Context-Bounded Analysis for Concurrent Programs with Dynamic Creation of Threads

[1] LIAFA, CNRS and University Paris Diderot, France
{atig,abou}@liafa.jussieu.fr
[2] Microsoft Research, Redmond
qadeer@microsoft.com

Abstract. Context-bounded analysis has been shown to be both efficient and effective at finding bugs in concurrent programs. According to its original definition, context-bounded analysis explores all behaviors of a concurrent program up to some fixed number of context switches between threads. This definition is inadequate for programs that create threads dynamically because bounding the number of context switches in a computation also bounds the number of threads involved in the computation. In this paper, we propose a more general definition of context-bounded analysis useful for programs with dynamic thread creation. The idea is to bound the number of context switches for each thread instead of bounding the number of switches of all threads. We consider several variants based on this new definition, and we establish decidability and complexity results for the analysis induced by them.

1 Introduction

The verification of multithreaded programs is a challenging problem both from the theoretical and the practical point of view. (We consider here programs with parallel threads which may use local variables as well as shared (global) variables.) Assuming that the variables of the program range over a finite domain (which can be obtained using some abstraction on the manipulated data), there are several aspects in multithreaded programs which make their analysis complex or even undecidable in general [13].

Indeed, it is well known that for instance in the case where each thread can be modeled as a finite-state system, the state space of the program grows exponentially w.r.t. the number of threads, and the reachability problem is PSPACE-hard. Moreover, if threads are modeled as pushdown systems, which corresponds to allowing unbounded depth (recursive) procedure calls in the program, then the reachability problem becomes undecidable as soon as two threads are considered.

Context-bounding has been proposed in [10] as a suitable technique for the analysis of multithreaded programs. The idea is to consider only the computations of the program that perform at most some fixed number of context switches between threads. (At each point only one thread is active and can modify the global variables, and a context-switch happens when the active thread terminates or is interrupted, and a pending one is activated.) The state space which must be explored may still be unbounded in presence of recursive procedure calls, but the context-bounded reachability problem

S. Kowalewski and A. Philippou (Eds.): TACAS 2009, LNCS 5505, pp. 107–123, 2009.

is decidable even in this case. In fact, context-bounding provides a very useful tradeoff between computational complexity and verification coverage. This tradeoff is based on three important properties. First, context-bounded verification can be performed more efficiently than unbounded verification. From the complexity-theoretic point of view, it can be seen that context-bounded reachability is an NP-complete problem (even in the case of pushdown threads). Second, many concurrency errors, such as data races and atomicity violations, are manifested in executions with few context switches [9]. Finally, verifying all executions of a concurrent program up to a context bound provides an intuitive and meaningful notion of coverage to the programmer.

In the last few years, several implementations and algorithmic improvements have been proposed for context-bounded verification [2,9,16,7,6]. For instance, context-bounded verification has been implemented in explicit-state model checkers such as CHESS [9] and SPIN [18]; it has also been implemented in symbolic model checkers such as SLAM [11], jMoped [16], and in [6].

While the concept of context-bounding is adequate for multithreaded programs with a (fixed) finite number of threads, the question we consider in this paper is whether this concept is still adequate when dynamic creation of threads is considered.

Dynamic thread creation is useful for modeling several important aspects, e.g., (1) unbounded number of concurrently execution of software modules such as file systems, device drivers, non-blocking data structures etc., or (2) creation of asynchronous activity such as forking a thread, queuing a closure to a threadpool with or without timers, callbacks, etc. Both these sources are very important for modeling operating system components; they are likely to become important even for application software as it becomes increasingly parallel in order to harness the power of multi-core architectures.

We argue that the "classical" notion of context-bounding which has been used so far in the existing work is actually too restrictive in this case. Indeed, bounding the number of context switches in a computation also bounds the number of threads involved. In this paper, we propose a more general definition of context-bounded analysis useful for programs with dynamic thread creation. The idea is to bound the number of context switches for each thread instead of bounding the number of switches of all threads. We consider several variants based on this new definition, and we establish decidability and complexity results for the analysis induced by them.

We introduce a notion of K-bounded computations where each of the involved threads can be interrupted and resumed at most K times. (We consider that when a thread is created, the number of context switches it can perform is the one of its ancestor minus 1.) Notice that the number of context switches by all threads in a computation is not bounded since the number of threads involved is not bounded.

In the case of finite-state threads, we prove that this problem is as hard as the coverability problem for Petri nets (which is EXPSPACE-complete). The reduction from our problem to the coverability problem of Petri nets is based on the simple idea of counting the number of pending threads for different values of the global and local states, as well as of the number of switches that these threads are allowed to perform. conversely, we prove that the coverability problem of Petri nets can be reduced to the 2-bounded reachability problem. These results show that in the case of dynamic thread creation, considering the notion of context-bounding for each individual thread makes the

complexity jumps from NP-completeness to EXPSPACE-completeness, even in the case of finite-state threads. Then, an interesting question is whether it is possible to have a notion of context-bounding with a lower complexity. We propose for that the notion of stratified context-bounding. The idea is to consider computations where the scheduling of the threads is ordered according to their number of allowed switches: First, threads of level K (the level means here the number of allowed switches) are scheduled generating threads of level $K-1$, then threads of level $K-1$ are scheduled, and so on. Next, it is possible to schedule again threads of level K and repeat the process, but this only for a finite number of times L. Again, notice that (K,L)-stratified computations may have an unbounded number of context switches since it is possible to schedule an unbounded number of threads at each level. This concept generalizes obviously the "classical" notion of context-bounding. We prove that, for finite-state threads, the (K,L)-stratified context-bounded reachability problem is NP-complete (i.e., it matches the complexity of the "classical" context-bounded reachability problem). The proof is by a reduction to the satisfiability problem of existential Presburger formulas.

Then, we consider the case of dynamic creation of pushdown threads. We prove that, surprisingly, the K-bounded reachability problem is in fact decidable, and that the same holds also for the (K,L)-stratified context-bounded reachability problem. To establish these results, we prove that these problems (for pushdown threads) can be reduced to their corresponding problems for finite-state threads. This reduction is not trivial. The main ideas behind the reduction are as follows: First, the K-bounded behaviors of each single thread can be represented by a labeled pushdown system which (1) makes visible (as labels) on its transitions the created threads, and (2) guesses points of interruption-resumption and the corresponding values of the global states. (These guesses are also made visible on the transitions.) Then, the main problem is to "synchronize" these labeled pushdown systems so that their guesses can be validated. The key observation is that it is possible to abstract these systems without loss of preciseness by finite-state systems. This is due to the fact that we can consider that some of the generated threads can be lost (since they can be seen as threads that are never activated), and therefore we can reason about the downward closure of the languages of the labeled pushdown systems mentionned above (w.r.t. suitable sub-word relation). This downward closure is in fact always regular and effectively constructible.

2 Preliminaries

Words and languages. Let Σ be a finite alphabet. We denote by Σ^* (resp. Σ^+) the set of all *words* (resp. non empty words) over Σ, and by ε the empty word. A language L is a (possibly infinite) set of words. Let $u \in \Sigma^*$ and $a \in \Sigma$. We denote by $|u|$ the length of u and by $|u|_a$ the number of occurrences of a in u. Consider a non empty word $u = a_1 \cdots a_n$. For any i such that $1 \leq i \leq n$, we denote by u_i the symbol a_i.

Given an alphabet Σ, we denote by $\preceq \subseteq \Sigma^* \times \Sigma^*$ the *subword relation* defined as follows: for every $u = a_1 \cdots a_n \in \Sigma^*$, and every $v = b_1 \cdots b_m \in \Sigma^*$, $u \preceq v$ iff $\exists i_1, \ldots, i_n \in \{1, \ldots, m\}$ such that $i_1 < i_2 < \ldots < i_n$ and $\forall j \in \{1, \ldots, n\}, a_j = b_{i_j}$. Given a language $L \subseteq \Sigma^*$, the *downward closure* of L (w.r.t. $\preceq$) is the set $L\downarrow = \{u \in \Sigma^* \mid \exists v \in L, u \preceq v\}$.

Finite State Automata. A Finite State Automaton (FSA) is a tuple $\mathcal{S} = (S, \Sigma, \delta, s^{init}, s^{final})$ where S is a finite set of states, Σ is a finite input alphabet, $\delta \subseteq S \times \big(\Sigma \cup \{\varepsilon\}\big) \times S$ is a finite set of transitions, $s^{init} \in S$ is the initial state, and $s^{final} \in S$ is the acceptor state. The language accepted by the finite state automaton $\mathcal{S}$ is denoted $L(\mathcal{S})$.

Labeled Pushdown Systems. A Labeled Pushdown System (LPDS) is defined by a tuple $\mathcal{P} = (G, \Sigma, \Gamma, \Delta)$ where G is a finite set of states, Σ is an input alphabet (actions), Γ is a stack alphabet, and Δ is a finite set of transition rules of the form: $g\gamma \overset{a}{\hookrightarrow} g'w'$ where $g, g' \in G$, $a \in \Sigma \cup \{\varepsilon\}$, $\gamma \in \Gamma$, and $w' \in \Gamma^*$ such that $|w'| \leq 2$.

A *configuration* of $\mathcal{P}$ is a tuple $\langle g, \sigma, w\rangle$ where $g \in G$ is a state, $\sigma \in \Sigma^*$ is an input word, and $w \in \Gamma^*$ is a stack content. We define the binary relation $\Rightarrow_{\mathcal{P}}$ between configurations as follows: $\langle g, a\sigma, \gamma w\rangle \Rightarrow_{\mathcal{P}} \langle g', \sigma, w'w\rangle$ iff $g\gamma \overset{a}{\hookrightarrow} g'w' \in \Delta$. The transition relation $\Rightarrow^*_{\mathcal{P}}$ is the reflexive transitive closure of the binary relation $\Rightarrow_{\mathcal{P}}$.

Given a labeled pushdown system $\mathcal{P} = (G, \Sigma, \Gamma, \Delta)$, two states $g, g' \in G$, and a stack symbol γ, let $L_{\mathcal{P}}(g\gamma, g') = \{\sigma \in \Sigma^* \mid \exists w \in \Gamma^* \; s.t \; \langle g, \sigma, \gamma\rangle \Rightarrow^*_{\mathcal{P}} \langle g', \varepsilon, w\rangle\}$. Clearly, $L_{\mathcal{P}}(g\gamma, g')$ is a context-free language, and conversely, every context-free language can be defined as a trace language of some labeled pushdown system.

We recall hereafter a result due to Courcelle [3] which will be used later in the paper.

Theorem 1. *Let $\mathcal{P} = (G, \Sigma, \Gamma, \Delta)$ be a LPDS, $g, g' \in G$ be two states, and $\gamma \in \Gamma$ be a stack symbol. Then, it is possible to construct a FSA $\mathcal{S} = (S, \Sigma, \delta, s^{init}, s^{final})$ such that $L(\mathcal{S}) = \big(L_{\mathcal{P}}(g\gamma, g')\big)\downarrow$, where in the worst case $|S|$ is exponential in $\big(|G| + |\Sigma| + |\Gamma|\big)$.*

Multi-sets. Let Ξ be a nonempty alphabet (possibly infinite). A *multi-set* over Ξ is a function $M : \Xi \to \mathbb{N}$. We denote by $M[\Xi]$ the collection of all multi-sets over Ξ and by $\emptyset$ the empty multi-set. Given two multi-sets M and M', we write $M' \leq M$ iff $M'(a) \leq M(a)$ for every $a \in \Xi$; and $M + M'$ (resp. $M - M'$ if $M' \leq M$) to denote the multi-set where $(M + M')(a) = M(a) + M'(a)$ (resp. $(M - M')(a) = M(a) - M'(a)$) for every $a \in \Xi$. For every word $u \in \Xi^*$, $[u]$ is the multi-set such that $[u](a) = |u|_a$ for every $a \in \Xi$. Sometimes, $[u]$ is called the Parikh image of u. This definition is extended in the straightforward manner to languages (sets of words) as follows $[L] = \{[u] \; : \; u \in L\}$.

Petri Nets. A Petri net is a pair $\mathcal{N} = (P, T)$ where P is a finite set of places and $T \subseteq P^* \times P^*$ is a finite set of transition rules. We often write $w \rightarrowtail w'$ to denote a transition $(w, w') \in T$. Given a transition $t = w \rightarrowtail w' \in T$, we define a relation $\overset{t}{\rightarrow} \subseteq \big(M[P] \times M[P]\big)$ between multi-sets over P as follows: $W \overset{t}{\rightarrow} W'$ iff $W \geq [w]$ and $W' = W + [w'] - [w]$. We define the transition relation $\rightarrow_{\mathcal{N}}$ on multi-sets over P by the union of $\overset{t}{\rightarrow}$, i.e., $\rightarrow_{\mathcal{N}} = \bigcup_{t \in T} \overset{t}{\rightarrow}$. The transition relation $\rightarrow^*_{\mathcal{N}}$ is the reflexive transitive closure of $\rightarrow_{\mathcal{N}}$.

The coverability problem for a Petri net $\mathcal{N}$ is the problem of deciding for two given places p and p' whether there is a multi-set W such that $[p'] \leq W$ and $[p] \rightarrow^*_{\mathcal{N}} W$.

Theorem 2. *The coverability problem for Petri nets is EXPSPACE-complete* [8,12].

Existential Presburger Formulas (EPF). Let $\mathcal{V}$ be a set of variables. We use $x, y, \ldots$ to range over variables in $\mathcal{V}$. The set of *existential* Presburger formulas is defined by the following grammar and interpreted over natural numbers:

$$t ::= 0 \mid 1 \mid x \mid t_1 + t_2 \qquad \phi ::= t_1 > t_2 \mid t_1 = t_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \exists x \cdot \phi_1$$

The semantics of these formulas is defined in the standard way. Given a formula ϕ with free variables $x_1, \ldots, x_n$, and a valuation $U : \mathcal{V} \to \mathbb{N}$, we denote $\phi(U)$ the truth value of ϕ for the valuation U. We say that a formula ϕ is satisfiable if there is some valuation U such that $\phi(U)$ is true. It is well-know that the satisfiability problem for existential Presburger formulas is decidable [17], and that:

Theorem 3. *The satisfiability problem for existential Presburger formulas is NP-complete.*

Given a language L over the alphabet $\Sigma = \{a_1, \ldots, a_n\}$, the set $[L]$ (called the Parikh image of L) is definable by a formula ϕ with free variables $x_{a_1}, \ldots, x_{a_n}$ if for every valuation U, $\phi(U)$ is true iff there is a word $u \in L$ such that $U(x_{a_i}) = [u](a_i)$ for every $i \in \{1, \ldots, n\}$. We recall a result about existential Presburger formulas given in [14,17].

Theorem 4. *Let $\mathcal{P} = (G, \Sigma, \Gamma, \Delta)$ be a LPDS, $g, g' \in G$ be two states, and $\gamma \in \Gamma$ be a stack symbol. Then, it is possible to compute in polynomial time an existential Presburger formula ϕ which defines $[L_{\mathcal{P}}(g\gamma, g')]$.*

3 Dynamic Networks of Concurrent Systems

3.1 Syntax

A Dynamic Network of Concurrent Pushdown System (DCPS) is a tuple $\mathcal{A} = (G, \Gamma, \Delta, g_0, \gamma_0)$ where G is a finite set of *states*, Γ is a finite set of *stack symbols*, g_0 is the initial state, γ_0 is the initial stack symbol, and Δ is a finite sets of transition rules of the forms: (1) $g\gamma \hookrightarrow g'w'$, or (2) $g\gamma \hookrightarrow g'w' \rhd \gamma'$ where $g, g' \in G$, $\gamma, \gamma' \in \Gamma$, $w' \in \Gamma^*$, and $|w'| \leq 2$.

A DCPS models dynamic multithreaded programs with (potentially recursive) procedure calls. Threads are modeled as pushdown processes which can spawn new processes. They have local variables and have also access to global (shared) variables. The values of the local variables are modeled using the stack alphabet Γ, whereas the values of the global variables are modeled using states in G. Rules of the form $g\gamma \hookrightarrow g'w'$ correspond to standard transitions of pushdown systems (popping γ and then pushing w' while changing the control state from g to g'), and rules of the form $g\gamma \hookrightarrow g'w' \rhd \gamma'$ are similar but in addition they create a new thread with an initial local state γ'. Notice that push (resp. pop) operations allow to model procedure calls (resp. returns).

When unbounded recursion is not considered, threads can be modeled as finite-state processes instead of pushdown systems. This corresponds to the special case where in all the transition rules defined above the pushed sequence w' is of size at most 1. We use the acronym DCFS for dynamic networks of concurrent finite-state system.

3.2 Semantics

A configuration of $\mathcal{A}$ is given by (1) a state g (the current value of the global store), (2) the local configuration of the *active* thread, which is a pair (w, i) where w is its call stack and i is its switch number (the number of interruptions/resumptions of the thread together with the switch number of its ancestor at the moment of its creation), and

(3) a multiset of the local configurations of the idle threads. Formally, a configuration of $\mathcal{A}$ is a tuple $\langle g,(w,i),M\rangle \in G\times\big(\Gamma^*\times\mathbb{N}\big)\times M[\Gamma^*\times\mathbb{N}]$. We assume that the initial configuration of $\mathcal{A}$ is $(g_0,(\gamma_0,0),\emptyset)$.

For a given $i\in\mathbb{N}$, the relation $\Rightarrow_i$ on configurations is $\rightarrow_i\cup\mapsto_i$, where $\rightarrow_i$ and $\mapsto_i$ are defined as follows:

- $\langle g,(\gamma w,i),M\rangle\rightarrow_i\langle g',(w'w,i),M'\rangle$ iff (1) there is a rule $g\gamma\hookrightarrow g'w'\in\Delta$ and $M=M'$, or (2) there is a rule $g\gamma\hookrightarrow g'w'\rhd\gamma'\in\Delta$ and $M'=M+[(\gamma',i+1)]$.
- $\langle g,(w,i),M+[(w',j)]\rangle\mapsto_i\langle g,(w',j),M+[(w,i+1)]\rangle$ for every $j\in\mathbb{N}$, $g\in G$, $M\in M[\Gamma^*\times\mathbb{N}]$, and $w,w'\in\Gamma^*$.

The relations $\rightarrow_i$ correspond to the execution of pushdown (pop and push) operations, with the possibility of creating new threads (added to the multiset of idle threads). The created threads get the switch number $i+1$. The relations $\mapsto_i$ correspond to context switches: The local configuration (w',j) of a waiting thread is taken from the multiset and given the status of active, while the local configuration (w,i) of the interrupted task is stored in the multiset after incrementing its switch number.

Let $\Rightarrow_{\leq B}=\bigcup_{i\leq B}\Rightarrow_i$ for every $B\in\mathbb{N}\cup\{\infty\}$. We write simply $\Rightarrow$ instead of $\Rightarrow_{\leq\infty}$. Finally, $\Rightarrow_i^*$ and $\Rightarrow_{\leq B}^*$ denote the transitive closure of $\Rightarrow_i$ and $\Rightarrow_{\leq B}$, respectively.

3.3 Reachability Problems

We consider the three following notions of reachability:

State reachability: A state g is said to be reachable iff $\langle g_0,(\gamma_0,0),\emptyset\rangle\Rightarrow^*\langle g,(w,j),M\rangle$ for some $(w,j)\in\Gamma^*\times\mathbb{N}$ and $M\in M[\big(\Gamma^*\times\mathbb{N}\big)]$. The state reachability problem (SRP for short) is, for a given DCPS $\mathcal{A}$ and $g\in G$, to determine whether g is reachable by $\mathcal{A}$.

K-bounded state reachability: Given $K\in\mathbb{N}$, a state $g\in G$ is said to be *K-reachable* iff $\langle g_0,(\gamma_0,0),\emptyset\rangle\Rightarrow_{\leq K}^*\langle g,(w,j),M\rangle$ for some $(w,j)\in\Gamma^*\times\{0,\ldots,K\}$, and $M\in M[\Gamma^*\times\{0,\ldots,K+1\}]$. The K-bounded state reachability problem (SRP[K] for short) is, for a given DCPS $\mathcal{A}$, $K\in\mathbb{N}$, and $g\in G$, to determine whether g is K-reachable by $\mathcal{A}$.

Observe that, in SRP[K], a bound K is imposed on the number of switches (interuptions/resumptions) performed by each thread (together with the switch number of its ancestor at the moment of its creation). However, due to dynamic creation of threads, bounding the number of switches of each thread does not bound the number of switches in the whole computations of the system (since an arbitrarily large number of threads can be involved in these computations).

L-stratified K-bounded state reachability: Given $K,L\in\mathbb{N}$, a state $g\in G$ is said to be $[K,L]$-reachable iff $\langle g_0,(\gamma_0,0),\emptyset\rangle(\Rightarrow_0^*\circ\cdots\circ\Rightarrow_K^*)^L\langle g,(w,j),M\rangle$ for some $(w,j)\in\Gamma^*\times\{0,\ldots,K\}$, and $M\in M[\Gamma^*\times\{0,\ldots,K+1\}]$. The L-stratified K-bounded state reachability problem (SRP[K,L] for short) is, for a given DCPS $\mathcal{A}$, $g\in G$, and $K,L\in\mathbb{N}$, to determine whether g is $[K,L]$-reachable.

In SRP[K,L], a special kind of K-bounded computations (called stratified computations) are considered: In one stratum of such a computation, threads are scheduled according to their increasing switch number (from 0 to K). This corresponds to the consideration of the relation $\Rightarrow_0^*\circ\cdots\circ\Rightarrow_K^*$. Then, a L-stratified computation is a sequence

of L strata. Observe that even in the case of stratified computations, an arbitrarily large number of context switches may occur along a computation due to dynamic creation of threads. Moreover, relaxing the bound L, i.e. considering arbitrarily large sequences of strata, corresponds to considering the SRP$[K]$ problem. Very particular stratified computations are those where the whole number of context switches is bounded [10].

4 Analysis of Dynamic Networks of Concurrent Finite-State Systems

In this section, we show that SRP and SRP$[K]$ are EXPSPACE-complete (Theorem 5), whereas, the problem SRP$[K,L]$ is NP-complete (Theorem 6).

Theorem 5. *The problems SRP and SRP$[K]$ for DCFSs, for every natural number $K \geq 2$, are EXPSPACE-complete.*

Proof: We prove that SRP[K] for DCFSs is polynomially reducible to the coverability problem for Petri nets and vice-versa. We give here a sketch of the proof (for more details see [1]). The constructions presented below can be adapted to show that the SRP problem for DCFSs is also EXPSPACE-complete.

From SRP$[K]$ for DCFSs to coverability problem for Petri nets. Given a natural number K and a DCFS $\mathcal{A} = (G, \Gamma, \Delta, g_0, \gamma_0)$, we construct a Petri net $\mathcal{N} = (P, T)$ which has the following structure:

- The set of places:
  - A place (w, j) is associated with each natural number $j \in \{0, \ldots, K+1\}$ and each stack configuration $w \in \Gamma \cup \{\varepsilon\}$. The number of tokens in the place (w, j) is the number of pending threads of $\mathcal{A}$ with local configuration (w, j).
  - A place (g, w, i) is associated with each $i \in \{0, \ldots, K\}$, each state $g \in G$, and each stack configuration $w \in \Gamma \cup \{\varepsilon\}$. A token in the place (g, w, i) represents the fact that g is the current value of the global store of $\mathcal{A}$ and that (w, i) is the local configuration of the active thread.
- The set of transitions:
  - For each $i \in \{0, \ldots, K\}$ and each rule $g_1\gamma \hookrightarrow g_2 w$ (resp. $g_1\gamma \hookrightarrow g_2 w \rhd \gamma'$) of $\mathcal{A}$, $\mathcal{N}$ has a transition $(g_1, \gamma, i) \rightarrowtail (g_2, w, i)$ (resp. $(g_1, \gamma, i) \rightarrowtail (g_2, w, i)(\gamma', i+1)$).
  - For each $i, j \in \{0, \ldots, K\}$, each state $g \in G$, and each pair of stack configurations $w, w' \in \Gamma \cup \{\varepsilon\}$, there is a transition $(g, w, i)(w', j) \rightarrowtail (g, w', j)(w, i+1)$ in T. This transition simulates a context switch of $\mathcal{A}$.

Notice that the size of $\mathcal{N}$ is polynomial in the size of $\mathcal{A}$.

Lemma 1. $(g_0, (\gamma_0, 0), \emptyset) \Rightarrow^* (g, (w, i), M)$ *for some* $(w, i) \in (\Gamma \cup \{\varepsilon\}) \times \{0, \ldots, K\}$ *and* $M \in M[(\Gamma \cup \{\varepsilon\}) \times \{0, \ldots, K+1\}]$ *iff* $[(g_0, \gamma_0, 0)] \rightarrow^*_{\mathcal{N}} [(g, w, i)] + M$.

From coverability problem for Petri nets to SRP[2] for DCFSs. Given a Petri net $\mathcal{N} = (P, T)$ and two places $p_0, p_f \in P$, we construct a DCFS $\mathcal{A} = (G, \Gamma, \Delta, g_0, \gamma_0)$ such that: the state g_f is 2-reachable by $\mathcal{A}$ iff there is a multi-set $W \in M[P]$ such that $[p_0] \rightarrow^*_{\mathcal{P}} W$ and $[p_f] \leq W$. Intuitively, $\mathcal{A}$ has a special stack symbol γ_1 such that the number of pending threads with local configuration $(\gamma_1, 1)$ gives an upper bound of the length

of the run of $\mathcal{N}$ simulated by $\mathcal{A}$. For each place $p \in P$, $\mathcal{A}$ has a stack symbol p; the number of such pending threads with stack content p denotes the current number of tokens in p. We now sketch the behavior of $\mathcal{A}$. The DCFS $\mathcal{A}$ guesses the length of the simulated run of $\mathcal{N}$ by creating a number of threads with local configuration $(\gamma_1, 1)$ from the initial configuration. Then, the simulation of a rule $t = w \rightarrowtail w' \in T$ is done in two steps. First, $\mathcal{A}$ checks if t can be fired by verifying if there is a pending thread with local configuration $(w_i, 2)$ for every $i \in \{1, \ldots, |w|\}$. Second, $\mathcal{A}$ uses pending threads with local configuration $(\gamma_1, 1)$ to create, for every $j \in \{1, \ldots, |w'|\}$, a thread with local configuration $(w'_j, 2)$. Finally, to check if there is a token in the place p_f, $\mathcal{A}$ verifies if there a pending thread with local configuration $(p_f, 2)$ and moves its state to g_f. Formally, $\mathcal{A}$ is built up from $\mathcal{N}$ as follows:

- The set of states:
  - $\mathcal{A}$ has two special states g_0 and g_f. The states g_0 and g_f are the initial state and the final state, respectively.
  - For each rule $t = w \rightarrowtail w' \in T$, $\mathcal{A}$ has the following sequences of global states $g_{(t,w_1)}, \ldots, g_{(t,w_{|w|})}$ and $g_{(t,w'_1)}, \ldots, g_{(t,w'_{|w'|})}$. The sequence of states $g_{(t,w_1)}, \ldots, g_{(t,w_{|w|})}$ is used to simulate taking iteratively a token from each place w_i for $i = 1$ to $|w|$. The sequence of states $g_{(t,w'_1)}, \ldots, g_{(t,w'_{|w'|})}$ is used to simulate adding iteratively a token to each place w'_j for $j = 1$ to $|w'|$.
- The set of stack alphabet:
  - $\mathcal{A}$ has two special stack symbols γ_0 and γ_1. The symbol γ_0 represents the initial stack content. Symbols γ_1 represent auxiliary threads that are "*consumed*" for simulating tokens generation by transitions of $\mathcal{N}$.
  - For each place $p \in P$, $\mathcal{A}$ has a stack symbol p. The number of pending threads with stack content p denotes the current number of token in the place p.
- The set Δ is the smallest set of rules satisfying the following conditions:
  - **Guessing the length of the run of $\mathcal{N}$:** The rule $g_0\gamma_0 \hookrightarrow g_0\gamma_0 \rhd \gamma_1$ is in Δ. This rule creates an arbitrary number of threads γ_1 with switch number 1. This gives an upper bound of the length of the run of $\mathcal{N}$ simulated by $\mathcal{A}$.
  - **Creation of the initial marking of $\mathcal{N}$:** The rule $g_0\gamma_0 \hookrightarrow g_0 \rhd p_0$ is in Δ. This rule creates a thread p_0. This corresponds to the initial multiset $[p_0]$ of $\mathcal{N}$.
  - **Simulation of a transition rule of $\mathcal{N}$:** A transition $t = w \rightarrowtail w'$ is simulated by checking iteratively that there is a pending thread with stack content w_i for $i = 1$ to $|w|$, and then, by creating iteratively a thread with initial stack content w'_j for $j = 1$ to $|w'|$.
    - **Initialization of the simulation:** For each transition $t = w \rightarrowtail w' \in T$, the rule $g_0 w_1 \hookrightarrow g_{(t,w_1)}$ is in Δ. This rule corresponds to start simulating t and to check that there is a pending thread with stack content w_1.
    - **Checking if the transition rule can be fired:** For each transition $t = w \rightarrowtail w' \in T$ and for each $i \in \{1, \ldots, |w| - 1\}$, the rules $g_{(t,w_i)} w_{i+1} \hookrightarrow g_{(t,w_{i+1})}$ are in Δ. These transitions simulate the operation of taking a token from each place in $w_2, \ldots, w_{|w|}$.
    - **Generating the output tokens:** For each transition $t = w \rightarrowtail w' \in T$ and for each $j \in \{1, \ldots, |w'| - 1\}$, the rules $g_{(t,w_{|w|})}\gamma_1 \hookrightarrow g_{(t,w'_1)}\gamma_1 \rhd w'_1$,

$g_{(t,w'_j)}\gamma_1 \hookrightarrow g_{(t,w'_{j+1})}\gamma_1 \rhd w'_{(t,j+1)}$, and $g_{(t,w'_{|w'|})}\gamma_1 \rightarrowtail g_0$ are in Δ. These transitions simulate the operation of adding a token to each place of $w'_1,\ldots,w'_{|w'|}$. Notice that if the thread γ_1 has a switch number 1, then the created threads $w'_1,\ldots,w'_{|w'|}$ have switch number 2.

- **Checking the final marking**: The rule $g_0 p_f \hookrightarrow g_f$ is in Δ. This corresponds to checking if there a multiset with at least one token in p_f reachable by $\mathcal{N}$.

The relation between $\mathcal{N}$ and $\mathcal{A}$ is given by the following lemma:

Lemma 2. *The control state g_f is 2-reachable by $\mathcal{A}$ iff there a marking $W \in M[P]$ such that $W \geq [p_f]$ and $[p_0] \rightarrow^*_{\mathcal{N}} W$.* □

We consider now the problem SRP$[K,L]$ for $K,L \in \mathbb{N}$. We prove that:

Theorem 6. *For every $K,L \in \mathbb{N}$, the problem SRP$[K,L]$ for DCFSs is NP-complete.*

Proof: NP-hardness is proved by a reduction from the coverability problem for acyclic Petri nets [15] to SRP$[K,1]$. This is done by a simple adaptation of the construction given in Theorem 5. The upper-bound is obtained by a reduction to the satisfiability problem of *existential* Presburger formulas. We sketch hereafter the proof for the special when $L = 1$. The extension to $L > 1$ is straightforward [1].

Let $\mathcal{A} = (G,\Gamma,\Delta,g_0,\gamma_0)$ be a DCFS and K be a natural number. We recall that a state $g_{K+1} \in G$ is $[K,1]$-reachable by $\mathcal{A}$ iff there is a sequence of states $g_1,\ldots,g_K \in G$, a sequence of stack configurations $w_1,\ldots,w_{K+1} \in \Gamma \cup \{\varepsilon\}$, and a sequence of multi-sets $M_1,\ldots,M_{K+1} \in M[(\Gamma \cup \{\varepsilon\}) \times \{0,\ldots,K+1\}]$ such that:

$$\langle g_0,(\gamma_0,0),\emptyset\rangle \Rightarrow^*_0 \langle g_1,(w_1,1),M_1\rangle \Rightarrow^*_1 \cdots \Rightarrow^*_K \langle g_{K+1},(w_{K+1},K+1),M_{K+1}\rangle \quad (a)$$

The problem of checking whether a state g_{K+1} is $[K,1]$-reachable by $\mathcal{A}$ can be rewriting as follows: Whether there are some $w_0,\ldots,w_{K+1} \in \Gamma \cup \{\varepsilon\}$, $g_1,\ldots,g_K \in G$, and $N_j,N'_j \in M[\Gamma \times \{j\}]$ for every $j \in \{0,\ldots,K+1\}$, such that: (1) $w_0 = \gamma_0$, (2) $N_0 = N'_0 = \emptyset$, (3) and for every $i \in \{0,\ldots,K\}$, we have:

$$\langle g_i,(w_i,i),N'_i + \textstyle\sum_{l=0}^{i-1}(N'_l - N_l)\rangle \Rightarrow^*_i \langle g_{i+1},(w_{i+1},i+1),N'_{i+1} + \sum_{l=0}^{i}(N'_l - N_l)\rangle \quad (b)$$

Observe that for every $i \in \{0,\ldots,K+1\}$, we have that $M_i = N'_i + \sum_{l=0}^{i-1}(N'_l - N_l)$ and that $N_i + [(w_i,i)]$ is the multi-set of executed threads with switch number i. Then, the equation (b) can be rewritten as follows: For every $i \in \{0,\ldots,K\}$, we have:

$$\langle g_i,(w_i,i),N_i\rangle \Rightarrow^*_i \langle g_{i+1},(w_{i+1},i+1),N'_{i+1}\rangle \text{ and } N_i \leq N'_i \qquad (c)$$

This means that $\mathcal{A}$ can reach the configuration $\langle g_{i+1},(w_{i+1},i+1),N'_{i+1}\rangle$ while executing all threads in the multiset $[(w_i,i)] + N_i$ which should be less than the number of generated threads with switch number i, i.e. $[(w_i,i)] + N'_i$. We can further simplify our problem as follows: A state g_{K+1} is $[K,1]$-reachable by $\mathcal{A}$ iff there are some $w_0,w'_0,w_1,\ldots,w_{K+1},w'_{K+1} \in \Gamma \cup \{\varepsilon\}$, $g_1,\ldots,g_K \in G$, and $N_j,N''_j \in M[\Gamma \times \{j\}]$ for every $j \in \{0,\ldots,K+1\}$, such that: (1) $w_0 = w'_0 = \gamma_0$, (2) $N_0 = N''_0 = \emptyset$, (3) and for every $i \in \{0,\ldots,K\}$, we have:

$$\langle g_i,(w_i,i),N_i\rangle \Rightarrow^*_i \langle g_{i+1},(w'_{i+1},i+1),N''_{i+1}\rangle \text{ and } N_i + [(w_i,i)] \leq N''_i + [(w'_i,i)] \quad (d)$$

Observe that if (c) holds, then (d) holds by simply considering $w'_i = w_i$ for every $i \in \{0,\ldots,K+1\}$. On the other hand, if (d) is true, then (c) is true by taking $N''_{K+1} = N'_{K+1}$, $w_{K+1} = w'_{K+1}$, and for every $i \in \{0,\ldots,K\}$, $N'_i = N''_i + [(w'_i,i)] - [(w_i,i)]$ which possible since $N_i + [(w_i,i)] \leq N''_i + [(w'_i,i)]$. In the latter case, we can show that for every $i \in \{0,\ldots,K\}$, $\langle g_i,(w_i,i),N_i\rangle \Rightarrow_i^* \langle g_{i+1},(w_{i+1},i+1),N'_{i+1}\rangle$ and $N_i \leq N'_i$.

Hence, a state g_{K+1} is $[K,1]$-reachable iff the three following requirements hold:

1. For each $i \in \{0,\ldots,K\}$, we have $\langle g_i,(w_i,i),N_i\rangle \Rightarrow_i^* \langle g_{i+1},(w'_{i+1},i+1),N''_{i+1}\rangle$.
2. For each $i \in \{1,\ldots,K\}$, we have $N_i + [(w_i,i)] \leq N''_i + [(w'_i,i)]$.
3. $N_0 = N''_0 = \emptyset$ and $w_0 = w'_0 = \gamma_0$.

(Notice indeed that requirements 1, 2 and 3 are equivalent to (d)).

Let us fix (guess) a sequence of states $\sigma = g_1,\ldots,g_K \in G$. We show how to compute an existential Presburger formula ϕ_σ, of polynomial size in the size of $\mathcal{A}$, which is satisfiable iff the state g_{K+1} is $[K,1]$-reachable by $\mathcal{A}$. To this end, we compute an existential Presburger sub-formula for each of the previous three requirements. These sub-formulas use the set $\mathcal{V} = \{x_{(w,i)}, y_{(w,i)} \mid w \in (\Gamma \cup \{\varepsilon\}) \wedge 0 \leq i \leq K+1\}$ as a set of free variables such that for each $i \in \{0,\ldots,K+1\}$ and for each $w \in \Gamma \cup \{\varepsilon\}$, the variable $x_{(w,i)}$ (resp. $y_{(w,i)}$) stands for the number of occurrences of (w,i) in the multiset $\big([(w_i,i)]+N_i\big)$ (resp. $\big([(w'_i,i)]+N''_i\big)$), i.e. $\big([(w_i,i)]+N_i\big)\big((w,i)\big)$ (resp. $\big([(w'_i,i)]+N''_i\big)\big((w,i)\big)$). Then, the formula ϕ_σ is obtained as the conjunction of these subformulas computed as follows:

- Checking the first requirement: For each $i \in \{0,\ldots,K\}$ and for each $g_i, g_{i+1} \in G$, we compute a formula $\phi_{(i,g_i,g_{i+1})}$ such that $\phi_{(i,g_i,g_{i+1})}(U)$ is true iff there are $w_i, w'_{i+1} \in \Gamma \cup \{\varepsilon\}$ and $N_i \in M[(\Gamma \cup \{\varepsilon\}) \times \{i\}]$, and $N''_{i+1} \in M[(\Gamma \cup \{\varepsilon\}) \times \{i+1\}]$ such that: (1) $\langle g_i,(w_i,i),N_i\rangle \Rightarrow_i^* \langle g_{i+1},(w'_{i+1},i+1),N''_{i+1}\rangle$, (2) $\big([(w_i,i)]+N_i\big)\big((w,i)\big) = U(x_{(w,i)})$, and (3) $\big([(w'_{i+1},i+1)]+N''_{i+1}\big)\big((w,i+1)\big) = U(y_{(w,i+1)})$ for all $w \in \Gamma \cup \{\varepsilon\}$.

To this end, we prove that the set of valuation $U : \mathcal{V} \to \mathbb{N}$, such that there is a computation $\langle g_i,(w_i,i),N_i\rangle \Rightarrow_i^* \langle g_{i+1},(w'_{i+1},i+1),N''_{i+1}\rangle$ of $\mathcal{A}$, where for every $w \in \Gamma \cup \{\varepsilon\}$, $\big([(w_i,i)]+N_i\big)\big((w,i)\big) = U(x_{(w,i)})$ and $\big([(w'_{i+1},i+1)]+N''_{i+1}\big)\big((w,i+1)\big) = U(y_{(w,i+1)})$, can be defined as the Parikh image of a language accepted by a finite state automaton $\mathcal{S}_{(i,g_i,g_{i+1})}$ with the input alphabet $(\Gamma \cup \{\varepsilon\}) \times \{i,i+1\}$. Then, we use Theorem 4 to construct the formula $\phi_{(i,g_i,g_{i+1})}$. The automaton $\mathcal{S}_{(i,g_i,g_{i+1})}$ has the following structure:

- The set of states:
  - $\mathcal{S}_{(i,g_i,g_{i+1})}$ has two special states: s^{init} as an initial state and s^{final} as a final state.
  - For each $g \in G$, the automaton $\mathcal{S}_{(i,g_i,g_{i+1})}$ has a state g^c. This represents that the current value of the global store of $\mathcal{A}$ is g when a context switch occurs.
  - For each $g \in G$ and for each $w \in \Gamma \cup \{\varepsilon\}$, the automaton $\mathcal{S}_{(i,g_i,g_{i+1})}$ has a state (g,w). This corresponds to the fact that the current value of the global store of $\mathcal{A}$ is g and the local configuration of the active thread is (w,i).
- The set of transitions:
  - **Initialization** For each $w_i \in \Gamma \cup \{\varepsilon\}$, the automaton $\mathcal{S}_{(i,g_i,g_{i+1})}$ has the transition $s^{init} \xrightarrow{(w_i,i)} (g_i,w_i)$. This transition corresponds to a guess of the local configuration of the first active thread (w_i,i).

- **Simulation of a transition:** For each rule $g\gamma \hookrightarrow g'w'$ (resp. $g\gamma \hookrightarrow g'w' \rhd \gamma'$) of $\mathcal{A}$, $\mathcal{S}_{(i,g_i,g_{i+1})}$ has the transition $(g,\gamma) \xrightarrow{(\gamma',i+1)} (g',w')$ (resp. $(g,\gamma) \xrightarrow{\varepsilon} (g',w')$).
- **Simulation of a context switch** For each state $g \in G$ and each pair of stack configurations w and w' in $\Gamma \cup \{\varepsilon\}$, the automaton $\mathcal{S}_{(i,g_i,g_{i+1})}$ has the transitions $(g,w) \xrightarrow{(w,i+1)} g^c$ and $g^c \xrightarrow{(w',i)} (g,w')$. These transitions simulate a context switch between two threads with local configurations (w,i) and (w',i).
- **End of the simulation:** For each $w \in \Gamma \cup \{\varepsilon\}$, the automaton $\mathcal{S}_{(i,g_i,g_{i+1})}$ has the transition $(g_{i+1},w) \xrightarrow{(w,i+1)} s^{final}$. This corresponds to the interruption of the thread with local configuration (w,i) and to the end of the simulation.

It can be checked that the size of the automaton $\mathcal{S}_{(i,g_i,g_{i+1})}$ is polynomial in the size of $\mathcal{A}$. The relation between $\mathcal{S}_{(i,g_i,g_{i+1})}$ and $\mathcal{A}$ is given by the following lemma:

Lemma 3. *A word u is in $L(\mathcal{S}_{(i,g_i,g_{i+1})})$ iff there are $w_i, w'_{i+1} \in (\Gamma \cup \{\varepsilon\})$, $N_i \in M[(\Gamma \cup \{\varepsilon\}) \times \{i\}]$, and $N''_{i+1} \in M[(\Gamma \cup \{\varepsilon\}) \times \{i+1\}]$ such that: (1) $\langle g_i,(w_i,i),N_i\rangle \Rightarrow_i^* \langle g_{i+1},(w'_{i+1},i+1),N''_{i+1}\rangle$, (2) $\big([(w_i,i)]+N_i\big)\big((w,i)\big) = [u]((w,i))$, and (3) $\big([(w'_{i+1},i+1)]+N''_{i+1}\big)\big((w,i+1)\big) = [u]((w,i+1))$ for every stack configuration $w \in \Gamma \cup \{\varepsilon\}$.*

As a consequence of Theorem 4 and Lemma 3, it is possible to compute in polynomial time and size an existential Presburger formula $\phi_{(i,g_i,g_{i+1})}$ that represents the Parikh image of the language $L(\mathcal{S}_{(i,g_i,g_{i+1})})$.

Lemma 4. *For every valuation U, $\phi_{(i,g_i,g_{i+1})}(U)$ is true iff there are $w_i, w'_{i+1} \in (\Gamma \cup \{\varepsilon\})$, $N_i \in M[(\Gamma \cup \{\varepsilon\}) \times \{i\}]$, and $N''_{i+1} \in M[(\Gamma \cup \{\varepsilon\}) \times \{i+1\}]$ such that: (1) $\langle g_i,(w_i,i),N_i\rangle \Rightarrow_i^* \langle g_{i+1},(w'_{i+1},i+1),N''_{i+1}\rangle$, (2) $\big([(w_i,i)]+N_i\big)\big((w,i)\big) = U(x_{(w,i)})$, and (3) $\big([(w'_{i+1},i+1)]+N''_{i+1}\big)\big((w,i+1)\big) = U(y_{(w,i+1)})$ for every $w \in \Gamma \cup \{\varepsilon\}$.*

- Checking the second requirement: The requirement that $N_i + [(w_i,i)] \leq N''_i + [(w'_i,i)]$, for every $i \in \{1,\ldots,K\}$, can be expressed by $\phi_{(i,2)} = \bigwedge_{w \in (\Gamma \cup \{\varepsilon\})} x_{(w,i)} \leq y_{(w,i)}$.

- Checking the third requirement: To express the requirements that $N_0 = N''_0 = \emptyset$ and $w_0 = w'_0 = \gamma_0$, we consider the formula ϕ_3 as the conjunction of the formulas: $x_{(\gamma_0,0)} = y_{(\gamma_0,0)} = 1$ and $x_{(w,0)} = y_{(w,0)} = 0$ for all $w \in \Gamma \cup \{\varepsilon\}$ such that $w \neq \gamma_0$.

Finally, the formula ϕ_σ is obtained as $\phi_3 \wedge \big(\bigwedge_{i \in \{0,\ldots,K\}} (\phi_{(i,g_i,g_{i+1})} \wedge \phi_{(i,2)})\big)$. It can be checked that the size of ϕ_σ is polynomial in K and in the size of $\mathcal{A}$. □

5 Analysis of Dynamic Networks of Concurrent Pushdown Systems

We consider now the case of DCPSs. It is well-known that the SRP is undecidable already for networks with two concurrent pushdown processes. We prove however that both problems SRP$[K]$ and SRP$[K,L]$ are decidable, for any given bounds K and L. For that, we prove the following fact.

Theorem 7. *For every $K,L \in \mathbb{N}$, the problems SRP$[K]$ and the SRP$[K,L]$ for DCPS are exponentially reducible to the corresponding problems for DCFS.*

The rest of this section is devoted to the proof of Theorem 7. Let us fix a DCPS $\mathcal{A} = (G,\Gamma,\Delta,g_0,\gamma_0)$. We show that it is possible to construct a DCFS $\mathcal{A}_{fs}$ such that

the problems SRP$[K]$ and SRP$[K,L]$ for $\mathcal{A}$ can be reduced to the corresponding problems for $\mathcal{A}_{fs}$. Let us present the main steps of this construction. For that, let us consider the problem SRP$[K]$, for some fixed $K \in \mathbb{N}$. Then, let us concentrate on the computations of one thread, and assume that this thread will be interrupted and resumed i times (with $i \leq K$) during its execution from some initial global state g and initial local state γ to some final global state g'. The computations of such a thread correspond to the run of a labeled pushdown system, built out of $\mathcal{A}$, which (1) performs the same operations on the stack and global states as the ones specified by Δ, (2) makes visible as transition labels the local state (element of Γ) of the spawned threads, and (3) nondeterministically guesses jumps from a global state to another one corresponding to the effect of context switches. These jumps are also made visible as transition labels under the form of pairs $(g_l, g_{l+1}) \in G \times G$ (meaning that the computation of the thread is interrupted at state g_l and is resumed at state g_{l+1}). The number of such jumps in each run is precisely i.

Then, the problem is to handle the composition of all the computations (unbounded number) of the generated threads and to make sure that the guesses made by each one of them (on their control state jumps due to context switches) are correct. The key observation which allows to solve this problem is that it is possible to assume without loss of preciseness that some of the generated threads can be ignored (or lost). Indeed, these threads can always be considered as threads which will never be scheduled. Therefore, the behaviors of each thread can be modeled using a finite-state automaton which recognizes the downward closure of the language of the labeled pushdown system of a thread w.r.t. the ordering on words where u is less than v if u can be obtained from v by erasing symbol in Γ. We know by Theorem 1 that this automaton is effectively constructible. So, let $\mathcal{S}_{(i,g,\gamma,g')}$ be the automaton modeling the computations of threads starting from g and γ and reaching g' after i interruptions-resumptions.

The next step is to synchronize the so-defined finite-state automata in order to represent valid computations of the whole system. For that, we define a DCFS $\mathcal{A}_{fs}$ which simulates the composition of these automata as follows:

- Assume that the initial global state is g_0 and that the initial thread has an starting local state γ_0. Then, $\mathcal{A}_{fs}$ guesses for this thread the number of switches i and its final state g, and starts simulating its behaviors according to the transitions of the automaton $\mathcal{S}_{(i,g_0,\gamma_0,g)}$. To initialize the simulation, $\mathcal{A}_{fs}$ has a rule $g_0\gamma_0 \hookrightarrow \$s^{init}_{(i,g_0,\gamma_0,g)}$, where $s^{init}_{(i,g_0,\gamma_0,g)}$ is the initial state of $\mathcal{S}_{(i,g_0,\gamma_0,g)}$. This rule allows to check that the control state is g_0 and to move to a special control state \$ corresponding to a simulation phase without context switches.
- During the simulation, when a transition $s \xrightarrow{\gamma} s'$ is encountered, a new thread γ is spawned by $\mathcal{A}_{fs}$. This is done using a rule $\$s \hookrightarrow \$s' \rhd \gamma$. The new thread will stay pending until $\mathcal{A}_{fs}$ can dispatch it.
- A pending thread γ which has never been activated can be dispatched by $\mathcal{A}_{fs}$ at the moment of a context switch. For that, $\mathcal{A}_{fs}$ has a rule $g\gamma \hookrightarrow \$s^{init}_{(i,g,\gamma,g')}$ where $s^{init}_{(i,g,\gamma,g')}$ is the initial state of $\mathcal{S}_{(i,g,\gamma,g')}$, for every possible starting and ending states g and g', and every possible number of context switches $i \leq K$.
- Encountering a transition $s \xrightarrow{(g_1,g_2)} s'$ means that the computation of the simulated thread has lead to the global store g_1, and that this computation will be interrupted

at this point and will be resumed later when the global store will become g_2 (due to the execution of some other threads). Then, $\mathcal{A}_{fs}$ moves from its control state \$ to a control state g_1 so that the control can be taken by another thread (which was waiting for g_1), and transforms the local state of the current thread (which may be interrupted) to (g_2, s'). Both of these operations are done using a rule $\$s \hookrightarrow g_1(g_2, s')$. In the case of $g_1 \neq g_2$, we observe that the only action that can be done by $\mathcal{A}_{fs}$ after executing this rule is a context switch, i.e., (g_2, s') becomes idle and some pending thread is activated (either dispatched for the first time, or resumed after some interruption). We have seen above how $\mathcal{A}_{fs}$ dispatches pending threads for the first time. The resumption of threads at control state g_1 is done by having rules of the form $g_1(g_1, s'') \hookrightarrow \$s''$ for all possible states s'' in $\mathcal{S}_{(i,g,\gamma,g')}$. Such a rule means that if a pending thread (g_1, s'') exists, then it can be resumed and the simulation of its behaviors is pursued from the state s'' (at which it was stopped at the last interruption). Similarly, (g_2, s') will be resumed when the rule $g_2(g_2, s') \hookrightarrow \$s'$ can be executed which can only happen if the global store becomes g_2.

– Finally, when a final state $s^{final}_{(i,g,\gamma,g')}$ of $\mathcal{S}_{(i,g,\gamma,g')}$ is reached, this means that the simulation of the current thread has been completed and therefore the global store must be g' (the guessed target state) at this point. Then, the execution of the rule $\$s^{final}_{(i,g,\gamma,g')} \hookrightarrow g'\bot$ allows to release the control so that some pending thread waiting for g' can be resumed.

Let us give in more details the construction described above.

5.1 Simulating Threads with Finite-State Automata

First, we define the DCPS $\mathcal{A}_{los}$ obtained from $\mathcal{A}$ by allowing losses of the generated threads. Let $\mathcal{A}_{los} = (G, \Gamma, \Delta_{los}, g_0, \gamma_0)$ be the DCPS such that $\Delta_{los} = \Delta \cup \{g\gamma \hookrightarrow g'w' \mid (g\gamma \hookrightarrow g'w' \rhd \gamma') \in \Delta\}$.

Lemma 5. *For every $K, L \in \mathbb{N}$, a control state $g \in G$ is K-reachable (resp. $[K,L]$-reachable) in $\mathcal{A}$ if and only if g is K-reachable (resp. $[K,L]$-reachable) in $\mathcal{A}_{los}$.*

Next, we show the construction of the automaton $\mathcal{S}_{(i,g,\gamma,g')}$ for some given $i \in \{0, \ldots, K\}$, $\gamma \in \Gamma$, and $g, g' \in G$. For that, we start by considering a labeled pushdown system simulating the behaviors of thread that reaches the state g' starting from g and the stack configuration γ after some number of jumps in the control state (representing guesses on the effect of context switches), The spawned thread as well as the guesses on the control state jumps made during the computation are made visible as labels on the transitions. Let $\mathcal{P} = (G, \Gamma \cup G \times G, \Gamma, \Delta_{\mathcal{P}})$ be the labeled pushdown system where $\Delta_{\mathcal{P}}$ is the smallest set of rule such that (1) for every $g_1\gamma_1 \hookrightarrow g_2 w \rhd \gamma_2$ (resp. $g_1\gamma_1 \hookrightarrow g_2 w$) in Δ_{los}, the rule $g_1\gamma_1 \overset{\gamma_2}{\hookrightarrow} g_2 w$ (resp. $g_1\gamma_1 \overset{\varepsilon}{\hookrightarrow} g_2 w$) is in $\Delta_{\mathcal{P}}$, and (2) for every $(g_1, g_2) \in G \times G$, and for every $\gamma_1 \in \Gamma$, the rule $g_1\gamma_1 \overset{(g_1,g_2)}{\hookrightarrow} g_2\gamma_1$ is in $\Delta_{\mathcal{P}}$.

Then, the set of behaviors represented by this labeled pushdown system which correspond to precisely i control switches (interruption-resumptions) is

$$L_{(i,g,\gamma,g')} = L_{\mathcal{P}}(g\gamma, g') \cap (\Gamma^* \cdot (G \times G) \cdot \Gamma^*)^i.$$

This set is context-free in general (since it is the intersection of a context-free language with a regular one). However, due to Lemma 5, we can consider without loss of preciseness the downward closure of $L_{(i,g,\gamma,g')}$ w.r.t. the sub-word relation corresponding to the deletion of symbols in Γ while preserving all symbols in $G \times G$, i.e., the set

$$L_{(i,g,\gamma,g')}\downarrow \cap (\Gamma^* \cdot (G \times G) \cdot \Gamma^*)^i.$$

By Theorem 1, this set regular an can be effectively represented by a finite-state automaton $\mathcal{S}_{(i,g,\gamma,g')} = (S_{(i,g,\gamma,g')}, \Gamma \cup G \times G, \delta_{(i,g,\gamma,g')}, s^{init}_{(i,g,\gamma,g')}, s^{final}_{(i,g,\gamma,g')})$. We assume w.l.o.g that all states in the automaton $\mathcal{S}_{(i,g,\gamma,g')}$ are co-reachable from the final state.

Lemma 6. *Let $i \in \mathbb{N}$ be a natural number, $\gamma \in \Gamma$ be a stack symbol, $g_1, g'_1, g_2, \ldots, g_{i+1}, g'_{i+1} \in G$ be a sequence of states, and $w_0, \ldots, w_i \in \Gamma^*$ be a sequence of stack contents. Then, $w_0(g'_1,g_2)w_1(g'_2,g_3)w_2 \cdots (g'_i,g_{i+1})w_i$ is accepted by $\mathcal{S}_{(i,g_1,\gamma,g'_{i+1})}$ iff there are $u_0, \ldots, u_{i+1} \in \Gamma^*$ such that, for every $l \in \{1,\ldots,i+1\}$ and $j \in \mathbb{N}$, $\langle g_l, (u_{l-1}, j), \emptyset\rangle \Rightarrow_j^* \langle g'_l, (u_l, j), M_{l-1}\rangle$ is a computation of $\mathcal{A}_{los}$ where $u_0 = \gamma$ and $M_{l-1}((\gamma', j+1)) = [w_{l-1}](\gamma')$ for all $\gamma' \in \Gamma$.*

The lemma above says that a word $w_0(g'_1,g_2)w_1(g'_2,g_3)w_2 \cdots (g'_i,g_{i+1})w_i$ is in $L(\mathcal{S}_{(i,g_1,\gamma,g'_{i+1})})$ iff the DCPS $\mathcal{A}_{los}$ is able to bring the value of the global variables from g_l to g'_l and the stack configuration of the simulated thread from u_{l-1} to u_l while creating the set of threads with initial stack symbols in $[w_{l-1}]$ for every $l \in \{1,\ldots,i+1\}$.

5.2 From DCPS to DCFS

We define the DCFS $\mathcal{A}_{fs} = (G_{fs}, \Gamma_{fs}, \Delta_{fs}, g_0, \gamma_0)$ where:

- $G_{fs} = G \cup \{\$\}$ is a finite set of states with $\$ \notin G$.
- Γ_{fs} is a finite set of stack alphabet defined as the union of the sets $\Gamma \cup \{\bot\}$, $S_{(i,g,\gamma,g')}$ and $G \times S_{(i,g,\gamma,g')}$ for all $(i,g,\gamma,g') \in \{0,\ldots,K\} \times G \times \Gamma \times G$, where $S_{(i,g,\gamma,g')}$ is the set of states of $\mathcal{S}_{(i,g,\gamma,g')}$.
- Δ_{fs} is the smallest set of transitions such that
  - *Initialize/Disptach:* For every $i \in \{0,\ldots,K\}$, every $\gamma \in \Gamma$, and every $g, g' \in G$, the rule $g\gamma \hookrightarrow \$s^{init}_{(i,g,\gamma,g')}$ is in Δ_{fs} where $s^{init}_{(i,g,\gamma,g')}$ is the initial state of $\mathcal{S}_{(i,g,\gamma,g')}$.
  - *Skip:* For every transition $s \xrightarrow{\varepsilon} s'$ of $\mathcal{S}_{(i,g,\gamma,g')}$, the rule $\$s \hookrightarrow \$s'$ is in Δ_{fs}.
  - *Spawn:*For every transition $s \xrightarrow{\gamma} s'$ of $\mathcal{S}_{(i,g,\gamma,g')}$, the rule $\$s \hookrightarrow \$s' \rhd \gamma$ is in Δ_{fs}.
  - *Interrupt:* For every transition $s \xrightarrow{(g_1,g_2)} s'$ of $\mathcal{S}_{(i,g,\gamma,g')}$, the rule $\$s \hookrightarrow g_1(g_2,s')$ is in Δ_{fs}.
  - *Resume:* For every $(g,s) \in \Gamma_{fs}$, the rule $g(g,s) \hookrightarrow \$s$ is in Δ_{fs}.
  - *Terminate:* The rule $\$s^{final}_{(i,g,\gamma,g')} \hookrightarrow g'\bot$ is in Δ_{fs}, where $s^{final}_{(i,g,\gamma,g')}$ is the final state of $\mathcal{S}_{(i,g,\gamma,g')}$.

Lemma 7. *For every $K, L \in \mathbb{N}$, a control state g is K-reachable (resp. $[K,L]$-reachable) in $\mathcal{A}$ if and only if g is K-reachable (resp. $[K,L]$-reachable) in $\mathcal{A}_{fs}$.*

The proof of the lemma above is technical and is given in details in [1]. We give hereafter a high level description of it. Let us consider a DCPS $\mathcal{A}_+$ which is *the union* of

$\mathcal{A}_{los}$ and $\mathcal{A}_{fs}$ in the sense that for each created thread with initial configuration $\gamma \in \Gamma$, $\mathcal{A}_+$ chooses nondeterministically whether the thread will be executed according to the rules of $\mathcal{A}_{los}$, or simulated according to the rules of $\mathcal{A}_{fs}$. Then, we define the rank of a computation of $\mathcal{A}_+$ to be the pair $(m,n) \in \mathbb{N} \times \mathbb{N}$ where m is the number of threads involved in the computation that follow the rules of $\mathcal{A}_{los}$ and n is the number of threads in the computation following the rules of $\mathcal{A}_{fs}$. Observe that computations of rank (m,n) where $n = 0$ (resp. $m = 0$) are precisely the computations of $\mathcal{A}_{los}$ (resp. $\mathcal{A}_{fs}$). We prove that for any computation of $\mathcal{A}_+$ of rank $(m+1,n)$ (resp. $(m,n+1)$), there exists a computations of $\mathcal{A}_+$ of rank $(m,n+1)$ (resp. $(m+1,n)$). This computation is obtained from the original one by simulating the execution of a thread that follows the rules of $\mathcal{A}_{los}$ (resp. $\mathcal{A}_{fs}$) by a thread that follows the rules of $\mathcal{A}_{fs}$ (resp. $\mathcal{A}_{los}$). This is possible thanks to Lemma 6. A consequence of this fact is that, for every $m \in \mathbb{N}$, a control state is K-reachable (resp. $[K,L]$-reachable) by a computation of rank $(m,0)$ (i.e., by a computation of $\mathcal{A}_{los}$ with m threads) if and only if it is K-reachable (resp. $[K,L]$-reachable) by a computation of rank $(0,m)$ (i.e. by a computation of $\mathcal{A}_{fs}$ with the m threads). This is precisely what Lemma 7 is saying.

Finally, Theorem 7 is an immediate consequence of Lemma 7. A corollary of Theorem 7 and Theorem 5 is the following fact.

Corollary 1. *For every $K \in \mathbb{N}$, the problem SRP$[K]$ for DCPS is in 2-EXPSPACE, and for every $K,L \in \mathbb{N}$, the problem SRP$[K,L]$ for DCPS is in NEXPTIME.*

6 Conclusion

We have proposed new concepts for context-bounded verification we believe that are natural and suitable for programs with dynamic thread creation. These concepts are based on the idea of bounding the number of switches for each thread and not for all the threads in a computation.

First, we have proved that even for finite-state threads, adopting such a notion of context-bounding leads in general to a problem which is as hard as the coverability problem of Petri nets. This means that, in theory, the complexity of this problem is high, but in practice, there are quite efficient techniques (based on iterative computation of under/upper approximations) developed recently for solving this problem which have been implemented and used successfully in [5,4]. Moreover, we have proposed a notion of stratified context-bounding for which the verification is in NP, i.e., as hard as in the case without dynamic thread creation. An interesting question is how to implement efficiently the analysis in this case using clever encodings in SMT solvers.

Moreover, we have proved that the considered problems are still decidable for the case of pushdown threads. This is done by a nontrivial reduction to the corresponding problems for finite-state threads. This reduction is based on computing the regular downward closure of context-free languages w.r.t. the sub-word relation. The downward closure computation may lead in general to an unavoidable exponential blow-up. This is due to the succinctness of context-free grammars w.r.t. finite state automata: For instance, the finite language $\{a^{2^N}\}$, for a fixed $N \geq 1$, can be defined with a context-free grammar of size N whereas a finite-state automaton representing it (or its downward

closure) is necessarily of size at least 2^N. An interesting open problem is whether there is an alternative proof technique which allows to avoid the downward closure construction. In practice, we believe that it would be possible to overcome this problem by for instance designing algorithms allowing to generate efficiently and incrementally (parts of the) downward closure.

Finally, in our models, we consider that each created thread inherits a switch number from its father (the one of its father plus 1). An alternative definition can be obtained by considering that each created thread is given the switch number 0. (Therefore, each thread can perform up to K switches.) For that model, we can prove (for more details see [1]) that our results concerning the reachability problems SRP and SRP$[K]$ hold with the same complexity bounds. However, the problem SRP$[K,L]$ for finite state threads (resp. pushdown threads) becomes EXPSPACE-complete (in 2-EXPSPACE) instead of NP-complete (NEXPTIME) for this definition.

References

1. Atig, M.F., Bouajjani, A., Qadeer, S.: Context-bounded analysis for concurrent programs with dynamic creation of threads. Technical report, LIAFA (October 2008)
2. Bouajjani, A., Esparza, J., Schwoon, S., Strejcek, J.: Reachability analysis of multithreaded software with asynchronous communication. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 348–359. Springer, Heidelberg (2005)
3. Courcelle, B.: On construction obstruction sets of words. In: EATCS 1991, vol. 44, pp. 178–185 (1991)
4. Ganty, P., Raskin, J.F., Begin, L.V.: A complete abstract interpretation framework for coverability properties of WSTS. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 49–64. Springer, Heidelberg (2005)
5. Geeraerts, G., Raskin, J.F., Begin, L.V.: Expand, enlarge and check: New algorithms for the coverability problem of WSTS. J. Comput. Syst. Sci. 72(1), 180–203 (2006)
6. Lal, A., Reps, T.W.: Reducing concurrent analysis under a context bound to sequential analysis. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 37–51. Springer, Heidelberg (2008)
7. Lal, A., Touili, T., Kidd, N., Reps, T.W.: Interprocedural analysis of concurrent programs under a context bound. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 282–298. Springer, Heidelberg (2008)
8. Lipton, R.: The reachability problem requires exponential time. Technical Report TR 66 (1976)
9. Musuvathi, M., Qadeer, S.: Iterative context bounding for systematic testing of multithreaded programs. In: PLDI 2004, pp. 446–455. ACM Press, New York (2007)
10. Qadeer, S., Rehof, J.: Context-bounded model checking of concurrent software. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 93–107. Springer, Heidelberg (2005)
11. Qadeer, S., Wu, D.: KISS: keep it simple and sequential. In: PLDI 2004, pp. 14–24. ACM, New York (2004)
12. Rackoff, C.: The covering and boundedness problem for vector addition systems. Theoretical Computer Science (1978)
13. Ramalingam, G.: Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. Program. Lang. Syst. 22(2), 416–430 (2000)

14. Seidl, H., Schwentick, T., Muscholl, A., Habermehl, P.: Counting in trees for free. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 1136–1149. Springer, Heidelberg (2004)
15. Stewart, I.A.: Reachability in some classes of acyclic petri nets. Fundam. Inform. 23(1), 91–100 (1995)
16. Suwimonteerabuth, D., Esparza, J., Schwoon, S.: Symbolic context-bounded analysis of multithreaded java programs. In: Havelund, K., Majumdar, R., Palsberg, J. (eds.) SPIN 2008. LNCS, vol. 5156, pp. 270–287. Springer, Heidelberg (2008)
17. Verma, K.N., Seidl, H., Schwentick, T.: On the complexity of equational Horn clauses. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS, vol. 3632, pp. 337–352. Springer, Heidelberg (2005)
18. Zaks, A., Joshi, R.: Verifying multi-threaded C programs with SPIN. In: Havelund, K., Majumdar, R., Palsberg, J. (eds.) SPIN 2008. LNCS, vol. 5156, pp. 325–342. Springer, Heidelberg (2008)