A Generic Framework for Reasoning about Dynamic Networks of Infinite-State Processes

Ahmed Bouajjani, Yan Jurski, Mihaela Sighireanu

A Generic Framework for Reasoning about Dynamic Networks of Infinite-State Processes

Ahmed Bouajjani, Yan Jurski, and Mihaela Sighireanu

LIAFA, University of Paris 7, Case 7014, 2 place Jussieu, 75251 Paris 05, France.
{abou,jurski,sighirea}@liafa.jussieu.fr

Abstract. We propose a framework for reasoning about unbounded dynamic networks of infinite-state processes. We propose Constrained Petri Nets (CPN) as generic models for these networks. They can be seen as Petri nets where tokens (representing occurrences of processes) are colored by values over some potentially infinite data domain such as integers, reals, etc. Furthermore, we define a logic, called CML (colored markings logic), for the description of CPN configurations. CML is a first-order logic over tokens allowing to reason about their locations and their colors. Both CPNs and CML are parametrized by a color logic allowing to express constraints on the colors (data) associated with tokens.
We investigate the decidability of the satisfiability problem of CML and its applications in the verification of CPNs. We identify a fragment of CML for which the satisfiability problem is decidable (whenever it is the case for the underlying color logic), and which is closed under the computations of post and pre images for CPNs. These results can be used for several kinds of analysis such as invariance checking, pre-post condition reasoning, and bounded reachability analysis.

1 Introduction

The verification of software systems requires in general the consideration of infinite-state models. The sources of infinity in software models are multiple. One of them is the manipulation of variables and data structures ranging over infinite domains (such as integers, reals, arrays, etc). Another source of infinity is the fact that the number of processes running in parallel in the system can be either a parameter (fixed but arbitrarily large), or it can be dynamically changing due to process creation. While the verification of parameterized systems requires reasoning uniformly about the infinite family of (static) networks corresponding to any possible number of processes, the verification of dynamic systems requires reasoning about the infinite number of all possible dynamically changing network configurations.

There are many works and several approaches on the verification of infinite-state systems taking into account either the aspects related to infinite data domains, or the aspects related to unbounded network structures due to parameterization or dynamism. Concerning systems with data manipulation, a lot of work has been devoted to the verification of, for instance, finite-structure systems with unbounded counters, clocks, stacks, queues, etc. (see, e.g., [1, 11, 30, 7, 5, 27, 26]). On the other hand, a lot of work has been done for the verification of parameterized and dynamic networks of boolean (or finite-data domain) processes, proposing either exact model-checking and reachability

analysis techniques for specific classes of systems (such as broadcast protocols, multithreaded programs, etc) [24, 25, 22, 16, 15], or generic algorithmic techniques (which can be approximate, or not guaranteed to terminate) such as network invariants-based approaches [31, 20], and (abstract) regular model checking [13, 17, 3, 12]. However, only few works consider both infinite data manipulation and parametric/dynamic network structures (see the paragraph on related work).

In this paper, we propose a generic framework for reasoning about parameterized and dynamic networks of concurrent processes which can manipulate (local and global) variables over infinite data domains. Our framework is parameterized by a data domain and a first-order theory on it (e.g., Presburger arithmetics on natural numbers). It consists of (1) expressive models allowing to cover a wide class of systems, and (2) a logic allowing to specify and to reason about the configurations of these models.

The models we propose are called Constrained Petri Nets (CPN for short). They are based on (place/transition) Petri nets where tokens are colored by data values. Intuitively, tokens represent different occurrences of processes, and places are associated with control locations and contain tokens corresponding to processes which are at a same control location. Since processes can manipulate local variables, each token (process occurrence) has several colors corresponding to the values of these variables. Then, configurations of our models are markings where each place contains a set of colored tokens, and transitions modify the markings as usual by removing tokens from some places and creating new ones in some other places. Transitions are guarded by constraints on the colors of tokens before and after firing the transition. We show that CPNs allow to model various aspects such as unbounded dynamic creation of processes, manipulation of local and global variables over unbounded domains such as integers, synchronization, communication through shared variables, locks, etc.

The logic we propose for specifying configurations of CPN's is called Colored Markings Logic (CML for short). It is a first order logic over tokens and their colors. It allows to reason about the presence of tokens in places, and also about the relations between the colors of these tokens. The logic CML is parametrized by a first order logic over the color domain allowing to express constraints on tokens.

We investigate the decidability of the satisfiability problem of CML and its applications in verification of CPNs. While the logic is decidable for finite color domains (such as booleans), we show that, unfortunately, the satisfiability problem of this logic becomes undecidable as soon as we consider as a color domain the set of natural numbers with the usual ordering relation (and without any arithmetical operations). We prove that this undecidability result holds already for the fragment $\forall^*\exists^*$ of the logic (in the alternation hierarchy of the quantifiers over token variables) with this color domain.

On the other hand, we prove that the satisfiability problem is decidable for the fragment $\exists^*\forall^*$ of CML whenever the underlying color logic has a decidable satisfiability problem, e.g., Presburger arithmetics, the first-order logic of addition and multiplication over reals, etc. Moreover, we prove that the fragment $\exists^*\forall^*$ of CML is effectively closed under post and pre image computations (i.e., computation of immediate successors and immediate predecessors) for CPN's where all transition guards are also in $\exists^*\forall^*$. We show also that the same closure results hold when we consider the fragment $\exists^*$ instead of $\exists^*\forall^*$.

These generic decidability and closure results can be applied in the verification of CPN models following different approaches such as pre-post condition (Hoare triples based) reasoning, bounded reachability analysis, and inductive invariant checking. More precisely, we derive from our results mentioned above that (1) checking whether starting from a $\exists^*\forall^*$ pre-condition, a $\forall^*\exists^*$ condition holds after the execution of a transition is decidable, that (2) the bounded reachability problem between two $\exists^*\forall^*$ definable sets is decidable, and that (3) checking whether a formula defines an inductive invariant is decidable for boolean combinations of $\exists^*$ formulas.

These results can be used to deal with non trivial examples of systems. Indeed, in many cases, program invariants and the assertions needed to establish them fall in the considered fragments of our logic. We illustrate this by carrying out in our framework the parametric verification of a Reader-Writer lock with an arbitrarily large number of processes. This case study was introduced in [28] where the authors provide a correctness proof for the case of one reader and one writer.

Proofs as well as the exposition of the Reader-Writer case study are provided in appendix.

Related work: The use of unbounded Petri nets as models for parametrized networks of processes has been proposed in many existing works such as [29, 24, 22]. However, these works consider networks of *finite-state* processes and do not address the issue of manipulating infinite data domains. The extension of this idea to networks of infinite-state processes has been addressed only in very few works [4, 21, 18, 2]. In [4], Abdulla and Jonsson consider the case of networks of 1-clock timed systems and show, using the theory of well-structured systems and well quasi orderings [1, 27], that the verification problem for a class of safety properties is decidable. Their approach has been extended in [21, 18] to a particular class of multiset rewrite systems with constraints (see also [2] for recent developments of this approach). Our modeling framework is actually inspired by these works. However, while they address the issue of deciding the verification problem of safety properties (by reduction to the coverability problem) for specific classes of systems, we consider in our work a general framework, allowing to deal in a generic way with various classes of systems, where the user can express assertions about the configurations of the system, and check automatically that they hold (using post-pre reasoning and inductive invariant checking) or that they do not hold (using bounded reachability analysis). Our framework allows to reason automatically about systems which are beyond the scoop of the techniques proposed in [4, 21, 18, 2] (such as, for instance, the parametrized Reader-Writer lock system of Appendix E).

In a series of papers, Pnueli et al. developed an approach for the verification of parameterized systems combining abstraction and proof techniques (see, e.g., [6]). This is probably one of the most advanced existing approaches allowing to deal with unbounded networks of infinite-state processes. We propose here a different framework for reasoning about these systems. In [6], the authors consider a logic on (parametric-bound) *arrays* of integers, and they identify a fragment of this logic for which the satisfiability problem is decidable. In this fragment, they restrict the shape of the formula (quantification over indices) to formulas in the fragment $\exists^*\forall^*$ similarly to what we do, and also the class of used arithmetical constraints on indices and on the associated values. In a recent work by Bradley and al. [19], the satisfiability problem of the logic of

unbounded arrays with integers is investigated and the authors provide a new decidable fragment, which is incomparable to the one defined in [6], but again which imposes similar restrictions on the quantification alternation in the formulas, and on the kind of constraints that can be used. In contrast with these works, we consider a logic on *multisets* of elements with *any* kind of associated data values, provided that the used theory on the data domain is decidable. For instance, we can use in our logic general Presburger constraints whereas [6] and [19] allow limited classes of constraints. On the other hand, we cannot specify faithfully unbounded arrays in our decidable fragment because formulas of the form $\forall\exists$ are needed to express that every non extremal element has a successor/predecessor. Nevertheless, for the verification of safety properties and invariant checking, expressing this fact is not necessary, and therefore, it is possible to handle in our framework all usual examples of parametrized systems (such as mutual exclusion protocols) considered in the works mentioned above.

Let us finally mention that there are recent works on logics (first-order logics, or temporal logics) over finite/infinite structures (words or trees) over infinite alphabets (which can be considered as abstract infinite data domains) [9, 8, 23]. The obtained positive results so far concern logics with very limited data domain (basically infinite sets with only equality, or sometimes with an ordering relation), and are based on reduction to complex problems such as reachability in Petri nets.

2 Colored Markings Logic

2.1 Preliminaries

Consider an enumerable set of *tokens* and let us identify this set with the set of natural numbers $\mathbb{N}$. Intuitively, tokens represent occurrences of (parallel) processes. We assume that tokens may have colors corresponding for instance to data values attached to the corresponding processes. Let $\mathbb{C}$ be a (potentially infinite) *token color domain*. Examples of color domains are the set of natural numbers $\mathbb{N}$ and the set of real numbers $\mathbb{R}$.

Colors are associated with tokens through *coloring functions*. Let Γ be a finite set of *token coloring symbols*. Each element in Γ is interpreted as a mapping from $\mathbb{N}$ (the set of tokens) to $\mathbb{C}$ (the set of colors). Then, let a valuation of the token coloring symbols be a mapping in $[\Gamma \to (\mathbb{N} \to \mathbb{C})]$.

To express constraints on token colors, we use first-order logics over the considered color domains. In the sequel we refer to such logics as *color logics*. Presburger arithmetics $\mathsf{PA} = (\mathbb{N}, \{0, 1, +\}, \{\leq\})$ is an example of such a logic. It is well known that the satisfiability problem of Presburger arithmetics is decidable. An interesting sublogic of PA is the *difference logic* $\mathsf{DL} = (\mathbb{N}, \{0\}, \{\leq_k : k \geq 0\})$ where, for every $u, v, k \in \mathbb{N}$, $u \leq_k v$ holds if and only if $u - v \leq k$. The *order logic* on natural numbers is the sublogic of DL defined by $\mathsf{OL} = (\mathbb{N}, \{0\}, \leq)$. Another example of a decidable logic which can be used as a color logic is the first-order theory of reals $\mathsf{FO}_{\mathbb{R}} = (\mathbb{R}, \{0, 1, +, \times\}, \{\leq\})$.

We consider that tokens can be located at *places*. Let $\mathbb{P}$ be a finite set of such places. A *marking* is a mapping in $[\mathbb{N} \to \mathbb{P} \cup \{\bot\}]$ which associates with each token the unique place where it is located if it is defined, or $\bot$ otherwise. A *colored marking* is a pair $\langle M, \mu \rangle$ where M is a marking and μ is a valuation of the token coloring symbols.

2.2 Syntax and semantics of CML

We define hereafter the syntax of the logic *colored markings logic* $\mathsf{CML}(L,\Gamma,\mathbb{P})$ which is parametrized with a color logic L, a finite set of token coloring symbols Γ, and a finite set of places $\mathbb{P}$. Then, let $L=(\mathbb{C},\Omega,\Xi)$ be the first-order logic over the color domain $\mathbb{C}$ of the set of functions Ω and the set of relations Ξ. In the sequel, we omit all or some of the parameters of CML when their specification is not necessary.

Let T be set of *token variables* and let C be set of *color variables*, and assume that $T\cap C=\emptyset$. The set of CML terms (called *token color terms*) is given by the grammar:

$$t ::= z \mid \gamma(x) \mid \omega(t_1,\ldots,t_n)$$

where $z\in C$, $\gamma\in\Gamma$, $x\in T$, and $\omega\in\Omega$. Then, the set of CML formulas is given by:

$$\varphi ::= x=y \mid p(x) \mid \xi(t_1,\ldots,t_m) \mid \neg\varphi \mid \varphi\vee\varphi \mid \exists z.\ \varphi \mid \exists x.\ \varphi$$

where $x,y\in T$, $z\in C$, $p\in\mathbb{P}\cup\{\bot\}$, $\xi\in\Xi$, and $t_1,\ldots,t_m$ are token color terms. Boolean connectives such as conjunction ($\wedge$) and implication ($\Rightarrow$), and universal quantification ($\forall$) can be defined in terms of $\neg$, $\vee$, and $\exists$. We also use $\exists x\in p.\ \varphi$ (resp. $\forall x\in p.\ \varphi$) as an abbreviation of the formula $\exists x.\ p(x)\wedge\varphi$ (resp. $\forall x.\ p(x)\Rightarrow\varphi$). Notice that the set of terms (resp. formulas) of L is included in the set of terms (resp. formulas) of $\mathsf{CML}(L)$.

The notions of free/bound occurrences of variables in formulas and the notions of closed/open formulas are defined as usual in first-order logics. In the sequel, we assume w.l.o.g. that in every formula, each variable is quantified at most once.

We define a satisfaction relation between colored markings and CML formulas. For that, we need first to define the semantics of CML terms. Given valuations $\theta\in[T\to\mathbb{N}]$, $\delta\in[C\to\mathbb{C}]$, and $\mu\in[\Gamma\to(\mathbb{N}\to\mathbb{C})]$, we define a mapping $\langle\langle\cdot\rangle\rangle_{\theta,\delta,\mu}$ which associates with each color term a value in $\mathbb{C}$:

$$\begin{aligned}\langle\langle z\rangle\rangle_{\theta,\delta,\mu} &= \delta(z)\\ \langle\langle\gamma(x)\rangle\rangle_{\theta,\delta,\mu} &= \mu(\gamma)(\theta(x))\\ \langle\langle\omega(t_1,\ldots,t_n)\rangle\rangle_{\theta,\delta,\mu} &= \omega(\langle\langle t_1\rangle\rangle_{\theta,\delta,\mu},\ldots,\langle\langle t_n\rangle\rangle_{\theta,\delta,\mu})\end{aligned}$$

Then, we define inductively the satisfaction relation $\models_{\theta,\delta}$ between colored markings $\langle M,\mu\rangle$ and CML formulas as follows:

$$\begin{aligned}\langle M,\mu\rangle\models_{\theta,\delta}\xi(t_1,\ldots,t_m) &\text{ iff } \xi(\langle\langle t_1\rangle\rangle_{\theta,\delta,\mu},\ldots,\langle\langle t_m\rangle\rangle_{\theta,\delta,\mu})\\ \langle M,\mu\rangle\models_{\theta,\delta}p(x) &\text{ iff } M(\theta(x))=p\\ \langle M,\mu\rangle\models_{\theta,\delta}x=y &\text{ iff } \theta(x)=\theta(y)\\ \langle M,\mu\rangle\models_{\theta,\delta}\neg\varphi &\text{ iff } \langle M,\mu\rangle\not\models_{\theta,\delta}\varphi\\ \langle M,\mu\rangle\models_{\theta,\delta}\varphi_1\vee\varphi_2 &\text{ iff } \langle M,\mu\rangle\models_{\theta,\delta}\varphi_1 \text{ or } \langle M,\mu\rangle\models_{\theta,\delta}\varphi_2\\ \langle M,\mu\rangle\models_{\theta,\delta}\exists x.\ \varphi &\text{ iff } \exists t\in\mathbb{T}.\ \langle M,\mu\rangle\models_{\theta[x\leftarrow t],\delta}\varphi\\ \langle M,\mu\rangle\models_{\theta,\delta}\exists z.\ \varphi &\text{ iff } \exists c\in\mathbb{C}.\ \langle M,\mu\rangle\models_{\theta,\delta[z\leftarrow c]}\varphi\end{aligned}$$

For every formula φ, we define $[\![\varphi]\!]_{\theta,\delta}$ the be the set of markings $\langle M,\mu\rangle$ such that $\langle M,\mu\rangle\models_{\theta,\delta}\varphi$. A formula φ is *satisfiable* iff there exist valuations θ and δ s.t. $[\![\varphi]\!]_{\theta,\delta}\neq\emptyset$.

2.3 Syntactical forms and fragments

Prenex normal form: A formula is in *prenex normal form* (PNF) if it is of the form

$$Q_1 y_1 Q_2 y_2 \dots Q_m y_m.\ \varphi$$

where (1) $Q_1, \dots, Q_m$ are (existential or universal) quantifiers, (2) $y_1, \dots, y_m$ are variables in $T \cup C$, and φ is a quantifier-free formula. It can be proved that for every formula φ in CML, there exists an equivalent formula φ' in prenex normal form.

Quantifier alternation hierarchy: We consider two families $\{\Sigma_n\}_{n\geq 0}$ and $\{\Pi_n\}_{n\geq 0}$ of fragments of CML defined according to the alternation depth of existential and universal quantifiers in their PNF:

- Let $\Sigma_0 = \Pi_0$ be the set of formulas in PNF where all quantified variables are in C,
- For $n \geq 0$, let Σ_{n+1} (resp. Π_{n+1}) be the set of formulas $Qy_1 \dots y_m.\ \varphi$ in PNF where $y_1, \dots, y_m \in T \cup C$, Q is the existential (resp. universal) quantifier $\exists$ (resp. $\forall$), and φ is a formula in Π_n (resp. Σ_n).

It is easy to see that, for every $n \geq 0$, Σ_n and Π_n are closed under conjunction and disjunction, and that the negation of a Σ_n formula is a Π_n formula and vice versa. For every $n \geq 0$, let $B(\Sigma_n)$ denote the set of all boolean combinations of Σ_n formulas. Clearly, $B(\Sigma_n)$ subsumes both Σ_n and Π_n, and is included in both Σ_{n+1} and Π_{n+1}.

Special form: The set of formulas in special form is given by the grammar:

$$\varphi ::= x = y \mid \xi(t_1, \dots, t_n) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists z.\ \varphi \mid \exists x \in p.\ \varphi$$

where $x, y \in T$, $z \in C$, $p \in \mathbb{P} \cup \{\bot\}$, $\xi \in \Xi$, and $t_1, \dots, t_n$ are token color terms. It is not difficult to see that for every closed formula φ in CML, there exists an equivalent formula φ' in special form. The transformation is based on the following fact: since variables are assumed to be quantified at most once in formulas, each formula $\exists x.\ \phi$ can be replaced by $\bigvee_{p \in \mathbb{P} \cup \{\bot\}} \exists x \in p.\ \phi_{x,p}$ where $\phi_{x,p}$ is obtained by substituting in ϕ each occurrence of $p(x)$ by *true*, and each occurrence of $q(x)$, with $p \neq q$, by *false*.

3 Satisfiability Problem

We investigate the decidability of the satisfiability problem of the logic CML(L), assuming that the underlying color logic L has a decidable satisfiability problem.

Let us mention that in the case of a finite color domain, for instance for the domain of booleans with equality and usual operations, the logic CML is decidable. The result is a direct consequence of the decidability of the class of relational monadic formulae in first-order logic, also known as the Löwenheim class with equality [10].

Then, let us consider the case of infinite data domains. First, we prove that as soon as we consider natural numbers with ordering, the satisfiability problem of CML is undecidable already for the fragment Π_2. The proof (see Appendix A) is by a reduction of

the halting problem of Turing machines. The idea is to encode a computation of a machine, seen as a sequence of tape configurations, using tokens with integer colors. Each token represents a cell in the tape of the machine at some computation step. Therefore the token has two integer colors, its position in the tape, and the position of its configuration in the computation (the corresponding computation step). The other informations such as the contents of the cell, the fact that a cell corresponds to the position of the head, and the control state, are encoded using a finite number of places. Then, using $\forall^*\exists^*$ formulas, it is possible to express that two consecutive configurations correspond indeed to a valid transition of the machine. Intuitively, this is possible because these formulas allow to relate each cell at some configuration to the corresponding cell at the next configuration.

Theorem 1. *The satisfiability problem of the fragment* Π_2 *of* $\mathsf{CML}(\mathsf{OL})$ *is undecidable.*

Nevertheless, we can prove the following generic decidability result for the fragment Σ_2 of our logic:

Theorem 2. *Let L be a colored tokens logic. If the satisfiability problem of L is decidable, then the fragment* Σ_2 *of* $\mathsf{CML}(L)$ *is decidable.*

The idea of the proof (see Appendix B) is to reduce the satisfiability problem of Σ_2 to the satisfiability problem of Σ_0 formulas (which are formulas in the color logic L). We proceed as follows: we prove first that the fragment Σ_2 has the small model property, i.e., every satisfiable formula φ in Σ_2 has a model of a bounded size (where the size is the number of tokens in each place). This bound corresponds actually to the number of existentially quantified token variables in the formula. Notice that this fact does not lead directly to an enumerative decision procedure for the satisfiability problem since the number of models of a bounded size is infinite in general (due to infinite color domains). Then, we use the fact that over a finite model, universal quantifications in φ can be transformed into finite conjunctions, in order to build a formula $\widehat{\varphi}$ in Σ_1 which is satisfiable if and only if the original formula φ is satisfiable. Actually, $\widehat{\varphi}$ defines precisely the upward-closure of the set of markings defined by φ (w.r.t. the inclusion ordering between sets of colored markings, extended to vectors of places). Finally, it can be shown that the Σ_1 formula $\widehat{\varphi}$ is satisfiable if and only if the Σ_0 obtained by transforming existential quantification over token into existential quantification over colors is decidable.

4 Constrained Petri Nets

Let T be a set of token variables and C be a set of color variables such that $T \cap C \neq \emptyset$. A *Constrained Petri Net* (CPN) is a tuple $S = (\mathbb{P}, L, \Gamma, \Delta)$ where $\mathbb{P}$ is a finite set of places, $L = (\mathbb{C}, \Omega, \Xi)$ is a colored tokens logic, Γ is a finite set of token coloring symbols, and Δ is a finite set of *constrained transitions* of the form:

$$\overrightarrow{x} \in \overrightarrow{p} \hookrightarrow \overrightarrow{y} \in \overrightarrow{q} \ : \ \varphi(\overrightarrow{x}, \overrightarrow{y}) \tag{1}$$

where $\overrightarrow{x} = (x_1, \ldots, x_n) \in T^n$, $\overrightarrow{y} = (y_1, \ldots, y_m) \in T^m$, $\overrightarrow{p} = (p_1, \ldots, p_n) \in \mathbb{P}^n$, $\overrightarrow{q} = (q_1, \ldots, q_m) \in \mathbb{P}^m$, and $\varphi(\overrightarrow{x}, \overrightarrow{y})$ is a $\mathsf{CML}(L, \Gamma, \mathbb{P})$ formula called the *transition guard*.

Given a fragment Θ of CML, we denote by $\mathsf{CPN}[\Theta]$ the class of CPN where all transition guards are formulas in the fragment Θ. Due to the (un)decidability results of section 3, we focus in the sequel on the classes $\mathsf{CPN}[\Sigma_2]$ and $\mathsf{CPN}[\Sigma_1]$.

Configurations of CPN's are colored markings. Constrained transitions define transformation rules of these markings. Given a CPN S, we define a transition relation $\rightarrow_S$ between colored markings as follows: For every two colored markings $\langle M,\mu\rangle$ and $\langle M',\mu'\rangle$, we have $\langle M,\mu\rangle \rightarrow_S \langle M',\mu'\rangle$ iff there exists a constrained transition of the form (1), and there exist tokens $t_1,\ldots,t_n$ and $t'_1,\ldots,t'_m$ s.t. $\forall i,j \in \{1,\ldots,n\}.\ i \neq j \Rightarrow t_i \neq t_j$, and $\forall i,j \in \{1,\ldots,m\}.\ i \neq j \Rightarrow t'_i \neq t'_j$, and

1. $\forall i \in \{1,\ldots,n\}.\ M(t_i) = p_i$ and $M'(t_i) = \bot$,
2. $\forall i \in \{1,\ldots,m\}.\ M(t'_i) = \bot$ and $M'(t'_i) = q_i$,
3. $\forall t \in \mathbb{N}$, if $\forall i \in \{1,\ldots,n\}.\ t \neq t_i$ and $\forall j \in \{1,\ldots,m\}.\ t \neq t'_j$, then $M(t) = M'(t)$ and $\forall \gamma \in \Gamma.\ \mu(\gamma)(t) = \mu'(\gamma)(t)$,
4. $\langle M, \mu \cup \mu'\rangle \models_{\theta,\delta_\emptyset} \varphi(\overrightarrow{x},\overrightarrow{y})$, where $\theta \in [T \rightarrow \mathbb{N}]$ is a valuation of the token variables such that $\forall i \in \{1,\ldots,n\}.\ \theta(x_i) = t_i$ and $\forall j \in \{1,\ldots,m\}.\ \theta(y_j) = t'_j$, $\delta_\emptyset$ is the empty domain valuation of color variables, and $\mu \cup \mu'$ is such that: for every $\gamma \in \Gamma$, and every token $t \in \mathbb{T}$, if $t \in \{t_1,\ldots,t_n\}$ then $\mu \cup \mu'(\gamma)(t) = \mu(\gamma)(t)$, if $t \in \{t'_1,\ldots,t'_m\}$ then $\mu \cup \mu'(\gamma)(t) = \mu'(\gamma)(t)$, and $\mu \cup \mu'(\gamma)(t) = \mu(\gamma)(t) = \mu'(\gamma)(t)$ otherwise.

Intuitively, the definition above says that firing a transition means that n different tokens $t_1,\ldots,t_n$ are deleted from the places $p_1,\ldots,p_n$ (1), and m new different tokens $t'_1,\ldots,t'_m$ are added to the places $q_1,\ldots,q_m$ (2), provided that the colors of all these (old and new) tokens satisfy the formula φ, which may also involve constraints on other tokens in the whole marking M (4). Moreover, this operation does not modify the rest of the tokens (others than $t_1,\ldots,t_n$ and $t'_1,\ldots,t'_m$) in the marking (3).

Given a colored marking $\mathcal{M}$, let $\mathsf{post}_S(\mathcal{M}) = \{\mathcal{M}' : \mathcal{M} \rightarrow_S \mathcal{M}'\}$ be the set of its immediate successors, and let $\mathsf{pre}_S(\mathcal{M}) = \{\mathcal{M}' : \mathcal{M}' \rightarrow_S \mathcal{M}\}$ be the set of its immediate predecessors. These definitions can be generalized to sets of colored markings in the obvious way. Finally, for every set of colored markings $\mathbb{M}$, let $\widetilde{\mathsf{pre}}_S(\mathbb{M}) = \overline{\mathsf{pre}_S(\overline{\mathbb{M}})}$, where $\overline{\cdot}$ denotes complementation (w.r.t. the set of all colored markings).

5 Modeling power of CPN

We show in this section how CPN can be used to model (unbounded) dynamic networks of parallel processes. We assume w.l.o.g. that all processes are identically defined. We consider that a process is defined by a finite control state machine supplied with variables and data structures ranging over potentially infinite domains (such as integer variables, reals, etc). Processes running in parallel can communicate and synchronize using various kinds of mechanisms (rendez-vous, shared variables, locks, etc). Moreover, they can dynamically spawn new (copies of) processes in the network.

Dynamic networks of processes: Let $\mathcal{L}$ be the set of control locations of each of the processes. (Remember that this set is the same for all processes.) We associate with each process control location $\ell \in \mathcal{L}$ a place. Then, each running process is represented

by a token, and in every marking, a place contains precisely the tokens representing processes which are at the corresponding control location.

Assume for the moment that processes do not manipulate (infinite domain) data. Then, a basic action $\ell \longrightarrow \ell'$ of a process moving its control from a location ℓ to another location ℓ' is modeled by a transition: $x \in \ell \hookrightarrow y \in \ell' : true$. An action spawning a new process $\ell \xrightarrow{\mathsf{spawn}(\ell_0)} \ell'$ is modeled using a transition which creates a new token in the initial control location of the new process: $x \in \ell \hookrightarrow y_1 \in \ell', y_2 \in \ell_0 : true$.

Local variables: Consider now that each process has a vector of n local variables $\overrightarrow{v} = (v_1, \ldots, v_n)$ over some (potentially infinite) data domain. Then, we consider a set of coloring symbols $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ associating with each token n colors (in the considered data domain) corresponding to the values of the local variables: for each process, represented by a token t, for each local variable v_i, $\gamma_i(t)$ defines the value of v_i.

A process action $\ell \xrightarrow{\overrightarrow{v} := \overrightarrow{f}(\overrightarrow{v})} \ell'$ which (in addition of changing the control location from ℓ to ℓ') performs the assignment $\overrightarrow{v} := \overrightarrow{f}(\overrightarrow{v})$, where $\overrightarrow{f}$ is a vector of expressions over the considered data domain, is modeled by the transition

$$x \in \ell \hookrightarrow y \in \ell' : \bigwedge_{i=1}^{n} \gamma_i(y) = f_i(\gamma_1(x), \ldots, \gamma_n(x))$$

For that, we use a token color logic which allows to express the effects of the actions. For instance, in the case of processes with integer variables and linear assignments, Presburger arithmetics (PA) can be used as colored tokens logic.

Global variables: Assume that processes share global variables $\overrightarrow{u} = \{u_1, \ldots, u_m\}$ (which are read and updated in a concurrent way). We associate with each global variable u_i a place g_i containing a single token t_i, and we associate with this token a color $\alpha(t_i)$ representing the value of u_i, where $\alpha \in \Gamma$ is a special coloring symbol. Then, a process action $\ell \xrightarrow{\overrightarrow{u} := \overrightarrow{f}(\overrightarrow{u}, \overrightarrow{v})} \ell'$ (assigning to global variables values depending on both global variables and local variables of the process) is modeled by the transition:

$$x \in \ell, x_1 \in g_1, \ldots, x_m \in g_m \hookrightarrow y \in \ell', y_1 \in g_1, \ldots, y_m \in g_m :$$
$$\Big(\bigwedge_{i=1}^{n} \gamma_i(y) = \gamma_i(x)\Big) \wedge \bigwedge_{i=1}^{m} \alpha(y_i) = f_i(\alpha(x_1), \ldots, \alpha(x_m), \gamma_1(x), \ldots, \gamma_n(x))$$

In the modeling above, we consider that the execution of the process action is atomic. When assignments are not atomic, we must transform each of assignment action into a sequence of atomic operations: read first the global variables and assign their values to local variables, then compute locally the new values to be assigned to global variables, and finally assign these values to global variables.

Rendez-vous synchronization: Synchronization between a finite number of processes can be modeled as in Petri nets. CPNs allow in addition to put constraints on the colors (data) of the involved processes.

Priorities: Various notion of priorities, such as priorities between different classes of processes (defined by properties of their colors), or priorities between different actions, can be modeled in CPNs. This can be done by imposing in transition guards that transitions (performed by processes or corresponding to actions) of higher priority are not enabled. These constraints can be expressed using Π_1 formulas. In particular, checking that a place p is empty can be expressed by $\forall x \in p.\ \mathit{false}$. (Which shows that as soon as universally quantified formulas are allowed in guards, our models are as powerful as Turing machines, even for color logics over finite domains.)

Process identities: It is possible to associate with each newly created process an identity defined by an integer number. For that, we consider a special coloring symbol $Id \in \Gamma$ associating to each token the identity of the process it represents. To ensure that different processes have different identities, we express in the guard of every transition which creates a process (i.e., adds a token to the place corresponding to its initial control location) the fact that the identity of this process does not exist already among tokens in places corresponding to control locations. This can easily be done using a universally quantified (Π_1) formula. Therefore, a spawn action $\ell \xrightarrow{\mathsf{spawn}(\ell_0)} \ell'$ is modeled by:

$$x \in \ell \hookrightarrow y_1 \in \ell', y_2 \in \ell_0 :$$
$$Id(x) = Id(y_1) \wedge \big(\bigwedge_{i=1}^{n} \gamma_i(y_1) = \gamma_i(x)\big) \wedge \bigwedge_{loc \in \mathcal{L}} \forall t \in loc.\ \neg(Id(y_2) = Id(t))$$

and the modeling of other actions (such as local/global variables assignments) can be modified accordingly in order to propagate the process identity through the transition. Notice that process identities are different from token values. Indeed, in some cases (e.g., for modeling value passing as described below), we may use different tokens (at some special places representing buffers for instance) corresponding to the same Id.

Locks: Locks can be simply modeled using global variables storing the identity of the owner process, or a special value (e.g. -1) if it is free. A process who acquires the lock must check if it is free, and then write his identity:

$$x_1 \in \ell, x_2 \in lock \hookrightarrow y_1 \in \ell', y_2 \in lock : \alpha(x_2) = -1 \wedge \alpha(y_2) = Id(x_1) \wedge \ldots$$

To release the lock, a process assigns -1 to the lock, which can be modeled in a similar way. Other kinds of locks, such as reader-writer locks, can also be modeled in our framework (see Appendix E). The modeling of such locks when the number of readers and writers can be arbitrarily large requires the use of universal quantification in guards.

Value passing, return values: Processes may pass/wait for values to/from other processes with specific identities. They can use for that shared arrays of data indexed by process identities. Such an array A can be modeled in our framework using a special place containing for each process a token. Initially, this place is empty, and whenever a new process is created, a token with the same identity is added to this place. Then, to model that a process read/write on $A[i]$, we use a transition which takes from the place associated with A the token with Id equal to i, read/modifies the value attached

with this token, and put the token again in the same place. For instance, an assignment action $\ell \xrightarrow{A[k]:=e} \ell'$ executed by some process is modeled by the transition:

$$x_1 \in \ell, x_2 \in A \hookrightarrow y_1 \in \ell', y_2 \in A :$$
$$Id(x_1) = Id(y_1) \wedge \big(\bigwedge_{i=1}^{n} \gamma_i(x_1) = \gamma_i(y_1)\big) \wedge Id(x_2) = k \wedge \alpha(y_2) = e \wedge Id(y_2) = Id(x_2)$$

Notice that, while it is possible to model using CPNs systems manipulating parametric-size arrays (using multisets of tokens with integer colors), we cannot express in the decidable fragment Σ_2 of CML the fact that a multiset indeed encodes an array of elements indexed by integers in some given interval. The reason is that, while we can express in Π_1 the fact that each token has a unique color in the interval, we need to use Π_2 formulas to say that for each color in the interval there exists a token with that color. Nevertheless, for the verification of safety properties and checking invariants, it is not necessary to require the latter property.

6 Computing post and pre images

We prove hereafter closure properties of CML fragments under the computation of immediate successors and predecessors for CPNs. The main result of this section is:

Theorem 3. *Let S be a $\mathsf{CPN}[\Sigma_n]$, for $n \in \{1,2\}$. Then, for every closed formula φ in the fragment Σ_n of CML, it is possible to construct two closed formulas φ_{post} and φ_{pre} in the same fragment Σ_n such that $[\![\varphi_{\mathsf{post}}]\!] = \mathsf{post}_S([\![\varphi]\!])$ and $[\![\varphi_{\mathsf{pre}}]\!] = \mathsf{pre}_S([\![\varphi]\!])$.*

We give hereafter a sketch of the proof. Let φ be a closed formula, and let τ be a transition $\overrightarrow{x} \in \overrightarrow{p} \hookrightarrow \overrightarrow{y} \in \overrightarrow{q} : \psi$ of the system S. W.l.o.g., we suppose that φ and ψ are in special form. We define hereafter the formulas φ_{post} and φ_{pre} for this single transition. The generalization to the set of all transitions is straightforward.

The construction of the formulas φ_{post} and φ_{pre} is not trivial because our logic does not allow to use quantification over places and color mappings (associated with coloring symbols). Intuitively, the idea is to express first the effect of deleting/adding tokens, and then composing these operations to compute the effect of a transition.

Let us introduce two transformations $\ominus$ and $\oplus$ corresponding to deletion and creation of tokens. These operations are inductively defined on the structure of special form formulas in Table 1.

The operation $\ominus$ is parameterized by a vector $\overrightarrow{z}$ of token variables to be deleted, a mapping `loc` associating with token variables in $\overrightarrow{z}$ the places from which they will be deleted, and a mapping `col` associating with each coloring symbol in Γ and each token variable in $\overrightarrow{z}$ a fresh color variable in C. Intuitively, $\ominus$ projects a formula on all variables which are not in $\overrightarrow{z}$. Rule $\ominus_1$ substitutes in a color formula $\xi(\overrightarrow{t})$ all occurences of colored tokens in $\overrightarrow{z}$ by fresh color variables given by the mapping `col`. A formula $x = y$ is unchanged by the application of $\ominus$ is the token variables x and y are not in $\overrightarrow{z}$; otherwise, rule $\ominus_2$ replaces $x = y$ by true if it is trivially true (i.e., we have the same variable in both sides of the equality) or by false if x or y is in $\overrightarrow{z}$. Indeed, each token

variable in $\overrightarrow{z}$ represents (by the semantics of CPN) a different token, and since this token is deleted by the transition rule, it cannot appear in the reached configuration. Rules $\ominus_3$ and $\ominus_4$ are straightforward. Finally, rule $\ominus_5$ does a case splitting according to the fact whether a deleted token is precisely the one referenced by the existential token quantification or not.

The operation $\oplus$ is parameterized by a vector $\overrightarrow{z}$ of token variables to be added and a mapping `loc` associating with each variable in $z \in \overrightarrow{z}$ a place (in which it will be added). Intuitively, $\oplus$ transforms a formula taking into account that the added tokens by the transition were not present in the previous configuration (and therefore not constrained by the original formula describing the configuration before the transition). Then, the application of $\oplus$ has no effect on color formulas $\xi(\overrightarrow{t})$ (rule $\oplus_1$). When equality of tokens is tested, rule $\oplus_2$ takes into account that all added tokens are distinct and different from the existing tokens. For token quantification, rule $\oplus_5$ says that quantified tokens of the previous configuration cannot be equal to the added tokens.

Then, we define φ_{post} to be the formula:

$$\exists \overrightarrow{y} \in \overrightarrow{q}.\, \exists \overrightarrow{c}.\, \big((\varphi \wedge \psi) \ominus (\overrightarrow{x}, \overrightarrow{x} \mapsto \overrightarrow{p}, \Gamma \mapsto (\overrightarrow{x} \mapsto \overrightarrow{c}))\big) \oplus (\overrightarrow{y}, \overrightarrow{y} \mapsto \overrightarrow{q})$$

In the formula above, we first delete the tokens corresponding to $\overrightarrow{x}$ from the current configuration φ intersected with the guard of the rule ψ. Then, we add tokens corresponding to $\overrightarrow{y}$. Finally, we close the formula by quantifying existentially the color variables and the token variables corresponding to the added tokens.

Similarly, we define φ_{pre} to be the formula:

$$\exists \overrightarrow{x} \in \overrightarrow{p}.\, \exists \overrightarrow{c}.\, \big((\varphi \oplus (\overrightarrow{x}, \overrightarrow{x} \mapsto \overrightarrow{p})) \wedge \psi\big) \ominus (\overrightarrow{y}, \overrightarrow{y} \mapsto \overrightarrow{q}, \Gamma \mapsto (\overrightarrow{y} \mapsto \overrightarrow{c}))$$

For predecessor computation, we add to the current configuration the tokens represented by the left hand side of the rule $\overrightarrow{x}$ in order to obtain a configuration on which the guard ψ can be applied. Then, we remove the tokens added by the rule using token variables $\overrightarrow{y}$. Finally, we close the formula by quantifying existentially the color variables and the token variables corresponding to the added tokens. It is easy to see that if φ and ψ are in a fragment Σ_n, for any $n \geq 1$, then both of the formulas φ_{post} and φ_{pre} are also in the same fragment Σ_n.

Corollary 1. *Let S be a* $\mathsf{CPN}[\Sigma_1]$. *Then, for every formula φ in Π_1, it is possible to construct a formula $\varphi_{\widetilde{\mathsf{pre}}}$ also in Π_1 s.t. $[\![\varphi_{\widetilde{\mathsf{pre}}}]\!] = \widetilde{\mathsf{pre}}_S([\![\varphi]\!])$.*

7 Applications in Verification

We show in this section how to use the results of the previous section to perform various kinds of analysis. Let us fix for the rest of the section a colored tokens logic L with a decidable satisfiability problem, and a CPN S defined over L and the logic $\mathsf{CML}(L)$.

7.1 Pre-post condition reasoning

Given a transition τ in S and given two formulas φ and φ', $\langle \varphi, \tau, \varphi' \rangle$ is a Hoare triple if whenever the condition φ holds, the condition φ' holds after the execution of τ. In other

$$
\begin{array}{lrcl}
\ominus_1: & \xi(\overrightarrow{t}) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & \xi(\overrightarrow{t})[\mathtt{col}(\gamma)(z)/\gamma(z)]_{\gamma\in\Gamma, z\in\overrightarrow{z}} \\
\ominus_2: & (x=y) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & \begin{cases} x=y & \text{if } x,y \notin \overrightarrow{z} \\ \text{true} & \text{if } x \equiv y \\ \text{false} & \text{otherwise} \end{cases} \\
\ominus_3: & (\neg\varphi) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & \neg(\varphi \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})) \\
\ominus_4: & (\varphi_1 \vee \varphi_2) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & (\varphi_1 \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})) \vee (\varphi_2 \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})) \\
\ominus_5: & (\exists x \in p.\ \varphi) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) & = & \exists x \in p.\ (\varphi \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col})) \vee \\
 & & & \quad \bigvee_{z\in\overrightarrow{z}:\mathtt{loc}(z)=p} (\varphi[z/x]) \ominus (\overrightarrow{z}, \mathtt{loc}, \mathtt{col}) \\
\oplus_1: & \xi(\overrightarrow{t}) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & \xi(\overrightarrow{t}) \\
\oplus_2: & (x=y) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & \begin{cases} x=y & \text{if } x,y \notin \overrightarrow{z} \\ \text{true} & \text{if } x \equiv y \\ \text{false} & \text{otherwise} \end{cases} \\
\oplus_3: & (\neg\varphi) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & \neg(\varphi \oplus (\overrightarrow{z}, \mathtt{loc})) \\
\oplus_4: & (\varphi_1 \vee \varphi_2) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & (\varphi_1 \oplus (\overrightarrow{z}, \mathtt{loc})) \vee (\varphi_2 \oplus (\overrightarrow{z}, \mathtt{loc})) \\
\oplus_5: & (\exists x \in p.\ \varphi) \oplus (\overrightarrow{z}, \mathtt{loc}) & = & \exists x \in p.\ (\varphi \oplus (\overrightarrow{z}, \mathtt{loc})) \wedge \bigwedge_{z\in\overrightarrow{z}:\mathtt{loc}(z)=p} \neg(x=z)
\end{array}
$$

Table 1. Definition of the $\oplus$ and $\ominus$ operators.

words, we must have $\mathsf{post}_\tau([\![\varphi]\!]) \subseteq [\![\varphi']\!]$, or equivalently that $\mathsf{post}_\tau([\![\varphi]\!]) \cap [\![\neg\varphi']\!] = \emptyset$. Then, by Theorem 3 and Theorem 2 we deduce the following:

Theorem 4. *If S is a $\mathsf{CPN}[\Sigma_2]$, then the problem whether $\langle \varphi, \tau, \varphi' \rangle$ is a Hoare triple is decidable for every transition τ of S, every formula $\varphi \in \Sigma_2$, and every formula $\varphi' \in \Pi_2$.*

7.2 Bounded reachability analysis

An instance of the bounded reachability analysis problem is a triple $(\mathit{Init}, \mathit{Target}, k)$ where Init and Target are two sets of configurations, and k is a positive integer. The problem consists in deciding whether there exists a computation of length at most k which starts from some configuration in Init and reaches a configuration in Target. In other words, the problem consists in deciding whether $\mathit{Target} \cap \bigcup_{0\le i\le k} \mathsf{post}_S^i(\mathit{Init}) \neq \emptyset$, or equivalently whether $\mathit{Init} \cap \bigcup_{0\le i\le k} \mathsf{pre}_S^i(\mathit{Target}) \neq \emptyset$. The following result is a direct consequence of Theorem 3 and Theorem 2.

Theorem 5. *If S is a $\mathsf{CPN}[\Sigma_2]$, then, for every $k \in \mathbb{N}$, and for every two formulas $\varphi_I, \varphi_T \in \Sigma_2$, the bounded reachability problem $([\![\varphi_I]\!], [\![\varphi_T]\!], k)$ is decidable.*

7.3 Checking invariance properties

An instance of the *invariance checking problem* is given by a pair of sets of configurations (colored markings) $(\mathit{Init}, \mathit{Inv})$, and consists in deciding whether starting from any configuration in Init, every computation of S can only visit configurations in Inv, i.e., $\bigcup_{k\ge 0} \mathsf{post}_S^k(\mathit{Init}) \subseteq \mathit{Inv}$. This problem is of course undecidable in general. However, a

deductive approach using inductive invariants (provided by the user) can be adopted. We show that our results allow to automatize the steps of this approach.

A set of configurations $\mathbb{M}$ is an *inductive invariant* if $\mathsf{post}_S(\mathbb{M}) \subseteq \mathbb{M}$, or equivalently, if $\mathbb{M} \subseteq \widetilde{\mathsf{pre}}_S(\mathbb{M})$. By Theorem 3 and Theorem 2, we have:

Theorem 6. *If S is a $\mathsf{CPN}[\Sigma_2]$, then for every formula φ in $B(\Sigma_1)$, the problem of checking whether φ defines an inductive invariant is decidable.*

The deductive approach for establishing an invariance property considers the *inductive invariance checking problem* given by a triple $(\mathit{Init}, \mathit{Inv}, \mathit{Aux})$ of sets of configurations, and which consists in deciding whether (1) $\mathit{Init} \subseteq \mathit{Aux}$, (2) $\mathit{Aux} \subseteq \mathit{Inv}$, and (3) Aux is an inductive invariant. Indeed, a (sound and) complete rule for solving an invariance checking problem $(\mathit{Init}, \mathit{Inv})$ consists in finding a set of configurations Aux allowing to solve the inductive invariance checking problem $(\mathit{Init}, \mathit{Inv}, \mathit{Aux})$. The following result follows directly from Theorem 3, Theorem 2, and the previous theorem.

Theorem 7. *If S is a $\mathsf{CPN}[\Sigma_2]$, then the inductive invariance checking problem is decidable for every instance $(\llbracket \varphi_{\mathit{Init}} \rrbracket, \llbracket \varphi \rrbracket, \llbracket \varphi' \rrbracket)$ where $\varphi_{\mathit{Init}} \in \Sigma_2$, and $\varphi, \varphi' \in B(\Sigma_1)$.*

Of course, the difficult part in applying the deductive approach is to find useful auxiliary inductive invariants. One approach to tackle this problem is to try to compute the largest inductive invariant included in Inv which is the set $\bigcap_{k \geq 0} \widetilde{\mathsf{pre}}_S^k(\mathit{Inv})$. Therefore, a method to derive auxiliary inductive invariants is to try iteratively the sets Inv, $\mathit{Inv} \cap \widetilde{\mathsf{pre}}_S(\mathit{Inv})$, $\mathit{Inv} \cap \widetilde{\mathsf{pre}}_S(\mathit{Inv}) \cap \widetilde{\mathsf{pre}}_S^2(\mathit{Inv})$, etc. In many practical cases, only few strengthening steps are needed to find an inductive invariant. (Indeed, the user is able in general to provide accurate invariant assertions for each control point of his system.) The result below implies that the steps of this iterative strengthening method can be automatized when $\mathsf{CPN}[\Sigma_1]$ models and Π_1 invariants are considered. This result is a direct consequence of Corollary 1.

Theorem 8. *If S is a $\mathsf{CPN}[\Sigma_1]$, then for every formula φ in Π_1 and every positive integer k, it is possible to construct a formula in Π_1 defining the set $\bigcap_{0 \leq i \leq k} \widetilde{\mathsf{pre}}_S^i(\llbracket \varphi \rrbracket)$.*

We show in the full paper the applicability of our framework on a nontrivial example. We present the verification of a Reader-Writer lock for an unbounded number of processes using the inductive invariant checking approach. This example has been considered in [28] for a fixed number of processes.

8 Conclusion

We have presented a framework for reasoning about dynamic/parametric networks of processes manipulating data over infinite domains. We have provided generic models for these systems and a logic allowing to specify their configurations, both being parametrized by a logic on the considered data domain. We have identified a fragment of this logic having a decidable satisfiability problem and which is closed under post and pre image computation, and we have shown the application of these results in verification.

The complexity of the decision procedure and of the post/pre computation is exponential in the size of the formula, and more precisely in the number of quantified variables. However, formulas which appear in the analysis of systems such as parametrized/dynamic networks (such as assertions expressing invariants at each particular control location) are naturally in special form (see definition in Section 2.3) where each token variable is bound to a unique place (this allows to avoid the case splitting according to all possible mappings between token variables and places), and moreover, new token variables introduced by post/pre computations are of a fixed small number (the number of synchronized processes by the considered transition which is in general equal to two). These facts reduce significantly the complexity in practice.

Our framework allows to deal in a uniform way with all classes of systems manipulating infinite data domains with a decidable first-order theory. In this paper, we have considered instantiations of this framework based on logics over integers or reals (which allows to consider systems with numerical variables). Different data domains can be considered in order to deal with other classes of systems such as multithreaded programs where each process (thread) has an unbounded stack (due to procedure calls). We will address in more details the issue of applying our framework to the verification of multithreaded programs in a forthcoming paper. Our future work includes also the extension of our framework to other classes of systems and features such as dynamic networks of timed processes, networks of processes with broadcast communication, interruptions and exception handling, etc.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proc. of LICS'96*, pages 313–321, 1996.
2. P. A. Abdulla and G. Delzanno. On the Coverability Problem for Constrained Multiset Rewriting. In *Proc. of AVIS'06, Satellite workshop of ETAPS'06*, Vienna, Austria, 2006.
3. P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A Survey of Regular Model Checking. In *Proc. of CONCUR'04*, volume 3170 of *LNCS*. Springer, 2004.
4. Parosh Aziz Abdulla and Bengt Jonsson. Verifying networks of timed processes (extended abstract). In Bernhard Steffen, editor, *Proc. of TACAS'98*, volume 1384 of *LNCS*, pages 298–312. Springer, 1998.
5. A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. of CAV'00*. LNCS 1855, 2000.
6. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L.D. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *Proc. of CAV'01*, volume 2102 of *LNCS*. Springer, 2001.
7. Bernard Boigelot. *Symbolic Methods for Exploring Infinite State Space*. PhD thesis, Faculté des Sciences, Université de Liège, volume 189, 1999.
8. M. Bojanczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. In *Proc. of PODS'06*. ACM, 2006.
9. M. Bojanczyk, A. Muscholl, Th. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proc. of LICS'06*. IEEE, 2006.
10. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Perspectives of Mathematical Logic. Springer-Verlag, 1997. Second printing (Universitext) 2001.

11. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. of CONCUR'97*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.
12. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract Regular Model Checking. In *Proc. of CAV'04*, volume 3114 of *LNCS*. Springer, 2004.
13. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular Model Checking. In *Proc. of CAV'00*, volume 1855 of *LNCS*. Springer, 2000.
14. A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. Technical Report 2007-01, LIAFA lab, January 2007. Available at `http://www.liafa.jussieu.fr/~abou/publis.html`.
15. A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *Proc. of CONCUR'05*, volume 3653 of *LNCS*. Springer, 2005.
16. A. Bouajjani and T. Touili. On computing reachability sets of process rewrite systems. In *Proc. of RTA'05*, volume 3467 of *LNCS*. Springer, 2005.
17. Ahmed Bouajjani. Languages, Rewriting systems, and Verification of Infinte-State Systems. In *Proc. of ICALP'01*, volume 2076 of *LNCS*. Springer Pub., 2001.
18. M. Bozzano and G. Delzanno. Beyond Parameterized Verification. In *Proc. of TACAS'02*, volume 2280 of *LNCS*, Grenoble, France, 2002. Springer Pub.
19. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays ? In *Proc. of VMCAI'06*, volume 3855 of *LNCS*. Springer, 2006.
20. E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks. *TOPLAS*, 19(5), 1997.
21. G. Delzanno. An assertional language for the verification of systems parametric in several dimensions. *Electr. Notes Theor. Comput. Sci.*, 50(4), 2001.
22. G. Delzanno, J.-F. Raskin, and L. Van Begin. Towards the automated verification of multithreaded java programs. In *TACAS*, volume 2280 of *LNCS*, pages 173–187. Springer, 2002.
23. S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. In *Proc. of LICS'06*. IEEE, 2006.
24. E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS'98*. IEEE, 1998.
25. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proceedings of LICS '99*, pages 352–359. IEEE Computer Society, 1999.
26. A. Finkel and J. Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *Proc. of FST&TCS'02*, volume 2556 of *LNCS*. Springer, 2002.
27. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
28. C. Flanagan, S.N. Freund, and S. Qadeer. Thread-modular verification for shared-memory programs. In *Proc. of ESOP'02*, pages 262–277. LNCS 2305, 2002.
29. S. M. German and P. A. Sistla. Reasoning about systems with many processes. *JACM*, 39(3), 1992.
30. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. of CAV'98*, volume 1427 of *LNCS*. Springer, 1998.
31. P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Proc. Intern. Workshop on Automatic Verification Methods for Finite State Systems*. LNCS 407, 1989.

A Proof of Theorem 1

We reduce the halting problem of a Turing machine to the satisfiability problem of $\mathsf{CML}[\mathbb{P},\mathsf{OL}]$. Let us describe informally how we encode computations of a given Turing machine: A computation is a sequence of tape configurations, each of them being a sequence of symbols. Then, we associate a different to token with each position of each configuration in the computation. we associate with each of these tokens two integer colors: *step* corresponding to the the position number of its corresponding configuration in the computation (i.e., to ordering number of the computation step corresponding to the configuration), and *cell* corresponding to its position number in its own configuration. The other informations we need to associated with these tokens are the tape symbol, whether it corresponds to the position of the head of the machine, and in this case, the control state of the machine. These informations range over finite sets and therefore can be encoded using a finite number of different places. For instance, a token x in the place A_Head_q encodes a cell carrying the letter A and which corresponds to the position of the head of the machine, and the state of the machine q. We show that we can encode valid succession of configurations using Π_2 formulas.

Let $M = (Q,\Gamma,B,q_f,\Delta)$ be a turing machine, where Q is a finite set of state, Γ is the finite tape alphabet containing B the default blank symbol, q_f is the final state, and Δ is the transition relation. Without loss of generality we suppose that the machine has no deadlocks, and once it reach the final state it stops.

We define the set $\mathbb{P}$ of places to be the product $\Gamma \times \{Head, Nohead\} \times Q$. In order to describe the encoding of the transition of the machine, we need to introduce the following notations:

$$Head\_q(x) ::= \bigvee_{A\in\Gamma} A\_Head\_q(x)$$

$$Head(x) ::= \bigvee_{q\in Q} Head\_q(x)$$

$$Blank(x) ::= \bigvee_{q\in Q} B\_Head\_q(x) \vee \bigvee_{q\in Q} B\_NotHead\_q(x)$$

$$SameLetterPassive(x,y) ::= \bigwedge_{A\in\Gamma,q\in Q} A\_NotHead\_q(x) \Leftrightarrow A\_NotHead_q(y)$$

$$SameLetterActivate(x,y) ::= \bigwedge_{A\in\Gamma,q\in Q} A\_NotHead\_q(x) \Leftrightarrow \bigvee_{p\in Q} A\_Head\_p(y)$$

Then, we give hereafter the encoding of a run of the machine starting from the initial configuration and reaching the control state q_f. The encoding is given by the conjunction of several formulas we define below.

Initially (i.e., at the first step), all the cells in the tape contain the blank symbol, the machine is in the state q_0 and the head is at the left most position of the tape (i.e., cell number 0). This is expressed by the following formula:

$$Init ::= (\forall x.step(x) = 0 \Rightarrow Blank(x)) \wedge \exists x.step(x) = cell(x) = 0 \wedge Head\_q_0(x)$$

Then, we need to require that there is a complete and sound encoding of each tape (at every computation step) :

$$\begin{aligned} Tapes ::= \quad & (\forall x.\ \forall n.\ \exists y.\ step(x) = step(y) \wedge cell(y) = n) \\ & \wedge \forall x,y.\ (step(x) = step(y) \wedge cell(x) = cell(y)) \Rightarrow x = y \end{aligned}$$

and that at each step there is at most one cell corresponding to the position of the head:

$$OneHeadperStep ::= \forall x,y.(Head(x) \wedge Head(y)) \Rightarrow step(x) \neq step(y)$$

Moreover, we need to require that each step is followed by another one:

$$Liveness ::= \forall x.\exists y.step(y) > step(x)$$

and that the computation reaches at some step the final state:

$$Acceptance ::= \exists x.Head\_q_f(x)$$

Now, it remains to encode a transition step, i.e., to require that every two successive configurations correspond to a valid transition in the machine. Consider the case of a transition of the form: if the machine is at state p, it reads a symbol a on the tape, writes b at the same position, moves its head to the left, and goes to state q. (The other cases can be handled in a similar way.)

Informally we use token variable t_1, t_2, t_3 as follows : t_1 corresponds to the holder of the head and carry the letter a and the state p, t_3 represents the cell that replaces t_1 in the next step and carry the letter b, and t_2 corresponds the cell which is just to the left of t_3, that is the next holder of the head and of the state q. The variable t_4 is used to express that t_1 and t_2 belong to consecutive steps. The variables x and y express that all cells of the tape different from t_1, t_2, t_3 remain the same.

$$\begin{aligned} \forall t_1,t_2,t_3\ \ & [a\_Head\_p(t_1) \wedge \\ & step(t_1) < step(t_3) \wedge \neg\exists t_4.\ (step(t_1) < step(t_4) < step(t_3)) \wedge cell(t_3) = cell(t_1) \\ & \wedge step(t_3) = step(t_2) \wedge \neg\exists t_4.\ cell(t_2) < cell(t_4) < cell(t_3) \\ &] \Rightarrow\ b\_NotHead(t_3) \wedge Head\_q(t_2) \\ & \quad \wedge \forall x,y.\ [step(x) = step(t_1) \wedge step(y) = step(t_2) \wedge cell(x) = cell(y) \wedge y \neq t_3] \\ & \quad\ \Rightarrow (SameLetterPassive(x,y) \wedge y \neq t_2) \vee (y = t_2 \wedge SameLetterActivate(x,y)) \end{aligned}$$

B Proof of Theorem 2

We reduce the satisfiability problem of Σ_2 formulas to the satisfiability problem of Σ_0 formulas (which correspond to formulas in the token color logic which is supposed to has a decidable satisfiability problem).

We prove first that the fragment Σ_2 has the small model property, i.e., every satisfiable formula φ in Σ_2 has a model of a bounded size (where the size is the number of tokens in each place). Let φ be a closed formula in Σ_2 in prenex form $\exists \overrightarrow{x}.\ \exists \overrightarrow{z}.\ \forall \overrightarrow{y}.\ \phi$

where $\overrightarrow{x}, \overrightarrow{y}$ are token variables and $\overrightarrow{z}$ color variables, and let us assume that all variables are different.

Suppose that there is a colored marking $\langle M, \mu \rangle$ which satisfies φ. This means that there exist a vector of tokens $\overrightarrow{t}$ (resp. of colors $\overrightarrow{c}$) and a mapping θ (resp. δ) associating these tokens (resp. colors) to the variables $\overrightarrow{x}$ (resp. $\overrightarrow{z}$) such that $\langle M, \mu \rangle \models_{\theta,\delta} \forall \overrightarrow{y}.\phi$.

It is easy to see that, for every universally quantified formula, that if it is satisfied by a marking, it is also satisfied by all its submarkings (w.r.t inclusion ordering). Let $\langle M', \mu' \rangle$ be the finite colored marking defined as the restriction of $\langle M, \mu \rangle$ to tokens and colors in the vectors $\overrightarrow{t}$ and $\overrightarrow{c}$. Then clearly, we have $\langle M', \mu' \rangle \models_{\delta,\theta} \forall \overrightarrow{y}.\phi$, and therefore we have $\langle M', \mu' \rangle \models \exists \overrightarrow{x}.\exists \overrightarrow{z}.\phi$.

This shows a small model property for the fragment Σ_2 : every satisfiable formula $\varphi = \exists \overrightarrow{x}.\exists \overrightarrow{z} \forall \overrightarrow{z}.\phi$ has a model of size less or equal than $|\overrightarrow{x}|$. However this fact does not imply the decidability of the satisfiability problem since the color domains are infinite.

Nevertheless, we show that it is possible to reduce the satisfiability problem from Σ_2 to Σ_1. We denote by $[\overrightarrow{y} \to \overrightarrow{x}]$ the set of mappings σ from elements of $\overrightarrow{y}$ to elements of $\overrightarrow{x}$. Since the model $\langle M', \mu' \rangle$ is finite, the fact that $\langle M', \mu' \rangle \models_{\delta,\theta} \forall \overrightarrow{y}.\phi$ is equivalent to the fact that

$$\langle M', \mu' \rangle \models \exists \overrightarrow{x}.\ \exists \overrightarrow{z}.\ \bigwedge_{\sigma \in [\overrightarrow{y} \to \overrightarrow{x}]} \phi[\overrightarrow{y} \leftarrow \sigma(\overrightarrow{y})]$$

Finally let us show that the satisfiability problem can be reduced from Σ_1 to Σ_0. Consider a formula $\varphi = \exists \overrightarrow{x}.\phi$.

First, we do the following transformations: (1) we eliminate token equality by enumerating all the possible equivalent classes for equality between the finite number of variables in $\overrightarrow{x}$, (2) we eliminate formulas of the form $p(x)$ by enumerating all the possible mappings from a token variable x to a place, and (3) we replace terms of the form $\gamma(x)$ by fresh color variables. Let us describe more formally these transformations.

Let $\mathcal{B}(\overrightarrow{x})$ be the set of all possible equivalence classes (w.r.t. to the equality relation) over elements of $\overrightarrow{x}$: an element e in $\mathcal{B}(\overrightarrow{x})$ is a mapping from $\overrightarrow{x}$ to $\overrightarrow{x}$ that gives a single representant for each class.

Clearly φ is equivalent to

$$\bigvee_{e \in \mathcal{B}(\overrightarrow{x})} \exists \overrightarrow{y_e}.\ \bigwedge_{i \neq j} (y_i \neq y_j) \wedge \tilde{\phi}_e(\overrightarrow{y_e})$$

where $\overrightarrow{y_e}$ is $e(\overrightarrow{x})$ and $\tilde{\phi}_e$ is build from ϕ by giving to equalities $x_i = x_j$ the value true iff the representant $e(x_i)$ and $e(x_j)$ are the same.

Similarly, we eliminate from $\tilde{\phi}_e$ the occurences of formulas $p(x_i)$. We already have defined the notation $[\overrightarrow{x} \to \mathbb{P}]$ to be the set of all the mappings from elements in $\overrightarrow{x}$ to elements in $\mathbb{P}$. We have an equivalent formula :

$$\bigvee_{e \in \mathcal{B}(\overrightarrow{x})} \ \bigvee_{\sigma \in [\overrightarrow{y_e} \to \mathbb{P}]} \exists \overrightarrow{y_e}.\ \bigwedge_{i \neq j} (y_i \neq y_j) \wedge \overrightarrow{y_e} \in \overrightarrow{p_\sigma} \wedge \bar{\phi}_{e,\sigma}(\overrightarrow{y_e})$$

where $\overrightarrow{p_\sigma}$ is the value of the mapping $\sigma(\overrightarrow{y_e})$, and $\bar{\phi}_{e,\sigma}$ is built from $\tilde{\phi}_e$ by replacing each occurence of $p(y)$ by *true* iff $\sigma(y) = p$.

Finally, for each coloring symbol $\gamma \in \Gamma$ and each token variable y in the vector $\overrightarrow{y_e}$, we define a color variable $s_{\gamma,y}$, and we let $\overrightarrow{s}$ to be a vector containing all such color variables.

It is now easy to see that the formula

$$\exists \overrightarrow{y_e} . \bigwedge_{i \neq j} (y_i \neq y_j) \wedge \overrightarrow{y_e} \in \overrightarrow{p_\sigma} \wedge \overline{\phi}_{e,\sigma}(\overrightarrow{y_e})$$

is satisfiable iff the following Σ_0 formula

$$\exists \overrightarrow{s} . \overline{\phi}_{\sigma,e}[\gamma(y) \leftarrow s_{\gamma,y}]_{\gamma \in \Gamma, y \in \overrightarrow{y_e}}$$

is satisfiable. Therefore, the satisfiability problem of Σ_2 can be reduced to satisfiability problem of Σ_0, which is a decidable problem by hypothesis.

C Reader-Writer lock: An example of modeling

Reader-writer is a classical synchronisation scheme used in operating systems or other large scale systems. Several readers and writers work on common data. Readers may read data in parallel but they are exclusive with writers. Writers can only work in exclusive mode with other threads. *Reader-writer lock* is used to implement such kind of synchronization. Readers have to acquire the lock in *read mode*, and writers in *write mode*.

The implementation in Java of atomic operations for acquire and release in read and write mode is classical. This implementation uses an *integer* w to identifies the thread holding the lock in write mode or -1 if no such thread exists (threads identifiers are suppsed to be positive integers). Also, an *integer set* r is used to store the identifiers of all threads holding the lock in read mode. Acquire and release operations are accessing variables w and r in mutual exclusion.

Our model of reader-writer lock follows the implementation above. The (global) lock variable is modeled by a place *rw* where each token represents a thread using the lock (i.e., it has acquired but not yet released the lock). Each token in *rw* has two colors: *ty* gives the type of the access to the lock (read or write), and *Id* gives the identifier of the thread represented by the token. The *Id* color is useful to ensure that the releasing of a lock is done by the thread which acquired it. Since acquire and release should be atomic operations, we model them by single transitions (see Table C).

Let consider the program using the reader-writer lock given in Table C. It consists on several Reader and Writer threads, a global reader-writer lock variable l, a global variable x, and a local variable y for Reader threads. Writer threads change the value of the global variable x after acquiring in write mode the lock. Reader threads are setting their local variable y to a value depending on x after acquiring in read mode the lock. (Let us assume that for example the variables range over the domain of positive integers.) Each thread has an unique identifier represented by the _pid local variable.

For this program, the safety property to verify is the absence of race on variable x: value of x should not change while the lock is held in read mode, i.e., a reader thread at line 3 has a value of local variable y equal to f(x).

```
   threads Writer:                threads Reader:
1:  l.acq_write(_pid);        1:  l.acq_read(_pid);
2:  x = g(x);                 2:  y = f(x);
3:  l.rel_write(_pid);        3:  l.rel_read(_pid);
4:                            4:
```

Table 2. Example of program using reader-writer lock.

The CPN model corresponding to this program is given in Table C. We use the logic DL as colored tokens logic. To each control point we associate a place (e.g., place $r3$ for control point corresponding to line 3 of Reader threads) and a transition (e.g., transition r_3 for statement at line 3 of Reader threads). The global variable x is modeled as explained in previous section: we have a place p_x containing an unique token which color α stores the current value of x. With each token in the places corresponding to Reader control points we associate a color y to model the local variable y. We denote by $\Gamma(t',t)$ the (conjunctive) formula expressing that t' and t have the same colors. It can be observed that the obtained model is a $\mathsf{CPN}[\Sigma_2]$.

$$
\begin{aligned}
w_1 :&\quad t \in w1 \hookrightarrow t' \in w2, l' \in rw \\
&\quad : \neg(\exists z \in rw.\, true) \wedge Id(l') = Id(t) \wedge ty(l') = W \wedge \Gamma(t',t) \\
w_2 :&\ t \in w2, t_x \in p_x \hookrightarrow t' \in w3, t'_x \in p_x \\
&\quad : \alpha(t'_x) = g(\alpha(t_x)) \wedge \Gamma(t',t) \\
w_3 :&\ t \in w3, l \in rw \hookrightarrow t' \in w4 \\
&\quad : Id(l) = Id(t) \wedge ty(l) = W \wedge \Gamma(t',t)
\end{aligned}
$$

$$
\begin{aligned}
r_1 :&\quad t \in r1 \hookrightarrow t' \in r2, l' \in rw \\
&\quad : \neg(\exists z \in rw.\, ty(z) = W) \wedge Id(l') = Id(t) \wedge ty(l') = R \wedge \Gamma(t',t) \\
r_2 :&\ t \in r2, t_x \in p_x \hookrightarrow t' \in r3, t'_x \in p_x \\
&\quad : y(t') = f(\alpha(t_x)) \wedge Id(t') = Id(t) \wedge \Gamma(t'_x, t_x) \\
r_3 :&\ t \in r3, l \in rw \hookrightarrow t' \in r4 \\
&\quad : Id(l) = Id(t) \wedge ty(l) = R \wedge \Gamma(t',t)
\end{aligned}
$$

Table 3. Model of reader-writer lock.

The race-free property that the system must satisfy can be expressed by the following Π_1 formula:

$$RF \equiv \forall t \in r3.\, \forall t_x \in p_x.\, y(t) = f(\alpha(t_x))$$

Actually, in order to establish the invariance of the property above, it must be strengthened by other auxiliary properties:

– Place p_x contains a single token:

$$A_x \equiv \forall x, x' \in p_x.\ x = x'$$

– Reader-writer lock is either kept by a set of readers or by a unique writer:

$$\begin{aligned} RW \equiv \forall u, u' \in rw.\ (ty(u) = ty(u')) \wedge (ty(u) = W \implies u = u') \\ \wedge\ (ty(u) = R \vee ty(u) = W) \end{aligned}$$

– For all threads in places $w2$ and $w3$ of the Writer, the tokens in the lock place have the same identities and are of writer type:

$$RW_w \equiv \forall w \in \{w2, w3\}.\ \forall l_w \in rw.\ Id(w) = Id(l_w) \wedge ty(l_w) = W$$

– If threads exist in places $r2$ and $r3$ of the Reader, then there is a token in the lock place with reader type:

$$RW_r \equiv (\exists r \in \{r2, r3\}.\ true) \implies (\exists l_r \in rw.\ ty(l_r) = R)$$

It can be seen that all the formulas above are in the fragment $B(\Sigma_1)$.

We show in Section E how to check these properties on the model given in Table C.

D Reader-writer lock: An example of Post image computation

Consider the reader-writer example of Section C. We show hereafter the computation of the post-image of the RW property w.r.t. the transition w_1.

First, we instantiate the $\phi_{\texttt{post}}$ formula given in the proof of Theorem 3 using RW for the initial closed formula ϕ, and the side condition of transition w_1 for ψ. The color Id of the token t deleted by w_1 is mapped to a new color variable Id_t. Colors symbols not used for the deleted tokens (e.g., ty and α for t and t') are not mapped to color variables.

$$\begin{aligned} \mathsf{post}_{w_1}(RW) \equiv \exists t' \in w2, l' \in rw.\ \exists Id_t.\ [RW \wedge \neg(\exists z \in rw.\ true) \wedge Id(l') = Id(t) \\ \wedge ty(l') = W \wedge Id(t') = Id(t) \\ ] \ominus (t \in w1, Id(t) \mapsto Id_t)) \\ \oplus (t' \in w2, l' \in rw) \end{aligned}$$

Next, we apply the definition of $\ominus$ and $\oplus$ operators, mainly rules $\ominus_1$, $\ominus_5$, and $\oplus_5$. For example, by applying rule $\oplus_5$ on formula $\exists z \in rw.true$ for added token $l' \in rw$, we obtain $\exists z \in rw.\ true \wedge z \neq l'$.

$$\begin{aligned} \mathsf{post}_{w_1}(RW) \equiv \exists t' \in w2, l' \in rw.\ \exists Id_t.\ RW'(l') \wedge \neg(\exists z \in rw.\ z \neq l') \\ \wedge Id(l') = Id_t \wedge ty(l') = W \wedge Id(t') = Id_t \end{aligned}$$

where:

$$\begin{aligned} RW'(l') \equiv \forall u, u' \in rw.\ (ty(u) = ty(u') \wedge (ty(u) = W \implies u = u') \\ \wedge (ty(u) = R \vee ty(u) = W)) \\ \vee u' = l' \vee u = l' \end{aligned}$$

To obtain this result, we applied the following property of the special form universal quantification: $(\forall x \in p.\ \phi(x)) \vee \phi' \leftrightarrow (\forall x \in p.\ \phi(x) \vee \phi')$ with $x \notin FreeVars(\phi')$. The resulting formula is in the Σ_2 fragment.

$$\begin{aligned}\mathsf{post}_{w_1}(RW) \equiv \exists t' \in w2, l' \in rw.\ \exists Id_t.\ \forall u, u', z.\\ (\neg rw(u) \vee \neg rw(u') \vee\\ (ty(u) = ty(u') \wedge (ty(u) = W \implies u = u') \wedge (ty(u) = R \vee ty(u) = W))\\ \vee u' = l' \vee u = l')\\ \wedge (\neg rw(z) \vee z = l') \wedge Id(l') = Id_t \wedge ty(l') = W \wedge Id(t') = Id_t\end{aligned}$$

The computation of the post-image w.r.t. other transitions of the model is given in Section E.

E Reader-writer lock example

We illustrate here the use of our framework by showing that it is possible to carry out an invariant proof within the decidable fragment of our CML logic. The proof presented here has been done manually following the construction presented in the paper, and therefore, our results can be used to automatize each of its steps.

We consider the example of Reader-Writer lock introduced in Section 7, and we prove that $I \equiv (A_x \wedge RW \wedge RW_w \wedge RW_r \wedge RF)$ is an invariant property for the CMRS model given in Table C. Consequently, we show that the corresponding program is free of data races on variable x.

Let us recall from Section C each component of the invariant formula above:

$$\begin{aligned}A_x &\equiv \forall x, x' \in p_x.\ x = x'\\ RW &\equiv \forall u, u' \in rw.\ (ty(u) = ty(u')) \wedge (ty(u) = R \vee ty(u) = W)\\ &\qquad \wedge (ty(u) = W \implies u = u')\\ RW_w &\equiv \forall w \in \{w2, w3\}.\ \forall l_w \in rw.\ Id(w) = Id(l_w) \wedge ty(l_w) = W\\ RW_r &\equiv (\exists r \in \{r2, r3\}.\ true) \implies (\exists l_r \in rw.\ ty(l_r) = R)\\ RF &\equiv \forall a \in r3, t_x \in p_x.\ y(a) = f(\alpha(t_x))\end{aligned}$$

Transition w_1 of the writer models the writer acquiring the lock in write mode. The post-image of I through this rule is given by Theorem 8.1:

$$\begin{aligned}\mathsf{post}_{w_1}(I) \equiv \exists t' \in w2.\ \exists l' \in rw.\ \exists Id_t.\\ [A_x \wedge RW \wedge RW_w \wedge RW_r \wedge RF\\ \wedge \neg(\exists z \in rw.\ true) \wedge Id(l') = Id(t) \wedge ty(l') = W \wedge Id(t) = Id(t')\\ ] \ominus (t \in w1, Id(t) \mapsto Id_t)\\ \oplus (t' \in w2, l' \in rw)\end{aligned}$$

We apply the definition given in Table 1 for $\oplus$ and $\ominus$ operations.

$$\begin{aligned}\mathsf{post}_{w_1}(I) \equiv \exists t' \in w2.\ \exists l' \in rw.\ \exists Id_t.\\ A_x \wedge RW'(l') \wedge RW'_w(t', l') \wedge RW'_r(l') \wedge RF\\ \wedge (\forall z \in rw.\ z = l') \wedge Id(l') = Id_t \wedge ty(l') = W \wedge Id_t = Id(t')\end{aligned}$$

where we denote by $\phi(v)$ a formula ϕ where variable v is free, and the subformula used are defined by:

$$\begin{aligned}
RW'(l') &\equiv \forall u, u' \in rw.\ (ty(u) = ty(u') \wedge (ty(u) = R \vee ty(u) = W) \\
&\qquad \wedge (ty(u) = W \Rightarrow u = u')) \\
&\qquad \vee u = l' \vee u' = l' \\
RW'_w(t', l') &\equiv \forall w \in \{w2, w3\}.\ \forall l_w \in rw.\ (Id(w) = Id(l_w) \wedge ty(l_w) = W) \\
&\qquad \vee (l_w = l') \vee (w = t') \\
RW'_r(l') &\equiv (\exists r \in \{r2, r3\}.\ true) \implies (\exists l_r \in rw.\ ty(l_r) = R \wedge l_r \neq l')
\end{aligned}$$

Due to the special form of formula (all tokens are localized), the formula corresponding to the post-image increases reasonably (6 atomic formula are added). Using the fact that subformulas A_x and RF do not contain free variables, we obtain the simpler formula below:

$$\begin{aligned}
&\mathsf{post}_{w_1}(I) \equiv A_x \wedge RF \wedge P \\
&\text{where } P \equiv \exists t' \in w2.\ \exists l' \in rw.\ \exists Id_t. \\
&\qquad RW'(l') \wedge RW'_w(t', l') \wedge RW'_r(l') \\
&\qquad \wedge (\forall z \in rw.\ z = l') \wedge Id(l') = Id_t \wedge ty(l') = W \wedge Id_t = Id(t')
\end{aligned}$$

In order to prove that $\mathsf{post}_{w_1}(I) \implies I$ is valid, we check the satisfiability of the formula $\mathsf{post}_{w_1}(I) \wedge \neg I$. We simplify this formula by removing subformula appearing in positive and negative form:

$$\begin{aligned}
\mathsf{post}_{w_1}(I) \wedge \neg I &\equiv A_x \wedge RF \wedge P \wedge (\neg A_x \vee \neg RF \vee \neg RW \vee \neg RW_w \vee \neg RW_r) \\
&\equiv A_x \wedge RF \wedge P \wedge (\neg RW \vee \neg RW_w \vee \neg RW_r) \\
&\equiv (A_x \wedge RF \wedge P \wedge \neg RW) \\
&\quad \vee (A_x \wedge RF \wedge P \wedge \neg RW_w) \\
&\quad \vee (A_x \wedge RF \wedge P \wedge \neg RW_r)
\end{aligned}$$

In the following, we show how the second disjunct is proven to be unsatisfiable. For the other disjuncts, the reasoning is very similar. First, we compute its prenex form: some quantified variables are renamed in order to have unique names for each quantified

variables; then, we move quantifiers behind the formula.

$$
\begin{aligned}
& A_x \wedge RF \wedge P \wedge \neg RW_w \\
& \equiv \exists t' \in w2.\ \exists l' \in rw.\ \exists w' \in \{w2, w3\}.\ \exists l'_w \in rw.\ \exists Id_t. \\
& \quad A_x \wedge RF \wedge RW'(l') \wedge RW'_w(t', l') \wedge RW'_r(l') \\
& \quad \wedge (\forall z \in rw.\ z = l') \wedge Id(l') = Id_t \wedge ty(l') = W \wedge Id_t = Id(t') \\
& \quad \wedge (Id(w') \neq Id(l'_w) \vee ty(l'_w) \neq W) \\
& \equiv \exists t' \in w2.\ \exists l' \in rw.\ \exists w' \in \{w2, w3\}.\ \exists l'_w \in rw.\ \exists l_r \in rw.\ \exists Id_t. \\
& \quad \forall x, x' \in p_x.\ \forall a \in r3.\ \forall t_x \in p_x.\ \forall u, u' \in rw \\
& \quad \forall w \in \{w2, w3\}.\ \forall l_w \in rw.\ \forall r \in \{r2, r3\}.\ \forall z \in rw. \\
& \quad\quad (x = x') \wedge y(a) = f(\alpha(t_x)) \\
& \quad\quad \wedge ((ty(u) = ty(u') \wedge (ty(u) = R \vee ty(u) = W) \wedge (ty(u) = W \implies u = u') \\
& \quad\quad\quad \vee u = l' \vee u' = l') \\
& \quad\quad \wedge ((Id(w) = Id(l_w) \wedge ty(l_w) = W) \vee (l_w = l') \vee (w = t')) \\
& \quad\quad \wedge (ty(l_r) = R \wedge l_r \neq l') \\
& \quad\quad \wedge z = l' \wedge Id(l') = Id_t \wedge ty(l') = W \wedge Id_t = Id(t') \\
& \quad\quad \wedge (Id(w') \neq Id(l'_w) \vee ty(l'_w) \neq W)
\end{aligned}
$$

Second, we reduce the Σ_2 formula above to a Σ_1 formula. The procedure defined by the proof of Theorem 4.1. generates $2^1 * 3^4 * 1 = 162$ conjuncts from the elimination of universally quantified variables. However, the conjuncts obtained are very simple because a lot of subformulas produced by this procedure are identities of (same) variable, which may be replaced by their truth value and so simplify the conjunct. It can be seen that the formula is unsatisfiable because it says that (1) all tokens in rw are equal to l' in subformula $\exists l' \in rw ... \forall z \in rw .... \ ... \wedge (z = l') ...$ and (2) there exists a token in rw, l_r which is not equal to l'. Clearly (1) and (2) are in contradiction, so the formula is unsatisfiable.

The transition w_2 of the writer models the writer mutating variable x. The post-image of I through this transition is:

$$
\begin{aligned}
\mathsf{post}_{w_2}(I) \equiv\ & \exists t' \in w3.\ \exists t'_x \in p_x.\ \exists Id_t, \alpha_x. \\
& [A_x \wedge RW \wedge RW_w \wedge RW_r \wedge RF \\
& \ \wedge \alpha(t'_x) = g(\alpha(t_x)) \wedge Id(t') = Id(t) \\
&] \ominus (t \in w2, t_x \in p_x, Id(t) \mapsto Id_t, \alpha(t_x) \mapsto \alpha_x) \\
& \ \oplus (t' \in w2, t'_x \in p_x)
\end{aligned}
$$

We apply the definition given in Table 1 for $\oplus$ and $\ominus$ operations and explicit subformulas changed by these operations.

$$\begin{aligned}
\mathsf{post}_{w_2}(I) \equiv\ & \exists t' \in w3.\ \exists t'_x \in p_x.\ \exists Id_t, \alpha_x. \\
& A'_x(t'_x) \wedge RW \wedge RW'_w(t', Id_t) \wedge RW_r \wedge RF'(t'_x, \alpha_x) \\
& \wedge \alpha(t'_x) = g(\alpha_x) \wedge Id(t') = Id_t \\
\text{where } A'_x(x') \equiv\ & \forall x_1, x'_1 \in p_x.\ (x'_1 = t'_x \vee x_1 = t'_x) \\
RW'_w(t', Id_t) \equiv\ & (\forall w \in \{w2, w3\}.\ (\forall l_w \in rw.\ (Id(w) = Id(l_w) \wedge ty(l_w) = W)) \\
& \qquad \vee (w3(w) \wedge w = t')) \\
\wedge\ & (\forall l_w \in rw.\ Id_t = Id(l_w) \wedge ty(l_w) = W) \\
RF'(t'_x, \alpha_x) \equiv\ & \forall a \in r3, t_x \in p_x. (y(a) = f(\alpha(t_x)) \vee t_x = t'_x) \wedge y(a) = f(\alpha_x)
\end{aligned}$$

Using the fact that subformulas RW and RW_r do not contain free variables, we obtain the simpler formula below:

$$\begin{aligned}
& \mathsf{post}_{w_2}(I) \equiv RW \wedge RW_r \wedge P \\
& \quad \text{where } P \equiv \exists t' \in w3.\ \exists t'_x \in p_x.\ \exists Id_t, \alpha_x.\ A'_x(t'_x) \wedge RW'_w(t', Id_t) \wedge RF'(t'_x, \alpha_x) \\
& \qquad\qquad \wedge \alpha(t'_x) = g(\alpha_x) \wedge Id(t') = Id_t
\end{aligned}$$

In order to prove that $\mathsf{post}_{w_2}(I) \implies I$ is valid, we check the satisfiability of the formula $\mathsf{post}_{w_1}(I) \wedge \neg I$, which may be simplified by eliminating subformula appearing in positive and negative form as follows:

$$\begin{aligned}
\mathsf{post}_{w_2}(I) \wedge \neg I &\equiv RW \wedge RW_r \wedge P \wedge (\neg A_x \vee \neg RF \vee \neg RW_w) \\
&\equiv (RW \wedge RW_r \wedge P \wedge \neg A_x) \\
&\quad \vee (RW \wedge RW_r \wedge P \wedge \neg RF) \\
&\quad \vee (RW \wedge RW_r \wedge P \wedge \neg RW_w)
\end{aligned}$$

It can be shown that each disjunct is unsatisfiable. For example, we consider the third disjunct which may be written as follows:

$$\begin{aligned}
& RW \wedge RW_r \wedge P \wedge \neg RW_w \\
& \equiv (\forall u, u' \in rw.\ (ty(u) = ty(u')) \wedge (ty(u) = R \vee ty(u) = W) \wedge (ty(u) = W \implies u = u')) \\
& \wedge\ RW_r \\
& \wedge\ (\exists t' \in w3.\ \exists t'_x \in p_x.\ \exists Id_t, \alpha_x.\ A'_x(t'_x) \wedge RF'(t'_x, \alpha_x) \\
& \qquad \wedge (\forall w \in \{w2, w3\}.\ (\forall l_w \in rw.\ (Id(w) = Id(l_w) \wedge ty(l_w) = W)) \\
& \qquad\qquad \vee (w3(w) \wedge w = t')) \\
& \qquad \wedge (\forall l_w \in rw.\ Id_t = Id(l_w) \wedge ty(l_w) = W) \\
& \qquad \wedge \alpha(t'_x) = g(\alpha_x) \wedge Id(t') = Id_t) \\
& \wedge\ \neg(\forall w \in \{w2, w3\}.\ \forall l_w \in rw.\ Id(w) = Id(l_w) \wedge ty(l_w) = W)
\end{aligned}$$

which can be proved by our method as been unsatisfiable. The reason is that $\neg RW_w$ is in contradiction with the subformula $RW'_w(t', Id_t) \wedge Id(t') = Id_t$ of P.

The transition w_3 models the releasing of the lock. The post-image of I through this rule is:

$$
\begin{aligned}
\mathsf{post}_{w_3}(I) \equiv\ & \exists t' \in w4.\ \exists Id_t, Id_l, ty_l. \\
& [A_x \wedge RW \wedge RW_w \wedge RW_r \wedge RF \\
& \wedge Id(t) = Id(l) \wedge ty(l) = W \wedge Id(t') = Id(t) \\
&] \ominus (t \in w3, l \in rw, Id(t) \mapsto Id_t, Id(l) \mapsto Id_l, ty(l) \mapsto ty_l) \\
& \oplus (t' \in w4) \\
\equiv\ & \exists t' \in w4.\ \exists Id_t, Id_l, ty_l. \\
& A_x \wedge RW'(ty_l) \wedge RW'_w(Id_t, Id_l, ty_l) \wedge RW'_r(Id_l, ty_l) \wedge RF \\
& \wedge Id_t = Id_l \wedge ty_l = W \wedge Id(t') = Id_t \\
\text{where } RW'(ty_l) \equiv\ & (\forall u, u' \in rw.\ \ (ty(u) = ty(u')) \wedge (ty(u) = R \vee ty(u) = W) \wedge \\
& \qquad\qquad (ty(u) = W \implies u = u')) \\
& \wedge (\forall u' \in rw.\ \ (ty_l = ty(u')) \wedge (ty_l = R \vee ty_l = W) \wedge \\
& \qquad\qquad (ty_l = W \implies false)) \\
& \wedge (ty_l = ty_l) \wedge (ty_l = R \vee ty_l = W) \wedge (ty_l = W \implies true) \\
RW'_w(Id_t, Id_l, ty_l) \equiv\ & (\forall w \in \{w2, w3\}.\ \forall l_w \in rw.\ Id(w) = Id(l_w \wedge ty(l_w) = W) \\
& \wedge\ (\forall l_w \in rw.\ Id_t = Id(l_w) \wedge ty(l_w) = W) \\
& \wedge\ (Id_t = Id_l \wedge ty_l = W) \\
RW'_r(Id_l, ty_l) \equiv\ & \neg(\exists r \in \{r2, r3\}.\ true) \vee (\exists l_r \in rw.\ ty(l_r) = R) \vee (ty_l = R)
\end{aligned}
$$

Using the same reasoning than for previous transitions, it can be shown that $\mathsf{post}_{w_3}(I)$ is included in I.

At this point, we shown that I is an invariant of the writer rules. Showing that I is an invariant by post-images of reader transitions is very similar.