

On the Coverability Problem for Constrained Multiset Rewriting

Parosh Aziz Abdulla¹ and Giorgio Delzanno²

¹ *Uppsala University, Sweden - email:parosh@it.uu.se*

² *Università di Genova, Italy - email:giorgio@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. In this paper we prove decidability of the coverability problem for CMRS. The algorithm is obtained by a non-trivial application of a methodology based on the theory of well- and better-quasi orderings. We report on using a prototype implementation to verify parameterized versions of a mutual exclusion and an authentication protocol.

Key words: Infinite state systems, Symbolic Verification

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 [21,1,9,7] 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

*This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs*

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 [8] 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 [5,4] 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, i.e., conjunctions of arithmetic constraints 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.

Our main result 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. 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 [20]. 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* [21,1,9,7]. 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 [8] analyzes BPPs augmented with integer parameters. BPPs

are computationally weaker than Petri nets, and therefore also weaker than CMRS. The papers [5,4] consider parameterized systems of timed automata. However, timed systems 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 in the deductive database language *Datalog_{gap}* [24]. In [19] a bottom-up query evaluation algorithm for this language has been applied 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 [11,17]. Relational automata [12] 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.

The paper [14] 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 [10,16]. Decidability of coverability for a strict subset of gap-order constraints (comparisons between variables only) interpreted over the reals have been studied in [15].

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 $x + c < y$. We provide two appendices, one containing proofs of lemma and theorems, and the other giving details of the case studies.

Notes We include an appendix with examples and proofs taken from the technical report [3] in which we also study other decision problems.

2 Constrained Multiset Rewriting

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

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.

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 $Var(\rho) = Var(L) \cup Var(R) \cup Var(\psi)$.

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., $\gamma\uparrow = \{\gamma' \mid \gamma \leq \gamma'\}$.

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 . For instance, consider the rule

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

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

The *coverability problem* is defined as follows:

Instance: A CMRS, an *initial* configuration γ_i , and a *final* predicate symbol p_f ; *Question:* $\gamma_i \xrightarrow{*} p_f$?

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

Theorem 2.1 *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 An Algorithm for Testing Coverability

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.

The algorithm operates on *constraints*, where each constraint ϕ characterizes an infinite set $\llbracket\phi\rrbracket$ 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 $\llbracket\phi\rrbracket = \{\gamma \mid \exists Val. (Val \models \psi) \wedge (Val(T) \leq \gamma)\}$ of configurations. Notice that if ψ is inconsistent, then $\llbracket\phi\rrbracket$ 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, $\gamma_i \xrightarrow{*} p_f$ is equivalent to $\gamma_i \xrightarrow{*} \phi_{fin}$ where ϕ_{fin} is the constraint $[p_f(x)] : true$.

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., $\llbracket\phi_1\rrbracket \supseteq \llbracket\phi_2\rrbracket$. 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

- (i) 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 $c_1 + c_2 < c_3$.
- (ii) if $(x <_c y) \in \psi$ and $(y = z) \in \psi$ then $(x <_c z) \in \psi$.
- (iii) if $(x <_c y) \in \psi$ and $(x = z) \in \psi$ then $(z <_c y) \in \psi$.
- (iv) if $(x = y) \in \psi$ and $(y = z) \in \psi$ then $(x = z) \in \psi$.
- (v) For each $x, y \in \mathbb{V}$, at most one conjunct in ψ contains both x and y .
- (vi) $Var(\psi) \subseteq Var(T)$.

Fig. 1. Normalization requirements.

$$\bigcup_{\phi_1 \in Pre(\phi)} \llbracket \phi_1 \rrbracket = \{\gamma \mid \exists \gamma' \in \llbracket \phi \rrbracket. \gamma \longrightarrow \gamma'\}.$$

For instance, given $\phi_1 = [p(x_1), q(x_2), q(x_3)] : \{x_1 <_2 x_2, x_2 <_1 x_3\}$, 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 $\gamma_1 \in \llbracket \phi_1 \rrbracket$ and $\gamma_2 \notin \llbracket \phi_1 \rrbracket$. Consider now $\phi_2 = [p(y_1), q(y_2)] : \{y_1 < y_2\}$ and $\phi_3 = [p(y_1), q(y_2)] : \{y_1 <_4 y_2\}$. Then $\phi_2 \sqsubseteq \phi_1$ and $\phi_3 \not\sqsubseteq \phi_1$.

Given an instance of the coverability problem (Section 2), defined by γ_i and the constraint ϕ_{fin} corresponding to p_f , 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 $\gamma_i \in \llbracket \phi \rrbracket$; 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 this section (Lemma 3.2). These above three conditions are sufficient conditions for semi-decidability of the problem. At the end of this section we also show that the algorithm is guaranteed to terminate. *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 requirements in Fig. 1.

Lemma 3.1 *For each constraint ϕ we can effectively compute a constraint ϕ_{norm} such that ϕ_{norm} is in normal form and such that $\llbracket \phi \rrbracket = \llbracket \phi_{norm} \rrbracket$.*

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 $Var(T)$. 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 3.2 *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 $\gamma \in \llbracket \phi \rrbracket$ iff there is a valuation Val of $Var(\phi)$ such that $Val(\psi) \wedge (\gamma \geq Val(T))$. Computability follows since there are only finitely many valuations Val with $\gamma \geq Val(T)$.

Consider $\phi_1 = (T_1 : \psi_1)$ and $\phi_2 = (T_2 : \psi_2)$ which are in normal form (by Lemma 3.1 this is not a restriction). Let \mathcal{R} be the set of renamings of $Var(T_1)$ such that $Ren(T_1) \leq T_2$. Then $\phi_1 \sqsubseteq \phi_2$ is characterized by the formula

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

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 $x <_c y$. An example of entailment test is given in appendix.

Now 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

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

An example of computation of Pre_ρ is given in appendix. *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 [6], 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. 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, \dots 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 from $\{1, \dots, m\}$ to $\{1, \dots, n\}$ 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 BQOs¹ [6,22].

¹ The technical definition of BQOs is quite complicated and can be found in e.g. [6]. The actual definition is not needed for understanding the rest of the paper, and is therefore

Lemma 3.3 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 $(\mathcal{P}(A), \preceq^{\mathcal{P}})$ are BQOs.

Flat Constraints. A flat constraint $\phi_{\mathcal{F}}$ is a word of the form $B_0 d_1 B_1 d_2 \cdots d_n B_n$ where $B_0, B_1, \dots, B_n \in \mathbb{P}^\otimes$ and $d_1, d_2, \dots, d_n \in \mathbb{N}$. 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, \dots, B_n are all non-empty. For a configuration γ , we have $\gamma \in \llbracket \phi_{\mathcal{F}} \rrbracket$ if there are natural $c_0, \dots, c_n \in \mathbb{N}$ 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 3.3. 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 3.4 The entailment relation \sqsubseteq is a WQO on flat constraints.

For instance, 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 $\gamma_1 \in \llbracket \phi_1 \rrbracket$ and $\gamma_2, \gamma_3 \notin \llbracket \phi_1 \rrbracket$. Now 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 $\phi_4 \not\sqsubseteq \phi_1$.

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

- X_0, X_1, \dots, X_n is a partitioning of $\text{Var}(\phi)$.
- 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 $\phi_{\mathcal{F}}$ 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 $\text{Var}(T) = \text{Var}(\phi)$ and hence B_i is not empty for each $i : 1 \leq i \leq n$.

For instance, 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. $[p] 3 [q, r] 5 [r]$.

omitted here.

Another one is given by $\{x_1\} 3 \{x_2\} 5 \{x_3, x_4\}$ resp. $[p] 3 [q] 5 [r, r]$. However, $\{x_1\} 3 \{x_2\} 3 \{x_3, x_4\}$ is not a flattening of the condition of ϕ .

Lemma 3.5 *For a constraint ϕ and a configuration γ , we have $\gamma \models \phi$ iff $\gamma \models \phi_{\beta}$ for some flattening ϕ_{β} of ϕ .*

From Lemma 3.3, Lemma 3.4, and Lemma 3.5, we get the following.

Lemma 3.6 *The set of constraints is a BQO and a WQO under entailment.*

In the technical report [3] we have proved that coverability remains decidable for the more general form of conditions $x + c < y$, $x = y$, $x < c$, $x > c$, $x = c$ where x, y are variables and c is a natural number. Furthermore, the symbolic procedure defined for CMRS can be naturally extended to non-monic predicates. Indeed, normalization, checking membership, and checking entailment can be performed exactly in the same manner as for the monadic case. This yields a semi-algorithm for checking coverability. However, as shown in [3], termination is no longer guaranteed for non-monic CMRS.

4 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-monic CMRS. 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.

5 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. [7]), and through the use of more efficient data structures (e.g. [18]).

References

- [1] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. A survey of regular model checking. *CONCUR 2004*: 348–360.
- [2] P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.
- [3] P. A. Abdulla and G. Delzanno. Constraint Multiset Rewriting. Technical Report, DISI, Università di Genova, January 2006.
Available at the URL <http://www.disi.unige.it/person/papers.html>
- [4] P. A. Abdulla, J. Deneux, and P. Mahata. Multi-clock timed networks. *LICS 2004*: 345–354.
- [5] P. A. and B. Jonsson. Model checking of systems with many identical timed processes. *TCS*, 290(1):241–264, 2003.
- [6] P. A. Abdulla and A. Nylén. Better is better than well: On efficient verification of infinite-state systems. *LICS 2000*: 132–140.
- [7] B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. *CAV 2003*: 223–235.
- [8] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *TCS*, 295(1-3):85-106, 2003.
- [9] A. Bouajjani and T. Touili. Extrapolating Tree Transformations. *CAV 2002*, 539-554.
- [10] M. Bozzano and G. Delzanno. Beyond parameterized verification. *TACAS 2002*: 221–235.
- [11] T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables. *ACM TOPLAS*, 21(4):747–789, 1999.
- [12] K. Čerāns. Deciding properties of integral relational automata. *ICALP 1994*: 35–46.
- [13] D. de Frutos Escrig, V. Valero Ruiz, and O. Marroquín Alonso. Decidability of properties of timed-arc Petri nets. *ICATPN 2000*: 187–206.
- [14] G. Delzanno. An assertional language for systems parametric in several dimensions. *VEPAS 2001*.
- [15] G. Delzanno. Constraint Multiset Rewriting. Technical Report TR-05-08, DISI, Università di Genova, 2005.
Available at the URL <http://www.disi.unige.it/index.php?research/techrep>
- [16] G. Delzanno and P. Ganty. Automatic verification of time sensitive cryptographic protocols. *TACAS 2004*, 342-356.

- [17] G. Delzanno and A. Podelski. Model checking in clp. TACAS 1999, 223-239.
- [18] G. Delzanno and J. F. Raskin. Symbolic representation of upward-closed sets. TACAS 2000: 426–440.
- [19] L. Fribourg and J. Richardson. Symbolic verification with gap-order constraints. T.R. LIENS-93-3, Lab. d’Informatique, ENS Paris, 1996.
- [20] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *J. of the ACM*, 39(3):675–735, 1992.
- [21] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *TCS*, 256:93–112, 2001.
- [22] A. Marcone. Foundations of bqo theory. *Trans. of the American Mathematical Society*, 345(2), 1994.
- [23] R. Mayr. Undecidable problems in unreliable computations. LATIN’2000, 377-386.
- [24] P. Revesz. *Introduction to Constraint Databases*. Springer, 2002.
- [25] F. B. Schneider, B. Bloom, and K. Marzullo. Putting time into proof outlines. *Real-Time: Theory in Practice*, 1992.
- [26] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. LICS ’86: pages 332–344.
- [27] F. Rosa Velardo, O. Marroquín-Alonso, and D. de Frutos Escrig. Mobile synchronizing petri nets: a choreographic approach for coordination in ubiquitous systems. MTCoord 2005.

Example of Entailment

For instance, consider the constraints

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

Then the set $\mathcal{R} = \{Ren_1, Ren_2\}$ where $Ren_1 = (x_1 \mapsto y_1, x_2 \mapsto y_2, x_3 \mapsto y_3, x_4 \mapsto y_4)$ and $Ren_2 = (x_1 \mapsto y_1, x_2 \mapsto y_3, x_3 \mapsto y_2, x_4 \mapsto y_4)$. Therefore, $\phi_1 \sqsubseteq \phi_2$ is characterized by validity of the formula

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

Example of Symbolic Computation of Predecessors

For instance, 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 $Ren_2 = (x_1 \mapsto w_1, x_2 \mapsto w_2)$, and $Ren = (y_1 \mapsto w_3, y_2 \mapsto w_1, y_3 \mapsto w_4)$.

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 .

6 Proof of Some Lemmas

Lemma 3.2

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

Computing Pre

Lemma 3.4

Suppose that $\phi_f = B_0 d_1 B_1 d_2 \cdots d_m B_m$ and $\phi'_f = B'_0 d'_1 B'_1 d'_2 \cdots d'_n B'_n$. By definition it follows that $\phi_f \sqsubseteq \phi'_f$ iff there is a strictly monotone injection h from $\{1, \dots, m\}$ to $\{1, \dots, n\}$ 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 3.3 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 3.3. 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 3.5

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, \dots, X_n be sets of variables such that

- $Var(\phi) = X_1 \cup \dots \cup X_n$, 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 $X_0 d_1 X_1 d_2 \dots d_n X_n$ is a flattening of $Var(\phi)$:

- Since Val assigns to each variable in $Var(\phi)$ a value, it follows that X_0, X_1, \dots, 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 $\phi_{fl} = B_0 d_1 B_1 d_2 \dots d_n B_n$ be the flattening of ϕ induced by F . We show that $\gamma \models \phi_{fl}$. Let c_0, \dots, 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 $\phi_{fl} = B_0 d_1 B_1 d_2 \dots d_n B_n$ be a flattening of ϕ , induced by a flattening $F = X_0 d_1 X_1 d_2 \dots d_n X_n$ of $Var(\phi)$. Suppose $\gamma \in \llbracket \phi_{fl} \rrbracket$. We show that $\gamma \in \llbracket \phi \rrbracket$, i.e., we show that there is a valuation Val of $Var(\phi)$ such that $Val \models \psi$ and $\gamma \geq Val(T)$.

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

Fig. 2. CMRS model of the Mutual Exclusion Protocol.

Since $\gamma \in \llbracket \phi \rrbracket$ if there are natural $c_1, \dots, c_n \in \mathbb{N}$ 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 $c \neq c_0, \dots, c_n$. Otherwise, we have $Val(T)(p(c_i)) = \sum_{x \in X_i} T(p(x)) = B_i(p) \leq \gamma(p(c_i))$.

Details of Case Studies

Description of the Mutual Exclusion Protocol (Fig. 2)

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. 2. Violations to mutual exclusion can be represented symbolically via the

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. 3. Model for the FSC system.

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. Rule 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. 3 and Fig. 4)

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. 3) 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 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. 4) 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 $waitC(i, id)$.

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. 3 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 logs in using an identifier Id 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 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.

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. 4. 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

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

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.