Constrained Multiset Rewriting

Parosh Aziz Abdulla[1] and Giorgio Delzanno[2]

[1] Uppsala University, Sweden
parosh@it.uu.se
[2] Università di Genova, Italy
giorio@disi.unige.it

Abstract. We investigate model checking of a computation model called *Constrained Multiset Rewriting Systems (CMRS)*. A CMRS operates on configurations which are multisets of monadic predicate symbols, each with an argument ranging over the natural numbers. The transition relation is defined by a finite set of rewriting rules which are conditioned by simple inequalities on variables and constants. This model is able to specify systems with an arbitrary number of components where the internal state of a component may contain values ranging over the natural numbers. We show that CMRS are strictly more powerful than two existing related models namely that of Petri nets and relational automata. We prove decidability of the coverability problem, and undecidability of repeated reachability and configuration reachability for CMRS. Furthermore, we show that decidability of coverability does not extend to some natural extensions of the model, including the case where the predicates are dyadic. We report on using a prototype implementation to verify parameterized versions of a mutual exclusion and an authentication protocol.

1 Introduction

In the last decade there has been an extensive research effort to extend the applicability of model checking to systems with infinite state spaces. There are at least two reasons why a system may have an infinite state space. One reason is that the system operates on *unbounded data structures*, such as real-valued clocks in timed automata, stacks in push-down automata, queues in communicating processes, integer variables in relational automata, markings in Petri nets, etc. The second source of infiniteness is *parameterization*, where the description of the system is parameterized by the number of components, and the aim is to verify correctness regardless of the size of the system.

Most existing algorithms for verification of parameterized systems, e.g., the ones in [19, 1, 8, 6] are designed to work in the special case where the individual components are finite-state processes. However, there are many examples of parameterized systems in which an individual component cannot be faithfully modelled as a finite-state process. One important challenge is therefore to consider parameterized systems in which the components themselves are infinite-state. This challenge has recently been undertaken by a number of works. For

instance, the paper [7] considers programs with an arbitrary number of recursive calls, modelled as *Basic Parallel Processes (BPPs)*, where each call may also pass an integer parameter to a subroutine. The papers [4, 3] consider parameterized systems in which components are timed automata. Such systems contain both sources of infiniteness, as they operate on an unbounded number of variables each of which ranges over an unbounded domain.

In this paper, we consider a new class of such systems, which we call *Constrained Multiset Rewriting Systems (CMRS)*. A CMRS operates on configurations which are multisets of monadic predicate symbols, each with an argument ranging over the natural numbers. Transitions between configurations are defined by a finite set of rewriting rules. Each rule is conditioned by *gap-order* formulas of the form $x + c < y$, $x = y$, $x < c$, $x > c$, or $x = c$, where x and y are variable ranging over the natural numbers and c is a natural number. This model can capture the behaviour of parameterized systems in which the internal states of individual components may contain values ranging over the natural numbers. There are several examples of classes of protocols which can be modelled in this manner, e.g. , mutual exclusion protocols where the natural number inside each process is used to define the identity of the process, and authentication protocols where the number is used to define the key assigned to the process.

In addition to the relevance of CMRS as a formalism for parameterized systems, they are also interesting as a computation model in their own right. For instance, CMRS subsume several existing models for infinite-state systems such as Petri nets and relational automata [11]. In fact, we show that CMRS are strictly more powerful than both these models.

Our first result for CMRS is that the coverability problem is decidable for CMRS. The algorithm for checking coverability is obtained by a non-trivial application of a methodology based on the theory of well- and better-quasi orderings. However, we prove that reachability and repeated reachability are undecidable. The undecidability results are based on a "weak simulation" of 2-Counter Machines. Furthermore, we will show that coverability becomes undecidable for some extensions of the basic model, e.g., for dyadic CMRS. Despite the undecidability result, the algorithm for the monadic fragment can be generalized to a (possibly non-terminating) symbolic procedure for CMRS models with n-ary predicates.

We have implemented a prototype based on our algorithm. We have verified parameterized versions of a mutual exclusion protocol in which the participants have identities taken from an infinite domain; and an extended version of the authentication protocol proposed in [27] for managing services in a distributed system with mobile agents.

Related Work One of the earliest works on model checking of parameterized systems was reported by German and Sistla [18]. The paper considers systems with unbounded numbers of finite-state processes, and reduces the problem to a corresponding one for *Vector Addition Systems*, a model which is computationally equivalent to Petri nets. Another important line of research in this area has been that of *Regular Model Checking* [19, 1, 8, 6], Regular model checking is a uniform

framework for the analysis of parameterized systems consisting of homogeneous finite-state processes connected in linear or ring-formed topologies. In all these works, the individual components are assumed to be finite-state, and therefore they cannot handle the classes of systems we consider here.

The work [7] analyzes BPPs augmented with integer parameters. BPPs are computationally weaker than Petri nets, and therefore also weaker than CMRS. The papers [4, 3] consider parameterized systems of timed automata. However, timed systems obviously exhibit behaviours, such as timed transitions, which are not present in untimed systems, and vice versa. This is reflected in the verification algorithms which are totally different in the two cases.

The constraints we consider in this paper correspond to the *gap-order* constraints introduced by Revesz in the deductive database language $Datalog_{gap}$ [23, 24]. The language is defined over a fixed set of integer variables which represent the attribute of the database relation. Revesz defined a bottom-up query evaluation algorithm for $Datalog_{gap}$. In [17] Fribourg and Richardson applied Revesz's algorithm for verifying concurrent systems with finitely many components and with a fixed number of integer variables, obtaining a model similar to vector addition systems. More general constraint-based verification procedures for systems with a fixed number of integer or real valued variables have been studied, e.g., in [10, 15]. Relational automata [11] operate also on finite sets of integer variables, where the transitions are guarded by the ordering of the variables. The main difference compared to the above works is that we deal with systems with an arbitrary (rather than fixed) number of variables. In fact, we show that this strictly increases the expressive power of the model, e.g. compared to relational automata.

The paper [13] considers combining multiset rewriting with a generic constraint system. A scheme is presented for symbolic computation of sets of predecessors. The method, instantiated on linear arithmetic constraints, has been applied to verify properties of a parameterized version of the Ticket mutual exclusion protocol, and to verify secrecy and authentication properties for time-sensitive cryptographic protocols [9, 14]. Termination is not guaranteed in the general setting of [13], and in particular no classes of systems and properties are provided with decidable/undecidable properties. More concrete constraints such as gap-orders, are crucial for efficient verification of the classes of protocols we consider in this paper.

Our model shares some similarities with the MASPN model (an extension of Petri Nets with mobility and authentication) studied in [27]. In MASPN tokens are coloured with identifiers taken from an infinite discrete domain. Our model can also be viewed as a coloured Petri net, where the colour of a token is defined by its integer attribute. The main difference is that in [27] only equality between variables is allowed (in addition to a mechanisms for generating new process identifiers). Therefore, the language is not sufficiently expressive to encode gap-order conditions of the form $x + c < y$. Secondly, although transition arcs can be labelled with a variable, the variables of outcoming arcs are forced to occur in the incoming arcs. Thus, the model cannot express the production of generic

values, a feature which we use, e.g., in modelling the authentication protocol mentioned above.

Outline In the next Section, we introduce the model of CMRS. In order to simplify the presentation, we consider first a restricted model where we only allow conditions which compare variables. We introduce a coverability algorithm in Section 3, and show how to compute each step of the algorithm in Section 4. In Section 5 we prove that termination is guaranteed for the algorithm. In Section 6 we describe how to extend the coverability algorithm to the full model, where we also allow comparisons with constants. In Section 7 we show undecidability of reachability and repeated reachability, and also undecidability of coverability for some extensions of the model (including the dyadic case). In Section 8 we compare the computational power of (monadic) CMRS with Petri nets and relational automata. In Section 9 we describe our case studies. Finally, we give some conclusions and directions for future research in Section 10.

We provide two appendices, one containing proofs of lemma and theorems, and the other giving details of the case studies.

2 Constrained Multiset Rewriting

In this section we give preliminaries and definition of the CMRS model.

Preliminaries We let $\mathbb{N}$ denote the set of natural numbers. We assume a set $\mathbb{V}$ of variables which range over the integers, and a set $\mathbb{P}$ of unary predicate symbols. For a set A, we use A^* and $A^{\otimes}$ to denote the sets of words and multisets over A respectively. For $a \in A$ and a multiset B over A we write $B(a)$to denote the number of occurrences of a in B. Sometimes, we write multisets as lists, so $[1, 5, 5, 1, 1]$ represents a multiset B over $\mathbb{N}$ where $B(1) = 3$, $B(5) = 2$ and $B(x) = 0$ for $x \neq 1, 5$. We use the usual relations and operations such as $\leq$, $+$, and $-$ on multisets. In particular, $B_2 \ominus B_1$ is the multiset B where $B(a) = B_2(a) - B_1(a)$ if $B_2(a) \geq B_1(a)$ while $B(a) = 0$ otherwise.

For a set $V \subseteq \mathbb{V}$, a *valuation* Val of V is a mapping from V to $\mathbb{N}$, and a renaming Ren of V is a mapping from V to $\mathbb{V}$. A renaming Ren need not be injective, i.e., several variables may be renamed to the same variable by Ren. We say that Ren is renaming to W if $Ren(x) \in W$ for each $x \in V$. When the set V is clear from the context, we do not mention it; simply saying valuation (rather than valuation of V) and renaming (rather than renaming of V)

Some time, we write the explicit definition of a renaming. For instance $Ren = (x_1 \mapsto w_1\ ,\ x_2 \mapsto w_2\ ,\ x_3 \mapsto w_3)$ stands for $Ren(x_1) = w_1$, $Ren(x_2) = w_2$, and $Ren(x_3) = w_3$. We use a similar notation for valuations.

Constrained Multiset Rewriting Systems A *condition* is a finite conjunction of formulas of the forms: $x <_c y$ or $x = y$, where $x, y \in \mathbb{V}$ and $c \in \mathbb{N}$. Here $x <_c y$ stands for $x + c < y$. Sometimes, we treat a condition ψ as set, and write e.g. $(x <_c y) \in \psi$ to indicate that $x <_c y$ is one of the conjuncts in ψ. A *term* is of

the form $p(x)$ where $p \in \mathbb{P}$ and $x \in \mathbb{V}$. A *ground term* is of the form $p(c)$ where $p \in \mathbb{P}$ and $c \in \mathbb{N}$. A *Constrained Multiset Rewriting System (CMRS)* $\mathcal{S}$ consists of a finite set of *rules* each of the form:

$$L \rightsquigarrow R \ : \ \psi$$

where L and R are multisets of terms, and ψ is a condition. We assume that ψ is consistent (otherwise, the rule is never enabled).

For a condition ψ, we use $Var(\psi)$ to denote the set of variables which occur in ψ. For a valuation Val, we use $Val(\psi)$ to denote the result of substituting each variable x in ψ by $Val(x)$. We use $Val \models \psi$ to denote that $Val(\psi)$ evaluates to *true*. Also, for a renaming Ren, we define $Ren(\psi)$ to be the condition we get by replacing each x in ψ by $Ren(x)$. For a multiset T of terms we define $Var(T)$, $Val(T)$, and $Ren(T)$ in a similar manner. In particular, $Ren(T)$ and $Val(T)$ are multisets of terms and ground terms respectively. For a rule ρ of the above form, we define $Var(\rho) = Var(L) \cup Var(R) \cup Var(\psi)$.

Configurations and Transitions A *configuration* is a multiset of ground terms. The set of rules induces a transition relation $\longrightarrow$ on configurations, where $\longrightarrow = \bigcup_{\rho \in \mathcal{S}} \xrightarrow{\rho}$, and $\xrightarrow{\rho}$ represents the effect of applying the rule ρ. More precisely, for a rule ρ of the above form, we have $\gamma_1 \xrightarrow{\rho} \gamma_2$ if there is a valuation Val such that the following three conditions are satisfied:

- $Val \models \psi$
- $\gamma_1 \geq Val(L)$
- $\gamma_2 = \gamma_1 - Val(L) + Val(R)$

We use $\xrightarrow{*}$ to denote the reflexive transitive closure of $\longrightarrow$. For a configuration γ and a set Γ of configurations, we use $\gamma \xrightarrow{*} \Gamma$ to denote that there is a $\gamma' \in \Gamma$ with $\gamma \xrightarrow{*} \gamma'$. For a configuration γ and a predicate symbol p, we use $\gamma \xrightarrow{*} p$ to denote that p occurs in some γ' with $\gamma \xrightarrow{*} \gamma'$.

A set Γ of configurations is said to be *upward closed* (with respect to multiset ordering $\leq$), if $\gamma \in \Gamma$ and $\gamma' \geq \gamma$ implies $\gamma' \in \Gamma$. For a configuration γ, we use $\gamma\uparrow$ to denote the *upward closure* of γ, i.e., $\gamma\uparrow = \{\gamma' \mid \gamma \leq \gamma'\}$.

Example The definition of the transition relation $\longrightarrow$ interprets a rule of the form given above as a collection of rewriting rules on ground terms. An instance is obtained by taking a valuation which satisfies ψ. Consider the rule:

$$[p(x)\ ,\ q(y)] \ \rightsquigarrow \ [q(z)\ ,\ r(x)\ ,\ r(w)] \ \ : \ \ \{x <_2 y\ ,\ x <_4 z\ ,\ z < w\}$$

A valuation which satisfies the condition is $Val(x) = 1,\ Val(y) = 4,\ Val(z) = 8,$ $Val(w) = 10$, Therefore, we have a transition
$[p(1), p(3), q(4)] \longrightarrow [p(3), q(8), r(1), r(10)]$

Coverability The *coverability problem* is defined as follows:
Instance: A CMRS, an *initial* configuration γ_{init}, and a *final* predicate symbol p_{fin}.
Question: $\gamma_{init} \xrightarrow{*} p_{fin}$?

In Section 3 we present an algorithm which solves the predicate reachability problem, and hence we have:

Theorem 1. *The coverability problem is decidable for CMRS.*

The coverability problem described above is equivalent to the problem of the reachability of the upward closure (i.e. coverability) of a set of configurations. Using standard techniques [26], we can show that checking several classes of safety properties for CMRS can be reduced to the coverability problem.

3 Algorithm

In this section we give an overview of the algorithm for solving the coverability problem based on the generic backward analysis algorithm presented in [2]. The difficult challenge in applying this methodology is to invent a symbolic representation (called *constraints*) which allows effective implementation of each step, and which guarantees termination of the algorithm.

Constraints The algorithm operates on *constraints*, where each constraint ϕ characterizes an infinite set $[\![\phi]\!]$ of configurations. A *constraint* ϕ is of the form $T : \psi$ where T is a multiset of terms and ψ is a condition. The constraint characterizes the (upward closed) set $[\![\phi]\!] = \{\gamma |\ \exists Val.\ (Val \models \psi) \wedge (Val(T) \leq \gamma)\}$ of configurations. Notice that if ψ is inconsistent, then $[\![\phi]\!]$ is empty. Such a constraint can be safely discarded in the reachability algorithm presented below. Therefore, we assume in the sequel that all conditions in constraints are consistent. We define $Var(\phi) = Var(T) \cup Var(\psi)$.

Observe that the coverability problem can be reduced to constraint reachability. More precisely, $\gamma_{init} \stackrel{*}{\longrightarrow} p_{fin}$ is equivalent to $\gamma_{init} \stackrel{*}{\longrightarrow} \phi_{fin}$ where ϕ_{fin} is the constraint $[p_{fin}(x)] : true$.

For constraints ϕ_1, ϕ_2, we use $\phi_1 \sqsubseteq \phi_2$ to denote that ϕ_1 is *entailed* by ϕ_2, i.e., $[\![\phi_1]\!] \supseteq [\![\phi_2]\!]$. For a constraint ϕ, we define $Pre(\phi)$ to be a finite set of constraints which characterize the configurations from which we can reach a configuration in ϕ through the application of a single rule. In other words $\bigcup_{\phi_1 \in Pre(\phi)} [\![\phi_1]\!] = \{\gamma |\ \exists \gamma' \in [\![\phi]\!].\ \gamma \longrightarrow \gamma'\}$.

Example Consider the constraint $\phi_1 = [p(x_1)\ ,\ q(x_2)\ ,\ q(x_3)]\ :\ \{x_1 <_2 x_2\ ,\ x_2 <_1 x_3\}$, and the configurations $\gamma_1 = [p(2), q(8), q(5), p(1)]$ and $\gamma_2 = [p(2), q(2), q(5), p(1)]$. Then $\gamma_1 \in [\![\phi_1]\!]$ and $\gamma_2 \notin [\![\phi_1]\!]$. Consider the constraints $\phi_2 = [p(y_1)\ ,\ q(y_2)]\ :\ \{y_1 < y_2\}$ and $\phi_3 = [p(y_1)\ ,\ q(y_2)]\ :\ \{y_1 <_4 y_2\}$. Then $\phi_2 \sqsubseteq \phi_1$ and $\phi_3 \not\sqsubseteq \phi_1$.

Symbolic Algorithm Given an instance of the coverability problem (Section 2), defined by γ_{init} and the constraint ϕ_{fin} corresponding to p_{fin}, the symbolic algorithm performs a fixpoint iteration starting from ϕ_{fin} and repeatedly applying *Pre* on the generated constraints. The iteration stops if either (i) we generate a constraint ϕ with $\gamma_{init} \in [\![\phi]\!]$; or (ii) we reach a point where, for each newly generated constraint ϕ, there is a constraint ϕ' generated in a previous iteration with $\phi' \sqsubseteq \phi$. We give a positive answer to the coverability problem in the first case, while we give a negative answer in the second case.

We observe that, in order to be able to implement such an algorithm, we need computability of (i) membership, (ii) entailment, and (iii) the function *Pre*. The computability of these three relations is shown in Section 4 (Lemma 2). These above three conditions are sufficient conditions for semi-decidability of the problem. In Section 5 we also show that the algorithm is guaranteed to terminate.

4 Computability

In this section, we show computability of membership, entailment, and the predecessor function for constraints. First, we define a normal form for constraints. A constraint $T : \psi$ is said to be in *normal form* whenever the condition ψ satisfies the following requirements:

1. if $(x <_{c_1} y) \in \psi$ and $(y <_{c_2} z) \in \psi$ then $(x <_{c_3} z) \in \psi$ for some c_3 with $c_1 + c_2 < c_3$.
2. if $(x <_c y) \in \psi$ and $(y = z) \in \psi$ then $(x <_c z) \in \psi$.
3. if $(x <_c y) \in \psi$ and $(x = z) \in \psi$ then $(z <_c y) \in \psi$.
4. if $(x = y) \in \psi$ and $(y = z) \in \psi$ then $(x = z) \in \psi$.
5. For each $x, y \in \mathbb{V}$, at most one conjunct in ψ contains both x and y.
6. $\mathit{Var}(\psi) \subseteq \mathit{Var}(T)$.

Lemma 1. *For each constraint ϕ we can effectively compute a constraint ϕ_{norm} such that ϕ_{norm} is in normal form and such that $[\![\phi]\!] = [\![\phi_{norm}]\!]$.*

The normalization procedure consists of repeatedly adding conjuncts to ψ which maintain properties 1-4 and removing conjuncts which violate property 5. When the above procedure stabilizes, we remove all conjuncts in ψ containing variables not in $\mathit{Var}(T)$. Normalization can also be used to check consistency: the constraint is consistent if and only if no inequalities of the form $x <_c x$ are generated.

Lemma 2. *Membership, entailment, and Pre are computable for constraints.*

The full proof of the lemma is given in the appendix. The main concepts are the following. For a constraint ϕ and a configuration γ, it follows by definition that $\gamma \in [\![\phi]\!]$ iff there is a valuation *Val* of $\mathit{Var}(\phi)$ such that $\mathit{Val}(\psi) \wedge (\gamma \geq \mathit{Val}(T))$. Computability follows since there are only finitely many valuations *Val* with $\gamma \geq \mathit{Val}(T)$.

Consider constraints $\phi_1 = (T_1 : \psi_1)$ and $\phi_2 = (T_2 : \psi_2)$ which are in normal form (by Lemma 1 this is not a restriction). Let $\mathcal{R}$ be the set of renamings of $\mathit{Var}(T_1)$ such that $\mathit{Ren}(T_1) \leq T_2$. Then $\phi_1 \sqsubseteq \phi_2$ is characterized by the formula by

$$\forall x_1 \cdots x_2.\ \left(\psi_2 \implies \bigvee_{Ren \in \mathcal{R}} \mathit{Ren}(\psi_1)\right)$$

Since the set $\mathcal{R}$ is finite, checking the formula amounts to checking the satisfiability of a Boolean combination of formulas of the forms $x = y$ or $x <_c y$.

Let $\mathcal{S}$ be a CMRS and ϕ_2 be a constraint. We define $Pre(\phi_2) = \bigcup_{\rho \in \mathcal{S}} Pre_\rho(\phi_2)$, where $Pre_\rho(\phi_2)$ describes the effect of running the rule ρ backwards from the configurations in ϕ_2. Let $\rho = (L \rightsquigarrow R : \psi)$ and $\phi_2 = (T_2 : \psi_2)$. Let W be any set of variables such that $|W| = |\,Var(\phi_2) \cup Var(\rho)|$. We define $Pre_\rho(\phi_2)$ to be the set of constraints of the form $T_1 : \psi_1$, such that there are renamings Ren, Ren_2 of $Var(\rho)$ and $Var(\phi_2)$ respectively to W, and

$$\bullet\ T_1 = Ren_2(T_2) \ominus Ren(R) + Ren(L) \qquad \bullet\ \psi_1 = Ren(\psi) \wedge Ren_2(\psi_2)$$

Example Consider the constraints

$$\phi_1 = [p(x_1)\ ,\ q(x_2)\ ,\ q(x_3)\ ,\ r(x_4)]\ :\ \{x_1 <_1 x_2\ ,\ x_2 <_3 x_4\ ,\ x_1 < x_3\ ,\ x_1 <_8 x_4\}$$
$$\phi_2 = [p(y_1)\ ,\ q(y_2)\ ,\ q(y_3)\ ,\ r(y_4)\ ,\ s(y_5)]\ :\ \{y_1 <_1 y_3\ ,\ y_2 <_3 y_3\ ,\ y_1 < y_4\ ,\ y_2 <_2 y_4\}$$

Then the set $\mathcal{R} = \{Ren_1, Ren_2\}$ where

$$Ren_1 = (x_1 \mapsto y_1\ ,\ x_2 \mapsto y_2\ ,\ x_3 \mapsto y_3\ ,\ x_4 \mapsto y_4)$$
$$Ren_2 = (x_1 \mapsto y_1\ ,\ x_2 \mapsto y_3\ ,\ x_3 \mapsto y_2\ ,\ x_4 \mapsto y_4)$$

Therefore, $\phi_1 \sqsubseteq \phi_2$ is characterized by validity of the formula

$$\begin{pmatrix} y_1 <_1 y_3 \\ y_2 <_3 y_3 \\ y_1 < y_4 \\ y_2 <_2 y_4 \end{pmatrix} \implies \begin{pmatrix} y_1 <_1 y_2 \\ y_2 <_3 y_4 \\ y_1 < y_3 \\ y_1 <_8 y_4 \end{pmatrix} \vee \begin{pmatrix} y_1 <_1 y_3 \\ y_3 <_3 y_4 \\ y_1 < y_2 \\ y_1 <_8 y_4 \end{pmatrix}$$

Consider the constraint $\phi = [q(x_1)\ ,\ s(x_2)\ ,\ r(x_2)]\ :\ \{x_1 < x_2\}$ and the rule $\rho = [p(y_1)\ ,\ p(y_3)] \rightsquigarrow [q(y_2)\ ,\ r(y_3)]\ :\ \{y_3 < y_2\}$. Fix $W = \{w_1, w_2, w_3, w_4, w_5\}$, and define $Ren_2 = (x_1 \mapsto w_1\ ,\ x_2 \mapsto w_2)$,
$Ren = (y_1 \mapsto w_3\ ,\ y_2 \mapsto w_1\ ,\ y_3 \mapsto w_4)$. Then one member of Pre_ρ is given by $[s(w_2)\ ,\ r(w_2)\ ,\ p(w_3)\ ,\ p(w_4)]\ :\ \{w_1 < w_2\ ,\ w_4 < w_1\}$, which after normalization becomes $[s(w_2)\ ,\ r(w_2)\ ,\ p(w_3)\ ,\ p(w_4)]\ :\ \{w_4 <_1 w_2\}$. Observe that (i) the normalization procedure may introduce new constants (1 in this case) which are not part of the original constraint; (ii) if we choose $Ren = (y_1 \mapsto w_3\ ,\ y_2 \mapsto w_1\ ,\ y_3 \mapsto w_2)$ then the resulting constraint will denote an empty set (its conditions will be inconsistent); (iii) the size of constraints may increase when computing Pre.

5 Termination

In [2] it is shown that the coverability algorithm is guaranteed to terminate in case $\sqsubseteq$ is a *well quasi-ordering (WQO)*. Following the methodology of [5], we show that $\sqsubseteq$ in fact satisfies a stronger property than WQO; namely that it is a *better quasi-ordering (BQO)*. The challenging task in applying the method is to find an "intermediate" class of constraints, here called *flat constraints*, and then showing (i) that flat constraints are BQO; and (ii) that each constraint is the union of a set of flat constraints.

5.1 WQOs and BQOs

A *quasi-ordering* or a *QO* for short, is a pair $(A, \preceq)$ where $\preceq$ is a reflexive and transitive (binary) relation on a set A. A QO $(A, \preceq)$ is a *well quasi-ordering* or a *WQO* for short, if for each infinite sequence $a_1, a_2, a_3, \ldots$ of elements of A , there are $i < j$ such that $a_i \preceq a_j$. Given a QO $(A, \preceq)$, we define a QO $(A^*, \preceq^*)$ on the set A^* such that $x_1\ x_2\ \cdots\ x_m \preceq^* y_1\ y_2\ \cdots\ y_n$ if and only if there is a strictly monotone injection h form $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$ such that $x_i \preceq y_{h(i)}$ for each $i : 1 \leq i \leq m$. A QO $(A^{\otimes}, \preceq^{\otimes})$ on the set $A^{\otimes}$ can be defined in a similar manner. We define the relation $\preceq^{\mathcal{P}}$ on the powerset $\mathcal{P}(A)$ of A, so that $A_1 \preceq^{\mathcal{P}} A_2$ if and only if $\forall b \in A_2 : \exists a \in A_1 : a \preceq b$.

In the following lemma we state some properties of BQOs[3] [5, 20].

Lemma 3. *1. Each BQO is WQO.*
2. If A is finite, then $(A, =)$ is BQO.
3. If $(A, \preceq)$ is BQO, then $(A^*, \preceq^*)$, $(A^{\otimes}, \preceq^{\otimes})$, and $\left(\mathcal{P}(A), \preceq^{\mathcal{P}}\right)$ are BQOs.

5.2 Flat Constraints

A *flat constraint* ϕ_{fl} is a of the form $B_0\ d_1\ B_1\ d_2\ \cdots\ d_n\ B_n$ where $B_0, B_1, \ldots, B_n \in \mathbb{P}^{\otimes}$ and $d_1, d_2, \ldots, d_n \in \mathbb{N}$. In other words, a flat constraint is a word which alternatively contains multisets of predicate symbols and natural numbers, starting and ending with a multiset of predicate symbols. Furthermore, we require that the multisets $B_0, B_1, \ldots, B_n$ are all non-empty. For a configuration γ, we have $\gamma \in [\![\phi_{fl}]\!]$ if there are natural $c_0, \ldots, c_n \in \mathbb{N}$ such that (i) $c_i - c_{i-1} > d_i$ for each $i : 1 \leq i \leq n$; and (ii) $\gamma(p(c_i)) \geq B_i(p)$ for each predicate symbol p and $i : 0 \leq i \leq n$.

We observe that Each B_i is a multiset over the finite set $\mathbb{P}$. Therefore the B_i's are BQO under the multiset ordering $\leq$ by properties 2 and 3 in Lemma 3. By a similar reasoning, the d_i's are BQO under the standard ordering $\leq$ on natural numbers. Since each flat constraint is a finite word of B_i's and d_i's , it follows by property 3 that flat constraints are BQO under $\sqsubseteq$. Finally $\sqsubseteq$ is WQO by property 1. This gives the following.

Lemma 4. *The entailment relation $\sqsubseteq$ is a BQO (and therefore also a WQO) on flat constraints.*

Example Consider the flat constraint $\phi_1 = [p]\ 4\ [q, r]\ 2\ [r]$, and the configurations $\gamma_1 = [p(1), q(7), r(7), r(11), q(3)]$, $\gamma_2 = [p(1), q(7), r(7), r(8), q(3)]$, and $\gamma_3 = [p(1), q(7), r(8), r(11), q(3)]$. Then $\gamma_1 \in [\![\phi_1]\!]$ and $\gamma_2, \gamma_3 \notin [\![\phi_1]\!]$.

Consider the flat constraints $\phi_2 = [p]\ 2\ [r]\ 1\ [r]$, $\phi_3 = [p]\ 5\ [r]$, and $\phi_4 = [p]\ 10\ [r]$. Then $\phi_2 \sqsubseteq \phi_1$, $\phi_3 \sqsubseteq \phi_1$, and $\phi_4 \not\sqsubseteq \phi_1$.

[3] The technical definition of BQOs is quite complicated and can be found in e.g. [5]. The actual definition is not needed for understanding the rest of the paper, and is therefore omitted here.

5.3 Translation

Consider a constraint $\phi = T : \psi$ in normal form. A *flattening* F of $Var(\phi)$ is a word of the form $X_0\ d_1\ X_1\ d_2\ \cdots\ d_n\ X_n$ where $d_1, d_2, \ldots, d_n \in \mathbb{N}$ and the following three conditions are satisfied:

- $X_0, X_1, \ldots, X_n$ is a partitioning of $Var(\phi)$.
- If $(x = y) \in \psi$ then $x, y \in X_i$ for some $i : 1 \leq i \leq n$.
- If $(x <_c y) \in \psi$, $x \in X_i$, and $y \in X_j$ then $c \leq \left(\sum_{k=i+1}^{j} d_k + 1\right)$.

Intuitively, variables which are required to be equal by ϕ, are put in the same X_i. Also, variables which are ordered according to ϕ, are placed sufficiently far apart to cover the corresponding gap. The *flattening* ϕ_{fl} of ϕ *induced* by F is a flat constraint $B_0\ d_1\ B_1\ d_2\ \cdots\ d_n\ B_n$ such that $B_i(p) = \sum_{x \in X_i} T(p(x))$ for each $p \in \mathbb{P}$ and $i : 1 \leq i \leq n$. Since ϕ is in normal form, it follows that $Var(T) = Var(\phi)$ and hence B_i is not empty for each $i : 1 \leq i \leq n$.

Example Consider the constraint

$$\phi = [p(x_1)\ ,\ q(x_2)\ ,\ r(x_3)\ ,\ r(x_4)]\ :\ \{x_1 <_2 x_2\ ,\ x_1 <_1 x_3\ ,\ x_2 <_3 x_4\ ,\ x_1 <_8 x_4\}$$

A flattening of the condition and the induced flat constraint is given by
$\{x_1\}\ 3\ \{x_2, x_3\}\ 5\ \{x_4\}$ resp. $[p]\ 3\ [q, r]\ 5\ [r]$.
Another one is given by $\{x_1\}\ 3\ \{x_2\}\ 5\ \{x_3, x_4\}$ resp. $[p]\ 3\ [q]\ 5\ [r, r]$.
On the other hand, $\{x_1\}\ 3\ \{x_2\}\ 3\ \{x_3, x_4\}$ is not a flattening of the condition of ϕ.

Lemma 5. *For a constraint ϕ and a configuration γ, we have $\gamma \models \phi$ iff $\gamma \models \phi_{fl}$ for some flattening ϕ_{fl} of ϕ.*

From Lemma 3, Lemma 4, and Lemma 5, we get the following.

Lemma 6. *The set of constraints is a BQO (and therefore also a WQO) under entailment.*

6 Adding Constants

In this section, we show how to extend our language, so that we can compare variables with constants which are natural numbers. We do this in a stepwise manner, showing how we can extend the definitions, lemmas , and algorithms of the previous sections.

A *condition* is a finite conjunction of formulas of one of the forms $x <_c y$, $x = y$, $c < x$, $x = c$, or $x < c$, where $x \in \mathbb{V}$ and $c \in \mathbb{N}$. Throughout the rest of this section we assume a CMRS $\mathcal{S}$. We let *cmax* be the maximum natural number which occurs in any rule in $\mathcal{S}$. A *constraint* is of the form $T : \psi$ where T is a multiset of terms and ψ is a condition. If $(x = c) \in \psi$ or $(x < c) \in \psi$ then $c \leq cmax$. In other words, constants which define values or upper bounds of variables are in the range from 0 to *cmax*.

We can generalize the definition of normal form and then extend Lemma 2 (see proof of Theorem 2) for details. The crucial observation here is that conjuncts of the form $x = c$ or $x < c$ which are added by the normalization procedure satisfy the property that $c \leq cmax$. This is reflected in the extended definition of *flat constraints* which will now be of the form
$D_1 \ \cdots D_{cmax} \ d_0 \ B_0 \ d_1 \ B_1 \ d_2 \ \cdots \ d_n \ B_n$. Here $B_0, \ldots, B_n \in \mathbb{P}^{\otimes}$ and $d_0, d_1, \ldots, d_n \in \mathbb{N}$ are defined as before. We have also an additional part, where $D_0, \ldots, D_{cmax}$ are members of $\mathbb{P}^{\otimes}$ and are used to encode the terms whose arguments are in the interval from 0 to $cmax$. Using a similar reasoning to Section 5 we can show termination of the algorithm for the extended model. From this we get the following.

Theorem 2. *The coverability problem is decidable for CMRS with constants.*

7 Undecidable Properties and Extensions

In this section, we show undecidability of a number of properties for CMRS. Furthermore, we show that some natural extensions of the model imply undecidability of the coverability problem.

Undecidable Properties First, we show undecidability of *repeated reachability*. The problem is defined as follows
Instance: A CMRS $\mathcal{S}$, an *initial* configuration γ_{init}, and a rule $\rho \in \mathcal{S}$.
Question: Is there an infinite sequence of transitions from γ_{init} invoking ρ infinitely often?

The idea of the undecidability proof is that CMRS can *weakly* simulate 2-counter machines. Weak simulation means that testing the value of one counter is replaced by two operations: (i) resetting the value of the counter to zero; (ii) reducing the value of the other counter by an arbitrary number. Mayr [21] shows that repeated reachability is undecidable for weak (lossy) counter machines. From this we get the following.

Theorem 3. *Repeated Reachability is undecidable for CMRS.*

From Theorem 3 it follows immediately that action-based LTL is undecidable for CMRS. It is also possible to consider repeated *predicate* (rather than rule) reachability. It is straightforward to show that the two problems can be reduced to each other.

Next, we consider *configuration reachability*. This problem is formulated in the same manner as coverability (Section 2), except that we now ask the question whether $\gamma_{init} \xrightarrow{*} \gamma_{fin}$? In other words, we ask about the reachability of a configuration rather than a predicate symbol. Using a method similar to that used by De Frutos et al [12] we can extend the proof of Theorem 3 to show undecidability of configuration reachability. The idea is still to simulate a weak counter machine. However, checking reachability of a concrete configuration gives the additional power of being able to analyze the final configuration. In particular,

we will only accept computations where no losses (counter decrements in connection with zero-testing) occur; thus effectively simulating a perfect (non-lossy) counter machine.

Theorem 4. *Configuration Reachability is undecidable for CMRS.*

Undecidable Extensions First, we consider dyadic CMRS, in which predicate symbols may have two arguments. We include binary predicate symbols, thus allowing terms of the form $p(x, y)$ where $p \in \mathbb{P}$ and $x, y \in \mathbb{V}$. Configurations and constraints can be extended in the obvious manner. Normalization, checking membership, and checking entailment can be performed exactly in the same manner as for the monadic case (Section 4). This yields a semi-algorithm for checking coverability. However, the symbolic algorithm is no longer guaranteed to terminate. In fact, we are able to show the following theorem by reduction from the reachability problem for Turing machines.

Theorem 5. *The coverability problem is undecidable for dyadic CMRS.*

Small extensions of the constraint language can drastically change the expressive power of the resulting CMRS . For instance, adding conditions of the form $x - y \leq c$ makes the model Turing-powerful. The reason is that conditions like the above, together with the gap constraints, allow to check exact differences between variables. This makes it possible to encode increment and decrement operations. Since we can also check equality, we get the full power of counter machines.

8 Related Models

In this section we show that the (monadic) CMRS model is strictly more powerful than that two related models, namely that of Petri nets and relational automata [11]. To compare the relative expressiveness of the models we will differentiate them using language equivalence. We assume a finite alphabet Σ. For words w, w', we write $w \preceq w'$ to denote that w is a (not necessarily contiguous) subword of w', and write w^R to denote the reverse of w. We consider a particular language over Σ, namely $L_{PAL} = \left\{w_1 w_2 |\ w_2 \preceq w_1^R\right\}$. We will show that L_{PAL} can be recognized by a CMRS, but not by any Petri net or relational automaton.

To do that, we regard CMRS as language acceptors. We assume a CMRS $\mathcal{S}$, and fix an *initial* configuration γ_{init} and an *accepting* predicate symbol p_{acc}. We also assume a labelling function λ from the rules of $\mathcal{S}$ to Σ. We define $L(\mathcal{S})$ to be the set of finite words $a_1 a_2 \cdots a_n$ such that there is a sequence $\gamma_0 \xrightarrow{\rho_1} \gamma_1 \xrightarrow{\rho_2} \gamma_2 \cdots \xrightarrow{\rho_n} \gamma_n$ of transitions where (i) $\gamma_0 = \gamma_{init}$, (ii) p_{acc} appears in γ_n, and (iii) $\lambda(\rho_i) = a_i$ for each $i : 1 \leq i \leq n$. We say that L is a *CMRS language* if there is a CMRS $\mathcal{S}$ with $L = L(\mathcal{S})$.

Lemma 7. *There is CMRS $\mathcal{S}$ with $L(\mathcal{S}) = L_{PAL}$.*

Petri Nets We assume familiarity with the model of Petri nets. We consider a Petri net $\mathcal{N}$ as a language acceptor and define $L(\mathcal{N})$ in a similar manner to CMRS. More precisely, we fix an initial marking M_{init}, an accepting marking M_{acc}, and a labelling function from the transitions of $\mathcal{N}$ to Σ. Then, we define define $L(\mathcal{N})$ by considering labels of transition sequences leading from M_{init} to a marking which covers M_{acc}.

For a Petri $\mathcal{N}$, It is straightforward to derive a CMRS $\mathcal{S}$ with $L(\mathcal{S}) = L(\mathcal{A})$. This follows since Petri nets are restricted CMRS which operate on an empty set of variables. In [22], it is shown that the language consisting of words of the form ww^R cannot be accepted by a Petri net. In fact, it is straightforward to use the same construction given in [22] to show that there is no Petri net $\mathcal{N}$ such that $L(\mathcal{N}) = L_{PAL}$. This, together with Lemma 7, implies the following.

Theorem 6. *There exist CMRS languages which are not Petri net languages.*

Relational Automata A *relational automaton* operates on a finite set of integer variables, and is of the form (Q, δ) where Q and δ are finite sets of (local) states and transitions respectively. A transition is a triple (q_1, op, q_2) where $q_1, q_2 \in Q$ and op is of one of the following three operations: (i) *reading*: $read(x)$ reads a new value of variable x, (ii) *assignment*: $x := y$ assigns the value of variable y to x; (ii) *testing*: $x < y$, $x = y$, $x < c$, $x = c$, and $x > c$ are guards which compare the values of variables x, y and the integer constant c. The language of a relational automaton can be defined in the same manner as for the previous models.

Given a relational automaton $\mathcal{A}$, we can derive a CMRS $\mathcal{S}$ with $L(\mathcal{S}) = L(\mathcal{A})$. The proof details are given in the appendix as part of the proof of Theorem 7. The idea is to have a predicate symbol to encode each local state in $\mathcal{A}$. Also, for each variable x in $\mathcal{A}$, we have a predicate symbol q_x, where the argument of q_x encodes the value of x. Then each transition of $\mathcal{A}$ is encoded by a corresponding rule in $\mathcal{S}$. On the other hand, we show that there is no relational automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L_{PAL}$. This gives the following.

Theorem 7. *There exist CMRS languages which are not relational automata languages.*

9 Case Studies

We have implemented a prototype based on our symbolic verification procedure in SICStus Prolog. The prototype can be applied to monadic and non-monadic CMRS (without guarantee of termination in the latter case). Here we report on two case studies carried out through the prototype. More details can be found in the appendix.

Mutual Exclusion This protocol is inspired by Fischer's mutual exclusion protocol for timed systems [25]. Here, we present a version which achieves synchronization through a shared variable rather than timing constraints. The goal of

the protocol is to guarantee mutual exclusion among an arbitrary number of processes. Each process is in one of the states *idle*, *waiting*, and *cs* (critical section). Each process reads from and writes to a shared variable v, whose value is either $\perp$ or the identity of one of the processes. The variable v may be in one of two modes: v_0 (which is the initial mode) and v_1. Processes may dynamically be created and deleted during the execution of the protocol. When a new process is created, it is given a unique identity. A process can move from state *idle* to *waiting*. In such a case, the variable v will be assigned the identity of that process. The variable may change mode to v_1. It is only then processes are allowed to enter *cs*. More precisely, a process may cross from *waiting* to *cs* only if the shared variable is in mode v_1 and the identity of the process is the one carried by v. Also, when the mode is v_1, processes are prevented from changing state to *waiting*. In case a process with the wrong identity tries to enter *cs*, it will be returned to the state *idle*. Our CMRS model consists of 9 rules (shown in the appendix). When running the prototype on this problem to prove mutual exclusion, we obtain a fixpoint after 10 iterations generating 24 constraints in 1.5 seconds.

The FSC Authentication Protocol The FSC system is a protocol introduced in [27] to specify the interaction between *forwarded agents*, *service provider agents* and *clients* in a system in which processes are distributed over a finite set of locations and in which processes can move from location to location. Communication may take place only between agents in the same location. The protocol works under the assumption that only the forwarder agents know the locations in which a service can be provided. Thus, as a first step, a client moves to the location of the forwarder to send a request for a given service. The forwarder assigns to the request a new identifier, and passes the identifier and the location of the provider to the client. After this step, the forwarder moves to the location of the provider, communicates the identifier of the request to the provider, and then goes back to his home location. The provider stores the identifier and then waits for a client to login, Clients use the information received by the forwarder to move to the provider location, and then use the identifier received in the first phase to login. The provider grants the service to a client only after having matched the identifier received from the forwarder with that used by the client to login.

For this protocol, we have defined two different CMRS models. The first model is based on the assumption (present in the original version of the protocol) that the set of locations is finite. This model uses only monadic predicates and consists of 10 rules. The second model is parameterized on the number of locations, identifiers, and agents, and uses dyadic predicates. In the monadic model with one agent of each type, we computed a fixpoint after 22 iterations generating 3049 constraints. Adding one provider we reached a fixpoint after 29 iterations computing more than 12000 constraints. For the model parameterized in all dimensions, we computed a fixpoint after 22 iterations, generating about 8000 constraints. The last experiment shows that the property holds for any number of

forwarders, providers and clients. The execution time for the fully parameterized version is 9479.6 seconds.

10 Conclusions and Future Work

We have considered multiset rewriting systems, where the elements of a multiset are uninterpreted predicate symbols. The predicates have arguments which range over the natural numbers, and the rewriting rules are conditioned by simple arithmetical constraints. We provide a symbolic reachability algorithm for solving the coverability problem. We have used a prototype based on the algorithm to verify two non-trivial examples of protocols consisting of arbitrary numbers of processes which contain integer variables. The results are promising, especially taking into consideration that no optimizations are performed on the code or data structures. One direction for future work is to improve performance through the incorporation of widening and acceleration techniques to enhance the fixpoint iteration (e.g. [6]), and through the use of more efficient data structures (e.g. [16]).

Our model can also be viewed as coloured Petri nets, where the colours of the tokens are natural numbers, and where firing conditions may contain gap-orders. We show that our model contains strictly the class of Petri net languages. The criterion we use for accepting words is that of coverability. In a similar manner to [22], we can define a hierarchy of accepting conditions including , e.g., exact reachability. In fact, the relative power of the models remain regardless of the chosen accepting conditions. This is also true if we choose other equivalence relations, such as bisimulation or languages of infinite words.

The variables in the rewriting rules are interpreted over the natural numbers. It is relevant to investigate other classes of interpretation domains such as the rational numbers or reals.

References

1. P.A. Abdulla, B. Jonsson, Marcus Nilsson, and M. Saksena. A survey of regular model checking. In *Proc. CONCUR 2004,* 15^{th} *Int. Conf. on Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 348–360, 2004.
2. Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.
3. Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata. Multi-clock timed networks. In *Proc. LICS '04,* 20^{th} *IEEE Int. Symp. on Logic in Computer Science*, pages 345–354, 2004.
4. Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003.
5. Parosh Aziz Abdulla and Aletta Nylén. Better is better than well: On efficient verification of infinite-state systems. In *Proc. LICS '00,* 16^{th} *IEEE Int. Symp. on Logic in Computer Science*, pages 132–140, 2000.

6. Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In *Proc.* 15^{th} *Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, 2003.
7. A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 2136, 2001.
8. Ahmed Bouajjani and Tayssir Touili. Extrapolating Tree Transformations. In *Proc.* 14^{th} *Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, 2002.
9. M. Bozzano and G. Delzanno. Beyond parameterized verification. In *Proc. TACAS '02,* 8^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 221–235, 2002.
10. T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables. *ACM Trans. on Programming Languages and Systems*, 21(4):747–789, 1999.
11. K. Čerāns. Deciding properties of integral relational automata. In Abiteboul and Shamir, editors, *Proc. ICALP '94,* 21^{st} *International Colloquium on Automata, Languages, and Programming*, volume 820 of *Lecture Notes in Computer Science*, pages 35–46. Springer Verlag, 1994.
12. D. de Frutos Escrig, V. Valero Ruiz, and O. Marroquín Alonso. Decidability of properties of timed-arc Petri nets. In *ICATPN 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 187–206, 2000.
13. G. Delzanno. An assertional language for systems parametric in several dimensions. In *Proc. VEPAS '01, Workshop on Verification of Parameterized Systems*, ENTCS, 2001.
14. G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. In *Proc. TACAS '04,* 10^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, number 2988 in Lecture Notes in Computer Science, 2004.
15. G. Delzanno and A. Podelski. Model checking in clp. In *Proc. TACAS '99,* 5^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, 1999.
16. G. Delzanno and J. F. Raskin. Symbolic representation of upward-closed sets. In *Proc. TACAS '00,* 6^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *Lecture Notes in Computer Science*, pages 426–440, 2000.
17. L. Fribourg and J. Richardson. Technical Report LIENS-93-3, Laboratoire d'Informatique, Ecole Normale Superieure, Paris, 1996.
18. S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.
19. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.
20. Alberto Marcone. Foundations of bqo theory. *Transactions of the American Mathematical Society*, 345(2), 1994.
21. R. Mayr. Undecidable problems in unreliable computations. In *Theoretical Informatics (LATIN'2000)*, number 1776 in Lecture Notes in Computer Science, 2000.
22. J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentic-Hall, 1981.
23. P. Revesz. A closed form evaluation for datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.
24. P. Revesz. *Introduction to Constraint Databases.* Springer, 2002.

25. F. B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. In de Bakker, Huizing, de Roever, and Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, 1992.
26. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86,* 1^{st} *IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.
27. F. Rosa Velardo, O. Marroquín-Alonso, and D. de Frutos Escrig. Mobile synchronizing petri nets: a choreographic approach for coordination in ubiquitous systems. In *MTCoord 2005, 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems*, 2005.

A Proof of Some Lemmas

Lemma 2

Computability of membership is obvious. We consider checking entailment and computing *Pre*.

Checking Entailment We write $F(\phi_1, \phi_2)$ to denote the formula

$$\forall x_1 \cdots x_2.\ \left(\psi_2 \implies \bigvee_{Ren \in \mathcal{R}} Ren(\psi_1)\right)$$

We show that $F(\phi_1, \phi_2)$ holds iff $\phi_1 \sqsubseteq \phi_2$. Let $\phi_1 = (T_1 : \psi_1)$ and $\phi_2 = (T_2 : \psi_2)$. Below we show both directions of the equivalence.

(if) Suppose that $F(\phi_1, \phi_2)$ holds and $\gamma \in [\![\phi_2]\!]$. We show that $\gamma \in [\![\phi_1]\!]$.

Since $\gamma \in [\![\phi_2]\!]$ it follows that there is a valuation Val_2 of $Var(\phi_2)$ such that $Val_2 \models \psi_2$ and $\gamma \geq Val_2(T_2)$.

From this and the fact that $F(\phi_1, \phi_2)$ holds, it follows that there is a renaming of $Var(\phi_1)$ such that $T_2 \geq Ren(T_1)$ and $Val_2 \models Ren(\psi_1)$. Define the valuation Val_1 of $Var(\phi_1)$ such that $Val_1(x) = Val_2(Ren(x))$. We have $\gamma \geq Val_2(T_2) \geq Val_2(Ren(T_1)) = Val_1(T_1)$. Also, we know that $Val_2 \models Ren(\psi_1)$ and hence $Val_1 \models \psi_1$. This implies that $\gamma \in [\![\phi_1]\!]$.

(only if) Suppose that $F(\phi_1, \phi_2)$ does not hold. We show that there is a γ such that $\gamma \in [\![\phi_2]\!]$ and $\gamma \notin [\![\phi_1]\!]$.

Since $F(\phi_1, \phi_2)$ does not hold, we know that there is a valuation Val_2 of $Var(\phi_2)$ such that $Val_2 \models \psi_2$ and for each renaming Ren of $Var(\phi_1)$ it is the case that either $Ren(T_1) \not\leq T_2$ or $Val_2 \not\models Ren(\psi_1)$.

Define $\gamma = Val_2(T_2)$. We show that $\gamma \notin [\![\phi_1]\!]$. Take any valuation Val_1 of $Var(\phi_1)$. Suppose that $\gamma \geq Val_1(T_1)$ and $Val_1 \models \psi_1$. We show that this leads to a contradiction. Define h to be a partial mapping from $\mathbb{N}$ to $Var(\phi_2)$ such that $h(c) = y$ for some y such that $Val_2(y) = c$. If no such a y exists, the value of $h(c)$ is undefined. Define Ren to be the renaming of $Var(\phi_1)$ where $Ren(x) = h(Val_1(x))$. Observe that $Var(\phi_1) = Var(T_1)$ by normality of ϕ_1. The function Ren is well-defined as follows: $Val_2(T_2) = \gamma \geq Val_1(T_1)$. Therefore, if $Val_1(x) = c$ for some $x \in Var(T_1)$ and $c \in \mathbb{N}$, then there exists a $y \in Var(T_2)$ with $Val_2(y) = c$. Notice that $Val_1(x) = Val_2(Ren(x))$ for each $x \in Var(T_1)$. Furthermore, by the definition of Ren it follows that, for any $x_1, x_2 \in Var(\phi_1)$, if $Val_1(x_1) = Val_1(x_2)$ then $Ren(x_1) = Ren(x_2)$; hence $Val_2(Ren(x_1)) = Val_2(Ren(x_2))$ implies $Ren(x_1) = Ren(x_2)$. From this and the fact that $Val_2(T_2) \geq Val_1(T_1) = Val_2(Ren(T_1))$, it follows that $T_2 \geq Ren(T_1)$. Since ϕ_1 is in normal form, we know that $Var(\psi_1) \subseteq Var(T_1)$. This means that $Val_1(x) = Val_2(Ren(x))$ for each $x \in Var(\psi_1)$. From this and $Val_1 \models \psi_1$ we get $Val_2 \models Ren(\psi_1)$. This contradicts the fact that for each renaming Ren of $Var(\phi_1)$ we have either $Ren(T_1) \not\leq T_2$ or $Val_2 \not\models Ren(\psi_1)$.

Computing Pre For a set Φ of constraints we let $[\![\Phi]\!] = \bigcup_{\phi\in\Phi}[\![\phi]\!]$. For a constraint ϕ_2 and a rule $\rho \in \mathcal{S}$, let $\Phi_1 = Pre_\rho(\phi_2)$. We show that $[\![\Phi_1]\!] = \left\{\gamma_1 |\ \exists\gamma_2 \in [\![\phi_2]\!].\ \gamma_1 \xrightarrow{\rho} \gamma_2\right\}$ We show the set inclusion in both directions.

$[\![\Phi_1]\!] \subseteq \left\{\gamma_1 |\ \exists\gamma_2 \in [\![\phi_2]\!].\ \gamma_1 \xrightarrow{\rho} \gamma_2\right\}$: Suppose that $\gamma_1 \in [\![\Phi_1]\!]$, i.e., there is $\phi_1 \in \Phi_1$ such that $\gamma_1 \in [\![\phi_1]\!]$. We show that there is $\gamma_2 \in [\![\phi_2]\!]$ such that $\gamma_1 \xrightarrow{\rho} \gamma_2$.

Since $\gamma_1 \in [\![\phi_1]\!]$ we know that there is a valuation Val_1' of $Var(\phi_1)$ such that $Val_1' \models \psi_1$ and $\gamma_1 \geq Val_1'(T_1)$. Let Val_1 be any valuation of $Ren(Var(\rho)) \cup Ren_2(Var(\phi_2))$ such that $Val_1'(x) = Val_1(x)$ for each $x \in Var(\phi_1)$. Since $\phi_1 \in Pre_\rho(\phi_2)$, there are renamings Ren, Ren_2 of $Var(\rho)$ and $Var(\phi_2)$ such that $T_1 = Ren_2(T_2) \ominus Ren(R) + Ren(L)$ and $\psi_1 = Ren(\psi) \wedge Ren_2(\psi_2)$. Define a valuation Val of $Var(\rho)$ such that $Val(x) = Val_1(Ren(x))$ for each $x \in Var(\rho)$. Also, define a valuation Val_2 of $Var(\phi_2)$ such that $Val(x) = Val_2(Ren_2(x))$ for each $x \in Var(\phi_2)$. Let $\gamma_2 = \gamma_1 - Val(L) + Val(R)$.

We know that $Val_1 \models \psi_1$. It follows that $Val_1 \models Ren(\psi)$ and hence $Val \models \psi$. Also, we know that $\gamma_1 \geq Val_1(T_1) \geq Val_1(Ren(L)) = Val(L)$. It follows that $\gamma_1 \xrightarrow{\rho} \gamma_2$. It remains to show that $\gamma_2 \in [\![\phi_2]\!]$.

Let T_2' be such that $Ren_2(T_2') = Ren_2(T_2) \ominus Ren(R)$, and let $T_2'' = T_2 - T_2'$. Also, define R'' such that $Ren(R'') = Ren_2(T_2'')$, and let $R' = R - R''$. Notice that all these four multisets are well-defined. We have

$$
\begin{array}{l}
\gamma_2 \ = \\
\gamma_1 - Val(L) + Val(R) \ \geq \\
Val_1(T_1) - Val(L) + Val(R) \ = \\
Val_1(Ren_2(T_2) \ominus Ren(R) + Ren(L)) - Val(L) + Val(R) \ = \\
Val_1(Ren_2(T_2) \ominus Ren(R)) + Val(R) \ = \\
Val_1(Ren_2(T_2')) + Val(R') + Val(R'') \ = \\
Val_1(Ren_2(T_2')) + Val(R') + Val_1(Ren(R'')) \ = \\
Val_1(Ren_2(T_2')) + Val(R') + Val_1(Ren_2(T_2'')) \ = \\
Val_2(T_2') + Val(R') + Val_2(T_2'') \ = \\
Val_2(T_2) + Val(R') \ \geq \\
Val_2(T_2)
\end{array}
$$

In other words, $\gamma_2 \geq Val_2(T_2)$. Also, we know that $Val_1 \models \psi_1$ and therefore $Val_1 \models Ren_2(\psi_2)$ which means that $Val_2 \models \psi_2$. It follows that $\gamma_2 \in [\![\phi_2]\!]$.

$\left\{\gamma_1 |\ \exists\gamma_2 \in [\![\phi_2]\!].\ \gamma_1 \xrightarrow{\rho} \gamma_2\right\} \subseteq [\![\Phi_1]\!]$: Suppose that $\gamma_1 \xrightarrow{\rho} \gamma_2$ and $\gamma_2 \in [\![\phi_2]\!]$. We show that $\gamma_1 \in [\![\phi_1]\!]$ for some $\phi_1 \in \Phi_1$.

Since $\gamma_2 \in [\![\phi_2]\!]$, we know that there is a valuation Val_2 of $Var(\phi_2)$ such that $Val_2 \models \psi_2$ and $\gamma_2 \geq Val_2(T_2)$. Since $\gamma_1 \xrightarrow{\rho} \gamma_2$, we know that there is a valuation of $Var(\rho)$ such that $Val \models \psi\ \gamma_2 = \gamma_1 - Val(L) + Val(R)$. Define renamings Ren, Ren_2 of $Var(\rho)$ and $Var(\phi_2)$ to W respectively such that:

- For $x \in Var(\rho)$ and $y \in Var(\phi_2)$, we have $Ren(x) = Ren_2(y)$ iff $Val(x) = Val_2(y)$.
- For $x, y \in Var(\rho)$, we have $Ren(x) = Ren(y)$ iff $Val(x) = Val(y)$. A similar condition holds in case $x, y \in Var(\phi_2)$.

In other words, x and y are renamed identically iff they are assigned identical values by either Val or Val_2. The renamings Ren, Ren_2 are well-defined since at most $|Var(\phi_2) \cup Var(\rho)|$ different values are assigned to the variables, and since by definition we have $|W| = |Var(\phi_2) \cup Var(\rho)|$,

Define $\phi_1 = (T_1 : \psi_1)$ where $T_1 = Ren_2(T_2) \ominus Ren(R) + Ren(L)$ and $\psi_1 = Ren(\psi) \wedge Ren_2(\psi_2)$. Below, we show that $\gamma_1 \in [\![\phi_1]\!]$.

Define a valuation Val_1 of $Var(\rho) \cup Var(\phi_2)$ such that $Val_1(y) = Val(x)$ if $Ren(x) = y$ and $Val_1(y) = Val_2(x)$ if $Ren_2(x) = y$. The valuation Val_1 is well-defined since $Ren(x_1) = Ren_2(x_2)$ implies $Val(x_1) = Val_2(x_2)$. We have

$$
\begin{array}{l}
\gamma_1 \ = \\
\gamma_2 - Val(R) + Val(L) \ = \\
\gamma_2 \ominus Val(R) + Val(L) \ \geq \\
Val_2(T_2) \ominus Val(R) + Val(L) \ = \\
Val_1(Ren_2(T_2)) \ominus Val_1(Ren(R)) + Val_1(Ren(L)) \ = \\
Val_1(Ren(T_2) \ominus Ren(R) + Ren(L)) \ = \\
Val_1(T_1)
\end{array}
$$

The equality in the second line above holds since $\gamma_2 = \gamma_1 - Val(L) + Val(R)$. This means that $\gamma_2 \geq Val(R)$ and therefore $\gamma_2 - Val(R) = \gamma_2 \ominus Val(R)$. Also, the next last inequality holds since, for each $x_1, x_2 \in Var(Ren(\rho)) \cup Var(Ren_2(\phi_2))$, $Val_1(x_1) = Val_1(x_2)$ implies $x_1 = x_2$. Furthermore, since $Val \models \psi$ and $Val_2 \models \psi_2$, it follows that $Val_1 \models Ren(\psi)$ and $Val_1 \models Ren_2(\psi_2)$, and hence $Val_1 \models \psi_1$. This implies that $\gamma_1 \in [\![\phi_1]\!]$.

Lemma 4

Suppose that $\phi_{fl} = B_0\ d_1\ B_1\ d_2\ \cdots\ d_m\ B_m$ and $\phi'_{fl} = B'_0\ d'_1\ B'_1\ d'_2\ \cdots\ d'_n\ B'_n$. By definition it follows that $\phi_{fl} \sqsubseteq \phi'_{fl}$ iff there is a strictly monotone injection h from $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$ such that

- $B_i \leq B'_{h(i)}$ for each $i : 1 \leq i \leq m$.
- $d_{i+1} \leq \left(\sum_{j=h(i)+1}^{h(i+1)} d'_j\right) + h(i+1) - h(i) - 1$, for each $i : 0 \leq i < m$.

The result follows from Lemma 3 as follows: Each B_i is a multiset over the finite set $\mathbb{P}$. Therefore the B_i's are BQO under the multiset ordering $\leq$ by properties 2 and 3 in Lemma 3. By a similar reasoning, the d_i's are BQO under the standard ordering $\leq$ on natural numbers. Since each flat constraint is a finite word of B_i's and d_i's , it follows by property 3 that flat constraints are BQO under $\sqsubseteq$. Finally $\sqsubseteq$ is WQO by property 1

Lemma 5

Let ϕ be of the form $T : \psi$. We show the implication in both directions.

(only if) Suppose that $\gamma \models \phi$, i.e., there is a valuation Val such that $Val \models \psi$ and $\gamma \geq Val(T)$. Let $X_0, \ldots, X_n$ be sets of variables such that

– $Var(\phi) = X_1 \cup \cdots \cup X_n$, and
– If $x \in X_i$ and $y \in X_j$ then $i \leq j$ iff $Val(x) \leq Val(y)$.

Let $d_i = Val(y) - Val(x) - 1$ for some $x \in X_{i-1}$, $y \in X_i$. In other words, we organize the variables in $Var(\phi)$ by their valuations according to Val. More precisely, variables of the same value are put into the same X_i, while d_i describes the difference of value between variables in X_{i-1} and X_i. We show that $X_0\ d_1\ X_1\ d_2\ \cdots\ d_n\ X_n$ is a flattening of $Var(\phi)$:

– Since Val assigns to each variable in $Var(\phi)$ a value, it follows that $X_0, X_1, \ldots, X_n$ is a partitioning of $Var(\phi)$.
– Suppose that $(x = y) \in \psi$. Since $Val \models \psi$ it follows that $Val(x) = Val(y)$. By definition it follows that x and y will belong to the same X_i.
– Suppose that $(x <_c y) \in \psi$. Since $Val \models \psi$ it follows that $Val(x)+c < Val(y)$. By definition it follows that $x \in X_i$ and $y \in X_j$ for some $i < j$. Also, by definition of d_i we know that $Val(y) - Val(x) = \left(\sum_{k=i+1}^{j} d_k\right) + j - i$. This implies $c \leq \left(\sum_{k=i+1}^{j} d_k + 1\right)$

Let $\phi_{fl} = B_0\ d_1\ B_1\ d_2\ \cdots\ d_n\ B_n$ be the flattening of ϕ induced by F. We show that $\gamma \models \phi_{fl}$. Let $c_0, \ldots, c_n$ be such that $c_i = Val(x)$ for some $x \in X_i$.

– By definition, we know that $c_{i-1} = Val(x)$ for some $x \in X_{i-1}$ and $c_i = Val(y)$ for some $x \in X_i$. This means that $c_i - c_{i-1} = d_{i+1} + 1$, i.e. , $c_i - c_{i-1} > d_{i+1}$.
– $\gamma(p(c_i)) \geq \sum_{x \in X_i} T(p(x)) = B_i(p)$.

(if) Let $\phi_{fl} = B_0\ d_1\ B_1\ d_2\ \cdots\ d_n\ B_n$ be a flattening of ϕ, induced by a flattening $F = X_0\ d_1\ X_1\ d_2\ \cdots\ d_n\ X_n$ of $Var(\phi)$. Suppose $\gamma \in [\![\phi_{fl}]\!]$. We show that $\gamma \in [\![\phi]\!]$, i.e., we show that there is a valuation Val of $Var(\phi)$ such that $Val \models \psi$ and $\gamma \geq Val(T)$.

Since $\gamma \in [\![\phi]\!]$ if there are natural $c_1, \ldots, c_n \in \mathbb{N}$ such that

– $c_i - c_{i-1} > d_i$ for each $i : 1 \leq i \leq n$; and
– $\gamma(p(c_i)) \geq B_i(p)$ for each $i : 0 \leq i \leq n$.

Define Val such that $Val(x) = c_i$ if $x \in X_i$.

First, we show that $Val \models \psi$.

– If $(x = y) \in \psi$ then, since F is a flattening of $Var(\phi)$, we know that $x, y \in X_i$ for some $i : 1 \leq i \leq n$. By definition of Val it follows that $Val(x) = c_i = Val(y)$.
– If $(x <_c y) \in \psi$ then, since F is a flattening of $Var(\phi)$, we know that $x \in X_i$ and $y \in X_j$ for some $i < j$. with $c \leq \left(\sum_{k=i+1}^{j} d_k + 1\right)$. We also know that $c_j - c_i \geq \left(\sum_{k=i+1}^{j} d_k\right) + j - i$. This means that $c_j - c_i > c$. By definition of Val it follows that $Val(y) - Val(x) = c_j - c_i > c$.

Now, we show that $\gamma \geq Val(T)$. By definition of Val we know that $Val(T)(p(c)) = 0$ if $c \neq c_0, \ldots, c_n$. Otherwise, we have $Val(T)(p(c_i)) = \sum_{x \in X_i} T(p(x)) = B_i(p) \leq \gamma(p(c_i))$.

Theorem 2

A constraint $T : \psi$ is said to be in *normal form* if ψ satisfies the conditions of Section 4, together with the following conditions:

- if $(x <_{c_1} y) \in \psi$ and $(y < c_2) \in \psi$ then $(x < c_3) \in \psi$ for some c_3 with $c_2 - c_1 > c_3$.
- if $(x <_{c_1} y) \in \psi$ and $(c_2 < x) \in \psi$ then $(c_3 < y) \in \psi$ for some c_3 with $c_1 + c_2 < c_3$.
- if $(x <_{c_1} y) \in \psi$ and $(x = c_2) \in \psi$ then $(c_1 + c_2 < y) \in \psi$.
- if $(x <_{c_1} y) \in \psi$ and $(y = c_2) \in \psi$ then $(x < c_2 - c_1) \in \psi$.
- if $(x = y) \in \psi$ and $(c < y) \in \psi$ then $(c < x) \in \psi$.
- if $(x = y) \in \psi$ and $(y = c) \in \psi$ then $(x = c) \in \psi$.
- if $(x = y) \in \psi$ and $(y < c) \in \psi$ then $(x < c) \in \psi$.

Lemma 1 still holds, i.e., for each constraint ϕ we can effectively compute a constraint ϕ_{norm} in normal form such that $[\![\phi]\!] = [\![\phi_{norm}]\!]$. Conjuncts of the form $x = c$ or $x < c$ which are added by the normalization procedure satisfy the property that $c \leq cmax$.

Checking entailment and computing predecessors can be carried out in the same manner as in Section 4. Again, conjuncts of the form $x = c$ or $x < c$ which are added when computing *Pre* satisfy the condition that $c \leq cmax$.

To prove termination, we modify the definition of flat constraints.

A *flat constraint* ϕ_{fl} is a of the form

$$D_1 \ \cdots D_{cmax} \ d_0 \ B_0 \ d_1 \ B_1 \ d_2 \ \cdots \ d_n \ B_n$$

where $D_0, \ldots, D_{cmax}, B_0, \ldots, B_n \in \mathbb{P}^{\otimes}$ and $d_0, d_1, \ldots, d_n \in \mathbb{N}$. Here we allow the multisets $D_1 \ \cdots D_{cmax}$ to be empty. For a configuration γ we have $\gamma \in [\![\phi_{fl}]\!]$ if there are natural $c_0, \ldots, c_n \in \mathbb{N}$ such that the following conditions are satisfied:

- $c_0 > d_0 + cmax$, and $c_i - c_{i-1} > d_i$ for each $i : 1 \leq i \leq n$
- $\gamma(p(i)) \geq D_i(p)$ for each predicate symbol p and $i : 0 \leq i \leq cmax$. Also, $\gamma(p(c_i)) \geq B_i(p)$ for each predicate symbol p and $i : 0 \leq i \leq n$.

For constraints

$$\phi_{fl} = D_1 \ \cdots D_{cmax} \ d_0 \ B_0 \ d_1 \ B_1 \ d_2 \ \cdots \ d_m \ B_m$$

and

$$\phi'_{fl} = D_1 \ \cdots D_{cmax} \ d_0 \ B'_0 \ d'_1 \ B'_1 \ d'_2 \ \cdots \ d'_n \ B'_n$$

it is the case that $\phi_{fl} \sqsubseteq \phi'_{fl}$ iff there is a strictly monotone injection h form $\{1, \ldots, m\}$ to $\{1, \ldots, n\}$ such that the conditions of Section 5 are satisfied and that $D_i \leq D'_i$ for each $i : 1 \leq i \leq cmax$.

Consider a constraint $\phi = T : \psi$ in normal form. A *flattening* F of $Var(\phi)$ is a word of the form

$$Y_0 \ Y_1 \ \cdots \ Y_{cmax} \ X_0 \ d_1 \ X_1 \ d_2 \ \cdots \ d_n \ X_n$$

where $X_1, \ldots, X_n$ and $d_1, \ldots, d_n$ satisfy the same conditions as in Section 5, and the following additional conditions are satisfied:

- $Y_0, Y_1, \ldots, Y_{cmax}, X_0, X_1, \ldots, X_n$ is a partitioning of $Var(\phi)$.
- If $(x = y) \in \psi$ then either $x, y \in Y_i$ for some $i : 1 \leq i \leq cmax$ or $x, y \in X_i$ for some $i : 1 \leq i \leq n$.
- If $(x <_c y) \in \psi$, then one of the following holds:
  - $x \in X_i$, $y \in X_j$, and $c \leq \left(\sum_{k=i+1}^{j} d_k + 1\right)$.
  - $x \in Y_i$, $y \in X_j$, and $c \leq \left(\sum_{k=0}^{j} d_k + 1\right) + cmax - i$.
  - $x \in Y_i$, $y \in Y_j$, and $c \leq j - i$.
- If $(c < x) \in \psi$ then one of the following holds:
  - $x \in X_i$ and $c \leq \left(\sum_{k=0}^{j} d_k + 1\right) + cmax - i$.
  - $x \in Y_i$ and $c < i$.
- If $(x = c) \in \psi$ then $x \in Y_c$.
- If $(x < c) \in \psi$ then $x \in Y_i$. fro some $i < c$.

The *flattening* ϕ_{fl} of ϕ *induced* by F is a flat constraint $D_1 \ \cdots D_{cmax}\ d_0\ B_0\ d_1\ B_1\ d_2\ \cdots\ d_n\ B_n$ where B_i satisfies the same condition as in Section 5 and:

- $D_i(p) = \sum_{x \in Y_i} T(p(x))$ for each $p \in \mathbb{P}$ and $i : 1 \leq i \leq cmax$.

In a similar manner to Lemma 5, we can show that each constraint can be translated to the union of a finite set of flat constraints. The proof can be carried out in a similar fashion, adding new cases corresponding to the additional types of conjuncts in conditions. This implies that Lemma 6 holds also for the new types of constraints introduced in this section.

Theorem 3 and Theorem 4

First,we prove Theorem 3.

We recall the model of a *2-counter machine (CM)* which is pair (Q, δ), where Q is a finite set of states, and δ is the transition function. A transition is of the form (q_1, op, q_2), where $q_1, q_2 \in Q$, and op is either an *increment* (of the form $cnt_1{+}{+}$ or $cnt_2{+}{+}$); a *decrement* (of the form $cnt_1{-}{-}$ or $cnt_2{-}{-}$); or a *zero-testing* (of the form $cnt_1 = 0?$ or $cnt_2 = 0?$). A *lossy counter machine (LCM)* is of the same form as a counter machine. The difference in semantics is in the zero-testing operation. More precisely, the zero-testing of cnt_1 is simulated by resetting the value cnt_1 to zero, and decreasing the value of cnt_2 by an arbitrary natural number (possibly 0). The zero-testing of cnt_2 is performed in a similar manner.

Assume an LCM $\mathcal{M} = (Q, \delta)$. We shall construct a CMRS $\mathcal{S}$ which will simulate $\mathcal{M}$. The simulation of $\mathcal{M}$ occurs in a sequence of *phases*. During each phase, $\mathcal{S}$ simulates increment and decrement transitions of $\mathcal{M}$. Each phase is indexed by a natural number which is incremented at the end of the phase. As soon as $\mathcal{M}$ performs a zero-testing of a counter, $\mathcal{S}$ enters an intermediate stage. After conclusion of the intermediate stage, a new phase is started and the index phase is increased.

The set of predicates symbols in $\mathcal{S}$ is divided into three groups:

– Two nullary predicate symbols q and q' for each $q \in Q$. We use q' during the intermediate stages of the simulation.
– Two predicate symbols cnt_1 and cnt_2, which encode the values of cnt_1 and cnt_2 respectively.
– A unary predicate p whose argument carries the index of the current phase. Furthermore, we use a predicate symbol p' to store the index of the previous phase during the intermediate stages of the simulation.

A configuration of $\mathcal{S}$ contains, during the a given phase of the simulation, the following ground terms:

– A term of the form q which encodes the current state of $\mathcal{M}$.
– A term of the form $p(c)$ where c is the index of the current phase.
– Terms of the form $cnt_1(c)$ where c is the index of the current phase. The number of such terms encodes the current value of cnt_1. There are also a number of terms of the form $cnt_1(d)$ where d is not equal to the the index of the current phase. Such terms are redundant and do not affect the encoding. Similar terms exist to encode cnt_2.

An increment transition $(q, cnt_1 + +, q_2) \in \delta$ is simulated by a rule of the form

$$[q_1\ ,\ p(x)] \rightsquigarrow [q_2\ ,\ p(x)\ ,\ cnt_1(x)] \quad :\quad true$$

We increase the value counter cnt_1 by adding one more term whose predicate symbol is cnt_1 and whose argument is equal to the index of the current phase.

A decrement transition $(q, cnt_1 - -, q_2) \in \delta$ is simulated by a rule of the form

$$[q_1\ ,\ p(x)\ ,\ cnt_1(x)] \rightsquigarrow [q_2\ ,\ p(x)] \quad :\quad true$$

We decrease the value counter cnt_1 by removing one of the corresponding terms from the configuration. Observe that terms whose arguments are less than index of the current phase are not used, and hence they do affect the encoding.

A transition $(q, cnt_1 = 0?, q_2) \in \delta$ is simulated by the following three rules:

$$\begin{array}{rcll} [q_1\ ,\ p(x)\ ,\ p'(x)] & \rightsquigarrow & [q_1'\ ,\ p(y)\ ,\ p'(x)] & :\ x < y \\ [q_1'\ ,\ cnt_2(x)\ ,\ p(y)\ ,\ p'(x)] & \rightsquigarrow & [q_1'\ ,\ cnt_2(y)\ ,\ p(y)\ ,\ p'(x)] & :\ true \\ [q_1'\ ,\ p(y)\ ,\ p'(x)] & \rightsquigarrow & [q_2\ ,\ p(y)\ ,\ p'(y)] & :\ true \end{array}$$

We enter the intermediate phase by changing from q_1 to q_1'. We store the current index using p', and generate a new index which is strictly larger than the current one. This resets counter cnt_1 since all terms in its encoding will now have too small arguments. Finally, we change the arguments of (some of) the terms encoding cnt_2 to the new index. Here, not all such terms may receive new arguments, and hence the value cnt_2 may "unintentionally" be reduced. We use *redundant terms* to refer to terms which have either cnt_1 or cnt_2 as predicate symbol, and whose arguments are smaller than the current index,

Mayr shows in [21] undecidability of the *repeated state reachability* problem for LCM: given a lossy counter machine, and two states q_{init} and q_{fin}, check

whether there is a computation starting from q_{init} (with both counter values being equal to zero) and visits q_{fin} infinitely often.

Using an idea similar to that used by De Frutos et al [12] we can extend the proof to show Theorem 4 as follows. The key observation here is that redundant terms are not removed during the simulation procedure described above. This means that any reachable configuration which does not contain redundant terms are generated by computations of the perfect (non-lossy) counter machines. We add the following rules to our CMRS:

$$\begin{array}{rcll} [q_{fin}\ ,\ cnt_1(x)] & \rightsquigarrow & [q'_{fin}] & : \ true \\ [q'_{fin}\ ,\ p(x)\ ,\ cnt_1(x)] & \rightsquigarrow & [q'_{fin}\ ,\ p(x)] & : \ true \\ [q'_{fin}\ ,\ p(x)\ ,\ cnt_2(x)] & \rightsquigarrow & [q'_{fin}\ ,\ p(x)] & : \ true \\ [q'_{fin}\ ,\ p(x)] & \rightsquigarrow & [q'_{fin}] & : \ true \end{array}$$

In other words, if we reach a configuration where $\mathcal{M}$ is in q_{fin}, then $\mathcal{S}$ initiates a procedure where it moves to q'_{fin} and then erases the terms corresponding to the counters. However, we only remove such terms if there arguments are equal to the index of the current phase. This implies that the configuration $[q'_{fin}]$ in $\mathcal{S}$ iff q_{fin} is reachable in the perfect CM.

Theorem 5

We show undecidability by reducing the reachability problem for Turing Machines. Assume a Turing Machine $\mathcal{T} = (Q, \Sigma, \delta)$ where Q is the set of states, Σ is the alphabet (including the empty symbol $\perp$), and δ is the transition function. Let $q_{init}, q_{fin} \in Q$. Suppose that all executions of $\mathcal{T}$ start from q_{init} with an empty tape. We construct an CMRS $\mathcal{S}$ and define an instance of the coverability problem which has a positive answer if and only if q_{fin} is reachable in $\mathcal{T}$.

The set of predicate symbols in $\mathcal{S}$ consists of the following:

- A nullary predicate symbols q, for each $q \in Q$.
- Two binary predicate symbols a and $a^\bullet$ for each $a \in \Sigma$.
- Two binary predicates *start* and $start_1$ which are used to initiate the encoding process.

We use the configurations of $\mathcal{S}$ to encode both the state and the tape of $\mathcal{T}$. A ground term $a(c_1, c_2)$ (in a configuration of $\mathcal{S}$) will encode one square of the tape. Intuitively, the predicate symbol a represents the alphabet symbol in that particular square. We use the arguments c_1 and c_2 to define the ordering of the squares along the tape. More precisely, the left argument c_1 is equal to the right argument of the term corresponding the square to the left, while c_2 is equal to the left argument of the term corresponding the square to the right. Symbols of the form $a^\bullet$ indicate the current position of the head. In other words, a term of the form $a^\bullet(c, d)$ can be explained as above, except that we also know that the head is pointing to the current square.

The CMRS $\mathcal{S}$ contains three groups of rules:

– *Initialization Rules*:

$$\begin{array}{rcll} [start(x,y)] & \rightsquigarrow & [\bot^{\bullet}(x,y)\ ,\ start_1(y,z)] & :\ y<z \\ [start_1(x,y)] & \rightsquigarrow & [\bot(x,y)\ ,\ start_1(y,z)] & :\ y<z \\ [start_1(x,y)] & \rightsquigarrow & q_{init} & :\ true \end{array}$$

The first rule generates the first empty square and let the rule point to it, indicated by the predicate symbol $\bot^{\bullet}$. The second rule generates an arbitrary number of empty squares. These are modelled by terms whose predicate symbol is $\bot$. Observe that the values of the arguments are strictly increasing (ensured by the condition $y < z$), and that the left argument of each square is equal to the right argument of the square generated in the previous step. The third rule of the group nondeterministically ends this procedure, and generates the initial state of $\mathcal{T}$ starting the simulation phase.

– *Simulation Rules*: For each transition of the form $(q_1, a_1/a_2, H, q_2) \in \delta$ and each $a_3 \in \Sigma$, we have a rule:

$$[q_1\ ,\ a_1^{\bullet}(x,y)\ ,\ a_3(y,z)] \rightsquigarrow [q_1\ ,\ a_2(x,y)\ ,\ a_3^{\bullet}(y,z)]\ :\ true$$

This rule changes the current state to q_2, changes a_1 to a_2, and moves the head one step to the right. A square recognizes its right number since its right argument is equal to the left argument of the neighbour (encoded by having the same variable y in the corresponding terms inside the rule). Notice the ordering of the squares is not changed by the rule.
For each transition of the form $(q_1, a_1/a_2, L, q_2) \in \delta$ and each $a_3 \in \Sigma$, we have a rule:

$$[q_1\ ,\ a_3(x,y)\ ,\ a_1^{\bullet}(y,z)] \rightsquigarrow [q_2\ ,\ a_3^{\bullet}(x,y)\ ,\ a_2(y,z)]$$

The rule can be explained in a similar manner to the one above.

Now, we define the required instance of the coverability problem by taking γ_{init} to be any configuration of the form $[start(c_1, c_2)]$ with $c_1 < c_2$, and taking $\gamma_{fin} = [q_{fin}]$.

Lemma 7

We give a CMRS $\mathcal{S}$ with $L(\mathcal{S}) = L_{PAL}$. The set of predicate symbols which appear in $\mathcal{S}$ consists of (i) a predicate symbol a for each $a \in \Sigma$, and (ii) two special symbols *guess* and *check*. We have three sets of rules in $\mathcal{S}$:

– For each $a \in \Sigma$, we have a rule which is labelled with a and which is of the form

$$[guess(x)] \rightsquigarrow [guess(y)\ ,\ a(x)]\ :\ \{x < y\}$$

Rules of this form are used to guess the initial part w_1 of the word. We keep track of the ordering among symbols inside w_1 through their arguments. These arguments are strictly increasing by definition of the rule.

– A rule of the form:

$$[guess(x)] \rightsquigarrow [check(x)] \ : \ \{true\}$$

This rule is used to switch mode from *guessing* to *checking*.

– For each $a \in \Sigma$, we have a rule which is labelled with a and which is of the form

$$[check(y)\ ,\ a(y)] \rightsquigarrow [check(x)] \ : \ \{x < y\}$$

This rule is used to check whether the second part of the word is a subword of w_1^R. This is done by traversing the symbols in the reverse direction (following their arguments backwards), possibly jumping over some of the symbols.

We define $\gamma_{init} = [r(0)]$ and $p_{acc} = check$.

Theorem 7

Assume a relational automaton $\mathcal{A} = (Q, \delta)$ which operates on a set X of variables. Fix initial and accepting states $q_{init}, q_{acc} \in Q$. A *variable state* v is mapping form X to $\mathbb{N}$. A *configuration* is of the form (q, v), where $q \in Q$ and v is a variable state. We define γ_{init} to be (q_{init}, v_{init}) where $v_{init}(x) = 0$ for all $x \in X$. A configuration γ_{acc} is said to be *accepting* if it is of the form (q_{acc}, v). For a transition $\rho \in \delta$ of the form (q_1, op, q_2), we let $\gamma_1 \xrightarrow{\rho} \gamma_2$ if $\gamma_1 = (q_1, v_1)$, $\gamma_2 = (q_2, v_2)$, and one of the following holds:

– op is of the form $read(x)$ and $v_2(y) = v_1(y)$ for each $y \in X - \{x\}$.
– op is of the form $y := x$, $v_2(z) = v_1(z)$ for each $z \in X - \{y\}$, and $v_2(y) = v_1(x)$.
– op is of the form $x < y$, $v_2 = v_1$, and $v_1(x) < v_1(y)$. Other testing operations are defined in a similar manner.

For configurations γ, γ' and a sequence $\sigma = \rho_1 \cdots \rho_n$ of transitions, we use $\gamma \stackrel{\sigma}{\Longrightarrow} \gamma'$ to denote that there are $\gamma_0, \gamma_1, \ldots, \gamma_n$ such that $\gamma = \gamma_0 \xrightarrow{\rho_1} \gamma_1 \xrightarrow{\rho_2} \gamma_2 \cdots \xrightarrow{\rho_n} \gamma_n = \gamma'$.

We assume a labelling function λ from transitions to the finite alphabet Σ. We extend λ to sequences of transitions such that $\lambda(\rho_1 \cdots \rho_n) = \lambda(\rho_1) \cdots \lambda(\rho_n)$. We define $L(\mathcal{A})$ to be the set of finite words w such that there is a sequence σ of transitions where

– $\gamma_{init} \stackrel{\sigma}{\Longrightarrow} \gamma_{acc}$ for some accepting configuration γ_{acc}; and
– $\lambda(\sigma) = w$.

We perform the proof in two steps, namely we show that

– for each relational automaton $\mathcal{A}$, there is a CMRS $\mathcal{S}$ with $L(\mathcal{S}) = L(\mathcal{A})$.
– there is no relational automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L_{PAL}$.

For a relational automaton $\mathcal{A}$ we derive a CMRS $\mathcal{S}$ with $L(\mathcal{S}) = L(\mathcal{A})$. The set of predicate symbols in $\mathcal{S}$ consists of the following: (i) for each $q \in Q$, there is a nullary predicate symbol q in $\mathcal{S}$; and (ii) for each variable x in $\mathcal{A}$, there is a predicate symbol q_x in $\mathcal{S}$. Furthermore, for each transition in $\mathcal{A}$, labelled with $a \in \Sigma$, there is a corresponding rule in $\mathcal{S}$ which is also labelled with a and which has the following form:

- $(q_1, \mathit{read}(x), q_2)$: $[q_1\ ,\ p_x(z)] \rightsquigarrow [q_2\ ,\ p_x(w)]\ :\ \{\mathit{true}\}$
- $(q_1, x := y, q_2)$: $[q_1\ ,\ p_x(z)\ ,\ p_y(w)] \rightsquigarrow [q_2\ ,\ p_x(w)\ ,\ p_y(w)]\ :\ \{\mathit{true}\}$
- $(q_1, x < y, q_2)$: $[q_1\ ,\ p_x(z)\ ,\ p_y(w)] \rightsquigarrow [q_2\ ,\ p_x(z)\ ,\ p_y(w)]\ :\ \{z < w\}$

Other testing operations can be encoded in a similar manner.

Now, we show that there is no relational automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L_{PAL}$. First, we state some definitions and properties. To do that, we fix a relational automaton $\mathcal{A} = (Q, \delta)$ operating on a set of variables X. Let cmax be the maximum natural number which appears in the transitions of $\mathcal{A}$.

For a variable state v, we define $\mathit{index}(v)$ of v to be a word of the form

$$X_0\ X_1\ \cdots\ X_{\mathit{cmax}}\ d_1\ X_{\mathit{cmax}+1}\ d_2\ X_{\mathit{cmax}+2}\ d_2\ \cdots\ d_n\ X_{\mathit{cmax}+n}$$

where

- $X_0, \ldots, X_{\mathit{cmax}+n}$ are disjoint subsets of X. X_i may not be empty if $i > \mathit{cmax}$.
- $X = X_0 \cup \cdots \cup X_{\mathit{cmax}+n}$.
- $d_1, \ldots, d_n \in \mathbb{N}$
- $x \in X_i$ for some $i : 0 \leq i \leq \mathit{cmax}$ iff $v(x) = i$.
- if $x \in X_i$, $y \in X_j$, $1 \leq i \leq \mathit{cmax}$, and $\mathit{cmax} < j \leq \mathit{cmax} + n$ then $v(y) - v(x) \geq \left(\sum_{k=\mathit{cmax}+1}^{j} d_k + 1\right) + \mathit{cmax} - i$
- if $x \in X_i$, $y \in X_j$, and $\mathit{cmax} < i, j \leq \mathit{cmax} + n$ then $v(y) - v(x) \geq \left(\sum_{k=i+1}^{j} d_k + 1\right)$

We define an ordering $\preceq$ on variable states such that $v \preceq v'$ iff

$$\mathit{index}(v) = X_0\ X_1\ \cdots\ X_{\mathit{cmax}}\ d_1\ X_{\mathit{cmax}+1}\ d_2\ X_{\mathit{cmax}+2}\ \cdots\ d_n\ X_{\mathit{cmax}+n}$$

$$\mathit{index}(v') = X_0\ X_1\ \cdots\ X_{\mathit{cmax}}\ d'_1\ X_{\mathit{cmax}+1}\ d'_2\ X_{\mathit{cmax}+2}\ \cdots\ d'_n\ X_{\mathit{cmax}+n}$$

and $d_i \leq d'_i$ for each $i : \mathit{cmax}+1 \leq i \leq \mathit{cmax}+n$. We extend $\preceq$ to configurations such that $(q_1, v_1) \preceq (q_2, v_2)$ if $q_1 = q_2$ and $v_1 \preceq v_2$. Using a similar reasoning to Lemma 4 we can show that $\preceq$ is a WQO.

To prove the theorem we need the following property: If $\gamma_{\mathit{init}} \stackrel{\sigma}{\Longrightarrow} \gamma_1$ then $\gamma_{\mathit{init}} \stackrel{\sigma}{\Longrightarrow} \gamma_2$ for each γ_2 with $\gamma_1 \preceq \gamma_2$. We show this by induction on the length of σ. The base case is trivial. For the induction step, suppose that $\gamma_{\mathit{init}} \stackrel{\sigma}{\Longrightarrow} \gamma'_1 \stackrel{\rho}{\longrightarrow} \gamma_1$. Let $\gamma_1 \preceq \gamma_2$. We show that $\gamma_{\mathit{init}} \stackrel{\sigma}{\Longrightarrow} \gamma'_2 \stackrel{\rho}{\longrightarrow} \gamma_2$ for some γ'_2. Let $\gamma'_1 = (q'_1, v'_1)$, $\gamma_1 = (q_1, v_1)$, $\gamma_2 = (q_1, v_2)$, and $\rho = (q'_1, \mathit{op}, q_1)$. We consider three cases depending on the form op:

– If op is of the form $read(x)$. Let

$$index(v_1') = \ X_0 \ \cdots \ X_{cmax} \ d_1 \ X_{cmax+1} \ \cdots \ d_n \ X_{cmax+n}$$

There are several cases depending on the set to which x belongs, before and after performing the reading operation. We consider one of the cases (the other cases can be explained in a similar manner), namely the case where $X_{cmax+i} = \{x\}$ and $v_1(x) = v_1'(y)$ for some $y \in X_j$ and $i : 1 \leq i < j < n$. We know that we can derive $index(v_1)$ from $index(v_1')$ by replacing the sequence

$$X_{cmax+i-1} \ d_i \ X_{cmax+i} \ d_{i+1} \ X_{cmax+i+1}$$

by the sequence

$$X_{cmax+i-1} \ d_i + d_{i+1} \ X_{cmax+i+1}$$

and also replacing the set X_j by the set $X_j \cup \{x\}$. Since $\gamma_1 \preceq \gamma_2$ it follows that $index(v_2)$ is of the same form as $index(v_1)$ except that each d_k is replaced some d_k' with $d_k' \geq d_k$. By the induction hypothesis, we know that $\gamma_{init} \stackrel{\sigma}{\Longrightarrow} \gamma_2'$ for each γ_2' with $\gamma_1' \preceq \gamma_2'$. Take any such a γ_2'. Let $\gamma_2' = (q_1', v_2')$. The index of v_2' is of the same form as v_1' except that d_k is replaced some d_k'' with $d_k'' \geq d_k$. In particular, we can pick γ_2' such that $d_k'' = d_k'$ for each $k : 1 \leq k \leq n$. Obviously $\gamma_2' \stackrel{\rho}{\longrightarrow} \gamma_2$.

– op is of the form $y := x$. Similar to the above case.
– op is of the form $x < y$. By the induction hypothesis, we know that $\gamma_{init} \stackrel{\sigma}{\Longrightarrow} \gamma_2'$ for some γ_2' such that the indices of $\gamma_1', \gamma_2', \gamma_1, \gamma_2$ are related in a similar manner as in the first case. Since $\gamma_1' \preceq \gamma_2'$ and $v_1'(x) < v_1'(y)$ it follows that $v_2'(x) < v_2'(y)$ and hence $\gamma_2' \stackrel{\rho}{\longrightarrow} \gamma_2$.

Having shown the above property, we turn back to the main result. Suppose that there is a relational automaton $\mathcal{A}$ which accepts L_{PAL}. We show that we get a contradiction. Take any symbols $a, b \in \Sigma$. Consider an infinite sequence of words $baab, b^2aab^2.b^4aab^4, \ldots$. All these words belong to L_{PAL} and are hence accepted by $\mathcal{A}$. This means that there are configurations $\gamma^1, \gamma_{acc}^1, \gamma_2, \gamma_{acc}^2, \ldots$ and sequences of transitions $\sigma_1, \sigma_1', \sigma_2, \sigma_2', \ldots$ such that for each i we have

– $\lambda(\rho_i) = b^i a$ and $\lambda(\rho_i') = ab^i$.
– $\gamma_{init} \stackrel{\sigma_i}{\Longrightarrow} \gamma^i \stackrel{\sigma_i'}{\Longrightarrow} \gamma_{acc}^i$
– γ_{acc}^i is accepting.

Since $\preceq$ is a WQO, it follows that there are j and k with $j < k$ and $\gamma^j \preceq \gamma^k$. This means (by the property shown above) that $\gamma_{init} \stackrel{\sigma_j}{\Longrightarrow} \gamma^k$ and hence $\mathcal{A}$ accepts the word $b^j aab^k$ which is a contradiction

1. $[\ init\ ] \rightarrow [\ v_0(X), initP(Id)\ ] : true$
2. $[\ initP(Id)\ ] \rightarrow [\ idle(Id), initP(Next)\ ] : Next > Id$
3. $[\ idle(X), v_0(Y)\ ] \rightarrow [\ waiting(X), v_0(X)\ ] : true$
4. $[\ v_0(X)\ ] \rightarrow [\ v_1(X)\ ] : true$
5. $[\ waiting(X), v_1(Y)\ ] \rightarrow [\ idle(X), v_1(Y)\ ] : X > Y$
6. $[\ waiting(X), v_1(Y)\ ] \rightarrow [\ idle(X), v_1(Y)\ ] : X < Y$
7. $[\ waiting(X), v_1(X)\ ] \rightarrow [\ cs(X), v_1(X)\ ] : true$
8. $[\ cs(X), v_1(Y)\ ] \rightarrow [\ idle(X), v_0(Y)\ ] : true$
9. $[\ idle(X)\ ] \rightarrow [\ ] : true$

Fig. 1. CMRS model of the Mutual Exclusion Protocol.

Details of Case Studies

Description of the Mutual Exclusion Protocol (Fig. 1)

In our CMRS model we use natural numbers to model process identities. A process is represented by a term $q(i)$ where q is the state and i is the identity of the process. The shared variable is represented by a term of similar form. Here q is the mode and i is the process identity currently assigned to it. The resulting monadic CMRS model consists of the 9 rules shown in Fig. 1. Violations to mutual exclusion can be represented symbolically via the constraint $cs(X), cs(Y) : true$ in which X and Y are unrelated. The initial configuration, namely $[init]$, is not part of the resulting sets of constraints. This proves that mutual exclusion holds for any number of processes.

Rule 1 defines the initial configuration $[init]$ of the system. Rule 2 dynamically generates new processes with distinct identities. Rules 3 models the transition from *idle* to *waiting* for a process with identifier X. The value of the global variable will now be updated to X. Rule 4 models the changing of mode to v_1. Rules $5 - 6$ model the fact that the processes with "wrong identities" are sent back to *idle*. Rule 7 models entering the critical section. The mode should be v_1 and the identity of the process should be the same as the one held by the global variable. Rule 8 models leaving the critical section. Here, the mode changes back to v_0. Rule 9 means that processes may be dynamically deleted.

Description of the FCS Models (Fig. 2 and Fig. 3)

For this protocol, we have defined two different CMRS models. In both models we model identifiers using natural numbers. The first model (shown in Fig. 2) is based on the original version of the protocol which assumes that the set of locations is finite. Monadic predicates like $waitC_i(id)$ can be used the to represent an agent with control state $waitC$, residing in location i, and storing id in his local memory. The resulting monadic model has 10 rules.

The second model (shown in Fig. 3) is parametric on the number of locations, identifiers, and agents. For instance, the agent state $waitC_i(id)$ is represented here as the dyadic predicate $waitC(i, id)$.

1. $[\ idleF_i, reqC_{i,k}, last(C)\ ] \rightarrow [\ goF_{i,j}(N), ackC_{i,j,k}(N), last(L)\ ] : L > N > C$
2. $[\ goF_{i,j}(N)\ ] \rightarrow [\ passF_{j,i}(N)\ ] : true$
3. $[\ passF_{j,i}(Id), idleS_j\ ] \rightarrow [\ gohomeF_{j,i}, waitS_j, storeS_j(Id)\ ] : true$
4. $[\ gohomeF_{j,i}\ ] \rightarrow [\ idleF_i\ ] : true$
5. $[\ waitS_j, loginC_{j,k}(Id)\ ] \rightarrow [\ checkS_j(Id), authC_{j,k}\ ] : true$
6. $[\ checkS_j(Id), storeS_j(Id), waitC_{j,k}\ ] \rightarrow [\ idleS_j, used(Id), servedC_{j,k}\ ] : true$
7. $[\ idleC_k\ ] \rightarrow [\ reqC_{i,k}\ ] : true$
8. $[\ ackC_{i,j,k}(Id)\ ] \rightarrow [\ loginC_{j,k}(Id)\ ] : true$
9. $[\ authC_{j,k}(Id)\ ] \rightarrow [\ waitC_{j,k}(Id)\ ] : true$
10. $[\ servedC_{j,k}\ ] \rightarrow [\ idleC_k\ ] : true$

Fig. 2. Model for the FSC system.

For both models we have tested the coverability problem for the constraint $used(X), used(Y) : X = Y$ with the aim of proving that each request identifier can be used only once. The last experiments shows that the property holds for any number of forwarders, providers and clients.

The forwarder can be in one of the following states $idleF_l$, $goF_{l,m}$, $passF_{l,m}$, and $gohomeF_{l,m}$ for $l, m \in \mathcal{L}$. $idleF_i$ is the initial state of a forwarder residing in location i. To define the forwarder agents, we have to make some assumptions on the clients. More precisely, let us suppose that the predicate $reqC_{i,k}$ represents a client (with home location k) requesting a service to the forwarder. Furthermore, let $ackC_{i,j,k}(N)$ denote the state of the client after having received the provider location j and the new identifier N. Rule (1) of Fig. 2 models the synchronization of a forwarder with a client residing in the same location. As explained before, we use *last* to ensure that the generated identifier is fresh. In state $goF_{i,j}(N)$ the forwarder can move to location j via rule (2) that swaps i and j. Location i is maintained in order to allow the forwarder to return home. Let us assume that the service provider is initially in state $idleS_j$. In state $passF_{j,i}(N)$ the forwarder can pass the identifier to the provider. using rule (3). The provider stores the identifier using predicate $storeS_j$, and then waits for the login of a client in state $waitS_j$. Finally, the forwarder moves back to the home location using rule (4).

Let us now discuss the authentication phase. We assume that clients have a special transition from state $loginC_{j,k}(Id)$ to state $authC_{j,k}$ for the login phase. When a client logins using an identifier $Id : csdetails.tex, v1.42005/10/1922 : 14 : 06paroshExp$ the provider stores it using predicate $checkS_j(Id)$. This step is modelled via rule (5). In state $checkS_j(Id)$ the server compares $Id : csdetails.tex, v1.42005/10/1922 : 14 : 06paroshExp$ with the identifier received from the forwarder. If they coincide, then the provider grants the service to the client in state waiting for it in state $waitC_{j,k}$ as shown in rule (6). The server stores the used identifier in the predicate $used(Id)$. Notice that equalities are left implicit in all previous rules by allowing different atoms to share the same variables.

1. $[init] \rightarrow [\ last(N)\ ]$ 2. $\epsilon \rightarrow [\ idleF(I)\ ]$ 3. $\epsilon \rightarrow [\ idleS(J)\ ]$ 4. $\epsilon \rightarrow [\ idleC(K)\ ]$
5. $[\ idleF(I), reqC(I,K), last(C)\ ] \rightarrow$
$[\ goF(I,J,N), ackC(I,J,K,N), last(L)\ ]\ :\ L > N, N > C$
6. $[\ goF(I,J,N)\ ] \rightarrow [\ passF(J,I,N)\ ]\ :\ true$
7. $[\ passF(J,I,Id), idleS(J)] \rightarrow [\ gohomeF(J,I), waitS(J), storeS(J,Id)\ ]\ :\ true$
8. $[\ gohomeF(J,I)\ ] \rightarrow [\ idleF(I)\ ]\ :\ true$
9. $[\ waitS(J), loginC(J,K,Id)\ ]\ \rightarrow\ [\ checkS(J,Id), authC(J,K)\ ]\ :\ true$
10. $[\ checkS(J,Id), storeS(J,Id), waitC(J,K)\ ] \rightarrow$
$[\ idleS(J), used(Id), servedC(J,K)\ ] : true$
11. $[\ idleC(K)\ ] \rightarrow [\ reqC(I,K)\ ] : true$
12. $[\ ackC(I,J,K,Id)\ ] \rightarrow [\ loginC(J,K,Id)\ ] : true$
13. $[\ authC(J,K,Id)\ ] \rightarrow [\ waitC(J,K,Id)\ ] : true$
14. $[\ servedC(J,K)\ ] \rightarrow [\ idleC(K)\ ] : true$

Fig. 3. Generalized CMRS model for the FSC system of [27].

To complete the specification of a client we need to add rules to pass from the initial state $idleC_k$ to the interaction with the forwarder and with the provider. Rules 7-10 model a good client. The last rule allows a client to restart the protocol. A bad behaving client could partially execute the protocol blocking either a forwarder or a provider.

Suppose that we consider only a finite number of agents as in [27], e.g., one forwarder, one provider, and one client all residing in different locations. Then the initial state configuration is the multiset $\gamma_0 = [idleF_i, idleS_j, idleC_k, last(0)]$. To specify a system with an arbitrary number of clients in location k, we only have to add the rule $\epsilon \rightarrow [\ idleC_k\ ]$ and consider the simplified initial configuration $\gamma_0' = [idleF_i, idleS_j, last(0)]$.

The generalized model of the FCS system is shown in Fig. 3. The initial configuration is $[init]$ (see rule 1). Rules $2-4$ generate (any number of) forwarders, providers and clients. In rules $2-5$ non-determinism is used to model the selection of the initial location or the location to which a client has to move. For instance, variable I in rule 2 and variable J in rule 5 occur only in the right-hand side, i.e., during a computation they can be assigned any value. This way we can work with an arbitrary number of locations and providers. All other rules are a natural generalization of our first model.