International Journal of Foundations of Computer Science
Vol. 25, No. 8 (2014) 1083–1096
© World Scientific Publishing Company
DOI: 10.1142/S0129054114400255

ADJACENT ORDERED MULTI-PUSHDOWN SYSTEMS

MOHAMED FAOUZI ATIG
Uppsala University, Sweden
mohamed_faouzi.atig@it.uu.se

K. NARAYAN KUMAR* and PRAKASH SAIVASAN†
Chennai Mathematical Institute, India
**kumar@cmi.ac.in*
†*saivasan@cmi.ac.in*

Received 15 December 2013
Accepted 24 June 2014
Communicated by Marie-Pierre Béal and Olivier Carton

Multi-pushdown systems are formal models of multi-threaded programs. As they are Turing powerful in their full generality, several decidable subclasses, constituting under-approximations of the original system, have been studied in the recent years. Ordered Multi-Pushdown Systems (OMPDSs) impose an order on the stacks and limit pop actions to the lowest non-empty stack. The control state reachability for OMPDSs is 2-ETIME-COMPLETE. We propose a restriction on OMPDSs, called Adjacent OMPDSs (AOMPDS), where values may be pushed only on the lowest non-empty stack or one of its two neighbours. We describe EXPTIME decision procedures for reachability and LTL model-checking and establish matching lower bounds. We demonstrate the utility of this model as an algorithmic tool via optimal reductions from other models.

Keywords: Concurrent pushdown systems; reachability problem; LTL model-checking; recursive Boolean programs.

1. Introduction

Verification of concurrent recursive programs is an important but difficult problem well studied over the last decade. The theory of pushdown systems has been used very effectively in analysing sequential recursive programs and forms the backbone of a number of verification tools. However, there is no such well established classical theory that underlies concurrent recursive programs. In the case where the number of threads is bounded, it is natural to consider the generalization of pushdown systems to multi-pushdown systems (MPDS), with one pushdown per thread (to model its call stack) and control states to model the contents of the shared memory.

In their full generality MPDSs are not analyzable as two stacks can simulate the tape of a turing machine. The main focus in this area has hence been to identify decidable subclasses. These subclasses are obtained by placing restrictions on the behaviours of MPDSs. Qadeer and Rehof [17] showed that if the number of switches from accessing one stack to another is bounded a priori then the control state reachability problem is decidable. Such behaviours are called *bounded context* behaviours. Subsequently, this idea of context-bounded analysis has been used and extended to a number of related models very effectively [7, 8, 11–13, 20, 22].

Such a restriction can be seen as a collection of runs of an unrestricted system, constituting an underapproximation of the real system. In the context of linear time model checking, where the aim is to look for a faulty run, verifying such an underapproximation guarantees that there are no faulty runs of the restricted form or identifies the possibility of a faulty run conforming to these restrictions. Several generalizations of the idea of context bounding have been proposed, notably

- *Bounded Phase MPDSs* [18, 19, 21]: A *phase* denotes a segment of an execution where all the pop operations are performed on a single stack. An a priori bound is placed on the number of phases.
- *Ordered Multi-pushdowns* (*OMPDSs*) [2, 9]: The stacks are numbered $1, 2, \ldots, n$ and pop moves are permitted only on the lowest non-empty stack.
- *Scope Bounded MPDSs* [23]: A bound is placed on the number of context switches between any push and its corresponding pop (the pop that removes this value from the stack).

The control state reachability problem is decidable for all these models, NP-Complete for bounded context systems [17] and Pspace-Complete [23] for scope bounded systems. However, for bounded-phase MPDSs and OMPDSs the problem is 2-Etime-Complete [2, 21]. It is interesting to note that these two models allow copying of a stack onto another while the first two do not. Furthermore, OMPDSs can simulate bounded-phase MPDSs [2].

The run of an OMPDS can be thought as consisting of a sequence of phases, where each phase identifies an active stack, the stack from which values may be popped. Further, during a phase associated with stack i, all stacks j with $j < i$ must be empty. We impose an additional requirement that values can be pushed only on stacks $i-1$, i or $i+1$ during such a phase to obtain the subclass of *Adjacent OMPDSs* (AOMPDSs). For this class we show that the control state reachability problem can be solved in Exptime and prove a matching lower bound. Observe, that this class has the ability to copy a stack onto another, and to our knowledge is the first such class in Exptime.

The *repeated reachability problem* has to do with infinite runs and the aim is to determine if there is any run that visits a particular set of states F infinitely often. In formal verification, this problem is of great importance as a solution immediately leads to a model checking algorithm for linear time temporal logic (LTL). We obtain an Exptime decision procedure for the repeated reachability problem (and LTL model-checking problem) for AOMPDSs.

We believe that this is the only known class of MPDSs with the repeated reachability problem in Exptime that allows the copying of stack contents while admitting infinite runs that change their active stack infinitely often.

We illustrate the power of AOMPDSs using three applications — first, we show that the class of unary OMPDSs, i.e., OMPDSs where the stack alphabet is singleton, (which can be thought of as counter systems with restrictions on decrement operations) can be reduced to AOMPDAs and hence obtain EXPTIME procedures for reachability, repeated reachability and LTL model-checking for this class. This is likely to be optimal since we also show NP-HARDNESS. Secondly, we show that the control state reachability problem for networks of recursive programs communicating via queues, whose connection topology is a directed forest, can be reduced to reachability in AOMPDS with polynomial blowup, obtaining an EXPTIME solution to this problem. This problem was proposed and solved in [21] where it is also shown that direct forest topology is the only one for which this problem is decidable. Finally, we show that the control state reachability problem for bounded-phase MPDSs can be reduced to that for AOMPDSs (with an exponential blow-up) through very simple reduction.

Other Related Work: In [15] Madhusudan and Parlato study the structure of runs of MPDSs as graphs. They argue that decidability is closely related to the boundedness of the tree-width of such graphs. The runs arising from all the restrictions of MPDSs described above have bounded tree-width. In [10] a new measure of complexity for multiply nested words called split-width is proposed, which is bounded whenever tree-width is, but seems to yield easier proofs of boundedness.

[20] applies the decidability of bounded-phase MPDSs to the analysis of recursive concurrent programs communicating via queues and obtains a characterization of the decidable topologies. These results have been extended in [11] by placing further restrictions on how messages are consumed from the queue.

In the setting of infinite runs, [1] describes a 2-EXPTIME decision procedure for the repeated reachability and LTL model-checking problems for OMPDSs and bounded-phase MPDSs (also see [6]). Recently, [24] and [4] show that LTL model checking for bounded scope systems can be solved in EXPTIME. In this context, it must be noted that with the bounded context and bounded-phase restrictions every run has to eventually use only a single stack and hence do not form natural restrictions on infinite runs. [3] investigates the existence of ultimately periodic infinite runs, i.e. runs of the form uv^ω, in MPDSs.

An extended abstract of this paper appeared as [5].

2. Preliminaries

Notation: Let $\mathbb{N}$ denote the set of non-negative integers. For every $i, j \in \mathbb{N}$ such that $i \leq j$, we use $[i..j]$ to denote the set $\{k \in \mathbb{N} \,|\, i \leq k \leq j\}$. Let Σ be a finite alphabet. We denote by Σ^* (Σ^ω) the set of all finite (resp. infinite) words over Σ, and by ϵ the empty word. We use Σ_ϵ to denote $\Sigma \cup \{\epsilon\}$. Let u be a word over Σ. The length of u is denoted by $|u|$ and $|\epsilon| = 0$. For every $j \in [1..|u|]$, we use $u(j)$ to denote the j^{th} letter of u.

Context-Free Grammars: A *context-free grammar* (CFG) G is a tuple $(\mathcal{X}, \Sigma, P)$ where $\mathcal{X}$ is a finite non-empty set of *variables* (or *nonterminals*), Σ is an alphabet of *terminals*, and $P \subseteq \big(\mathcal{X} \times (\mathcal{X}^2 \cup \Sigma \cup \{\epsilon\})\big)$ a finite set of *productions* and denote the production (X, w) by $X \Rightarrow_G w$. We also write $\Rightarrow$ instead of $\Rightarrow_G$ when the identity of G is clear from the context. Given strings $u, v \in (\Sigma \cup \mathcal{X})^*$ we say $u \Rightarrow_G v$ if there exists a production $(X, w) \in R$ and some words $y, z \in (\Sigma \cup \mathcal{X})^*$ such that $u = yXz$ and $v = ywz$. We use $\Rightarrow_G^*$ for the reflexive transitive closure of $\Rightarrow_G$. For every nonterminal symbol $X \in \mathcal{X}$, we define the context-free language generated from X by $L_G(X) = \{w \in \Sigma^* \mid X \Rightarrow_G^* w\}$.

3. Multi-Pushdown Systems

In this section, we will formally introduce *multi-pushdown systems* and then define two subclasses, namely *ordered multi-pushdown systems* and *adjacent ordered multi-pushdown systems.* A multi-pushdown system has $n \geq 1$ read-write memory tapes (stacks) with a last-in-first-out rewriting policy and optionally a read only input tape. We do not require an input tape as the purpose of this paper is to prove the decidability of the reachability problem.

Definition 1 (MPDS). *A* Multi-PushDown System (*MPDS*) *is a tuple* $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ *where:* (1) $n \geq 1$ *is the number of stacks,* (2) Q *is the non-empty set of states,* (3) Γ *is the finite set of stack symbols containing the special stack symbol* $\perp$, (4) $q_0 \in Q$ *is the initial state,* (5) $\gamma_0 \in \Gamma \setminus \{\perp\}$ *is the initial stack symbol, and* (6) $\Delta \subseteq ((Q \times (\Gamma_\epsilon)^n) \times (Q \times (\Gamma^*)^n))$ *is the transition relation such that if* $((q, \gamma_1, \gamma_2, \ldots, \gamma_n), (q', \alpha_1, \alpha_2, \ldots, \alpha_n))$ *is in* Δ, *then for all* $i \in [1..n]$, *we have:* $|\alpha_i| \leq 2$, *if* $\gamma_i \neq \perp$ *then* $\alpha_i \in (\Gamma \setminus \{\perp\})^*$ *and* $\alpha_i \in ((\Gamma \setminus \{\perp\})^* \cdot \{\perp\})$ *otherwise.*

A stack content of M is an element of $\mathit{Stack}(M) = (\Gamma \setminus \{\perp\})^* \{\perp\}$. A configuration of the MPDS M is a $(n+1)$ tuple $(q, w_1, w_2, \cdots, w_n)$ with $q \in Q$, and $w_1, w_2, \ldots, w_n \in \mathit{Stack}(M)$. The set of configurations of the MPDS M is denoted by $\mathcal{C}(M)$. The *initial configuration* c_M^{init} of the MPDS M is $(q_0, \perp, \ldots, \perp, \gamma_0\perp)$.

If $t = ((q, \gamma_1, \ldots, \gamma_n), (q', \alpha_1, \ldots, \alpha_n))$ is an element of Δ, then $(q, \gamma_1 w_1, \ldots, \gamma_n w_n) \xrightarrow{t}_M (q', \alpha_1 w_1, \ldots, \alpha_n w_n)$ for all $w_1, \ldots, w_n \in \Gamma^*$ such that $\gamma_1 w_1, \ldots, \gamma_n w_n \in \mathit{Stack}(M)$. We define the transition relation $\rightarrow_M$ as $\bigcup_{t \in \Delta} \xrightarrow{t}_M$. Observe that the stack symbol $\perp$ marks the bottom of the stack and our transition relation does not allow this $\perp$ to be popped. We write $\rightarrow_M^*$ to denote the reflexive and transitive closure of the relation$\rightarrow_M$, representing runs of the system. For every sequence of transitions $\rho = t_1 t_2 \ldots t_m \in \Delta^*$ and two configurations $c, c' \in \mathcal{C}(M)$, we write $c \xrightarrow{\rho}{}_M^* c'$ to denote that one of the following two cases holds: (1) $\rho = \epsilon$ and $c = c'$, or (2) there are configurations $c_0, \cdots, c_m \in \mathcal{C}(M)$ such that $c_0 = c$, $c' = c_m$, and $c_i \xrightarrow{t_{i+1}}_M c_{i+1}$ for all $i \in [0..m-1]$.

An ordered multi-pushdown system is a multi-pushdown system in which one can pop only from the first non-empty stack.

Definition 2 (OMPDS). *An* Ordered *MPDS* (*OMPDS*) *is a MPDS* $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ *where for each transition* $((q, \gamma_1, \ldots, \gamma_n), (q', \alpha_1, \ldots, \alpha_n)) \in \Delta$ *there is an index* $i \in [1..n]$ *such that* $\gamma_1 = \cdots = \gamma_{i-1} = \bot$, $\gamma_i \in (\Gamma \setminus \{\bot\})$, *and* $\gamma_{i+1} = \cdots = \gamma_n = \epsilon$ *and further one of the following properties holds:* (1) Operate on the stack i: $\alpha_j = \bot$ *for all* $j < i$ *and* $\alpha_j = \epsilon$ *for all* $j > i$, (2) Push on the stack $j < i$: $\alpha_i = \epsilon$, $\alpha_k = \bot$ *if* $j \neq k < i$, $\alpha_j \in (\Gamma \cdot \{\bot\})$, *and* $\alpha_k = \epsilon$ *if* $k > i$, *or* (3) Push on the stack $j > i$: $\alpha_i = \epsilon$, $\alpha_k = \bot$ *if* $k < i$, $\alpha_k = \epsilon$ *if* $j \neq k > i$ *and* $|\alpha_j| = 1$. *If we further restrict the choice of* j *in item* 2 *above to be only* $i - 1$ *and in item* 3 *to be* $i + 1$ *we get the subclass of* Adjacent OMPDSs (*AOMPDSs*).

Observe that in any AOMPDS, a transition that pops a symbol from stack i is only allowed to push values on one of the stacks $i-1, i$ or $i+1$ depending on whether transition arises from items 1, 2 or 3 above. We write $\Delta_{i,i-1}, \Delta_{i,i}$ and $\Delta_{i,i+1}$ for these transitions respectively.

Definition 3 (Reachability Problem). *Given a MPDS* M *and a state* q *of* M, *the* reachability problem *is to decide whether* $(q_0, \bot, \cdots, \bot, \gamma_0\bot) \rightarrow_M^* (q, \bot, \cdots, \bot)$.

4. The Reachability Problem for AOMPDS

The purpose of this section is to prove the following theorem:

Theorem 4. *The reachability problem for Adjacent Ordered Multi-Pushdown System is* EXPTIME-COMPLETE.

Upper Bound: Let $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ be an AOMPDS with $n > 1$ (the case where $n = 1$ boils down to the reachability of pushdown systems which is well-known to be in PTIME). The proof of EXPTIME-containment is through an inductive construction that reduces the reachability problem for M to the reachability problem for a pushdown system with only an exponential blow up in size. The key step is to show that we can reduce the reachability problem for M to the reachability problem for an $(n-1)$-AOMPDS. The feature of our reduction is that there is no blowup in the state space and the size of the stack alphabet increases quadratically in the number of states. A non-linear blow up in the number of states will result in a complexity higher than EXPTIME.

We plan to use a single stack to simulate both the first and second stacks of M. It is useful to consider runs of M to understand how this works. Any run ρ of M starting at the initial configuration naturally breaks up into segments $\sigma_0\rho_1\sigma_1 \ldots \rho_k\sigma_k$ where the segments ρ_i contain configurations where stack 1 is non-empty while in any configuration in the σ_i's stack 1 is empty. Clearly the content of stack 1 at the beginning of ρ_i contains exactly two symbols, and we assume it to be $a_i\bot$. We further assume that ρ_i begins at control state q_i and the segment σ_i in state q_i'. What is the contribution of the segment ρ_i, which is essentially the run of a pushdown automaton starting and ending at the empty stack configuration, to this run?

Firstly, it transforms the local state from q_i to q'_i. Secondly, a word w_i is pushed on to stack 2 during this segment. It also, consumes the value a_i from stack 1 in this process, but that is not relevant to the rest of the computation. To simulate the effect of ρ_i it would thus suffice to jump from state q_i to q'_i and push the word w_i on stack 2. There are potentially infinitely many possible runs of the form ρ_i that go from q_i to q'_i while removing a_i from stack 1 and thus infinite possibilities for the word that is pushed on stack 2. However, it is easy to see that this set of words $L(q_i, a_i, q'_i)$ is a CFL. If the language $L(q_i, a_i, q'_i)$ is a regular language, we could simply *summarize* this run by depositing a word from this language on stack 2 and then proceed with the simulation of stack 2. However, since it is only a CFL this is not possible. Instead, we have to interleave the simulation of stack 2 with the simulation of stack 1, using stack 2, and there is no a priori bound on the number of switches between the stacks in such a simulation. For OMPDSs such a simulation would not work as the values pushed by the segments of executions of stack 1 and stack 2 on a third stack would get permuted and this explains why OMPDS needs a different, more expensive decision procedure.

To simulate the effect of ρ_i, we jump directly to q'_i and push a non-terminal symbol (from the appropriate CFG) that generates the language $L(q_i, a_i, q'_i)^R$ (reverse, because stacks are last in first out) on stack 2. Now, when we try to simulate σ'_i, we might encounter a nonterminal on top of stack 2 instead of a terminal symbol belonging to stack 2. In this case, we rewrite the nonterminal using one of the rules of the CFG applicable to this nonterminal. In effect, we produce a left-most derivation of a word from $L(q_i, a_i, q'_i)$ in a lazy manner, interspersed within the execution involving stack 2, generating terminals only when they need to be consumed. This is the main idea in the construction that is formalized below.

We define $\Delta_1 = \Delta_{(1,1)} \cup \Delta_{(1,2)}$, $\Delta_i = \Delta_{(i,i)} \cup \Delta_{(i,i+1)} \cup \Delta_{(i,i-1)}$ for all $2 \leq i < n$, and $\Delta_n = \Delta_{(n,n)} \cup \Delta_{(n,n-1)}$.

We construct a context-free grammar $G_M = (NT, (\Gamma \setminus \{\perp\}), P)$ from the AOMPDA M. The set of non-terminals NT is $(Q \times (\Gamma \setminus \{\perp\}) \times Q)$. The set of productions P is defined as the smallest set of rules satisfying: (1) For every two states $p, p' \in Q$, and every transition $((q, \gamma, \epsilon, \ldots, \epsilon), (q', \gamma_1\gamma_2, \epsilon, \ldots, \epsilon))$ in Δ such that $\gamma, \gamma_1, \gamma_2 \in (\Gamma \setminus \{\perp\})$, we have $(q, \gamma, p) \Rightarrow_{G_M} (q', \gamma_1, p')(p', \gamma_2, p)$, (2) For every state $p \in Q$, and every transition $((q, \gamma, \epsilon, \ldots, \epsilon), (q', \gamma', \epsilon, \ldots, \epsilon))$ in Δ such that $\gamma, \gamma' \in (\Gamma \setminus \{\perp\})$, we have $(q, \gamma, p) \Rightarrow_{G_M} (q', \gamma', p)$, (3) For every transition $((q, \gamma, \epsilon, \ldots, \epsilon), (q', \epsilon, \epsilon, \ldots, \epsilon))$ in Δ such that $\gamma \in (\Gamma \setminus \{\perp\})$, we have $(q, \gamma, q') \Rightarrow_{G_M} \epsilon$, and (4) For every transition $((q, \gamma, \epsilon, \ldots, \epsilon), (q', \epsilon, \gamma', \epsilon, \ldots, \epsilon))$ in Δ such that $\gamma, \gamma' \in (\Gamma \setminus \{\perp\})$, we have $(q, \gamma, q') \Rightarrow_{G_M} \gamma'$.

Then, it is easy to see that the context-free grammar summarizes the effect of the first stack on the second one. Formally, we have:

Lemma 5. *The context free language $L_{G_M}((q, \gamma, q'))$ is equal to the set of words $\{w^R \in (\Gamma \setminus \{\perp\})^* \mid \exists \rho \in \Delta_1^*.\ (q, \gamma\perp, w_2, \ldots, w_n) \xrightarrow{\rho}_M (q', \perp, w \cdot w_2, \ldots, w_n)\}$ where w^R denotes the reverse of the word w.*

We are now ready to show that reachability problems on M can be reduced to reachability problems on an $(n-1)$-AOMPA N. Furthermore, the number of states of N is linear in $|Q|$, the size of the stack alphabet of N is $O(|Q|^2|\Gamma|)$ and the number of transitions is $O(|Q|^3.|\Delta|)$. The upper-bound claimed in Theorem 4 then follows by a simple induction.

Let $F \subseteq Q$ be the set of states whose reachability we are interested in, we show how to construct an $(n-1)$-AOMPA N such that the reachability question on M can be reduced to reachability question on N. Formally, N is defined by the tuple $(n-1, Q, \Gamma \cup NT, \Delta', q_0, \gamma_0)$ where Δ' is defined as the smallest set satisfying the following conditions:

(1) For any transition $((q, \bot, \gamma_2, \ldots, \gamma_n), (q', \bot, \alpha_2, \ldots, \alpha_n)) \in \Delta$, we have $((q, \gamma_2, \ldots, \gamma_n), (q', \alpha_2, \ldots, \alpha_n)) \in \Delta'$.
(2) For any transition $((q, \bot, \gamma_2, \epsilon, \ldots, \epsilon), (q', \gamma\bot, \epsilon, \ldots, \epsilon)) \in \Delta_{(2,1)}$, we have $((q, \gamma_2, \epsilon, \ldots, \epsilon), (q'', (q', \gamma, q''), \epsilon, \ldots, \epsilon)) \in \Delta'$ for all $q'' \in Q$.
(3) For any production rule $X \Rightarrow_{G_M} w$ and state $q \in Q$, we have $((q, X, \epsilon, \ldots, \epsilon), (q, w^R, \epsilon, \ldots, \epsilon)) \in \Delta'$.

Lemma 6. *The set of states F is reachable in M iff F is reachable in N.*

The fact that even a single contiguous segment of moves using stack 1 in M may now be interleaved arbitrarily with the executions involving other stacks in N, makes proof somewhat involved. Towards the proof, we define a relation between the configurations of N and M systems. For any configuration $c \in \mathcal{C}(M)$ and $d \in \mathcal{C}(N)$, we say cRd iff one of the following is true: (1) d is of the form $(q, \bot, w_3, \cdots, w_n)$ and c is of the form $(q, \bot, \bot, w_3, \cdots, w_n)$, or (2) d is of the form $(q, \eta_1 v_1 \eta_2 v_2 \cdots \eta_m v_m \bot, w_3, \cdots, w_n)$ and c is of the form $(q, \bot, u_1 v_1 u_2 v_2 \cdots u_m v_m \bot, w_3, \cdots, w_n)$ where $v_1, u_1, v_2, u_2, \ldots, v_m, u_m \in (\Gamma \setminus \{\bot\})^*$, $\eta_1, \eta_2, \ldots, \eta_m \in NT^*$ and $\eta_k \Rightarrow^*_{G_M} u_k^R$ for all $k \in [1..m]$.

Thus, cRd verifies that it is possible to replace the nonterminals appearing in stack 2 in d by words they derive (and by tagging an additional empty stack for the missing stack 1) to obtain c. We now show that this relation faithfully transports runs (to configurations from the initial configuration) in both directions. This is the import of Lemmas 7 and 8, which together guarantee that the state reachability in M reduces to state reachability in N.

Lemma 7. *Let $c_1, c_2 \in (Q \times \{\bot\} \times (Stack(M))^{n-1})$ be two configurations such that $c_M^{init} \to^*_M c_1$ and $c_M^{init} \to^*_M c_2$. If $c_1 \xrightarrow{\rho}_M c_2$, with $\rho \in \cup_{i=3}^n \Delta_i \cup (\Delta_{(2,1)}\Delta_1^*) \cup \Delta_{(2,2)} \cup \Delta_{(2,3)}$, then for every configuration $d_1 \in \mathcal{C}(N)$ such that $c_1 R d_1$, there is a configuration $d_2 \in \mathcal{C}(N)$ such that $c_2 R d_2$ and $d_1 \to^*_N d_2$.*

Lemma 8. *Let $d_1, d_2 \in \mathcal{C}(N)$ be two configurations of N such that $c_N^{init} \to^*_N d_1 \xrightarrow{t}_N d_2$ for some $t \in \Delta'$. Then for every configuration $c_2 \in \mathcal{C}(M)$ such that $c_2 R d_2$, there is a configuration $c_1 \in \mathcal{C}(M)$ such that $c_1 R d_1$ and $c_1 \to^*_M c_2$.*

The proofs of these lemmas follow from simple but tedious case analysis and the details are omitted for space constraints. Observe that these lemmas are asymmetric. This is due to the fact that Lemma 8 may not hold in the forward direction (since N may choose different replacement of nonterminals than the one specified by c_1).

Lower Bound: It is known that the following problem is EXPTIME-complete [11]: Given a pushdown automaton $\mathcal{P}$ recognizing a CFL L, and $(n-1)$ finite state automata $\mathcal{A}_2, \ldots, \mathcal{A}_n$ recognizing the regular languages $L_2, \ldots, L_n$ respectively, is $L \cap \bigcap_{i=2}^{n} L_i$ non-empty? We can show that this problem can be reduced, in polynomial time, to the reachability problem for an AOMPDS M with n-stacks. The idea is the following: The first stack is used to simulate $\mathcal{P}$ and write down a word that is accepted to the second stack. Each other stack is then used to check acceptance by one of the finite automata, whilst transferring the contents to the following stack.

5. Repeated Reachability for AOMPDS

In this section, we show that the linear-time model checking problem is EXPTIME-COMPLETE for AOMPDS. In the following, we assume that the reader is familiar with ω-regular properties expressed in the linear-time temporal logics [16] or the linear time μ-calculus [25]. For more details, the reader is referred to [16, 25, 26]. Checking whether a MPDS satisfies a property expressed in such a logic reduces to solving the *repeated state reachability* problem, i.e., checking if there is an infinite run that visits control states from a given set F infinitely often.

We use the following theorem to reduce the repeated state reachability problem for OMPDSs to the reachability problem for OMPDSs. We write $\overrightarrow{\perp}^{i}$ to mean a sequence $\perp, \perp, \ldots, \perp$ of length i.

Theorem 9 ([1]). *Let $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ be an OMPDS and q_f be a state of M. There is an infinite run starting from c_M^{init} that visits infinitely often the state q_f if and only if there are $i \in [1..n]$, $q \in Q$, and $\gamma \in \Gamma \setminus \{\perp\}$ such that:*

- $c_M^{init} \rightarrow_M^{*} (q, \overrightarrow{\perp}^{i-1}, \gamma w, w_{i+1}, \ldots, w_n)$ *for some* $w, w_{i+1}, \ldots, w_n \in \Gamma^*$.
- $(q, \overrightarrow{\perp}^{i-1}, \gamma\perp, \overrightarrow{\perp}^{n-i}) \xrightarrow{\rho_1}_M (q_f, w_1, \ldots, w_n) \xrightarrow{\rho_2}_M (q, \overrightarrow{\perp}^{i-1}, \gamma w_i', w_{i+1}', \ldots, w_n')$ *for some* $w_1, \ldots, w_n, w_i', \ldots, w_n' \in \Gamma^*$, $\rho_1 \in \Delta'^*$ *and* $\rho_2 \in \Delta'^+$ *where* Δ' *contains all the transitions of the form* $((q, \overrightarrow{\perp}^{j-1}, \gamma_j, \epsilon, \ldots, \epsilon), (q, \alpha_1, \ldots, \alpha_n)) \in \Delta$ *such that* $1 \leq j \leq i$ *and* $\gamma_j \in (\Gamma \setminus \{\perp\})$.

It is possible to formulate each of the two items in Theorem 9 as simple reachability queries on two AOMPDSs whose sizes are polynomial in the size of M.

Theorem 10. *Let $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ be an AOMPDS and q_f be a state of M. Then checking whether there is an infinite run starting from c_M^{init} that visits the state q_f infinitely often can be solved in time $O(|M|)^{poly(n)}$.*

As an immediate consequence of Theorem 10 we have the following corollary.

Theorem 11. *The model-checking problem for AOMPDS $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ w.r.t. formulas in LTL and linear μ-calculus, i.e. checking if there is an infinite run starting from c_M^{init} satisfying the formula, is decidable in time $O(|M|^{poly(n,m)})$ (where m is the size of the formula).*

6. Applications of AOMPDSs

6.1. *Unary ordered multi-pushdown systems*

The class of Unary Ordered Multi-Pushdown Systems is the subclass of OMPDS where the stack alphabet contains just one letter other than $\perp$ (i.e., $|\Gamma| = 2$).

Definition 12 (UOMPDS). *An* Unary OMPDS (*UOMPDS*) *is an OMPDS $(n, Q, \Gamma, \Delta, q_0, \gamma_0)$ such that $|\Gamma| = 2$.*

We prove that the reachability problem for UOMPDS is in EXPTIME by reducing it to the reachability problem for AOMPDS. The key observation is that in a unary OMPDA the order in which elements are pushed on a stack is not important. So, given an UOMPDS M with n-stacks and an alphabet $\Gamma = \{a, \perp\}$ we construct an AOMPDS N with n-stacks and alphabet $\Gamma' = \{(a, 1), (a, 2), \ldots, (a, n)\} \cup \{\perp\}$.

The control states of N are precisely the control states of M. The occurrence of the letter (a, i) in any stack in N denotes the occurrence of an a on stack i, so that, counting the number of occurrences of (a, i)'s across all the stacks in a configuration of N gives the contents of stack i in the corresponding configuration in M.

If the top element of the left-most non-empty stack i is (a, i) then N simulates a corresponding move of M. If this move involves a pushing α on stack i, it is simulated by pushing $\pi_i(\alpha)$ on stack i where π_i is the function with $\pi_i(a) = (a, i)$, $\pi_i(\perp) = \perp$ and extended homomorphically to $\{a, \perp\}^*$. If it involves a pushing α on stack j, $j > i$ (respectively $j < i$) then $\pi_j(\alpha)$ is pushed on stack $i + 1$ (respectively $i - 1$). If the top of the left-most nonempty stack i is (a, j) with $j < i$ (respectively $j > i$) then the value is simply copied to stack $i - 1$ (respectively $i + 1$). This gives us the following theorem.

Theorem 13. *The reachability, repeated reachability and LTL model-checking for UOMPDS are all solvable in* EXPTIME. *All these problems are also NP-Hard even for Adjacent UOMPDS.*

The lower-bound is via a reduction to the NP-COMPLETE problem of deciding the emptiness of the intersection of n finite automata over a unary alphabet [14].

6.2. *An application to concurrent recursive queue systems*

La Torre *et al.* [20], study the decidability of control state reachability in networks of concurrent processes communicating via queues. Each component process may be recursive, i.e., equipped with a pushdown store, and such systems are called *recursive queuing concurrent programs* (RQCP) in [20]. Further, the state space of the entire system may be global or we may restrict each process to have its own local state space (so that the global state space is the product of the local states). In the terminology of [20] the latter are called RQCPs without shared memory.

An architecture describes the underlying topology of the network, i.e., a graph whose vertices denote the processes and edges correspond to communication channels (queues). One of the main results in [20] is a precise characterization of the architectures for which the reachability problem for RQCP's is decidable. Understandably, given the expressive power of queues and stacks, this class is very restrictive. To obtain any decidability at all, one needs the *well-queuing* assumption, which prohibits any process from dequeuing a message from any of its incoming channels as long as its stack is non-empty. They show that, even under the well-queuing assumption, the only architectures for which the reachability problem is decidable for RQCPs without shared memory are the so called *directed forest* architectures. A directed tree is a tree with an identified root and where all edges are oriented away from the root towards the leaves. A directed forest is a disjoint union of directed trees. They use a reduction to the reachability problem for bounded-phase MPDSs and obtain a double exponential decision procedure.

We now show that this problem can be reduced to the reachability problem for AOMPDS and obtain an EXPTIME upper-bound.[a] The reduction is sketched below. An EXPTIME upper-bound is also obtained via tree-width bounds [15] (Thm. 4.6).

Theorem 14. *The control state reachability problem for RQCPs with a directed forest architecture, without shared memory and under the well-queuing assumption can be solved in* EXPTIME.

Proof. (Sketch) We only consider the directed tree architecture and the result for the directed forest follows quite easily from this. An observation, from [20], is that it suffices to only consider executions with the following property: if q is a child of p then p executes all its steps (and hence deposits all its messages for q) before q executes. We fix some topologically sorted order of the tree, say $p_1, p_2, \ldots, p_m$ where p_1 is the root. The AOMPDS we construct only simulates those executions of the RQCP in which all moves of p_i are completed before p_{i+1} begins its execution. We call such a run of the RQCP as a *canonical run*. The number of stacks used is $2m - 1$. The message alphabet is $\Gamma \times \{1, \ldots, m\} \cup \bigcup_{1 \leq i \leq m} \Sigma_i$, where Γ is the communication message alphabet and Σ_i is the stack alphabet of process p_i. We write Γ_i to denote $\Gamma \times \{i\}$ and $w \downarrow \Sigma$ to denote the restriction of a word to the letters in Σ.

As we simulate a canonical run ρ of the RQCP in the order $p_1, \ldots, p_m$, the invariant we maintain is that, at the beginning of the simulation of process p_i, the contents of stack $2i - 1$ is some α so that $\alpha \downarrow \Gamma_i$ is the contents of the unique input

[a]The argument in Theorem 4 can also be adapted to show EXPTIME-HARDNESS.

channel to p_i as p_i begins its execution in ρ. Thus, we can simulate p_i's contribution to ρ, by popping from stack $2i-1$ when a value is to be consumed from the input queue. If top of stack $2i-1$ does not belong ot Γ_i, then we transfer it to stack $2i$. When p_i sends a message to any other process p_j in ρ (which must be one of its children in the tree) we simulate it by tagging the message with the process identity and pushing it on stack $2i$. Finally, as observed in [20], the stack for p_i can also be simulated on top of stack $2i-1$ since a value is dequeued only when its local stack is empty (according to the well-queuing assumption). At the end of the simulation of process p_i, we empty any contents left on stack $2i-1$ (transferring elements of $\Gamma \times \{i+1,\dots,m\}$ to stack $2i$). Finally, we copy stack $2i$ onto stack $2i+1$ and simulate process p_{i+1} using stack $2i+1$ (thus ensuring that there is no reversal of the contents of the queues). The state space is linear in the size of the RQCP and hence we conclude that the reachability problem for RQCPs can be solved in EXPTIME using Theorem 4. □

6.3. *Bounded-phase reachability for MPDSs*

We say that a MPDS is *simple* if each transition examines the top of at most one stack. That is, if $((q,\gamma_1,\dots,\gamma_n),(q',\alpha_1,\dots,\alpha_n)) \in \Delta$, then either $\gamma_1 = \gamma_2 = \dots = \gamma_n = \epsilon$ (a stack oblivious transition) or there is an $i : 1 \le i \le n$ such that $\gamma_i \in \Gamma$ and $\gamma_j = \epsilon$ for $j \ne i$ (an i-transition).

A run $c \xrightarrow{\rho}{}^*_M c'$ is an *i-run* if every transition that occurs in ρ is either a stack oblivious transition or an i-transition. A run $c \xrightarrow{\rho}{}^*_M c'$ is a *1-phase run* if it is an i-run for some stack i. A run $c \xrightarrow{\rho}{}^*_M c'$ is a *k-phase run* if we may write $\rho = \rho_1.\rho_2.\dots.\rho_k$ with $\rho_i \in \Delta^*$, $c = c_0 \xrightarrow{\rho_1}{}^*_M c_1 \xrightarrow{\rho_1}{}^*_M c_2 \dots \xrightarrow{\rho_k}{}^*_M c_k = c'$ and each $c_i \xrightarrow{\rho_{i+1}}{}^*_M c_{i+1}$ is a 1-phase run. We say that such a run is a *good k-phase run* if $k \le n$ and for all $1 \le i \le k$, $c_i \xrightarrow{\rho_{i+1}}{}^*_M c_{i+1}$ is an i-run in which the stacks $1, 2 \dots (i-1)$ are empty in every configuration.

The (*good*) *k-phase reachability problem* is to decide for a simple MPDS M, a number k and a state $q \in Q$, whether there is a l-phase run (good l-phase run) $(q_0, \bot, \bot, \dots, \gamma_0\bot) \rightarrow^*_M (q, \alpha_1, \alpha_2, \dots, \alpha_n)$ with $\alpha_i \in \Gamma^*$ for some $l \le k$. The k-phase reachability problem is shown to be 2-ETIME-Complete in [19]. This problem is interesting as it identifies a verifiable under-approximation of MPDSs and generalizes the context bounding restriction. We provide a reduction of this problem to the reachability problem for AOMPDSs, providing a simple proof of decidability for BPMPDSs and illustrating the expressive power of AOMPDS. We first observe that the k-phase reachability problem can be transformed to a good k-phase reachability problem.

Lemma 15. *Let $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ be a simple MPDS, k an integer and $q \in Q$. Then, the k-phase reachability problem for q in M can be reduced to the good k-phase reachability problem for some q' in a simple MPDS $M' = (k+1, Q', \Gamma \cup \{\#\}, \Delta', q_0', \gamma_0)$ where $|Q'| = \mathcal{O}(|Q|.n^k)$ and $|\Delta'| = \mathcal{O}(|\Delta|.n^{2k})$. Further, every run of M' is actually a good l-phase run for some $l \le k$.*

Proof. (sketch) The automaton begins by guessing a sequence $s_1, s_2, \ldots, s_k$, $1 \leq s_j \leq n$, of stacks that would be used in the k-phase run that is to be simulated. Notice that any stack s may appear more than once in this sequence (or even not at all). We inductively ensure that when this automaton begins its ith phase, the contents of stack i are exactly the contents of stack s_i at the beginning of phase i of the k-phase run of M which is being simulated.

A transition in M taken during phase i may push values into not only stack s_i but also the other stacks. The activity on stack s_i is simulated accurately using stack i. Further, we simply disregard the values pushed on any stack s that does not appear among $s_{i+1}, \ldots, s_k$ since these values are never used. Finally, any value pushed on to a stack s that appears among $s_{i+1}, \ldots, s_k$ is pushed on stack j where j is the smallest number such that $s_j = s$. At the end of the simulation of phase i, if stack s_i does not appear among $s_{i+1}, \ldots, s_k$ then we simply empty its contents before switching to simulating phase $i+1$ using stack $i+1$. If stack s_i is used again and j is the least number greater than i with $s_j = s_i$ then we transfer the contents of stack i to stack j via stack $i+1$ (so that the order is not reversed). The new stack symbol # is used as a marker on stack $i+1$ during the copying. The details are easy to formalize. $\square$

Next, we show that any good k-phase reachability problem for any simple MPDS M can be reduced to the reachability problem for an AOMPDS M'. Thus, using the EXPTIME complexity of AOMPDS, we get a 2-EXPTIME algorithm for BPMPDS.

Lemma 16. *Any good k-phase reachability problem for a simple MPDS $M = (n, Q, \Gamma, \Delta, q_0, \gamma_0)$ can be reduced to the state reachability problem for an AOMPDS $M' = (2^{n-1}, Q', \Gamma', \Delta', q'_0, \gamma'_0)$ where:* (1) $|Q'| = \mathcal{O}(|Q|.2^{\mathcal{O}(n)})$, (2) $|\Delta'| = \mathcal{O}(2^{\mathcal{O}(k)}.|\Delta|)$, *and* (3) $|\Gamma'| = \mathcal{O}(2^{\mathcal{O}(k)}.|\Gamma|)$.

Proof. (sketch) Observe that in a good k-phase run, during the ith phase values are popped only from stack i, which is also the leftmost nonempty stack. Further values are pushed only on stacks numbered i or higher. To simulate such a run using an AOMPDS we should ensure that the values are pushed only on stacks i or $i+1$ (or $i-1$, but given the nature of a good k-phase run this will be unnecessary). Our strategy is to push the values meant for all the stacks other than i into stack $i+1$, after appropriately tagging them with the identity of their destination. This naive strategy has some problems. Firstly, when operating on stack i in phase i we will encounter values meant for other stacks (with a tag identifying the destination stack j, $j > i$). We simply transfer these values as and when they are encountered to stack $i+1$. The second problem is that when values tagged with stack j are eventually transferred to stack j during phase $j-1$, they may not occur in the right order. One reason for this reordering is that as we transfer contents from stack i to stack $i+1$, the values meant for a future stack j get reversed in order.

This is a relatively minor issue which can be handled by inserting an additional stack, if necessary, to carry out one more reversal.

A more serious problem that results in an exponential increase in the number of stacks is the following — phase i in M may involve pushing values on stack j, $j > i$ and we instead tag it with j and push it on stack $i + 1$. However, it is quite possible that inside stack i there might be values for stack j pushed during earlier phases which will be uncovered later and transferred to stack $i + 1$, resulting in shuffling the values meant for stack j that are generated by i and those that are only transferred by i. To get round this we need to keep several copies of each stack, one for each stack whose phase may push values into this stack.

If the number of phases (and stacks) in M is just 1 or 2 then there is no problem as the aforementioned shuffling does not occur for trivial reasons. Suppose M has 3 stacks (and phases). Then we keep two copies of stack 3 which call say 3.1 and 3.2. We assume the order of the stacks is $1, 2, 3.2, 3.1$. Whenever a value is to be pushed onto stack 3 during phase 1, it is marked as destined for 3.1 and pushed on stack 2. During phase 2 any value destined for stack 3 is marked as destined for stack 3.2 and pushed on to the next stack (which is also 3.2). The phase corresponding to stack 3 is now broken up into two phases – first a phase on stack 3.2 and another on stack 3.1. The phase change from stack 3.2 to stack 3.1 takes place only when stack 3.2 is empty. Notice that this precisely captures the fact that any value pushed by phase 1 on stack 3 is accessible only after every value pushed by phase 2 on stack 3 has been removed. Observe that if M has 4 stacks then we will need 4 copies of stack 4 ($4.3.2, 4.3.1, 4.2, 4.1$ in that order) where the any value meant for stack 4 during a phase on stack $i \in \{1, 2, 3.2, 3.1\}$) is marked as destined for stack $4.i$. Thus, if there were n phases (and stacks) we construct end up with an AOMPDS with 2^{n-1} phases and stacks. It also turns out that the number of stacks between any stack i and its consumers (stacks of the form $j.i, j \in [1..n]$) is always even. Thus no additional reversal is needed. This construction can be easily formalised to obtain an AOMPDS as required and the proof of correctness is simple but tedious. $\square$

References

[1] M. F. Atig, Global model checking of ordered multi-pushdown systems, *FSTTCS*, *LIPIcs* **8** (Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010), pp. 216–227.

[2] M. F. Atig, B. Bollig and P. Habermehl, Emptiness of multi-pushdown automata is 2ETIME-complete, *DLT*, *LNCS* **5257** (Springer, 2008), pp. 121–133.

[3] M. F. Atig, A. Bouajjani, M. Emmi and A. Lal, Detecting fair non-termination in multithreaded programs, *CAV*, *LNCS* **7358** (Springer, 2012), pp. 210–226.

[4] M. Atig, A. Bouajjani, K. Narayan Kumar and P. Saivasan, Linear-time model-checking for multithreaded programs under scope-bounding, *ATVA*, *LNCS* **0** (Springer Berlin/Heidelberg, 2012), pp. 152–166.

[5] M. F. Atig, K. N. Kumar and P. Saivasan, Adjacent ordered multi-pushdown systems, *DLT*, *LNCS* **7907** (Springer, 2013), pp. 58–69.

[6] B. Bollig, A. Cyriac, P. Gastin and M. Zeitoun, Temporal logics for concurrent recursive programs: Satisfiability and model checking, *MFCS*, *LNCS* **6907** (Springer, 2011), pp. 132–144.
[7] A. Bouajjani, J. Esparza, S. Schwoon and J. Strejcek, Reachability analysis of multithreaded software with asynchronous communication, *FSTTCS*, *LNCS* **3821** (Springer, 2005), pp. 348–359.
[8] A. Bouajjani, S. Fratani and S. Qadeer, Context-bounded analysis of multithreaded programs with dynamic linked structures, *CAV*, *LNCS* **4590** (2007), pp. 207–220.
[9] L. Breveglieri, A. Cherubini, C. Citrini and S. Crespi-Reghizzi, Multi-push-down languages and grammars, *Int. J. Found. Comput. Sci.* **7**(3) (1996) 253–292.
[10] A. Cyriac, P. Gastin and K. N. Kumar, MSO decidability of multi-pushdown systems via split-width, *CONCUR*, *LNCS* **7454** (Springer, 2012), pp. 547–561.
[11] A. Heußner, J. Leroux, A. Muscholl and G. Sutre, Reachability analysis of communicating pushdown systems, *FOSSACS*, *LNCS* **6014** (Springer, 2010), pp. 267–281.
[12] A. Lal and T. W. Reps, Reducing concurrent analysis under a context bound to sequential analysis, *FMSD* **35**(1) (2009) 73–97.
[13] A. Lal, T. Touili, N. Kidd and T. W. Reps, Interprocedural analysis of concurrent programs under a context bound, *TACAS*, *LNCS* **4963** (Springer, 2008), pp. 282–298.
[14] K.-J. Lange and P. Rossmanith, The emptiness problem for intersections of regular languages, *MFCS*, *LNCS* **629** (Springer, 1992), pp. 346–354.
[15] P. Madhusudan and G. Parlato, The tree width of auxiliary storage, *POPL* (ACM, 2011), pp. 283–294.
[16] A. Pnueli, The temporal logic of programs, *FOCS* (IEEE, 1977), pp. 46–57.
[17] S. Qadeer and J. Rehof, Context-bounded model checking of concurrent software, *TACAS*, *LNCS* **3440** (Springer, 2005), pp. 93–107.
[18] A. Seth, Global reachability in bounded phase multi-stack pushdown systems, *CAV*, *LNCS* **6174** (Springer, 2010), pp. 615–628.
[19] S. L. Torre, P. Madhusudan and G. Parlato, A robust class of context-sensitive languages, *LICS* (IEEE Computer Society, 2007), pp. 161–170.
[20] S. L. Torre, P. Madhusudan and G. Parlato, Context-bounded analysis of concurrent queue systems, *TACAS*, *LNCS* **4963** (Springer, 2008), pp. 299–314.
[21] S. L. Torre, P. Madhusudan and G. Parlato, An infinite automaton characterization of double exponential time, *CSL*, *LNCS* **5213** (Springer, 2008), pp. 33–48.
[22] S. L. Torre, P. Madhusudan and G. Parlato, Reducing context-bounded concurrent reachability to sequential reachability, *CAV*, *LNCS* **5643** (Springer, 2009), pp. 477–492.
[23] S. L. Torre and M. Napoli, Reachability of multistack pushdown systems with scope-bounded matching relations, *CONCUR*, *LNCS* **6901** (Springer, 2011), pp. 203–218.
[24] S. L. Torre and M. Napoli, A temporal logic for multi-threaded programs, *IFIP TCS*, *LNCS* **7604** (Springer, 2012), pp. 225–239.
[25] M. Y. Vardi, A temporal fixpoint calculus, *POPL* (1988), pp. 250–259.
[26] M. Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification (preliminary report), *LICS* (IEEE Computer Society, 1986), pp. 332–344.