LTL Model-Checking for Dynamic Pushdown Networks Communicating via Locks

Fu Song, Tayssir Touili

LTL Model-Checking for Dynamic Pushdown Networks Communicating via Locks

Fu Song[1] and Tayssir Touili[2]

[1] Shanghai Key Laboratory of Trustworthy Computing, East China Normal University
fsong@sei.ecnu.edu.cn
[2] CNRS and Université Paris Diderot,touili@liafa.univ-paris-diderot.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. Extension of DPNs with locks allows processes to synchronize via locks. Thus, DPNs with locks are a well adapted formalism to model multi-threaded programs that synchronize via locks. Therefore, it is important to have model-checking algorithms for DPNs with locks. However, in general, the model-checking problem of DPNs with locks against reachability properties, and hence Linear Temporal Logic (LTL), is undecidable. To obtain decidable results, we study in this work the model-checking problem of DPNs with well-nested locks against single-indexed Linear Temporal Logic (LTL) properties of the form $\bigwedge \mathbf{E}\ f_i$ s.t. f_i is a LTL formula interpreted over the PDS i. We show that this model-checking problem is decidable. We propose an automata-based approach for computing the set of configurations of a DPN with locks that satisfy the corresponding single-indexed LTL formula.

1 Introduction

Multithreaded programming paradigm where processes communicate via locks has become more popular. But, it is notoriously difficult to write multithreaded programs whose bugs are commonly concurrency related and are hard to reproduce and fix. Hence, we need a formalism to model such programs and we need automated verification techniques to analyze them. Dynamic Pushdown Networks (DPN) [4] are a natural model of 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 (process) 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 [4, 21, 20, 9, 17, 27, 23]. However, DPNs cannot model communication between processes. Recent works [20, 19] extended DPNs with locks (L-DPN) where PDSs can communicate via locks. This allows to model multi-threaded programs where processes communicate via locks. Indeed, locks are frequently used in multi-threaded programs as synchronisation primitives. Thus, it is crucial to have automated techniques to verify DPNs with locks. However, only reachability properties are studied for L-DPNs [20, 19] with some restricted lock usages. In this work, we go further and consider the

model-checking problem of L-DPNs against LTL, which can describe more interesting properties of program behaviors that has not been tackled yet.

In general, the model checking problem of DPNs against unrestricted LTL formulas (where atomic propositions can be interpreted over the control states of two or more threads) is undecidable. We have shown in [23] that this problem becomes decidable if we consider single-indexed LTL properties (where a single-indexed LTL formula is a formula of the form $\bigwedge \mathbf{E}\ f_i$ such that f_i is a LTL formula interpreted over the PDS i). On the other hand, the model checking problem of pushdown networks with a finite number of threads (without thread creation) communicating via locks against pairwise reachabillity properties is undecidable [13]. [13] shows that it becomes decidable if locks are accessed in a well-nested style, i.e., each process releases only the latest acquired lock that has not been released yet.

In this work, we combine these ideas and show that model-checking single indexed LTL properties is decidable for L-DPNs if locks are accessed in a well-nested style. It is non-trivial to do LTL model checking for L-DPNs, since the number of instances of PDSs can be unbounded. Checking independently whether all the different PDSs satisfy the corresponding subformula $\mathbf{E}\ f_i$ 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, and we have to guarantee that all the created instances are correctly synchronized via locks. In our previous work [23], we have shown how to solve single-indexed LTL model-checking for L-DPNs without locks, i.e., DPNs. However, the approach of [23] cannot be directly applied to perform single-indexed LTL model-checking for L-DPNs due to locks. 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 model-checking for L-DPNs to the membership problem of L-DPNs with Büchi acceptance condition. This latter problem is reduced to the membership problem of DPNs with Büchi acceptance condition. For this, we introduce lock structures which include the set of held locks, the set of finally acquired locks, the set of (infinitely) used locks, the set of initial release locks, release/acquisition histories. The lock structures are an extension of acquisition structures introduced in [20]. [20] presents an approach for checking pairwise reachability of L-DPNs with well-nested lock access. The acquisition structures are used to get rid of locks in L-DPNs such that pairwise reachability of L-DPNs can be reduced to *constrained* pairwise reachability of DPNs. For pairwise reachability, one only needs to consider finite runs, as a configuration of a L-DPN reaches another configuration only using finite steps. However, we have to consider infinite runs of L-DPNs when we study LTL model-checking. Due to the infinite runs of L-DPNs, we need to introduce the set of infinitely used locks in our lock structures. Indeed, we need to assure that a finally acquired lock cannot be infinitely used.

In order to get rid of lock interaction between processes when checking membership of a L-DPN with Büchi acceptance condition, we compute a DPN with Büchi acceptance condition whose runs mimic the runs of L-DPNs by associating lock structures to the control locations. The runs of the computed DPN with Büchi acceptance condition are lock-free. Thus, to check whether a DPN with Büchi acceptance has a run, it is sufficient to independently check whether each process (pushdown system) has an

accepting run. However, we still need to deal with the fact that the number of processes is unbounded. To solve this problem, we follow the approach of [23]: we compute, for every process i, a kind of finite automaton $\mathcal{A}_i$ which accepts all the configurations from which the pushdown system i has an accepting run, while recording the set of all the initial configurations of the spawn processes. We show how to check whether a DPN with Büchi acceptance condition has a run by querying these automata $\mathcal{A}_i$. Thus, we can check whether the L-DPN satisfies the LTL formula by querying these automata.

Outline. Section 2 gives the basic definitions used in this work. Section 3 shows how to reduce the single-indexed LTL model-checking problem for L-DPNs to the membership problem of L-DPNs with Büchi acceptance condition. Section 4 presents an efficient approach for checking membership of L-DPNs with Büchi acceptance condition. Section 5 discusses related works. Due to lack of space, proofs are omitted. They can be found in the full version of this paper [24].

2 Preliminaries

2.1 Dynamic Pushdown Networks with Locks

Definition 1. *A* Dynamic Pushdown Network *with Locks (L-DPN)* $\mathcal{M}$ *is a tuple* $(Act, \mathbb{L}, \mathcal{P}_1, ..., \mathcal{P}_n)$ *s.t.* $\mathbb{L}$ *is a finite set of locks, Act is a finite set of actions* $\{acq(l), rel(l), \tau \mid l \in \mathbb{L}\}$ *where the action* $acq(l)$ *(resp.* $rel(l)$*) for every* $l \in \mathbb{L}$ *denotes the* acquisition *(resp.* release*) of the lock* l *and the action* τ *denotes all the lock-unrelated* internal actions*; for every* $i \in \{1, ..., n\}$, $\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i)$ *is a* Dynamic Pushdown System with Locks *(L-DPDS), where* P_i *is a finite set of control states s.t.* $P_k \cap P_i = \emptyset$ *for* $k \neq i$, Γ_i *is a finite stack alphabet,* Δ_i *is a finite set of transition rules in the following forms:*

$$\text{(Non-Spawn)} \quad p_0\gamma_0 \overset{a}{\hookrightarrow}_i p_1\omega_1$$
$$\text{(Spawn)} \quad p_0\gamma_0 \overset{\tau}{\hookrightarrow}_i p_1\omega_1 \rhd p_2\omega_2$$

where $a \in Act$, $p_0, p_1 \in P_i$, $\gamma_0 \in \Gamma_i$, $\omega_1 \in \Gamma_i^*$ *and* $p_2\omega_2 \in P_j \times \Gamma_j^*$ *for some* $j : 1 \leq j \leq n$.

Intuitively, a L-DPDS is a pushdown system if the L-DPDS does not have any Spawn-rule and $\mathbb{L} = \emptyset$. A L-DPN consists of a set of pushdown systems (PDSs) running in parallel where each PDS can dynamically create new instances of PDSs during the run and where instances of PDSs can communicate via locks. A L-DPN $(Act, \mathbb{L}, \mathcal{P}_1, ..., \mathcal{P}_n)$ (resp. L-DPDS) is called a dynamic pushdown network (DPN) (resp. dynamic pushdown system (DPDS)) if $\mathbb{L} = \emptyset$ and will sometimes be denoted by $(\mathcal{P}_1, ..., \mathcal{P}_n)$, where the actions are omitted in the transition rules.

For every $i \in \{1, ..., n\}$, a *local configuration* of a L-DPDS $\mathcal{P}_i$ is a tuple $(p\omega, L)$ such that $L \subseteq \mathbb{L}$ is the set of locks held by the instance of $\mathcal{P}_i$, $p \in P_i$ is its control location and $\omega \in \Gamma_i^*$ is its stack content. A *global configuration* of $\mathcal{M}$ is a *multiset* over $\bigcup_{i=1}^{n} P_i \times \Gamma_i^* \times 2^{\mathbb{L}}$, in which each element $(p\omega, L) \in P_i \times \Gamma_i^* \times 2^{\mathbb{L}}$ denotes the local configuration of an instance of $\mathcal{P}_i$ running in parallel in the network. In the setting of DPNs and DPDSs, the set of locks in local and global configurations is sometimes omitted. Given a global configuration $\mathcal{G}$, the set of locks held at $\mathcal{G}$ is $(\bigcup_{(p\omega,L)\in\mathcal{G}} L)$,

denoted by $hold(\mathcal{G})$. The set of free locks at $\mathcal{G}$ is $\mathbb{L} \setminus hold(\mathcal{G})$, denoted by $free(\mathcal{G})$. A global configuration $\mathcal{G}$ containing only one element $(p\omega, L)$ will sometimes be denoted by $(p\omega, L)$.

In this work, w.l.o.g., we assume that the global initial configuration $\mathcal{G}$ of $\mathcal{M}$ contains only one element, i.e., initially, there is only one instance of a L-DPDS $\mathcal{P}_i$ for some $i \in \{1, ..., n\}$. Indeed, a L-DPN whose global initial configuration consists of several local configurations can be transformed into an equivalent L-DPN having a single local configuration as global initial configuration by adding some Spawn-rules. A *global run* of $\mathcal{M}$ starting from a global initial configuration $(p_0\omega_0, L_0)$ is a binary tree T rooted by $(p_0\omega_0, L_0)$ such that the leaves of T (denoted by $leaves(T)$) is the current global configuration. Each node in T can either have only the right child (no left child) or have both the right and left children. The right child of a node $(p\omega, L)$ is the next local configuration of the instance that reaches $(p\omega, L)$, while the left child of a node $(p\omega, L)$ is the local initial configuration of the new instance that is dynamically created when the instance moves from $(p\omega, L)$ to its right child. Formally, the progress of the global run T is defined as follows: for every local configuration $(p\gamma u, L)$ of an instance of a L-DPDS $\mathcal{P}_i$ running in parallel in the network for some $\gamma \in \Gamma_i$ such that $(p\gamma u, L) \in leaves(T)$:

β_1 : If there exists a transition rule $p\gamma \overset{\tau}{\hookrightarrow}_i p'\omega \in \Delta_i$, then $(p'\omega u, L)$ can be the right child of the node $(p\gamma u, L)$ (note that $(p\gamma, L)$ has no left child). This means that this instance can move from $(p\gamma u, L)$ to $(p'\omega u, L)$, moving the control location p to p' and replacing the stack content γ by ω without changing the set of held locks L. (Note that the other instances running in parallel in the network stay at the same local configurations.)

β_2 : If there exists a transition rule $p\gamma \overset{\tau}{\hookrightarrow}_i p'\omega \rhd p_2\omega_2 \in \Delta_i$ s.t. $p_2\omega_2 \in P_j \times \Gamma_j^*$, then $(p'\omega u, L)$ and $(p_2\omega_2, \emptyset)$ can be the right and left children of the node $(p\gamma u, L)$, respectively. This means that this instance can move from $(p\gamma u, L)$ to $(p'\omega u, L)$. Moreover, a new instance of the L-DPDS $\mathcal{P}_j$ is created and it starts from the local configuration $(p_2\omega_2, \emptyset)$. In this work, we suppose w.l.o.g., that the set of locks currently held by this new instance is empty.

β_3 : If there exists a transition rule $p\gamma \overset{acq(l)}{\hookrightarrow}_i p'\omega \in \Delta_i$ such that $l \in free(leaves(T))$ (i.e., l is a free lock at the current global configuration $leaves(T)$), then $(p'\omega u, L \cup \{l\})$ can be the right child of the node $(p\gamma u, L)$. This means that this instance can move from $(p\gamma u, L)$ to $(p'\omega u, L \cup \{l\})$ and hold the lock l.

β_4 : If there exists a transition rule $p\gamma \overset{rel(l)}{\hookrightarrow}_i p'\omega \in \Delta_i$, then $(p'\omega u, L \setminus \{l\})$ can be the right child of the node $(p\gamma u, L)$. This means that this instance can move from $(p\gamma u, L)$ to $(p'\omega u, L \setminus \{l\})$ and free the lock l if it is held.

Intuitively, the root of T is the local initial configuration of the initial instance (i.e., the instance that is not created on the runtime), each left child in a global run T is a local initial configuration of a newly created instance. Each rightmost path is a trace of an instance running in parallel in the global run. A *local run* of an instance running in parallel in T is the rightmost path starting from the root or a left child. The definition of the global runs as trees allows us to know which instance creates a new instance, where the new instance is created and the local initial configuration of the newly created

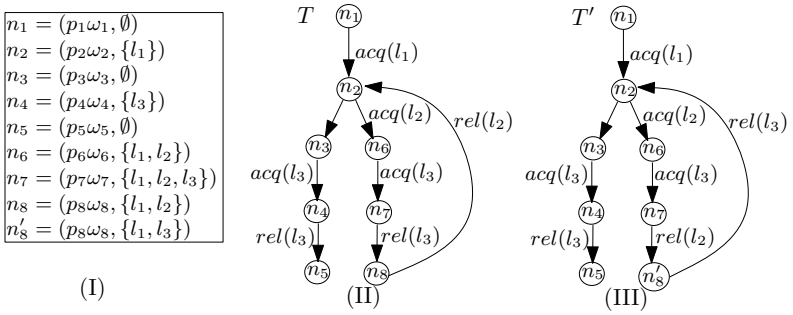

Fig. 1. (a) and (b) are two global runs using locks.

instance. This is important when reasoning about locks. W.l.o.g., we suppose that each local run in a global run is an infinite sequence of local configurations.

Nested Lock Access. A local run uses locks in a *well-nested style* iff the local run releases only the latest acquired lock that has not been released yet by this local run. A global run of $\mathcal{M}$ uses locks in a *well-nested style* iff each local run of the global run uses locks in a well-nested style. In this work, we consider L-DPNs that use locks in a nested style. This is because even reachability, and hence LTL, is known to be undecidable for pushdown networks that use locks in an arbitary style [13]. We assume hereafter that locks are accessed in a nested style. W.l.o.g., in this work, we consider only non-reentrant locks, where a process cannot acquire a lock that it already has (it has to release it first). Reentrant locks can be simulated with non-reentrant locks [20].

Example 1. Figure 1(II) and Figure 1(III) show two global runs called T and T' that use locks l_1, l_2 and l_3. Each edge is labeled by the corresponding action. The nodes $n_1, ..., n_8$ and n'_8 denote the local configurations shown in Figure 1(I). $n_1(n_2n_6n_7n_8)^*$ (resp. $n_1(n_2n_6n_7n'_8)^*$) is a local run in T (resp. T') during which a new instance is created when moving from n_2 to n_6. This new instance has the local run $n_3n_4n_5$. In T, we can see that locks are accessed in a nested style. While, in T', the locks l_1 and l_2 are not accessed in a nested style, since the lock l_2 is released before the release of the latest acquired lock l_3.

2.2 The Linear Temporal Logic (LTL) and Büchi Automata

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

Definition 2. *The set of LTL formulas is given by (where* $ap \in$ AP*):*

$$\psi \ ::= ap \ \mid \ \neg\psi \ \mid \ \psi \wedge \psi \ \mid \ \boldsymbol{X}\psi \ \mid \ \psi \boldsymbol{U} \psi.$$

Given an ω-word $\eta = \pi_0\pi_1...$ over 2^{AP}, let $\eta(k)$ denote π_k, and η_k denote the *suffix* of η starting from π_k. The notation $\eta \models \psi$ indicates that η satisfies ψ, where $\models$ is inductively defined as follows: $\eta \models ap$ iff $ap \in \eta(0)$; $\eta \models \neg\psi$ iff $\eta \not\models \psi$; $\eta \models \psi_1 \wedge \psi_2$ iff $\eta \models \psi_1$ and $\eta \models \psi_2$; $\eta \models \mathbf{X}\psi$ iff $\eta_1 \models \psi$; $\eta \models \psi_1 \mathbf{U} \psi_2$ iff there exists $k \geq 0$ such that $\eta_k \models \psi_2$ and for every $j : 1 \leq j < k$, $\eta_j \models \psi_1$.

Definition 3. *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 $\pi_0\pi_1...$ is an infinite sequence of states $q_0q_1...$ s.t. $q_0 = g^0$ and $(q_j, \pi_j, q_{j+1}) \in \theta$ for every $j \geq 0$. A run is *accepting* iff it infinitely often visits some states in F.

Theorem 1. *[26] Given a LTL formula f, one can construct a BA $\mathcal{B}_f$ s.t. $\Sigma = 2^{AP}$ recognizing all the ω-words that satisfy f.*

3 Single-indexed LTL for L-DPNs

3.1 The Model-Checking Problem

The model-checking problem of L-DPNs against double-indexed LTL properties where the validity of atomic propositions depends on two or more L-DPDSs is undecidable. Indeed, model-checking double-indexed LTL properties for pushdown networks even without interaction with each other is undecidable [13]. Thus, to obtain decidability results, we consider in this work the model-checking problem of L-DPNs against single-indexed LTL properties of the form $\bigwedge_{i=1}^{n} \mathbf{E}\ f_i$ s.t. for every $i \in \{1, ..., n\}$, f_i is a LTL formula interpreted over the L-DPDS $\mathcal{P}_i$ (Formulas of the form $f = \bigwedge_{i=1}^{n} \mathbf{A}\ f_i$ can be checked by taking their negation and then model checking for each $i : 1 \leq i \leq n$, the simpler single-indexed formula $\mathbf{E}\neg f_i$ for which we only need to check whether each newly created instance of $\mathcal{P}_i$ satisfies $\mathbf{E}\neg f_i$ or not). From now on, we fix a L-DPN $\mathcal{M} = (Act, \mathbb{L}, \mathcal{P}_1, ..., \mathcal{P}_n)$ s.t. for every i, $i \in \{1, ..., n\}$, $\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i)$ and a single-indexed LTL formula $f = \bigwedge_{i=1}^{n} \mathbf{E}\ f_i$ such that for every $i \in \{1, ..., n\}$, f_i is interpreted over the L-DPDS $\mathcal{P}_i$.

Given a labeling function $\lambda : \bigcup_{i=1}^{n} P_i \longrightarrow 2^{AP}$ that assigns to each control location a set of atomic propositions and given a global run T of the L-DPN $\mathcal{M}$, a local run $\sigma = (p_0\omega_0, L_0)(p_1\omega_1, L_1)...$ of an instance of $\mathcal{P}_i$ running in parallel in T satisfies $\mathbf{E}\ f_i$ iff the ω-word $\lambda(p_0)\lambda(p_1)...$ satisfies f_i. T satisfies f iff each local run of each instance of $\mathcal{P}_i$ for $i \in \{1, ..., n\}$ running in parallel in T satisfies $\mathbf{E}\ f_i$. A global initial configuration $\mathcal{G}$ satisfies f iff $\mathcal{M}$ has a global run T starting from $\mathcal{G}$ such that T satisfies f.

3.2 The Model-Checking Approach

We start by defining L-BDPNs, a kind of L-DPNs with Büchi acceptance condition.

Definition 4. *A L-DPN with Büchi acceptance condition (L-BDPN for short) $\mathcal{BM}$ is a tuple $(\mathcal{M}, F_1, ..., F_n)$ such that $\mathcal{M} = (Act, \mathbb{L}, \mathcal{P}_1, ..., \mathcal{P}_n)$ is a L-DPN and for every $i \in \{1, ..., n\}$, $F_i \subseteq P_i$ is a set of accepting control locations.*

Global/local runs in L-BDPN are defined as the ones in L-DPN. A local run of an instance of $\mathcal{P}_i$ in $\mathcal{BM}$ is *accepting* iff this local run infinitely often visits some control locations in F_i. A global run in $\mathcal{BM}$ is *accepting* iff each local run in this global run is accepting. Let $\mathcal{L}(\mathcal{BM})$ be the set of global initial configurations from which $\mathcal{BM}$ has an

accepting global run. A L-BDPN $(\mathcal{M}, F_1, ..., F_n)$ is called dynamic pushdown network with Büchi acceptance condition (BDPN for short) if $\mathcal{M}$ is a DPN.

We reduce the problem of checking whether a L-DPN $\mathcal{M}$ satisfies a single-indexed LTL formula $f = \bigwedge_{i=1}^{n} \mathbf{E}\, f_i$ to the membership problem in L-BDPNs: we will compute a L-BDPN $\mathcal{BM}_f = (\mathcal{M}_f, F_1, ..., F_n)$ such that every L-DPDS in $\mathcal{BM}_f$ is a kind of product of the L-DPDS $\mathcal{P}_i$ with a BA $\mathcal{B}_i$, where $\mathcal{B}_i$ exactly accepts all the ω-words that satisfy f_i. We show that $\mathcal{BM}_f$ has an accepting global run starting from a global configuration $(p\omega, L)$ iff $(p\omega, L)$ satisfies f. Intuitively, determining whether a local run of a L-DPDS satisfies $\mathbf{E}\, f_i$ can be reduced to check whether the ω-word yielded by the local run is accepted by $\mathcal{B}_i$ or not, i.e., whether the corresponding local run in $\mathcal{BM}_f$ is accepting or not. Indeed, this is an extension of the automata-based approach for standard LTL model-checking for PDSs [2, 8].

In the rest of this section, we show how to compute the L-BDPN $\mathcal{BM}_f$ from the L-DPN $\mathcal{M}$ and the single-indexed LTL formula $f = \bigwedge_{i=1}^{n} \mathbf{E}\, f_i$. For every $i \in \{1, ..., n\}$, let $\mathcal{B}_i = (G_i, \Sigma_i, \theta_i, g_i^0, F_i)$ be the *BA* recognizing all the ω-words that satisfy f_i. We define the L-BDPN $\mathcal{BM}_f = (\mathcal{M}_f, F'_1, ..., F'_n)$ with $\mathcal{M}_f = (Act, \mathbb{L}, \mathcal{P}'_1, ..., \mathcal{P}'_n)$ such that for every $i \in \{1, ..., n\}$, $F'_i = P_i \times F_i$, $\mathcal{P}'_i = (P_i \times G_i, \Gamma_i, \Delta'_i)$ and Δ'_i is computed as follows: for every $(g_1, \lambda(p), g_2) \in \theta_i$, $a \in Act$ and $p_2\omega_2 \in P_j \times \Gamma_j^*$, we have:

1. $[p, g_1]\gamma \overset{a}{\hookrightarrow} [p_1, g_2]\omega_1 \in \Delta'_i$ iff $p\gamma \overset{a}{\hookrightarrow} p_1\omega_1 \in \Delta_i$;
2. $[p, g_1]\gamma \overset{\tau}{\hookrightarrow} [p_1, g_2]\omega_1 \rhd [p_2, g_j^0]\omega_2 \in \Delta'_i$ iff $p\gamma \overset{\tau}{\hookrightarrow} p_1\omega_1 \rhd p_2\omega_2 \in \Delta_i$.

Intuitively, for every $i \in \{1, ..., n\}$, $\mathcal{P}'_i$ is a product of $\mathcal{P}_i$ and the BA $\mathcal{B}_i$. $\mathcal{B}_i$ has an accepting run $g_0 g_1 ...$ over an ω-word $\lambda(p_0)\lambda(p_1)...$ that corresponds to a local run $\sigma = (p_0\omega_0, L_0)(p_1\omega_1, L_1)...$ of $\mathcal{P}_i$ iff $\mathcal{P}'_i$ has a local run $\sigma' = ([p_0, g_i^0]\omega_0, L_0)\ ([p_1, g_1]\omega_1, L_1)...$ that infinitely often visits some control locations in F'_i. $\mathcal{P}_i$ and $\mathcal{P}'_i$ have the same lock usages. Moveover, for every newly created instance by a local run of $\mathcal{P}_i$ which starts from $(p_2\omega_2, \emptyset)$ and has to satisfy f_j (if $p_2 \in P_j$), the corresponding local run of $\mathcal{P}'_i$ will create a new instance starting from $([p_2, g_j^0]\omega_2, \emptyset)$ which has to be accepting. Thus, we obtain the following theorem.

Theorem 2. *For every global configuration $(p\omega, L) \in P_i \times \Gamma_i^* \times 2^{\mathbb{L}}$ of $\mathcal{M}$, $(p\omega, L)$ satisfies f iff $\mathcal{BM}_f$ has an accepting run starting from $([p, g_i^0]\omega, L)$.*

4 Membership Problem of L-BDPNs

It is non-trivial to check the membership problem of a L-BDPN $\mathcal{BM}$, as we cannot independently compute the local configurations from which a L-DPDS has an accepting local run. Indeed, we have to guarantee the lock collaboration between the different instances in the network. To solve this problem, we introduce lock structures which include the set of held locks, the set of finally acquired locks, the set of (infinitely) used locks, the set of initial release locks and release/acquisition histories. We reduce the membership problem of L-BDPNs to the membership problem of BDPNs by associating each control location with lock structures and imposing constraints on the lock structures during the runs of BDPNs so that we can get rid of lock interactions. The

membership problem of BDPNs could be checked by individually verifying whether all the local runs of global runs in BDPNs are accepting. But this problem is not trivial, as the number of local runs is unbounded. To overcome this problem, for every $i \in \{1, ..., n\}$, we will compute a kind of automaton $\mathcal{A}_i$ such that $\mathcal{A}_i$ recognizes all the local configurations of $\mathcal{P}_i$ in BDPN from which $\mathcal{P}_i$ has an accepting local run and the set of local initial configurations of the dynamically created new instances by this local run. Finally, we show how to check whether a BDPN has an accepting global run from a global configuration by querying the obtained automata $\{\mathcal{A}_1, ..., \mathcal{A}_n\}$. Let us fix a L-BDPN $\mathcal{BM} = (\mathcal{M}, F_1, ..., F_n)$ s.t. $\mathcal{M} = (Act, \mathbb{L}, \mathcal{P}_1, ..., \mathcal{P}_n)$ and for every $i \in \{1, ..., n\}$, $\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i)$. Let us fix an index i for $i \in \{1, ..., n\}$.

4.1 Lock Structures

Given a global run T of a L-BDPN $\mathcal{BM}$, let T^c denote the subtree of T rooted by the local configuration c. Along a subtree T^c, an acquisition of a lock l without a corresponding release of l is called *final acquisition*, a release of a lock l without a corresponding acquisition of l is called *initial release*, an acquisition of a lock l together with its corresponding release is called *usage*. If a lock l is infinitely often used in T^c, then, T^c has an *infinite usage* of l. A *lock structure* $\widetilde{l}$ of a global run T at a local configuration c is a tuple $(L, A, AH, R, RH, U, \overset{\infty}{U})$, where L is the set of locks held by the instance at c, A (resp. R) is the set of final acquisition (resp. initial release) locks at c along the subtree T^c, U and $\overset{\infty}{U}$ are sets of usage and infinite usage locks at c along T^c, $AH \subseteq \mathbb{L} \times \mathbb{L}$ is an acquisition graph such that $(l, l') \in AH$ iff a usage of l' occurs after a final acquisition of l, $RH \subseteq \mathbb{L} \times \mathbb{L}$ is a release graph such that $(l, l') \in RH$ iff a usage of l occurs before an initial release of l'.

In an accepting global run of $\mathcal{BM}$ with lock structure $(L, A, AH, R, RH, U, \overset{\infty}{U})$, the following conditions have to hold:

- C_1 : Each initially-held lock l is neither used nor finally acquired unless there is a corresponding initial release of l, i.e., $(L \setminus R) \cap (U \cup A) = \emptyset$
- C_2 : A finally acquired lock l is not finally acquired again nor infinitely used;
- C_3 : The acquisition graph AH and release graph RH are acyclic.

Intuitively, the set of locks $(L \setminus R)$ denotes all the initially held locks that will not be released during the run. Thus, these locks should not be used $(L \setminus R) \cap U = \emptyset$ nor finally acquired $(L \setminus R) \cap A = \emptyset$. It is natural to have that a finally acquired lock cannot be acquired again nor infinitely used, since it will not be released in the future. The fact that the graphs RH and AH are acyclic ensures that the acquisition and release of locks do not have any cyclic dependence. Suppose AH has edges $(l_1, l_2), ..., (l_m, l_{m+1})$ for some $m > 1$ such that $l_1 = l_{m+1}$ (i.e., AH has a cycle), then for every $i : 1 \leq i \leq m$, T has a final acquisition of the lock l_i that should be performed before a usage of l_{i+1} (according to the definition of acquisition graphs). On the other hand, the usage of l_i should occur before the final acquisition of the lock l_i. Thus, the final acquisition of the lock l_i should be done before a final acquisition of the l_{i+1}. Since $l_1 = l_{m+1}$, then, the final acquisition of l_1 should be performed before a final acquisition of l_1. This means that T is not a global run of $\mathcal{BM}$. Therefore, AH should be acyclic. The fact that RH

should be acyclic can be explained similarly. Note that acyclic graphs AH and RH can guarantee that the run do not have any cyclic dependence of locks only if locks are accessed in a well-nested style.

A lock structure $(L, A, AH, R, RH, U, \overset{\infty}{U})$ is *consistent* iff both AH and RH are acyclic (the criteria C_3), and $(L \setminus R) \cap (U \cup A) = \emptyset$ (the criteria C_1), and $\overset{\infty}{U} \subseteq U$. Let $\mathcal{LS}$ be the set of all the consistent lock structures. Note that, according to the above explanation, the lock structure of an accepting global run of a L-BDPN $\mathcal{BM}$ should be consistent.

4.2 From L-BDPNs to BDPNs

We compute in this section a BDPN $\mathcal{BM}'$ from the L-BDPN $\mathcal{BM}$ such that $\mathcal{BM}$ has an accepting global run iff $\mathcal{BM}'$ has an accepting global run. We define the BDPN $\mathcal{BM}' = (\mathcal{M}', F'_1, ..., F'_n)$ s.t. $\mathcal{M}' = (\emptyset, \emptyset, \mathcal{P}'_1, ..., \mathcal{P}'_n)$, where for every $i \in \{1, ..., n\}$, $F'_i = \{(p, L, A, AH, R, RH, U, \overset{\infty}{U}) \in F_i \times \mathcal{LS} \mid U = \overset{\infty}{U}\}$, $\mathcal{P}'_i = (P_i \times \mathcal{LS}, \Gamma_i, \Delta')$, and Δ' is computed as follows: for every consistent lock structure $(L, A, AH, R, RH, U, \overset{\infty}{U}) \in \mathcal{LS}$,

α_1 : $p\gamma \overset{\tau}{\hookrightarrow}_i p_1\omega_1 \in \Delta_i$ iff $(p, L, A, AH, R, RH, U, \overset{\infty}{U})\gamma \hookrightarrow_i (p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\omega_1 \in \Delta'_i$;

α_2 : $p\gamma \overset{acq(l)}{\hookrightarrow}_i p_1\omega_1 \in \Delta_i$ and $l \in L$ iff

α_{21} : $(p, L \setminus \{l\}, A \cup \{l\}, AH \cup \{l\} \times U, R, RH, U, \overset{\infty}{U})\gamma \hookrightarrow_i (p_1, L, A, U, R, RH, U, \overset{\infty}{U})\omega_1 \in \Delta'_i$ if $l \notin R \cup A$,

α_{22} : $(p, L \setminus \{l\}, A, AH, R \setminus \{l\}, (RH \setminus \mathbb{L} \times \{l\}) \cup \{l\} \times R \setminus \{l\}, U \cup \{l\}, \overset{\infty}{U})\gamma \hookrightarrow_i (p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\ \omega_1 \in \Delta'_i$ if $l \in R$;

α_3 : $p\gamma \overset{rel(l)}{\hookrightarrow}_i p_1\omega_1 \in \Delta_i$ iff $(p, L', A, AH, R \cup \{l\}, RH, U, \overset{\infty}{U})\gamma \hookrightarrow_i (p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\omega_1 \in \Delta'_i$ and $L = L' \setminus \{l\}$;

α_4 : $p\gamma \overset{\tau}{\hookrightarrow}_i p_1\omega_1 \rhd p_2\omega_2 \in \Delta_i$ iff there exists $(\emptyset, A_1, AH_1, \emptyset, \emptyset, U_1, \overset{\infty}{U_1}) \in \mathcal{LS}$ s.t. $A \cap (A_1 \cup \overset{\infty}{U_1}) = \emptyset$, $\overset{\infty}{U} \cap A_1 = \emptyset$ and $(p, L, A \cup A_1, AH \cup AH_1, R, RH, U \cup U_1, \overset{\infty}{U} \cup \overset{\infty}{U_1})\gamma \hookrightarrow_i (p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\omega_1 \rhd (p_2, \emptyset, A_1, AH_1, \emptyset, \emptyset, U_1, \overset{\infty}{U_1})\omega_2 \in \Delta'_i$.

Suppose $(L, A, AH, R, RH, U, \overset{\infty}{U})$ is the lock structure at the local configuration $(p_1\omega_1\omega, L)$. Intuitively, the global runs of the BDPN $\mathcal{BM}'$ mimic the global runs of the L-BDPN $\mathcal{BM}$. The lock structures and their relations between local configurations in the global runs of the BDPN $\mathcal{BM}'$ are used to guarantee the criteria of locks and lock usage policy in the global runs of the L-BDPN $\mathcal{BM}$. Thus, we can get rid of lock interactions between instances. The intuition behind each rule is explained as follows.

If the transition rule $p\gamma \overset{\tau}{\hookrightarrow}_i p_1\omega_1$ of $\mathcal{P}_i$ is fired in a global run of the L-BDPN $\mathcal{BM}$ moving from the local configuration $(p\gamma\omega, L)$ to the local configuration $(p_1\omega_1\omega, L)$, then the lock structure at $(p_1\omega_1\omega, L)$ should be equal to the lock structure

at $(p\gamma\omega, L)$, since the action τ is not a lock-related action. Thus, we add the transition rule $(p, L, A, AH, R, RH, U, \overset{\infty}{U})\gamma \hookrightarrow_i (p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\ \omega_1$ into $\mathcal{P}'_i$ for every lock structure $(L, A, AH, R, RH, U, \overset{\infty}{U}) \in \mathcal{LS}$ which allows a global run of $\mathcal{BM}'$ to move from the local configuration $(p, L, A, AH, R, RH, U, \overset{\infty}{U})\gamma\omega$ to the local configuration $(p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\omega_1\omega$.

If the transition rule $p\gamma \overset{acq(l)}{\hookrightarrow}_i p_1\omega_1$ of $\mathcal{P}_i$ is used in a global run of the L-BDPN $\mathcal{BM}$ moving from the local configuration $(p\gamma\omega, L')$ to the local configuration $(p_1\omega_1\omega, L)$, then l should not be held at $(p\gamma\omega, L')$ and will be held at $(p_1\omega_1\omega, L)$ (i.e., $L' = L\setminus\{l\}$ and $l \in L$) due to the lock usage policy. The infinite usage locks at $(p\gamma\omega, L')$ and $(p_1\omega_1\omega, L)$ will be identical, as the local run moves only one step.

Moreover, if l is neither a final acquisition nor an initial release at $(p_1\omega_1\omega, L)$ (i.e., $l \notin R \cup A$), then l must be a final acquisition at $(p\gamma\omega, L')$ and the usage of locks U should be performed after the final acquisition of l. Thus, we add a transition rule $(p, L \setminus \{l\}, A \cup \{l\}, AH \cup \{l\} \times U, R, RH, U, \overset{\infty}{U})\gamma \hookrightarrow_i (p_1, L, A, U, R, RH, U, \overset{\infty}{U})\omega_1$ into $\mathcal{P}'_i$ in Item α_{21} which allows a global run of the BDPN to move from the local configuration $(p, L \setminus \{l\}, A \cup \{l\}, AH \cup \{l\} \times U, R, RH, U, \overset{\infty}{U})\gamma\omega$ to the local configuration $(p_1, L, A, U, R, RH, U, \overset{\infty}{U})\omega_1\omega$.

Otherwise if there is an initial release of l at $(p_1\omega_1\omega, L')$ (i.e., $l \in R$), then there is a usage of l at $(p\gamma\omega, L)$ and there is no initial release of l at $(p\gamma\omega, L)$ which implies that the release graph at $(p\gamma\omega, L)$ does not have any edge in $\mathbb{L} \times \{l\}$, and this usage of l should be performed before the initial release of locks in $R \setminus \{l\}$, i.e. the release graph at $(p\gamma\omega, L)$ is $(RH \setminus \mathbb{L} \times \{l\}) \cup \{l\} \times R$. Thus, we add a transition rule $(p, L \setminus \{l\}, A, AH, R \setminus \{l\}, (RH \setminus \mathbb{L} \times \{l\}) \cup \{l\} \times R, U \cup \{l\}, \overset{\infty}{U})\gamma \hookrightarrow_i (p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\omega_1 \in \Delta'_i$ at Item α_{22} which allows a global run of the BDPN to move from the local configuration $(p, L \setminus \{l\}, A, AH, R \setminus \{l\}, (RH \setminus \mathbb{L} \times \{l\}) \cup \{l\} \times R, U \cup \{l\}, \overset{\infty}{U})\gamma\omega$ to the local configuration $(p_1, L, A, U, R, RH, U, \overset{\infty}{U})\omega_1\omega$.

If the transition rule $p\gamma \overset{rel(l)}{\hookrightarrow}_i p_1\omega_1$ of $\mathcal{P}_i$ is fired in a global run of the L-BDPN $\mathcal{BM}$ moving from the local configuration $(p\gamma\omega, L')$ to the local configuration $(p_1\omega_1\omega, L)$, then the held lock at $(p\gamma\omega, L')$ will be released when moving to $(p_1\omega_1\omega, L)$ (i.e., $L = L' \setminus \{l\}$) and l is an initial release at $(p\gamma\omega, L')$. The sets of final acquisition locks, acquisition graphs and release graphs, the sets of usage locks and the sets of infinite usage locks are identical at $(p_1\omega_1\omega, L)$ and $(p\gamma\omega, L')$. Thus, we add the transition rule $(p, L', A, AH, R \cup \{l\}, RH, U, \overset{\infty}{U})\gamma \hookrightarrow_i (p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\omega_1$ into $\mathcal{P}'_i$ at Item α_3 which allows a global run of the BDPN to move from the local configuration $(p, L', A, AH, R \cup \{l\}, RH, U, \overset{\infty}{U})\gamma\omega$ to the local configuration $(p_1, L, A, U, R, RH, U, \overset{\infty}{U})\omega_1\omega$ for $L = L' \setminus \{l\}$.

If the transition rule $p\gamma \overset{\tau}{\hookrightarrow}_i p_1\omega_1 \triangleright p_2\omega_2$ of $\mathcal{P}_i$ is used in a global run of the L-BDPN $\mathcal{BM}$ moving from the local configuration $(p\gamma\omega, L')$ to the local configuration $(p_1\omega_1\omega, L)$ and creating a new instance that starts from $(p_2\omega_2, \emptyset)$, then the sets of held locks and the sets of initial release locks are identical at $(p\gamma\omega, L')$ and $(p_1\omega_1\omega, L)$. We also "guess" the lock structure $(\emptyset, A_1, AH_1, \emptyset, \emptyset, U_1, \overset{\infty}{U_1})$ along the subtree root-

ed by the new local configuration $(p_2\omega_2, \emptyset)$, where the set of held locks and the set of initial release locks at the new local configuration are empty. The lock structure $(\emptyset, A_1, AH_1, \emptyset, \emptyset, U_1, \overset{\infty}{U_1})$ will be abided by this subtree, i.e., the lock usages in the global run starting from $(p_2\omega_2, \emptyset)$ should satisfy the lock structure $(\emptyset, A_1, AH_1, \emptyset, \emptyset, U_1, \overset{\infty}{U_1})$. Moreover, the set of final acquisition locks A cannot be finally acquired (A_1) nor infinitely used $(\overset{\infty}{U_1})$ by the subtree rooted by $(p_2\omega_2, \emptyset)$, i.e., $A \cap (A_1 \cup \overset{\infty}{U_1}) = \emptyset$ (the criteria C_2). Indeed, if some instance of the global run from $(p_1\omega_1\omega, L)$ will finally acquire a lock l, then, the global run starting from $(p_2\omega_2, \emptyset)$ cannot finally nor infinitely acquire the lock l. Similarly, the set of infinitely used locks along the subtree rooted by $(p\omega_1\omega, L)$ cannot be finally acquired by the global run starting from $(p_2\omega_2, \emptyset)$, i.e., $\overset{\infty}{U} \cap A_1 = \emptyset$ (the criteria C_2). Therefore, the set of usage (resp. final acquisition, infinite usage, acquisition graph and release graph) locks at $(p\gamma\omega, L')$ is the union of the ones at $(p_1\omega_1\omega, L)$ and $(p_2\omega_2, \emptyset)$ respectively. Thus, we add the transition rule $(p, L, A \cup A_1, AH \cup AH_1, R, RH, U \cup U_1, \overset{\infty}{U} \cup \overset{\infty}{U_1})\gamma \hookrightarrow_i (p_1, L, A, AH, R, RH, U, \overset{\infty}{U})\omega_1 \triangleright (p_2, \emptyset, A_1, AH_1, \emptyset, \emptyset, U_1, \overset{\infty}{U_1})\omega_2$ into $\mathcal{P}'_i$ at Item α_4 which allows a global run of the BDPN to move from the local configuration $(p, L, A \cup A_1, AH \cup AH_1, R, RH, U \cup U_1, \overset{\infty}{U} \cup \overset{\infty}{U_1})\gamma\omega$ to the local configuration $(p_1, L, A, U, R, RH, U, \overset{\infty}{U})\omega_1\omega$ and create a new instance starting from $(p_2, \emptyset, A_1, AH_1, \emptyset, \emptyset, U_1, \overset{\infty}{U_1})\omega_2$.

To guarantee that the set of infinite usage locks associated with each control location of the BDPN $\mathcal{BM}'$ is exactly the set of infinite usage locks used in the global run, for every $i \in \{1, ..., n\}$, we let F'_i be the set $\{(p, L, A, AH, R, RH, U, \overset{\infty}{U}) \in F_i \times \mathcal{LS} \mid U = \overset{\infty}{U}\}$.

Thus, we get the following theorem.

Theorem 3. *The L-BDPN $\mathcal{BM}$ has an accepting global run iff the BDPN $\mathcal{BM}'$ has an accepting global run.*

4.3 Membership problem of BDPNs

Let us fix a BDPN $\mathcal{BM} = (\mathcal{M}, F_1, ..., F_n)$ s.t. $\mathcal{M} = (\emptyset, \emptyset, \mathcal{P}_1, ..., \mathcal{P}_n)$ and for every $i \in \{1, ..., n\}$, $\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i)$, and fix an index i for $i \in \{1, ..., n\}$. For every transition rule of the form $p_0\gamma_0 \overset{\tau}{\hookrightarrow}_i p_1\gamma_1 \triangleright p_2\omega_2$, $(p_2\omega_2, \emptyset)$ is called a dynamically created local initial configuration (DCLIC). Let $\mathcal{D}_i$ be the set of DCLICs of the DPDS $\mathcal{P}_i$ and $\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i$. Let $\mathcal{L}(\mathcal{P}_i)$ denote the set of all the tuples $(c, D) \in P_i \times \Gamma_i^* \times 2_i^{\mathcal{D}}$ s.t. $\mathcal{P}_i$ has an accepting local run starting from c and D is exactly the set of local configurations of the newly created instances during this accepting local run. In this section, we show how to check whether a BDPN $\mathcal{BM}$ has an accepting global run or not. For this, we compute for every $i \in \{1, ..., n\}$, a kind of finite automaton $\mathcal{A}_i$ which exactly accepts $L(\mathcal{P}_i)$. Then, we show how to check whether the BDPN $\mathcal{BM}$ has an accepting global run by querying the automata $\{\mathcal{A}_1, ..., \mathcal{A}_n\}$.

To finitely represent (infinite) sets of local configurations of DPDSs and DCLICs generated by DPDSs, we use Multi-automata.

Definition 5. *A Multi-automaton* (MA) *is a tuple $\mathcal{A}_i = (Q_i, \Gamma_i, \delta_i, I_i, Acc_i)$, where Q_i is a finite set of states, $I_i \subseteq P_i$ is a finite set of initial states corresponding to the control locations of the DPDS $\mathcal{P}_i$, $Acc_i \subseteq Q_i$ is a finite set of final states, $\delta_i \subseteq (Q_i \times \Gamma_i) \times 2^{\mathcal{D}_i} \times Q_i$ is a finite set of transition rules.*

We write $p \xrightarrow{\gamma/D}_i q$ instead of $(p, \gamma, D, q) \in \delta_i$, where D is a set of DCLICs. We define the relation $\longrightarrow_i^* \subseteq (Q_i \times \Gamma_i^*) \times 2^{\mathcal{D}_i} \times Q_i$ as the smallest relation s.t.: (1) $q \xrightarrow{\epsilon/\emptyset}_i^* q$ for every $q \in Q_i$, (2) if $q \xrightarrow{\gamma/D}_i q_1$ and $q_1 \xrightarrow{\omega/D_k}_i^* q_2$, then $q \xrightarrow{\gamma\omega/D \cup \bigcup_{k=1}^{m} D_k}_i^* q_2$. Let $\mathcal{L}(\mathcal{A}_i)$ be the set of tuples $(p\omega, D) \in P_i \times \Gamma_i^* \times 2^{\mathcal{D}_i}$ s.t. $p \xrightarrow{\omega/D}_i^* q$ for some $q \in Acc_i$.

Lemma 1. *[23] For every $i \in \{1, ..., n\}$, we can construct a MA $\mathcal{A}_i$ in time $\mathbf{O}(|\Delta_i| \cdot |\Gamma_i| \cdot |P_i|^3 \cdot 2^{|\mathcal{D}_i|})$ such that $\mathcal{L}(\mathcal{A}_i) = \mathcal{L}(\mathcal{P}_i)$.*

By Lemma 1, for every $i \in \{1, ..., n\}$, we can compute a MA $\mathcal{A}_i$ such that $\mathcal{L}(\mathcal{A}_i) = \mathcal{L}(\mathcal{P}_i)$. Having the set of MAs $\{\mathcal{A}_1, ..., \mathcal{A}_n\}$ in hand, we can check whether the BDPN $\mathcal{BM}$ has an accepting global run starting from a given global initial configuration $c \in P_i \times \Gamma_i^*$ for some $i \in \{1, ..., n\}$ as follows: we check whether there exists $D \subseteq \mathcal{D}$ such that $(c, D) \in \mathcal{L}(\mathcal{P}_i)$ and for every $d \in D$ with $d \in P_j \times \Gamma_j^*$, there exists $D_d \subseteq \mathcal{D}$ such that $(d, D_d) \in \mathcal{L}(\mathcal{P}_j)$, etc. This condition is recursive. It can be solved, because the set D of DCLICs is finite. However, this procedure is not efficient. To obtain a more efficient procedure, we will compute the largest set of DCLICs $\mathcal{D}_{fp} \subseteq \mathcal{D}$ such that for every $c \in \mathcal{D}_{fp}$ with $c \in P_i \times \Gamma_i^*$ for any $i \in \{1, ..., n\}$, the BDPN $\mathcal{BM}$ has an accepting global run from c. Thus, to check whether or not the BDPN $\mathcal{BM}$ has an accepting global run from c, it is sufficient to check whether or not there exists $D \subseteq \mathcal{D}_{fp}$ such that $(c, D) \in \mathcal{L}(\mathcal{P}_i)$.

Let $\{\mathcal{A}_1, ..., \mathcal{A}_n\}$ be the set of MAs s.t. for every $i \in \{1, ..., n\}$, $\mathcal{A}_i = (Q_i, \Gamma_i, \delta_i, I_i, Acc_i)$ is the MA computed by Lemma 1. Intuitively, $\mathcal{D}_{fp}$ should be equal to the set of local configurations $c \in \mathcal{D}$ s.t. there exists $D \subseteq \mathcal{D}_{fp}$ s.t. $(c, D) \in \mathcal{L}(\mathcal{A}_i)$ with $c \in P_i \times \Gamma_i^*$. Thus, $\mathcal{D}_{fp}$ can be defined as the greatest fixpoint of the function $F(X) = \{c \in \mathcal{D} \mid \exists D \subseteq X \text{ s.t. } (c, D) \in \mathcal{L}(\mathcal{A}_i) \text{ with } c \in P_i \times \Gamma_i^*\}$. This set can then be computed iteratively as follows: $\mathcal{D}_{fp} = \bigcap_{j \geq 0} D_j$, where $D_0 = \mathcal{D}$ and $D_{j+1} = \{c \in \mathcal{D} \mid \exists D \subseteq D_j \text{ s.t. } (c, D) \in \mathcal{L}(\mathcal{A}_i) \text{ with } c \in P_i \times \Gamma_i^*\}$. Since $\mathcal{D} \times \mathcal{LS}$ is a finite set, and for every $j \geq 0$, D_{j+1} is a subset of D_j, there always exists a fixpoint $m \geq 0$ such that $D_m = D_{m+1}$. Then, we can get that $\mathcal{D}_{fp} = D_m$.

For every $(c, D) \in \mathcal{D} \times 2^{\mathcal{D}}$, to avoid checking whether $(c, D) \in \mathcal{L}(\mathcal{A}_i)$ with $c \in P_i \times \Gamma_i^*$ at each step when computing $D_0, D_1, ...$, we can compute all these tuples that satisfy this condition once and store them in a hash table. We can show that whether or not $(c, D) \in \mathcal{L}(\mathcal{A}_i)$ can be decided in time $\mathbf{O}(|c| \cdot |\delta_i| \cdot |Q_i| \cdot 2^{|\mathcal{D}_i|})$ for every $i \in \{1, ..., n\}$. Let $\mathbf{I(c)}$ denote the index i of the local configuration $c \in P_i \times \Gamma_i^*$. Thus, we can get the hash table in time $\mathbf{O}(\sum_{c \in \mathcal{D}}(|c| \cdot |\delta_{\mathbf{I(c)}}| \cdot |Q_{\mathbf{I(c)}}| \cdot 2^{|\mathcal{D}_{\mathbf{I(c)}}|}))$. Given D_j and the hash table, we can compute D_{j+1} in time $\mathbf{O}(\sum_{c \in \mathcal{D}} 2^{|\mathcal{D}_{\mathbf{I(c)}}|})$. Thus we can get $\mathcal{D}_{fp}$ in time $\mathbf{O}(\sum_{c \in \mathcal{D}}(|c| \cdot |\delta_{\mathbf{I(c)}}| \cdot |Q_{\mathbf{I(c)}}| \cdot 2^{|\mathcal{D}_{\mathbf{I(c)}}|} + |\mathcal{D}|^2 \cdot 2^{|\mathcal{D}|}))$.

Lemma 2. *We can compute $\mathcal{D}_{fp}$ in time $\mathbf{O}(\sum_{c \in \mathcal{D}}(|c| \cdot |\delta_{\mathbf{I(c)}}| \cdot |Q_{\mathbf{I(c)}}| \cdot 2^{|\mathcal{D}_{\mathbf{I(c)}}|} + |\mathcal{D}|^2 \cdot 2^{|\mathcal{D}|}))$ s.t. for every $c \in \mathcal{D}$, the BDPN $\mathcal{BM}$ has an accepting global run from c iff $c \in \mathcal{D}_{fp}$.*

From Lemma 1 and Lemma 2, we get that:

Theorem 4. *The membership problem of BDPNs is decidable in time* $\mathbf{O}(\sum_{i=1}^{n} |\Delta_i| \cdot |\Gamma_i| \cdot |P_i|^3 \cdot 2^{|\mathcal{D}_i|} + \sum_{c \in \mathcal{D}}(|c| \cdot |P_{\mathbf{I(c)}}|^3 \cdot |\Gamma_{\mathbf{I(c)}}| \cdot 2^{2|\mathcal{D}_{\mathbf{I(c)}}|} + |\mathcal{D}|^2 \cdot 2^{|\mathcal{D}|}))$.

The complexity follows the fact that for each $i : 1 \leq i \leq n$, $|\delta_i|$ is at most $\mathbf{O}(|P_i|^2 \cdot |\Gamma_i| \cdot 2^{|\mathcal{D}_i|})$ and $|Q_i|$ is at most $\mathbf{O}(|P_i|)$. From Theorem 4 and Theorem 3, we get that:

Theorem 5. *The membership problem of L-BDPNs is decidable in time* $\mathbf{O}(\sum_{i=1}^{n} |\Delta_i| \cdot |\Gamma_i| \cdot |P_i|^3 \cdot |\mathcal{LS}|^4 \cdot 2^{|\mathcal{D}_i| \cdot |\mathcal{LS}|} + \sum_{c \in \mathcal{D}}(|c| \cdot |P_{\mathbf{I(c)}}|^3 \cdot |\mathcal{LS}|^3 \cdot |\Gamma_{\mathbf{I(c)}}| \cdot 2^{2|\mathcal{D}_{\mathbf{I(c)}}| \cdot |\mathcal{LS}|} + |\mathcal{D}|^2 \cdot |\mathcal{LS}|^2 \cdot 2^{|\mathcal{D}| \cdot |\mathcal{LS}|}))$.

The complexity follows the fact that for each $i : 1 \leq i \leq n$, $|P'_i|$ in the BDPN $\mathcal{BM}'$ is at most $\mathbf{O}(|P_i| \cdot |\mathcal{LS}|)$, $|\Delta'_i|$ in $\mathcal{BM}'$ is at most $\mathbf{O}(|\Delta_i| \cdot |\mathcal{LS}|)$ and the number of DCLICs of $\mathcal{P}'_i$ in $\mathcal{BM}'$ is at most $\mathbf{O}(|\mathcal{D}_i| \cdot |\mathcal{LS}|)$. From Theorem 5 and Theorem 2, we get that:

Theorem 6. *The model-checking problem of L-DPNs against single-indexed LTL is decidable in time in time* $\mathbf{O}(\sum_{i=1}^{n} |\Delta_i| \cdot |\Gamma_i| \cdot |P_i|^3 \cdot |\mathcal{LS}|^4 \cdot 2^{4|f_i|} \cdot 2^{|\mathcal{D}_i| \cdot |\mathcal{LS}|} + \sum_{c \in \mathcal{D}}(|c| \cdot |P_{\mathbf{I(c)}}|^3 \cdot |\mathcal{LS}|^3 \cdot |\Gamma_{\mathbf{I(c)}}| \cdot 2^{3|f_i|} \cdot 2^{2|\mathcal{D}_{\mathbf{I(c)}}| \cdot |\mathcal{LS}|} + |\mathcal{D}|^2 \cdot |\mathcal{LS}|^2 \cdot 2^{|\mathcal{D}| \cdot |\mathcal{LS}|}))$.

The complexity follows the fact that for each $i : 1 \leq i \leq n$, f_i introduces the factor $2^{|f_i|}$ into $|P_i|$ and $|\Delta_i|$.

5 Related work

DPNs and L-DPNs: DPN was introduced in [4]. Several other works use DPN and its extensions to model multi-threaded programs [4, 9, 20, 21, 27]. All these works consider only reachability issues. Ground Tree Rewrite Systems [10] and process rewrite systems [5, 22] are two models of multi-threaded programs with procedure calls and thread creation. However, [22] considers only reachability and [10, 5] consider only subclasses of LTL. In the work [23], we show that single-indexed LTL and CTL model-checking for DPNs is decidable and present automata-based approaches for single-indexed LTL and CTL model-checking for DPNs. However, the model DPN in [23] do not allow communications via locks. In this work, we extend the approach of [23] to single-indexed LTL model-checking for L-DPNs by introducing lock structures. However, we cannot extend the results to single-indexed CTL model-checking for L-DPNs. Since the model-checking problem of L-DPNs against single-indexed CTL will be reduced to the membership problem of *alternating* L-BDPNs whose local runs are trees and each branch of these trees can finally acquire the same locks. Thus, the resulting *alternating* L-BDPNs do not use locks in a well-nested style.

Pushdown networks with communication between processes are studied in e.g. [3, 7, 1, 25, 6]. These works consider systems with a fixed number of threads. [17, 18] use parallel flow graphs to model multi-threaded programs. However, all these works consider only reachability. [28] considers safety properties of multi-threaded programs.

Lock structures: [15] was the first to introduces (forward) acquisition histories that contain only the set of held locks and the acquisition graphs to check pairwise reachability properties and a subclass of LTL on pushdown networks communicating via well-nested locks without thread creation. [13, 14] extends the forward acquisition histories

of [15] with backwards acquisition histories (i.e., the release graphs) to check single-indexed LTL and fragments of LTL and CTL properties for two threads communicating via well-nested locks. [18] extended the acquisition histories of [15] to check pairwise reachability properties of programs with reentrant monitors and dynamic thread creation. [16] proposes an acquisition history based decision procedure to check whether a pushdown network without dynamic thread creation satisfies properties represented by a kind of finite automaton. The decision procedure of [16] uses only one reachability query of each pushdown system. While, in the worst case, [15] has to perform an exponential number of individual reachability queries of each pushdown system to handle a temporal operator. In order to compute predecessor sets of regular sets of configurations of L-DPNs, [20] introduces acquisition structures for L-DPNs, an extension of acquisition histories. [20] shows that the computation of backward reachable configurations of L-DPNs can be reduced to compute the one of DPNs [4]. However, all these works do not consider dynamic thread creation and/or cannot check more complex properties that could be expressed in single-indexed LTL formulas.

Following [20], in this work, we introduce lock structures which are an extension of acquisition structures with the set of infinite usage locks. This is because [20] consider pairwise reachability whose runs are finite, while this work considers model-checking single-indexed LTL properties in which the runs of L-DPNs can be infinite (for liveness properties). However, an infinitely used lock cannot be finally acquired by instances.

[11] introduces bounded lock chains, a generalization of well-nested locks. If the number of acquired locks between the acquisition of each lock and its corresponding release is bounded, then, this pushdown network uses locks in terms of *bounded lock chain*. [11] shows that pairwise reachability is decidable for pushdown networks without thread creation when the lock chains are bounded. [12] shows that model-checking for pushdown networks without thread creation against LTL is decidable when LTL formulas use only the following operators: next-time $\mathbf{X}$, eventually $\mathbf{F}$, infinitely-often F_∞, conjunction $\wedge$ and disjunction $\vee$, and the problem is undecidable when the LTL formula use until $\mathbf{U}$ or always G operators. [6] introduces another kind of lock access style, called *contextual locking*. A program uses locks in terms of *contextual locking* if all the acquired locks in each procedure are released before this procedure returns. [6] proposed a decision procedure to check pairwise reachability for pushdown networks (without dynamic thread creation) with contextual locking. [19] extends the result of [6] showing that pairwise reachability for L-DPNs with contextual locking is decidable. However, all these works consider only the pairwise reachability problem. It is unknown whether our approach can check single-indexed LTL properties for L-DPNs with bounded lock chains or contextual locking. We leave these questions as future work.

References

1. M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, pages 356–371, 2008.
2. A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In *CONCUR'97*. LNCS 1243, 1997.

3. A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *POPL'03*. ACM, 2003.
4. A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR*, pages 473–487, 2005.
5. L. Bozzelli, M. Kretínský, V. Rehák, and J. Strejcek. On decidability of LTL model checking for process rewrite systems. *Acta Inf.*, 46(1), 2009.
6. R. Chadha, P. Madhusudan, and M. Viswanathan. Reachability under contextual locking. In *TACAS*, pages 437–450, 2012.
7. S. Chaki, E. M. Clarke, N. Kidd, T. W. Reps, and T. Touili. Verifying concurrent message-passing c programs with recursive calls. In *TACAS*, pages 334–349, 2006.
8. J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithm for model checking pushdown systems. In *CAV'00*, volume 1885 of *LNCS*, 2000.
9. T. M. Gawlitza, P. Lammich, M. Müller-Olm, H. Seidl, and A. Wenner. Join-lock-sensitive forward reachability analysis for concurrent programs with dynamic process creation. In *VMCAI*, pages 199–213, 2011.
10. S. Göller and A. W. Lin. The complexity of verifying ground tree rewrite systems. In *LICS*, pages 279–288, 2011.
11. V. Kahlon. Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise cfl-reachability for threads communicating via locks. In *LICS*, 2009.
12. V. Kahlon. Reasoning about threads with bounded lock chains. In *CONCUR*, pages 450–465, 2011.
13. V. Kahlon and A. Gupta. An automata-theoretic approach for model checking threads for LTL properties. In *LICS*, pages 101–110, 2006.
14. V. Kahlon and A. Gupta. On the analysis of interacting pushdown systems. In *POPL*, pages 303–314, 2007.
15. V. Kahlon, F. Ivancic, and A. Gupta. Reasoning about threads communicating via locks. In *Computer Aided Verification*, 2005.
16. N. Kidd, P. Lammich, T. Touili, and T. W. Reps. A decision procedure for detecting atomicity violations for communicating processes with locks. In *SPIN*, pages 125–142, 2009.
17. P. Lammich and M. Müller-Olm. Precise fixpoint-based analysis of programs with thread-creation and procedures. In *CONCUR*, pages 287–302, 2007.
18. P. Lammich and M. Müller-Olm. Conflict analysis of programs with procedures, dynamic thread creation, and monitors. In *SAS*, pages 205–220, 2008.
19. P. Lammich, M. Müller-Olm, H. Seidl, and A. Wenner. Contextual locking for dynamic pushdown networks. In *SAS*, 2013.
20. P. Lammich, M. Müller-Olm, and A. Wenner. Predecessor sets of dynamic pushdown networks with tree-regular constraints. In *CAV*, pages 525–539, 2009.
21. D. Lugiez. Forward analysis of dynamic network of pushdown systems is easier without order. *Int. J. Found. Comput. Sci.*, 22(4):843–862, 2011.
22. R. Mayr. Process rewrite systems. *Inf. Comput.*, 156(1-2):264–286, 2000.
23. F. Song and T. Touili. Model checking dynamic pushdown networks. In *APLAS*, pages 33–49, 2013.
24. F. SONG and T. Touili. LTL Model-Checking for Dynamic Pushdown Networks Communicating via Locks. Technical report, `ftp://222.73.57.93/CONCUR14.pdf`, 2014.
25. T. Touili and M. F. Atig. Verifying parallel programs with dynamic communication structures. *Theor. Comput. Sci.*, 411(38-39):3460–3468, 2010.
26. M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):183–221, 1986.
27. A. Wenner. Weighted dynamic pushdown networks. In *ESOP*, pages 590–609, 2010.
28. E. Yahav. Verifying safety properties of concurrent java programs using 3-valued logic. In *POPL*, pages 27–40, 2001.