefore, appending this round as a continuation of run ρ, we get an $(m+1)$-rounds run with the properties stated in the lemma. $\square$

Let $Q^T \subseteq Q$ be a target set of locations of M. By the above lemma, we can solve the reachability problem for SMPDS by checking if any vertex $(q, A_1, q_1, \ldots, A_n, q_n)$, with $q \in Q^T$, can be reached in $\mathcal{G}_M$ from an initial vertex. Observe that the number of different k-layered automata for an alphabet Γ_i is at most $\chi_i = 2^{k|Q|+k|\Gamma_h||Q|^2+k^2|Q|^2}$. Since $\mathcal{G}$ has exactly $|Q_I|$ initial vertices, and all the other reachable vertices have the first and the last component that coincide, the number of vertices of $\mathcal{G}_M$ is at most $|Q_I| + \chi^n |Q|^n$ where $\chi = \max\{\chi_i \mid i \in [n]\}$, and thus exponential in the number of stacks, in the number of rounds and in the number of locations. Since we can explore the graph $\mathcal{G}_M$ on-the-fly, the proposed algorithm can be implemented in space polynomial in all the above parameters. With fairly standard constructions, it is possible to reduce the membership problem for a Turing machine to the location reachability problem for both a 2-stack k-SMPDS and an n-stack 1-SMPDS, thus showing PSPACE-hardness in both n and k. Therefore, by Lemma 3 we get:

Theorem 4. *The location reachability problem for* SMPDS *is* PSPACE-*complete, and hardness can be shown both with respect to the number of stacks and the number of rounds.*

5 Configuration Reachability for SMPDS

Regularity of the Reachable Configurations. Fix an SMPDS $M = (k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$. For $q \in Q$, with $Reach_q$ we denote the set of tuples $(w_1, \ldots, w_n)$ such that $(q, w_1, \ldots, w_n)$ is configuration which is reachable from an initial configuration.

Observe that in general this set may not be expressible as a finite union of the Cartesian product of regular sets. For example, consider the 2-stack 1-SMPDS M_1 with $\Gamma_1 = \{a\}$, $\Gamma_2 = \{b\}$, $Q_I = \{q_0\}$ and set of transitions $\delta = \{(q_0, q_1, a), (q_1, q_0, b)\}$. The only initial configuration is $\langle q_o, \varepsilon, \varepsilon \rangle$. It is simple to verify that $Reach_M(q_0) = \{(a^n, b^n) \mid n \geq 0\}$ that clearly cannot be expressed as finite union of the Cartesian product of regular languages.

In this section, we show that the sets $Reach_q$ are recognized by an n-tape finite automaton. An *n-tape finite automaton* (n-FA) A is $(S, I, \Sigma, \Delta, F)$ where S is a finite set of states, $I \subseteq S$ is the set of initial states, Σ is a finite set of input symbols, $F \subseteq S$ is the set of final states, and $\Delta \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times [n] \times S$ is the set of transitions. The meaning of a transition rule (s, τ, i, s') is that for a state s the control of A can move to a state s' on reading the symbol τ from tape i. Every time a symbol τ is read from a tape the corresponding head moves to next symbol (clearly, if $\tau = \varepsilon$, the head does not move). A tuple $(w_1, \ldots, w_n)$

is accepted by A if there is a run from $s \in I$ to $s' \in F$ that consumes the words $w_1, \ldots, w_n$ (one on each tape).

Now, we construct an n-FA R_q recognizing $Reach_q$ for a location q of M. A main part of this automaton is the reachability graph $\mathcal{G}_M$ definend in Section 4. In particular, the automaton R_q starts from a vertex $(q, A_1, q_1, \ldots, A_n, q_n) \in V_M$ and then moves backwards along a path up to an initial vertex of G_M. We recall that from Lemma 3, each edge on a path of $\mathcal{G}_M$ corresponds to a round of a run of M and thus moving backwards along a path corresponds to visiting a run of $\mathcal{G}_M$ round-by-round starting from the last round (*round-switch mode*). Therefore, on each visited vertex v, R_q can read the symbols (if any) that, along the explored run, are pushed onto a stack in the current round and then never popped out, and this can be done by simulating just the top layers of the k-layered automata of v that are required to move into this round (*simulation mode*).

To implement these two main modes of the automaton R_q, we store in each state a vertex v of $\mathcal{G}_M$ (the currently explored vertex) and a tuple of the states which are currently visited for each k-layered automaton A of v. We also couple each such state with a count-down counter that maintains the number of rounds that the corresponding A will stay idle before executing the top layer transitions (when this counter is 0, A gets executed). Such counters are used to synchronize the two modes of R_q. When all the counters are non-zero, then R_q moves backwards in $\mathcal{G}_M$, and decrements all the counters, and this is repeated until some counters are 0. As long as there is a counter d set to 0, R_q can execute for the corresponding k-layered automaton either a top-layer transition or an ε-transition to a state q in a lower layer. In both cases, the state coupled with d gets updated accordingly to the taken transition, and only in the second case d is set to the difference between the top layer and the layer of q. The automaton R_q accepts when it enters a state where all the counters are 0, the coupled states are all q_F (the final state of each k-layered automaton), and the current vertex of G is an initial vertex.

Formally, the n-FA $R_q = (S_q, I_q, \widetilde{\Gamma}_n, \Delta_q, F_q)$ is defined as follows.

- The set of states S_q of R_q contains tuples of the form $(v, q_1, d_1, \ldots, q_n, d_n)$ where $v = (p_n, A_1, p_1, \ldots, A_n, p_n) \in V_M$ and for $i \in [n]$, $q_i \in Q \cup \{q_F\}$ and $d_i \in [0, k]$.
- The set of initial states I_q contains states of the form $(v, q_1, 0, \ldots, q_n, 0)$ where $v = (q_n, A_1, q_1, \ldots, A_n, q_n)$ and $q_n = q$.
- The set of final states F_q contains states of the form $(v, q_F, 0, \ldots, q_F, 0)$ where v is an intial vertex of $\mathcal{G}_M$.
- Denoting $s = (v, q_1, d_1, \ldots, q_n, d_n)$ and $s' = (v', q'_1, d'_1, \ldots, q'_n, d'_n)$, where $v = (p_n, A_1, p_1, \ldots, A_n, p_n)$ and $v' = (p'_n, A'_1, p'_1, \ldots, A'_n, p'_n)$, the set of transition Δ_q contains tuples (s, τ, h, s') such that $s \notin F_q$ and one of the following cases applies (in the following description, if a component of s' is not mentioned then it is equal to the same component of s):

  [simulate within the top layer] $d_h = 0$ and $(\langle q_h, t\rangle, \tau, \langle q'_h, t\rangle)$ is a transition of A_h where t is the top-layer index of A_h;

  [simulate exit from the top layer] $d_h = 0$, $d'_h = t - t'$, $(\langle q_h, t\rangle, \tau, \langle q'_h, t'\rangle)$ is a transition of A_h, where t is the top-layer index of A_h and $t' < t$;

[round-switch] for $i \in [n]$, $d_i > 0$ and $d'_i = d_i - 1$, there is an edge from v' to v in $\mathcal{G}_M$.

Observe that Δ_q is such that in any run of R_q whenever a component q_i of a visited state $(v, q_1, d_1, \ldots, q_n, d_n)$ becomes q_F, then it stays q_F up to the end of the run.

With a standard proof by induction, by Lemma 3, we get that R_q accepts the language $Reach_q$. Therefore, we have the following theorem.

Theorem 5. *For each $q \in Q$, the set* Reach_q *is accepted by an n-*FA.

Reachability of SMPDS. Fix an SMPDS $M = (k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$ and a set of configurations $T = P \times \mathcal{L}(B_1) \times \ldots \times \mathcal{L}(B_n)$, where $P \subseteq Q$.

By Theorem 5, the reachability problem for SMPDS can be reduced to checking the emptiness of $\cup_{q \in P}(Reach_q \cap L)$ where $L = \mathcal{L}(B_1) \times \ldots \times \mathcal{L}(B_n)$. Denoting with $A_i \times B_i$ the standard cross product construction synchronized on the input symbols (ε transitions can be taken asynchronously), the construction of R_q given above can be modified such that in the simulation mode it tracks the behaviors of $A_i \times B_i$ instead of just the k-layered automaton A_i. Denote with R_q^T the resulting n-FA. We observe that in R_q^T, the simulation of each B_i starts from the initial states, and then the B_i-component gets updated only in the simulation mode in pair with the couped k-layered automaton. For the lack of space we omit the explicit construction of R_q^T.

The number of states of each R_q^T is at most $|V_M|\,(|Q|+1)^n(k+1)^n\,\chi^n$, where χ is the maximum over the number of states of $B_1, \ldots, B_n$. Recall that the number of vertices $|V_M|$ of $\mathcal{G}_M$ is exponential in n, $|Q|$ and k. Thus, the number of states of R_q^T is also exponential in the same parameters. Again, we can explore on-the-fly the state space of each R_q^T, thus we can check the emptiness of R_q^T in polynomial space, and in time exponential in n, $|Q|$ and k. Since each instance of the location reachability is also an instance of the general reachability problem, by Theorems 4 and 5 we get:

Theorem 6. *The reachability problem for* SMPDS *is* PSPACE*-complete, and hardness can be shown both with respect to the number of stacks and the number of rounds.*

6 Conclusion and Future Work

We have introduced a decidable restriction of multistack pushdown systems that allows unboundedly many context switches. Bounding the scope of the matching relations of push and pop operations on each stack has the effect of bounding the amount of information that can flow out of a stack configuration into the other stacks in the rest of the computation. For the resulting model we have shown that the set of reachable configurations is recognized by a multitape finite automaton, and that location and configuration reachability can be solved essentially by searching a graph of size exponential in the number of control states, stacks and rounds in a scope.

We see mainly two future directions in this research. We think that it would be interesting to experiment the effectiveness of the verification methodology based on our approach, by implementing our algorithms in a verification tool and compare them with competing tools. If on one side the considered reachability problem has a theoretically higher complexity compared to the case of bounded context-switching, on the other side smaller values of the bound on the number of context switches are likely to suffice for several systems. Moreover, our model presents the right features for studying the model-checking of concurrent software with respect to properties that concern non-terminating computations or require to explore unboundedly many contexts.

References

1. Atig, M.F., Bouajjani, A., Qadeer, S.: Context-bounded analysis for concurrent programs with dynamic creation of threads. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 107–123. Springer, Heidelberg (2009)
2. Bouajjani, A., Fratani, S., Qadeer, S.: Context-bounded analysis of multithreaded programs with dynamic linked structures. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 207–220. Springer, Heidelberg (2007)
3. La Torre, S., Madhusudan, P., Parlato, G.: Sequentializing parameterized programs, `http://www.dia.unisa.it/dottorandi/parlato/papers/sequ-parameterized.pdf`
4. La Torre, S., Madhusudan, P., Parlato, G.: Context-bounded analysis of concurrent queue systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
5. 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)
6. La Torre, S., Madhusudan, P., Parlato, G.: Analyzing recursive programs using a fixed-point calculus. In: PLDI, pp. 211–222 (2009)
7. La Torre, S., Madhusudan, P., Parlato, G.: Reducing context-bounded concurrent reachability to sequential reachability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 477–492. Springer, Heidelberg (2009)
8. La Torre, S., Madhusudan, P., Parlato, G.: Model-checking parameterized concurrent programs using linear interfaces. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 629–644. Springer, Heidelberg (2010)
9. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS, pp. 161–170 (2007)
10. Lal, A., Reps, T.: Reducing concurrent analysis under a context bound to sequential analysis. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 37–51. Springer, Heidelberg (2008)
11. Lal, A., Touili, T., Kidd, N., Reps, T.W.: Interprocedural analysis of concurrent programs under a context bound. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 282–298. Springer, Heidelberg (2008)
12. Madhusudan, P., Parlato, G.: The tree width of auxiliary storage. In: POPL, pp. 283–294 (2011)

13. Musuvathi, M., Qadeer, S.: Iterative context bounding for systematic testing of multithreaded programs. In: PLDI, pp. 446–455 (2007)
14. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. TACAS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
15. Schwoon, S.: Model-Checking Pushdown Systems. Ph.D. thesis, Technische Universität München (2002)
16. Suwimonteerabuth, D., Esparza, J., Schwoon, S.: Symbolic context-bounded analysis of multithreaded java programs. In: Havelund, K., Majumdar, R. (eds.) SPIN 2008. LNCS, vol. 5156, pp. 270–287. Springer, Heidelberg (2008)