Model Checking Dynamic Pushdown Networks with Locks and Priorities

Marcio Diaz, Tayssir Touili

Model Checking Dynamic Pushdown Networks with Locks and Priorities

Marcio Diaz[1] and Tayssir Touili[2]

[1] LIPN and University Paris Diderot, France
diaz@lipn.univ-paris13.fr
[2] LIPN, CNRS and University Paris 13, France
touili@lipn.univ-paris13.fr

Abstract. A dynamic pushdown network (DPN) is a set of pushdown systems (PDSs) where each process can dynamically create new instances of PDSs. DPNs are a natural model of multi-threaded programs with (possibly recursive) procedure calls and thread creation. A PL-DPN is an extension of DPNs that allows threads to synchronize using locks and priorities. Transitions in a PL-DPN can have different priorities and acquire/release locks. We consider in this work model checking PL-DPNs against single-indexed LTL and CTL properties of the form $\bigwedge f_i$ such that f_i is a LTL/CTL formula over the PDS i. We show that these model checking problems are decidable. We propose automata-based approaches for computing the set of configurations of a PL-DPN that satisfy the corresponding single-indexed LTL/CTL formula.

1 Introduction

Writing multi-threaded programs is notoriously difficult, as concurrency related bugs are hard to find and reproduce. This difficulty is increased if we consider that several software systems consist of different components that react to the environment and use resources like CPU or memory according to a real time need. For instance, in systems that control automobiles we can have a component in charge of the music sub-system and another component in charge of the braking sub-system. Obviously, the braking sub-system should have a higher priority access to the resources needed, since a delay in the action of the brakes can cost lives.

The programming model used in the vast majority of these software systems, used from automobiles to spacecrafts, defines a set of threads that perform computation monitoring or respond to events. Each thread is typically assigned a priority and are scheduled by a priority round-robin preemptive scheduler: if a thread with a higher static priority becomes ready to run, the currently running thread will be preempted and returned to the wait list for its priority level. The round-robin scheduling policy allows each thread to run only for a fixed amount of time before it must yield its processing slot to another thread of the same priority.

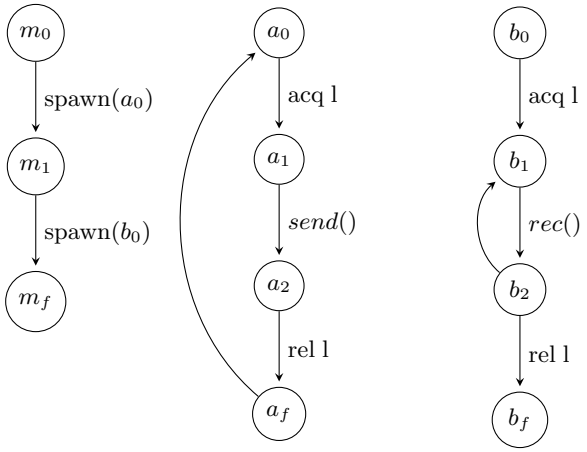

Fig. 1. Control-flow graph of a main thread starting at m_0 that creates two child threads starting at a_0 and b_0. The child threads execute a loop and use a lock l.

The use of threads with different priorities and other synchronization primitives, like locks, can easily lead to a large number of undesirable behaviors. Consider for example the control flow graph of Figure 1. It consists of three threads: a main thread M starting at control location m_0 that creates two threads A and B. The thread A takes and releases (uses) a lock l inside a loop, and the thread B loops while holding the lock l (between b_1 and b_2). Suppose that A and B should act like daemon threads, continuously running in the background reacting to events. This means that the LTL formula $G\ F\ a_1 \wedge G\ F\ b_1$, saying that a_1 *is executed frequently often and* b_1 *is executed frequently often*, should be valid for all executions.

But this is not the case in the program of Figure 1: once thread B starts executing its loop and holding lock l, thread A is going to starve since it cannot take the lock until it is released. A similar problem occurs if threads A and B have different priority of execution, the thread with lower priority will starve. The program of Figure 1 shows that there is a real need for formal methods to find automatic verification techniques for checking *liveness properties* in *multi-threaded programs with locks and priorities*. Indeed, starvation or absence of livelocks are among the most crucial properties that need to be checked for multi-threaded programs.

Dynamic pushdown networks (DPNs) [13] are a natural model for multi-threaded programs with (possibly recursive) procedure calls and thread creation. A DPN consists of a finite set of pushdown systems (PDSs), each of them models a sequential program that can dynamically create new instances of PDSs. The model-checking problems of DPNs against Linear Temporal Logic (LTL), Computation Tree Logic (CTL) and reachability properties are well studied in the literature [6–10, 13].

However, DPNs cannot model communication between processes. Previous works [5, 16, 24, 25] extended DPNs with locks and priorities (called PL-DPN),

where PDSs can communicate using locks and priorities. This allows to model multithreaded programs where threads communicate via locks and where each thread can have a different priority. Indeed, locks and priorities are frequently used in multithreaded programs as synchronisation primitives. However, only reachability properties are studied for PL-DPNs [25] with some restricted lock and priority usages.

In general, the model checking problem of DPNs against unrestricted LTL or CTL formulas (where atomic propositions can be interpreted over the control states of two or more threads) is undecidable. In [6] it is shown that this problem becomes decidable if we consider single-indexed LTL/CTL properties (formulas of the form $\bigwedge f_i$ s.t. f_i is a LTL/CTL formula interpreted over the PDS i). On the other hand, pairwise reachability of PL-DPNs without thread creation is undecidable in the general case [4, 25]. It becomes decidable if locks are accessed in a well-nested style [4], where each thread can only releases the latest acquired lock, and a thread does not change its priority while holding a lock [25].

In this work, we combine these ideas and show that model-checking single indexed LTL/CTL properties is decidable for PL-DPNs under these restrictions. It is non-trivial to do LTL/CTL model checking for PL-DPNs, since the number of instances of PDSs can be unbounded. Checking independently whether all the different PDSs satisfy the corresponding subformula f_i is not correct. Indeed, we do not need to check whether an instance of a PDS j satisfies f_j if this instance is not created during a run. The approach of [6] cannot be directly applied to perform single-indexed LTL/CTL model-checking for PL-DPNs due to locks and priorities. Indeed, we have to consider communication between each instance of PDSs running in parallel in the network. To overcome this problem, we will reduce single-indexed LTL/CTL model-checking for PL-DPNs to the membership problem of PL-DPNs with Büchi acceptance condition (PL-BDPNs). This latter problem is reduced to the membership problem of DPNs with Büchi acceptance condition (BDPNs).

In [25] we presented an approach for checking pairwise reachability of PL-DPNs using priority-lock structures, an extension of acquisition structures introduced in [16]. This structure is used to get rid of locks and priorities in PL-DPNs such that pairwise reachability of PL-DPNs can be reduced to constrained pairwise reachability on DPNs. It works by keeping track of the locks and priorities used in a run. For pairwise reachability, we only need to consider finite runs, as a configuration of a PL-DPN reaches another configuration only using finite steps. However, we have to consider infinite runs of PL-DPNs when we study LTL/CTL model checking.

In this work, we adapted the priority-lock structures to keep track also of the infinitely used locks and priorities. Indeed, we need to assure that a finally acquired lock cannot be infinitely used and that an infinitely used priority does not block other threads. Also, in the case of CTL model checking, we modified the priority-lock structure to keep track of the locks and priorities of differents branched runs.

After getting rid of locks and priorities using the modified priority-lock structure, we construct Büchi dynamic PDSs (resp. alternating Büchi dynamic PDSs) which are a synchronization of a PDS i and the LTL (resp. CTL) formula f_i. The language accepted by a Büchi dynamic PDS corresponds to the configurations that satisfy the formula f_i. This language is computed by the automata-based approach for standard LTL/CTL model checking for PDSs [6].

Thus, the contributions of this paper are:

- An algorithm for single-indexed LTL Model Checking for PL-DPNs, developed in section 4.
- An algorithm for single-indexed CTL Model Checking for PL-DPNs, developed in section 5.

For lack of space proofs can be found in the extended version of this paper at [1].

2 Model Definition

Let L be the set of all locks and I be the set of all priorities. A PL-DPN can be seen as a collection of threads running in parallel, each of them having a set of acquired locks and a priority. They are able to:

1. Perform pushdown operations. This can be used to model calls and returns from (possible recursive) functions.
2. Change its priority if its set of acquired locks is empty. Removing this constraint leads to undecidability [25].
3. Acquire a lock that does not belong to any set of acquired locks (of the running threads).
4. Release a lock belonging to its set of acquired locks.
5. Create a new thread with any (initial) priority and an empty set of acquired locks.

Definition 1. *Dynamic Pushdown System with Locks and Priorities (PL-DPDS) is a tuple $\mathcal{P} = (P, \Gamma, \delta, \eta_p, \eta_l)$, where P is a finite set of control states, Γ is a finite stack alphabet, η_p is a function from control states to priorities from I, η_l is a function from control states to set of locks from L, δ is a finite set of transition rules of the following forms:*

1. $p\gamma \hookrightarrow^{\tau} qw$, *with* $\eta_p(q) = \eta_p(p)$ *and* $\eta_l(q) = \eta_l(p)$;
2. $p\gamma \xhookrightarrow{ch(x)} qw$, *with* $\eta_p(q) = x$ *and* $\eta_l(q) = \eta_l(p) = \emptyset$;
3. $p\gamma \xhookrightarrow{acq\ l} qw$, *with* $\eta_p(q) = \eta_p(p)$, $\eta_l(q) = \eta_l(p) \cup \{l\}$ *and* $l \notin \eta_l(p)$;
4. $p\gamma \xhookrightarrow{rel\ l} qw$, *with* $\eta_p(q) = \eta_p(p)$, $\eta_l(q) = \eta_l(p) \setminus \{l\}$ *and* $l \in \eta_l(p)$;
5. $p\gamma \hookrightarrow^{\tau} q_1w_1 \ \rhd\ q_2w_2$, *with* $\eta_p(q_1) = \eta_p(p)$, $\eta_l(q_1) = \eta_l(p)$ *and* $\eta_l(q_2) = \emptyset$.

where $p, q_1, q_2 \in P, \gamma \in \Gamma$, $w, w_1, w_2 \in \Gamma^*, l \in L, x \in I$.

A *local configuration* $p\omega \in P\Gamma^*$ of a PL-DPDS $\mathcal{P} = (P, \Gamma, \delta, \eta_p, \eta_l)$, represents the state of a thread. The state of a thread consists of a priority, a set of acquired locks and a stack. The priority of a thread and the set of acquired locks are represented by its control state p and can be retrieved from it by using the functions η_p and η_l, respectively. The stack of a thread is represented by the sequence of stack letters $\omega \in \Gamma^*$.

The function η_p assigns a priority to each control state. Intuitively, this means that a thread can be in configurations with different priorities. The function η_l assigns a set of locks to each control state. This set of locks represents the locks held (acquired but not yet released) by the thread at such configuration.

Definition 2. *A Dynamic Pushdown Network with Priorities and Locks (PL-DPN) is a tuple* $(Act, L, I, \mathcal{P}_1, \ldots, \mathcal{P}_n)$ *such that* L *is a set of locks,* I *is set of priorities (natural numbers),* Act *is a finite set of actions* $\{acq(l), rel(l) | l \in L\} \cup \{ch(x) | x \in I\} \cup \{\tau\}$*, where the action* $acq(l)$ *(resp.* $rel(l)$*) for every* $l \in L$ *denotes the* acquisition *(resp.* release*) of the lock* l*, the action* $ch(x)$ *denotes the* change to priority x *and the action* τ *denotes a pushdown action. For every* $i \in \{1, \ldots, n\}$*,* $\mathcal{P}_i$ *is a PL-DPDS.*

A *global configuration* of a PL-DPN M is a sequence of local configurations, each of them corresponding to the configuration of one of the threads running in parallel on the system. Let $Conf_M$ be the set of all global configurations of a PL-DPN M.

Following previous works we assume that locks are used in a well-nested fashion, i.e. a process has to release locks in the opposite order of acquisition, an assumption that is often satisfied in practice. Note that for non-well-nested locks even simple reachability problems are undecidable [23].

2.1 Example

The PL-DPN modeling the program of Figure 1 is defined as follows $M = (\{acq\ l, rel\ l, ch\ l, \tau\}, \{l\}, \{1\}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3)$ where:

- $\mathcal{P}_1 = (\{p_1\}, \{m_0, m_1, m_f\}, \{p_1 m_0 \overset{\tau}{\hookrightarrow} p_1 m_1 \triangleright p_1 a_0, p_1 m_1 \overset{\tau}{\hookrightarrow} p_1 m_f \triangleright p_1 b_0\}, \eta_p, \eta_l)$ such that $\eta_p(p_1) = 1$ and $\eta_l(p_1) = \emptyset$.
- $\mathcal{P}_2 = (\{p_1, p_{1,l}\}, \{a_0, a_1, a_2, a_f\}, \{p_1 a_0 \overset{acq\ l}{\longrightarrow} p_{1,l} a_1, p_{1,l} a_1 \overset{\tau}{\hookrightarrow} p_{1,l} a_2, p_{1,l} a_2 \overset{\tau}{\hookrightarrow} p_1 a_f, p_1 a_f \overset{\tau}{\hookrightarrow} p_1 a_0\}, \eta_p, \eta_l)$ such that $\eta_p(p_1) = \eta_p(p_{1,l}) = 1$ and $\eta_l(p_1) = \emptyset, \eta_l(p_{1,l}) = \{l\}$.
- $\mathcal{P}_3 = (\{p_1, p_{1,l}\}, \{b_0, b_1, b_2, b_f\}, \{p_1 b_0 \overset{acq\ l}{\longrightarrow} p_{1,l} b_1, p_{1,l} b_1 \overset{\tau}{\hookrightarrow} p_{1,l} b_2, p_{1,l} b_2 \overset{rel\ l}{\longrightarrow} p_1 b_f, p_{1,l} b_2 \overset{\tau}{\hookrightarrow} p_{1,l} b_1\}, \eta_p, \eta_l)$ such that $\eta_p(p_1) = \eta_p(p_{1,l}) = 1$ and $\eta_l(p_1) = \emptyset, \eta_l(p_{1,l}) = \{l\}$.

The initial configuration of this PL-DPN M is $p_1 m_0$.

3 Semantics of the Model

The semantics of PL-DPNs is defined such that:

- Transitions of threads with highest priority should be executed first.
- Transitions that manipulate locks should follow the locking rules:
  - A transition attempting to acquire a lock can only be executed if the lock is free, i.e. does not belong to any set of acquired locks.
  - A transition attempting to release a lock can only be executed if the lock is in possession of the corresponding thread, i.e. in its set of acquired locks.

We overload the functions η_p and η_l to global configurations as follows: for all $c = p_1\omega_1 \dots p_n\omega_n \in Conf_M$, $\eta_p(p_1\omega_1 \dots p_n\omega_n) := max(\eta_p(p_1), \dots, \eta_p(p_n))$ and $\eta_l(p_1\omega_1 \dots p_n\omega_n) := \eta_l(p_1) \cup \dots \cup \eta_l(p_n)$.

Definition 3. *The transition relation $\longrightarrow_M$ is defined as the smallest relation in $Conf_M \times Conf_M$ such that $\forall c_1, c_2 \in Conf_M$:*

1. $c_1\ p\gamma r\ c_2 \longrightarrow_M c_1\ q\omega r\ c_2$, *if* $\eta_p(p) = \eta_p(c_1\ p\gamma r\ c_2)$ *and* $p\gamma \xhookrightarrow{act} q\omega \in \Delta$, *s.t.* $act \in \{\tau, rel\ l\} \cup \{ch(x) \mid x \in I\}$;
2. $c_1\ p\gamma r\ c_2 \longrightarrow_M c_1\ q\omega r\ c_2$, *if* $\eta_p(p) = \eta_p(c_1\ p\gamma r\ c_2)$, $l \notin \eta_l(c_1\ p\gamma r\ c_2)$ *and* $p\gamma \xhookrightarrow{acq\ l} q\omega \in \Delta$;
3. $c_1\ p\gamma r\ c_2 \longrightarrow_M c_1\ q_2\omega_2\ q_1\omega_1 r\ c_2$, *if* $\eta_p(p) = \eta_p(c_1\ p\gamma r\ c_2)$ *and* $p\gamma \xhookrightarrow{\tau} q_1\omega_1 \rhd q_2\omega_2 \in \Delta$;

where $p, q, q_1, q_2 \in P, \gamma \in \Gamma, \omega, \omega_1, \omega_2, r \in \Gamma^*, l \in L$.

The semantics above says that:

1. A thread in a local configuration with control state p and top of stack γ can move to a local configuration with control state q, replacing the top of its stack γ by w, if there is a τ, *ch(x) or release* rule $p\gamma \xhookrightarrow{lab} qw \in \Delta$ and its priority $(\eta_p(p))$ is equal to the highest priority among all the threads $(\eta_p(c_1\ p\gamma r\ c_2))$;
2. A thread in a local configuration with control state p and top of stack γ can move to a local configuration with control state q, replacing the top of its stack γ by w, if there is an *acquire* rule $p\gamma \xhookrightarrow{acq\ l} qw \in \Delta$, the lock that the rule attempts to take is free $(l \notin \eta_l(c_1\ p\gamma r\ c_2))$, and its priority $(\eta_p(p))$ is equal to the highest priority among all the threads $(\eta_p(c_1\ p\gamma r\ c_2))$;
3. A thread in a local configuration with control state p and top of stack γ can move to a local configuration with control state q_1, replacing the top of its stack γ by w_1 and create another thread in control state q_2 with stack w_2, if there is a rule $p\gamma \xhookrightarrow{\tau} q_1w_1 \rhd q_2w_2 \in \Delta$ and its priority $(\eta_p(p))$ is equal to the highest priority among all the threads $(\eta_p(c_1\ p\gamma r\ c_2))$.

Note that the semantics of locks corresponds to the one of spin-locks, found in most of the libraries for threads (like Pthreads). Spin-locks are similar to mutexes, but they might have lower overhead for very short-term blocking. When the calling thread requests a spin-lock that is already held by another thread, the calling thread spins in a loop to test if the lock has become available. This means that if a thread with lower priority, holding a lock l, is interrupted by a thread with higher priority, attempting to acquire the same lock, then the program becomes blocked (assuming there is only one CPU). In this paper we assume that programs are free of deadlocks since they can be detected using the technique of our previous work [25].

Given a configuration c, the set of immediate predecessors of c in a PL-DPN M is defined as $pre_M(c) = \{c' \in Conf_M : c' \longrightarrow_M c\}$. This notation can be generalized straightforwardly to sets of configurations. Let pre^*_M denote the reflexive-transitive closure of pre_M. For the rest of this paper, we assume that we have fixed a PL-DPN $M = (Act, L, I, \mathcal{P}_1, \ldots, \mathcal{P}_n)$.

4 Single-Indexed LTL Model Checking for PL-DPNs

4.1 Linear Temporal Logic (LTL) and Büchi Automata

From now on, we fix a finite set of atomic propositions AP.

Definition 4. *The set of LTL formulas is given by (where $q \in AP$):*

$$\varphi ::= q \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid X\varphi \mid F\ \varphi \mid G\ \varphi \mid \varphi_1\ U\ \varphi_2$$

Given an ω-word $\alpha = \alpha_0\alpha_1 \ldots$ over 2^{AP}, let α^k denote the suffix of α starting from α_k. The notation $\alpha \models \varphi$ indicates that α satisfies φ, where $\models$ is inductively defined as follows: $\alpha \models q$ if $q \in \alpha_0$; $\alpha \models \neg\varphi$ if $\alpha\neg \models \varphi$; $\alpha \models \varphi_1 \wedge \varphi_2$ if $\alpha \models \varphi_1$ and $\alpha \models \varphi_2$; $\alpha \models X\varphi$ if $\alpha^1 \models \varphi$; $\alpha \models \varphi_1 U \varphi_2$ if there exists $k \geq 0$ such that $\alpha^k \models \varphi_2$ and for every $j : 1 \leq j < k$, $\alpha_j \models \varphi_1$. The temporal operators F and G can be defined using the following equivalences: $F\ \varphi \equiv true\ U\varphi$, $G\ \varphi \equiv \neg F\ \neg\varphi$.

Definition 5. *A Büchi automaton (BA) $\mathcal{B}$ is a tuple $(G, \Sigma, \theta, g^0, F)$, where G is a finite set of states, Σ is the input alphabet, $\theta \subseteq G \times \Sigma \times G$ is a finite set of transitions, $g^0 \in G$ is the initial state and $F \subseteq G$ is a finite set of accepting states.*

A run of $\mathcal{B}$ over an ω-word $\alpha_0\alpha_1 \ldots$ is a sequence of states $q_0q_1 \ldots$ s.t. $q_0 = g^0$ and $(q_i, \alpha_i, q_{i+1}) \in \theta$ for every $i \geq 0$. A run is accepting iff it infinitely often visits some states in F.

Theorem 1. *(From [3]) Given a LTL formula f we can construct a BA $\mathcal{B}_f$ (s.t. $\Sigma = 2^{\mathcal{AP}}$) recognizing all the ω-words that satisfy f .*

4.2 The Model Checking Problem

The model checking problem of PL-DPNs against doble-indexed LTL formulas where the validity of atomic propositions depends on two or more PL-DPNs is undecidable [4].

In this work, in order to obtain decidability results, we consider the model-checking problem of PL-DPNs against single-indexed LTL properties of the form $f = \bigwedge_{i=1}^{n} f_i$, where f_i is interpreted over the PL-DPDS $\mathcal{P}_i$.

Let λ be a labeling function $\lambda : \bigcup_i P_i \to 2^{AP}$, that assigns to each control location of the PL-DPN M a set of atomic propositions.

Definition 6. *Given a labeling function λ, a local run $r = p_0 w_0 p_1 w_1 \dots$ of the PL-DPDS P_i satisfies f_i, denoted by $r \models f_i$, iff the ω-word $\lambda(p_0)\lambda(p_1)\dots$ satisfies f_i.*

Definition 7. *A global run R satisfies $f = \bigwedge_i f_i$, denoted by $R \models f$, iff all local runs of each instance of $\mathcal{P}_i$ running in parallel in R satisfy f_i.*

Definition 8. *A PL-DPN M, with initial configuration $p_0\gamma_0$, satisfies $f = \bigwedge_i f_i$, denoted by $M \models f$, iff all global runs starting with $p_0\gamma_0$ satisfy f.*

From now on, we fix a single-indexed LTL formula $f = \bigwedge_{i=1}^{n} f_i$.

4.3 Priority-Lock Structures

Definition 9. *(From [25]) A* priority-lock structure *of a global run R of a PL-DPN under DPN semantics, is defined as either a tuple $[\![x, y, g_r, g_a, la]\!]$ or the symbol $\perp$.*

In [25] is given an algorithm to compute a priority-lock structure from a global run R such that we get $\perp$ if the run is not a valid under PL-DPN semantics, or we get the tuple $[\![x, y, g_r, g_a, la]\!]$ if the run is valid under PL-DPN semantics, where:

- x is the lowest transition priority, from the control states visited during the run,
- y is the highest final priority, from the control states of the final configuration,
- g_r is a set of dependencies between lock *usages* (acquire and release of a lock) and *final releases* of a lock (release without acquisition),
- g_a is a set of dependencies between lock *usages* and *initial acquisitions* of a lock (acquisition of a lock without the corresponding release),
- la set of lock actions and their corresponding priorities.

In this work we just need to know that given a PL-DPN M and a regular set of configurations S, we can construct a DPN M', with priority-lock structures embedded in the control states, such that the predecessor configurations of S over M are the predecessor configurations of S over M' with a priority-lock structure not equal to $\perp$. Formally, from [25]:

Theorem 2. $pre^*_M(S) = \{p\omega \mid (p,s)\omega \in pre^*_{M'}(S \times [\![\infty, 0, \emptyset, \emptyset, \emptyset]\!]) \wedge s \neq \bot\}$.

Using the previous theorem we can reduce LTL/CTL model checking on the PL-DPN M to a series of pre^* queries over the DPN M'. In order to keep the queries consistent with each other, taking in account the priorities and locks, we will need to inspect the priority-lock structure stored in the configurations. For that, given a control state p in the DPN M', let $X(p), U(p), R(p)$ and $A(p)$ be the lowest transition priority, set of lock usages, set of initial releases and set of final acquisitions, respectively, embedded in the control state p.

Example The PL-DPN M of Example 2.1 can be reduced to the DPN $M' = (\{\tau\}, \mathcal{P}'_1, \mathcal{P}'_2, \mathcal{P}'_3)$ where:

- $\mathcal{P}'_1 = (\{p'_0 = (p_1, [\![1,1,\emptyset,\emptyset,\emptyset]\!]), p'_1 = (p_1, [\![1,1,\emptyset,\emptyset,\{(l,usg,1,1)\}]\!]),$
  $p'_2 = (p_1, [\![1,1,\emptyset,\emptyset,\{(l,acq,1,1)\}]\!]),$
  $p'_3 = (p_1, [\![1,1,\emptyset,\emptyset,\{(l,acq,1,1),(l,usg,1,1)\}]\!]), p'_4 = (p_1, \bot),\}, \{m_0, m_1, m_f\},$
  $\{p'_1 m_0 \overset{\tau}{\hookrightarrow} p'_1 m_1 \rhd p'_1 a_0, p'_3 m_0 \overset{\tau}{\hookrightarrow} p'_1 m_1 \rhd p'_2 a_0, p'_3 m_0 \overset{\tau}{\hookrightarrow} p'_2 m_1 \rhd p'_1 a_0,$
  $p'_4 m_0 \overset{\tau}{\hookrightarrow} p'_2 m_1 \rhd p'_2 a_0, p'_2 m_0 \overset{\tau}{\hookrightarrow} p'_0 m_f \rhd p'_2 b_0, p'_1 m_0 \overset{\tau}{\hookrightarrow} p'_0 m_f \rhd p'_1 b_0,\})$
- $\mathcal{P}'_2$ and $\mathcal{P}'_3$ are defined similar. In particular, they have the same set of control states.

4.4 The Model-Checking Approach

The next step is to define a DPN with Büchi acceptance condition.

Definition 10. *A Büchi DPDS (BDPDS) is a tuple $\mathcal{BP} = (P, \Gamma, \Delta, F)$, where (P, Γ, Δ) is a DPDS and $F \subseteq P$ is a finite set of accepting control locations.*

For $i \in \{1, \ldots, n\}$, let $\mathcal{B}_i = (G_i, \Sigma_i, \theta_i, g_i^0, F_i)$ be the Büchi automaton recognizing the ω-words that satisfy the LTL formula f_i.

Definition 11. *We define BDPDSs $\mathcal{BP}_i = ((P_i \times G_i) \times (2^L \times I), \Gamma_i, \Delta'_i, F'_i)$ where $F'_i = \{((p,g),(A(p),X(p))) \mid (p,g) \in P_i \times F_i)\}$. Δ'_i is computed such that for every $(g_1, \lambda(p), g_2) \in \theta_i$, $a \in Act$, $x, x_1, x_2 \in I$, $u, u_1, u_2 \in 2^L$ and $(p_2, s_2)\omega_2 \in P_j \times \Gamma_j^*$ we have:*

1. $((p,g_1),(u,x))\gamma \overset{a}{\hookrightarrow} ((p_1,g_2),(u,x))\omega_1 \in \Delta'_i$, *if* $p\gamma \overset{a}{\hookrightarrow} p_1\omega_1 \in \Delta_i$,
2. *If* $p\gamma \overset{a}{\hookrightarrow} p_1\omega_1 \rhd p_2\omega_2 \in \Delta_i$,
   2.1. $((p,g_1),(u_1 \cup u_2, x))\gamma \overset{a}{\hookrightarrow} ((p_1,g_2),(u_1,x_1))\omega_1 \rhd ((p_2,g_j^0),(u_2,x_2))\omega_2 \in \Delta'_i$, *if* $A(p_1) \cap u_1 = A(p_2) \cap u_2 = \emptyset$ *and* $x_1 = x_2$,
   2.2. $((p,g_1),\bot)\gamma \overset{a}{\hookrightarrow} ((p_1,g_2),(u_1,x_1))\omega_1 \rhd ((p_2,g_j^0),(u_2,x_2))\omega_2 \in \Delta'_i$, *otherwise.*

Sometimes we write a configuration $((p',g),(u,x))$ of a $\mathcal{BP}_i$ as (p,s), where $p = (p',g)$ is called control state and $s = (u,x)$ is called "priority-lock structure".

Let $L(\mathcal{BP}_i)$ be the set of all the tuples $((p,g_i^0),(u,x))\gamma, D)$ such that $\mathcal{BP}_i$ has an accepting run starting from the configuration $((p,g_i^0),(u,x))\gamma$, using infinitely the lowest priority x, the set of locks u and spawning the set of configurations D. We can compute the language of $\mathcal{BP}_i$ using the algorithm from [6]:

Theorem 3. *(From [6]) For every BDPDS $\mathcal{BP}_i = (P_i, \Gamma_i, \Delta_i, F_i)$ we can construct a finite automaton A_i such that $L(A_i) = L(\mathcal{BP}_i)$.*

4.5 Main Algorithm

Given a PL-DPN $M = (Act, \mathcal{P}_1, \ldots, \mathcal{P}_n)$ and a single-indexed LTL formula $f = \bigwedge_i f_i$, in order to check if $M \models f$, we proceed as follows:

1. Create the DPN $M' = (\mathcal{P}'_1, \ldots, \mathcal{P}'_n)$, as in Section 5.1.
2. Create Buchi automatons $\mathcal{B}_i^{\neg}$ satisfying the formulas $\neg f_i$.
3. Construct BDPDSs $\mathcal{B}_i^{\neg}\mathcal{P}'_i$ from the DPN M' and the Buchi automatons $\mathcal{B}^{\neg}$ of (2), as in Definition 11.
4. If an initial configuration $((p'_0, g_0), (u, x))\gamma_0$ is in X, the set of configurations that satisfy the formula $\neg f$, with some set of locks u and priority x then $M \not\models f$. Otherwise we continue to the next step, to be sure there are no livelocks.
5. Create Buchi automatons $\mathcal{B}_i$ satisfying the formulas f_i.
6. Construct BDPDSs $\mathcal{B}_i\mathcal{P}'_i$ from them the DPN M' and the Buchi automatons $\mathcal{B}_i$ of (5), as in Definition 11.
7. If an initial configuration $((p'_0, g_0), \bot)\gamma_0$ is in Y, the set of of configurations that satisfy the formula f, then there is a livelock and $M \not\models f$, otherwise $M \not\models f$.

We can construct the set X in the following iterative way:

1. $X' = \bigcup_i L(\mathcal{B}_i^{\neg}\mathcal{P}'_i)$.
2. $X = \{p\gamma \mid (p\gamma, D) \in Z \wedge D \cap X' \neq \emptyset\}$.
3. If $X \neq X'$, set $X' = X$ and go to 2. Otherwise return X.

where Z is the language of initial configurations of all infinite paths in each DPDS $\mathcal{P}'_i$. We can construct the set Y in the following iterative way:

1. $Y' = \bigcup_i L(\mathcal{B}_i\mathcal{P}'_i)$.
2. $Y = \{(p\gamma, D) \in Y' \mid \forall p'\gamma' \in D \; \exists D' \subseteq Conf_{M'} \; s.t. \; (p'\gamma', D') \in Y'\}$.
3. If $Y \neq Y'$, set $Y' = Y$ and go to 2. Otherwise return Y.

Theorem 4. *A PL-DPN M satisfies a single-indexed LTL formula f ($M \models f$) iff there is not initial configuration in X with non-bottom priority-lock structure and there is not initial configuration in Y with bottom priority-lock structure.*

4.6 Example

We want to check if the single-indexed LTL formula $f = f_1 \wedge f_2 \wedge f_3$, where $f_1 = \text{true}$, $f_2 = GF\ a_1$ and $f_3 = GF\ b_1$, is satisfied by the PL-DPN $M = (Act, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_2)$ of Example 2.1.

The first step was to create the DPN $M' = (\mathcal{P}'_1, \mathcal{P}'_2, \mathcal{P}'_3)$ as in Example 4.3. The second step is to create Buchi automatons $\mathcal{B}_1^{\neg}$, $\mathcal{B}_2^{\neg}$ and $\mathcal{B}_3^{\neg}$ recognizing the ω-words that satisfy the formulas $\neg f_1 = \text{false}$, $\neg f_2 = FG\ \neg a_1$ and $\neg f_3 = FG\ \neg b_1$, respectively. Then we create the BDPDSs $\mathcal{B}_1^{\neg}\mathcal{P}'_1$, $\mathcal{B}_2^{\neg}\mathcal{P}'_2$ and $\mathcal{B}_2^{\neg}\mathcal{P}'_3$ using Definition 11. The next step is to construct the set of configurations X, we get:

1. $X' = L(\mathcal{B}^{\neg}\mathcal{P}_1) \cup L(\mathcal{B}^{\neg}\mathcal{P}_2) \cup L(\mathcal{B}^{\neg}\mathcal{P}_3) = \emptyset \cup \emptyset \cup \emptyset = \emptyset$.
2. $X = \{p\gamma \mid (p\gamma, D) \in Z) \wedge D \cap \emptyset \neq \emptyset\} = \emptyset$.

We have that $X = \emptyset$, this means that the negation of f is not satisfied, but still can be the case that we have some livelock. Thus we continue calculating Y. The algorithm proceeds as follows:

1. $L(\mathcal{BP}'_1) = \{(((p'_3, g_0), \bot)m_0, \{((p'_1, g_0), (\{l\}, 1))a_0, ((p'_2, g_0), (\emptyset, 1))b_0\}), \dots\}$.
2. $L(\mathcal{BP}'_2) = \{((p'_1, g_0), (\{l\}, 1))a_0, \emptyset)\}$ with $A(p'_1) = \emptyset$.
3. $L(\mathcal{BP}'_3) = \{((p'_2, g_0), (\emptyset, 1))b_0, \emptyset)\}$ with $A(p'_2) = \{l\}$.
4. $Y' = L(\mathcal{BP}'_1) \cup L(\mathcal{BP}'_2) \cup L(\mathcal{BP}'_3)$.
5. $Y = \{(((p'_3, g_0), \bot)m_0, \{((p'_1, g_0), (\{l\}, 1))a_0, ((p'_2, g_0), (\emptyset, 1))b_0\})\}$.

We can observe that Y has the initial configuration $((p'_3, g_0), \bot)m_0$. This configuration has a $\bot$ priority-lock structure, since the child corresponding to thread A infinitely uses lock l and the child corresponding to thread B acquire lock l without releasing it (see the rules of Definition 11). This means that there is a livelock and then the formula f is not always satisfied int the PL-DPN M.

5 Single-Indexed CTL model checking of PL-DPNs

In this section we consider single-indexed CTL model checking for PL-DPNs. For technical reasons we suppose that CTL formulas are given in positive normal form, i.e. only atomic propositions are negated.

Definition 12. *The set of CTL formulas is given by (where $a \in AP$):*

$$\varphi ::= q \mid \neg q \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid AX\varphi \mid EX\varphi \mid A(\varphi U \varphi) \mid E(\varphi U \varphi) \mid A(\varphi R \varphi) \mid E(\varphi R \varphi)$$

The other standard CTL operators can be expressed by the above operators. For instance $EF\varphi = E(\text{true}\, U \varphi)$, $AF\varphi = A(\text{true}\, U \varphi)$, $EG\varphi = E(\text{false}\, R \varphi)$ and $AG\varphi = A(\text{false}\, R \varphi)$. The closure $cl(\varphi)$ is the set of all the sub-formulas of φ (including φ). Let $At(\varphi) = \{a \in AP \mid a \in cl(\varphi)\}$ and $cl_R(\varphi) = \{\theta \in cl(\varphi) \mid \theta = E(\varphi_1 R \varphi_2) \vee \theta = A(\varphi_1 R \varphi_2)\}$.

Let $\lambda : AP \to 2^{\bigcup P_i \times \Gamma_i^*}$ a valuation assigning to each atomic proposition a finite set of local configurations. A local configuration c satisfies a CTL formula f_i (denoted by $c \models f_i$) iff there exists $D \subseteq D_i$ such that $c \models_D f_i$ holds, where $\models_D$ is inductively defined as follows:

- $c \models_\emptyset a$ if $c \in \lambda(a)$.
- $c \models_\emptyset \neg a$ if $c \notin \lambda(a)$.
- $c \models_D \varphi_1 \wedge \varphi_2$ if $\exists D_1, D_2 \subseteq \bigcup D_i$ such that $D = D_1 \cup D_2$, $c \models_{D_1} \varphi_1$ and $c \models_{D_2} \varphi_2$;
- $c \models_D \varphi_1 \vee \varphi_2$ if $c \models_D \varphi_1$ or $c \models_D \varphi_2$;
- $c \models_D AX\varphi$ if for every $c_1, \dots, c_m \in Conf_M$ such that $c \Rightarrow c_j \triangleright D_j$, $c_j \models_{D_j} \varphi$ and $D = \bigcup_j D_j$;

- $c \models_D EX\varphi$ if there exist $c' \in Conf_M$ such that $c \Rightarrow_i c' \rhd D'$, $c' \models_{D''} \varphi$ and $D = D' \cup D''$;
- $c \models_D A(\varphi_1 U \varphi_2)$ if for every path $c_0 c_1 \dots$ with $c_0 = c$ for every $m \geq 1, \exists D'_m \subseteq D$, such that $c_{m-1} \Rightarrow c_m \rhd D'_m$, and $\exists k$ such that $c_k \models_{D_k} \varphi_2$, $\forall j < k$ $c_j \models_{D_j} \varphi_1$.
- $c \models_D E(\varphi_1 U \varphi_2)$ if exists a path $c_0 c_1 \dots$ with $c_0 = c$, for every $m \geq 1$, $\exists D'_m$ such that $c_{m-1} \Rightarrow c_m \rhd D'_m$ and $\exists k$ such that $c_k \models_{D_k} \varphi_2$, $\forall k, j < k$ $c_j \models_{D_j} \varphi_1$.
- $c \models_D A(\varphi_1 R \varphi_2)$ if for every path $c_0 c_1 \dots$ with $c_0 = c$, for every $m \geq 1$ $\exists D'_m$ such that $c_{m-1} \Rightarrow c_m \rhd D'_m$ and either for all j, $c_j \models_{D_j} \varphi_2$ or $\exists k$ such that $c_k \models_{D''_k} \varphi_1$ and $\forall j \leq k$, $c_j \models_{D_j} \varphi_2$.
- $c \models_D E(\varphi_1 R \varphi_2)$ if exists a path $c_0 c_1 \dots$ with $c_0 = c$, for every $m \geq 1$ $\exists D'_m$ such that $c_{m-1} \Rightarrow c_m \rhd D'_m$ and either for all j, $c_j \models_{D_j} \varphi_2$ or $\exists k$ such that $c_k \models_{D''_k} \varphi_1$ and $\forall j \leq k$, $c_j \models_{D_j} \varphi_2$.

Intuitively, $c \models_D f_i$ means that c satisfies f_i and the executions that made c satisfy f_i spawn the configurations in D, i.e. when a transition rule $q\gamma \hookrightarrow p_1 w_1 \rhd p_2 w_2$ is used to make f_i satisfied, $p_2 w_2$ is in D.

5.1 Priority-Lock Alternating BDPDSs

Definition 13. *A priority-lock alternating BDPDS (PL-ABDPDS) is a tuple $\mathcal{BP}_i = (P_i, \Gamma_i, \Delta_i, F_i)$, where P_i is a finite set of control locations, Γ_i is the stack alphabet, $F_i \subseteq P_i$ is a set of accepting control locations, Δ_i is a finite set of transition rules in the form of $p\gamma \hookrightarrow \{p_1\omega_1, \dots, p_h\omega_h\} \rhd \{q_1 u_1, \dots, q_k u_k\}$.*

An PL-ABDPDS $\mathcal{BP}$ induces a relation $\rightarrow$ defined as follows: for every $\omega \in \Gamma^*$ if $p\gamma \hookrightarrow \{p_1\omega_1, \dots, p_h\omega_h\} \rhd \{q_1 u_1, \dots, q_k u_k\} \in \Delta$, then $p\gamma\omega \rightarrow \{p_1\omega_1\omega, \dots, p_h\omega_h\omega\} \rhd \{q_1 u_1, \dots, q_k u_k\}$. Intuitively, if $\mathcal{BP}$ is at the configuration $p\gamma\omega$, it can fork into h copies in the configurations $p_1\omega_1\omega, \dots, p_h\omega_h\omega$ and creates k new instances. We write $p\gamma \hookrightarrow \{p_1\omega_1, \dots, p_h\omega_h\}$ if $p\gamma \hookrightarrow \{p_1\omega_1, \dots, p_h\omega_h\} \rhd \emptyset$.

A run is *accepting* if each branch of this run *infinitely often* visits some control locations in F. Let $L(\mathcal{BP})$ be the set of all the pairs (c, D) such that $\mathcal{BP}$ has an accepting run from c and that creates the set of configurations D.

5.2 Computing an Alternating BDPDS

To perform single-indexed CTL model checking for PL-DPNs we follow the approach for LTL model checking, but in this case we need alternating BDPDSs, since CTL formulas can be translated to alternating Büchi automata.

On [6] it is shown how to model check DPNs against CTL formulas using alternating BDPDS. Here we reduce CTL model checking in a PL-DPN to CTL model checking in a DPN by using priority-lock structures and applying the result of [6].

Let $\mathcal{BP}_i = (P'_i, \Gamma_i, \Delta'_i, F_i)$ be the PL-ABDPDS such that $P'_i = P_i \times cl(f_i)$, $F_i = \{(p, a) \mid a \in cl(f_i) \cap AP, p \in f(a)\} \cup \{(p, \neg a) \mid \neg a \in cl(f_i), a \in AP, p \notin$

$f(a)\} \cup P_i \times cl_R(f_i)$, where $cl_R(f_i)$ is the set of formulas of $cl(f_i)$ of the form $E(\varphi_1 R \varphi_2)$ or $A(\varphi_1 R \varphi_2)$; and Δ'_i is the smallest set of transitions rules such that for every control location $p \in P_i$, every subformula $\varphi \in cl(f_i)$ and every $\gamma \in \Gamma_i$ we have:

1. If $\varphi = a, a \in AP$ and $p \in f(a)$ then $(p, \varphi)\gamma \hookrightarrow (p, \varphi)\gamma \in \Delta'$;
2. If $\varphi = \neg a, a \in AP$ and $p \notin f(a)$ then $(p, \varphi)\gamma \hookrightarrow (p, \varphi)\gamma \in \Delta'$;
3. If $\varphi = \varphi_1 \wedge \varphi_2$ then $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_1)\gamma, (p, \varphi_2)\gamma\} \in \Delta'_i$;
4. If $\varphi = \varphi_1 \vee \varphi_2$ then $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_1)\gamma\} \in \Delta'_i$ and $(p, \varphi_2)\gamma \hookrightarrow \{(p, \varphi_2)\gamma\} \in \Delta'_i$;
5. If $\varphi = EF\varphi_1$ then: if $p\gamma \hookrightarrow p'\omega \rhd p''\omega' \in \Delta_i$ then $(p, \varphi)\gamma \hookrightarrow \{(p', \varphi_1)\omega\} \rhd \{p''\omega'\} \in \Delta'_i$, similar for non-spawning rules;
6. If $\varphi = AX\varphi_1$ then $(p, \varphi)\gamma \hookrightarrow \{(p', \varphi_1)\omega \mid p\gamma \hookrightarrow p'\omega \rhd p''\omega' \in \Delta_i\} \rhd \{p''\omega' \mid p\gamma \hookrightarrow p'\omega \rhd p''\omega' \in \Delta'_i\}$;
7. If $\varphi = E(\varphi_1 U \varphi_2)$ then: $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_1)\gamma, (p, \varphi_2)\gamma\} \in \Delta'_i$, and if $p\gamma \hookrightarrow p'\omega \rhd p''\omega' \in \Delta_i$ then $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_1)\gamma, (p', \varphi)\omega\} \rhd \{p''\omega'\} \in \Delta'_i$, similar for non-spawning rules;
8. If $\varphi = A(\varphi_1 U \varphi_2)$ then: $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_2)\gamma\} \in \Delta'_i$, if $p\gamma \hookrightarrow p'\omega \rhd p''\omega' \in \Delta_i$ then $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_1)\gamma, (p', \varphi)\omega \mid p\gamma \hookrightarrow p'\omega \rhd p''\omega' \in \Delta_i\} \rhd \{p''\omega'\} \in \Delta'_i$, similar for non-spawning rules;
9. If $\varphi = E(\varphi_1 R \varphi_2)$ then: if $p\gamma \hookrightarrow p'\omega \rhd p''\omega' \in \Delta_i$ then $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_2)\gamma, (p, \varphi_1)\gamma\} \in \Delta'_i$ and $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_2)\gamma, (p', \varphi)\omega\} \rhd \{p''\omega'\} \in \Delta'_i$;
10. If $\varphi = A(\varphi_1 R \varphi_2)$ then $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_2)\gamma, (p, \varphi_1)\gamma\} \in \Delta'_i$ and $(p, \varphi)\gamma \hookrightarrow \{(p, \varphi_2)\gamma, (p', \varphi)\omega \mid p\gamma \to p'\omega \rhd p''\omega' \in \Delta_i\} \rhd \{p''\omega' \mid p\gamma \hookrightarrow p'\omega \rhd p''\omega' \in \Delta_i\} \in \Delta'_i$.

References

1. Extended version: `https://github.com/marcio-diaz/cuddly-sniffle/blob/master/ltl-ctl-pl-dpn.pdf`.
2. Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In CONCUR'97. LNCS 1243, 1997.
3. Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. J. Comput. Syst. Sci., 32(2):183-221, 1986.
4. Vineet Kahlon and Aarti Gupta. An automata-theoretic approach for model checking threads for LTL properties. In LICS, pages 101-110, 2006.
5. Peter Lammich, Markus Müller-Olm, Helmut Seidl, and Alexander Wenner. Contextual locking for dynamic pushdown networks. In SAS, 2013.
6. Fu Song and Tayssir Touili. Model checking dynamic pushdown networks. In APLAS, 2013.
7. Alexander Wenner. Weighted dynamic pushdown networks. In ESOP, pages 590-609, 2010.
8. Peter Lammich and Markus Müller-Olm. Precise fixpoint-based analysis of programs with thread creation and procedures. In CONCUR, pbages 287-302, 2007
9. Thomas Martin Gawlitza, Peter Lammich, Markus M?ller-Olm, Helmut Seidl, and Alexander Wenner. Join-lock-sensitive forward reachability analysis for concurrent programs with dynamic process creation. In VMCAI, pages 199-213, 2011.

10. Denis Lugiez. Forward analysis of dynamic network of pushdown systems is easier without order. Int. J. Found. Comput. Sci., 22(4):843-862, 2011.
11. Gawlitza, T. M., Lammich, P., Müller-Olm, M., Seidl, H. and Wenner, A. Join-lock-sensitive forward reachability analysis for concurrent programs with dynamic process creation. In Verification, Model Checking, and Abstract Interpretation, pages 199-213, 2011.
12. Wenner, A. Weighted dynamic pushdown networks. In Programming Languages and Systems, pages 590-609, 2010.
13. Bouajjani, A., Müller-Olm, M., and Touili, T. Regular symbolic analysis of dynamic networks of pushdown systems. In CONCUR, pages 473-487, 2005.
14. Kidd, N., Jagannathan, S., and Vitek, J. One stack to run them all. In Model Checking Software, pages 245-261, 2010.
15. Atig, M. F., Bouajjani, A., and Touili, T. Analyzing Asynchronous Programs with Preemption. In FSTTCS, pages 37-48, 2008.
16. P. Lammich, M. Müller-Olm, and A. Wenner. Predecessor sets of dynamic pushdown networks with tree-regular constraints. In CAV, pages 525-539, 2009.
17. Bouajjani, A., Esparza, J., Schwoon, S., and Strejček, J. Reachability Analysis of Multithreaded Software with Asynchronous Communication. In FSTTCS, pages 348-359, 2005.
18. P. Lammich and M. Müller-Olm. Precise fixpoint-based analysis of programs with thread creation and procedures. In CONCUR, pages 287-302, 2007.
19. Mayr, R. Process rewrite systems. Information and Computation, 156(1), pages 264-286, 2000.
20. S. Goller and A. W. Lin. The complexity of verifying ground tree rewrite systems. In LICS, pages 279-288, 2011.
21. Rance Cleaveland and Matthew Hennessy. Priorities in process algebras. Inf. Comput., 87(1-2):58-77, 1990.
22. Lugiez, D. Forward analysis of dynamic network of pushdown systems is easier without order. In Reachability Problems, pages 127-140, 2009.
23. Kahlon, V., Ivancic, F., Gupta, A. Reasoning about threads communicating via locks. In Computer Aided Verification, pages 505-518, 2005.
24. Diaz, M., Touili, T. Reachability Analysis of Dynamic Pushdown Networks with Priorities. In NETYS, 2017.
25. Diaz, M., Touili, T. Dealing with Priorities and Locks for Concurrent Programs. In ATVA, 2017.