Information and Computation 275 (2020) 104588

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Reachability of scope-bounded multistack pushdown systems

Salvatore La Torre [a,*], Margherita Napoli [a], Gennaro Parlato [b]

[a] Università degli Studi di Salerno, Italy
[b] Università degli Studi del Molise, Italy

ARTICLE INFO

Article history:
Received 21 December 2018
Received in revised form 7 January 2020
Accepted 6 May 2020
Available online 13 May 2020

Keywords:
Multistack pushdown automata
Reachability
Verification

ABSTRACT

A multi-stack pushdown system is a natural model of concurrent programs. The basic verification problems are undecidable and a common trend is to consider under-approximations of the system behaviors to gain decidability. In this paper, we restrict the semantics such that a symbol that is pushed onto a stack s can be popped only within a given number of contexts involving s, i.e., we bound the scope (in terms of number of contexts) of matching push and pop transitions. This restriction permits runs with unboundedly many contexts even between matching push and pop transitions (for systems with at least three stacks). 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 can be captured by a finite automaton.

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 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 behavior 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.

The analysis of multi-stack pushdown systems within a bounded number of execution contexts (in each context only one stack is used) has been proposed as an effective method for finding bugs in concurrent programs [1]. 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 [2]). 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 [3,1].

* Corresponding author at: Università degli Studi di Salerno, Dipartimento di Informatica, Via Giovanni Paolo II, 132 - 84084 Fisciano (SA), Italy.
E-mail addresses: slatorre@unisa.it (S. La Torre), gennaro.parlato@unimol.it (G. Parlato).

https://doi.org/10.1016/j.ic.2020.104588
0890-5401/

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 bounding the number of context-switching 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 over a bounded number of contexts of the same stack, i.e., when executing a returned call a thread can be preempted only for a bounded number of times. In other words, we bound the *scope* of the matching push and pop operations of a stack s in terms of the number of context switches from s. Note that under such a restriction, whenever a symbol is pushed onto a stack s, it is popped either within a bounded number of contexts of s or never. 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).

Our main technical contribution is to show the decidability of the reachability problem for SMPDS. We consider both the *location* and *configuration* reachability problems. The configuration reachability problem asks whether a configuration (a control state along with the stack contents) within a set of target configurations of the SMPDS is reachable. We consider sets of target configurations given as the cross product of a set of control states and a regular language of stack contents for each thread. In the location reachability problem the request is only with respect to a set of target control states (location) regardless of the actual stack contents of the reached configurations.

For the location reachability, we give a fixed-point decision algorithm based on the notion of *thread interface* introduced in [4]. A *h-thread interface* summarizes the starting and ending control states of h consecutive contexts of a thread within a run of a MPDS, assuming that the thread stack is empty when the first of such contexts starts. Our algorithm first guesses for each thread a thread interface, then it advances in the simulation of a run of the MPDS by stitching the contexts of these interfaces until one of them will be entirely consumed (this corresponds to advancing in the run by macro-steps covering each an entire context). At this point, a new thread interface is taken for the corresponding thread and the simulation is resumed. The algorithm stores *tuples* formed of a control state and a thread interface suffix for each thread (the remaining parts of the guessed thread interfaces still to be used). It halts as soon as a target control state is reached or no new such tuples can be added (and thus no new simulation is possible). For termination, we show that any k-scoped run (i.e., a run where the scope of the matching relations is bounded by k) can be captured by using only h-thread interfaces with $h \leq k$, and thus for a given k, a finite number of thread interfaces will suffice to explore all such runs. Therefore, by restricting the guesses to only h-thread interfaces with $h \leq k$, the algorithm is guaranteed to terminate. As for the complexity, our algorithm can be implemented to take time exponential in k and n and polynomial in d, where n is the number of threads and d is the number of control states of the SMPDS.

In thread interfaces, the content of thread stack is entirely abstracted away. Thus for the general reachability problem we introduce a new abstraction called *layered stack automaton*. For a thread T, an ℓ-layered stack automaton captures the top portion of its stack which corresponds to the symbols that were pushed within the last t contexts of T. The automaton is structured into layers that are added incrementally by applying for each layer a saturation procedure similar to the one given in [5] for standard (one-stack) pushdown systems. Since in k-scoped runs only the symbols that were pushed within the last k contexts of a thread can be popped, we can restrict to ℓ-layered stack automata with $\ell \leq k$ to keep track of the meaningful top portion of the stack during a computation. We then relate the layered automata of a thread via a *successor* relation: a layered automaton A is a successor of a layered automaton B if A is obtained by adding a new layer to B via the saturation procedure. We thus reconstruct the stack content through the portions captured by bounded layered automata connected via the successor relation. We call the resulting finite automaton a *thread automaton*. To capture the set of reachable configurations, we thus construct another finite automaton $\mathcal{R}$ that uses as components a thread automaton for each thread and synchronizes all of them by picking the next context (among the next possible ones for each thread automaton) such that it can be stitched to the last processed one. Assuming that the stack contents of the target set are expressed by finite automata, we can modify $\mathcal{R}$ to simulate such automata in parallel with the thread automata by a standard cross product, that reduces the configuration reachability to standard reachability for finite automata. As for the complexity, the outlined algorithm can be implemented to take time exponential in k^2, d^2 and n and polynomial in the size of the target set representation, where n is the number of threads and d is the number of control states of the SMPDS.

We observe that both our algorithms can be implemented to take polynomial space. For the first one, at each iteration instead of maintaining a set of tuples we can just maintain one tuple (the last computed one) and nondeterministically select the next rule to apply. The algorithm will halt as soon as we compute a tuple with control state in the target set or we have reached a number of iterations that equals the number of possible different tuples. This can be determined by counting the number of different thread interfaces with bound k. For the second one, we recall that it is well known that reachability in finite automata can be decided in logarithmic space, and since we can explore the resulting automaton on-the-fly, we clearly get a PSPACE upper bound also for the configuration reachability for SMPDS. We show that both location reachability and configuration reachability for SMPDS are PSPACE-complete by providing a matching lower bound. Indeed, we show the upper bound is tight with respect to both the number of stacks and the bound k. For this, we sketch two reductions from the membership problem of Turing machines working in polynomial space to the location reachability problem for respectively n-stack 2-SMPDS and 2-stack $2k$-SMPDS.

As a further result, we compare the state coverage by exploring scoped runs as opposed to other restrictions for MPDS that have been introduced in the literature. As observed above, the bounded scope restriction is an extension of bounded context-switching. Interestingly, we show that this restriction allows us to achieve a state space exploration of MPDS that is orthogonal with respect to bounding the number of phases [6], restricting to ordered runs [7], and restricting to runs that can be encoded in bounded path-trees [8].

The rest of the paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we introduce the notion of MPDS with related notation and definitions. In Section 4, we define the reachability problem and discuss the state coverage ensured by bounded scope runs as opposed to existing limitations. Our solutions to the location reachability and configuration reachability problems are given respectively in Section 5 and Section 6. We also address the computational complexity in the respective sections. We give our conclusions and future directions in Section 7.

2. Related work

In this paper we re-elaborate the results of [9] as follows. First, here we use the notion of bounded scope runs that was introduced later in [10]. This notion captures more behaviors with respect to the original one given in [9] and in particular can account for unboundedly many contexts of the other threads between a push and its matching pop transition. We thus re-elaborate accordingly the decision algorithm for the configuration reachability problem from [9] and show its correctness in detail. For the location reachability, we give a simpler algorithm that adapts the solution given in [11], which was also given for the original notion of bounded scope runs and by assuming a round-robin scheduling of the threads. Finally, the discussion on the state space coverage of bounded scope runs is given in more detail and extended to account for the results appeared after the publication of [9].

Our fixed-point algorithm for the location reachability of MPDS uses the concept of thread interface introduced in [4]. Thread interfaces are a simpler artifact than finite automata and can be easily encoded for efficient symbolic search. In fact, our fixed-point algorithm has a direct implementation in the tool GETAFIX, a framework that supports the writing in a fixed-point calculus of model-checkers for sequential and concurrent Boolean programs (see [12]).

Our decision algorithm for configuration reachability of MPDS relies on the saturation procedure used for the analysis of pushdown systems [5]. This procedure was already reused in [1] for solving the reachability of MPDS within a bounded number of context switches. As in [1], we compute the set of reachable configurations as tuples of automata accepting configurations of each stack, and construct the automata by iterating the saturation algorithm from [5] into layers, each for execution context. However, in [1] 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 of the corresponding thread.

Since their introduction [9], the theory of bounded scope MPDS has been enriched with more results. In [13], SMPDS define a robust class of visibly languages that enjoys the main properties of regular languages such as: decidability of emptiness, membership, inclusion, equivalence and universality; closure under union, intersection, complement and determinization; MSO characterization and Parikh theorem. SMPDS also admit *sequentialization*, i.e., simulation by standard pushdown systems, and the corresponding class of behavior graphs (nested words with multiple stack relations expressing system computations) has bounded treewidth [11]. Bounded treewidth for this class is also shown in [14] via the notion of splitwidth. Finally, SMPDS have a natural and meaningful semantics for infinite computations which allows to observe also infinitely many interactions between the different threads. The model-checking problem of SMPDS against linear temporal logic is shown to be decidable for LTL in [15] and for a concurrent version of CARET [16] in [10].

The bounded scope restriction is introduced as a generalization of the bounded context-switching [1]. This notion has been successfully used in recent research: model-checking tools for concurrent programs (see [12,17–22]); translations of concurrent programs to sequential programs reducing bounded context-switching reachability to sequential reachability [23,17,21,22]; model-checking tools for Boolean abstractions of parameterized programs (concurrent programs with unboundedly many threads each running one of finitely many codes) [4]; sequentialization algorithms for parameterized programs [24]; model-checking of programs with unbounded dynamic creation of threads [25] and more liberal scheduling of threads [26,27]; analysis of systems with heaps [28], systems communicating using queues [29], and weighted pushdown systems [3], complexity results [30].

More decidable restrictions that extend bounded context switching for MPDS have been considered in literature. In [6], the notion of context is relaxed and the behaviors 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 location reachability problem in this model turns out to be 2ETIME-complete [6,31]. In [32] the set of predecessor configurations up to k phases is shown to be regular. Model-checking for bounded phase MPDS is studied in [33–35]. 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 and this is indeed confirmed by our results which show that the set of configurations that are reachable within a bounded number of phases can be incomparable with the one reachable by bounded scope runs. 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.

Another decidable restriction of MPDS is *ordered* MPDS [7], where symbols can be popped from stack i only if all the stacks from 1 to $i-1$ are empty. Visibly 2-stack ordered MPDS are studied in [36,37]. The closure under complement for ordered MPDSs, and thus the decidability of inclusion, equivalence and universality, is given in [8].

In MPDS with *budgets* each thread can perform at most k consecutive context switches unless its stack depth goes below the given bound d [38]. This restriction in some sense enforces the bounded scope restriction when the stacks pass the depth threshold, and does not seem to add to it more than a finite state store of exponential size (which can be explored taking polynomial space).

A general decidability result shows that most of the syntactic restrictions placed on MPDS lead to classes of graphs representing the runs of the MPDS that are MSO-definable and of bounded treewidth [39]. Bounded scope and bounded phase restrictions have been studied for concurrent collapsible MPDS [40], and the bounded phase restriction also for parity games on MPDS[41]. The notion of bounded context-switching has been recently investigated for valence systems [42].

3. Multi-stack pushdown systems with scope-bounded runs

In this section we introduce the notations and definitions we will use in the rest of the paper. We assume that the reader is familiar with the basic concepts on finite automata, trees and graphs.

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 $[1, j]$.

Multistack pushdown systems. 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 state by maintaining unchanged the stack contents. For the ease of presentation and without loss of generality we assume that 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 $\delta \subseteq (Q \times Q) \cup (Q \times Q \times \widetilde{\Gamma}_n) \cup (Q \times \widetilde{\Gamma}_n \times Q)$ is the transition relation. A pushdown system (PDS) is a MPDS with just one stack. For each $i \in [n]$, with T^i_M we denote the i-th thread of M, i.e., the PDS $(Q, Q_I, \Gamma_i, \delta_i)$ where $\delta_i = \delta \cap ((Q \times Q) \cup (Q \times Q \times \Gamma_i) \cup (Q \times \Gamma_i \times Q))$. □

We fix an n-stack alphabet $\widetilde{\Gamma}_n = \bigcup_{i=1}^n \Gamma_i$ 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_i, for some $i \in [n]$. A *configuration* of a MPDS M is a tuple $C = \langle\langle q, w_1, \ldots, w_n \rangle\rangle$, where $q \in Q$ and each $w_i \in \Gamma_i^*$ is a stack content. Moreover, C is *initial* if $q \in Q_I$ and $w_i = \varepsilon$ for every $i \in [n]$. A transition $\langle\langle q, w_1, \ldots, w_n \rangle\rangle \rightarrow_M \langle\langle q', w'_1, \ldots, w'_n \rangle\rangle$ is such that one of the following cases holds (M is omitted whenever it is clear from the context):

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

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.

For a MPDS M and $h \in [n]$, a *context* of thread T^h_M, *h-context* for short, is a portion of a run of M where the pop and push transitions are all over stack h. More formally, a *h-context* from C to C', denoted $C \rightsquigarrow^h_M C'$, is a run $C \rightsquigarrow_M C'$ whose transitions are all from δ_h, i.e., involve only thread T^h_M.

For the ease of presentation, in the following, we will abuse the notation and identify runs of the h-th thread and h-contexts. In particular, we will also denote a context $\langle\langle q, w_1, \ldots, w_n \rangle\rangle \rightsquigarrow^h_M \langle\langle q', w'_1, \ldots, w'_n \rangle\rangle$ simply as $\langle\langle q, w_h \rangle\rangle \rightsquigarrow^h_M \langle\langle q', w'_h \rangle\rangle$. Similarly, a transition $\langle\langle q, w_1, \ldots, w_n \rangle\rangle \rightarrow_M \langle\langle q', w'_1, \ldots, w'_n \rangle\rangle$ within an h-context will be also denoted as $\langle\langle q, w_h \rangle\rangle \rightarrow^h_M \langle\langle q', w'_h \rangle\rangle$.

For each thread T^h_M, we are also interested in sequences of h-contexts where each context builds on the stack content left by the previous one in the sequence. In other words, such a sequence would form a run of T^h_M except that each context does not need to start from the control state that ended the previous one. Formally, a *multiple context run* of T^h_M is a sequence of h-contexts $\rho_1, \ldots, \rho_m$ such that $w_1 = \varepsilon$ and $w'_i = w_{i+1}$ for $i \in [m-1]$, where $\rho_i = \langle\langle q, w_i \rangle\rangle \rightsquigarrow^h_M \langle\langle q', w'_i \rangle\rangle$ for $i \in [m]$.

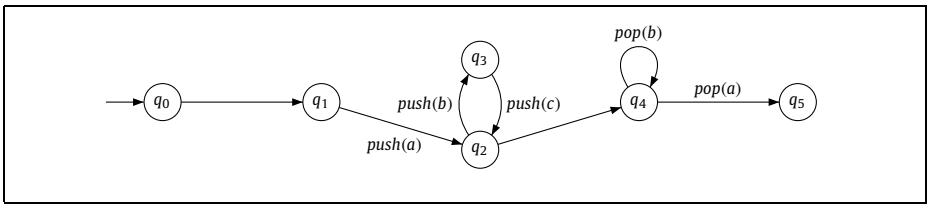

Fig. 1. Graphical representation of the MPDS M_1 from Example 1.

Example 1. Fig. 1 gives a 2-stack MPDS M_1 with stack alphabets $\Gamma_1 = \{a, b\}$ and $\Gamma_2 = \{c\}$. The starting state of M_1 is q_0 and its transition relation is: $\{(q_0, q_1), (q_1, q_2, a), (q_2, q_3, b), (q_3, q_2, c), (q_2, q_4), (q_4, b, q_4), (q_4, a, q_5)\}$.

A typical execution of the system M_1 from the initial state q_0 starts with an internal move to q_1 and then pushing a onto stack 1. Thus, M_1 iteratively pushes b onto stack 1 and c onto stack 2, reaching a configuration of the form $\langle\langle q_2, b^r a, c^r\rangle\rangle$. From this configuration, M_1 can move to q_4 and pop b from stack 1. When all b's are popped out, M_1 can also pop a from stack 1 and finally reach the control state q_5 with stack 1 empty, in a configuration of the form $\langle\langle q_5, \varepsilon, c^r\rangle\rangle$. □

Scope-bounded runs. 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 consider here a semantics that restricts this. In particular, given $k > 0$, we restrict the runs of a MPDS such that a pop transition from stack h is allowed to execute only when the symbol at the top of the stack was pushed within the last k contexts of h. Thus, we place a constraint on the matching relations (i.e., the relations defined by pairing the pushes and the corresponding pops) that can be defined in the runs.

We introduce first some notation. We fix a run $\rho = C_0 \rightarrow_M C_1 \cdots \rightarrow_M C_m$ and denote $\rho[i, j] = C_i \rightarrow_M \cdots C_j$ and $|\rho| = m$.

A *decomposition* of ρ is $\rho_1, \ldots, \rho_\ell$ where $\rho_i = \rho[j_{i-1}, j_i]$ for $i \in [\ell]$ where $0 = j_0 < \ldots < j_\ell = m$. Note that the sequence of all h-contexts from a decomposition of ρ always forms a multiple context run of T^h_M, however not all the multiple context runs of T^h_M can be completed to form a run of M.

An h-context $\rho[i, j]$ is *maximal* if it is either the entire run, i.e., $i = 0$ and $j = m$, or it: (1) contains at least a push or a pop transition and (2) cannot be extended by including other push or pop transitions of stack h, i.e., for each $i' \leq i$ and $j' \geq j$ such that $\rho[i', j']$ is still an h-context, both $\rho[i', i]$ and $\rho[j, j']$ do not contain push or pop transitions of stack h. We observe that according to this definition a maximal context can be extended with an internal transition and the resulting context be still maximal. As an example, consider the run described in Example 1 with $r = 2$. The portion $\langle\langle q_1, \varepsilon, \varepsilon\rangle\rangle \rightarrow_M \langle\langle q_3, ba, \varepsilon\rangle\rangle$ and its extension with the first transition, i.e., $\langle\langle q_0, \varepsilon, \varepsilon\rangle\rangle \rightsquigarrow_M \langle\langle q_3, ba, \varepsilon\rangle\rangle$, are both maximal 1-contexts. Any other extension would include at least a push onto stack 2 and thus it would not be a 1-context. Also, note that $\langle\langle q_0, \varepsilon, \varepsilon\rangle\rangle \rightarrow_M \langle\langle q_1, \varepsilon, \varepsilon\rangle\rangle$ is not maximal for any stack and can be part only of a maximal 1-context. The internal transition (q_2, q_4) instead can be part of the maximal 1-context $\langle\langle q_2, b^2 a, c^2\rangle\rangle \rightsquigarrow_M \langle\langle q_5, \varepsilon, c^2\rangle\rangle$ or of the maximal 2-context $\langle\langle q_3, b^2 a, c\rangle\rangle \rightsquigarrow_M \langle\langle q_4, b^2 a, c^2\rangle\rangle$. In general, internal moves can be added to adjacent maximal contexts still resulting into maximal contexts.

With $contexts_h(i, j)$ we denote the number of h-contexts contained in a decomposition of $\rho[i, j]$ into maximal contexts. Note that $contexts_h(i, j)$ is well-defined since the number of maximal h-contexts is independent of the actual decomposition that is chosen.

For a stack h and indices $i, j \in [0, m-1]$, (i, j) is *h-matching* in ρ if: $i < j$, $C_i \rightarrow C_{i+1}$ is a transition that pushes a symbol onto stack h and this symbol stays onto stack h until transition $C_j \rightarrow C_{j+1}$ pops it. When this is the case, we say that a push $C_i \rightarrow C_{i+1}$ and a pop $C_j \rightarrow C_{j+1}$ match each other.

A run ρ is *k-scoped* if for each stack h and indices $i, j \in [0, m]$ such that $(i, j-1)$ is h-matched, we have $contexts_h(i, j) \leq k$. Analogously a multiple context run of T^h_M, say $\rho_1, \ldots, \rho_m$, is *k-scoped* if for each $i, j \in [m]$ such that there is a push in ρ_i that is matched in ρ_j, we have $j - i < k$. Clearly, the multiple context runs obtained from a decomposition into maximal contexts of a k-scoped run are k-scoped too.

Example 2. Fig. 2 gives a 3-stack MPDS M_2 with stack alphabets $\Gamma_1 = \{a\}$, $\Gamma_2 = \{b\}$ and $\Gamma_3 = \{c\}$. The starting state of M_2 is q_0 and its transition relation is: $\{(q_0, q_1, a), (q_1, q_2, b), (q_2, q_3, c), (q_3, b, q_1), (q_1, a, q_4)\}$.

A typical execution of M_2 starts from the initial state q_0 and pushes a on stack 1. Then it iterates the following steps: push b on stack 2, push c on stack 3 and pop b. After $r \in \mathbb{N}$ such iterations, M_2 pops a from stack 1 and reaches configuration $\langle\langle q_4, \varepsilon, \varepsilon, c^r\rangle\rangle$.

Fig. 3 gives a graphical representation of such an execution for $r = 2$. In the figure, we denote by dashed arrows the matching push and pop transitions, and use different colors to distinguish among the different threads. Full arrows capture the linear ordering among the transition within the run. A context is thus denoted with a chain of nodes of the same color linked though full arrows. Except for the middle context that is formed of a pop transition followed by a push transition of $T^2_{M_2}$, any other context of the run is formed of just one transition. This run is clearly 2-scoped. In fact, as shown by the dashed arrows, each matched push transition is paired with a pop transition within the next context of the same thread.

We observe that indeed all the runs of M_2 are 2-scoped. In fact, any possible run of M_2 is a prefix of the above described execution for some $r \geq 0$. Moreover, symbols a and b are popped out at most within the next context of the respective stacks, and symbols c are never popped. □

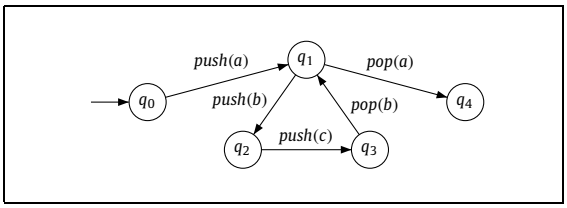

Fig. 2. Graphical representation of the MPDS M_2 from Example 2.

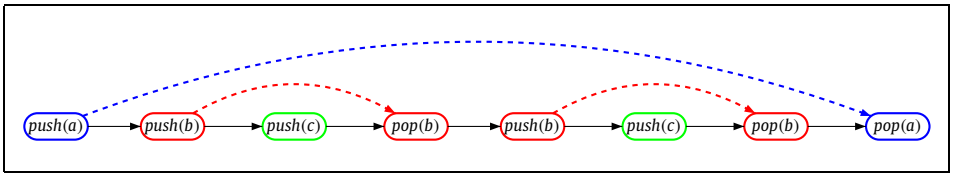

Fig. 3. Graphical illustration of the push/pop matching and context splitting of a run of the MPDS M_2 from Example 2. The matching push and pop transitions are linked with dashed arrows and the colors blue, red and green are used to denote respectively the contexts of $T^1_{M_2}$, $T^2_{M_2}$ and $T^3_{M_2}$.

Definition 2. (SCOPE-BOUNDED MPDS) A *k-scoped multi-stack pushdown system* (k-SMPDS) with n stacks is $M = (k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$ where $k \in \mathbb{N}$ and $(Q, Q_I, \widetilde{\Gamma}_n, \delta)$ is a MPDS. A run of M is any k-scoped run of $(Q, Q_I, \widetilde{\Gamma}_n, \delta)$. □

For a MPDS $M = (Q, Q_I, \widetilde{\Gamma}_n, \delta)$, we often denote with (k, M) the corresponding k-SMPDS $(k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$.

4. Reachability in MPDS

In this section, we will define the reachability problem for MPDS and recall the main restrictions that have been studied in the literature. Then, we will discuss the state space coverage for scope-bounded runs and compare it with the other known restrictions.

Reachability. A *target set of configurations* for M is $S \times R_1 \times \ldots \times R_n$ such that $S \subseteq Q$ and for $i \in [n]$, $R_i \subseteq \Gamma_i^*$ is a regular language. Given a MPDS $M = (Q, Q_I, \widetilde{\Gamma}_n, \delta)$ and a target set of configurations F, the *reachability problem* asks to determine whether there is a run of M from an initial configuration C_0 to a configuration $C \in F$. We consider also a restricted version of this problem where we are only interested in the control state and not in the stack contents of the reached configurations. Formally, the *location reachability problem* for MPDS is defined as the reachability problem with respect to a target set of the form $S \times \Gamma_1^* \times \ldots \times \Gamma_n^*$. In the following, we will also refer to the general reachability problem, where the reached configurations matter, as the *configuration reachability problem*.

It is well known (see for example [43]) 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.

Known decidable restrictions. Decidability can be gained by imposing some restrictions on the runs of a MPDS. Below, we recall the main restrictions that have been studied in the literature (the bounding parameters are assumed to be encoded in unary).

Bounded-context switching. A *k-context run* of M is a run formed as the concatenation of k contexts [1] (a similar restriction can be obtained by restricting to k rounds). The reachability problem *within k contexts* is the (location) reachability restricted to the sole k-contexts runs.

Theorem 2. *[1,3] The (location) reachability problem within k contexts for* MPDS *is NP-complete.*

Bounded-phase. A *phase* 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) [6]. 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.

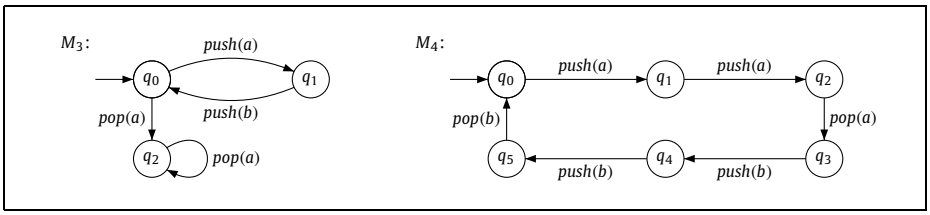

Fig. 4. Graphical representation of the MPDS M_3 and M_4.

Theorem 3. *[6,31] The location reachability problem within k phases for* MPDS *is* 2ETIME*-complete.*[1]

Ordered runs. A run of a MPDS M is *ordered* if the stacks of M are numbered $1, 2, \ldots, n$ and all the pop transitions are executed only on the lowest numbered non-empty stack [7]. Recently a further restriction has been introduced, called *adjacent ordered*, with additional requirement that, during a phase associated to stack i push transitions are allowed only on stacks $i-1$ and $i+1$ [44]. It is known that the runs up to k phases of a MPDS can be simulated by ordered runs using $2k$ stacks [45].

Theorem 4. *[44,45] The location reachability problem for* MPDS *restricted to ordered (resp. adjacent ordered) runs is* 2ETIME*-complete (resp.* EXPTIME*-complete).*

Bounded path-tree. A *stack tree* encoding a run ρ of a MPDS M is a binary tree obtained as follows. The first transition of ρ labels the root, and then each following transition labels the left child of the node labeled by the previous transition unless it is a matched pop transition. If this is the case instead, it labels the right child of the matching push transition. ρ is *k-path-tree* if it can be encoded into a *stack tree* and there is a walk in the tree that, starting from the root, visits all the nodes such that: the nodes are discovered according to the linear order of the corresponding transitions in ρ and each node is not visited more than k times [8].

It is known that the runs up to d phases of a MPDS are k-path-tree with $k = 2^d + 2^{d-1} + 1$ and ordered runs of a MPDS with n stacks are k-path-tree with $k = (n+1) \cdot 2^{n-1} + 1$ (see [8]).

Theorem 5. *[8] The location reachability problem for* MPDS *restricted to k-path-tree runs is* EXPTIME*-complete.*

Comparing the state space coverage. In the following, we briefly discuss the scope-bounded restriction in terms of coverage of the reachable state space of a multi-stack pushdown system.

We start observing that though all the runs of the MPDS M_2 given in Example 2 are 2-scoped, in general given a $k > 0$, k-scoped runs may not suffice to cover the entire state space of a MPDS. In fact, consider the MPDS M_1 from Example 1: for each $k \geq 1$, the configuration $\langle\langle q_5, \varepsilon, c^k \rangle\rangle$ is reachable in the SMPDS $(k+1, M_1)$ and is not reachable in any of the SMPDS (h, M_1) for $h \leq k$. Thus we can state the following result.

Lemma 6. *For any* MPDS *M and $k > 0$, if a configuration C' is reachable from C in the* SMPDS *(k, M), then C' is also reachable from C in M. Vice-versa, there is a* MPDS *M' such that for each $k > 0$ there is a reachable configuration C that is not reachable in the* SMPDS *(k, M').*

Now, fix $\Gamma_1 = \{a\}$, $\Gamma_2 = \{b\}$. Let M_3 be the 2-stack MPDS from Fig. 4. Since the only pop transitions are from the same stack, any run of M_3 is 1-phase. It is simple to see that a configuration $C_k = \langle\langle q_2, \varepsilon, b^k \rangle\rangle$ for $k \in \mathbb{N}$, is not reachable in the SMPDS (h, M_3) for any $h \leq k$.

Moreover, let M_4 be the 2-stack MPDS from Fig. 4. Since along any run, the stack symbols are popped within the same contexts where they are pushed, any run of M_4 is 1-scoped. However, a configuration $C_k = \langle\langle q_0, a^k, b^k \rangle\rangle$ for $k \in \mathbb{N}$ is not reachable with a run with less than $2k$ phases.

Thus, we get that the notions of scope-bounded reachability and phase-bounded reachability are not comparable. Therefore, they give two orthogonal ways of exploring the state space of a MPDS.

Lemma 7. *There is a* MPDS *M such that any reachable configuration can be reached within one phase, and for each $k > 0$ there is a configuration C that is not reachable in the* SMPDS *(k, M).*

There is a MPDS *M such that any reachable configuration can be reached also in $(1, M)$, and for any $k > 0$ there is a configuration C that is not reachable within k phases.*

[1] 2ETIME is the class of languages accepted by Turing machines in $2^{2^{O(n)}}$ time.

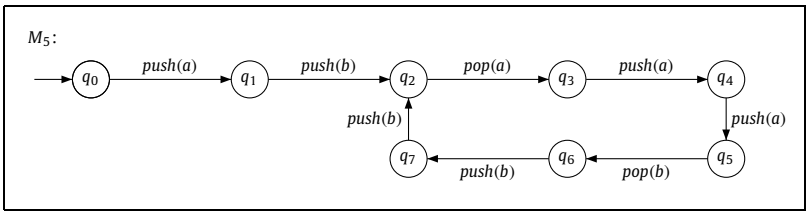

Fig. 5. Graphical representation of the MPDS M_5.

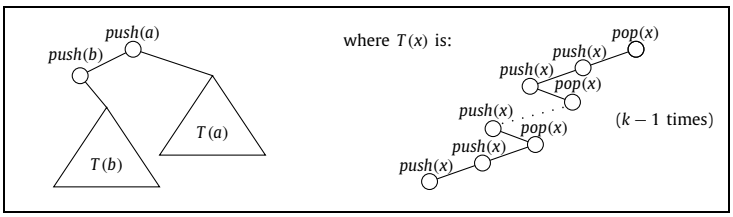

Fig. 6. Stack tree of the M_5 run reaching $\langle\langle q_2, a^k, b^k\rangle\rangle$.

The same result also holds with respect to (adjacent) ordered runs. In fact, consider again the 2-stack MPDS M_3 from Fig. 4. Since all the pop transitions involve stack 1, all the runs are clearly ordered. Moreover, since there are only two stacks, they are also adjacent ordered. As already observed above, for each $k > 0$, configuration $\langle\langle q_2, \varepsilon, b^k\rangle\rangle$ is not reachable in the SMPDS (k, M_3). On the other hand, in Example 2, any configuration $\langle\langle q_1, a, \varepsilon, c^k\rangle\rangle$, with $k > 0$, is not reachable trough an ordered run: a b needs to be popped out from stack 2 when stack 1 is not empty. Thus we have the following result.

Lemma 8. *There is a* MPDS *M such that any reachable configuration can be visited through (adjacent) ordered runs, and for each $k > 0$ there is a configuration that is not reachable in the* SMPDS *(k, M).*

There is a MPDS *M such that any reachable configuration can be reached also in $(2, M)$, and there is a configuration that is not reachable through (adjacent) ordered runs.*

Finally, we observe that any run ρ of the 2-stack MPDS M_3 is also 3-path-tree. In fact, given ρ, the corresponding stack tree is formed of a leftmost path where pushes of a alternate with pushes of b, and a right child for each matched push. To visit the nodes of the stack tree to recover the order in ρ it suffices to visit it in a depth-first-search fashion, thus each node is visited at most 3 times. We also observe that any 1-scoped run of a MPDS generates a stack tree such that the linear ordering can be recovered by visiting it in a depth-first-search fashion (1-scoped runs can be easily simulated by a standard pushdown automaton since a symbol is popped from the stack only in the context in which it is pushed), and thus is 3-path-tree.

Furthermore, consider the 2-stack MPDS M_5 from Fig. 5. Since along any run, each popped stack symbol was pushed within the previous context, we get that any run of M_5 is 2-scoped. However, consider the run that leads to a configuration $C_k = \langle\langle q_2, a^k, b^k\rangle\rangle$ for $k \in \mathbb{N}$. This run visits $k-1$ times the entire loop of M_5 and corresponds to the sequence of stack operations $ab(\bar{a}a^2\bar{b}b^2)^{k-1}$, where we have denoted $push(x)$ with x and $pop(x)$ with $\bar{x}$ for $x \in \{a, b\}$. The corresponding stack tree is given in Fig. 6. In order to visit its nodes according to the order given by the run, we start from the root and then visit its left child. Then, we go back to the right child of the root and proceed on the leftmost path of this subtree (i.e., $T(a)$). This way we discover the first $\bar{a}a^2$ sequence. Then, we go back to the left child of the root and visit its right child (i.e., the root of $T(b)$) thus discovering the first $pop(b)$ transition. We then proceed on the leftmost path and discover two $push(b)$ transitions, and so on. Observe that to match the sequence corresponding to an entire loop starting from q_2 in M_5, i.e., $\bar{a}a^2\bar{b}b^2$, we visit twice the root of the stack tree. Thus, to recover the entire sequence we visit the root exactly $2(k-1)+1$ times and thus configuration C_k is not reachable with a d-path-tree run for $d < 2(k-1)+1$.

Lemma 9. *There is a* MPDS *M such that any reachable configuration can be visited through 3-path-tree runs, and for each $k > 0$ there is a configuration that is not reachable in the* SMPDS *(k, M).*

There is a MPDS *M such that any reachable configuration can be reached also in $(2, M)$, and for any $k > 0$ there is a configuration that is not reachable through by a k-path-tree run of M.*

5. Solving location reachability for SMPDS

In this section, we address the location reachability problem for SMPDS. We start by defining an abstraction, called *thread interface*, that summarizes multiple context runs of a thread. Each thread interface is a tuple of pairs of control

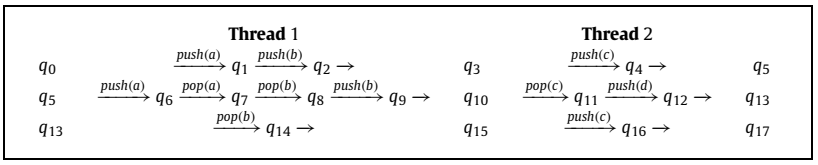

Fig. 7. A sample 2-scoped run of a MPDS with two stacks.

states representing the starting and the ending control states of the contexts that occur in the corresponding runs. Thus an interface exactly captures the interaction of a thread with the rest of the system in some runs and therefore it is a suitable abstraction for solving location reachability by first exploring the computations of the single threads. However, since the stack content is entirely abstracted away, we will need a richer abstraction to deal with the general reachability problem that will be addressed in the next section.

For the rest of this section we fix a bound $k > 0$, a k-SMPDS $M = (k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$ and $\widetilde{\Gamma} = \bigcup_{i=1}^{n} \Gamma_i$.

5.1. Thread interfaces

We start defining the notion of thread interface. Then, we show that when restricting to k-scoped runs, the whole computation of a single thread across unboundedly many contexts can be indeed captured by composing thread interfaces of size at most k. Moreover, under some conditions, the thread interfaces of a thread can be composed with the thread interfaces of the other threads to summarize entire runs of a MPDS.

A thread interface is essentially an ordered tuple of pairs denoting each the starting and the ending control states of a thread context. The contexts are listed in the order they occur in a run assuming that the first context starts with an empty stack and each of the following contexts starts with the stack content that is left by the preceding one. In other words, a thread interface stores the starting and the ending control states of the contexts of a multiple context run, and thus captures all the multiple context runs that share this "interface". Formally, we have:

Definition 3. (THREAD INTERFACE) For each $h \in [n]$, a *h-thread interface* of M is a possibly empty tuple of pairs $I = \langle in_j, out_j \rangle_{j \in [m]}$, for some $m \in \mathbb{N}$ (the *dimension* of I, also denoted $\dim(I)$), such that if $m > 0$ there exists a multiple context run $\rho_1, \ldots, \rho_m$ of T_M^h where for every $j \in [m]$, $\rho_j = \langle\langle in_j, w_j \rangle\rangle \rightsquigarrow_M^h \langle\langle out_j, w'_j \rangle\rangle$.

For the rest of this section, we use as running example a MPDS M_5 with a run as in Fig. 7. We observe that the run is 2-scoped. From the above definition, $J_1 = (\langle q_0, q_3 \rangle, \langle q_5, q_{10} \rangle, \langle q_{13}, q_{15} \rangle)$ is a 1-thread interface of M_5 of dimension 3 and $J_2 = (\langle q_3, q_5 \rangle, \langle q_{10}, q_{13} \rangle, \langle q_{15}, q_{17} \rangle)$ is a 2-thread interface of M_5 of dimension 3.

For a thread interface $I = \langle in_j, out_j \rangle_{j \in [m]}$, a *prefix* of I is any $\langle in_j, out_j \rangle_{j \in [m']}$ with $m' \leq m$ and a *suffix* of I is any $\langle in_j, out_j \rangle_{j \in [i,m]}$ with $i \geq 1$. Directly from the definition, we get that any prefix of a thread interface is also a thread interface. Clearly this is not true in general for the suffixes since some stack symbol that is popped in a context of a suffix might have been pushed in one of the contexts of the omitted prefix.

Proposition 10. *Any prefix of a h-thread interface is also a h-thread interface.*

For $i = 1, 2$, let $I_i = \langle in_j^i, out_j^i \rangle_{j \in [m_i]}$ be a h-thread interface of M, for some $h \in [n]$. We define two internal operations over thread interfaces of a given thread. With $I_1 \bowtie_1 I_2$ we denote the tuple obtained by appending I_2 to I_1. Formally, $I_1 \bowtie_1 I_2 = \langle in_j, out_j \rangle_{j \in [m_1 + m_2]}$ where $in_j = in_j^1$ and $out_j = out_j^1$ for $j \in [m_1]$, and $in_{m_1+j} = in_j^2$ and $out_{m_1+j} = out_j^2$ for $j \in [m_2]$. The other operation is a variation of $\bowtie_1$ where the last pair of I_1 is composed with the first pair of I_2. It is defined when I_1 and I_2 are both not empty. Formally, if $m_1, m_2 > 0$ and $out_{m_1}^1 = in_1^2$, then we denote with $I_1 \bowtie_2 I_2$ the tuple $\langle in_j, out_j \rangle_{j \in [m_1 + m_2 - 1]}$ where $in_j = in_j^1$ and $out_j = out_j^1$ for $j \in [m_1 - 1]$, $in_{m_1} = in_{m_1}^1$, $out_{m_1} = out_1^2$, and $in_{m_1+j} = in_{j+1}^2$ and $out_{m_1+j} = out_{j+1}^2$ for $j \in [m_2 - 1]$.

Directly from the definition of thread interface we get that both compositions define thread interfaces.

Lemma 11. *Let $I_i = \langle in_j^i, out_j^i \rangle_{j \in [m_i]}$ be a h-thread interface of M, for some $h \in [n]$ and $i = 1, 2$.*

- *$I_1 \bowtie_1 I_2$ is a h-thread interface of dimension $m_1 + m_2$.*
- *If $out_{r_1}^1 = in_1^2$, then $I_1 \bowtie_2 I_2$ is a h-thread interface of dimension $m_1 + m_2 - 1$.*

As observed in Section 3, a run decomposition naturally defines multiple context runs and thus also thread interfaces. Given a k-scoped run ρ of a MPDS M with n stacks and let $\rho_1, \ldots, \rho_\ell$ be a decomposition of ρ into contexts, for $h \in [n]$ the

h-thread interface $I = \langle in_j, out_j\rangle_{j\in[m]}$ *defined by* $\rho_1, \ldots, \rho_\ell$ is such that there are exactly m h-contexts in the decomposition and the j-th h-context in the decomposition starts at in_j and ends at out_j, for $j \in [m]$.

A *canonical* thread interface for ρ is a thread interface defined by a decomposition into maximal contexts. Interestingly, it is possible to capture each canonical thread interface as the composition by $\bowtie_1$ and $\bowtie_2$ of thread interfaces of bounded dimension. In fact, for the k-scoped restriction, in each run a push onto stack h is either matched within the next k maximal h-contexts (including the current one) or never. As an example, consider again the 2-scoped run from Fig. 7. Note that J_1 and J_2 are canonical thread interfaces for it, and $J_1 = (\langle q_0, q_3\rangle, \langle q_5, q_8\rangle) \bowtie_2 (\langle q_8, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$ and $J_2 = (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle) \bowtie_1 (\langle q_{15}, q_{17}\rangle)$ (the interfaces used in the compositions are all of dimension at most 2).

The above property is formally stated in the following lemma.

Lemma 12. *Let $k \in \mathbb{N}$, M be a* MPDS *with n stacks, ρ be a k-scoped run of M, and for $h \in [n]$, I be a canonical h-thread interface for ρ.*

There exist h-thread interfaces $I_0, \ldots, I_s$ of dimension at most k such that $I = I_0 \bowtie_{j_1} I_1 \ldots \bowtie_{j_s} I_s$ with $j_1, \ldots j_s \in [2]$.

Proof. Let $I = \langle in_j, out_j\rangle_{j\in[m]}$ be a canonical h-thread interface for a run ρ. Thus, there exists a decomposition of ρ with exactly m maximal h-contexts, say $\rho_1, \ldots, \rho_m$, such that ρ_j starts at in_j and ends at out_j, for $j \in [m]$.

We claim that if $m > k$, there are two h-thread interfaces I_1 and I_2 both of dimension less than m such that either $I = I_1 \bowtie_1 I_2$ or $I = I_1 \bowtie_2 I_2$. We consider two cases.

The first case is when there are no push transitions in ρ_1 that are matched by a pop transition outside ρ_1. Since no stack content pushed within ρ_1 is used later, we get that $I_2 = \langle in_j, out_j\rangle_{j\in[2,m]}$ is a h-thread interface. From Proposition 10, $I_1 = \langle in_1, out_1\rangle$ is also a h-thread interface. Since both have dimension less than m and $I = I_1 \bowtie_1 I_2$, we are done with this case.

In the other case, i.e., when there is a push transition that is matched outside ρ_1, we take the first such push transition in ρ_1. Denote with t its matching pop transition. Since ρ is k-scoped, t must occur in some ρ_i with $i \le k$. Also, according to the stack behavior, if a portion $\rho[j, j']$ of ρ starts with a push transition of stack h and ends with a pop transition of the same stack, then any other transition involving stack h within $\rho[j, j']$ must be matched and the matching transition must also occur within $\rho[j, j']$. Thus, let ρ_i' and ρ_i'' be the two contexts obtained by splitting ρ_i at the control state q reached after taking transition t. From Proposition 10, we get that $I_1 = (\langle in_1, out_1\rangle, \ldots, \langle in_{i-1}, out_{i-1}\rangle, \langle in_i, q\rangle$ is a h-thread interface. Moreover, since no stack content that is pushed in $\rho_1, \ldots, \rho_{i-1}, \rho_i'$ is popped in $\rho_i'', \rho_{i+1}, \ldots, \rho_m$, $I_2 = (\langle q, out_i\rangle, \langle in_{i+1}, out_{i+1}\rangle, \ldots, \langle in_m, out_m\rangle)$ is also a h-thread interface. Also, from $i \in [2, k]$ and $k < m$, we have that $i \in [2, m-1]$ must hold. Thus, we get that both interfaces have dimension less than m, thus $I = I_1 \bowtie_2 I_2$ holds and the claim is proved also in this case.

By recursively applying the above claim to the resulting thread interfaces until we get only thread interfaces of dimension at most k, we get the lemma. □

Note that the above lemma does not hold if I is an arbitrary (non-canonical) thread interface. In fact, in a non-canonical thread interface we can have several consecutive pairs that correspond to portions of a same maximal context. Thus, in order to capture such thread interfaces the bound k may not suffice.

Thread interfaces can be composed to summarize entire runs of a given MPDS. We recall that each pair of a thread interface essentially gives the control state update that is effected by a corresponding context of the underlying multiple context run. Thus starting from an initial state, we can simulate a run by iteratively selecting pairs of the thread interfaces to update the current control state, and the run can be constructed by *stitching* together the contexts corresponding to the selected pairs. Therefore, a set of thread interfaces summarizes a set of runs of a MPDS if we can order all their pairs such that: (1) each pair can be stitched to the following one, i.e., the ending control state of a pair matches the starting control state of the following one, and (2) the pairs from a same thread interface occur in the same order as within the thread interface. This condition is formally captured by the following definition.

For $h \in [n]$, let $I_h = \langle in_j^h, out_j^h\rangle_{j\in[m_h]}$ be a h-thread interface of M and denote $\mathcal{D} = \bigcup_{h\in[n]}\{\langle h, j\rangle \mid j \in [m_h]\}$. For $\ell, \ell' \in [n]$, we say that $I_1, \ldots, I_n$ can be *stitched* from in_1^ℓ through $out_{m_{\ell'}}^{\ell'}$ if there exists a 1-to-1 mapping $next : \mathcal{D} \setminus \{\langle \ell', m_{\ell'}\rangle\} \to \mathcal{D} \setminus \{\langle \ell, 1\rangle\}$ such that for each $\langle h, j\rangle \in \mathcal{D} \setminus \{\langle \ell', m_{\ell'}\rangle\}$:

1. $out_j^h = in_{j'}^{h'}$ where $\langle h', j'\rangle = next(h, j)$ (i.e., *next* returns a pair that can be stitched to the given pair);
2. denoting $next^1(x) = next(x)$ and $next^i(x) = next(next^{i-1}(x))$ for $i > 1$, if $next^i(h, j) = \langle h, j'\rangle$ for some $i \in \mathbb{N}$ then $j < j'$ (i.e., the ordering induced by *next* is consistent with the local ordering within each thread interface).

Note that since *next* is a 1-to-1 mapping, it defines a linear ordering among all the pairs of $I_1, \ldots, I_n$.

Fig. 8 illustrates the stitching of linear interfaces J_1 and J_2 in our running example. In the figure, map *next* is denoted with the dashed edges. Precisely, *next* is defined as $next(1, j) = \langle 2, j\rangle$ for $j \in [3]$ and $next(2, j) = \langle 1, j+1\rangle$ for $j \in [2]$. The sequence of pairs defined by *next* is $\langle q_0, q_3\rangle, \langle q_3, q_5\rangle, \langle q_5, q_{10}\rangle, \langle q_{10}, q_{13}\rangle, \langle q_{13}, q_{15}\rangle, \langle q_{15}, q_{17}\rangle$, which corresponds to the perfect interleaving of the two thread interfaces J_1 and J_2. Thus, according to the above definition, J_1 and J_2 can be stitched from q_0 through q_{17}.

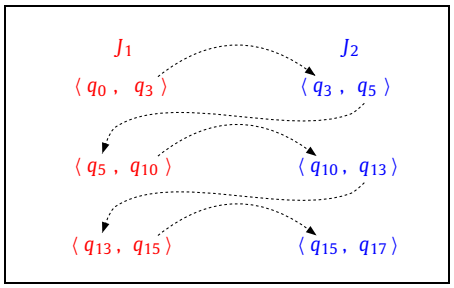

Fig. 8. Stitching of linear interfaces.

As already observed we can show that runs of MPDS can be fully characterized by tuples of thread interfaces.

Theorem 13. *Let M be a* MPDS *with n stacks, and q be a control state of M. Then, there is a run of M that reaches q iff there are $I_1, \ldots, I_n$ such that:*

1. *for $h \in [n]$, I_h is a (canonical) h-thread interface of M, and*
2. *$I_1, \ldots, I_n$ can be stitched from an initial state of M through q.*

Proof. The forward direction follows directly from the definitions. In fact, consider a run ρ from an initial state q_0 to q. Fix a decomposition of ρ into contexts and take the corresponding h-thread interfaces I_h for each $h \in [n]$. Define *next* to match the ordering of pairs according to the fixed decomposition. Clearly, each pair in the sequence can be stitched to the following one, and *next* is a 1-to-1 mapping and is consistent with the pair ordering within each thread interface I_h. Therefore, $I_1, \ldots, I_n$ can be stitched from q_0 through q.

For the converse direction, let $I_h = \langle in_j^h, out_j^h \rangle_{j \in [m_h]}$ be a h-thread interface of M for $h \in [n]$ and suppose that $I_1, \ldots, I_n$ can be stitched from an initial state q_0 of M through q. Thus, there are $\ell, \ell' \in [n]$ such that $in_1^\ell = q_0$, $out_{m_{\ell'}}^{\ell'} = q$ and denoting $\mathcal{D} = \bigcup_{h \in [n]} \{\langle h, j \rangle \mid j \in [m_h]\}$, there is a 1-to-1 mapping $next : \mathcal{D} \setminus \{\langle \ell', m_{\ell'} \rangle\} \to \mathcal{D} \setminus \{\langle \ell, 1 \rangle\}$ such that (1) $out_j^h = in_{j'}^{h'}$ where $\langle h', j' \rangle = next(h, j)$ and (2) if $next^i(h, j) = \langle h, j' \rangle$ for some $i \in \mathbb{N}$ then $j < j'$. For $h \in [n]$ and $j \in [m_h]$, from Definition 3, we can take contexts $\langle\langle in_j^h, u_j^h \rangle\rangle \rightsquigarrow_M^h \langle\langle out_j^h, v_j^h \rangle\rangle$ such that $u_1^h = \varepsilon$ and for $j < m_h$, $u_{j+1}^h = v_j^h$.

Denote $m = \sum_{h \in [n]} m_h$. We construct inductively on $j \in [m]$ a run of M by stitching together the above contexts in the order given by *next*.

We start with the context corresponding to the first pair of I_ℓ, i.e., $\langle\langle in_1^\ell, \varepsilon \rangle\rangle \rightsquigarrow_M^\ell \langle\langle out_1^\ell, v_1^\ell \rangle\rangle$. This clearly gives the run $\rho_1 = \langle\langle q_0, \varepsilon, \ldots, \varepsilon \rangle\rangle \rightsquigarrow_M^{h_1} \langle\langle q_1, w_1^1, \ldots, w_1^n \rangle\rangle$ where $h_1 = \ell$, $q_1 = out_1^\ell$, $w_1^\ell = v_1^\ell$ and $w_j^h = \varepsilon$ for $h \neq \ell$ (recall that $q_0 = in_1^\ell$).

For the inductive step, denote $\rho_j = \langle\langle q_0, \varepsilon, \ldots, \varepsilon \rangle\rangle \rightsquigarrow_M^{h_1} \ldots \rightsquigarrow_M^{h_j} \langle\langle q_j, w_j^1, \ldots, w_j^n \rangle\rangle$ with $j > 0$ and suppose that the last context of ρ_j corresponds to the i_j-th pair of I_{h_j}. Thus, $q_j = out_{i_j}^{h_j}$ holds.

Now, let $\langle h_{j+1}, i_{j+1} \rangle = next(h_j, i_j)$. From property (1) of mapping *next*, we get that $out_{i_j}^{h_j} = in_{i_{j+1}}^{h_{j+1}}$ and thus $q_j = in_{i_{j+1}}^{h_{j+1}}$. Hence, in order to execute the context $\langle\langle in_{i_{j+1}}^{h_{j+1}}, u_{i_{j+1}}^{h_{j+1}} \rangle\rangle \rightsquigarrow_M^{h_{j+1}} \langle\langle out_{i_{j+1}}^{h_{j+1}}, v_{i_{j+1}}^{h_{j+1}} \rangle\rangle$ from configuration $\langle\langle q_j, w_j^1, \ldots, w_j^n \rangle\rangle$ we just need to show that $w_j^{h_{j+1}} = u_{i_{j+1}}^{h_{j+1}}$ holds.

For this we observe that from property (2) of *next*, in ρ_j the ordering of the contexts of a thread conforms to the ordering of the corresponding pairs in the thread interface. In particular, if $i_{j+1} = 1$, $\langle\langle in_{i_{j+1}}^{h_{j+1}}, u_{i_{j+1}}^{h_{j+1}} \rangle\rangle \rightsquigarrow_M^{h_{j+1}} \langle\langle out_{i_{j+1}}^{h_{j+1}}, v_{i_{j+1}}^{h_{j+1}} \rangle\rangle$ is the first context of thread $T_M^{h_{j+1}}$ in ρ_j, otherwise, the previous context of $T_M^{h_{j+1}}$ corresponds to the $i_{j+1} - 1$-th pair of $I_{h_{j+1}}$. Clearly, $w_j^{h_{j+1}} = \varepsilon$ in the first case and $w_j^{h_{j+1}} = v_{i_{j+1}-1}^{h_{j+1}}$ in the second case. Since by definition $u_1^{h_{j+1}} = \varepsilon$ and $u_{i_{j+1}}^{h_{j+1}} = v_{i_{j+1}-1}^{h_{j+1}}$, we have that $w_j^{h_{j+1}} = u_{i_{j+1}}^{h_{j+1}}$ must hold.

To conclude the proof we observe that *next* is a 1-to-1 mapping that is not defined on $\langle \ell', m_{\ell'} \rangle$. Thus, the last context of ρ_m must correspond to this pair, and therefore $q_m = out_{m_{\ell'}}^{\ell'} = q$, that concludes the proof. □

5.2. A fixed-point algorithm for location reachability

In this section, we present a fixed-point algorithm to solve the location reachability problem for k-SMPDS that is based on the computation of canonical thread interfaces.

The algorithm. One way to solve this problem is to compute nondeterministically n thread-interfaces, one for each thread, and then by Theorem 13 check whether they form an M computation reaching a target control state. Unfortunately, this would

Algorithm 1.

Initialization

$\mathcal{I} = \{(q, \tau_\emptyset) \mid q \in Q_I\}$

Simulation

Let $(q, \tau) \in \mathcal{I}$ and q' such that $first(\tau(T^i_M)) = \langle q, q' \rangle$, for some $i \in [n]$.
Add to $\mathcal{I}$ the pair (q', τ') where

$$\tau'(t) = \begin{cases} tail(\tau(T^i_M)) & \text{if } t = T^i_M \\ \tau(t) & \text{otherwise.} \end{cases}$$

Thread-interface progression

Let $(q, \tau) \in \mathcal{I}$ and $\tau(T^i_M)$ be an empty thread interface for some $i \in [n]$.
Add to $\mathcal{I}$ any pair (q, τ') where

$$\tau'(t) = \begin{cases} I & \text{if } t = T^i_M \\ \tau(t) & \text{otherwise} \end{cases}$$

and I is a thread interface of T^i_M such that $dim(I) \leq k$.

Fig. 9. Rules of Algorithm 1 (fixed-point algorithm solving the location reachability problem for SMPDS).

only yield a semi-algorithm as we do not know, a priori, the number of contexts that are needed for each thread in order to conclude that the target is not reachable. However, according to Lemma 12 such thread interfaces can be computed by portions of bounded dimension. Moreover, the conditions of Theorem 13 can be checked by such portions. In particular, the fixed-point algorithm we propose will take for each thread a thread interface of bounded dimension, then it will advance in the simulation of a run of the MPDS by stitching the contexts of these interfaces until one of them will be entirely consumed (this corresponds to advancing in the run by macro-steps covering each an entire context). At this point, a new thread interface of bounded dimension is taken for the corresponding thread and the simulation is resumed. The iterations halt as soon as a target control state is reached or no new simulations are possible. Since the number of thread interfaces is finite for a given bound, the algorithm always terminates.

In the following we will refer to this algorithm as Algorithm 1. In greater detail, Algorithm 1 computes pairs of the form (q, τ) where q is a control state (the state reached after the last simulated context) and τ is a map that assigns a (possibly empty) thread-interface suffix to each thread. At each step of the algorithm, we either consume the first pair of a thread-interface suffix (*simulation rule*) or append a thread interface (*thread-interface progression rule*).

The simulation rule can be applied to any pair of the form (q, τ) such that there is a thread-interface suffix mapped by τ whose first pair starts with q. For a thread-interface suffix $I = \langle in_j, out_j \rangle_{j \in [i,m]}$, we denote with $first(I)$ the first pair of I, i.e., $\langle in_i, out_i \rangle$, and with $tail(I)$ the rest of I, i.e., $\langle in_j, out_j \rangle_{j \in [i+1,m]}$. An application of the simulation rule to (q, τ) with $\tau(t) = \langle in_j, out_j \rangle_{j \in [m]}$ for a thread t and $q = in_1$ would yield (out_1, τ') where $\tau'(t) = tail(\tau(t))$ and $\tau'(t') = \tau(t')$ for $t \neq t'$.

The thread-interface progression rule is very simple: from (q, τ) such that $\tau(t)$ is empty for some thread t, we can add (q, τ') such that $\tau'(t)$ is any thread interface of t of dimension at most k and $\tau'(t') = \tau(t')$ for $t' \neq t$.

We denote with $\mathcal{I}$ the set of pairs computed by the algorithm and with $\tau_\emptyset$ the map that assigns each thread with an empty thread-interface. The set $\mathcal{I}$ is initialized to all pairs of the form $(q, \tau_\emptyset)$ where q is an initial control state.

The detailed rules of Algorithm 1 are given in Fig. 9. The thread interfaces of dimension at most k can be computed in a standard way, see for example [12], and thus we omit it. The algorithm halts as soon as a control state in the target set is reached or no more tuples can be added to the set $\mathcal{I}$. In the first case, it outputs YES, otherwise it outputs NO. Termination of Algorithm 1 is guaranteed since there are finitely many control states and thread interfaces (and thus thread-interface suffixes).

Correctness of the algorithm. To show correctness, we will argue that the fixed-point algorithm described above discovers the existence of a run ρ that leads to a target control state by computing the thread interfaces (one for each thread) for a maximal decomposition of ρ.

We start by illustrating this with an example. Consider again the run of Fig. 7 along with the corresponding canonical thread interfaces $J_1 = (\langle q_0, q_3 \rangle, \langle q_5, q_8 \rangle) \bowtie_2 (\langle q_8, q_{10} \rangle, \langle q_{13}, q_{15} \rangle)$ and $J_2 = (\langle q_3, q_5 \rangle, \langle q_{10}, q_{13} \rangle) \bowtie_1 (\langle q_{15}, q_{17} \rangle)$. Fig. 10 gives a sequence of steps of our algorithm that mimics this run using the above decomposition of J_1 and J_2 via $\bowtie_1$ and $\bowtie_2$. Starting from $(q_0, \tau_\emptyset)$ through two applications of the thread-interface progression rule (denoted with $\overset{P}{\longmapsto}$ in the figure), we add the pair (q_0, τ_1) where $\tau_1(T^1_M) = (\langle q_0, q_3 \rangle, \langle q_5, q_8 \rangle)$ and $\tau_1(T^2_M) = (\langle q_3, q_5 \rangle, \langle q_{10}, q_{13} \rangle)$ (τ_1 maps each thread to the first thread interface in the considered decomposition of respectively J_1 and J_2). Then, through two applications of the simulation rule (denoted with $\overset{S}{\longmapsto}$ in the figure), we add (q_5, τ_2) where $\tau_2(T^1_M) = (\langle q_5, q_8 \rangle)$ and $\tau_2(T^2_M) = (\langle q_{10}, q_{13} \rangle)$. Now, from

$$\left(q_0, \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix}\right) \overset{P}{\longmapsto} \left(q_0, \begin{bmatrix} (\langle q_0, q_3\rangle, \langle q_5, q_8\rangle) \\ \emptyset \end{bmatrix}\right) \overset{P}{\longmapsto} \left(q_0, \begin{bmatrix} (\langle q_0, q_3\rangle, \langle q_5, q_8\rangle) \\ (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle) \end{bmatrix}\right) \overset{S}{\longmapsto}$$

$$\left(q_3, \begin{bmatrix} (\langle q_5, q_8\rangle) \\ (\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle) \end{bmatrix}\right) \overset{S}{\longmapsto} \left(q_5, \begin{bmatrix} (\langle q_5, q_8\rangle) \\ (\langle q_{10}, q_{13}\rangle) \end{bmatrix}\right) \overset{S}{\longmapsto} \left(q_8, \begin{bmatrix} \emptyset \\ (\langle q_{10}, q_{13}\rangle) \end{bmatrix}\right) \overset{P}{\longmapsto}$$

$$\left(q_8, \begin{bmatrix} (\langle q_8, q_{10}\rangle, \langle q_{13}, q_{15}\rangle) \\ (\langle q_{10}, q_{13}\rangle) \end{bmatrix}\right) \overset{S}{\longmapsto} \left(q_{10}, \begin{bmatrix} (\langle q_{13}, q_{15}\rangle) \\ (\langle q_{10}, q_{13}\rangle) \end{bmatrix}\right) \overset{S}{\longmapsto}$$

$$\left(q_{13}, \begin{bmatrix} (\langle q_{13}, q_{15}\rangle) \\ \emptyset \end{bmatrix}\right) \overset{S}{\longmapsto} \left(q_{15}, \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix}\right) \overset{P}{\longmapsto} \left(q_{15}, \begin{bmatrix} \emptyset \\ (\langle q_{15}, q_{17}\rangle) \end{bmatrix}\right) \overset{S}{\longmapsto} \left(q_{17}, \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix}\right)$$

Fig. 10. An example of application of the rules of Algorithm 1.

(q_5, τ_2) first $\langle q_5, q_8\rangle$ is consumed, then $(\langle q_8, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$ is added for thread T_M^1, and finally $\langle q_8, q_{10}\rangle$ is consumed. This corresponds to simulate the maximal context of T_M^1 from q_5 through q_{10}. Also, note that the two consecutive applications of the simulation rule concern the same thread. This captures the semantics of composition $\bowtie_2$, and thus for thread T_M^1 the sequence of rule applications from Fig. 10 computes J_1 exactly as $(\langle q_0, q_3\rangle, \langle q_5, q_8\rangle) \bowtie_2 (\langle q_8, q_{10}\rangle, \langle q_{13}, q_{15}\rangle)$. For thread T_M^2, the sequence from Fig. 10 computes J_2 as $(\langle q_3, q_5\rangle, \langle q_{10}, q_{13}\rangle) \bowtie_1 (\langle q_{15}, q_{17}\rangle)$ (the simulation of the second thread interface starts after that a context of the other thread has been simulated).

Theorem 14. *Let M be an* MPDS *with n stacks, q be an M control state, $\mathcal{I}$ be the set computed by Algorithm 1 and $k \in \mathbb{N}$. Then, q is reachable in a k-scoped run of M iff $(q, \tau) \in \mathcal{I}$ for some map τ.*

Proof. We start proving the forward direction.

Let ρ be a k-scoped run of M ending at a configuration with control state q. We show that $(q, \tau_\emptyset)$ is added to $\mathcal{I}$ by our algorithm.

First, recall that from Theorem 13, there must be $I_1, \ldots, I_n$ and a 1-to-1 mapping *next* such that: (1) for $h \in [n]$, I_h is a canonical h-thread interface of M, and (2) $I_1, \ldots, I_n$ can be stitched from an initial state of M through q in the ordering given by *next*.

Starting from $(q_0, \tau_\emptyset)$, a sequence of applications of the rules of our algorithm that leads to add $(q, \tau_\emptyset)$ to $\mathcal{I}$ can be obtained by consuming the pairs of $I_1, \ldots, I_n$ in the ordering given by *next*. In fact, from Lemma 12, for $h \in [n]$, we get $I_h = I_{h,0} \bowtie_{j_{h,1}} \ldots \bowtie_{j_{h,s_h}} I_{h,s_h}$ where $I_{h,0}, \ldots, I_{h,s_h}$ are h-thread interfaces of dimension at most k and $j_{h,1}, \ldots, j_{h,s_h} \in [2]$. Thus, each thread interface I_h is explored by adding via the thread-interface progression rule its bounded portions: we first add $I_{h,0}$, then we add $I_{h,1}$ when $I_{h,0}$ is entirely consumed, and so on. The run is simulated by consuming the pairs in the added interfaces. Note that for $h \in [n]$, each pair $\langle q, q'\rangle$ of I_h is either (1) a pair also of some $I_{h,i}$, or (2) is split into a pair $\langle q, q''\rangle$ at the end of some $I_{h,i}$ and a pair $\langle q'', q'\rangle$ at the beginning of $I_{h,i+1}$, and $j_{h,i+1} = 2$ (i.e., $I_{h,i}$ and $I_{h,i+1}$ are composed through $\bowtie_2$). Thus to consume the pairs from this second case, we need two applications of the simulation rule interleaved with an application of the thread-interface progression rule. For the other pairs one application of the simulation rule suffices. Recall also that the simulation rule updates the control state, with the ending state of the consumed pair. Thus, once all the pairs of $I_1, \ldots, I_n$ will be consumed, the control state is updated to q and therefore the pair $(q, \tau_\emptyset)$ is added to $\mathcal{I}$.

For the other direction, we show a stronger property, that is, if π is the sequence of applications of the algorithm rules that leads to add (q, τ) to $\mathcal{I}$ then there is a k-scoped run ρ of M that reaches q and can be decomposed into contexts $\rho_1, \ldots, \rho_m$ such that: denoting in_i and out_i respectively the starting and ending control states of ρ_i for $i \in [m]$, $(in_1, out_1), \ldots, (in_m, out_m)$ is the ordered sequence of control state pairs consumed in the applications of the simulation rule in π. We show this property by induction on the length of π.

The base case is trivial. For the initialization rule, $(q, \tau_\emptyset) \in \mathcal{I}$ for each initial control state q of M, and clearly q is reachable in M (within zero steps).

Suppose now by induction that the above statement holds for each pair that is added to $\mathcal{I}$ with at most $d > 0$ rule applications. Consider a sequence π of $d+1$ rule applications that adds (q, τ) to $\mathcal{I}$ from a pair (q', τ'). The interesting case is when (q, τ) is added from (q', τ') by applying the simulation rule. In fact, if (q, τ) is added from (q', τ') by applying the thread-interface progression rule, we get that $q = q'$ holds and thus the property holds directly from the induction hypothesis.

If (q, τ) is added by applying the simulation rule, from the definition, there must be a pair $\langle q', q\rangle$ that starts one of the thread-interface suffixes mapped by τ'. Denote with I such suffix and with T the corresponding thread. Moreover, let $J = I' \bowtie_1 I$ be the thread interface that was added in the last thread progression rule applied to thread T along π.

If I' is empty, i.e., I is exactly the thread interface added by the thread progression, then by definition there is a multiple context run ρ^I corresponding to I. Thus, we apply the induction hypothesis to (q', τ') and denote ρ' the corresponding

k-scoped run of M. Define ρ as the run obtained by appending the first context of ρ^I to ρ'. Since no previously pushed symbol is popped in the added context, ρ is still k-scoped. Moreover, we append to ρ' a context that matches the pair of control states used in the simulation rule, thus by applying the induction hypothesis we get that ρ matches the sequence of control state pairs used in the application of simulation rules in π, and we are done with this case. For the remaining case, i.e., when I' is not empty, we reason analogously, except for arguing that ρ is k-scoped. In fact, the context we add is not the first one in a multiple context run, and thus can pop stack symbols introduced in the previous contexts of such run. However, since we bound the size of the thread interfaces to k, the popped symbols were certainly pushed in the last k contexts of thread T and thus ρ is k-scoped, that concludes the proof. □

As for the computational complexity of our fixed-point algorithm, we observe the following. Computing a thread interface takes time polynomial in $|Q|$ (reachability of single-stack pushdown systems) and the number of different tuples of the form $\langle p_j, q_j \rangle_{j \in [m]}$ where $p_j, q_j \in Q$ and $m \leq k$ is $O(|Q|^{2k})$. Thus the total number of different pairs (q, τ) that can be added to $\mathcal{I}$ is $O(|Q|^{2kn+1})$. Further, the initialization takes constant time, and from each tuple at most n simulation steps (one for each thread interface suffix) and $O(|Q|^{2kd})$ thread-interface progression steps can be taken where d is the number of components of the tuple containing an empty thread interface (we can select a new thread interface for each one of the empty components). Moreover, the number of different tuples that have an empty thread interface in the same d components is $O(|Q|^{2k(n-d)+1})$, thus the overall number of different thread-interface progression steps from all such tuples is $O(|Q|^{2kn+1})$. Therefore, the total number of different thread progression steps that can be taken is $O(2^n |Q|^{2kn+1})$, and thus our fixed point algorithm can be implemented to take $O(2^n |Q|^{2kn+1})$ time.

We further observe that we can implement our algorithm to use only polynomial space in the number of threads and the bound k. In fact, at each iteration we can store only the current pair not all the set $\mathcal{I}$ and select nondeterministically the next rule to apply (in this case thread interfaces are picked by first guessing a tuple and then checking that it is indeed a thread interface). The algorithm will halt as soon as we compute a pair with control state in the target set or we have reached a number of iterations that equals the maximum number of pairs that can be added to $\mathcal{I}$. This shows that the location reachability problem for SMPDS is in PSPACE.

This upper bound is also tight for both parameters. In fact, by fairly standard constructions we can reduce the membership problem for a Turing machine working in polynomial space to both the location reachability problem for an n-stack 2-SMPDS and a 2-stack $2k$-SMPDS. In the first case, we use the stacks as registers and maintain the configuration of the Turing machine one cell for each stack. A cell update will require first to read the content of the corresponding stack by popping it, and then to push the new content onto it. Cells are updated in a round-robin fashion simulating a scan of the tape from left to right. Control states are used guide the round robin and store the content of the neighbor cells: the content of the left cell is read, while that of the right cell is nondeterministically guessed and then checked when reading it. Since each symbol that is pushed is popped in the next context of the same stack, the described SMPDS is 2-scoped. In the second case, the tape content is maintained into a stack and its updated by moving it into the other stack. Thus each pop from one stack is followed by a push onto the other stack, and thus the maximum number of maximal contexts between a push and a matching pop of the same stack is $2k$.

We get the following theorem (where k is assumed to be encoded in unary).

Theorem 15. *The location reachability problem for k-SMPDS is PSPACE-complete, and hardness can be shown both with respect to the number of stacks and the bound k.*

6. Solving the configuration reachability problem for SMPDS

In this section, we address the reachability problem for SMPDS for an arbitrary set of target configurations given as the cross product of regular languages of stack contents (one for each stack). We will show that reachability in this case is still PSPACE-complete as for the location reachability problem addressed in the previous section.

Since we need to account also for the stack contents, we introduce a new abstraction called *layered stack automaton*. For a thread T, an ℓ-layered stack automaton captures the top portion of its stack which corresponds to the symbols that were pushed within the last ℓ contexts of T. The automaton is structured into layers that are added incrementally by applying for each layer a saturation procedure similar to the one given in [5] for standard (one-stack) pushdown systems. We recall that iterating such a saturation procedure a bounded number of times is quite straightforward and was already used in [1] to give a decision algorithm for the reachability problem up to k context-switches. However, in our case, we need to account for unboundedly many context-switches and the notion of layered stack automaton by itself does not suffice (we would need to use layered stack automata with unbounded number of layers).

Since we restrict to k-scoped computations, only the symbols that were pushed within the last k contexts are used in the pop transitions. Thus, we can use ℓ-layered stack automata with $\ell \leq k$ to keep track of the meaningful top portion of the stack during a computation. We hence relate the layered automata via a *successor* relation capturing that a layered automaton A is a successor of a layered automaton B if A is obtained by adding a new layer to B via the saturation procedure.

The bounded layered automata connected via the successor relation form a *thread automaton*, that is, a finite automaton that accepts for a thread all the stack contents that can occur in a configuration that is reachable along a k-scoped run. In

a thread automaton, the states keep track of the contexts of a thread and, via the layered automaton, of the portion of the stack that was pushed within it. The automaton explores the contexts backwards and at each context that yields a portion w of the stack that is not popped out in the rest of the run, it simulates the top layer of the current layered automaton by reading w. We observe that a run of a thread automaton also captures a thread interface by listing its pairs in the reverse order.

To solve the reachability problem for k-SMPDS, we thus construct a finite automaton $\mathcal{R}$ that for each tuple of the form $(w_1, \ldots, w_n)$ such that $\langle\langle q, w_1, \ldots, w_n\rangle\rangle$ is a configuration that is reachable within a k-scoped run, it accepts at least an interleaving of $w_1, \ldots, w_n$. This automaton uses as components a thread automaton for each thread and synchronizes all of them by picking the next context (among the possible next ones) such that it can be stitched to the last processed one. Thus, assuming that the stack contents of the target set are expressed by finite automata, we can modify $\mathcal{R}$ to simulate such automata in parallel with the thread automata by a standard cross product, that reduces the configuration reachability to standard reachability for finite automata.

For the rest of this section we fix a bound $k > 0$, a k-SMPDS $M = (k, Q, Q_I, \widetilde{\Gamma}_n, \delta)$, and $\Gamma = \bigcup_{i=1}^{n} \Gamma_i$.

6.1. Layered stack automata

For $\ell \geq 0$, an ℓ-layered stack automaton A of M is essentially a finite automaton structured into $(\ell + 1)$ layers whose set of states contains ℓ copies of each $q \in Q$ (one for each layer) 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 stack h. Transitions are only between states of the same layer or from a layer to a lower layer, i.e., they are of the form (s, γ, s') where $\gamma \in \Gamma_h \cup \{\varepsilon\}$ and the layer of s' is not larger than the layer of s (note that layered automata may have ε-transitions). Moreover, there are no transitions leaving from q_F and every state is either isolated or connected to q_F by a run. Formally, we have:

Definition 4. (LAYERED STACK AUTOMATON) Given $\ell \geq 0$, an *ℓ-layered stack 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:

1. $S = \bigcup_{i=0}^{\ell} S_i$ is the set of states where $S_0 = \{q_F\}$ and $S_i = \{\langle q, i\rangle \mid q \in Q\}$, for $i \in [\ell]$;
2. $\Delta \subseteq S \times (\Gamma_h \cup \{\varepsilon\}) \times S$ is the transition relation such that if $(s, \gamma, s') \in \Delta$, for $s \in S_i$, $s' \in S_j$, then $i > 0$ and $i \geq j$;
3. 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);
4. there is at least a state $s \in S_\ell$ that is not isolated.

For $i \in [0, \ell]$, S_i denotes the *layer i* of A. S_ℓ is called the *top layer* and ℓ is referred to as the *top-layer index*. For states s_1, s_2, the language accepted by A from s_1 to s_2 is denoted $\mathcal{L}(A, s_1, s_2)$. Moreover, if $s_1 = \langle q, \ell\rangle$ and $s_2 = q_F$, we also denote $\mathcal{L}(A, s_1, s_2)$ simply as $\mathcal{L}(A, q)$ and say that a configuration $\langle\langle q, w\rangle\rangle$ is *accepted* by A if $w \in \mathcal{L}(A, q)$. □

Note that two ℓ-layered stack automata over the same alphabet Γ_h may differ only on the set of transitions and the only layered stack automaton over the alphabet Γ_h of top-layer index 0 has only the state q_F and no transitions. In the following, we denote with A_h^ϵ the layered stack automaton of top-layer index 0 with input alphabet Γ_h. Moreover, we often refer to a state $\langle q, i\rangle$ of a layered stack automaton as the copy of q in layer i.

6.1.1. Saturation procedure

Let A be an ℓ-layered stack automaton A with alphabet Γ_h.

With $Sat(A)$ we denote the layered stack automaton A' over Γ_h, obtained by applying to A the saturation procedure from [5] with respect to the internal transitions and the push and pop transitions involving stack h. The procedure is applied such that the new transitions that are added are all leaving from the top-layer states and only states in the top k layers are involved (according to the k-scoped limitation). Namely, let $\ell > 0$ be the top layer index (if $\ell = 0$, *Sat* 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 $(q, q') \in \delta$: $(\langle q', \ell\rangle, \varepsilon, \langle q, \ell\rangle)$ is added to set of transitions provided that $\langle q, \ell\rangle$ is connected to q_F;
- for a push transition $(q, q', \gamma) \in \delta$: $(\langle q', \ell\rangle, \gamma, \langle q, \ell\rangle)$ is added to set of transitions provided that $\langle q, \ell\rangle$ is connected to q_F;
- for a pop transition $(q, \gamma, q') \in \delta$: $(\langle q', \ell\rangle, \varepsilon, \langle q'', \ell'\rangle)$, with $\ell - k < \ell' \leq \ell$, is added to the set of transitions provided that there is a path of A' labeled γ from $\langle q, \ell\rangle$ to a state $\langle q'', \ell'\rangle$ (note that such a path may contain an arbitrary number of ε-edges; also, $\langle q'', \ell'\rangle$ is not isolated, and thus, connected to q_F by definition).

Note that all the transitions that are added in the above saturation procedure either stay within the top layer or are from a top-layer state to a lower layer state. In particular, the transitions that cross layers are added only through a pop transition and thus are ε-transitions. Moreover, the ending state of an added transition is connected to q_F through a path. Thus, we get:

Proposition 16. *if A is an ℓ-layered stack automaton then $Sat(A)$ is also an ℓ-layered stack automaton.*

We further remark that since the saturation procedure adds only transitions to states of the form $\langle p, \ell' \rangle$ for $p \in Q$ and $\ell' \in [\ell - k + 1, \ell]$, no direct transition to q_F is added even if $\ell < k$.

In the following, we will be interested in layered automata whose layers match the contexts of k-scoped multiple context runs. We say that an ℓ-layered automaton A *matches* a k-scoped multiple context run $\rho_1, \ldots, \rho_\ell$ of T^h_M if: for every configuration $\langle\langle q, y \rangle\rangle$ of ρ_ℓ such that $y \in \mathcal{L}(A, q)$, there is an accepting run of A over y where for every $i \in [\ell]$ and every symbol γ of y that was pushed in context ρ_i, γ is read through a transition within layer i.

The crucial properties of the above saturation procedure are stated in the following lemma.

Lemma 17. *Let M be a k-SMPDS, A and A' be ℓ-layered stack automata of M over Γ_h, and $\langle q, \ell \rangle$ be the sole state which is not isolated in the top layer of A. If $A' = Sat(A)$ then:*

1. *if there is a path in A' from a state $\langle p_1, \ell \rangle$ to a state $\langle p_2, \ell \rangle$ labeled with $x \in \Gamma_h^*$, then there is a context $\langle\langle p_2, \varepsilon \rangle\rangle \rightsquigarrow^h_M \langle\langle p_1, x \rangle\rangle$ (i.e., paths in the top-layer summarize contexts);*
2. *for every $y' \in \mathcal{L}(A', q')$, there exists a context $\langle\langle q, y \rangle\rangle \rightsquigarrow^h_M \langle\langle q', y' \rangle\rangle$ such that $y \in \mathcal{L}(A, q)$ (i.e., each configuration of thread T^h_M which is accepted by A' is reachable from a configuration accepted by A);*
3. *for every k-scoped multiple context run $\rho_1, \ldots, \rho_\ell$, where $\rho_\ell = \langle\langle q, y \rangle\rangle \rightsquigarrow^h_M \langle\langle q', y' \rangle\rangle$, if $y \in \mathcal{L}(A, q)$ and A matches $\rho_1, \ldots, \rho_\ell$ then $y' \in \mathcal{L}(A', q')$ (i.e., the last context of any k-scoped multiple context run of T^h_M that is matched by A always ends with a configuration accepted by A' whenever it starts from a configuration accepted by A).*

Proof. We prove parts 1 and 2 by induction on the number of transitions that are added in the saturation procedure. In the following, for $d \geq 0$, we will denote with A^d the ℓ-layered automaton resulting by adding d transitions to A though the saturation procedure.

We start with part 1. The base case is trivial since there are no transitions at the top layer of A.

For the induction step, suppose that the statement holds after that $d \geq 0$ transitions have been added in the saturation procedure and let $e = (\langle q_1, \ell \rangle, a, \langle q_2, \ell \rangle)$, for $a \in \Gamma_h \cup \{\varepsilon\}$, be the transition that is added next.

Let π be any top-layer path of A^{d+1} containing e, i.e., $\pi = \pi_1.e.\pi_2$, and let $x = x_1.a.x_2$ be the word labeling π with x_i labeling π_i for $i \in [2]$. Denoting $\langle p_1, \ell \rangle$ and $\langle p_2, \ell \rangle$ respectively the starting and ending states of π, since π_1 and π_2 are also paths of A^d we can apply the induction hypothesis and get $\langle\langle q_1, \varepsilon \rangle\rangle \rightsquigarrow^h_M \langle\langle p_1, x_1 \rangle\rangle$ and $\langle\langle p_2, \varepsilon \rangle\rangle \rightsquigarrow^h_M \langle\langle q_2, x_i \rangle\rangle$, thus we only need to show the existence of a context $\langle\langle q_2, \varepsilon \rangle\rangle \rightsquigarrow^h_M \langle\langle q_1, a \rangle\rangle$. We do this by case inspection on the rule that is applied to add e.

If e is added by a push or an internal transition of M, by applying such a transition we clearly get $\langle\langle q_2, \varepsilon \rangle\rangle \rightarrow^h_M \langle\langle q_1, a \rangle\rangle$ and thus $\langle\langle q_2, \varepsilon \rangle\rangle \rightsquigarrow^h_M \langle\langle q_1, a \rangle\rangle$ holds. If e is added by a pop transition, from the corresponding rule of the saturation procedure, there must be a pop transition of the form (p, γ, q_1) and a path π' of A^d labeled with γ from $\langle p, \ell \rangle$ to $\langle q_2, \ell \rangle$ (note that we are assuming that π lays entirely in the top layer). By applying the induction hypothesis to π', we get that $\langle\langle q_2, \varepsilon \rangle\rangle \rightsquigarrow^h_M \langle\langle p, \gamma \rangle\rangle$ holds. Thus extending this context by applying the pop transition (p, γ, q_1), we get $\langle\langle q_2, \varepsilon \rangle\rangle \rightsquigarrow^h_M \langle\langle p, \gamma \rangle\rangle \rightarrow^h_M \langle\langle q_1, \varepsilon \rangle\rangle$ that concludes this part of the proof (recall that $a = \varepsilon$ in this case).

We prove now part 2 of the lemma. The base case trivially holds since all the languages $\mathcal{L}(A, q')$ for $q' \neq q$ are empty.

For the induction step, suppose that the statement holds after that $d \geq 0$ transitions have been added in the saturation procedure and let e be the transition that is added next.

Pick any $y' \in \mathcal{L}(A^{d+1}, q')$. We recall that, by definition, in an ℓ-layered stack automaton cross-layer transitions can only take to states of lower indexed layers. Thus, a path labeled with y' from $\langle q', \ell \rangle$ through q_F can be split into three parts: a first part π_1 that stays all within the top layer, a transition $e' = (\langle q_1, \ell \rangle, a, \langle q_2, i \rangle)$ to a lower indexed layer $i < \ell$, and final part π_2 that leads to q_F. According to this splitting, y' can be decomposed as $x.a.z$. Observe that since π_1 is from $\langle q', \ell \rangle$ through $\langle q_1, \ell \rangle$, we can apply part 1 of the lemma, and thus there is a context $\sigma = \langle\langle q_1, \varepsilon \rangle\rangle \rightsquigarrow^h_M \langle\langle q', x \rangle\rangle$.

If $e' \neq e$, since any transition added by the saturation procedure starts from a top-layer state, then e must occur in π_1 and thus $a.z \in \mathcal{L}(A^d, q'_1)$. By the induction hypothesis, we get that there exists a context $\langle\langle q, y \rangle\rangle \rightsquigarrow^h_M \langle\langle q_1, a.z \rangle\rangle$ such that $y \in \mathcal{L}(A, q)$. By combining this with context σ above, we get $\langle\langle q, y \rangle\rangle \rightsquigarrow^h_M \langle\langle q_1, a.z \rangle\rangle \rightsquigarrow^h_M \langle\langle q', x.a.z \rangle\rangle$, and thus the statement holds in this case.

Now, suppose that $e' = e$, i.e., the transition leaving the top layer is e. We recall that in the saturation procedure the crossing layer transitions are only added through a pop transition and in this case the added transition is labeled with ε (i.e., $a = \varepsilon$). Thus, according to the pop-transition rule, there must be a pop transition of M of the form (p, γ, q_1) and there is a path of A^d from $\langle p, \ell \rangle$ to $\langle q_2, i \rangle$ that is labeled with γ. Hence, there is a path of A^d from $\langle p, \ell \rangle$ that is labeled with $\gamma.z$ (note that the saturation procedure does not add transitions that do not have at least an endpoint at the top level and π_2 starts from layer $i < \ell$), and therefore, $\gamma.z \in \mathcal{L}(A^d, p)$. By applying the induction hypothesis, we get that there is a context $\langle\langle q, y \rangle\rangle \rightsquigarrow^h_M \langle\langle p, \gamma.z \rangle\rangle$. We can then extend this context with the pop transition thus obtaining the context $\langle\langle q, y \rangle\rangle \rightsquigarrow^h_M \langle\langle q_1, z \rangle\rangle$. Again by combining this with context σ above, we get $\langle\langle q, y \rangle\rangle \rightsquigarrow^h_M \langle\langle q_1, z \rangle\rangle \rightsquigarrow^h_M \langle\langle q', x.a.z \rangle\rangle$ (recall $a = \varepsilon$), and thus the statement holds also in this case.

For part 3 of the lemma, we proceed by induction on the number d of transition steps in a context that starts from configuration $\langle\langle q, y\rangle\rangle$ with $y \in \mathcal{L}(A, q)$.

The base case (zero transitions) trivially holds for $q = q'$ and $y = y'$.

For the induction step, consider a context σ of the form $\langle\langle q, y\rangle\rangle \rightsquigarrow_M^h \langle\langle p, z\rangle\rangle \rightarrow_M^h \langle\langle q', y'\rangle\rangle$ with $d+1$ transitions and such that $y \in \mathcal{L}(A, q)$. By the induction hypothesis, $z \in \mathcal{L}(A', p)$. If transition $\langle\langle p, z\rangle\rangle \rightarrow_M^h \langle\langle q', y'\rangle\rangle$ is an internal one, then $y' = z$ and the saturation procedure would add a transition from $\langle q', \ell\rangle$ to $\langle p, \ell\rangle$ in A' labeled with ε. Similarly, if the transition pushes a symbol γ onto stack h, then $y' = \gamma.z$ and a transition from $\langle q', \ell\rangle$ to $\langle p, \ell\rangle$ labeled with γ is added in A'. In both cases, from $z \in \mathcal{L}(A', p)$ we get $y' \in \mathcal{L}(A', q')$. In the remaining case, i.e., when $\langle\langle p, z\rangle\rangle \rightarrow_M^h \langle\langle q', y'\rangle\rangle$ is a pop transition (p, γ, q'), then $z = \gamma y'$ and since σ is by hypothesis a context of a k-scoped run, γ was pushed onto the stack in a context ρ_i with $\ell - k < i \leq \ell$ (i.e., γ was pushed either in this context or in one of the previous $k-1$ contexts). Since by the induction hypothesis $z \in \mathcal{L}(A', p)$ and A matches the k-scoped multiple context run $\rho_1, \ldots, \rho_\ell$, there is a path of A' labeled γ from $\langle p, \ell\rangle$ to a state $\langle q'', i\rangle$ and a path from $\langle q'', i\rangle$ to q_F labeled y'. By the saturation algorithm, a transition $(\langle q', \ell\rangle, \varepsilon, \langle q'', i\rangle)$ must be added to A' and therefore $y' \in \mathcal{L}(A', q')$. Thus also part 3 holds, and therefore the lemma holds. □

6.1.2. Successor relation for layered automata

Given a bound m, by Lemma 17 we can construct a set of layered automata that accept all the stack contents that can occur in a configuration of any k-scoped run of M with at most m-maximal contexts. For this, we define an operation that adds a new layer to a layered automaton A and connects through an ε transition a state from the top layer of the resulting automaton to a non-isolated state of the top layer of A.

Formally, for an ℓ-layered automaton A with $\ell > 0$ and a top-layer state $\langle q, \ell\rangle$ of A that is connected to q_F, with $Add(A, p, q)$ we denote the $(\ell+1)$-layered stack automaton obtained from A by adding the transition $(\langle p, \ell+1\rangle, \varepsilon, \langle q, \ell\rangle)$. We extend this function also to A_ϵ^h, and denote with $Add(A_\epsilon^h, p, q_F)$ the 1-layered stack automaton containing only the transition $(\langle p, 1\rangle, \varepsilon, q_F)$.

For an ℓ-layered stack automaton over Γ_h with $\ell > 0$ and control states p, q of M where $\langle q, \ell\rangle$ is connected to q_F in A, we define the *successor* of A by (p, q), denoted $Succ(A, p, q)$, as $Sat(Add(A, p, q))$. Consistently with what we have done for Add, we extend this notion to A_ϵ^h and denote $Succ(A_\epsilon^h, p, q_F) = Sat(Add(A_\epsilon^h, p, q_F))$.

According to Lemma 17, if A accepts contents of stack h that can be reached in the final configuration of a k-scoped run ρ of M and matches the corresponding multiple context run, q is the control state of the last h-context in ρ and p is the control state of the ending configuration of ρ, then $Succ(A, p, q)$ accepts all the stack contents that can be reached by extending ρ with a context of stack h that starts with a configuration with control state p.

We iterate the definition of $Succ$ and define $Succ(A, \langle p_i, q_i\rangle_{i\in[m]})$ inductively as $Succ(A, p_1, q_1)$, if $m = 1$, and $Succ(A', p_m, q_m)$ where $A' = Succ(A, \langle p_i, q_i\rangle_{i\in[m-1]})$, otherwise.

From Proposition 16 and the above definitions, we get:

Proposition 18. *If A is an ℓ-layered stack automaton, then $Succ(A, \langle p_i, q_i\rangle_{i\in[m]})$ is an $(\ell+m)$-layered stack automaton.*

As already observed, from the definition of Sat, all the transitions that are added start from top-layer states and no transition is deleted. Thus we have:

Proposition 19. *Let A be an ℓ-layered automaton. If $A' = Succ(A, \langle p_i, q_i\rangle_{i\in[m]})$ then for each p and $\ell_p \leq \ell$: $(\langle p, \ell_p\rangle, \tau, \langle q, \ell_q\rangle)$ is a transition of A if and only if it is also a transition of A'.*

As a corollary of the above proposition we have that if a layered automaton A' is obtained by iterating the function $Succ$ from a layered automaton A, then the set of configurations accepted from the copy of q in layer ℓ' of A is the same as the one accepted from the copy of q in the same layer of A'.

Corollary 20. *Let A be an ℓ-layered automaton. If $A' = Succ(A, \langle p_i, q_i\rangle_{i\in[m]})$ then $\mathcal{L}(A, \langle q, \ell'\rangle, q_F) = \mathcal{L}(A', \langle q, \ell'\rangle, q_F)$ for each q and $\ell' \leq \ell$.*

Moreover, again from the definition of Sat, the transitions over a stack symbol are added only for the push transitions and are internal to the top layer. Thus in the construction of $Succ(A_\epsilon^h, \langle p_i, q_i\rangle_{i\in[\ell]})$ the addition of these transitions will occur in different layers matching a sequence of contexts $\rho_1, \ldots, \rho_\ell$ where $\rho_i = \langle\langle p_i, w_i\rangle\rangle \rightsquigarrow_M^h \langle\langle q_{i+1}, w_{i+1}\rangle\rangle$ for $i \in [\ell]$. Thus we have:

Proposition 21. *Let $A = Succ(A_\epsilon^h, \langle p_i, q_i\rangle_{i\in[\ell]})$. A matches any k-scoped multiple run $\rho_1, \ldots, \rho_\ell$ of T_M^h where $w_1 = \varepsilon$ and $\rho_i = \langle\langle p_i, w_i\rangle\rangle \rightsquigarrow_M^h \langle\langle q_{i+1}, w_{i+1}\rangle\rangle$ for $i \in [\ell]$.*

The following property is crucial in our solution for the reachability problem.

Theorem 22. *Let M be a k-SMPDS and $A = Succ(A^h_\epsilon, \langle p_i, q_i\rangle_{i\in[\ell]})$.*

A is an ℓ-layered stack automaton such that:

$w_{i+1} \in \mathcal{L}(A, \langle q_{i+1}, i\rangle, q_F)$ for $i \in [\ell]$ if and only if a k-scoped multiple context run $\rho_1, \ldots, \rho_\ell$ of T^h_M exists such that $\rho_i = \langle\langle p_i, w_i\rangle\rangle \rightsquigarrow^h_M \langle\langle q_{i+1}, w_{i+1}\rangle\rangle$ for $i \in [\ell]$.

Proof. From Proposition 18, we get that A is an ℓ-layered stack automaton. Denoting $A_m = Succ(A^h_\epsilon, \langle p_i, q_i\rangle_{i\in[m]})$ for $m \in [\ell]$. By Corollary 20, to complete the proof we only need to show by induction on $m \geq 1$ that:
$w_{m+1} \in \mathcal{L}(A_m, q_{m+1})$ if and only if a k-scoped multiple context run $\rho_1, \ldots, \rho_m$ of T^h_M exists such that $\rho_i = \langle\langle p_i, w_i\rangle\rangle \rightsquigarrow^h_M \langle\langle q_{i+1}, w_{i+1}\rangle\rangle$ for $i \in [m]$.

For the base case, i.e., $m = 1$, we observe that since $A_1 = Succ(A^h_\epsilon, p_1, q_1)$, if $w_2 \in \mathcal{L}(A_1, q_2)$ then there must be a path from $\langle q_2, 1\rangle$ through $\langle p_1, 1\rangle$ labeled with w_2 (recall that p_1 is connected to $q_1 = q_F$ through an ε-transition). From part 1 of Lemma 17, we get that $\langle\langle p_1, \varepsilon\rangle\rangle \rightsquigarrow^h_M \langle\langle q_2, w_2\rangle\rangle$. Vice-versa, if there is a run $\langle\langle p_1, \varepsilon\rangle\rangle \rightsquigarrow^h_M \langle\langle q_2, w_2\rangle\rangle$, from part 3 of Lemma 17, since $\varepsilon \in \mathcal{L}(A_1, p_1)$ we get that $w_2 \in \mathcal{L}(A_1, q_2)$.

For the induction step, we assume that the theorem holds for $m-1$. Denote $A' = Add(A_{m-1}, p_m, q_m)$. Clearly, by definition, $A_m = Sat(A')$ holds.

Assume first that $w_{m+1} \in \mathcal{L}(A_m, q_{m+1})$. From part 2 of Lemma 17, then there is a context $\rho_m = \langle\langle p_m, w_m\rangle\rangle \rightsquigarrow^h_M \langle\langle q_{m+1}, w_{m+1}\rangle\rangle$ such that $w_m \in \mathcal{L}(A', p_m)$. Since A' is obtained from A_{m-1} by adding an ε-transition from p_m to q_m, we also get that $w_m \in \mathcal{L}(A_{m-1}, q_m)$. By applying the induction hypothesis, we get that a k-scoped multiple context run $\rho_1, \ldots, \rho_{m-1}$ of T^h_M exists such that $\rho_i = \langle\langle p_i, w_i\rangle\rangle \rightsquigarrow^h_M \langle\langle q_{i+1}, w_{i+1}\rangle\rangle$ for $i \in [m-1]$. Thus, we get that there is a k-scoped multiple context run $\rho_1, \ldots, \rho_m$ of T^h_M such that $\rho_i = \langle\langle p_i, w_i\rangle\rangle \rightsquigarrow^h_M \langle\langle q_{i+1}, w_{i+1}\rangle\rangle$ for $i \in [m]$.

For the other direction, assume that there exists a k-scoped multiple context run $\rho_1, \ldots, \rho_m$ of T^h_M such that $\rho_i = \langle\langle p_i, w_i\rangle\rangle \rightsquigarrow^h_M \langle\langle q_{i+1}, w_{i+1}\rangle\rangle$ for $i \in [m]$. From the induction hypothesis we get that $w_m \in \mathcal{L}(A_{m-1}, q_m)$, and thus $w_m \in \mathcal{L}(A', p_m)$. Since there is a context $\langle\langle p_m, w_m\rangle\rangle \rightsquigarrow^h_M \langle\langle q_{m+1}, w_{m+1}\rangle\rangle$, from Proposition 21 and part 3 of Lemma 17 then also $w_{m+1} \in \mathcal{L}(A_m, q_{m+1})$ and thus the lemma holds. □

Directly from the definition of thread interface, we get that for states q_i and p_i as in the above lemma, $\langle p_i, q_{i+1}\rangle_{i\in[m]}$ is a h-thread interface of M. Thus, the following corollary holds.

Corollary 23. *Let M be a k-SMPDS. If $Succ(A^h_\epsilon, \langle p_i, q_i\rangle_{i\in[m]})$ is defined, then the tuple $\langle p_i, q_{i+1}\rangle_{i\in[m]}$ is a h-thread interface of M.*

6.2. Capturing thread configurations for unboundedly many contexts: thread automata

We start giving a property that will be used to construct an automaton that accepts the thread configurations that are reachable in k-scoped multiple context runs with any number of contexts.

For an ℓ-layered automaton A, denote with $Top_k(A)$ the layered automaton obtained from A by keeping only the top k layers (along with layer 0). Since in a layered automaton each state is either isolated or connected to q_F by a run, by removing the layers of A we preserve this property for the remaining states in $Top_k(A)$. Namely, we define $Top_k(A)$ as follows. If $\ell \leq k$, $Top_k(A)$ is A, otherwise $Top_k(A)$ is the k-layered automaton whose transition set is the smallest set such that: if $(\langle q, j\rangle, \gamma, s)$ is a transition of A with $j > \ell - k$, then $(\langle q, j - \ell + k\rangle, \gamma, s')$ is a transition of $Top_k(A)$ where s' is $\langle q', j' - \ell + k\rangle$, if $s = \langle q', j'\rangle$ for some $j' \in [\ell - k + 1, k]$, and is q_F, otherwise.

We recall that for an ℓ-layered automaton A, *Sat* adds only transitions from a top-layer state to states of the form $\langle p, \ell'\rangle$ where $p \in Q$ and $\ell' \in [\ell - k + 1, \ell]$, and no direct transition to q_F is added even if $\ell < k$. Thus directly from the definitions we get the following property:

Proposition 24. *Given an ℓ-layered automaton A and denoting $A' = Top_k(A)$:*
$Top_k(Succ(A', \langle p_i, q_i\rangle_{i\in[m]})) = Top_k(Succ(A, \langle p_i, q_i\rangle_{i\in[m]}))$.

We use functions *Succ* and Top_k to define the notion of *thread automaton*. The thread automaton of a thread T^h_M, denoted $\mathcal{A}_{h,M}$, is designed to accept all and only the stack contents w of T^h_M that can occur in a k-scoped multiple context run of T^h_M. For this, $\mathcal{A}_{h,M}$ explores backwards the contexts of T^h_M that can occur in such a run and simulates the transitions of layered automata connected through *Succ* (which according to Theorem 22 guarantees that w can be reached on a multiple context run of T^h_M). By exploiting Propositions 19 and 24, this can be done with layered automata with at most k layers using function Top_k to keep the number of layers within the bound k.

The states of $\mathcal{A}_{h,M}$ are of the form (d, p, A, q, q') where $d \in [k]$, A is an ℓ-layered stack automaton with $\ell \leq k$, and p, q, and q' are control states of M. Each state (d, p, A, q, q') denotes contexts starting from q and ending at q', moreover:

- A captures the top portions of the stack that are reachable by such contexts;
- p is the current control state, and if $d = 1$, it is updated by simulating A on the top-layer copy of p, otherwise it stays unchanged;

- d is used to guide the simulation through the chain of *Succ*-connected layered automata; in particular, if $d > 1$, $\mathcal{A}_{h,M}$ just moves to a state corresponding to a previous context (recall that contexts are explored backwards) updating the A component via the composed function $Top_k \circ Succ$ and decreasing d by 1; as soon as $d = 1$ holds, $\mathcal{A}_{h,M}$ simulates A starting from the top-layer copy of p until a transition that crosses the top layer is taken; when this occurs, denoting with ℓ the top layer of A and with ℓ' the layer of the target of the crossing transition, d is then updated to $\ell - \ell' + 1$, thus enforcing that the d component will evaluate 1 again when a context corresponding to layer ℓ' of A is reached.

Formally, the *thread automaton* $\mathcal{A}_{h,M} = (Q_{h,M}, Q^0_{h,M}, \Gamma_h, \Delta_{h,M}, Q^F_{h,M})$ is such that:

1. the set of states $Q_{h,M}$ contains all the tuples of the form (d, p, A, q, q') where $d \in [0, k]$, A is an ℓ-layered stack automaton on alphabet Γ_h with $\ell \in [k]$, and $p, q, q' \in Q$ are such that the copies of q and q' in the top layer of A are not isolated (and thus connected to q_F);
2. the set of initial states $Q^0_{h,M} \subseteq Q_{h,M}$ is the set of all the states of the form $(1, p, A, q, q')$ where the top-layer copy of p in A is not isolated and $p = q'$;
3. the set of final states $Q^F_{h,M} \subseteq Q_{h,M}$ is the set of all the states of the form $(1, p, A, q, q')$ where A is 1-layered, $p = q$ and $A = Succ(A^h_\epsilon, p, q_F)$;
4. $\Delta_{h,M} \subseteq Q_{h,M} \times (\Gamma_h \cup \{\varepsilon\}) \times Q_{h,M}$ is the set of all the transitions (s_1, τ, s_2) where $s_1 \notin Q^F_{h,M}$ (i.e., there are no transitions from a final state) and denoting $s_i = (d_i, p_i, A_i, q_i, q'_i)$ for $i \in [2]$, one of the following cases applies (in the following description, if a component of s_2 is not mentioned then it is equal to the same component of s_1):

   [simulation] $d_1 = 1$ and $(\langle p_1, \ell\rangle, \tau, \langle p_2, \ell\rangle)$ is a transition of A_1 where ℓ is the top-layer index of A_1;
   [simulation end] $d_1 = 1$, there is a transition $(\langle p_1, \ell_1\rangle, \tau, \langle p_2, \ell_2\rangle)$ of A_1 where $\ell_2 < \ell_1$ and ℓ_1 is the top-layer index of A_1 (i.e., the A_1 transition crosses the top layer), and $d_2 = \ell_1 - \ell_2 + 1$ (i.e., there is no further stack content to parse in this context and in the next $\ell_1 - \ell_2 - 1$ contexts that will be processed);
   [context update] $\tau = \varepsilon$, $d_1 > 1$, $A_1 = Top_k(Succ(A_2, q_1, q'_2))$ and $d_2 = d_1 - 1$ (i.e., there is no stack content to read from this context thus the automaton just moves to a previous context).

The following lemma states a property on the structure of the accepting runs of $\mathcal{A}_{h,M}$.

Lemma 25. *Any accepting run ρ of $\mathcal{A}_{h,M}$ can be decomposed as*

$$\rho^s_1, \rho^e_1, \rho^c_1, \dots \rho^s_{\ell-1}, \rho^e_{\ell-1}, \rho^c_{\ell-1}, \rho^s_\ell$$

where each ρ^s_i is a possibly empty sequence of simulation transitions, each ρ^e_i is a single simulation-end transition and each ρ^c_i is a non-empty sequence of context-update transitions.

Proof. We observe that each run starts from a state of the form $(1, p, A, q, q')$ and thus only simulation and simulation-end transitions can be taken. As soon as a simulation-end transition is taken, $\mathcal{A}_{h,M}$ enters a state with $d \geq 2$ as first component and thus the next transition can only be a context-update one. Since each context-update transition decreases by 1 the value of this state component, after $d - 1$ context-update transitions a state with 1 in its first component is entered and thus only simulation and simulation-end transitions are possible. Finally, since a final state has 1 as first component, an accepting run must end either with a context-update or a simulation transition (recall ρ^s_ℓ is possibly empty). □

We observe that along a run of $\mathcal{A}_{h,M}$ the layered automaton in the states can only change by taking context-update transitions. Let $\rho^s_1, \rho^e_1, \rho^c_1, \dots \rho^s_{\ell-1}, \rho^e_{\ell-1}, \rho^c_{\ell-1}, \rho^s_\ell$ be a decomposition of a run ρ as above. We denote with $lsa(\rho)$ the sequence $A_1 \dots A_m$ of the layered stack automata occurring in $\rho^c_1, \dots \rho^c_{\ell-1}$.

The following theorem states the wished property for tread automata.

Theorem 26.

1. *For a layered stack automaton $A = Succ(A^h_\epsilon, \langle p_i, q_i\rangle_{i\in[m]})$, if $w \in \mathcal{L}(A, q)$ then w is accepted by $\mathcal{A}_{h,M}$ starting from $(1, q, Top_k(A), p_m, q)$.*
2. *If ρ is an accepting run of $\mathcal{A}_{h,M}$ over w starting from $(1, q, A_m, p, q)$ and with $lsa(\rho) = A_m \dots A_1$, then there is a sequence $\langle p_i, q_i\rangle_{i\in[m]}$ such that $w \in \mathcal{L}(A'_m, q)$ and $A_j = Top_k(A'_j)$ for $j \in [m]$, where $A'_j = Succ(A^h_\epsilon, \langle p_i, q_i\rangle_{i\in[j]})$ for $j \in [m]$.*

Proof. We start showing part 1. Denoting $A = Succ(A^h_\epsilon, \langle p_i, q_i\rangle_{i\in[m]})$ assume $w \in \mathcal{L}(A, q)$.

We recall that *Sat* adds only transitions from a top layer state to states of the form $\langle p, \ell'\rangle$ where $p \in Q$, $\ell' > 0$ and $\ell' \in [\ell - k + 1, \ell]$. Moreover, $Succ(A^h_\epsilon, \langle p_i, q_i\rangle_{i\in[m]})$ is defined provided that $q_1 = q_F$. Thus, each run of A accepting w from $\langle q, m\rangle$ must be of the form

$$\langle p'_1, m_1\rangle \overset{w_1}{\rightsquigarrow} \langle q'_1, m_1\rangle \xrightarrow{\varepsilon} \langle p'_2, m_2\rangle \overset{w_2}{\rightsquigarrow} \dots \langle p'_d, m_d\rangle \overset{w_d}{\rightsquigarrow} \langle q'_d, m_d\rangle \xrightarrow{\varepsilon} q_F$$

where $w = w_1 w_2 \dots w_d$, $m = m_1$, $p'_1 = q$, $m_d = 1$, $q'_d = p_1$, and $0 < m_i - m_{i+1} < k$ for $i \in [d-1]$.

For $j \in [m]$, denote $A_j = Succ(A^h_\epsilon, \langle p_i, q_i\rangle_{i\in[j]})$ and $A'_j = Top_k(A_j)$.

Moreover, denote $q_F = \langle p'_{d+1}, 0\rangle$, $m_{d+1} = 0$, and for $j \in [d]$, $\rho_j = \langle p'_j, m_j\rangle \overset{w_j}{\rightsquigarrow} \langle q'_j, m_j\rangle \xrightarrow{\varepsilon} \langle p'_{j+1}, m_{j+1}\rangle$. For $j \in [d]$, we thus get from Proposition 19 that ρ_j is also a run of A_{m_j} and hence by the definition of Top_k, $\rho'_j = \langle p'_j, k_j\rangle \overset{w_j}{\rightsquigarrow} \langle q'_j, k_j\rangle \xrightarrow{\varepsilon} \langle p'_{j+1}, \ell_j\rangle$ is a run of A'_{m_j} where $\ell_j = m_{j+1} - m_j + k_j$ and $k_j \le k$ is the top layer index of A'_{m_j}.

Using the above runs ρ'_j we can construct a corresponding run π of $\mathcal{A}_{h,M}$ over w as the composition of runs $\pi_1, \dots, \pi_d$ defined as follows.

The starting state of π_1 is $s_1 = (1, p'_1, A'_{m_1}, p_{m_1}, p'_1)$, that is, we wish to simulate A'_{m_1} starting from the top-layer copy of p'_1 and $A'_{m_1} = Top_k(A_{m_1})$ where $A_{m_1} = Succ(A_{m_1-1}, \langle p_{m_1}, q_{m_1}\rangle)$ (recall $m = m_1$).

For each $j \in [d-1]$, we construct π_j as a run of the form

$$s_j \overset{w_j}{\rightsquigarrow}_{\mathcal{A}_{h,M}} s'_j \xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} s''_j \xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} \dots \xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} s_{j+1}.$$

The first portion $s_j \overset{w_j}{\rightsquigarrow}_{\mathcal{A}_{h,M}} s'_j$ is obtained by simulation transitions that correspond to the transitions of the run $\langle p'_j, k_j\rangle \overset{w_j}{\rightsquigarrow}_{A'_{m_j}} \langle q'_j, k_j\rangle$. In particular, we let $s_j = (1, p'_j, A'_{m_j}, p_{m_j}, q_{m_j+1})$ and $s'_j = (1, q'_j, A'_{m_j}, p_{m_j}, q_{m_j+1})$.

The next transition $s'_j \xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} s''_j$ is a simulation-end transition that captures transition $\langle q'_j, k_j\rangle \xrightarrow{\varepsilon}_{A'_{m_j}} \langle p'_{j+1}, \ell_j\rangle$. Thus, by definition s''_j is of the form $(k_j - \ell_j + 1, p'_{j+1}, A'_{m_j}, p_{m_j}, q_{m_j+1})$.

For the final part $s''_j \xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} \dots \xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} s_{j+1}$ we use context-update transitions. In particular, from the definitions, we have that $k_j - \ell_j = m_j - m_{j+1}$. Thus, from $A_{i+1} = Succ(A_i, p_{i+1}, q_{i+1})$ for $i \in [m-1]$ and Proposition 24, we get that $\mathcal{A}_{h,M}$ has context-update transitions to form the following run (where we have denoted $r_j = m_j - m_{j+1}$):

$$\begin{aligned} s''_j = (r_j + 1, p'_{j+1}, A'_{m_j}, p_{m_j}, q_{m_j+1}) &\xrightarrow{\varepsilon} (r_j, p'_{j+1}, A'_{m_j-1}, p_{m_j-1}, q_{m_j}) \dots \\ &\xrightarrow{\varepsilon} (1, p'_{j+1}, A'_{m_{j+1}}, p_{m_{j+1}}, q_{m_{j+1}+1}) = s_{j+1}. \end{aligned}$$

We construct the remaining run π_d by simply using simulation transitions. From the construction of $\pi_1, \dots, \pi_{d-1}$, π_d must start from $s_d = (1, p'_d, A'_{m_d}, p_{m_d}, q_{m_d+1})$. We recall that $m_d = 1$, thus $s_d = (1, p'_d, A'_1, p_1, q_2)$. From $\langle p'_d, k_d\rangle \overset{w_d}{\rightsquigarrow}_{A'_1} \langle q'_d, k_d\rangle$, we get $(1, p'_d, A'_1, p_1, q_2) \overset{w_d}{\rightsquigarrow}_{\mathcal{A}_{h,M}} (1, q'_d, A'_1, p_1, q_2)$. Since $q'_d = p_1$, A'_1 is 1-layered and $A'_1 = Succ(A^h_\epsilon, p_1, q_F)$, we get that $(1, q'_d, A'_1, p_1, q_2)$ is final.

We thus have defined a run π of $\mathcal{A}_{h,M}$ over $w = w_1 \dots w_d$ that starts from $s_1 = (1, p'_1, A'_{m_1}, p_{m_1}, p'_1)$ and ends at a final state. We recall that $p'_1 = q$, $m_1 = m$, $A_m = A$ and $A'_m = Top_k(A_m)$. Thus indeed $s_1 = (1, q, Top_k(A), p_m, q)$ and we are done with this part of the proof.

Now we show part 2 of the theorem. For this suppose that w is accepted by $\mathcal{A}_{h,M}$ and let $\rho = s_m \xrightarrow{\tau_m}_{\mathcal{A}_{h,M}} \dots \xrightarrow{\tau_1}_{\mathcal{A}_{h,M}} s_0$ be an accepting run over w where $s_i = (d_i, p'_i, A_i, p_i, q_i)$ for $i \in [m]$.

Let $\rho^s_1, \rho^e_1, \rho^c_1, \dots \rho^s_{d-1}, \rho^e_{d-1}, \rho^c_{d-1}, \rho^s_d$ be a decomposition of ρ as in Lemma 25, and let $w = w_1 \dots w_d$ the corresponding decomposition of w.

We have the following facts:

1. s_m is an initial state, thus $d_m = 1$ and $p'_m = q_m$ must hold;
2. s_0 is a final state, thus $d_0 = 1$ and $A_0 = Succ(A^h_\epsilon, p_0, q_F)$ must hold;
3. for $j \in [d]$, ρ^s_j is of the form $s_{m_j} \overset{w_j}{\rightsquigarrow}_{\mathcal{A}_{h,M}} s_{m'_j}$ where for $i \in [m'_j, m_j]$: $d_i = 1$, $A_i = A_{m_j}$, $p_i = p_{m_j}$ and $q_i = q_{m_j}$ (i.e., the only component that is updated in the simulation transitions is p'_i); note that $m_1 = m$ and $m'_d = 0$;
4. for $j \in [d-1]$, ρ^e_j is of the form $s_{m'_j} \xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} s_{m'_j-1}$, that is,

$$(1, p'_{m'_j}, A_{m_j}, p_{m_j}, q_{m_j}) \xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} (k_j + 1, \bar{p}_j, A_{m_j}, p_{m_j}, q_{m_j})$$

   where $k_j = \ell'_j - \ell''_j > 0$ and $\langle p'_{m'_j}, \ell'_j\rangle \xrightarrow{\varepsilon}_{A_{m_j}} \langle \bar{p}_j, \ell''_j\rangle$;
5. for $j \in [d-1]$, ρ^e_j is of the form $s_{m'_j-1} \overset{\varepsilon}{\rightsquigarrow}_{\mathcal{A}_{h,M}} s_{m_{j+1}}$, that is,

$$\begin{aligned} (k_j + 1, \bar{p}_j, A_{m_j}, p_{m_j}, q_{m_j}) &\xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} (k_j, \bar{p}_j, A_{m'_j-2}, p_{m'_j-2}, q_{m'_j-2}) \dots \\ &\xrightarrow{\varepsilon}_{\mathcal{A}_{h,M}} (1, \bar{p}_j, A_{m_{j+1}}, p_{m_{j+1}}, q_{m_{j+1}}) \end{aligned}$$

   where for $i \in [m_{j+1}, m'_j - 2]$, $A_{i+1} = Top_k(Succ(A_i, p_{i+1}, q_i))$.

We denote $B_d = A_0$ and for $j \in [d-1]$, $B_j = Succ(B_{j+1}, \langle p_{i+1}, q_i \rangle_{i \in [m_{j+1}, m'_j - 2]})$. By an inductive application of Proposition 24 we can show that $A_{m_j} = Top_k(B_j)$ for $j \in [d]$. Thus to conclude the proof we only need to show that $w \in \mathcal{L}(B_1, p'_m)$ (recall $m = m_1$). We prove this by induction showing that $w_j \dots w_d \in \mathcal{L}(B_j, p'_{m_j})$ for $j \in [d]$.

From fact 3 above, we have $s_{m_d} \overset{w_d}{\rightsquigarrow}_{\mathcal{A}_{h,M}} s_0$ and since $B_d = A_0 = Succ(A^h_\epsilon, p_0, q_F)$, we get that the base case $w_d \in \mathcal{L}(B_d, q_{m_d})$ clearly holds.

Suppose now that $w_{j+1} \dots w_d \in \mathcal{L}(B_{j+1}, p'_{m_{j+1}})$ holds. From Corollary 20, we get $w_{j+1} \dots w_d \in \mathcal{L}(B_j, \langle p'_{m_{j+1}}, \ell \rangle, q_F)$ where ℓ is the top-layer index of B_{j+1}. Moreover, again from fact 3 above $s_{m_j} \overset{w_j}{\rightsquigarrow}_{\mathcal{A}_{h,M}} s_{m'_j}$ and from fact 4 $s_{m'_j} \overset{\varepsilon}{\rightarrow}_{\mathcal{A}_{h,M}} s_{m'_j - 1}$ (recall that $p'_{m'_j - 1} = p'_{m_{j+1}}$). Thus, from the definition of $\mathcal{A}_{h,M}$, we get $\langle p'_{m_j}, \ell'_j \rangle \overset{w_j}{\rightsquigarrow}_{A_{m_j}} \langle p'_{m'_j}, \ell'_j \rangle \overset{\varepsilon}{\rightarrow}_{A_{m_j}} \langle p'_{m_{j+1}}, \ell''_j \rangle$ where ℓ'_j is the top layer index of A_{m_j}. Also, from fact 5 and the definition of B_j, we get $k_j = \ell'_j - \ell''_j = m'_j - 1 - m_{j+1}$. From Proposition 18, the top layer index of B_j is $\ell' = \ell + m'_j - 1 - m_{j+1}$ (recall that ℓ denotes the top layer index of B_{j+1}), and thus $k_j = \ell' - \ell$. By the inductive application of Proposition 24 mentioned above, we have $A_{m_j} = Top_k(B_j)$, and thus $\langle p'_{m_j}, \ell' \rangle \overset{w_j}{\rightsquigarrow}_{B_j} \langle p'_{m'_j}, \ell' \rangle \overset{\varepsilon}{\rightarrow}_{B_j} \langle p'_{m_{j+1}}, \ell \rangle$. Therefore by applying the inductive hypothesis we get $w_j \dots w_d \in \mathcal{L}(B_j, p'_{m_j})$ that concludes the proof. □

6.3. Solving configuration reachability

By Theorem 26 and Corollary 23, each run of a thread automaton corresponds to a thread interface (that is explored backwards along the run). Thus to solve the configuration reachability problem we can construct a finite automaton $\mathcal{R}$ that composes the thread automata, one for each thread of the MPDS, synchronizing them on context-switches. This would clearly achieve the effect of stitching together the corresponding thread interfaces and thus by Theorem 13, it would suffice to witness the existence of a corresponding k-scoped run of the MPDS.

Concerning to the technical construction, automaton $\mathcal{R}$ operates essentially in two modes: a simulation mode and a context-switching mode. In the *simulation mode*, $\mathcal{R}$ executes for one of the thread automata a sequence of simulation transitions followed by a simulation-end transition, and then switches to the context-switching mode. In the *context-switching mode*, it attempts to stitch (backwards) to the computation of M simulated so far, the context shown in the current state of one of the thread automata and if it succeeds it updates the state of M reached so far and the state of the thread automaton by taking a context-update transition (that will give the next context to use for this thread). From this mode, $\mathcal{R}$ switches back to the simulation mode if one of the thread automata can execute either a simulation or a simulation-end transition. To implement these behaviors, we use states of the form $(h, q, \bar{q}_1, \dots, \bar{q}_n)$ where h denotes the mode, q is the state of M (backwards) reached so far, and $\bar{q}_1, \dots, \bar{q}_n$ are the current states of the thread automata. We use $h = 0$ to denote the context-switching mode, and $h \in [n]$ to denote the thread automaton currently executed in the simulation mode. The acceptance condition requires that we have reached an initial state of M and all the thread automata have reached a final state.

Formally, denoting $\mathcal{A}_{h,M} = (Q_{h,M}, Q^0_{h,M}, \Gamma_h, \Delta_{h,M}, Q^F_{h,M})$ the thread automaton for thread T^h_M, we define $\mathcal{R}$ as the finite automaton $(S, I, \Gamma, \Delta, F)$ where:

- the set of states S is $[0, n] \times Q \times \prod_{i \in [n]} Q_{i,M}$;
- the set of initial states I is the set of all tuples of the form $(h, q, \bar{q}_1, \dots, \bar{q}_n)$ where $h \in [n]$, $\bar{q}_i = (1, p_i, A_i, q_i, p_i) \in Q^0_{i,M}$ for $i \in [n]$ and $p_h = q$;
- the set of final states F is the set of all states of the form $(h, q, \bar{q}_1, \dots, \bar{q}_n)$ where $q \in Q_I$ (i.e., q is an initial state of M) and $\bar{q}_i \in Q^F_{i,M}$ for $i \in [n]$;
- for $i \in [2]$, denote $s_i = (h_i, q_i, \bar{q}_{i,1}, \dots, \bar{q}_{i,n})$ where for $j \in [n]$, $\bar{q}_{i,j} = (d_{i,j}, p_{i,j}, A_{i,j}, q_{i,j}, q'_{i,j})$, the set of transition Δ is the set of the tuples (s_1, τ, s_2) such that $s_1 \notin F$ and one of the following cases applies (in the following description, if a component of s_2 is not mentioned then it is equal to the same component of s_1):

  [simulation mode] $h_1 > 0$ and the following holds:
  - $d_{1,h_1} = 1$ and $\bar{q}_{1,h_1} \overset{\tau}{\rightarrow}_{\mathcal{A}_{h,M}} \bar{q}_{2,h_1}$ (i.e., this is either a simulation or simulation-end transition of $\mathcal{A}_{h_1,M}$);
  - if $d_{2,h_1} > 1$ then $h_2 = 0$ (change to context-switching mode);

  [context-switching mode] $h_1 = 0$, $\tau = \varepsilon$ and the following holds:
  - if there exists $h \in [n]$ such that $d_{1,h} = 1$ then $h_2 = h$ (change to simulation mode),
  - otherwise (keep staying in context-switching mode), there exists a $h \in [n]$ such that:
    - $d_{1,h} > 1$ and $\bar{q}_{1,h} \overset{\tau}{\rightarrow}_{\mathcal{A}_{h,M}} \bar{q}_{2,h}$ (i.e., this is a context-update transition of $\mathcal{A}_{h,M}$);
    - $q'_{1,h} = q_1$ (i.e., the current context of $\mathcal{A}_{h,M}$ ends with the state of M reached so far and thus we can stitch it to the current computation);
    - $q_2 = q_{1,h}$ (i.e., the state of M reached so far is updated to the initial state of the current context of $\mathcal{A}_{h,M}$);

Given two words $v, w \in \Sigma^*$, with $v \,|||\, w$ we denote the shuffle product of v and w, that is, the set of words $v_1 w_1 \dots v_m w_m$ where $= v_1 \dots v_m$, $w = w_1 \dots w_m$ and $v_i, w_i \in \Sigma^*$. With $|||_{i\in[n]} w_i$ we denote its generalization to n words $w_1, \dots, w_n$. These operators generalize to languages as usual and we omit it here. We get the following theorem that relates k-scoped runs to $\mathcal{R}$ runs.

Theorem 27. *Given $q \in Q$ and $w_i \in \Gamma^*$ for $i \in [n]$, the following statements are equivalent:*

1. *There is a k-scoped run of the form $\langle\langle q_0, \varepsilon, \dots, \varepsilon\rangle\rangle \rightsquigarrow_M \langle\langle q, w_1, \dots, w_n\rangle\rangle$ with $q_0 \in Q_I$.*
2. *There is a run of $\mathcal{R}$ over a word $w' \in |||_{i\in[n]} w_i$ that starts from an initial state of the form $(h, q, \bar{q}_1, \dots, \bar{q}_n)$ and ends at a final state of the form $(h', q_0, \bar{q}'_1, \dots, \bar{q}'_n)$.*

Proof. We start showing the implication from 1 to 2. Consider a k-scoped run ρ of the form $\langle\langle q_0, \varepsilon, \dots, \varepsilon\rangle\rangle \rightsquigarrow_M \langle\langle q, w_1, \dots, w_n\rangle\rangle$. By taking a splitting of ρ into maximal contexts, we can define n k-scoped multiple context runs, say $\rho_1, \dots, \rho_n$, one for each thread. For $i \in [n]$, let ρ_i be formed of $\rho_{i,1}, \dots, \rho_{i,m_i}$ where $\rho_{i,j} = \langle\langle p_{i,j}, w_{i,j}\rangle\rangle \rightsquigarrow^j_M \langle\langle q_{i,j+1}, w_{i,j+1}\rangle\rangle$ for $j \in [m_i]$. Clearly, for $i \in [n]$, $w_{i,m_i+1} = w_i$ must hold.

Denoting $A_i = Succ(A^i_\epsilon, \langle p_{i,j}, q_{i,j}\rangle_{j\in[m_i]})$, from Theorem 22 we get that $w_i \in \mathcal{L}(A_i, q_{i,m_i+1})$ and thus by part 1 of Theorem 26, w_i is accepted by $\mathcal{A}_{i,M}$ from a state of the form $(1, q_{i,m_i+1}, Top_k(A), p_{i,m_i}, q_{i,m_i+1})$.

We can construct an accepting run of $\mathcal{R}$ over an interleaving of the words $w_1, \dots, w_n$ that reflects the reversed order of the considered splitting of ρ. The run starts from a state of the form $(h, q, \bar{q}_1, \dots, \bar{q}_n)$ where each $\bar{q}_i$ corresponds to the context ρ_{i,m_i} and $h \in [n]$ is such that the context of $\bar{q}_h$ ends with q_{h,m_h+1}. Then, for each $i \in [n]$, $\mathcal{R}$ simulates the transitions of an accepting run of $\mathcal{A}_{i,M}$ over w_i. For this, it alternates between the thread automata according to the reversed order of the considered splitting of ρ. As soon as the first maximal context of ρ is processed, $\mathcal{R}$ updates the state of M reached so far with the starting state q_0 of ρ, and since all the thread automata have reached their respective final states, this state is also final and we are done with the proof of the implication from 1 to 2.

For the implication from 2 to 1, consider a run ρ of $\mathcal{R}$ over a word $w' \in |||_{i\in[n]} w_i$ that starts from an initial state of the form $(h, q, \bar{q}_1, \dots, \bar{q}_n)$ and ends at a final state of the form $(h', q_0, \bar{q}'_1, \dots, \bar{q}'_n)$. Recall that the stack alphabets Γ_i for $i \in [n]$ are disjoint. Thus, from ρ we can uniquely define the runs ρ_i of $\mathcal{A}_{i,M}$ over w_i for $i \in [n]$. From the construction of $\mathcal{R}$, we have that each ρ_i is accepting, starts from $\bar{q}_i$ and ends at $\bar{q}'_i$.

Denoting $\bar{q}_i = (1, q, A, p, q)$ and $lsa(\rho_i) = A_{i,m_i} \dots A_{i,1}$, from part 2 of Theorem 26 there is a sequence $\langle p_{i,j}, q_{i,j}\rangle_{j\in[m_i]}$ such that $A'_{i,j} = Succ(A^i_\epsilon, \langle p_{i,j}, q_{i,j}\rangle_{i\in[j]})$ and $A_{i,j} = Top_k(A'_{i,j})$ for $j \in [m_i]$, and $w_i \in \mathcal{L}(A'_{i,m_i}, q)$.

Thus, by Theorem 22, there is a k-scoped multiple context run $\rho_{i,1}, \dots, \rho_{i,m_i}$ of T^i_M such that $\rho_{i,j} = \langle\langle p_{i,j}, w_{i,j}\rangle\rangle \rightsquigarrow^i_M \langle\langle q_{i,j+1}, w_{i,j+1}\rangle\rangle$ for $j \in [m_i]$ and $w_{i,m_i+1} = w_i$.

To conclude the proof, we observe that we can stitch together the contexts from the k-scoped multiple context runs according to the (reversed) sequence of contexts explored in the run ρ while in the context-switching mode thus obtaining a k-scoped accepting run of M starting from q_0. Since $(h, q, \bar{q}_1, \dots, \bar{q}_n)$ is initial for $\mathcal{R}$ we get $q_0 \in Q_I$ and we are done. □

6.3.1. Computational complexity of configuration reachability of SMPDS

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

By Theorem 27, the reachability problem for SMPDS can be reduced to checking the emptiness of $\bigcup_{q\in P}(\mathcal{L}(\mathcal{R}, q) \cap L)$ where $L = \mathcal{L}(B_1) \,|||\, \dots \,|||\, \mathcal{L}(B_n)$. Denoting with $\mathcal{A}_{i,M} \times B_i$ the standard cross product construction synchronized on the input symbols (ε transitions can be taken asynchronously), the construction of $\mathcal{R}$ can be modified such that in the simulation mode it tracks the behaviors of $\mathcal{A}_{i,M} \times B_i$ instead of just the thread automaton $\mathcal{A}_{i,M}$. Denote with $\mathcal{R}^T$ the resulting finite state automaton. We observe that in $\mathcal{R}^T$, 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 layered stack automaton. We omit further details on the construction of $\mathcal{R}^T$.

We recall that the number of states of each $\mathcal{A}_{h,M}$ is at most $(k+1)|Q|^3\alpha$ where $\alpha = O(2^{(k|Q|)^2})$ is the number of different ℓ-layered stack automata with $\ell \leq k$. Thus the number of states of $\mathcal{R}^T$ is at most $(n+1)|Q|(k+1)^n|Q|^{3n}\alpha^n\beta^n$ where β is the maximum over the number of states of $B_1, \dots, B_n$. Thus, the number of states of $\mathcal{R}^T$ is exponential in n, $|Q|$ and k. Since we can explore on-the-fly the state space of $\mathcal{R}^T$, we can check the emptiness of $\mathcal{R}^T$ in polynomial space, and in time exponential in n, $|Q|^2$ and k^2. Since each instance of the location reachability is also an instance of the general reachability problem, by Theorems 15 and 27 we get:

Theorem 28. *The reachability problem for* SMPDS *is* PSPACE*-complete, and hardness can be shown both with respect to the number of stacks and the bound k.*

7. Conclusion

We have introduced a decidable restriction of multistack pushdown systems that allows for unboundedly many context switches. Compared to the bounded context-switching analysis, by bounding the scope of the matching relations of push

and pop transitions we can explore a (significantly) larger number of computations of a given MPDS and possibly with a smaller value of the bounding parameter. For example, a run that alternates pushes of two stacks is 1-scoped while it can have any number of contexts. In general, 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).

The main limitation introduced by restricting to bounded scope computations is to bound the amount of information that can flow out of a stack configuration into the other stacks. This makes this notion in some sense *orthogonal* to bounding the number of phases and allowing pop transitions only from the least indexed non-empty stack. In fact, in bounded phase computations we can transfer an unbounded amount of information from a stack to the others but only for a bounded number of times. In the ordered computations instead, unbounded information can only be transferred from the least indexed non-empty stack to stacks of higher index.

We have shown detailed comparisons with the other restrictions introduced in the literature for the analysis of MPDS, and studied the computational complexity of the location and configuration reachability problems. In particular, the problems turn out to be both PSPACE-complete and our decision procedures are exponential in the number of control states, the number of stacks and the bound on the scope.

We remark that the original notion of bounded scope runs introduced in [9] was based on the notion of round. We recall that for an n-stack MPDS M, a *round* of M is the concatenation of n contexts where stack h is the active stack of the h-th concatenated context. Thus, a run is *k-round scoped*[2] if the pop transitions are allowed only when the symbol at the top of the stack was pushed within the last k rounds. It is simple to see that the k-scoped restriction used here is a relaxation of the k-round scoped one. In fact, in k-round scoped runs the number of contexts occurring between a push and its matching pop is always bounded while in k-scoped runs it can be unbounded. Moreover, each k-round scoped run is clearly also a k-scoped run. As an example, consider the MPDS M_2 of Example 2. As observed in Example 2 all the runs of M_2 are all 2-scoped. Instead, for each bound $k > 0$, there are runs of M_2 that are not k-round scoped. In fact, configuration $\langle\langle q_4, \varepsilon, \varepsilon, c^k\rangle\rangle$ can be reached only by allowing to pop a in round $k+1$.

As future 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.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank the anonymous reviewers for their useful and helpful comments on our manuscript. This work was partially supported by GNCS 2019 grant (project “Metodi formali per tecniche di verifica combinata”), and FARB grants ORSA179492 and ORSA188702.

References

[1] S. Qadeer, J. Rehof, Context-bounded model checking of concurrent software, in: N. Halbwachs, L.D. Zuck (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings, in: Lecture Notes in Computer Science, Springer, 2005, pp. 93–107.

[2] M. Musuvathi, S. Qadeer, Iterative context bounding for systematic testing of multithreaded programs, in: J. Ferrante, K.S. McKinley (Eds.), Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, San Diego, California, USA, June 10-13, ACM, 2007, pp. 446–455.

[3] A. Lal, T. Touili, N. Kidd, T.W. Reps, Interprocedural analysis of concurrent programs under a context bound, in: Ramakrishnan and Rehof [46], pp. 282–298, https://doi.org/10.1007/978-3-540-78800-3_20.

[4] S. La Torre, P. Madhusudan, G. Parlato, Model-checking parameterized concurrent programs using linear interfaces, in: Touili et al. [47], pp. 629–644, https://doi.org/10.1007/978-3-642-14295-6_54.

[5] S. Schwoon, Model-checking pushdown systems, Ph.D. thesis, Technische Universität München, 2002.

[6] S. La Torre, P. Madhusudan, G. Parlato, A robust class of context-sensitive languages, in: 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings, IEEE Computer Society, 2007, pp. 161–170.

[7] L. Breveglieri, A. Cherubini, C. Citrini, S. Crespi-Reghizzi, Multi-push-down languages and grammars, Int. J. Found. Comput. Sci. 7 (3) (1996) 253–292, https://doi.org/10.1142/S0129054196000191.

[8] S. La Torre, M. Napoli, G. Parlato, A unifying approach for multistack pushdown automata, in: E. Csuhaj-Varjú, M. Dietzfelbinger, Z. Ésik (Eds.), Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part i, in: Lecture Notes in Computer Science, vol. 8634, Springer, 2014, pp. 377–389.

[2] Note that in [9] this is referred to as k-scoped.

[9] S. La Torre, M. Napoli, Reachability of multistack pushdown systems with scope-bounded matching relations, in: J. Katoen, B. König (Eds.), CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings, in: Lecture Notes in Computer Science, vol. 6901, Springer, 2011, pp. 203–218.
[10] S. La Torre, M. Napoli, A temporal logic for multi-threaded programs, in: J.C.M. Baeten, T. Ball, F.S. de Boer (Eds.), Theoretical Computer Science - 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings, in: Lecture Notes in Computer Science, vol. 7604, Springer, 2012, pp. 225–239.
[11] S. La Torre, G. Parlato, Scope-bounded multistack pushdown systems: fixed-point, sequentialization, and tree-width, in: D. D'Souza, T. Kavitha, J. Radhakrishnan (Eds.), IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India, in: LIPIcs, vol. 18, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 173–184.
[12] S. La Torre, P. Madhusudan, G. Parlato, Analyzing recursive programs using a fixed-point calculus, in: M. Hind, A. Diwan (Eds.), Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, Dublin, Ireland, June 15-21, 2009, ACM, 2009, pp. 211–222.
[13] S. La Torre, M. Napoli, G. Parlato, Scope-bounded pushdown languages, Int. J. Found. Comput. Sci. 27 (2) (2016) 215–234, https://doi.org/10.1142/S0129054116400074.
[14] A. Cyriac, P. Gastin, K.N. Kumar, MSO decidability of multi-pushdown systems via split-width, in: M. Koutny, I. Ulidowski (Eds.), CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings, in: Lecture Notes in Computer Science, vol. 7454, Springer, 2012, pp. 547–561.
[15] M.F. Atig, A. Bouajjani, K.N. Kumar, P. Saivasan, Linear-time model-checking for multithreaded programs under scope-bounding, in: S. Chakraborty, M. Mukund (Eds.), Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings, in: Lecture Notes in Computer Science, Springer, 2012, pp. 152–166.
[16] R. Alur, K. Etessami, P. Madhusudan, A temporal logic of nested calls and returns, in: K. Jensen, A. Podelski (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings, in: Lecture Notes in Computer Science, vol. 2988, Springer, 2004, pp. 467–481.
[17] A. Lal, T.W. Reps, Reducing concurrent analysis under a context bound to sequential analysis, in: A. Gupta, S. Malik (Eds.), Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings, in: Lecture Notes in Computer Science, vol. 5123, Springer, 2008, pp. 37–51.
[18] D. Suwimonteerabuth, J. Esparza, S. Schwoon, Symbolic context-bounded analysis of multithreaded java programs, in: K. Havelund, R. Majumdar, J. Palsberg (Eds.), Model Checking Software, 15th International SPIN Workshop, Los Angeles, CA, USA, August 10-12, 2008, Proceedings, in: Lecture Notes in Computer Science, vol. 5156, Springer, 2008, pp. 270–287.
[19] S.K. Lahiri, S. Qadeer, Z. Rakamaric, Static and precise detection of concurrency errors in systems code using SMT solvers, in: Bouajjani and Maler [48], pp. 509–524, https://doi.org/10.1007/978-3-642-02658-4_38.
[20] J. Alglave, D. Kroening, M. Tautschnig, Partial orders for efficient bounded model checking of concurrent software, in: N. Sharygina, H. Veith (Eds.), Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings, in: Lecture Notes in Computer Science, vol. 8044, Springer, 2013, pp. 141–157.
[21] O. Inverso, E. Tomasco, B. Fischer, S. La Torre, G. Parlato, Bounded model checking of multi-threaded C programs via lazy sequentialization, in: A. Biere, R. Bloem (Eds.), Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings, in: Lecture Notes in Computer Science, vol. 8559, Springer, 2014, pp. 585–602.
[22] E. Tomasco, O. Inverso, B. Fischer, S. La Torre, G. Parlato, Verifying concurrent programs by memory unwinding, in: C. Baier, C. Tinelli (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings, in: Lecture Notes in Computer Science, vol. 9035, Springer, 2015, pp. 551–565.
[23] S. La Torre, P. Madhusudan, G. Parlato, Reducing context-bounded concurrent reachability to sequential reachability, in: Bouajjani and Maler [48], pp. 477–492, https://doi.org/10.1007/978-3-642-02658-4_36.
[24] S. La Torre, P. Madhusudan, G. Parlato, Sequentializing parameterized programs, in: S.S. Bauer, J. Raclet (Eds.), Proceedings Fourth Workshop on Foundations of Interface Technologies, FIT 2012, Tallinn, Estonia, 25th March 2012, in: EPTCS, vol. 87, 2012, pp. 34–47.
[25] M.F. Atig, A. Bouajjani, S. Qadeer, Context-bounded analysis for concurrent programs with dynamic creation of threads, Logical Methods in Computer Science 7 (4), https://doi.org/10.2168/LMCS-7(4:4)2011.
[26] M. Emmi, S. Qadeer, Z. Rakamaric, Delay-bounded scheduling, in: Ball and Sagiv [49], pp. 411–422, https://doi.org/10.1145/1926385.1926432.
[27] A. Bouajjani, M. Emmi, G. Parlato, On sequentializing concurrent programs, in: E. Yahav (Ed.), Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings, in: Lecture Notes in Computer Science, vol. 6887, Springer, 2011, pp. 129–145.
[28] A. Bouajjani, S. Fratani, S. Qadeer, Context-bounded analysis of multithreaded programs with dynamic linked structures, in: W. Damm, H. Hermanns (Eds.), Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings, in: Lecture Notes in Computer Science, vol. 4590, Springer, 2007, pp. 207–220.
[29] S. La Torre, P. Madhusudan, G. Parlato, Context-bounded analysis of concurrent queue systems, in: Ramakrishnan and Rehof [46], pp. 299–314, https://doi.org/10.1007/978-3-540-78800-3_21.
[30] P. Chini, J. Kolberg, A. Krebs, R. Meyer, P. Saivasan, On the complexity of bounded context switching, in: K. Pruhs, C. Sohler (Eds.), 25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria, in: LIPIcs, vol. 87, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, pp. 27:1–27:15.
[31] S. La Torre, P. Madhusudan, G. Parlato, An infinite automaton characterization of double exponential time, in: M. Kaminski, S. Martini (Eds.), Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings, in: Lecture Notes in Computer Science, vol. 5213, Springer, 2008, pp. 33–48.
[32] A. Seth, Global reachability in bounded phase multi-stack pushdown systems, in: Touili et al. [47], pp. 615–628, https://doi.org/10.1007/978-3-642-14295-6_53.
[33] B. Bollig, D. Kuske, R. Mennicke, The complexity of model checking multi-stack systems, in: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25–28, 2013, IEEE Computer Society, 2013, pp. 163–172.
[34] K. Bansal, S. Demri, Model-checking bounded multi-pushdown systems, in: A.A. Bulatov, A.M. Shur (Eds.), Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings, in: Lecture Notes in Computer Science, vol. 7913, Springer, 2013, pp. 405–417.
[35] B. Bollig, C. Aiswarya, P. Gastin, M. Zeitoun, Temporal logics for concurrent recursive programs: satisfiability and model checking, J. Appl. Log. 12 (4) (2014) 395–416, https://doi.org/10.1016/j.jal.2014.05.001.
[36] B. Bollig, On the expressive power of 2-stack visibly pushdown automata, Log. Methods Comput. Sci. 4 (4), https://doi.org/10.2168/LMCS-4(4:16)2008.
[37] D. Carotenuto, A. Murano, A. Peron, 2-visibly pushdown automata, in: T. Harju, J. Karhumäki, A. Lepistö (Eds.), Developments in Language Theory, 11th International Conference, DLT 2007, Turku, Finland, July 3-6, 2007, Proceedings, in: Lecture Notes in Computer Science, Springer, 2007, pp. 132–144.

[38] P.A. Abdulla, M.F. Atig, O. Rezine, J. Stenman, Budget-bounded model-checking pushdown systems, Form. Methods Syst. Des. 45 (2) (2014) 273–301, https://doi.org/10.1007/s10703-014-0207-y.
[39] P. Madhusudan, G. Parlato, X. Qiu, Decidable logics combining heap structures and data, in: Ball and Sagiv [49], pp. 611–622, https://doi.org/10.1145/1926385.1926455.
[40] M. Hague, Saturation of concurrent collapsible pushdown systems, in: A. Seth, N.K. Vishnoi (Eds.), IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013, December 12-14, 2013, Guwahati, India, in: LIPIcs, vol. 24, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013, pp. 313–325.
[41] A. Seth, Games on multi-stack pushdown systems, in: S.N. Artëmov, A. Nerode (Eds.), Logical Foundations of Computer Science, International Symposium, LFCS 2009, Deerfield Beach, FL, USA, January 3-6, 2009. Proceedings, in: Lecture Notes in Computer Science, vol. 5407, Springer, 2009, pp. 395–408.
[42] R. Meyer, S. Muskalla, G. Zetzsche, Bounded context switching for valence systems, in: S. Schewe, L. Zhang (Eds.), 29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China, in: LIPIcs, vol. 118, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 12:1–12:18.
[43] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.
[44] M.F. Atig, K.N. Kumar, P. Saivasan, Adjacent ordered multi-pushdown systems, Int. J. Found. Comput. Sci. 25 (8) (2014) 1083–1096, https://doi.org/10.1142/S0129054114400255.
[45] P.A. Abdulla, S. Aronis, M.F. Atig, B. Jonsson, C. Leonardsson, K. Sagonas, Stateless model checking for TSO and PSO, Acta Inform. 54 (8) (2017) 789–818, https://doi.org/10.1007/s00236-016-0275-0.
[46] C.R. Ramakrishnan, J. Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings, Lecture Notes in Computer Science, vol. 4963, Springer, 2008.
[47] T. Touili, B. Cook, P.B. Jackson (Eds.), Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6174, Springer, 2010.
[48] A. Bouajjani, O. Maler (Eds.), 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings, Lecture Notes in Computer Science, vol. 5643, Springer, 2009.
[49] T. Ball, M. Sagiv (Eds.), Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011, ACM, 2011, http://dl.acm.org/citation.cfm?id=1926385.