

# Parameterized Verification of Asynchronous Shared-Memory Systems

Javier Esparza<sup>1</sup>, Pierre Ganty<sup>2,\*</sup>, and Rupak Majumdar<sup>3</sup>

<sup>1</sup> TU Munich

<sup>2</sup> IMDEA Software Institute

<sup>3</sup> MPI-SWS

**Abstract.** We characterize the complexity of the safety verification problem for parameterized systems consisting of a leader process and arbitrarily many anonymous and identical contributors. Processes communicate through a shared, bounded-value register. While each operation on the register is atomic, there is no synchronization primitive to execute a sequence of operations atomically.

We analyze the complexity of the safety verification problem when processes are modeled by finite-state machines, pushdown machines, and Turing machines. The problem is coNP-complete when all processes are finite-state machines, and is PSPACE-complete when they are pushdown machines. The complexity remains coNP-complete when each Turing machine is allowed boundedly many interactions with the register. Our proofs use combinatorial characterizations of computations in the model, and in case of pushdown-systems, some language-theoretic constructions of independent interest.

## 1 Introduction

We conduct a systematic study of the complexity of safety verification for *parameterized asynchronous shared-memory systems*. These systems consist of a *leader* process and arbitrarily many identical *contributors*, processes with no identity, running at arbitrarily relative speeds and subject to faults (a process can crash). The shared-memory consists of a read/write register that all processes can access to perform either a read operation or a write operation. The register is bounded: the set of values that can be stored is finite. We do insist that read/write operations execute atomically but sequences of operations do not: no process can conduct an atomic sequence of reads and writes while excluding all other processes. The parameterized verification problem for these systems asks to check if a safety property holds no matter how many contributors are present. Our model subsumes the case in which all processes are identical by having the leader process behave like yet another contributor. The presence of a distinguished leader adds (strict) generality to the problem.

We analyze the complexity of the safety verification problem when leader and contributors are modeled by finite state machines, pushdown machines, and even Turing machines. Using combinatorial properties of the model that allow simulating arbitrarily many contributors using finitely many ones, we show that if leader and contributors are finite-state machines the problem is coNP-complete. The case in which leader and contributors are pushdown machines was first considered by Hague [17], who gave a coNP

---

\* Supported by the Spanish projects with references TIN2010-20639 and TIN2012-39391-C04.

lower bound and a 2EXPTIME upper bound. We close the gap and prove that the problem is PSPACE-complete. Our upper bound requires several novel language-theoretic constructions on bounded-index approximations of context-free languages. Finally, we address the bounded safety problem, i.e., deciding if no error can be reached by computations in which no contributor nor the leader execute more than a given number  $k$  of steps (this does not bound the length of the computation, since the number of contributors is unbounded). We show that (if  $k$  is given in unary) the problem is coNP-complete not only for pushdown machines, but also for arbitrary Turing machines. Thus, the safety verification problem when the leader and contributors are poly-time Turing machines is also coNP-complete.

These results show that non-atomicity substantially reduces the complexity of verification. In the atomic case, contributors can ensure that they are the only ones that receive a message: the first contributor that reads the message from the store can also erase it within the same atomic action. This allows the leader to distribute identities to contributors. As a consequence, the safety problem is at least PSPACE-hard for state machines, and undecidable for pushdown machines (in the atomic case, the safety problem of two pushdown machines is already undecidable). A similar argument shows that the bounded safety problem is PSPACE-hard. In contrast, we get several coNP upper bounds, which opens the way to the application of SAT-solving or SMT-techniques.

Besides intellectual curiosity, our work on this model is motivated by practical distributed protocols implemented on wireless sensor networks. In these systems, a central co-ordinator (the base station) communicates with an arbitrary number of mass-produced tiny agents (or motes) that run concurrently and asynchronously. The motes have limited computational power, and for some systems such as vehicular networks anonymity is a requirement [20]. Further, they are susceptible to crash faults. Implementing atomic communication primitives in this setting is expensive and can be problematic: for instance, a process might crash while holding a lock. Thus, protocols in these systems work asynchronously and without synchronization primitives. Our algorithms provide the foundations for safety verification of these systems.

Full proofs of our results can be found in the associated technical report [13].

*Related Works.* Parameterized verification problems have been extensively studied both theoretically and practically. It is a computationally hard problem: the reachability problem is undecidable even if each process has a finite state space [2]. For this reason, special cases have been extensively studied. They vary according to the main characteristics of the systems to verify like the communication topology of the processes (array, tree, unordered, etc); their communication primitives (shared memory, unreliable broadcasts, (lossy) queues, etc); or whether processes can distinguish from each other (using ids, a distinguished process, etc). Prominent examples include broadcast protocols [12,14,9,8], where finite-state processes communicate via broadcast messages, asynchronous programs [15,23], where finite-state processes communicate via unordered channels, finite-state processes communicating via ordered channels [1], micro architectures [21], cache coherence protocols [10,6], communication protocols [11], multithreaded shared-memory programs [5,7,19,22].

Besides the model of Hague [17], the closest model to ours that has been previously studied [16] is that of distributed computing with identity-free, asynchronous

processors and non-atomic registers. The emphasis there was the development of distributed algorithm primitives such as time-stamping, snapshots, and consensus, using either unbounded registers or an unbounded number of bounded registers.

It was left open if these primitives can be implemented using a bounded number of bounded registers. Our decidability results indicate that this is not possible: the safety verification problem would be undecidable if such primitives could be implemented.

## 2 Formal Model: Non-atomic Networks

We describe our formal model, called *non-atomic networks*. We take a language-theoretic view, identifying a system with the language of its executions.

*Preliminaries.* A *labeled transition system* (LTS) is a quadruple  $\mathcal{T} = (\Sigma, Q, \delta, q_0)$ , where  $\Sigma$  is a finite set of *action labels*,  $Q$  is a (non necessarily finite) set of *states*,  $q_0 \in Q$  is the *initial state*, and  $\delta \subseteq Q \times \Sigma \cup \{\varepsilon\} \times Q$  is the *transition relation*, where  $\varepsilon$  is the empty string. We write  $q \xrightarrow{a} q'$  for  $(q, a, q') \in \delta$ . For  $\sigma \in \Sigma^*$ , we write  $q \xrightarrow{\sigma} q'$  if there exist  $q_1, \dots, q_n \in Q$  and  $a_0, \dots, a_n \in \Sigma \cup \{\varepsilon\}$ ,  $q \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \dots q_n \xrightarrow{a_n} q'$  such that  $a_0 \dots a_n = \sigma$ . The sequence  $q \dots q'$  is called a *path* and  $\sigma$  its *label*. A *trace* of  $\mathcal{T}$  is a sequence  $\sigma \in \Sigma^*$  such that  $q_0 \xrightarrow{\sigma} q$  for some  $q \in Q$ . Define  $L(\mathcal{T})$ , the *language* of  $\mathcal{T}$ , as the set of traces of  $\mathcal{T}$ . Note that  $L(\mathcal{T})$  is *prefix closed*:  $L(\mathcal{T}) = \text{Pref}(L(\mathcal{T}))$  where  $\text{Pref}(L) = \{s \mid \exists u: s u \in L\}$

To model concurrent executions of LTSs, we introduce two operations on languages: the shuffle and the asynchronous product. The *shuffle* of two words  $x, y \in \Sigma^*$  is the language  $x \sqcup y = \{x_1 y_1 \dots x_n y_n \in \Sigma^* \mid \text{each } x_i, y_i \in \Sigma^* \text{ and } x = x_1 \dots x_n \wedge y = y_1 \dots y_n\}$ . The shuffle of two languages  $L_1, L_2$  is the language  $L_1 \sqcup L_2 = \bigcup_{x \in L_1, y \in L_2} x \sqcup y$ . Shuffle is associative, and so we can write  $L_1 \sqcup \dots \sqcup L_n$  or  $\sqcup_{i=1}^n L_i$ .

The *asynchronous product* of two languages  $L_1 \subseteq \Sigma_1^*$  and  $L_2 \subseteq \Sigma_2^*$ , denoted  $L_1 \parallel L_2$ , is the language  $L$  over the alphabet  $\Sigma = \Sigma_1 \cup \Sigma_2$  such that  $w \in L$  iff the projections of  $w$  to  $\Sigma_1$  and  $\Sigma_2$  belong to  $L_1$  and  $L_2$ , respectively.<sup>1</sup> If a language consists of a single word, e.g.  $L_1 = \{w_1\}$ , we abuse notation and write  $w_1 \parallel L_2$ . Asynchronous product is also associative, and so we write  $L_1 \parallel \dots \parallel L_n$  or  $\parallel_{i=1}^n L_i$ .

Let  $\mathcal{T}_1, \dots, \mathcal{T}_n$  be LTSs, where  $\mathcal{T}_i = (\Sigma_i, Q_i, \delta_i, q_{0i})$ . The *interleaving*  $\sqcup_{i=1}^n \mathcal{T}_i$  is the LTS with actions  $\bigcup_{i=1}^n \Sigma_i$ , set of states  $Q_1 \times \dots \times Q_n$ , initial state  $(q_{01}, \dots, q_{0n})$ , and a transition  $(q_1, \dots, q_n) \xrightarrow{a} (q'_1, \dots, q'_n)$  iff  $(q_i, a, q'_i) \in \delta_i$  for some  $1 \leq i \leq n$  and  $q_j = q'_j$  for every  $j \neq i$ . Interleaving models parallel composition of LTSs that do not communicate at all. The language  $L(\mathcal{T}_1 \sqcup \dots \sqcup \mathcal{T}_n)$  of the interleaving is  $\sqcup_{i=1}^n L(\mathcal{T}_i)$ .

The *asynchronous parallel composition*  $\parallel_{i=1}^n \mathcal{T}_i$  of  $\mathcal{T}_1, \dots, \mathcal{T}_n$  is the LTS having  $\bigcup_{i=1}^n \Sigma_i$  as set of actions,  $Q_1 \times \dots \times Q_n$  as set of states,  $(q_{01}, \dots, q_{0n})$  as initial state, and a transition  $(q_1, \dots, q_n) \xrightarrow{a} (q'_1, \dots, q'_n)$  if and only if

1.  $a \neq \varepsilon$  and for all  $1 \leq i \leq n$  either  $a \notin \Sigma_i$  and  $q_i = q'_i$  or  $a \in \Sigma_i$  and  $(q_i, a, q'_i) \in \delta_i$ , or;
2.  $a = \varepsilon$ , and there is  $1 \leq i \leq n$  such that  $(q_i, \varepsilon, q'_i) \in \delta_i$  and  $q_j = q'_j$  for every  $j \neq i$ .

Asynchronous parallel composition models the parallel composition of LTSs in which an action  $a$  must be simultaneously executed by every LTSs having  $a$  in its alphabet.  $L(\mathcal{T}_1 \parallel \dots \parallel \mathcal{T}_n)$ , the language of the asynchronous parallel composition, is  $\parallel_{i=1}^n L(\mathcal{T}_i)$ .

<sup>1</sup> Observe that the  $L_1 \parallel L_2$  depends on  $L_1, L_2$  **and** also their underlying alphabet  $\Sigma_1$  and  $\Sigma_2$ .

*Non-atomic Networks.* We fix a finite non-empty set  $\mathcal{G}$  of *global values*. A *read-write alphabet* is any set of the form  $A \times \mathcal{G}$ , where  $A$  is a set of *read* and *write actions*, or just *reads* and *writes*. We denote a letter  $(a, g) \in A \times \mathcal{G}$  by  $a(g)$ , and write  $\mathcal{G}(a_1, \dots, a_n)$  instead of  $\{a_1, \dots, a_n\} \times \mathcal{G}$ .

In what follows, we consider LTSs over read-write alphabets. We fix two LTSs  $\mathcal{D}$  and  $\mathcal{C}$ , called the *leader* and the *contributor*, with alphabets  $\mathcal{G}(r_d, w_d)$  and  $\mathcal{G}(r_c, w_c)$ , respectively, where  $r_d, r_c$  are called reads and  $w_c, w_d$  are called writes. We write  $w_\star$  (respectively,  $r_\star$ ) to stand for either  $w_c$  or  $w_d$  (respectively,  $r_c$  or  $r_d$ ). We also assume that for each value  $g \in \mathcal{G}$  there is a transition in the leader or contributor which reads or writes  $g$  (if not, the value is never used and is removed from  $\mathcal{G}$ ).

Additionally, we fix an LTS  $\mathcal{S}$  called a *store*, whose states are the global values of  $\mathcal{G}$  and whose transitions, labeled with the read-write alphabet, represent possible changes to the global values on reads and writes. No read is enabled initially. Formally, the *store* is an LTS  $\mathcal{S} = (\Sigma, \mathcal{G} \cup \{g_0\}, \delta_{\mathcal{S}}, g_0)$ , where  $\Sigma = \mathcal{G}(r_d, w_d, r_c, w_c)$ ,  $g_0$  is a designated initial value not in  $\mathcal{G}$ , and  $\delta_{\mathcal{S}}$  is the set of transitions  $g \xrightarrow{r_\star(g)} g$  and  $g' \xrightarrow{w_\star(g)} g$  for all  $g \in \mathcal{G}$  and all  $g' \in \mathcal{G} \cup \{g_0\}$ . Observe that fixing  $\mathcal{D}$  and  $\mathcal{C}$  also fixes  $\mathcal{S}$ .

**Definition 1.** Given a pair  $(\mathcal{D}, \mathcal{C})$  of a leader  $\mathcal{D}$  and contributor  $\mathcal{C}$ , and  $k \geq 1$ , define  $\mathcal{N}_k$  to be the LTS  $\mathcal{D} \parallel \mathcal{S} \parallel \sqcup_k \mathcal{C}$ , where  $\sqcup_k \mathcal{C}$  is  $\sqcup_{i=1}^k \mathcal{C}$ . The (non-atomic)  $(\mathcal{D}, \mathcal{C})$ -network  $\mathcal{N}$  is the set  $\{\mathcal{N}_k \mid k \geq 1\}$ , with language  $L(\mathcal{N}) = \bigcup_{k=1}^{\infty} L(\mathcal{N}_k)$ . We omit the prefix  $(\mathcal{D}, \mathcal{C})$  when it is clear from the context.

Notice that  $L(\mathcal{N}_k) = L(\mathcal{D}) \parallel L(\mathcal{S}) \parallel \sqcup_k L(\mathcal{C})$  and  $L(\mathcal{N}) = L(\mathcal{D}) \parallel L(\mathcal{S}) \parallel \sqcup_{\infty} L(\mathcal{C})$ , where  $\sqcup_{\infty} L(\mathcal{C})$  is given by  $\bigcup_{k=1}^{\infty} \sqcup_k L(\mathcal{C})$ .

*The Safety Verification Problem.* A trace of a  $(\mathcal{D}, \mathcal{C})$ -network  $\mathcal{N}$  is unsafe if it ends with an occurrence of  $w_c(\#)$ , where  $\#$  is a special value of  $\mathcal{G}$ . Intuitively, an occurrence of  $w_c(\#)$  models that the contributor raises a flag because some error has occurred. A  $(\mathcal{D}, \mathcal{C})$ -network  $\mathcal{N}$  is *safe* iff its language contains no unsafe trace, namely  $L(\mathcal{N}) \cap \Sigma^* w_c(\#) = \emptyset$ . (We could also require the leader to write  $\#$ , or to reach a certain state; all these conditions are easily shown equivalent.)

Given a machine  $M$  having an LTS semantics over some read-write alphabet, we denote its LTS by  $\llbracket M \rrbracket$ . Given machines  $M_D$  and  $M_C$  over read-write alphabets, The *safety verification problem* for machines  $M_D$  and  $M_C$  consists of deciding if the  $(\llbracket M_D \rrbracket, \llbracket M_C \rrbracket)$ -network is safe. Notice that the size of the input is the size of the machines, and not the size of the LTSs thereof, which might even be infinite.

Our goal is to characterize the complexity of the safety verification problem considering various types of machines for the leader and the contributors. We first establish some fundamental combinatorial properties of non-atomic networks.

### 3 Simulation and Monotonicity

We prove two fundamental combinatorial properties of non-atomic networks: the Simulation and Monotonicity Lemmas. Informally, the Simulation Lemma states that a leader cannot distinguish an unbounded number of contributors from the parallel composition of at most  $|\mathcal{G}|$  *simulators*—LTSs derivable from the contributors, one for each value

of  $\mathcal{G}$ . The Monotonicity Lemma states that non-minimal traces (with respect to a certain subword order) can be removed from a simulator without the leader “noticing”, and, symmetrically, non-maximal traces can be removed from the leader without the simulators “noticing”.

### 3.1 Simulation

*First Writes and Useless Writes.* Let  $\sigma$  be a trace. The *first write* of  $g$  in  $\sigma$  by a contributor is the first occurrence of  $w_c(g)$  in  $\sigma$ . A *useless write* of  $g$  by a contributor is any occurrence of  $w_c(g)$  that is immediately overwritten by another write. For technical reasons, we additionally assume that useless writes are not first writes.

*Example 1.* In a network trace  $w_d(g_1)_1 w_c(g_2)_2 w_c(g_3)_3 r_d(g_3)_4 w_c(g_2)_5 w_c(g_1)_6$  where we have numbered occurrences,  $w_c(g_2)_2$  is a first write of  $g_2$ , and  $w_c(g_2)_5$  is a useless write of  $g_2$  (even though  $w_c(g_2)_2$  is immediately overwritten).

We make first writes and useless writes explicit by adding two new actions  $f_c$  and  $u_c$  to our LTSs, and adequately adapting the store.

**Definition 2.** *The extension of an LTS  $\mathcal{T} = (\mathcal{G}(r, w), Q, \delta, q_0)$  is the LTS  $\mathcal{T}^E = (\mathcal{G}(r, w, f, u), Q, \delta^E, q_0)$ , where  $f, u$  are the first write and useless write actions, respectively, and*

$$\delta^E = \delta \cup \{(q, f(g), q'), (q, u(g), q') \mid (q, w(g), q') \in \delta\} .$$

We define an *extended store*, whose states are triples  $(g, W, b)$ , where  $g \in \mathcal{G}$ ,  $W: \mathcal{G} \rightarrow \{0, 1\}$  is the *write record*, and  $b \in \{0, 1\}$  is the *useless flag*. Intuitively,  $W$  records the values written by the contributors so far. If  $W(g) = 0$ , then a write to  $g$  must be a first write, and otherwise a regular write or a useless write. The useless flag is set to 1 by a useless write, and to 0 by other writes. When set to 1, the flag prevents the occurrence of a read. The flag only ensures that between a useless write and the following write no read happens, i.e., that a write tagged as useless will indeed be semantically useless. A regular or first write may be semantically useless or not.

**Definition 3.** *The extended store is the LTS  $S^E = (\Sigma_E, \mathcal{G}_E, \delta_{S^E}, c_0)$  where*

- $\Sigma_E = \mathcal{G}(r_d, w_d, r_c, w_c, f_c, u_c)$ ;
- $\mathcal{G}_E$  is the set of triples  $(g, W, b)$ , where  $g \in \mathcal{G} \cup \{g_0\}$ ,  $W: \mathcal{G} \rightarrow \{0, 1\}$ , and  $b \in \{0, 1\}$ ;
- $c_0$  is the triple  $(g_0, W_0, 0)$ , where  $W_0(g) = 0$  for every  $g \in \mathcal{G}$ ;
- $\delta_{S^E}$  has a transition  $(g, W, b) \xrightarrow{a} (g', W', b')$  where  $g' \in \mathcal{G}$  iff one of the following conditions hold:
  - $a = r_*(g)$ ,  $g' = g$ ,  $W' = W$ , and  $b = b' = 0$ ;
  - $a = w_d(g')$ ,  $W' = W$  and  $b' = 0$ ;
  - $a = f_c(g')$ ,  $W(g') = 0$ ,  $W' = W[W(g')/1]$ , and  $b' = 0$ ;
  - $a = w_c(g')$ ,  $W(g') = 1$ ,  $W' = W$ , and  $b' = 0$ ;
  - $a = u_c(g')$ ,  $W(g') = 1$ ,  $W' = W$ , and  $b' = 1$ .

*The extension of  $N_k$  is  $N_k^E = \mathcal{D} \parallel S^E \parallel \sqcup_k C^E$  and the extension of  $\mathcal{N}$  is the set  $\mathcal{N}^E = \{N_k^E \mid k \geq 1\}$ . The languages  $L(N_k^E)$  and  $L(\mathcal{N}^E)$  are defined as in Def. 1.*

It follows immediately from this definition that if  $v \in L(N^E)$  then the sequence  $v'$  obtained of replacing every occurrence of  $f_c(g), u_c(g)$  in  $v$  by  $w_c(g)$  belongs to  $L(N)$ . Conversely, every trace  $v'$  of  $L(N)$  can be transformed into a trace  $v$  of  $L(N^E)$  by adequately replacing some occurrences of  $w_c(g)$  by  $f_c(g)$  or  $u_c(g)$ .

In the sequel, we use sequences of first writes to partition sets of traces. Define  $\mathcal{Y}$  to be the (finite) set of sequences over  $\mathcal{G}(f_c)$  with no repetitions. By the very idea of “first writes” no sequence of  $\mathcal{Y}$  writes the same value twice, hence no word in  $\mathcal{Y}$  is longer than  $|\mathcal{G}|$ . Also define  $\mathcal{Y}_\#$  to be those words of  $\mathcal{Y}$  which ends with  $f_c(\#)$ . Given  $\tau \in \mathcal{Y}$ , define  $P_\tau$  to be the language given by  $(\Sigma_E \setminus \mathcal{G}(f_c))^* \sqcup \tau$ .  $P_\tau$  contains all the sequences over  $\Sigma_E$  in which the subsequence of first writes is exactly  $\tau$ . For  $S \subseteq \mathcal{Y}$ ,  $P_S = \bigcup_{\sigma \in S} P_\sigma$ .

*The Copycat Lemma.* Intuitively, a *copycat* contributor is one that follows another contributor in all its actions: it reads or writes the same value immediately after the other reads or writes. Informally, the copycat lemma states that any trace of a non-atomic network can be extended with copycat contributors.

Consider first the non-extended case. Clearly, for every trace of  $\mathcal{N}_k$  there is a trace of  $\mathcal{N}_{k+1}$  in which the leader and the first  $k$  contributors behave as before, and the  $(k+1)$ -th contributor behaves as a copycat of one of the first  $k$  contributors, say the  $i$ -th: if the  $i$ -th contributor executes a read  $r_c(g)$ , then the  $(k+1)$ -th contributor executes the same read *immediately after*, and the same for a write.

*Example 2.* Consider the trace  $r_c(g_0) w_d(g_1) r_c(g_1) w_c(g_2)$  of  $\mathcal{D} \parallel \mathcal{S} \parallel \mathcal{C}$ . Then the sequence  $r_c(g_0)^2 w_d(g_1) r_c(g_1)^2 w_c(g_2)^2$  is a trace of  $\mathcal{D} \parallel \mathcal{S} \parallel (\mathcal{C} \sqcup \mathcal{C})$ .

For the case of extended networks, a similar result holds, but the copycat copies a first write by a regular write: if the  $i$ -th contributor executes an action other than  $f_c(g)$ , the copycat contributor executes the same action immediately after, but if the  $i$ -th contributor executes  $f_c(g)$ , then the copycat executes  $w_c(g)$ .

**Definition 4.** We say  $u \in \mathcal{G}(r_d, w_d)^*$  is compatible with a multiset  $M = \{v_1, \dots, v_k\}$  of words over  $\mathcal{G}(f_c, w_c, u_c, r_c)$  (possibly containing multiple copies of a word) iff  $u \parallel L(S^E) \parallel \sqcup_{i=1}^k v_i \neq \emptyset$ . Let  $\tau \in \mathcal{Y}$ . We say  $u$  is compatible with  $M$  following  $\tau$  iff  $P_\tau \cap (u \parallel L(S^E) \parallel \sqcup_{i=1}^k v_i) \neq \emptyset$ .

**Lemma 1.** Let  $u \in \mathcal{G}(r_d, w_d)^*$  and let  $M$  be a multiset of words over  $\mathcal{G}(r_c, f_c, w_c, u_c)$ . If  $u$  is compatible with  $M$ , then  $u$  is compatible with every  $M'$  obtained by erasing symbols from  $\mathcal{G}(r_c)$  and  $\mathcal{G}(u_c)$  from the words of  $M$ .

*Proof.* Erasing reads and useless writes (that no one reads) by contributors does not affect the sequence of values written to the store and read by someone, hence compatibility is preserved.  $\square$

**Lemma 2 (Copycat Lemma).** Let  $u \in \mathcal{G}(r_d, w_d)^*$ , let  $M$  be a multiset over  $L(\mathcal{C}^E)$  and let  $v' \in M$ . Given a prefix  $v$  of  $v'$  we have that if  $u$  is compatible with  $M$ , then  $u$  is compatible with  $M \oplus v[f_c(g)/w_c(g)]$ .<sup>2</sup>

<sup>2</sup> Throughout the paper, we use  $\{\}, \oplus, \ominus$ , and  $\geq$  for the multiset constructor, union, difference and inclusion, respectively. The word  $w[a/b]$  results from  $w$  by replacing all occurrences of  $a$  by  $b$ .

*Example 3.*  $r_d(g_1)$  is compatible with  $f_c(g_1)f_c(g_2)$ . By the Copycat Lemma  $r_d(g_1)$  is also compatible with  $\{f_c(g_1)f_c(g_2), w_c(g_1)w_c(g_2)\}$ . Indeed,  $f_c(g_1)w_c(g_1)r_d(g_1)f_c(g_2)w_c(g_2) \in L(S^E)$  is a trace (even though  $f_c(g_2)$  is useless).

*The Simulation Lemma.* The simulation lemma states that we can replace unboundedly many contributors by a finite number of LTSs that “simulate” them. In particular the network is safe iff its simulation is safe.

Let  $v \in L(C^E)$ . Let  $\#v$  be the number of times that actions of  $\mathcal{G}(f_c, w_c)$  occur in  $v$ , minus one if the last action of  $v$  belongs to  $\mathcal{G}(f_c, w_c)$ . E.g.,  $\#v = 1$  for  $v = f_c(g_1)r_c(g_1)$  but  $\#v = 0$  for  $v = r_c(g_1)f_c(g_1)$ . The next lemma is at the core of the simulation theorem.

**Lemma 3.** *Let  $u \in L(\mathcal{D})$  and let  $M = \{v_1, \dots, v_k\}$  be a multiset over  $L(C^E)$  compatible with  $u$ . Then  $u$  is compatible with a multiset  $\tilde{M}$  over  $L(C^E) \cap \mathcal{G}(r_c, u_c)^* \mathcal{G}(f_c, w_c)$ .*

*Proof.* Since  $u$  is compatible with  $M$ ,  $u \parallel L(S^E) \parallel \sqcup_{i=1}^k v_i \neq \emptyset$ . Lemma 1 shows that we can drop from  $M$  all the  $v_i$  such that  $v_i \in \mathcal{G}(r_c, u_c)^*$ . Further, define  $\#M = \sum_{i=1}^k \#v_i$ . We proceed by induction on  $\#M$ . If  $\#M = 0$ , then all the words of  $M$  belong to  $\mathcal{G}(r_c, u_c)^* \mathcal{G}(f_c, w_c)$ , and we are done. If  $\#M > 0$ , then there is  $v_i \in M$  such that  $v_i = \alpha_i \sigma \beta_i$ , where  $\alpha_i \in \mathcal{G}(r_c, u_c)^*$ ,  $\sigma \in \mathcal{G}(f_c, w_c)$ , and  $\beta_i \neq \varepsilon$ . Let  $g$  be the value written by  $\sigma$ , and let  $v_{k+1} = \alpha_i w_c(g)$ . By Lemma 2,  $u$  is compatible with  $\{v_1, \dots, v_{k+1}\}$ , and so there is  $v' \in u \parallel L(S^E) \parallel \sqcup_{i=1}^{k+1} v_i$  in which the write  $\sigma$  of  $v_i$  occurs in  $v'$  immediately before the write of  $v_{k+1}$ . We now let the writes occur in the reverse order, which amounts to replacing  $v_i$  by  $v'_i = \alpha_i u_c(g) \beta_i$  and  $v_{k+1}$  by  $v'_{k+1} = \alpha_i \sigma$ . This yields a new multiset  $M' = M \ominus \{v_i\} \oplus \{v'_i, v'_{k+1}\}$  compatible with  $u$ . Since  $\#M' = \#M - 1$ , we then apply the induction hypothesis to  $M'$ , obtain  $\tilde{M}$  and we are done.  $\square$

**Definition 5.** *For all  $g \in \mathcal{G}$ , let  $L_g = L(C^E) \cap \mathcal{G}(r_c, u_c)^* f_c(g)$ . Define  $S_g$  be an LTS over  $\mathcal{G}(r_c, u_c, f_c, w_c)$  such that  $L(S_g) = \text{Pref}(L_g \cdot w_c(g)^*)$ . Define the LTS  $N^S = \mathcal{D} \parallel S^E \parallel \sqcup_{g \in \mathcal{G}} S_g$  which we call the simulation of  $N^E$ .*

**Lemma 4.** *Let  $u \in L(\mathcal{D})$  and let  $M = \{v_1, \dots, v_k\}$  be a multiset over  $L(C^E) \cap \mathcal{G}(r_c, u_c)^* \mathcal{G}(f_c, w_c)$  compatible with  $u$ . Then  $u$  is compatible with a set  $S = \{s_g\}_{g \in \mathcal{G}}$  where  $s_g \in L(S_g)$ .*

*Proof.* Let us partition the multiset  $M$  as  $\{M_g\}_{g \in \mathcal{G}}$  such that  $M_g$  contains exactly the traces of  $M$  ending with  $f_c(g)$  or  $w_c(g)$ . Note that some  $M_g$  might be empty. Each non-empty  $M_g$  is of the form  $M_g = \{x_1 f_c(g), x_2 w_c(g), \dots, x_n w_c(g)\}$  where  $n \geq 1$ , and  $x_i \in \mathcal{G}(r_c, u_c)^*$  for every  $1 \leq i \leq n$ . Define  $M'_g$  as empty if  $M_g$  is empty, and  $M'_g$  as  $M_g$  together with  $n - 1$  copies of  $x_1 w_c(g)$ . The copycat lemma shows that  $u$  is compatible with  $\bigoplus_{g \in \mathcal{G}} M'_g$ . Let us now define the multiset  $M''_g$  to be empty if  $M'_g$  is empty, and the multiset of exactly  $n$  elements given by  $x_1 f_c(g)$  and  $n - 1$  copies of  $x_1 w_c(g)$  if  $M'_g$  is not empty. Again we show that  $u$  is compatible with  $\bigoplus_{g \in \mathcal{G}} M''_g$ . The reason is that the number  $n - 1$  of actions  $w_c(g)$  in each  $M''_g$  does not change (compared to  $M_g$ ) and each  $w_c(g)$  action can happen as soon as  $f_c(g)$  has occurred.

Now define  $S$  consisting of one trace  $s_g$  for each  $g \in \mathcal{G}$  such that  $s_g = \varepsilon$  if  $M''_g = \emptyset$ ; and  $s_g = x_1 f_c(g) w_c(g)^{n-1}$  if  $M''_g$  consists of  $x_1 f_c(g)$  and  $n - 1$  copies of  $x_1 w_c(g)$ .

We have that  $u$  is compatible with  $S$  because the number of  $f_c(g)$  and  $w_c(g)$  actions in  $M''_g$  and  $s_g$  does not change and each  $w_c(g)$  action can happen as soon as  $f_c(g)$  has

occurred. Note that it need not be the case that  $s_g \in L(C^E)$ . However each  $s_g \in L(S_g)$  (recall that each  $L(S_g)$  is prefix closed).  $\square$

**Corollary 1.** *Let  $u \in L(\mathcal{D})$  and let  $M = \{v_1, \dots, v_k\}$  be a multiset over  $L(C^E)$  compatible with  $u$ . Then  $u$  is compatible with a set  $S = \{s_g\}_{g \in \mathcal{G}}$  where  $s_g \in L(S_g)$ .*

In Lemmas 1,2,3 and 4 and Corollary 1 compatibility is preserved. We can further show that it is preserved following a given sequence of first writes. For example, in Lem. 3 if  $u$  is compatible with  $M$  following  $\tau$  then  $u$  is compatible with  $\tilde{M}$  following  $\tau$ .

**Lemma 5 (Simulation Lemma).** *Let  $\tau \in \mathcal{Y}$ :*

$$L(\mathcal{N}^E) \cap P_\tau \neq \emptyset \quad \text{iff} \quad L(\mathcal{N}^S) \cap P_\tau \neq \emptyset .$$

*Proof.* ( $\Rightarrow$ ): The hypothesis and the definition of  $\mathcal{N}^E$  shows that there is  $k \geq 1$  such that  $P_\tau \cap (L(\mathcal{D}) \parallel L(S^E) \parallel \sqcup_k L(C^E)) \neq \emptyset$ .

Therefore we conclude that there exists  $u \in L(\mathcal{D})$  and  $M = \{v_1, \dots, v_k\}$  over  $L(C^E)$  such that  $u$  is compatible with  $M$  following  $\tau$ . Corollary 1 shows that  $u$  is compatible following  $\tau$  with a set  $S = \{s_g\}_{g \in \mathcal{G}}$  where  $s_g \in L(S_g)$ . Therefore we have  $P_\tau \cap (u \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(S_g)) \neq \emptyset$ , hence that  $P_\tau \cap (L(\mathcal{D}) \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(S_g)) \neq \emptyset$  and finally that  $P_\tau \cap L(\mathcal{N}^S) \neq \emptyset$ .

( $\Leftarrow$ ): The hypothesis and the definition of  $\mathcal{N}^S$  shows that  $P_\tau \cap (L(\mathcal{D}) \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(S_g)) \neq \emptyset$ . Hence we find that there exists  $u \in L(\mathcal{D})$  and a set  $\{x_g\}_{g \in \mathcal{G}}$  where  $x_g \in L(S_g)$  such that  $P_\tau \cap (u \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} x_g) \neq \emptyset$ . The prefix closure of  $L(S_g)$  shows that either  $x_g$  does not have a first write or  $x_g = v_g f_c(g) w_c(g)^{n_g}$  for some  $v_g f_c(g) \in L_g$  and  $n_g \in \mathbb{N}$ . In the former case, that is  $x_g \in \mathcal{G}(r_c, u_c)^*$ , Lemma 1 shows that discarding the trace does not affect compatibility. Then define the multiset  $M$  containing for each remaining trace  $x_g = v_g f_c(g) w_c(g)^{n_g}$  the trace  $v_g f_c(g)$  and  $n_g$  traces  $v_g w_c(g)$ .  $M$  contains no other element. Using a copycat-like argument, it is easy to show  $M$  is compatible with  $u$  and further that compatibility follows  $\tau$ . Finally, because  $v_g f_c(g) \in L(C^E) \cap \mathcal{G}(r_c, u_c)^* \mathcal{G}(f_c)$  and because  $C^E$  is the extension of  $C$  we find that every trace of  $M$  is also a trace of  $C^E$ , hence that there exists  $k \geq 1$  such that  $P_\tau \cap (L(\mathcal{D}) \parallel L(S^E) \parallel \sqcup_k L(C^E)) \neq \emptyset$ , and finally that  $L(\mathcal{N}^E) \cap P_\tau \neq \emptyset$ .  $\square$

Let us now prove an equivalent safety condition.

**Proposition 1.** *A  $(\mathcal{D}, C)$ -network  $\mathcal{N}$  is safe iff  $L(\mathcal{N}^S) \cap P_{\mathcal{I}_\#} = \emptyset$ .*

*Proof.* From the semantics of non-atomic networks,  $\mathcal{N}$  is unsafe if and only if  $L(\mathcal{N}) \cap (\Sigma^* w_c(\#)) \neq \emptyset$ , equivalently,  $L(\mathcal{N}^E) \cap (\Sigma_E^* f_c(\#)) \neq \emptyset$  (by definition of extension), which in turn is equivalent to  $L(\mathcal{N}^E) \cap P_{\mathcal{I}_\#} \neq \emptyset$  (by definition of  $P_{\mathcal{I}_\#}$ ), if and only if  $L(\mathcal{N}^S) \cap P_{\mathcal{I}_\#} \neq \emptyset$  (by the simulation lemma).  $\square$

### 3.2 Monotonicity

Before stating the monotonicity lemma, we need some language-theoretic definitions. For an alphabet  $\Sigma$ , define the *subword ordering*  $\leq \subseteq \Sigma^* \times \Sigma^*$  on words as  $u \leq v$  iff  $u$  results from  $v$  by deleting some occurrences of symbols. Let  $L \subseteq \Sigma^*$ , define  $S \subseteq L$  to be



- *cover* of  $L$  if for every  $u \in L$  there is  $v \in S$  such that  $u \leq v$ ;
- *support* of  $L$  if for every  $u \in L$  there is  $v \in S$  such that  $v \leq u$ .

Observe that for every  $u, v \in S$  such that  $u < v$ : if  $S$  is a cover then so is  $S \setminus \{u\}$ , and if  $S$  is a support then so is  $S \setminus \{v\}$ .

Recall that  $\mathcal{N}^S = \mathcal{D} \parallel S^E \parallel \sqcup_{g \in \mathcal{G}} S_g$ . It is convenient to introduce a fourth, redundant component that does not change  $L(\mathcal{N}^S)$ , but exhibits important properties of it. Recall that the leader cannot observe the reads of the contributors, and does not read the values introduced by useless writes. We introduce a local copy  $S_D^E$  of the store with alphabet  $\mathcal{G}(r_d, w_d, f_c, w_c)$  that behaves like  $S^E$  for writes and first writes of the contributors, but has neither contributor reads nor useless writes in its alphabet. Formally:

**Definition 6.** *The leader store  $S_D^E$  is the LTS  $(\Sigma_D^E, \mathcal{G}_D^E, \delta_D^E, c_0)$ ,*

- $\Sigma_D^E = \mathcal{G}(r_d, w_d, f_c, w_c)$ ;
- $\mathcal{G}_D^E$  is the set of pairs  $(g, W)$ , where  $g \in \mathcal{G} \cup \{g_0\}$  and  $W : \mathcal{G} \rightarrow \{0, 1\}$ ;
- $c_0$  is the pair  $(g_0, W_0)$ , where  $W_0(g) = 0$  for every  $g \in \mathcal{G}$ ;
- $\delta_D^E$  has a transition  $(g, W) \xrightarrow{a} (g', W')$  where  $g' \in \mathcal{G}$  iff one of the following conditions hold: a)  $a = w_d(g')$  and  $W' = W$ ; b)  $a = r_d(g)$ ,  $g' = g$ , and  $W' = W$ ; c)  $a = f_c(g')$ ,  $W(g') = 0$ , and  $W' = W[W(g')/1]$ ; d)  $a = w_c(g')$ ,  $W(g') = 1$ , and  $W' = W$ .

It follows easily from this definition that  $L(S_D^E)$  is the projection of  $L(S^E)$  onto  $\Sigma_D^E$ , and so  $L(S^E) = L(S_D^E) \parallel L(S^E)$  holds. Now, define  $\mathcal{DS} = \mathcal{D} \parallel S_D^E$ , we find that:

$$\begin{aligned}
 L(\mathcal{N}^S) &= L(\mathcal{D}) \parallel S^E \parallel \sqcup_{g \in \mathcal{G}} S_g && \text{def. 5} \\
 &= L(\mathcal{D}) \parallel L(S_D^E) \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(S_g) \\
 &= L(\mathcal{D}) \parallel S_D^E \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(S_g) \\
 &= L(\mathcal{DS}) \parallel S^E \parallel \sqcup_{g \in \mathcal{G}} S_g && (1)
 \end{aligned}$$

**Lemma 6 (Monotonicity Lemma).** *Let  $\tau \in \mathcal{Y}$  and let  $\hat{L}_\tau$  be a cover of  $L(\mathcal{DS}) \cap P_\tau$ . For every  $g \in \mathcal{G}$ , let  $\underline{L}_g$  be a support of  $L_g$ , and let  $\underline{S}_g$  be an LTS such that  $L(\underline{S}_g) = \text{Pref}(\underline{L}_g \cdot w_c^*(g))$ :*

$$(L(\mathcal{DS}) \cap P_\tau) \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(S_g) \neq \emptyset \text{ iff } \hat{L}_\tau \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(\underline{S}_g) \neq \emptyset .$$

The proof of the monotonicity lemma breaks down into monotonicity for the contributors (Lemma 7) and for the leader (Lemma 9).

**Lemma 7 (Contributor Monotonicity Lemma).** *For every  $g \in \mathcal{G}$ , let  $\underline{L}_g$  be a support of  $L_g$ , and let  $\underline{S}_g$  be an LTS such that  $L(\underline{S}_g) = \text{Pref}(\underline{L}_g w_c^*(g))$ . Let  $u \in \mathcal{G}(r_d, w_d)^*$  and  $\tau \in \mathcal{Y}$ :*

$$(u \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(S_g)) \cap P_\tau \neq \emptyset \text{ iff } (u \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(\underline{S}_g)) \cap P_\tau \neq \emptyset .$$

*Proof.* ( $\Leftarrow$ ): It suffices to observe that since  $\underline{L}_g \subseteq L_g$  we have  $L(\underline{S}_g) \subseteq L(S_g)$  and we are done. ( $\Rightarrow$ ): Since  $L_g \subseteq \mathcal{G}(r_c, u_c)^* f_c(g)$  and  $\underline{L}_g \subseteq L_g$  we find that for every word  $w' \in L_g \setminus \underline{L}_g$  there exists a word  $w \in \underline{L}_g$  resulting from  $w'$  by erasing symbols in  $\mathcal{G}(u_c, r_c)$ . Hence, Lemma 1 shows that erasing symbols in  $\mathcal{G}(u_c, r_c)$  does not affect compatibility. The proof concludes by observing that compatibility is further preserved for  $\tau$ , and we are done.  $\square$

The leader monotonicity lemma requires the following technical observation.

**Lemma 8.** *Let  $\tau \in \mathcal{Y}$  and  $L \subseteq \mathcal{G}(r_c, f_c, w_c, u_c)^*$  satisfying the following condition: if  $\alpha f_c(g)\beta_1\beta_2 \in L$ , then  $\alpha f_c(g)\beta_1 w_c(g)\beta_2 \in L$ . For every  $v, v' \in P_\tau \cap L(S_{\mathcal{D}}^E)$ :*

$$\text{if } v \parallel L(S^E) \parallel L \neq \emptyset \quad \text{and } v' \geq v, \quad \text{then } v' \parallel L(S^E) \parallel L \neq \emptyset .$$

Because  $v, v' \in P_\tau \cap L(S_{\mathcal{D}}^E)$  over alphabet  $\Sigma_{\mathcal{D}}^E = \mathcal{G}(r_d, w_d, f_c, w_c)$  and  $v' \geq v$  we find that  $v$  can be obtained from  $v'$  by erasing factors that are necessarily of the form  $w_\star(g) r_d(g)^*$  or  $r_d(g)$ . In particular  $v, v' \in P_\tau$  shows that  $\text{Proj}_{\mathcal{G}(f_c)}(v) = \text{Proj}_{\mathcal{G}(f_c)}(v') = \tau$ .<sup>3</sup> The proof of Lem. 8 is by induction on the number of those factors.

**Lemma 9 (Leader Monotonicity Lemma).** *Let  $\tau \in \mathcal{Y}$  and  $L \subseteq \mathcal{G}(r_c, f_c, w_c, u_c)^*$  satisfying: if  $\alpha f_c(g)\beta_1\beta_2 \in L$ , then  $\alpha f_c(g)\beta_1 w_c(g)\beta_2 \in L$ . For every cover  $\hat{L}_\tau$  of  $P_\tau \cap L(\mathcal{DS})$ :*

$$(P_\tau \cap L(\mathcal{DS})) \parallel L(S^E) \parallel L \neq \emptyset \quad \text{iff} \quad \hat{L}_\tau \parallel L(S^E) \parallel L \neq \emptyset .$$

*Proof.* ( $\Leftarrow$ ): It follows from  $\hat{L}_\tau \subseteq (P_\tau \cap L(\mathcal{DS}))$ . ( $\Rightarrow$ ): We conclude from the hypothesis that there exists  $w \in P_\tau \cap L(\mathcal{DS})$  such that  $w \parallel L(S^E) \parallel L \neq \emptyset$ . Since  $\hat{L}_\tau$  is a cover  $P_\tau \cap L(\mathcal{DS})$ , we find that there exists  $w' \in \hat{L}_\tau$  such that  $w' \geq w$  and  $w' \in P_\tau \cap L(\mathcal{DS})$ . Finally,  $\mathcal{DS} = \mathcal{D} \parallel S_{\mathcal{D}}^E$  shows that  $w, w' \in P_\tau \cap L(S_{\mathcal{D}}^E)$ , hence that  $w' \parallel L(S^E) \parallel L \neq \emptyset$  following Lem. 8, and finally that  $\hat{L}_\tau \parallel L(S^E) \parallel L \neq \emptyset$  because  $w' \in \hat{L}_\tau$ .  $\square$

## 4 Complexity of Safety Verification of Non-atomic Networks

Recall that the *safety verification problem* for machines  $M_D$  and  $M_C$  consists in deciding if the  $(\llbracket M_D \rrbracket, \llbracket M_C \rrbracket)$ -network is safe. Notice that the size of the input is the size of the machines, and not the size of its LTSs, which might even be infinite. We study the complexity of safety verification for different machine classes.

Given two classes of machines  $\mathcal{D}, \mathcal{C}$  (like finite-state machines or push-down machines, see below), we define *the class of  $(\mathcal{D}, \mathcal{C})$ -networks* as the set  $\{(\llbracket D \rrbracket, \llbracket C \rrbracket)\text{-network} \mid D \in \mathcal{D}, C \in \mathcal{C}\}$  and denote by  $\text{Safety}(\mathcal{D}, \mathcal{C})$  the restriction of the safety verification problem to pairs of machines  $M_D \in \mathcal{D}$  and  $M_C \in \mathcal{C}$ . We study the complexity of the problem when leader and contributors are *finite-state machines* (FSM) and *pushdown machines* (PDM).<sup>4</sup> In this paper a FSM is just another name for a finite-state LTS, and the LTS  $\llbracket A \rrbracket$  of a FSM  $A$  is  $A$ , i.e.  $\llbracket A \rrbracket = A$ . We define the size  $|A|$  of a FSM  $A$  as the size of its transition relation. A (read/write) *pushdown machine* is a tuple  $P = (\mathcal{Q}, \mathcal{G}(r, w), \Gamma, \Delta, \gamma_0, q_0)$ , where  $\mathcal{Q}$  is a finite set of *states* including the *initial state*  $q_0$ ,  $\Gamma$  is a *stack alphabet* that contains the *initial stack symbol*  $\gamma_0$ , and  $\Delta \subseteq (\mathcal{Q} \times \Gamma) \times (\mathcal{G}(r, w) \cup \{\varepsilon\}) \times (\mathcal{Q} \times \Gamma^*)$  is a set of *rules*. A *configuration* of a PDM  $P$  is a pair  $(q, \gamma) \in \mathcal{Q} \times \Gamma^*$ . The LTS  $\llbracket P \rrbracket$  over  $\mathcal{G}(r, w)$  associated to  $P$  has  $\mathcal{Q} \times \Gamma^*$  as states,  $(q_0, \gamma_0)$  as initial state, and a transition  $(q, \gamma y) \xrightarrow{a} (q', \gamma' y')$  iff  $(q, \gamma, a, q', \gamma') \in \Delta$ . Define the size of a rule  $(q, \gamma, a, q', \gamma') \in \Delta$  as  $|\gamma'| + 5$  and the size  $|P|$  of a PDM as the sum of the size of rules in  $\Delta$ .

<sup>3</sup>  $\text{Proj}_{\Sigma'}(w)$  returns the projection of  $w$  onto alphabet  $\Sigma'$ .

<sup>4</sup> We also define FSA and PDA as the automaton (i.e. language acceptor) counterpart of FSM and PDM, respectively. As expected, definitions are identical except for an additional accepting component given by a subset of states in which the automaton accepts.

*Determinism.* We show that lower bounds (hardness) for the safety verification problems can be achieved already for *deterministic* machines. An LTS  $\mathcal{T}$  over a read-write alphabet is *deterministic* if for every state  $s$  and every pair of transitions  $s \xrightarrow{a_1} s_1$  and  $s \xrightarrow{a_2} s_2$ , if  $s_1 \neq s_2$  then  $a_1$  and  $a_2$  are reads, and they read different values. Intuitively, for any state of a store  $\mathcal{S}$ , a deterministic LTS  $\mathcal{T}$  can take at most one transition in  $\mathcal{S} \parallel \mathcal{T}$ . A  $(\mathcal{D}, \mathcal{C})$ -network is deterministic if  $\mathcal{D}$  and  $\mathcal{C}$  are deterministic LTSs. Given a class  $\mathcal{X}$  of machines, we denote by  $d\mathcal{X}$  the subclass of machines  $M$  of  $\mathcal{X}$  such that  $\llbracket M \rrbracket$  is a deterministic LTS over the read-write alphabet. Notice that this notion does not coincide with the usual definition of a deterministic automaton.

The observation is that a network with non-deterministic processes can be simulated by deterministic ones while preserving safety; intuitively, the inherent non-determinism of interleaving can simulate non-deterministic choice in the machines.

**Lemma 10 (Determinization Lemma).** *There is a polynomial-time procedure that takes a pair  $(\mathcal{D}, \mathcal{C})$  of LTSs and outputs a pair  $(\mathcal{D}', \mathcal{C}')$  of deterministic LTSs such that the  $(\mathcal{D}, \mathcal{C})$ -network is safe iff the  $(\mathcal{D}', \mathcal{C}')$ -network is safe.*

We prove the lemma by eliminating non-determinism as follows. Suppose  $\mathcal{D}$  is non-deterministic by having transitions  $(q, r_d(g), q')$  and  $(q, r_d(g), q'')$ . To resolve this non-determinism, we define  $\mathcal{D}'$  and  $\mathcal{C}'$  by modifying  $\mathcal{D}$  and  $\mathcal{C}$  as follows: we add new states  $q_1, q_2, q_3, q_4$  to  $\mathcal{D}$  and replace the two transitions  $(q, r_d(g), q')$  and  $(q, r_d(g), q'')$  by the transitions  $(q, r_d(g), q_1)$ ,  $(q_1, w_d(\mathbf{nd}), q_2)$ ,  $(q_2, r_d(0), q_3)$ ,  $(q_3, w_d(g), q')$ ,  $(q_2, r_d(1), q_4)$  and  $(q_4, w_d(g), q'')$ . Let  $q_0$  be the initial state of  $\mathcal{C}$ . We add two new states  $\hat{q}$  and  $\tilde{q}$  to  $\mathcal{C}$  and the transitions  $(q_0, r_c(\mathbf{nd}), \hat{q})(\hat{q}, w_c(0), \tilde{q})(\tilde{q}, w_c(1), q_0)$ . Finally, we extend the store to accommodate the new values  $\{0, 1, \mathbf{nd}\}$ . It follows that  $\mathcal{D}'$  has one fewer pair of non-deterministic transitions than  $\mathcal{D}$ . Similar transformations can eliminate other non-deterministic transitions (e.g., two writes from a state) or non-determinism in  $\mathcal{C}$ .

#### 4.1 Complexity of Safety Verification for FSMs and PDMs

We characterize the complexity of the safety verification problem of non-atomic networks depending on the nature of the leader and the contributors. We show:

Safety(dFSM, dFSM), Safety(PDM, FSM)	coNP-complete
Safety(dPDM, dPDM), Safety(PDM, PDM)	PSPACE-complete

**Theorem 1.** *Safety(dFSM, dFSM) is coNP-hard.*

We show hardness by a reduction from 3SAT to the complement of the safety verification problem. Given a 3SAT formula, we design a non-atomic network in which the leader and contributors first execute a protocol that determines an assignment to all variables, and uses subsets of contributors to store this assignment. For a variable  $x$ , the leader writes  $x$  to the store, inviting proposals for values. On reading  $x$ , contributors non-deterministically write either “ $x$  is 0” or “ $x$  is 1” on the store, possibly over-writing each other. At a future point, the leader reads the store, reading the proposal that was last written, say “ $x$  is 0.” The leader then writes “commit  $x$  is 0” on the store. Every contributor that reads this commitment moves to a state where it returns 0 every time the value of  $x$  is asked for. Contributors that do not read this message are stuck and do

not participate further. The commitment to 1 is similar. This protocol ensures that each variable gets assigned a consistent value.

Then, the leader checks that each clause is satisfied by querying the contributors for the values of variables (recall that contributors return consistent values) and checking each clause locally. If all clauses are satisfied, the leader writes a special symbol  $\#$ . The safety verification problem checks that  $\#$  is never written, which happens iff the formula is unsatisfiable. Finally, Lemma 10 ensures all processes are deterministic.

**Theorem 2.** *Safety(PDM, FSM) is in coNP.*

*Proof.* Fix a  $(\mathcal{D}, C)$ -network  $\mathcal{N}$ , where  $P_D$  is a PDM generating  $\mathcal{D} = \llbracket P_D \rrbracket$ , and  $C$  is a FSM. Hence  $L(\mathcal{D})$  is a context-free language and  $L(C)$  is regular. Prop. 1 and Def. 5 (of  $\mathcal{N}^S$ ) show that the  $(\mathcal{D}, C)$ -network  $\mathcal{N}$  is accepting iff  $L(\mathcal{D} \parallel S^E \parallel \bigsqcup_{g \in \mathcal{G}} S_g) \cap P_{\gamma_{\#}} \neq \emptyset$ . Since  $C$  is given by a FSM, so is  $C^E$ . Further,  $L_g = L(C^E) \cap \mathcal{G}(r_d, u_c)^* f_c(g)$  has a support captured by those paths in  $C^E$  starting from the initial state and whose label ends by  $f_c(g)$  and in which no state is entered more than once. Therefore if  $C^E$  has  $k$  states then the set of paths starting from the initial state, of length at most  $k + 1$  and whose label ends with  $f_c(g)$  is a support, call it  $\underline{L}_g$ , of  $L_g$ . Next, Lem. 7 shows that deciding  $L(\mathcal{D} \parallel S^E \parallel \bigsqcup_{g \in \mathcal{G}} S_g) \cap P_{\gamma_{\#}} \neq \emptyset$  is equivalent to  $L(\mathcal{D} \parallel S^E \parallel \bigsqcup_{g \in \mathcal{G}} \text{Pref}(\underline{L}_g \cdot w_c(g)^*)) \cap P_{\gamma_{\#}} \neq \emptyset$ .

Note that this last check does not directly provide a NP algorithm for non-safety because, due to the write records,  $S^E$  is exponentially larger than  $|\mathcal{G}|$ . So, we proceed by pushing down sequences of first writes and obtain the following equivalent statement:  $L(\mathcal{D}) \parallel (L(S^E) \cap P_{\gamma_{\#}}) \parallel (\bigsqcup_{g \in \mathcal{G}} L(\text{Pref}(\underline{L}_g \cdot w_c(g)^*)) \cap P_{\gamma_{\#}}) \neq \emptyset$ .

Now, we get an NP algorithm as follows: (a) guess  $\tau \in \gamma_{\#}$  (this can be done in time polynomial in  $|\mathcal{G}|$ ); (b) construct in polynomial time a FSA  $A_1$  for  $L(S^E) \cap P_{\tau}$  ( $A_1$  results from  $S^E$  by keeping the  $|\tau|$  write records corresponding to  $\tau$ ); (c) for each  $g \in \tau$ , guess  $z_g \in \underline{L}_g$  (the guess can be done in polynomial time); (d) guess  $z \in (\bigsqcup_{g \in \mathcal{G}} z_g) \cap P_{\tau}$  (this fixes a sequence of reads, useless writes and first writes of the contributors according to  $\tau$ ); (e) construct in polynomial time a FSA  $A_2$  such that  $L(A_2)$  is the least language containing  $z$  and if  $\alpha f_c(g) \beta_1 \beta_2 \in L(A_2)$  then  $\alpha f_c(g) \beta_1 w_c(g) \beta_2 \in L(A_2)$  (intuitively we add self-loops with write actions of  $\mathcal{G}(w_c)$  to the FSA accepting  $z$  such that  $w_c(g)$  occurs provided  $f_c(g)$  has previously occurred); (f) construct in time polynomial in  $|P_D|$  a context-free grammar (CFG)  $G_D$  such that  $L(G_D) = L(P_D)$ ; (g) construct in polynomial time a CFG  $G$  such that  $L(G) = L(G_D) \parallel L(A_1) \parallel L(A_2)$  (this can be done in time polynomial in  $|G_D| + |A_1| + |A_2|$  as shown in the companion technical report [13]); (h) check in polynomial time whether  $L(G) \neq \emptyset$ .  $\square$

The complexity of the problem becomes higher when all the processes are PDMs.

**Theorem 3.** *Safety(dPDM, dPDM) is PSPACE-hard.*

PSPACE-hardness is shown by reduction from the acceptance problem of a polynomial-space deterministic Turing machine. The proof is technical. The leader and contributors simulate steps of the Turing machine in rounds. The stack is used to store configurations of the Turing machine. In each round, the leader sends the current configuration of the Turing machine to contributors by writing the configuration one element at a time on to the store and waiting for an acknowledgement from some contributor that the

element was received. The contributors receive the current configuration and store the next configuration on their stacks. In the second part of the round, the contributors send back the configuration to the leader. The leader and contributors use their finite state to make sure all elements of the configuration are sent and received.

Additionally, the leader and the contributors use the stack to count to  $2^n$  steps. If both the leader and some contributor count to  $2^n$  in a computation, the construction ensures that the Turing machine has been correctly simulated for  $2^n$  steps, and the simulation is successful. The counting eliminates bad computation sequences in which contributors conflate the configurations from different steps due to asynchronous reads and writes.

Next we sketch the upper PSPACE bound that uses constructions on approximations of context-free languages. Those are detailed in technical report [13].

**Theorem 4.** *Safety(PDM, PDM) is in PSPACE.*

*Proof.* Let  $P_D$  and  $P_C$  be PDMs respectively generating  $\mathcal{D} = \llbracket P_D \rrbracket$  and  $C = \llbracket P_C \rrbracket$ , hence  $L(\mathcal{D})$  and  $L(C)$  are context-free languages. Proposition 1 shows that the  $(\mathcal{D}, C)$ -network  $\mathcal{N}$  is accepting iff  $L(\mathcal{N}^S) \cap P_{\gamma_\#} \neq \emptyset$  iff  $L(\mathcal{DS} \parallel S^E \parallel \sqcup_{g \in \mathcal{G}} S_g) \cap P_{\gamma_\#} \neq \emptyset$  (by (1)). From the construction of the Simulation Lemma, for each  $g \in \mathcal{G}$  the language  $L_g = L(C^E) \cap \mathcal{G}(r_d, u_c)^* f_c(g)$  is context-free, and so is  $L(S_g)$ . Given  $P_C$  we compute in polynomial time a PDA  $P_g$  such that  $L(P_g) = L_g$ . Next,

$$\begin{aligned} & L(\mathcal{DS} \parallel S^E \parallel \sqcup_{g \in \mathcal{G}} S_g) \cap P_{\gamma_\#} \neq \emptyset \\ \text{iff } & (L(\mathcal{DS}) \cap P_{\gamma_\#}) \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} L(S_g) \neq \emptyset \\ \text{iff } & (L(\mathcal{DS}) \cap P_{\gamma_\#}) \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} \text{Pref}(L(P_g) \cdot w_c(g)^*) \neq \emptyset \end{aligned} \quad (2)$$

$$\text{iff } (\bigcup_{\tau \in \gamma_\#} \hat{L}_\tau) \parallel L(S^E) \parallel \sqcup_{g \in \mathcal{G}} \text{Pref}(L(P_g) \cdot w_c(g)^*) \neq \emptyset \quad (3)$$

(2) follows from definition of  $S_g$  and  $L_g = L(P_g)$ ; (3) follows from Lem. 6 and by letting  $\hat{L}_\tau$  and  $\underline{L}(P_g)$  be a cover and support of  $L(\mathcal{DS}) \cap P_\tau$  and  $L(P_g)$ , respectively.

Next, for all  $g \in \mathcal{G}$  we compute a FSA  $A_g$  such that  $L(A_g)$  is a support of  $L(P_g)$ . Our first language-theoretic construction shows that the FSA  $A_g$  can be computed in time exponential but space polynomial in  $|P_g|$ . Then, because  $L(S^E)$  is a regular language, we compute in space polynomial in  $|P_D| + |P_C|$  a FSA  $A_C$  such that  $L(A_C) = L(S^E) \parallel \sqcup_{g \in \mathcal{G}} \text{Pref}(L(A_g) \cdot w_c(g)^*)$ . Hence, by (3) and because of  $\gamma_\#$  (guessing and checking  $\tau \in \gamma_\#$  is done in time polynomial in  $|\mathcal{G}|$ ) we find that it suffices to prove  $\hat{L}_\tau \parallel L(A_C) \neq \emptyset$  is decidable in space polynomial in  $|P_D| + |P_C|$ .

To compute a cover  $\hat{L}_\tau$  of  $L(\mathcal{DS}) \cap P_\tau$ , we need results about the  $k$ -index approximations of a context-free language [4]. Given a CFG  $G$  in CNF and  $k \geq 1$ , we define the  $k$ -index approximation of  $L(G)$ , denoted by  $L^{(k)}(G)$ , consisting of the words of  $L(G)$  having a derivation in which every intermediate word contains at most  $k$  occurrences of variables. We further introduce an operator  $\bowtie$  which, given  $G$  and FSA  $A$ , computes in polynomial time a context-free grammar  $G \bowtie A$  such that  $L(G \bowtie A) = L(G) \parallel L(A)$ . We prove the following properties:

1.  $L^{(3m)}(G)$  is a cover of  $L(G)$ , where  $m$  is the number of variables of  $G$ ;
2. for every FSA  $A$  and  $k \geq 1$ ,  $L^{(k)}(G \bowtie A) = L^{(k)}(G) \parallel L(A)$ ;
3.  $L^{(k)}(G) \neq \emptyset$  on input  $G$ ,  $k$  can be decided in  $\text{NSPACE}(k \log(|G|))$ .

Equipped with these results, the proof proceeds as follows. Let  $G_D$  be a context-free grammar such that  $L(G_D) = L(P_D)$ . It is well-known that  $G_D$  can be computed in time polynomial in  $|P_D|$ . Next, given  $\tau$ , we compute a grammar  $G_D^\tau$  recognizing  $P_\tau \cap L(\mathcal{DS})$  as follows. The definition of  $\mathcal{DS}$  shows that  $P_\tau \cap L(\mathcal{DS}) = L(\mathcal{D}) \parallel (L(S_{\mathcal{D}}^E) \cap P_\tau)$ . We then compute a FSA  $S_{\mathcal{D}}^\tau$  such that  $L(S_{\mathcal{D}}^\tau) = L(S_{\mathcal{D}}^E) \cap P_\tau$ . It can be done in time polynomial in  $|P_D| + |P_C|$  because it is a restriction of  $S_{\mathcal{D}}^E$  where write records are totally ordered according to  $\tau$  and there are exactly  $|\tau|$  of them. Therefore we obtain,  $P_\tau \cap L(\mathcal{DS}) = L(G_D) \parallel L(S_{\mathcal{D}}^\tau)$  because  $L(G_D) = L(\mathcal{D})$ . Define  $G_D^\tau$  as the CFG  $G_D \bowtie S_{\mathcal{D}}^\tau$  which can be computed in polynomial time in  $G_D$  and  $S_{\mathcal{D}}^\tau$ , hence in  $|P_D| + |P_C|$ . Clearly  $L(G_D^\tau) = P_\tau \cap L(\mathcal{DS})$ . Further,  $L^{(k)}(G_D^\tau)$  is a cover of  $L(G_D^\tau)$  for some  $k \leq p(|P_D|)$ , where  $p$  is a suitable polynomial.

By item 2,  $L^{(k)}(G_D^\tau) \parallel L(A_C) = L^{(k)}(G_D^\tau \bowtie A_C)$ , where the grammar  $G_D^\tau \bowtie A_C$  can be constructed in exponential time and space polynomial in  $|P_D| + |P_C|$ . Now we apply a generic result of complexity (see e.g. Lemma 4.17, [3]), slightly adapted: given functions  $f_1, f_2: \Sigma^* \rightarrow \Sigma^*$  and  $g: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$  if  $f_i$  can be computed by a  $s_{f_i}$ -space-bounded Turing machine, and  $g$  can be computed by a  $s_{g_1}(|x_1|) \cdot s_{g_2}(|x_2|)$ -space-bounded Turing machine, then  $g(f_1(x), f_2(x))$  can be computed in  $\log(|f_1(x)| + |f_2(x)|) + s_{f_1}(|x|) + s_{f_2}(|x|) + s_{g_1}(|f_1(x)|) \cdot s_{g_2}(|f_2(x)|)$  space. We have

- $f_1$  is the function that computes  $G_D^\tau \bowtie A_C$  on input  $(P_D, P_C)$ , and  $f_2$  is the function that on input  $P_D$  computes  $3m$ , where  $m$  is the number of variables of  $G_D^\tau$ . So the output size of  $f_1$  is exponential in the input size, while it is polynomial for  $f_2$ . Moreover,  $s_{f_i}$  for  $i = 1, 2$  is polynomial.
- $g$  is the function that on input  $(G_D^\tau \bowtie A_C, 3m)$  yields 1 if  $L^{(3m)}(G_D^\tau \bowtie A_C) \neq \emptyset$ , and 0 otherwise, where  $m$  is the number of variables of  $G_D^\tau$ . By (3)  $s_{g_1}$  is logarithmic, and  $s_{g_2}$  is linear.

Finally, the generic complexity result shows that  $g \circ f$  can be computed in space polynomial in  $|P_D| + |P_C|$ , and we are done.  $\square$

We note that our three language-theoretic constructions (the construction of automaton  $A_g$  that is a cover of  $L(P_g)$  of size at most exponential in  $|P_g|$ , and results 1, 2, and 3 in the proof above) improve upon previous constructions, and are all required for the optimal upper bound. Hague [17] shows an alternate doubly exponential construction using a result of Ehrenfeucht and Rozenberg. This gave a 2EXPTIME algorithm. Even after using our exponential time construction for  $A_g$ , we can only get an EXPTIME algorithm, since the non-emptiness problem for (general) context-free languages is P-complete [18]. Our bounded-index approximation for the cover and the space-efficient emptiness algorithm for bounded-index languages are crucial to the PSPACE upper bound.

## 4.2 The Bounded Safety Problem

Given  $k > 0$ , we say that a  $(\mathcal{D}, C)$ -network is  $k$ -safe if all traces in which the leader and each contributor make at most  $k$  steps are safe; i.e., we put a bound of  $k$  steps on the runtime of each contributor, and consider only safety within this bound. Here, a step consists of a read or a write of the shared register. The bound does not limit the total length of traces, because the number of contributors is unbounded. The *bounded safety* problem asks, given  $\mathcal{D}$ ,  $C$ , and  $k$  written in unary, if the  $(\mathcal{D}, C)$ -network is  $k$ -safe.

Given a class of (D,C)-networks, we define  $\text{BoundedSafety}(\mathcal{D}, \mathcal{C})$  as the restriction of the  $k$ -safety problem to pairs of machines  $M_D \in \mathcal{D}$  and  $M_C \in \mathcal{C}$ , where we write  $k$  in unary. A closer look to Theorem 1 shows that its proof reduces the satisfiability problem for a formula  $\phi$  to the bounded safety problem for a (D,C)-network and a number  $k$ , all of which have polynomial size  $|\phi|$ . This proves that  $\text{BoundedSafety}(\text{dFSM}, \text{dFSM})$  is coNP-hard. We show that, surprisingly, bounded safety remains coNP-complete for pushdown systems, and, even further, for arbitrary Turing machines. Notice that the problem is already coNP-complete for one single Turing machine.

We sketch the definition of the Turing machine model (TM), which differs slightly from the usual one. Our Turing machines have two kind of transitions: the usual transitions that read and modify the contents of the work tape, and additional transitions with labels in  $\mathcal{G}(r, w)$  for communication with the store. The machines are input-free, i.e., the input tape is always initially empty.

**Theorem 5.**  $\text{BoundedSafety}(\text{TM}, \text{TM})$  is coNP-complete.

*Proof.* Co-NP-hardness follows from Theorem 1. To prove  $\text{BoundedSafety}(\text{TM}, \text{TM})$  is in NP we use the simulation lemma. Let  $M_D, M_C, k$  be an instance of the problem, where  $M_D, M_C$  are Turing machines of sizes  $n_D, n_C$  with LTSs  $\mathcal{D} = \llbracket M_D \rrbracket$  and  $\mathcal{C} = \llbracket M_C \rrbracket$ , and let  $n_D + n_C = n$ . In particular, we can assume  $|\mathcal{G}| \leq n$ , because we only need to consider actions that appear in  $M_D$  and  $M_C$ . If the  $(\mathcal{D}, \mathcal{C})$ -network is not  $k$ -safe, then by definition there exist  $u \in L(\mathcal{D})$  and a multiset  $M = \{v_1, \dots, v_k\}$  over  $L(\mathcal{C}^E)$  such that  $u$  is compatible with  $M$  following some  $\tau \in \mathcal{Y}_\#$ ; moreover, all of  $u, v_1, \dots, v_m$  have length at most  $k$ . By Cor. 1 and Lem. 1 (showing we can drop traces without a first or regular write), there exists a set  $S = \{s_{g_1}, \dots, s_{g_m}\}$  with  $m \leq |\mathcal{G}| \leq n$ , where  $s_{g_i} \in L_{g_i} \cdot w_c(g_i)^*$ , and numbers  $i_1, \dots, i_m$  such that  $u$  is compatible with  $\{s_{g_1} w_c(g_1)^{i_1}, \dots, s_{g_m} w_c(g_m)^{i_m}\}$  following  $\tau$ . Since each of the  $s_{g_i}$  is obtained by suitably renaming the actions of a trace, we have  $|s_{g_i}| \leq k$ . Moreover, since the  $w_c(g_j)^{i_j}$  parts provide the writes necessary to execute the reads of the  $s_{g_i}$  sequences, and there are at most  $k \cdot (m + 1) \leq k \cdot (n + 1)$  of them, the numbers can be chosen so that  $i_1, \dots, i_m \leq O(n \cdot k)$  holds.

We present a nondeterministic polynomial algorithm that decides if the  $(\mathcal{D}, \mathcal{C})$ -network is  $k$ -unsafe. The algorithm guesses  $\tau \in \mathcal{Y}_\#$  and traces  $u, s_{g_1}, \dots, s_{g_m}$  of length at most  $k$ . Since there are at most  $n + 1$  of those traces, this can be done in polynomial time. Then, the algorithm guesses numbers  $i_1, \dots, i_m$ . Since the numbers can be chosen so that  $i_1, \dots, i_m \leq O(n \cdot k)$ , this can also be done in polynomial time. Finally, the algorithm guesses an interleaving of  $u, s_{g_1} w_c(g_1)^{i_1}, \dots, s_{g_m} w_c(g_m)^{i_m}$  and checks compatibility following  $\tau$ . This can be done in  $O(n^2 \cdot k)$  time. If the algorithm succeeds, then there is a witness that  $(L(\mathcal{D}) \parallel L(S^E) \parallel \bigsqcup_{g \in \mathcal{G}} L(S_g)) \cap P_\tau \neq \emptyset$  holds, which shows, by Prop. 1 and Def. 5 (of  $\mathcal{N}^S$ ) that the  $(\mathcal{D}, \mathcal{C})$ -network is unsafe.  $\square$

A TM is poly-time if it takes at most  $p(n)$  steps for some polynomial  $p$ , where  $n$  is the size of (the description of) the machine in some encoding. As a corollary, we get that the safety verification problem when leaders and contributors are poly-time Turing machines is coNP-complete. Note that the coNP upper bound holds even though the LTS corresponding to a poly-time TM is exponentially larger than its encoding.

## References

1. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems. In: LICS 1996, pp. 313–321. IEEE Computer Society (1996)
2. Apt, K.R., Kozen, D.C.: Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters* 22(6), 307–309 (1986)
3. Arora, S., Barak, B.: *Computational Complexity—A Modern Approach*. CUP (2009)
4. Brainerd, B.: An analog of a theorem about context-free languages. *Information and Control* 11(56), 561–567 (1967)
5. Clarke, E.M., Talupur, M., Veith, H.: Proving Ptolemy right: The environment abstraction framework for model checking concurrent systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 33–47. Springer, Heidelberg (2008)
6. Delzanno, G.: Constraint-based verification of parameterized cache coherence protocols. *Formal Methods in System Design* 23(3), 257–301 (2003)
7. Delzanno, G., Raskin, J.-F., Van Begin, L.: Towards the automated verification of multi-threaded java programs. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 173–187. Springer, Heidelberg (2002)
8. Delzanno, G., Sangnier, A., Zavattaro, G.: Parameterized verification of ad hoc networks. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 313–327. Springer, Heidelberg (2010)
9. Dimitrova, R., Podelski, A.: Is lazy abstraction a decision procedure for broadcast protocols? In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 98–111. Springer, Heidelberg (2008)
10. Emerson, E.A., Kahlon, V.: Exact and efficient verification of parameterized cache coherence protocols. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 247–262. Springer, Heidelberg (2003)
11. Emerson, E.A., Namjoshi, K.S.: Verification of parameterized bus arbitration protocol. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 452–463. Springer, Heidelberg (1998)
12. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: LICS 1999, pp. 352–359. IEEE Computer Society (1999)
13. Esparza, J., Ganty, P., Majumdar, R.: Parameterized verification of asynchronous shared-memory systems. *CoRR abs/1304.1185* (2013)
14. Finkel, A., Leroux, J.: How to compose Presburger-accelerations: Applications to broadcast protocols. In: Agrawal, M., Seth, A.K. (eds.) FSTTCS 2002. LNCS, vol. 2556, pp. 145–156. Springer, Heidelberg (2002)
15. Ganty, P., Majumdar, R.: Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.* 6, 1–6 (2012)
16. Guerraoui, R., Ruppert, E.: Anonymous and fault-tolerant shared-memory computing. *Distributed Computing* 20(3), 165–177 (2007)
17. Hague, M.: Parameterised pushdown systems with non-atomic writes. In: Proc. of FSTTCS 2011. LIPIcs, vol. 13, pp. 457–468. Schloss Dagstuhl (2011)
18. Jones, N.D., Laaser, W.T.: Complete problems for deterministic polynomial time. In: Proc. of STOC 1974, pp. 40–46. ACM (1974)
19. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 645–659. Springer, Heidelberg (2010)
20. Laurendeau, C., Barbeau, M.: Secure anonymous broadcasting in vehicular networks. In: LCN 2007, pp. 661–668. IEEE Computer Society (2007)



21. McMillan, K.L.: Verification of an implementation of tomasulo's algorithm by compositional model checking. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 110–121. Springer, Heidelberg (1998)
22. La Torre, S., Madhusudan, P., Parlato, G.: Model-checking parameterized concurrent programs using linear interfaces. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 629–644. Springer, Heidelberg (2010)
23. Viswanathan, M., Chadha, R.: Deciding branching time properties for asynchronous programs. *Theoretical Computer Science* 410(42), 4169–4179 (2009)