Rewriting Systems with Data

A Framework for Reasoning about Systems with Unbounded Structures over Infinite Data Domains*

Ahmed Bouajjani[1], Peter Habermehl[12], Yan Jurski[1], and Mihaela Sighireanu[1]

[1] LIAFA, CNRS & U. Paris 7, Case 7014, 2 place Jussieu, 75251 Paris 05, France.
[2] LSV, CNRS & ENS Cachan, 61 av Président Wilson, 94235 Cachan, France.

Abstract. We introduce a uniform framework for reasoning about infinite-state systems with unbounded control structures and unbounded data domains. Our framework is based on constrained rewriting systems on words over an infinite alphabet. We consider several rewriting semantics: factor, prefix, and multiset rewriting. Constraints are expressed in a logic on such words which is parametrized by a first-order theory on the considered data domain. We show that our framework is suitable for reasoning about various classes of systems such as recursive sequential programs, multithreaded programs, parametrized and dynamic networks of processes, etc. Then, we provide generic results (1) for the decidability of the satisfiability problem of the fragment $\exists^*\forall^*$ of this logic provided that the underlying logic on data is decidable, and (2) for proving inductive invariance and for carrying out Hoare style reasoning within this fragment. We also show that the reachability problem if decidable for a class of prefix rewriting systems with integer data.

1 Introduction

Software verification requires in general reasoning about infinite-state models. The sources of infinity in software models are multiple. They can be related for instance to the complex control these system may have due, e.g., to recursive procedure calls, communication through fifo channels, dynamic creation of concurrent processes, or the consideration of a parametric number of parallel processes. Other important sources of infinity are related to the manipulation of variables and (dynamic) data structures ranging over infinite data domains such as integers, reals, arrays, heap structures like lists and trees, etc.

In the last few years, a lot of effort has been devoted to the development of theoretical frameworks for the formal modeling and the automatic analysis of several classes of software systems. Rewriting systems (on words or terms), as well as related automata-based frameworks, have been shown to be adequate for reasoning about various classes of systems such as recursive programs, multithreaded programs, parametrized or dynamic networks of identical processes, communicating systems through fifo-channels, etc. (see, e.g., [11, 4, 13] for survey

* Partially supported by the french ANR project ACI-06-SETI-001.

papers). These works address in general the problem of handling systems with complex control structures, but where the manipulated data range of finite domains, basically booleans. Other existing works address the problem of handling models with finite control structures, but which manipulate variables over infinite data domains such as counters, clocks, etc., or unbounded data structures (over finite alphabets) such as stacks, queues, limited forms of heap memory (e.g., lists, trees), etc. [2, 14, 29, 8, 6, 26, 25, 27, 12, 15]. Notice that the boundary between systems with infinite control and systems with infinite data is not sharp. For instance, recursive programs can be modeled as prefix rewrite systems which are equivalent to pushdown systems, and (classes of) multithreaded programs can be modeled using multiset rewrite systems which are equivalent to Petri nets and to vector addition systems (a particular class of counter machines).

As already said, in all the works mentionned above, only one source of infinity is taken into account (while the others are either ignored or abstracted away). Few works dealing with different sources of infinity have been carried out nevertheless, but the research on this topic is still in its emerging phase [5, 1, 21, 19, 17, 3, 16]. In this paper, we propose a uniform framework for reasoning about infinite-state systems with both unbounded control structures and unbounded data domains. Our framework is based on word rewriting systems over infinite alphabets where each element is composed from a label over a finite set of symbols and a vector of data in a potentially infinite domain. Words over such an alphabet are called data words and rewriting systems on such words are called data word rewriting systems (DWRS for short). A DWRS is a set of rewriting rules with constraints on the data carried by the elements of the words.

The framework we propose allows to consider different rewriting semantics and different theories on data, and allows also to apply in a generic way decision procedures and analysis techniques. The rewriting semantics we consider are either the factor rewriting semantics (which consists in replacing any factor in the word corresponding to the left hand side or a rule by the right hand side), as well as the prefix and the multiset rewriting semantics. The constraints in the rewriting systems are expressed in a logic called DWL which is an extension of the monadic first-order theory of the natural ordering on positive integers (corresponding to positions on the word) with a theory on data allowing to express the constraints on the data values at each position of the word. The theory on data, which is a parameter of the logic DWL, can be any fist-order theory such as Presburger arithmetics, or the first-order theory on reals.

We show that this framework is expressive enough to model various classes of infinite-state systems. Prefix rewriting systems are used to model recursive programs with global and local variables over infinite data domains. Factor rewriting systems are used for modeling parametrized networks of processes with a linear topology (i.e., there is a total ordering between the identities of the processes). This is for instance the case of various parallel and/or distributed algorithms. (We give as an example a model for the Lamport's Bakery algorithm for mutual exclusion.) Multiset rewriting systems can be used for modeling multithreaded programs or dynamic/parametrized networks where the information about identities of processes is not relevant. This is the case for various systems such as cache coherence protocols (see, e.g., [23]).

We address the decidability of the satisfiability problem of the logic DWL. We show that this problem is undecidable for very weak theories on data already for the fragment of $\forall^*\exists^*$ formulas. On the other hand, we prove the generic result that whenever the underlying theory on data has a decidable satisfiability problem, the fragment of $\exists^*\forall^*$ formulas of DWL has also a decidable satisfiability problem.

Then, we address the issue of automatic analysis of DWRS models. We provide two kinds of results. First, we consider the problem of carrying out post and pre condition reasoning based on computing immediate successors and immediate predecessors of sets of configurations. We prove, again in a generic way, that the fragment of $\exists^*\forall^*$ formulas in DWL is effectively closed under the computation of post and pre images by rewriting systems with constraints in $\exists^*\forall^*$. We show how this result, together with the decidability result of the satisfiability problem in $\exists^*\forall^*$, can be used for deciding whether a given assertion is an inductive invariant of a system, or whether the specification of an action is coherent, that is, the execution of an action starting from the pre condition leads to configurations satisfying the post condition. The framework we present here generalizes the one we introduced recently in [16] based on constrained multiset rewriting systems. Our generalization to word factor and prefix rewriting systems allows to deal in a uniform and natural way with a wider class of systems where reasoning about linearly ordered structures is needed.

Finally, we consider the problem of solving the reachability problem for a subclass of DWRS. We provide a new decidability result of this problem for the class of context-free prefix rewriting systems (i.e., where the left hand side of each rule is of size 1) over the data domain of integers with difference constraints. (Extensions of this class lead to undecidabilty.) This results generalizes a previous result we have established few years ago in [17] for a more restricted class of systems where not all difference constraints were allowed.

Related work: Regular model checking has been defined as a uniform framework for reasoning about infinite-state systems [29, 28, 18, 4]. However, this framework is based on finite-state automata and transducers over finite alphabets which does not allow to deal in a simple and natural way with systems with both unbounded control and data domains. The same holds for similar frameworks based on word/tree rewriting systems over a finite alphabet (e.g., [11, 13]).

Works on the analysis of models for systems with two sources of infinity such as networks of infinite-stat processes are not very numerous in the literature. In [5], the authors consider the case of networks of 1-clock timed systems and show that the verification problem for a class of safety properties is decidable under some restrictions on the used constraints. Their approach has been extended in [21, 19] to a particular class of multiset rewrite systems with constraints (see also [3] for recent developments of this approach). In [17], we have considered the case of prefix rewrite systems with integer data which can be seen as models of recursive programs with one single integer parameter. Again, under some restrictions on the used arithmetical constraints, we have shown the decidability of the reachability problem. The result we prove in section **??** generalization our previous result of [17].

Recently, we have defined a generic framework for reasoning about parametrized and dynamic networks of infinite-state processes based on constrained multiset rewrite systems [16]. The work we present generalizes that work to other classes of rewriting systems.

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., [7]). In [7], 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. [20], 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 [7], 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, our decidable logical fragment has a weaker ability of expressing ordering constraints on positions (used, e.g., to represent identities of processes in parametrized/dynamic networks), but allows *any* kind of data, provided that the used theory on the considered data domain is decidable. For instance, we can use in our logic general Presburger constraints whereas [7] and [20] allow limited classes of constraints.

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) [10, 9, 24]. The obtained positive results so far concern logics with 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. Contrary to these works, our approach is to prefer weakening the first-order/model language for describing the structures while preserving the capacity of expressing constraints on data. We believe that this approach could be more useful in practice since it allows to cover a large class of applications as this paper tries to show.

2 A logic for reasoning about words over data domains

2.1 Preliminaries

Let Σ be a finite alphabet, and let $\mathbb{D}$ be a potentially infinite *data domain*. For a given $N \in \mathbb{N}$ such that $N \geq 1$, words over $\Sigma \times \mathbb{D}^N$ are called N-dim *data words*. Let $(\Sigma \times \mathbb{D}^N)^*$ (resp. $(\Sigma \times \mathbb{D}^N)^\omega$) be the set of finite (resp. infinite) data words, and let $(\Sigma \times \mathbb{D}^N)^\infty$ be the union of these two sets. Given a data word

σ, we denote by $|\sigma|$ the (finite or infinite) length of σ. A word $\sigma \in (\Sigma \times \mathbb{D}^N)^\infty$ can be considered as a mapping from $[0, |\sigma|)$ to $\Sigma \times \mathbb{D}^N$, i.e., $\sigma = \sigma(0)\sigma(1)\ldots$. Given $e = (A, d_1, \ldots, d_N) \in \Sigma \times \mathbb{D}^N$, let *label*$(e)$ denote the element A and let *data*(e) denote the vector $(d_1, \ldots, d_N)$. For $k \in \{1, \ldots, N\}$, $data_k(e)$ denotes the elements d_k of e. These notations are generalized in the obvious manner to words over $\Sigma \times \mathbb{D}^N$.

2.2 A first-order logic over data words

We introduce herefater the *data word logic* (DWL for short) which is a first order logic allowing to reason about data words by considering the labels as well as the data values at each of their positions. The logic DWL is parameterized by a (first-order) logic on the considered data domain $\mathbb{D}$, i.e., by the set of operations and the set of basic predicates (relations) allowed on elements of $\mathbb{D}$.

Let Ω be a finite set of functions over $\mathbb{D}$, and let Ξ be a finite set of relations over $\mathbb{D}$. Consider also a set of *position variables* $\mathcal{I}$ ranging over positive integers and a set of *data variables* $\mathcal{D}$ ranging over data values in $\mathbb{D}$, and assume that $\mathcal{I} \cap \mathcal{D} = \emptyset$. Then, the set of *terms* of $\mathsf{DWL}(\mathbb{D}, \Omega, \Xi)$ is given by the grammar:

$$t ::= u \mid \delta_k[x] \mid o(t_1, \ldots, t_n)$$

where $k \in \{1, \ldots, N\}$, $x \in \mathcal{I}$, $u \in \mathcal{D}$, and $o \in \Omega$. The set of *formulas* of $\mathsf{DWL}(\mathbb{D}, \Omega, \Xi)$ is given by:

$$\varphi ::= 0 < x \mid x < y \mid A[x] \mid r(t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists u.\ \varphi \mid \exists x.\ \varphi$$

where $x, y \in \mathcal{I}$, $u \in \mathcal{D}$, $A \in \Sigma$, and $r \in \Xi$.

As usual, boolean connectives such as conjunction $\wedge$ and implication $\Rightarrow$ are defined in terms of disjunction $\vee$ and negation $\neg$, and universal quantification $\forall$ is defined as the dual of existential quantification $\exists$. We also define equality $=$ and disequality $\neq$ in terms of $<$ and boolean connectives. Let $x = 0$ be an abbreviation of $\neg(0 < x)$ and let $x = y$ be an abbreviation of $\neg(x < y) \wedge \neg(y < x)$. We also write as usual $t \leq t'$ for $t < t' \vee t = t'$, where t and t' represent either position variables or 0. Then, let $t \neq t'$ be an abbreviation of $\neg(t = t')$, for $t, t' \in \mathcal{I} \cup \{0\}$. We denote by $\mathsf{DWL}_=$ the set of DWL formulas where the only comparisons constraints between position variables, and between position variables and 0 are equality or disequality constraints.

The notions of bound and free variables are defined as usual in first-order logic. Given a formula φ, the set of free variables in φ is denoted $FV(\varphi)$.

Formulas are interpreted on finite or infinite words over the alphabet $\Sigma \times \mathbb{D}^N$. Intuitively, position variables correspond to positions in the considered word. The formula $A[x]$ is true if A is the label of the element at the position corresponding to the position variable x. The term $\delta_k[x]$ represents the k^{th} data value attached to the element at the position corresponding to x. Terms are built from such data values and from data variables by applying operations in Ω. Formulas of the form $r(t_1, \ldots, t_n)$ allow to express constraints on data values at different positions of the word.

Formally, we define a satisfaction relation between such models and formulas. Let $\sigma \in (\Sigma \times \mathbb{D}^N)^\infty$. In order to interpret open formulas, we need valuations of position and data variables. Given $\mu : \mathcal{I} \to \mathbb{N}$ and $\nu : \mathcal{D} \to \mathbb{D}$, the satisfaction relation is inductively defined as follows:

$$
\begin{aligned}
\sigma \models_{\mu,\nu} 0 < x &\text{ iff } 0 < \mu(x) \\
\sigma \models_{\mu,\nu} x < y &\text{ iff } \mu(x) < \mu(y) \\
\sigma \models_{\mu,\nu} A[x] &\text{ iff } \mathit{label}(\sigma(\mu(x))) = A \\
\sigma \models_{\mu,\nu} r(t_1, \dots, t_m) &\text{ iff } r(\langle t_1 \rangle_{\sigma,\mu,\nu}, \dots, \langle t_m \rangle_{\sigma,\mu,\nu}) \\
\sigma \models_{\mu,\nu} \neg\varphi &\text{ iff } \sigma \not\models_{\mu,\nu} \varphi \\
\sigma \models_{\mu,\nu} \varphi_1 \vee \varphi_2 &\text{ iff } \sigma \models_{\mu,\nu} \varphi_1 \text{ or } \sigma \models_{\mu,\nu} \varphi_2 \\
\sigma \models_{\mu,\nu} \exists u.\ \varphi &\text{ iff } \exists d \in \mathbb{D}.\ \sigma \models_{\mu,\nu[u \leftarrow d]} \varphi \\
\sigma \models_{\mu,\nu} \exists x.\ \varphi &\text{ iff } \exists i \in \mathbb{N}.\ i < |\sigma| \text{ and } \sigma \models_{\mu[x \leftarrow i],\nu} \varphi
\end{aligned}
$$

where the mapping $\langle \cdot \rangle_{\sigma,\mu,\nu}$, associating to each term a data value, is inductively defined as follows:

$$
\begin{aligned}
\langle u \rangle_{\sigma,\mu,\nu} &= \nu(u) \\
\langle \delta_k[x] \rangle_{\sigma,\mu,\nu} &= \mathit{data}_k(\sigma(\mu(x))) \\
\langle o(t_1, \dots, t_n) \rangle_{\sigma,\mu,\nu} &= o(\langle t_1 \rangle_{\sigma,\mu,\nu}, \dots, \langle t_n \rangle_{\sigma,\mu,\nu})
\end{aligned}
$$

Given a formula φ, let $[\![\varphi]\!]_{\mu,\nu} = \{\sigma \in (\Sigma \times \mathbb{D}^N)^\infty \ :\ \sigma \models_{\mu,\nu} \varphi\}$. A formula φ is *satisfiable* if and only if there exist valuations μ and ν such that $[\![\varphi]\!]_{\mu,\nu} \neq \emptyset$. The subscripts of $\models$ and $[\![\cdot]\!]$ are omitted in the case of a closed formula.

2.3 Quantifier alternation hierarchy

A formula is in *prenex form* if it is written $Q_1 z_1 Q_2 z_2 \dots Q_m z_m.\ \varphi$ where (1) $Q_1, \dots, Q_m \in \{\exists, \forall\}$, (2) $z_1, \dots, z_m \in \mathcal{I} \cup \mathcal{D}$, and φ is a quantifier-free formula. It can be proved that for every formula φ, there exists an equivalent formula φ' in prenex form.

We consider two families $\{\Sigma_n\}_{n \geq 0}$ and $\{\Pi_n\}_{n \geq 0}$ of sets of formulas defined according to the alternation depth of existential and universal quantifiers in their prenex form:

- $\Sigma_0 = \Pi_0$ is the set of formulas where all quantified variables are in $\mathcal{D}$,
- For $n \geq 0$, Σ_{n+1} (resp. Π_{n+1}) is the set of formulas $Q z_1 \dots z_m.\ \varphi$ where $z_1, \dots, z_m \in \mathcal{I} \cup \mathcal{D}$, Q is the existential (resp. universal) quantifier $\exists$ (resp. $\forall$), and φ is a formula in Π_n (resp. Σ_n).

It can be seen 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}.

2.4 Data independent formulas

A DWL formula is *data independent* if it does not contain occurrences of data predicates of the form $r(t_1, \ldots, t_n)$ and of quantification over data variables. Syntactically, the set of data independent formulas is the same as the set of formulas of the monadic first-order logic over integers with the usual ordering relation. (Projections on the alphabet Σ of their models define star-free regular languages.) Interpreted over data words, these formulas satisfy the following closure properties: for every data words σ and σ', and for every data independent formula φ, if $label(\sigma) = label(\sigma')$, then $\sigma \models_{\mu,\nu} \varphi$ if and only if $\sigma' \models_{\mu,\nu} \varphi$.

3 The satisfiability problem

We investigate in this section the decidability of the satisfiability problem of DWL. First, we can prove that the logic is undecidable for very simple data theories starting from the fragment Π_2. The proof 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, as a data word. Each position corresponds to a cell in the tape of the machine at some configuration in the computation. We associate to each position (1) a positive integer value corresponding to its rank in a configuration, and (2) a label encoding informations (ranging over a finite domain) such as the contents of the cell, the fact that a cell corresponds to the location of the head, and the control state of the machine. Then, using DWL formulas in the Π_2 (i.e., $\forall^*\exists^*$) fragment, 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. We need for that to use the ordering on positions to talk about successive configurations, and the equality on the values attached to positions to relate cells with the same rank in these successive configurations. For the logic $\mathsf{DWL}_=$, since we do not have an ordering on positions, we need to attach another value to position representing their rank in the configurations.

Theorem 1. *The satisfiability problem of the fragment Π_2 is undecidable for $\mathsf{DWL}(\mathbb{N}, =)$ and $\mathsf{DWL}_=(\mathbb{N}, 0, <)$.*

Then, the main result of this section is that whenever the underlying theory on data has a decidable satisfiability problem, the fragment Σ_2 has also a decidable satisfiability problem.

Theorem 2. *If the satisfiability problem for $\mathsf{FO}(\mathbb{D}, \Omega, \Xi)$ is decidable, then the satisfiability problem of the fragment Σ_2 of $\mathsf{DWL}(\mathbb{D}, \Omega, \Xi)$ is also decidable.*

The rest of the section is devoted to the proof of theorem above. We show that the satisfiability problem in the fragment Σ_2 of $\mathsf{DWL}(\mathbb{D}, \Omega, \Xi)$ can be reduced to the satisfiability problem in logic on data $\mathsf{FO}(\mathbb{D}, \Omega, \Xi)$.

First of all, we need to introduce a slight modification in the definition of data words: So far, we have considered that a data word σ is total mappings from the interval $[0, |\sigma|)$ to the alphabet $\Sigma \times \mathbb{D}^N$. Let us consider now that a word σ is a total mapping from a set of natural numbers S_σ to the alphabet $\Sigma \times \mathbb{D}^N$, where S_σ is not necessarily an interval. Clearly, there is an isomorphism π_σ from $[0, |\sigma|)$ to S_σ, and this isomorphism is monotonic. Let us denote $[\sigma]$, for every word σ, the (unique) mapping from $[0, |\sigma|)$ to $\Sigma \times \mathbb{D}^N$ such that, for every $i \in [0, |\sigma|)$, $[\sigma](i) = \sigma(\pi_\sigma(i))$.

Furthermore, assume that in the definition of the satisfaction relation $\models$ between data words and DWL formulas, the last line (the case of existential quantification over position variables) is substituted by: $\sigma \models_{\mu,\nu} \varphi$ iff $\exists i \in S_\sigma .\ \sigma \models_{\mu[x \leftarrow i],\nu} \varphi$. Then, it can be checked that the following holds.

Lemma 1. *For every data word σ, for every* DWL *formula φ, and for every position/data variable valuations μ and ν, we have $\sigma \models_{\mu,\nu} \varphi$ iff $[\sigma] \models_{\mu,\nu} \varphi$.*

The lemma above implies that, for every two data words σ and σ' such that $[\sigma] = [\sigma']$, we have $\sigma \models_{\mu,\nu} \varphi$ iff $\sigma' \models_{\mu,\nu} \varphi$, for every φ, μ, and ν.

Before starting the proof, we need to introduce a syntactical form of Σ_n formulas, for any $n \geq 1$. We say that a formula in such a fragment is in *special form* if it is a finite disjunction of formulas of the form

$$\exists x_1, \ldots, x_n \exists \mathbf{u} \forall \mathbf{y}.\ \big((\bigwedge_{1 \leq i < j \leq n} x_i < x_j) \wedge \varphi \big)$$

where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y}$ are position variables, and $\mathbf{u}$ is a vector of data variables. It is easy to show that every formula in the fragment Σ_n has an equivalent Σ_n formula in special form.

We are now ready to stat the proof of Theorem 2. Let φ be a DWL formula, and assume w.l.o.g. that φ is closed, in special form, and given by:

$$\varphi = \exists \mathbf{x}.\ \exists \mathbf{u}.\ \forall \mathbf{y}.\ \psi$$

where $\mathbf{x}$ and $\mathbf{y}$ are vectors of position variables, $\mathbf{u}$ is a vector of data variables. Assume also that φ is satisfiable, which means that there is a data word σ such that $\sigma \models \varphi$.

Then, let Θ be the set of all possible (partial or total) mappings between the variables in $\mathbf{y}$ and the variables in $\mathbf{x}$. Then, we have $\sigma \models \exists \mathbf{x}.\ \exists \mathbf{u}.\ \varphi^{(1)}$ where

$$\varphi^{(1)} = \bigwedge_{\theta \in \Theta} \forall \mathbf{y}.\ \Big(\big((\bigwedge_{y \in dom(\theta)} y = \theta(y)) \wedge (\bigwedge_{y \notin dom(\theta)} \bigwedge_{x \in \mathbf{x}} y \neq x) \big) \Rightarrow \psi \Big) \qquad (1)$$

This means that there are positions $\mathbf{i}$ in the domain of σ, and there are data values $\mathbf{d}$, such that

$$\sigma \models_{\mu,\nu} \varphi^{(1)} \qquad (2)$$

where μ and ν are valuations associating $\mathbf{i}$ with $\mathbf{x}$ and $\mathbf{d}$ with $\mathbf{u}$, respectively.

Consider now the data word $\sigma' = \sigma|_{\mathbf{i}}$, i.e., the subword of σ corresponding to the positions in $\mathbf{i}$. Then, it can be seen that (2) implies that:

$$\sigma' \models_{\mu,\nu} \bigwedge_{\substack{\theta\in\Theta\\ dom(\theta)=\mathbf{y}}} \forall \mathbf{y}.\ \big(\bigwedge_{y\in\mathbf{y}} y = \theta(y) \Rightarrow \psi\big) \tag{3}$$

which is equivalent to $\sigma' \models \varphi^{(2)}$ where

$$\varphi^{(2)} = \exists \mathbf{x}.\ \exists \mathbf{u}.\ \bigwedge_{\substack{\theta\in\Theta\\ dom(\theta)=\mathbf{y}}} \bigwedge_{y\in\mathbf{y}} \psi[\theta(y)/y] \tag{4}$$

Conversely, every minimal model (according to the size of its domain) of the formula $\varphi^{(2)}$ above is necessarily a model of the formula $\exists\mathbf{x}.\ \exists\mathbf{u}.\ \varphi^{(1)}$, which is equivalent to the formula φ. Therefore, we have reduced the satisfiability problem of Σ_2 to the satisfiability problem in Σ_1.

The last step of the proof is to reduce the satisfiability problem of the Σ_1 formula $\varphi^{(2)}$ to the satisfiability problem of a pure data formula in $\mathsf{FO}(\mathbb{D}, \Omega, \Xi)$. For that, we must get rid of the comparisons between position variables, and of constraints on position labels.

Since the formula φ in special form, the values associated with the position variables $\mathbf{x}$ are in the same order as their indices. Then, let $\varphi^{(3)} = \exists\mathbf{x}.\ \exists\mathbf{u}.\ \psi'$ be the formula obtained from $\varphi^{(2)}$ by replacing each constraint $x_i < x_j$ by *true* if $i < j$, or by *false* otherwise. The formula $\varphi^{(3)}$ is equivalent to the formula $\varphi^{(2)}$ but has no comparison constraints between position variables. Now, since the alphabet Σ is finite, we can build an equivalent formula to $\varphi^{(3)}$ which has no label constraints: we consider a disjunction on all possible mappings λ from $\mathbf{x}$ to Σ. For each of these mapping λ, we replace in $\varphi^{(3)}$ each occurrence of a formula $A[x]$ by *true* if $\lambda(x) = A$, or by *false* otherwise. Let $\varphi^{(4)}$ be the so obtained formula.

Finally, we define a $\mathsf{FO}(\mathbb{D}, \Omega, \Xi)$ formula which is satisfiable if and only if $\varphi^{(4)}$ is satisfiable. This formula is obtained by replacing in $\varphi^{(4)}$ terms involving positions variables by data variables: for each variable $x \in \mathbf{x}$ and for each rank $k \in \{1, \ldots, N\}$, we associate a fresh data variable $v_{x,k}$. Then, we remove in $\varphi^{(4)}$ the quantification of $\mathbf{x}$ and we substitute each occurrence of a term $\delta_k(x)$ by the variable $v_{x,k}$.

4 Rewriting systems over data words

4.1 Rewriting rules

A *data word rewriting rule* over the logic DWL has the form:

$$A_0 \cdots A_n \mapsto B_0 \cdots B_m \quad : \quad \varphi$$

where $A_i, B_j \in \Sigma$ for all $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, m\}$, and φ is a DWL formula such that (1) $FV(\varphi) = \{x_0, \ldots, x_n\} \cup \{y_0, \ldots, y_m\}$, and (2) all the occurrences in φ of the variables y_j are in terms of the form $\delta_k(y_j)$ for $0 \leq k \leq N$. We assume in this definition that the left hand side of a rule has at least one symbol ($n \geq 0$), and that its right hand side can be empty. When $B_0 \cdots B_m$ is empty, the formula φ has only free variables $\{x_0, \ldots, x_n\}$ related to the left hand side of the rule.

Intuitively, the application of a rewriting rule to a data word σ (leading to a new word σ') consists in replacing in σ a subword γ such that $label(\gamma)$ is equal to $A_0 \cdots A_n$ by another word γ' such that $label(\gamma')$ is equal to $B_0 \cdots B_m$, provided that the formula φ, relating the data values in σ with data values in γ', is satisfied. Each variable x_k (resp. y_k) represents the position in σ (resp. σ') of the k^{th} elements of γ (resp. γ'). The formula φ can constrain the positions corresponding to elements of γ as well as their attached data values w.r.t. data values at other position in σ. Moreover, the formula φ can constrain the data values in the new word by relating these data values with data values attached to positions in σ.

4.2 Rewriting semantics

A *rewriting system* is given by a set of rewrite rules and a rewriting semantics. Several rewriting relations between words can be considered depending on the adopted semantics of rewriting. Given a set Δ of data word rewriting rules, we consider here four relations $\Rightarrow_{\Delta,f}$, $\Rightarrow_{\Delta,p}$, and $\Rightarrow_{\Delta,m}$ corresponding respectively to factor, prefix, and multiset rewriting. Subscripts are omitted whenever the considered rewriting system and/or rewriting semantics are know from the context.

Let us start by defining the semantics of factor and prefix rewriting. For that, let us fix a rewrite system Δ. Then, for every $\sigma, \sigma' \in (\Sigma \times \mathbb{D}^N)^*$, we have $\sigma \Rightarrow_f \sigma'$ (resp. $\sigma \Rightarrow_p \sigma'$) if and only if there exists a rewrite rule "$A_0 \cdots A_n \mapsto B_0 \cdots B_m \ : \ \varphi$" and there exist data words $\alpha, \beta, \gamma, \gamma' \in (\Sigma \times \mathbb{D}^N)^*$ such that

- *factor rewriting*: $\sigma = \alpha\gamma\beta$ and $\sigma' = \alpha\gamma'\beta$,
- *prefix rewriting*: $\sigma = \gamma\beta$, $\sigma' = \gamma'\beta$, and $|\alpha| = 0$,

with $label(\gamma) = A_0 \cdots A_n$, $label(\gamma') = B_0 \cdots B_m$ and

$$\sigma \models \varphi[(|\alpha| + i)/x_i]_{0 \leq i \leq n}[data_k(\gamma'(j))/\delta_k[y_j]]_{0 \leq k \leq N, 0 \leq j \leq m}$$

Now, in order to define the multiset rewriting relation, we consider the equivalence relation between words which abstracts away the ordering between symbols: Given $\sigma, \sigma' \in (\Sigma \times \mathbb{D}^N)^*$, we have $\sigma \simeq \sigma'$ if and only if there exists a permutation π of $\{0, \ldots, |\sigma| - 1\}$ such that $\sigma(\pi(0)) \cdots \sigma(\pi(|\sigma| - 1)) = \sigma'$. Then, for every $\sigma, \sigma' \in (\Sigma \times \mathbb{D}^N)^*$, we have $\sigma \Rightarrow_m \sigma'$ if and only if $\exists \theta, \theta' \in (\Sigma \times \mathbb{D}^N)^*$ such that $\sigma \simeq \theta$, $\theta \Rightarrow_f \theta'$, and $\theta' \simeq \sigma'$.

It can be seen that for every σ, σ' such that $\sigma \simeq \sigma'$, and for every formula φ in $\mathsf{DWL}_=$, we have $\sigma \models \varphi$ if and only if $\sigma' \models \varphi$. This fact is not true in general for

DWL formulas. Therefore, in the case of multiset rewriting, we assume naturally that all the constraints in the rewriting rules are in $\mathsf{DWL}_=$.

Given a set of rewriting rules Δ, the corresponding factor, prefix, and multiset rewriting system are denoted Δ_f, Δ_p, and Δ_m, respectively. Let $\mathsf{DWRS}_\sharp$ be the class of all $\sharp$-rewriting systems, for $\sharp \in \{f, p, m\}$.

5 Models of infinite-state systems

5.1 Recursive programs with data

We show hereafter that sequential programs with recursive procedure calls can be translated into prefix rewriting systems. We consider that a program has several procedures, and we assume that it uses a set of global variables $\mathbf{g} = (g_1, \ldots, g_N)$ and that each procedure has a set of local variables $\mathbf{l} = (l_1, \ldots, l_M)$. (We assume w.l.o.g. that the local variables are the same for all procedures, all of them ranging over some data domain $\mathbb{D}$.)

A program is given by its inter-procedural control flow graphs (ICFG for short) which is a collection of control flow graphs (CFG), one for each of its procedures. Nodes in the CFG of a procedure represent control points in its source code, and edges represent transitions from a control point to another one. We assume that each procedure Π has an initial node n_{in}^{Π}. Edges in CFGs are labeled by statements which can be either (1) tests over the values of the global/local variables, (2) assignments of the global/local variables, (3) procedure calls, or (4) procedure returns leading to a termination control point. Variables are assigned values of expressions built from global and local variables using a set of operations Ω. Tests over variables are first-order assertions based on a set of predicates Ξ.

Consider an ICFG, and let $\mathcal{N}$ be the set of its nodes. We associate with the considered ICFG a prefix rewriting systems over the alphabet $(\mathcal{N} \cup \{G\}) \times \mathbb{D}^{N+M}$ where G is a special symbol, N is the number of global variables, and M is the number of local variables of each procedure. Indeed, we consider that a configuration of the recursive program defined by the ICFG is represented by a finite word of the form $(G, \mathbf{d_0})(n_1, \mathbf{d_1})(n_2, \mathbf{d_2}) \cdots (n_\ell, \mathbf{d}_\ell)$ where $n_i \in \mathcal{N}$ for all $i \geq 1$ and $\mathbf{d_i} \in \mathbb{D}^{N+M}$ for all $i \geq 0$. The element $(G, \mathbf{d_0})$ at position 0 of the word is used to store the value of the global variables: we assume that for every $k \in \{1, \ldots, N\}$, the value of the variable g_k is equal to the k^{th} element of the vector $\mathbf{d_0}$. Moreover, the rest of the word $(n_1, \mathbf{d_1})(n_2, \mathbf{d_2}) \cdots (n_\ell, \mathbf{d}_\ell)$ represents the call stack of the program. In the element $(n_i, \mathbf{d_i})$ of this stack, n_i represents the point at which the control of the program will return after all the calls higher in stack (i.e., of index less than i in our word representation) will be done, and $\mathbf{d_i}$ represents the values of the local variables which must be restored when the control will reach the point n_i: we assume that for every $k \in \{N+1, \ldots, N+M\}$, the value of the variable l_k is equal to the k^{th} element of the vector $\mathbf{d_i}$. Then, the set of rewriting rules of the system associated with the considered ICFG is defined as follows:

Test: $n \xrightarrow{\varphi(\mathbf{g},\mathbf{l})} n'$ where φ is a $\mathsf{FO}(\mathbb{D}, \Omega, \Xi)$ formula, is modeled by:

$$Gn \mapsto Gn' \;:\; \varphi\zeta \wedge \varphi_{id}$$

where ζ is the substitution $[\delta_k[x_0]/g_k]_{1 \leq k \leq N}[\delta_k[x_1]/l_k]_{N+1 \leq k \leq N+M}$, and

$$\varphi_{id} = \bigwedge_{i=1}^{N} \bigwedge_{j=N+1}^{N+M} \delta_i[y_0] = \delta_i[x_0] \wedge \delta_j[y_1] = \delta_j[x_1]$$

Assignment: $n \xrightarrow{(\mathbf{g},\mathbf{l}):=\mathbf{t}(\mathbf{g},\mathbf{l})} n'$ where $\mathbf{t}$ is a vector of Ω-terms, is modeled by:

$$Gn \mapsto Gn' \;:\; \bigwedge_{i=1}^{N} \bigwedge_{j=N+1}^{N+M} \delta_i[y_0] = t_i\zeta \wedge \delta_j[y_1] = t_j\zeta$$

where ζ is the substitution defined in the previous case.

Procedure call: $n \xrightarrow{\mathsf{call}(\Pi)} n'$ is modeled by:

$$Gn \mapsto Gn_{in}^{\Pi}n' \;:\; \varphi'_{id}$$

where

$$\varphi'_{id} = \bigwedge_{i=1}^{N} \bigwedge_{j=N+1}^{N+M} \delta_i[y_0] = \delta_i[x_0] \wedge \delta_j[y_2] = \delta_j[x_1]$$

Procedure return: $n \xrightarrow{\mathsf{return}} n'$ is modeled by:

$$Gn \mapsto G \;:\; \bigwedge_{i=1}^{N} \delta_i[y_0] = \delta_i[x_0]$$

More general prefix rewriting systems can be used in order to handle applications where stack inspection is needed. Indeed, the side constraints we allow in the rewriting rules can be used for the expression of global conditions on the stack content that must be satisfied before the execution of certain actions. This is important for modeling various control access and resource-usage scenarios. For instance, operations on security-critical objects can be executed only if certain conditions are satisfied, e.g., (1) all procedures in the call stack have a certain permission, or (2) a "privileged" procedure is present in the call stack and all procedures higher in the stack have a permission. These constraints can be expressed as DWL (data independent) formulas in the fragment Σ_2:

$$\forall x.\ (x_1 \leq x \Rightarrow \mathtt{perm}[x])$$

$$\exists x.\ \big(x_1 \leq x \wedge \mathtt{privilege}[x] \wedge \forall y.\ ((x_1 \leq y \wedge y < x) \Rightarrow \mathtt{perm}[y])\big)$$

5.2 Dynamic/parametrized networks of processes

Unbounded networks of identical processes can be modeled using rewriting systems. We assume that each process is defined by an extended automaton, i.e., a finite-control machine manipulating a set of variables $\mathbf{v} = (v_1, \ldots, v_N)$ ranging over some given data domain $\mathbb{D}$. More precisely, an extended automaton is defined by a finite set of control locations $\mathcal{Q}$, and a set of transitions between these locations. Each transition is labeled by a statement which can be either a test over the values of the variables, or an assignments of the variables. As in section 5.1, assigned values to variables are defined using expressions built from variables and a set of operations Ω, and tests are first-order assertions based on a set of predicates Ξ.

Consider a network of n processes, where n is an arbitrary positive integer (greater than 1). We represent a configuration of such a network by a word of length n over the alphabet $\mathcal{Q} \times \mathbb{D}^N$. Then, to reason uniformly about networks with an arbitrary number of processes, (1) we consider the set of all finite words over $\mathcal{Q} \times \mathbb{D}^N$ as possible configurations, and (2) we model the dynamics of the whole family of networks with an arbitrary size by means of a rewriting system. We use different rewriting semantics depending on the topology of the network. In general, using factor rewriting systems allows to reason about networks with a linear topology, i.e., where processes are arranged sequentially (or sometimes as a ring). This corresponds to the case where an ordering is assumed between the process identities (inducing a notion of neighborhood). Multiset rewriting systems are used when the ordering between process is not relevant. This is the case of many systems such as cache coherence protocols [23] and some classes of multithreaded programs [22, 16].

Data rewriting systems we consider allow to model various communication (and synchronization) schemas between processes (e.g., shared variables, rendezvous), tests on local and global configurations, as well as dynamic creation and deletion of processes.

As an example, we give hereafter the model corresponding to (a simplified version of) the Lamport's Bakery protocol for mutual exclusion. As usual in such protocols, the algorithms handle a set of processes which compete for entering into a critical section. The model of each process is a machine with tree control locations: `nocs`, `req`, and `cs`. The location `nocs` correspond to activities of the processes outside the critical section. When the process needs to enter the critical section, it takes a ticket with a number (a positive integer) which is bigger that the number of all existing tickets, and moves to the control location `req`. Then, the process waits at this location for his turn to enter the critical section, that is, until the number on its tickets become the smallest of all numbers on existing tickets. In case of a conflict (since it may happen actually that two processes obtain the same ticket number), the process with the smallest rank (identity) enters the critical section. Then, the process can exit the critical section and return to the control location `nocs`.

The Bakery protocol can be modeled by the following factor rewriting system Δ_{bakery} defined over the alphabet $\{\mathtt{nocs}, \mathtt{req}, \mathtt{cs}\} \times \mathbb{N}$.

$$\begin{aligned}
\mathtt{nocs} \mapsto \mathtt{req} &: \forall i.\ \delta[y_0] > \delta[i] \\
\mathtt{req} \mapsto \mathtt{cs} &: \forall i.\ \big(\delta[i] > 0 \Rightarrow (\delta[x_0] < \delta[i] \ \vee\ \delta[x_0] = \delta[i] \wedge x_0 < i)\big) \wedge \\
&\quad\ \delta[y_0] = \delta[x_0] \\
\mathtt{cs} \mapsto \mathtt{nocs} &: \delta[y_0] = 0
\end{aligned}$$

Notice that Δ_{bakery} is a system of the class $\mathsf{DWRS}_f[\Pi_1]$ since all side constraints in the rule are universally quantified formulas.

6 Post and pre condition reasoning

We address in this section the problem of checking the validity of assertions on the configurations of systems modeled by data word rewriting systems. We show that the fragment Σ_2 of DWL is effectively closed under the computation of one (forward or backward) rewriting step of rewriting systems in $\mathsf{DWRS}[\Sigma_2]$ (for the three considered ewriting semantics). We show how to use this result in checking inductive invariance of given assertions, and for carrying out Hoare-style reasoning about our models.

6.1 post and pre operators

We define hereafter the operators of immediate successors and immediate predecessors. Let Δ be a set of data word rewriting rules over the alphabet $\Sigma \times \mathbb{D}^N$. Then, for every finite data word $\sigma \in (\Sigma \times \mathbb{D}^N)^*$, we define, for any $\sharp \in \{f, p, m\}$:

$$\begin{aligned}
\mathsf{post}_{\Delta,\sharp}(\sigma) &= \{\sigma' \in (\Sigma \times \mathbb{D}^N)^* \ :\ \sigma \Rightarrow_{\Delta,\sharp} \sigma'\} \\
\mathsf{pre}_{\Delta,\sharp}(\sigma) &= \{\sigma' \in (\Sigma \times \mathbb{D}^N)^* \ :\ \sigma' \Rightarrow_{\Delta,\sharp} \sigma\}
\end{aligned}$$

representing, respectively, the set of immediate successors and predecessors of σ in the rewrite system $\Delta_\sharp$. Then, let $\mathsf{post}^*_{\Delta,\sharp}$ and $\mathsf{pre}^*_{\Delta,\sharp}$ be the reflexive-transitive closure of $\mathsf{post}_{\Delta,\sharp}$ and $\mathsf{pre}_{\Delta,\sharp}$ respectively, i.e., $\mathsf{post}^*_{\Delta,\sharp}(\sigma)$ (resp. $\mathsf{pre}^*_{\Delta,\sharp}(\sigma)$) is the set of all successors (resp. predecessors) of σ in $\Delta_\sharp$. These definitions can be generalized straightforwardly to sets of words.

6.2 Computing post and pre images

The main result of this section is the following:

Theorem 3. *Let $\Delta_\sharp$ be a rewriting system in $\mathsf{DWRS}_\sharp[\Sigma_n]$, for $\sharp \in \{f, p, m\}$ and $n \geq 2$. Then, for every DWL closed formula φ in the fragment Σ_n, the sets $\mathsf{post}_{\Delta,\sharp}([\![\varphi]\!])$ and $\mathsf{pre}_{\Delta,\sharp}([\![\varphi]\!])$ are effectively definable by DWL formulas in the same fragment Σ_n.*

The rest of the section is devoted to the proof of the theorem above. Let us consider first the problem of computing post images in the case of a factor rewriting system.

Let $\exists\mathbf{z}.\ \phi$ be a formula in $\Sigma_{\geq 2}$, and let $\tau = A_0 \ldots A_n \mapsto B_0 \ldots B_m\ :\ \varphi(\mathbf{x}, \mathbf{y})$ be a data rewriting rule, with $\mathbf{x} = \{x_0, \ldots, x_n\}$ and $\mathbf{y} = \{y_0, \ldots, y_m\}$. We suppose w.l.o.g. that the sets of variables $\mathbf{x}, \mathbf{y}$, and $\mathbf{z}$ are disjoint.

By definition of the factor rewriting semantics, the positions associated with the variables $\mathbf{x}$ are consecutive and correspond to a factor $A_0 \ldots A_n$ in the rewritten word. We strengthen the constraint φ of the rule τ in order to make this fact explicit. Then, we define the formula

$$\varphi^{(1)} = \varphi \wedge \big(\bigwedge_{i \in [0,n-1]} \neg(\exists t.\ x_i < t < x_{i+1}) \big) \wedge \bigwedge_{i \in [0,n]} A_i[x_i]$$

By definition of data word rewriting systems, all the occurrences of positions variables $\mathbf{y}$ in the constraint φ are used in terms of the form $\delta_k[y]$. Then, we can eliminate all occurrences of all variables in $\mathbf{y}$ by replacing each $\delta_k[y]$ in $\varphi^{(1)}$ by a fresh data variable in a vector $\mathbf{v}$. Let $\xi : \mathbf{y} \times [1, N] \to \mathbf{v}$ be the bijective mapping such that $\delta_k[y]$ is replaced by $\xi(y, k)$. We define:

$$\varphi^{(2)} = \varphi^{(1)}[\delta_k[y] \leftarrow \xi(y, k)]_{y \in \mathbf{y}, k \in [1,N]}$$

Then, the rule τ can be applied only on words satisfying

$$\exists\mathbf{z}.\ \phi \wedge \exists\mathbf{x}.\ \exists\mathbf{v}.\ \varphi^{(2)} \tag{5}$$

This formula could be written in special form: For every vector $\mathbf{t}$ of (fresh) position variables such that $|\mathbf{x}| \leq |\mathbf{t}| \leq |\mathbf{x}| + |\mathbf{z}|$, consider the formula

$$\bigvee_{\theta \in \Theta} \exists\mathbf{t}.\ \exists\mathbf{v}.\ (\bigwedge_{t_i, t_j \in \mathbf{t}, i<j} t_i < t_j) \wedge (\phi \wedge \varphi^{(2)})[\mathbf{x} \leftarrow \theta(\mathbf{x}), \mathbf{z} \leftarrow \theta(\mathbf{z})] \tag{6}$$

where Θ is the set of all total mappings from $\mathbf{x} \cup \mathbf{z}$ to $\mathbf{t}$. Then, the formula (5) is equivalent to the disjunction of all the formulas (6) for all the possible vectors $\mathbf{t}$ defined as above. Let us focus in the sequel on one disjunct of the resulting formula. Then, consider that such a disjunct is the formula:

$$\psi = \exists t_1 \ldots t_{p-1} \exists t_p \ldots t_{p+n} \exists t_{p+n+1} \ldots t_q \exists\mathbf{v}.\ \phi^{(1)}$$

with $\forall i \in [0, n], \theta(x_i) = t_{p+i}$, $p \geq 1, p + n \leq q$.

Let σ be a model of ψ. By definition of factor rewriting, the rule τ eliminates from σ the factor corresponding to the position associated with the variables $t_p..t_{p+n}$, and insert at position t_p a new word of length m (labeled $B_0 \cdots B_m$). By Lemma 1, we can assume that the distance between the positions corresponding to t_p and t_{p+n+1} in σ is at least $m + 1$. Therefore, there is enough room for inserting new positions in σ corresponding to the right hand the rule. These positions will be associated with $\mathbf{y}$.

The formula $\phi^{(2)}$ below gives the constraints on positions and labels resulting from the insertion of right hand side of the rule τ:

$$\phi^{(2)} = \Big(\bigwedge_{i \in [0,m]} B_i(y_i) \Big) \wedge t_{p-1} < y_0 \wedge y_m < t_{p+n+1}$$
$$\wedge \Big(\bigwedge_{\substack{i,j \in [0,m] \\ i<j}} y_i < y_j \wedge \neg(\exists x.\ y_i < x < y_j) \Big) \wedge \Big(\bigwedge_{\substack{k \in [1,N] \\ y \in \mathbf{y}}} \delta_k(y) = \xi(y,k) \Big)$$

Let $\mathbf{w}$ be a new data variable vector of length $n \cdot N$, and let η be a bijective mapping from $\{t_p, \ldots, t_{p+n}\} \times [1, N]$ to $\mathbf{w}$. (We use the mapping η for substituting occurrences of terms $\delta_k[x]$ in $\phi^{(1)}$ by fresh data variables.)

Then, the formula corresponding to $\mathsf{post}_{\tau,f}([\![\psi]\!])$ is given by:

$$\exists y_1 \ldots y_m \exists \mathbf{w} \exists t_1 \ldots t_{p-1} \exists t_{p+n+1} \ldots t_q \exists \mathbf{v}.\ \phi^{(3)} \wedge \phi^{(2)}$$

where the formula $\phi^{(3)}$ is the result of the application to $\phi^{(1)}$ of a transformation $\ominus$ defined inductively in Table 6.2:

$$\phi^{(3)} = \phi^{(1)} \ominus (t_p \ldots t_{p+n}, y_0 \ldots y_m, \mathtt{lab}, \eta, \{t_1, \ldots, t_{p-1}\}, \{t_{p+n+1}, \ldots, t_q\})$$

where for all $i \in [0, n]$, $\mathtt{lab}(t_{p+i}) = A_i$.

The first parameter of the operator $\ominus$, called $\mathbf{x}$, is a set of position variables that are deleted. The second parameter, called $\mathbf{y}$, is a set of position variables that are not concerned by the constraint. The third parameter of $\ominus$, the mapping $\mathtt{lab}$, associates with position variables in $\mathbf{x}$ their label in $A_0, \ldots, A_n$. The fourth parameter, η, associates with each position variable $x \in \mathbf{x}$ and each integer $k \in [1, N]$ a variable $\eta(x, k)$ in $\mathbf{v}$. The last parameters, *Inf* and *Sup*, are sets of position variables which are ordered, by the context, before resp. after the variables in $\mathbf{x}$. Intuitively $\ominus$ deletes from a formula all occurences of the variables in $\mathbf{x}$ and all constraints concerning them and preserves all constraints concerning the rest of the configuration.

Notice that the obtained formula remains in the same fragment as the original formula since only a prefix of existential quantification is added.

It is easy to adapt the construction above in order to deal with prefix rewriting or multiset rewriting semantics. Indeed, prefix rewriting is particular case where the rewriting position is always the position 0. For multiset rewriting, the construction is simplified since ordering constraints are not used (see [16]).

Finally, let us mention that it is possible to define a symmetrical (and very similar) construction for $\mathsf{pre}_{\tau,\varphi}$ images.

6.3 Application in verification

Invariance checking consists in deciding whether a given property (1) is satisfied by the set of initial configurations, and (2) is stable under the transition relation of a system. Formally, given a rewriting system Δ and a closed formula φ_{init} defining the set of initial configurations, we say that a closed formula φ

$$
\begin{aligned}
(0 < z) \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) &= \begin{cases} 0 < z & \text{if } z \in \mathit{Inf} \\ \text{false} & \text{if } z \in \mathit{Sup} \text{ or } z = x_i \in \mathbf{x} \text{ with } i > 0 \\ 0 < \mathrm{y}_0 & \text{if } z \text{ is } x_0 \end{cases} \\
(z < z') \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) &= \begin{cases} z < z' & \text{if } \ z, z' \in \mathit{Inf} \text{ or } z, z' \in \mathit{Sup} \\ \text{true} & \text{if } z \in \mathit{Inf} \text{ and } (z' \in \mathit{Sup} \text{ or } z' \in \mathbf{x}) \\ & \text{or } z \in \mathbf{x}\ z' \in \mathit{Sup} \\ & \text{or } z, z' = x_i, x_j \in \mathbf{x} \text{ with } i < j \\ \text{false} & \text{otherwise} \end{cases} \\
A[z] \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) &= \begin{cases} \text{true} & \text{if } z \in \mathbf{x} \text{ and } \mathtt{lab}(z) = A \\ \text{false} & \text{if } z \in \mathbf{x} \text{ and } \mathtt{lab}(z) \neq A \\ A[z] & \text{otherwise} \end{cases} \\
r(\ldots, t_i, \ldots) \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) &= r(\ldots, t_i[\delta_k(x) \leftarrow \eta(x,k)]_{x \in \mathbf{x}}, \ldots) \\
(\neg\varphi) \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) &= \neg(\varphi \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup})) \\
(\varphi_1 \vee \varphi_2) \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) &= \varphi_1 \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) \vee \\
&\quad\ \varphi_2 \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) \\
(\exists u.\ \varphi) \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) &= \exists u.\ (\varphi \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup})) \\
(\exists z.\ \varphi) \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup}) &= \exists z. \bigwedge_{y \in \mathbf{y}} (z \neq y) \wedge (\varphi \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf} \cup \{z\}, \mathit{Sup})) \vee \\
&\quad\ \exists z. \bigwedge_{y \in \mathbf{y}} (z \neq y) \wedge (\varphi \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup} \cup \{z\})) \vee \\
&\quad\ \bigvee_{x \in \mathbf{x}} \varphi[z \leftarrow x]) \ominus (\mathbf{x}, \mathbf{y}, \mathtt{lab}, \eta, \mathit{Inf}, \mathit{Sup})
\end{aligned}
$$

Table 1. The operation $\ominus$

is an *inductive invariant* of (Δ, φ_{in}) if and only if (1) $[\![\varphi_{init}]\!] \subseteq [\![\varphi]\!]$, and (2) $\mathsf{post}_\Delta([\![\varphi]\!]) \subseteq [\![\varphi]\!]$. Clearly, (1) is equivalent to $[\![\varphi_{init}]\!] \cap [\![\neg\varphi]\!] = \emptyset$, and (2) is equivalent to $\mathsf{post}_\Delta([\![\varphi]\!]) \cap [\![\neg\varphi]\!] = \emptyset$. (Notice that this fact is also equivalent to $[\![\varphi]\!] \cap \mathsf{pre}_\Delta([\![\neg\varphi]\!]) = \emptyset$.)

Corollary 1. *The problem whether a formula $\varphi \in B(\Sigma_1)$ is an inductive invariant of (Δ, φ_{init}), where $\Delta \in \mathsf{DWRS}[\Sigma_2]$ and $\varphi_{init} \in \Sigma_2$, is decidable.*

For example, consider the system $\Delta_{bakery} \in \mathsf{DWRS}_f[\Pi_1]$ introduced in section 5.2. To prove that mutual exclusion is ensured, we check that the formula $\varphi_{mutex} = \forall x, y.\ x \neq y \Rightarrow \neg(\mathtt{cs}[x] \wedge \mathtt{cs}[y])$ (i.e., it is impossible to have two different processes in the critical section simultaneously) is implied by an inductive invariant φ_{inv} of $(\Delta_{bakery}, \varphi_{init})$ where $\varphi_{init} = \forall x.\ \mathtt{nocs}[x]$ (i.e., all processes are idle). For that, we consider the formula

$$\begin{aligned}\varphi_{inv} = \forall x.\ \mathtt{cs}[x] \Rightarrow\\ \delta[x] \neq 0 \wedge \forall y.\ x \neq y \Rightarrow (\delta[y] = 0 \ \vee\ \delta[x] < \delta[y] \ \vee\ \delta[x] = \delta[y] \wedge x < y)\end{aligned}$$

Notice that all formulas φ_{mutex}, φ_{init}, and φ_{inv} are in the fragment Π_1. Then, the validity of $\varphi_{inv} \Rightarrow \varphi_{mutex}$ can be checked automatically by Theorem 2 since it is a $B(\Sigma_1)$ formula, and the inductive invariance of φ_{inv} for $(\Delta_{bakery}, \varphi_{init})$ can be decided by Corollary 1.

Hoare-style reasoning consists in, given two properties expressed by formulas φ_1 and φ_2, and a given set of rules Δ, deciding whether starting from configurations satisfying φ_1, the property φ_2 necessarily hold after the application of the rules in Δ. Formally, this consists in checking that $\mathsf{post}_\Delta([\![\varphi_1]\!]) \subseteq [\![\varphi_2]\!]$. In that case, we say that $(\varphi_1, \Delta, \varphi_2)$ constitutes a Hoare triple.

Corollary 2. *The problem whether $(\varphi_1, \Delta, \varphi_2)$ is a Hoare triple, where $\varphi_1 \in \Sigma_2$, $\Delta \in \mathsf{DWRS}[\Sigma_2]$ and $\varphi_2 \in \Pi_2$, is decidable.*

7 Reachability analysis for integer context-free systems

In this section, we show that for restricted word rewriting systems (called $\mathsf{CFS_{DL}}$), the reachability problem of sets described by data-independent formulas is decidable. We consider a class of context-free prefix rewriting rules with integer data and constraints in the difference logic. To show decidability, we use a slight generalization of $\mathbb{Z}$-input 1-counter machines introduced in [17] to represent set of finite data words (subsets of $(\Sigma \times \mathbb{Z})^*$). Then, we show that given $\mathsf{CFS_{DL}}$ Δ, and given a set of data words described by a $\mathbb{Z}$-input 1-counter machine M, it is possible to compute a machine M' representing the set of all reachable words (by the iterative application of rules in Δ). This allows then to prove decidability of the reachability problem for $\mathsf{CFS_{DL}}$.

In the sequel, we consider the logic DWL based on *difference logic* (DL) given as $\mathsf{DWL}(\mathbb{Z}, \{0\}, \{\leq_k :\ k \in \mathbb{Z}\})$ where for every $u, v, k \in \mathbb{Z}$, $(u, v) \in \leq_k$ iff

$u - v \leq k$. Then, *context-free systems with difference constraints* ($\mathsf{CFS_{DL}}$) are sets Δ of data word rewriting rules with one symbol on the left-hand side and zero, one or two symbols on the right-hand side. The formulas φ appearing in the rules are from $\mathsf{DWL}(\mathbb{Z}, \{0\}, \leq_k : k \in \mathbb{Z})$.

A $\mathbb{Z}$-input 1-counter machine[3] M is described by a finite set of states Q, an initial state $q_0 \in Q$, a final state $q_f \in Q$, a non-accepting state *fail* $\in Q$, and a counter c that contains initially 0. The initial configuration is given by the tuple $(q_0, 0)$. It reads pieces of input of the form $S(i)$ where S is a symbol out of Σ and $i \in \mathbb{Z}$ is an integer number. The instructions have the following form (q is different from q_f and *fail*):

1. $(q :\ c := c + 1; \mathsf{goto}\ q')$
2. $(q :\ c := c - 1; \mathsf{goto}\ q')$
3. $(q :\ \mathsf{If}\ c \geq 0\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
4. $(q :\ \mathsf{If}\ c = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
5. $(q :$ Read input $S(i)$. $\mathsf{If}\ S = X\ \mathsf{and}\ i = K\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
6. $(q :$ Read input $S(i)$. $\mathsf{If}\ S = X\ \mathsf{and}\ i \# c + K\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$,
7. $(q :\ \mathsf{If}\ P(c)\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$, where P is a unary Presburger predicate.

where $\# \in \{\leq, \geq, =\}$, $X \in \Sigma$ and $K \in \mathbb{Z}$ is an integer constant.

The language $L(M) \subseteq (\Sigma \times \mathbb{Z})^*$ is defined in a straightforward manner. It is easy to see that for a data independent formula φ one can construct a machine M_φ whose language is $[\![\varphi]\!]$.

For any M we have the following theorem.

Theorem 4. *Let Δ be a $\mathsf{CFS_{DL}}$ and M a $\mathbb{Z}$-input 1-counter machine. Then a $\mathbb{Z}$-input 1-counter machine M' with $L(M') = \mathsf{post}^*_{\Delta,p}(L(M))$ can be effectively constructed.*

The proof is done in several steps and follows the line of the proof given in in [17] for less general classes of rewriting systems and of counter machines M.

- The set $\{d \mid X(d) \Rightarrow^*_p \epsilon\}$ can be characterized by a Presburger formula with one free variable (difference + modulo constraints). This is done by using a translation to alternating one-counter automata.
- The decreasing rules (with ϵ on the right-hand side) of Δ can be eliminated from Δ. To do this, modulo constraints have to be added to the difference logic. Modulo constraints can be eliminated by coding the information in the control states.
- The set $post^*_{\Delta,p}(L(M))$ is then computed by (1) putting M into a special form and (2) applying saturation rules adding a finite number of new transitions to it.

Now we can state the main result of this section.

[3] this definition generalizes the one in [17] by allowing difference constraints in the read instructions

Theorem 5. *The problem* $\mathsf{post}^*_{\Delta,p}([\![\varphi_1]\!]) \cap [\![\varphi_2]\!] = \emptyset$ *is decidable for a* $\mathsf{CFS_{DL}}$ Δ *and two data independent formulas* φ_1 *and* φ_2.

We give a sketch of the proof. We (1) construct a machine M for $[\![\varphi_1]\!]$, (2) obtain the machine M' for $\mathsf{post}^*_{\Delta,p}([\![\varphi_1]\!])$ using theorem 4, (3) observe that intersection with $[\![\varphi_2]\!]$ can be done by computing the regular set over Σ corresponding to $[\![\varphi_2]\!]$ and restricting M' to words in this set and (4) observe that emptiness of a $\mathbb{Z}$-input 1-counter machine is decidable since emptiness of 1-counter machines is decidable.

$\mathsf{CFS_{DL}}$ allow to model recursive programs with one integer parameter. However, having only one symbol in the left-hand side of rules does not allow to model return values. Let us therefore consider extensions of $\mathsf{CFS_{DL}}$ with more than one symbol in the left-hand side of rules. If we allow rewrite rules with two symbols in the left-hand side where only the data attached to the first appears in constraints, the reachability problem is already undecidable[4]. This model corresponds to having integer return values. On the other hand, we can model return values from a finite domain by adding to the rules of $\mathsf{CFS_{DL}}$ a symbol to the beginning of the left and the right hand sides, provided the constraints do not use the data attached to these symbols. For this extension the reachability problem can be shown to be still decidable.

8 Conclusion

We have presented a generic framework for reasoning about infinite-state systems with unbounded control structures manipulating data over infinite domains. This framework extend and unify several of our previous works [11, 17, 16].

The framework we propose is based on constrained rewriting systems on words over infinite alphabets. The constraints are expressed in a logic which is parametrized by a theory on the considered data domain. We provide generic results for the decidability of the satisfiability problem of the fragment Σ_2 of this logic, and for proving inductive invariance and for carrying out Hoare style reasoning within this fragment.

We have shown that our framework can be used for handling a wide class of systems: recursive sequential programs, multithreaded programs, distributed algorithms, etc. Actually, it is not difficult to consider other rewriting semantics than those considered in the paper. For instance, all our results extend quite straightforwardly to cyclic rewriting allowing to deal with fifo queues. Therefore, our framework can also be used to reason about communicating systems through fifo channels which may contain data over infinite domains. This is particularly useful for handling in a parametric way communication protocols where message have sequence numbers (such as the sliding window protocol). Another potential application of our framework concern programs manipulating dynamic linked lists.

---

[4] A two-counter machine can be simulated : one counter is coded as the data value, the other one is coded by the number of symbols

Ongoing and future work include the extension of our framework to rewriting systems on more general structures like trees and some classes of graphs.

References

1. P. Abdulla and A. Nylen. Timed Petri Nets and BQOs. In *ICATPN'01*, volume 2075 of *LNCS*. Springer Pub., 2001.
2. 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.
3. 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.
4. 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.
5. P.A. Abdulla and B. Jonsson. Verifying networks of timed processes (extended abstract). In *Proc. of TACAS'98.* LNCS 1384, 1998.
6. 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.
7. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L.D. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *Proc. of CAV'01.* LNCS 2102, 2001.
8. B. Boigelot. *Symbolic Methods for Exploring Infinite State Space.* PhD thesis, Faculté des Sciences, Université de Liège, volume 189, 1999.
9. 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.
10. 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.
11. A. Bouajjani. Languages, Rewriting systems, and Verification of Infinte-State Systems. In *Proc. of ICALP'01*, volume 2076 of *LNCS*. Springer Pub., 2001.
12. A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with Lists Are Counter Automata. In *Proc. Intern. Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *LNCS*, Seattle, USA, August 2006. Springer Pub.
13. A. Bouajjani and J. Esparza. Rewriting Models for Boolean Programs. In *Proc. Intern. Conf. on Rewriting Techniques and Applications (RTA'06)*, volume 4098 of *LNCS*, Seattle, USA, August 2006. Springer Pub.
14. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. of CONCUR'97.* LNCS 1243, 1997.
15. A. Bouajjani, P. Habermehl, A. Rogalewicz, and T. Vojnar. Abstract Tree Regular Model Checking of Complex Dynamic Data Structures. In *Proc. Intern. Static Analysis Symposium (SAS'06)*, volume 4218 of *LNCS*, Seoul, Korea, August 2006. Springer Pub.
16. A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07.* LNCS, 2007.
17. Ahmed Bouajjani, Peter Habermehl, and Richard Mayr. Automatic Verification of Recursive Procedures with one Integer Parameter. *Theoretical Computer Science*, 295, 2003.

18. Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular Model Checking. In *Proc. 12th Intern. Conf. on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, Chicago (IL), USA, July 2000. Springer Pub.
19. M. Bozzano and G. Delzanno. Beyond Parameterized Verification. In *Proc. of TACAS'02*, volume 2280 of *LNCS*, Grenoble, France, 2002. Springer Pub.
20. 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.
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. Giorgio Delzanno. Constraint-based Verification of Parameterized Cache Coherence Protocols. *Formal Methods in System Design*, 23(3), 2003.
24. S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. In *Proc. of LICS'06*. IEEE, 2006.
25. 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.
26. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
27. P. Habermehl, R. Iosif, and T. Vojnar. Automata-Based Verification of Programs with Tree Updates. In *TACAS'06*, volume 2075 of *LNCS*. Springer Pub., 3920.
28. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic Model Checking with Rich Assertional Languages. In *CAV'97*, volume 1254 of *LNCS*. Springer, 1997.
29. P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proc. of CAV'98*, volume 1427 of *LNCS*. Springer, 1998.