Constraint Multiset Rewriting

Giorgio Delzanno

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Via Dodecaneso 35, 16146 Genova - Italy
E-mail: giorgio@disi.unige.it

Abstract. We investigate the foundations of the specification language MSR($\mathcal{C}$), that combines *multiset rewriting over first order atomic formulas* and a generic *constraints system* $\mathcal{C}$, and of a corresponding verification procedure for checking safety properties. Our method has been applied to the analysis of mutual exclusion, cache coherence and security protocols. All the case-studies belong to a class of concurrent systems for which verification is often undecidable. Indeed their state-space is infinite in more than one dimension: they are parametric in the number of components (processes/sessions) and have local data defined over infinite domains. Our verification procedure is based on symbolic backward exploration. Infinite sets of configurations of an MSR($\mathcal{C}$) specification are represented symbolically via assertions defined over multisets and constraints. Constraints and term manipulation operations are combined together in order to effectively manipulate assertions. All operations are defined for a generic constraint system $\mathcal{C}$. By a careful choice of the constraints system and of the syntax of rules, we isolate a new class of constraint multiset rewriting specifications (hence of infinite-state systems) for which verification of safety properties is decidable. Decidability is obtained by applying the theory of well and better quasi orderings.

1 Introduction

In [16] we proposed a constraint-based method for the automatic verification of systems and protocols parametric in several dimensions. The method has three components: a specification language, called MSR($\mathcal{C}$), that combines multiset rewriting and constraints, an assertional language to represents infinite sets of configurations, and a symbolic verification procedure working on assertions.

The specification language MSR($\mathcal{C}$) combines *multiset rewriting over first order atomic formulas* [12] with a *constraint system* $\mathcal{C}$ (a parameter of the definition of the specification logic). This formalism allows us to separate the specification of the process structure (specified in multiset rewriting) from the specification of the relation over local data of different processes (specified using constraints). Their combination allows us to specify concurrent systems parametric in the number of processes and in which processes have an internal state defined over an infinite domain.

The assertional language allows us to symbolically represent infinite sets of configurations. Assertions are defined over a combination of multisets of atomic formulas and constraints. The denotation of an assertion is an upward closed set of configurations. This kind of representation is adequate to symbolically represent bad (with respect a given invariant property) states of parametric systems. For instance, violations of mutual exclusion in a multitasking system can be represented via all configurations (with any number of processes) containing at least two processes in their critical section. Term manipulation (unification and subsumption of multisets of first order atomic formulas) and constraint programming (satisfiability, entailment, and variable elimination) are used to effectively handle sets of assertions.

The verification procedure (parametric on the constraint system) performs a symbolic backward exploration of the state-space of an MSR($\mathcal{C}$) specification. Assertions are used to represent infinite sets of visited configurations. The procedure is defined via a symbolic predecessor operator that, given a finite set of assertions S, computes a finite set of assertions S' that represents the predecessor configurations of the denotation of S. The resulting procedure can be used to attack control state reachability problems, i.e., reachability of some configurations belonging to an upward closed set. As mentioned before, safety properties can often be reduced to this kind of verification problems. Our verification procedure represents a natural extension of the constraint-based symbolic backward reachability algorithms used for Petri Nets we presented in [15, 17].

The resulting method has been applied to several different classes of infinite-state systems. In [7] we used an instance of the MSR($\mathcal{C}$) scheme with arithmetic constraints to verify properties of parameterized versions of mutual exclusion protocols. In [8] we extended the specification language with a broadcast communication mechanism and we applied the resulting method to verify safety properties for cache coherence protocols. In [18] we studied the applicability of an instance with integer constraints to verification of safety properties of π-calculus specifications. In [19] we used a constraint system combining arithmetics and equalities over first order terms to verify secrecy and authentication properties of time-sensitive cryptographic protocols with an unbounded number of parallel sessions.

Both reachability and control state reachability are undecidable for most of the instances of MSR($\mathcal{C}$) used in the previously described applications. Although our verification procedure was not guaranteed to terminate, abstractions and pruning techniques worked well in our practical experiments.

Technical Contribution As an integration of the practical results in [7, 8, 18, 19], in the present paper we present the foundation of the scheme MSR($\mathcal{C}$) with parametric constraint system $\mathcal{C}$, and we investigate (fragments of) instances for which our verification procedure terminates. More in detail, our contributions are as follows.

1. We present a proof of soundness of our verification procedure for specifications defined over a generic constraint system $\mathcal{C}$. Soundness holds for a constraint language closed under conjunction and variable elimination (existential quantification). Furthermore, every step is effective provided the constraint system is equipped with effective satisfiability, variable elimination and entailment procedures.
2. Furthermore, we present a completeness proof for the monadic fragment of the instance MSR($\mathcal{N}$) of the language scheme in which constraints are taken from a subclass of linear arithmetic formulas. $\mathcal{N}$-constraints allow us to handle an infinite collection of values and to compare them using relations like $>$, $\geq$, and $=$. This domain can be used to represent local variables, process identifiers, priorities, and so on. Monadic MSR($\mathcal{N}$) is a a natural extension of Petri Nets in which data are attached to tokens. In the completeness proof we show that our verification procedure always terminates when taking in input a Monadic MSR($\mathcal{N}$) specification and a finite set of assertions (with monadic atomic formulas and $\mathcal{N}$-constraints). This result is consistent with the results obtained for Petri Nets (backward reachability always terminates when considering upward closed sets of

markings only). The proof exploits the theory of well and better quasi orderings, an approach inspired to [1, 3, 22].
3. Finally, we show that Two Counters Machines can be encoded in dyadic MSR($\mathcal{N}$), a restriction of MSR($\mathcal{N}$) in which we allow only binary predicates. This result implies that Monadic MSR($\mathcal{N}$) is strictly included in dyadic MSR($\mathcal{N}$).

The analysis of fragments of MSR($\mathcal{C}$) allowed us to isolate a new class of infinite-state concurrent systems for which safety properties can be automatically verified. As discussed at the end of the paper, this result has interesting practical applications, e.g., for the verification of mutual exclusion protocols.

Related Work Our approach is inspired to the constraint-based methodology suggested in [1, 27] where constraints (assertions) are viewed as a general way of representing infinite sets of upward closed sets of configurations of transition systems, and are used for symbolic backward exploration. In our setting we use formulas combining different specification languages (constrained configurations) as assertions. The resulting framework extends the constraint-based approach we used for Petri Nets in [15, 17]. Instances of the general approach of [1] are presented in [2, 3]. In [2, 3] *existential regions* are proposed as symbolic representation of configurations for parameterized Timed Petri Nets. In our work we consider however problems and constraint systems that do not depend on the notion of *time*. Networks of *finite-state* processes can be analyzed using the automata theoretic approach of [10, 5, 14, 32, 27], where sets of global states are represented as *regular languages*, and transitions as relations on languages. Symbolic exploration can then be performed using operations over automata with ad hoc accelerations (see e.g. [6]), or with automated abstractions techniques (see e.g. [5]). As oppposed to the automata theoretic approach, in our setting we handle parameterized systems in which individual components have local variables that range over *unbounded* values. The aforementioned features also distinguish our approach from the *verification with invisible invariants* method of [4]. As an example, in our model of the ticket algorithm local variables and tickets range over unbounded integers. Furthermore, the abstraction with NC-constraints does not make the resulting symbolic representation finite. Nevertheless, termination can be guaranteed by applying the theory of well quasi ordering. A formulation of the *single-server* ticket algorithm with 2 processes, but unbounded global and local variables, has been automatically verified using constraint-based model checkers equipped with a Presburger constraint solver [11], and a *real arithmetic* one [20]. However, we are not aware of other methods that can automatically handle parameterized models like the one in Section 8.

Our approach is related to previous works connecting Constraint Logic Programming and verification, see e.g. [20]. In this setting transition systems are encoded via CLP programs used to encode the *global* state of a system and its updates. We refine this idea by using multiset rewriting and constraints to *locally* specify updates to the *global* state. The notion of *constrained configuration* extends that of *constrained atom* of [20]. The *locality* of rules allows us to consider *rich* denotations (upward-closures) instead of *flat* ones (instances) like, e.g., in [20]. This way, we can lift the approach to the parameterized case.

Finally, the use of constraints, backward reachability, and better quasi orderings distinguish our hybrid method from classical approaches based on multiset and AC rewriting techniques (see e.g. [12, 31]). For instance, validation approaches based on rewriting like [13, 21, 25] are used for debugging and not for verification. Furthermore, since they use a forward search, they cannot exploit the property of upward closed sets of configurations (typically, the initial states are not upward closed) and of the predecessor operator. The effectiveness and the decidability results for our verification procedure strongly relies on the backward search strategy.

As mentioned in the introduction, in this paper we present unifying foundations for the approaches defined (using different constraint solvers) in [16, 7, 18, 19] and give with full proofs of all results. The decidable fragments studied in the present paper are defined over arithmetic constraints interpreted over reals or rationals. We are currently working on a generalization of the results for integer constraints claimed in [7] (the class of constraints considered there can be generalized to constraints with special types of arithmetic operations). The technique used for the decidability result is an extension to a constraint logic of a result we obtained in [9] for a monadic fragment of linear logic with equality constraints only. As explained in Section 6, the introduction of constraints like $<$ makes the proof much more complicated.
More information on the method and its applications can be found at the URL

http://www.disi.unige.it/person/DelzannoG/MSR/

Plan of the Paper In Section 2 we introduce the formalism MSR($\mathcal{C}$) we use as specification language.
In Section 3 we define the assertional language, in Section 4 we define the symbolic predecessor operator, and in Section 5 we define the symbolic backward reachability procedure.
In Section 6 we investigate the decidable fragments called Monadic MSR($\mathcal{N}$), and in Section 7 we compare Monadic MSR($\mathcal{N}$) with Dyadic MSR($\mathcal{N}$).
In Section 8 we discuss a case-study.
In Appendix A we recall the main definitions of Petri Nets and Two Counter Machines, and in Appendix B we summarize known results from the theory of well and better quasi orderings (taken from [1, 3]) useful to understand the proofs presented in Section 6.

2 Constraint Multiset Rewriting

To formally introduce our specification logic, we need some preliminary definitions.

A *constraint system* is a tuple $\mathcal{C} = \langle \mathcal{V}, \mathcal{L}, \mathcal{D}, Sol, \sqsubseteq^c \rangle$ where:

- $\mathcal{V}$ is a denumerable set of variables;
- $\mathcal{L}$ is a first-order language defining a set of formulas with free variables in $\mathcal{V}$, called constraints, closed with respect to variable renaming, existential quantification and conjunction, and allowing equalities between variables; we assume that $true$ and $false$ are always in $\mathcal{L}$;
- $\mathcal{D}$ is the interpretation domain of constraints;
- $Sol(\varphi)$ is the set of mappings (solutions) $\mathcal{V} \to \mathcal{D}$ of a constraint φ that preserves the usual semantics of equalities, $\wedge$ and $\exists$ (intersection and projection of the solutions);
- $\sqsubseteq^c$ is a relation such that $\varphi \sqsubseteq^c \psi$ implies $Sol(\varphi) \subseteq Sol(\psi)$ (the entailment relation).

A constraint φ is said to be satisfiable if $Sol(\varphi) \neq \emptyset$. ($Sol(false) = \emptyset$, whereas all mappings are in $Sol(true)$). In the rest of the paper we will often write the conjunction $\varphi \wedge \psi$ as φ, ψ (we replace $\wedge$ with a comma). Furthermore, we will use the notation $\langle x_1 \mapsto d_1, x_2 \mapsto d_2, \ldots \rangle$ to denote a solution σ mapping x_i to d_i for $i \geq 1$, and the notation $\sigma_{|V}$ to denote the *restriction* of the evaluation σ to the variables in $V \subseteq \mathcal{V}$.

As an example, linear (integer or real) arithmetics form a constraint system in which constraints are conjunctions of atomic formulas of the form $a_1x_1 + \ldots + a_nx_n @ a_{n+1}$ where $a_i \in \mathbb{Z}$ for $i : 1, \ldots, n+1$, and $@ \in \{=, \leq, \geq, <, >\}$. The entailment relation is the standard logical implication.

Let $\mathcal{P}$ be a finite set of predicate symbols, and $\mathcal{C} = \langle \mathcal{V}, \mathcal{L}, \mathcal{D}, Sol, \sqsubseteq^c \rangle$ a constraint system. An atomic formula over $\mathcal{P}$ and $\mathcal{V}$ has the form $p(x_1, \ldots, x_n)$ (with $n \geq 0$), where $p \in \mathcal{P}$, and $x_1, \ldots, x_n$ are distinct variables in $\mathcal{V}$. An atomic formula A is *ground* if $A = p(d_1, \ldots, d_n)$, where $p \in \mathcal{P}$ and $d_1, \ldots, d_n \in \mathcal{D}$. A *multiset* of atomic formulas $A_1, \ldots, A_k$ (with $k \geq 1$) over $\mathcal{P}$ and $\mathcal{V}$, is indicated as $A_1 \mid \ldots \mid A_k$, where A_i and A_j must have distinct variables for $i \neq j$, and $\mid$ is an associative and commutative constructor.

Terminology: In the rest of the paper will use $\mathcal{M}$, $\mathcal{N}$, ... to denote *multisets* of atomic formulas. Furthermore, we will use $\oplus$ and $\ominus$ to denote *multiset union* and *multiset difference*, respectively, ϵ to denote the empty multiset, and $\preccurlyeq$ to denote *multiset inclusion* ($\mathcal{M} \preccurlyeq \mathcal{N}$ if and only if $Occ_A(\mathcal{M}) \leq Occ_A(\mathcal{N})$ for every atomic formula A, where $Occ_A(\mathcal{M})$ denotes the number of occurrences of A in $\mathcal{M}$). Finally, we will use $\sigma(A)$ and $\sigma(\mathcal{M})$ to denote the application of the natural extension of $\sigma : \mathcal{V} \to \mathcal{D}$ to an atomic formula A and to a multiset of atomic formulas $\mathcal{M}$, respectively. We are now ready to define the framework of multiset rewriting with constraints.

2.1 The MSR($\mathcal{C}$) Specification Language Scheme

An MSR($\mathcal{C}$) *specification* is a tuple $\langle \mathcal{P}, \mathcal{C}, \mathcal{I}, \mathcal{R} \rangle$, where $\mathcal{P}$ is a finite set of predicate symbols, $\mathcal{C}$ is a constraint system, $\mathcal{I}$ is a set of *initial configurations*, and $\mathcal{R}$ is a set of MSR($\mathcal{C}$) *rules* over $\mathcal{P}$ and $\mathcal{C}$.

A *configuration* is a multiset of ground atomic formulas.

An MSR($\mathcal{C}$) *rule* over $\mathcal{P}$ and $\mathcal{C}$ has the form

$$\mathcal{M} \longrightarrow \mathcal{M}' : \varphi$$

where $\mathcal{M}$ and $\mathcal{M}'$ are two multisets of atomic formulas with distinct variables and defined over $\mathcal{P}$ and $\mathcal{V}$, and $\varphi \in \mathcal{L}$. Note that either $\mathcal{M}$ or $\mathcal{M}'$ can be the empty multiset ϵ. We will show an example right after describing the operational semantics of this language. The operational semantics describes the update from a configuration $\mathcal{M}$ to one of its possible successor configurations $\mathcal{M}'$. $\mathcal{M}'$ is obtained from $\mathcal{M}$ by rewriting (modulo associativity and commutativity) the left-hand side of an instance of a rule into the corresponding right-hand side. In order to be fireable, the left-hand side must be included in $\mathcal{M}$. Since instances and rules are selected in a non deterministic way, in general a configuration can have a (possibly infinite) set of (one-step) successors.

Formally, a rule $\mathcal{H} \longrightarrow \mathcal{B} : \varphi$ from $\mathcal{R}$ is enabled at $\mathcal{M}$ *via* $\sigma \in Sol(\varphi)$ if and only if $\sigma(\mathcal{H}) \preccurlyeq \mathcal{M}$. Firing rule R enabled at $\mathcal{M}$ via σ yields the new configuration

$$\mathcal{M}' = \sigma(\mathcal{B}) \oplus (\mathcal{M} \ominus \sigma(\mathcal{H}))$$

We use $\mathcal{M} \Rightarrow_R \mathcal{M}'$ to denote the firing of R at $\mathcal{M}$ yielding $\mathcal{M}'$.

A run is a sequence of configurations $\mathcal{M}_0\mathcal{M}_1 \ldots \mathcal{M}_k$ with $\mathcal{M}_0 \in \mathcal{I}$ and for which there exists

a sequences of rules $R_0 R_1 \dots R_{k-1} \in \mathcal{R}$ such that $\mathcal{M}_i \Rightarrow_{R_i} \mathcal{M}_{i+1}$ for $i \geq 0$.
Given a set S of configurations the successor operator $Post$ is defined as

$$Post(S) = \{\mathcal{M}' \mid \mathcal{M} \Rightarrow_R \mathcal{M}',\ \mathcal{M} \in S,\ R \in \mathcal{R}\}$$

whereas, the predecessor operator Pre is defined as

$$Pre(S) = \{\mathcal{M} \mid \mathcal{M} \Rightarrow_R \mathcal{M}',\ \mathcal{M}' \in S,\ R \in \mathcal{R}\}$$

The reflexive and transitive closure of the predecessor (successor) operator is denoted Pre^* ($Post^*$). Finally, the reachability set is defined as the set $Post^*(\mathcal{I})$, where $\mathcal{I}$ is the set of initial configurations.

Example 1. Consider a systems composed by an arbitrary number of visitors concurrently accessing a web site and by a counter that keeps track of the number of accesses. Since there is no upper-bound on the number of visitors, the counter can assume arbitrarily large values. To specify this scenario, we choose the constraint system of linear integer arithmetics and set $\mathcal{P} = \{init, count, visit, enter\}$ and $\mathcal{I} = \{init\}$. Furthermore, we assume that $x, x', y, y', \ldots \in \mathcal{V}$. The specification is given via the following rules:

$$\begin{array}{ll}
r_1. & \epsilon \longrightarrow visit \,:\, true \\
r_2. & visit \longrightarrow \epsilon \,:\, true \\
r_3. & init \longrightarrow count(x) \,:\, x = 0 \\
r_4. & count(x) \mid visit \longrightarrow count(x') \mid enter(y') \,:\, y' = x', x' = x + 1
\end{array}$$

Visitors are injected into the system at any time by the first rule (since the left-hand side is the empty multiset, it is fireable at any configuration). The second rule specifies the behavior of visitors leaving the system before accessing the web site (since the right-hand side is the empty multiset, they simply disappear from the current configuration). The third rule specifies the initial configuration of the counter (its value is zero). The fourth rule is used to count the number of accesses to the web page. Each visitor gets the current value of the counter as a result of the access (via the predicate *enter*). The following run shows that the structure of configurations can grow in width (we can add an arbitrary number of occurrences of the atomic formula *visit*):

$$init \Rightarrow_{r_3} count(0) \Rightarrow_{r_1} count(0) \mid visit \Rightarrow_{r_1} count(0) \mid visit \mid visit \Rightarrow_{r_2} count(0) \mid visit$$

Given $\mathcal{M} = count(0) \mid visit \mid visit$, the following run shows an update of the local values of processes represented via atomic formulas:

$$init \stackrel{*}{\Rightarrow} \mathcal{M} \Rightarrow_{r_4} count(1) \mid enter(1) \mid visit \Rightarrow_{r_4} count(2) \mid enter(1) \mid enter(2)$$

Given $\mathcal{M}' = count(2) \mid enter(1) \mid enter(2)$, the following run shows that, by injecting new visitors, configurations can grow in width while local data get strictly increasing values:

$$\begin{array}{l}
init \stackrel{*}{\Rightarrow} \mathcal{M}' \Rightarrow_{r_2} count(2) \mid enter(1) \mid enter(2) \mid visit \\
\quad \Rightarrow_{r_4} count(3) \mid enter(1) \mid enter(2) \mid enter(3)
\end{array}$$

Thus, the state space of this simple example is infinite in at least two dimensions: the number of processes (occurrences of atomic formulas) and the set of possible values that the counter can take. This feature makes verification of properties of MSR($\mathcal{C}$) a challenging problem.

2.2 Expressive Power

MSR($\mathcal{C}$) is a natural extension of formalisms like Petri Nets (see Appendix A for their definition). To model a Petri Net, we can resort to the instance MSR$_{PN}$ of MSR($\mathcal{C}$) in which we only consider atomic propositions. The key idea is to model the occurrence of a token in place p_i via the atomic formula a_i. In this encoding we need no constraints since the set of tokens in a certain place are indistinguishable (i.e. they have no attached data). A marking $\mathbf{m} = \langle m_1, \ldots, m_n \rangle$ corresponds then to an MSR$_{PN}$ configuration in which we have m_i occurrences of proposition a_i for $i : 1, \ldots, n$. A Petri Net transition t_j can be defined then via the MSR$_{PN}$ rule $\mathcal{A} \leftarrow \mathcal{B}$ in which a_i occurs in $\mathcal{A}$ if p_i is an input place for t_j, and a_i occurs in $\mathcal{B}$ if p_i is an output place for t_j for $i : 1, \ldots, n$. The operational semantics of a Petri Net coincide with that of its MSR$_{PN}$ encoding.

Example 2. As an example, let us consider a Petri Net with places $P = \{p_1, p_2, p_3\}$, and a transition t with p_1 and p_2 as input places and p_3 as output. The MSR$_{PN}$ encoding of t is

$$a_1 \mid a_2 \rightarrow a_3$$

Firing t at marking $\mathbf{m_0} = \langle 2, 3, 0 \rangle$ is the same as applying R to the configuration

$$a_1 \mid a_1 \mid a_2 \mid a_2 \mid a_2$$

(notice that $\mid$ is associative and commutative). Indeed, the new configuration

$$a_1 \mid a_2 \mid a_2 \mid a_3$$

corresponds to the marking $\mathbf{m_1} = \langle 1, 2, 1 \rangle$.

The introduction of constraints augments the expressive power of our language. To prove this claim, we show next how to encode the Turing equivalent formalism of Two Counters Machines (see Appendix A for a definition) in the instance MSR($\mathcal{L}$) in which $\mathcal{L}$ is the constraint system of linear integer constraints. The configuration $\langle p, c_1 = n_1, c_2 = n_2 \rangle$ of a Two Counters Machine can be modeled in MSR($\mathcal{L}$) as the atomic formula $p(n_1, n_2)$ where p is the current control location, and n_i (a natural number) is the current value of the counter c_i. The instructions (for the first counter) are encoded as follows

$$\text{Zero} : p(x, y) \rightarrow p'(x', y') : x = 0, x' = x, y' = y$$

$$\text{Inc} : \; p(x, y) \rightarrow p'(x', y') : x' = x + 1, y' = y$$

$$\text{Dec} : \; p(x, y) \rightarrow p'(x', y') : x > 0, x' = x - 1, y' = y$$

where p and p' represent the old and new location, respectively. The operations on the second counter are defined in a similar way. The initial state $\langle Loc = p, c_1 = 0, c_2 = 0 \rangle$ is defined via the rule

$$\text{Init} : init \rightarrow p(x, y) : x = y = 0$$

This encoding shows that, as opposed to Petri Nets, reachability is undecidable for MSR($\mathcal{L}$).

3 Assertional Language

As shown in Example 1, the reachability set of an MSR($\mathcal{C}$) specification is in general infinite. To explore the state-space of this kind of models, we need to find symbolic representations of infinite sets of configurations. In this paper we will focus our attention on symbolic representations of *upward closed sets* of configurations. The upward closure of a set S, denoted $Up(S)$, is defined as follows (where $\preccurlyeq$ is multiset inclusion)

$$Up(S) = \{\mathcal{N} \mid \mathcal{M} \preccurlyeq \mathcal{N}, \mathcal{M} \in S\}.$$

We say that S is *upward closed* if $S = Up(S)$.

The first motivation to study the class of upward closed sets of configurations is that it is closed under applications of the predecessor operator. Indeed we have that $Up(Pre(S)) = Pre(Up(S))$, i.e., $Up(Pre(S)) = Pre(S)$ whenever S is upward closed. Thus, in a backward search that starts from an upward closed set we can only generate upward closed sets.

The second motivation is that there exists a strong relation between upward closed sets and verification of safety properties (invariant properties that hold in every reachable state). Indeed, violations of safety property are often upward closed sets of configurations. To illustrate this idea, let us go back to Example 1. The specification in this example satisfies the property that all reachable states have the form

$$count(k) \oplus E_k \oplus V$$

where E_k is the multiset $enter(1) \mid enter(2) \mid \ldots \mid enter(k)$ for $k \geq 0$, and V is any multiset built on the symbol *visit*. To prove that this property is invariant, we can reason by contraposition. We first identify all possible configurations that violate the property, and then show that they are not reachable from the initial state *init*, e.g., by performing a backward exploration of the state-space using the predecessor operator. As mentioned before, using this strategy we only need to work on upward closed sets. In our example, the set of violations U consists of configurations like

$$count(v) \mid enter(w) \ \oplus \ Q$$

where $w > v$ and Q is any possible multiset. U corresponds to the upward closure of the (infinite) set of configurations

$$V = \{\mathcal{M} \mid \mathcal{M} = (count(v) \mid enter(w)),\ and\ v < w\}$$

V corresponds to the set of minimal violations of the invariant: If a configuration N contains an element of V ($N \in Up(V)$), then N violates the invariant.

We still have to solve the problem of finitely representing the set of minimal violations. This is not always possible. However, in our example they have a regular structure that we can exploit by translating the extensional description into the following intensional description, we will refer to as a *constrained configuration*,

$$count(x) \mid enter(y) \ : \ x < y$$

where x, y are *free variables* and $x < y$ is a linear constraint. If we use the formula to represent the upward closure of its ground instances, we obtain a finite and compact representation of infinite sets of configurations. This assertional language can be used in a backward search scheme provided we find ways to symbolically compute predecessor states. In the following sections we will investigate this problem.

3.1 Constrained Configurations

A constrained configuration over $\mathcal{P}$ (the set of predicate names) and $\mathcal{C}$ (the constraint system) is a formula

$$p_1(x_{11},\ldots,x_{1k_1}) \mid \ldots \mid p_n(x_{n1}\ldots,x_{nk_n}) \,:\, \varphi$$

where $p_1,\ldots,p_n \in \mathcal{P}$, $\varphi \in \mathcal{L}$ is a *satisfiable* constraint, and $x_{11},\ldots,x_{nk_n}$ are *distinct* variables in $\mathcal{V}$.

Terminology: We will use $M, N, \ldots$ to denote constrained configurations and $S, S', \ldots$ to denote sets of constrained configurations.

The set of ground instances of $\mathcal{M} \,:\, \varphi$ is defined as

$$\mathit{Inst}(\mathcal{M} \,:\, \varphi) = \{\sigma(\mathcal{M}) \mid \sigma \in \mathit{Sol}(\varphi)\}$$

The previous definition is extended to sets as follows

$$\mathit{Inst}(S) = \bigcup_{C \,\in\, S} \mathit{Inst}(C)$$

Let S be a set of constrained configurations with pairwise distinct variables. The *denotation* of a set of constrained configurations is defined then as:

$$[\![S]\!] = \mathit{Up}(\mathit{Inst}(S))$$

Thus, we always take the upward closure (with respect to multiset inclusion) of ground instances of constrained configurations in S.

As an example, let C be the constrained configuration $\mathit{count}(x) \mid \mathit{enter}(y) \,:\, x < y$. Then, if linear constraints are interpreted over natural numbers $\mathit{Inst}(C)$ contains configurations like $\mathit{count}(1) \mid \mathit{enter}(2)$, and $\mathit{count}(4) \mid \mathit{enter}(6)$, whereas $[\![C]\!]$ also contains configurations like $\mathit{count}(1) \mid \mathit{enter}(2) \mid \mathit{enter}(0)$, $\mathit{count}(1) \mid \mathit{enter}(2) \mid \mathit{enter}(0)\ \mathit{visit}$, etc.

3.2 Parallel with Petri Nets

Constrained configurations in MSR_{PN} can be used to symbolically represent upward closed sets of markings (a well-studied notion in the theory of Petri Nets [1, 22]). To illustrate, let us go back to Example 2. The marking $\mathbf{m_0}$ is represented by the MSR_{PN} configuration $\mathcal{M}$ defined as the multiset without constraints

$$a_1 \mid a_1 \mid a_2 \mid a_2 \mid a_2$$

If we interpret $\mathcal{M}$ as a constrained configuration, i.e., we lift its denotation to its upward closure, we obtain that

$$[\![\mathcal{M}]\!] = \{\mathcal{M}' \mid \mathcal{M} \preccurlyeq \mathcal{M}'\}$$

By definition of the submultiset relation, we obtain that $\mathcal{M}' \in [\![\mathcal{M}]\!]$ if and only if the number of occurrences of a_1, a_2 and a_3 in $\mathcal{M}'$ is greater or equal than 2, 3, and 0, respectively. Thus, $[\![\mathcal{M}]\!]$ coincides with the upward closure of the marking $\mathbf{m_0}$ (all markings that, for each place, have at least the same number of tokens as $\mathbf{m_0}$).

3.3 Unification of Constrained Configurations

In order to manipulate constrained configurations we combine multiset and constraint operations. The first operation we consider is the extension of term unification.

Given two atomic formulas $A = p(x_1, \ldots, x_k)$ and $B = q(y_1, \ldots, y_l)$, we use $A = B$ as a shorthand for the constraint $x_1 = y_1 \wedge \ldots \wedge x_k = y_k$, provided $p = q$ and $k = l$.

Given $M = (A_1 \mid \ldots \mid A_n : \varphi)$, and $N = (B_1 \mid \ldots \mid B_m : \psi)$, we say that $\mu \in \mathcal{L}$ is a *unifier* for M and N, written $M =_\theta N$ if and only if $m = n$ and there exists a permutation $j_1, \ldots, j_n$ of $1, \ldots, n$ such that the constraint

$$\mu \equiv \varphi \wedge \psi \wedge \bigwedge_{i=1}^{n} A_i = B_{j_i}$$

is *satisfiable*.

As an example, let M be the constrained configuration

$$p(x) \mid q(y) \mid r(z) : \; x \geq y \wedge y = z$$

and N be the constrained configuration

$$q(u) \mid p(v) \mid r(w) : \; v = u$$

Then, if we consider the permutation $p(v) \mid q(u) | \, r(w) : \; v = u$ of N we see that $M =_\theta N$ where

$$\begin{aligned} \theta &\equiv (x = v, u = y, z = w, x \geq y, y = z, v = u) \\ &\equiv (x = v, v = u, u = y, y = z, z = w) \end{aligned}$$

3.4 Entailment Relation

An entailment is any binary relation *Entails* defined over constrained configurations that satisfies the following property:

$$\mathit{Entails}(N, M) \text{ implies } [\![N]\!] \subseteq [\![M]\!]$$

An entailment allows us to compare constrained configurations, a key operation for the definition of a symbolic search procedure.

Given an entailment *Entails*, we can define an entailment relation $\mathit{Entails}^S$ over sets of constrained configurations by comparing elements pointwise. Formally,

$$\mathit{Entails}^S(S, S') \text{ holds iff for every } N \in S \text{ there exists } M \in S' \text{ such that } \mathit{Entails}(N, M).$$

The following property immediately follows from this definition.

Proposition 1. *If Entails is an entailment, then* $\mathit{Entails}^S(S, S')$ *implies* $[\![S]\!] \subseteq [\![S']\!]$.

3.5 Generic Entailment Relation

Given a constraint system $\mathcal{C}$ it is possible to define a generic entailment relation *GenEntails* parametric on the operations of $\mathcal{C}$. Let N and M be two constrained configurations. To obtain sufficient conditions for $[\![N]\!] \subseteq [\![M]\!]$ to hold, we proceed as follows. We first select a submultiset $\mathcal{N}'$ in N that unifies the multiset component of M via a constraint μ (N must contain a subconfiguration with the same structure as M) and such that the constraint in N entails (its denotations are contained in those of) the constraint ξ obtained by conjoining μ with the constraint in M. To compare them, they must be defined over the same set of variables. For this purpose, we existentially quantify all variables of N in ξ and obtain a constraint with the same variables as M.

Formally, given $M = (A_1 \mid \ldots \mid A_n : \varphi)$ and $N = (B_1 \mid \ldots \mid B_k : \psi)$ the relation $GenEntails(N, M)$ holds if there exist $j_1, \ldots, j_n$ *distinct* indexes in $\{1, \ldots, k\}$, and constraints μ and ξ such that

- $(A_1 \mid \ldots \mid A_n) : true =_\mu (B_{j_1} \mid \ldots \mid B_{j_n}) : true$ and
- $\xi \equiv \exists x_1 \ldots \exists x_r.\ (\mu \wedge \varphi)$ where $x_1, \ldots, x_r$ are the variables in N
- $\xi \sqsubseteq^c \psi$ (i.e. the solutions of ξ are solutions of ψ).

Notice that the previous conditions imply that $n \leq k$.

As an example, let M be the constrained configuration $p(v) \mid q(w) :\ v \geq w$ and N be the constrained configuration $p(x) \mid q(y) \mid r(z) :\ x \geq y - 1 \wedge y = z$. If we consider the submultiset $p(x) \mid q(y)$ of N, we have that $\mu \equiv (x = v \wedge y = w)$ is a unifier for $p(x) \mid q(y) : true$ and $p(v) \mid q(w) : true$. Furthermore, by conjoining μ and the constraint of N we obtain the constraint ξ defined as $x = v \wedge y = w \wedge x \geq y - 1 \wedge y = z$. In order to compare φ and ξ we eliminate all variables in N occurring in ξ, namely x, y, and z, and obtain the equivalent constraint $\xi' = v \geq w - 1$ defined over the same variables of φ. At this point, we notice that $v \geq w - 1$ entails $v \geq w$ (as constraints). Thus, we have that $GenEntails(N, M)$ holds.

The entailment relation *GenEntails* provides sufficient conditions that are parametric on $\mathcal{C}$ for testing containment and equivalence of constrained configurations. Specifically, the following property holds.

Proposition 2. *GenEntails is an entailment relation.*

Proof. Given $M = (A_1 \mid \ldots \mid A_n : \varphi)$ and $N = (B_1 \mid \ldots \mid B_k : \psi)$, suppose that $GenEntails(N, M)$ holds. This implies that there exist $j_1, \ldots, j_n$ such that $(A_1 \mid \ldots \mid A_n) : true =_\mu (B_{j_1} \mid \ldots \mid B_{j_n}) : true$, $\xi \equiv \exists x_1 \ldots \exists x_r.\ (\mu \wedge \varphi)$ where $x_1, \ldots, x_r$ are the variables in M, and $\xi \sqsubseteq^c \psi$ (i.e. the solutions of ξ are solutions of ψ).
Now let $\mathcal{N}'$ be a configuration in $[\![N]\!]$, i.e., $\mathcal{N}' = (B_1\sigma \mid \ldots \mid B_k\sigma)$, where σ is a solution of ψ. Notice that ν contains only equalities between variables occurring in the same position in A_i and B_{j_i} for $i : 1, \ldots, n$. By definition of constrained configurations the variables are all distinct. This means that a solution σ for ψ can be extended to a solution for ξ in which the variables at same position in A_i and B_{j_i} are mapped to the same value. Since, by hypothesis, solutions of ξ are solutions of φ, we have that the instance $A_1\sigma \mid \ldots \mid A_n\sigma$ belongs to $[\![M]\!]$. Furthermore, since ν is a unifier we know that $A_i\sigma = B_i\sigma$ for $i : 1, \ldots, n$. Thus, $\mathcal{N}'$ belongs to the upward closure of an element in $[\![M]\!]$. □

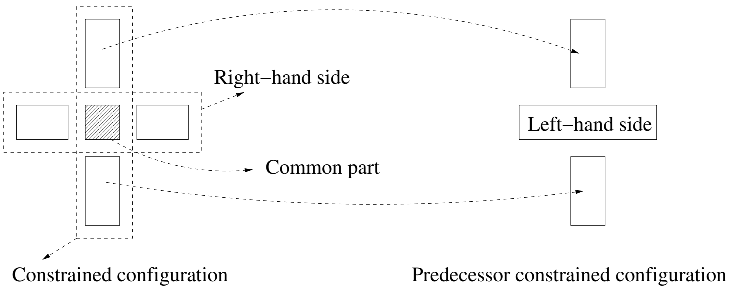

Fig. 1. Symbolic computation of predecessors.

4 The Symbolic Predecessor Operator *SPre*

In this section we will define the more important operation needed to build the symbolic backward exploration procedure: a symbolic version $SPre$ of the predecessor operator. The intuition behind its definition is as follows.

Let $M = (\mathcal{M} : \varphi)$ be a constrained configuration and $R = (\mathcal{A} \longrightarrow \mathcal{B} : \psi)$ be a rule. As shown in Fig. 1, to compute all configurations that can produce via R configurations in the denotation of M, we need to find a subconfiguration shared in common between M and the right-hand side of R (the result of the application of R), and add the remaining part of M to the left-hand side of R. As opposed to standard rewriting, this backward rewriting step is performed on subconfigurations of $\mathcal{M}$. The reason is that, since M denotes an upward closed set of configurations, we can always find ways to complete the selected subconfiguration with a multiset that makes possible the backward application of R. As a simple example, let $M = (a \mid b \ : \ true)$ and R be the rule $d \to b \mid c$. Although the right-hand side of R does not match M, the configuration $a \mid b \mid c$ obtained by adding c to M (i.e. it belongs to $[\![M]\!]$) is a successor of $a \mid d$, i.e., $a \mid d \in Pre([\![M]\!])$. The same result can be obtained by removing the subconfiguration b from M and by adding the remaining multiset to a.

Since all operations must be done at the constraint level, we first select two submultisets of $\mathcal{M}$ and $\mathcal{B}$ that can be unified via a constraint μ. Then, we build a new constrained configuration N by adding the left-hand side of the rule R to the subconfiguration obtained by removing from $\mathcal{M}$ the part in common with $\mathcal{B}$. To keep track of the relation among data variables, the unifier μ is conjoined with the constraints in M and R and then projected over the variables not occurring in the resulting multiset of atomic formulas. Variable elimination allows us to maintain only the data relations of interest for the variables in N. The formal definition of the symbolic predecessor operator is given below.

Given $M = (\mathcal{M} : \varphi)$,

$$
\begin{array}{rl}
SPre(M) = \{\ N \mid & \text{there exist } R \in \mathcal{R},\ \text{three multisets } \mathcal{M}',\ \mathcal{B}',\ \mathcal{N},\ \text{and a constraint } \mu \text{ s.t.} \\
& R = (\mathcal{A} \longrightarrow \mathcal{B} : \psi),\ \ \mathcal{M}' \preccurlyeq \mathcal{M},\ \ \mathcal{B}' \preccurlyeq \mathcal{B},\ \ (\mathcal{M}' : \varphi)\ =_\mu\ (\mathcal{B}' : \psi), \\
& \mathcal{N} = \mathcal{M} \ominus \mathcal{M}',\ N = (\mathcal{A} \oplus \mathcal{N}\ :\ \mu'),\ \mu' \equiv \exists x_1 \ldots . x_k . \mu \\
& \text{and } x_1, \ldots, x_k \text{ are the variables of } \mu \text{ that do not occur in } \mathcal{A} \oplus \mathcal{N} \}
\end{array}
$$

The operator $SPre$ is extended to a finite set S as follows

$$SPre(S) = \bigcup_{M \in S} SPre(M)$$

As an example, consider the rule

$$p(x, y) \longrightarrow q(x', y') : x \geq 0, y = 1, x' = x, y = y$$

and the constrained configuration $M = (q(u, w) : u = 1, w \geq 0)$. Then, $SPre(M)$ contains

$$p(x, y) : x = 1, y = 1$$

as well as

$$p(x, y) \mid q(u, w) : x \geq 0, y = 1, u = 1, w \geq 0$$

e.g., $p(4, 1) \mid q(1, 5)$ rewrites into $q(4, 1) \mid q(1, 5) \in [\![M]\!]$. The latter constrained configuration is obtained by setting $\mathcal{M}' = \mathcal{B}' = \epsilon$ (the empty multiset) when applying $SPre$ to M.

4.1 Properties of $SPre$

Theorem 1. *For any constrained configuration M*

$$[\![SPre(M)]\!] = Pre([\![M]\!])$$

Proof. Suppose that $M = (\mathcal{M} : \varphi)$.
(1) We first prove that $[\![SPre(M)]\!] \subseteq Pre([\![M]\!])$.
If $\mathcal{E} \in [\![SPre(M)]\!]$, then, by definition of $SPre$, there exist $R \in \mathcal{R}$, multisets $\mathcal{M}', \mathcal{B}', \mathcal{N}'$, and a unifier μ such that

- $\mathcal{M} = \mathcal{M}' \oplus \mathcal{N}$,
- $R = (\mathcal{A} \longrightarrow \mathcal{B} : \psi)$, and $\mathcal{B} = \mathcal{B}' \oplus \mathcal{B}''$,
- $(\mathcal{M}' : \varphi) \ =_\mu \ (\mathcal{B}' : \psi)$,
- $\mu' \equiv \exists x_1 \ldots . x_k . \mu$ where $x_1, \ldots, x_k$ are all variables not in $\mathcal{A} \oplus \mathcal{N}$
- and $\mathcal{E} = (\mathcal{A} \oplus \mathcal{N}\sigma \oplus \mathcal{N}')$ for some $\sigma \in Sol(\mu')$.

Now, it is easy to check that, since $\sigma \in Sol(\mu')$, then there exists $\gamma \in Sol(\mu)$ such that $\sigma = \gamma_{|V}$, where $V = Var(\mu) \setminus \{x_1, \ldots, x_k\}$.
From this property and by definition of unifier, we obtain that $\gamma \in Sol(\psi)$ and $\gamma \in Sol(\varphi)$.
As a consequence, we have that

$$(\mathcal{M}'\gamma \oplus \mathcal{N}\gamma) \in Inst(M)$$

and

$$R\gamma = (\mathcal{A}\gamma \longrightarrow (\mathcal{B}'\gamma \oplus \mathcal{B}''\gamma))$$

is an instance of R.
Furthermore, we have that

$$(\mathcal{M}'\gamma \oplus \mathcal{N}\gamma \oplus \mathcal{B}''\gamma \oplus \mathcal{N}') \in Up(Inst(()M)) = [\![M]\!]$$

Now since $\mathcal{M}'\gamma = \mathcal{B}'\gamma$ and $\gamma_{|V} = \sigma$ where V are the variables of $\mathcal{A}$ and $\mathcal{N}$, if we apply backward $R\gamma$ we obtain

$$\mathcal{E} = (\mathcal{A}\sigma \oplus \mathcal{N}\sigma \oplus \mathcal{N}') \in Pre([\![M]\!])$$

(2) Now we prove that $Pre(\llbracket M \rrbracket) \subseteq \llbracket SPre(M) \rrbracket$.
If $\mathcal{E} \in Pre(\llbracket M \rrbracket)$, then, by definition of Pre, there exists $\mathcal{E}' \in \llbracket M \rrbracket$ and an instance of $R \in \mathcal{R}$ whose backward application produces $\mathcal{E}$. This means that, in order to be applicable backward, the right-hand side, say $\mathcal{R}$, of the instance of R has to match a submultiset of $\mathcal{E}'$. Since, by definition, $\mathcal{E}' = \mathcal{M}\sigma \oplus \mathcal{Q}$ for $\sigma in Sol(\varphi)$ if $\mathcal{R} \preccurlyeq \mathcal{E}'$, then we can split $\mathcal{M}$ in two parts. A subconfiguration $\mathcal{M}'$ overlapping with $\mathcal{R}$ and the remaining part $\mathcal{N}$. The same holds for Q.
More precisely, we have that there exist $R \in \mathcal{R}$, multisets $\mathcal{M}', \mathcal{B}', \mathcal{N}'$, and substitutions σ and γ such that

- $\mathcal{M} = \mathcal{M}' \oplus \mathcal{N}$,
- $R = (\mathcal{A} \longrightarrow \mathcal{B} : \psi)$, and $\mathcal{B} = \mathcal{B}' \oplus \mathcal{B}''$,
- $\mathcal{M}'\sigma = \mathcal{B}'\gamma$, and $\mathcal{E} = (\mathcal{A}\gamma \oplus \mathcal{N}\sigma \oplus \mathcal{N}')$ for some $\sigma \in Sol(\varphi)$ and $\gamma \in Sol(\psi)$.

In other words we assume that $\mathcal{E}' = \mathcal{M}'\sigma \oplus \mathcal{N}\sigma \oplus \mathcal{B}''\gamma \oplus \mathcal{N}' \in \llbracket M \rrbracket$ and $\mathcal{E}$ is obtained by applying backwards $R\gamma$ to M'.
Now suppose $\mathcal{M}' = A_1 \mid \ldots \mid A_n$ and $\mathcal{B}' = B_1 \mid \ldots \mid B_n$. Since by hypothesis $\mathcal{M}'\sigma = \mathcal{B}'\gamma$ and $\mathcal{M}'$ and $\mathcal{B}'$ have distinct variables, it follows that there exists a permutation $j_1, \ldots, j_n$ of $1, \ldots, n$ such that the term $\langle A_1, \ldots, A_n \rangle$ and $\langle B_{j_1}, \ldots, B_{j_n} \rangle$ are unifiable via the mapping $\sigma \cup \gamma$.
Indeed, if $A_i = p(x_1, \ldots, x_k)$ and $B_{j_i} = q(y_1, \ldots, y_k)$, $p = q$ and constraint $x_1 = y_1 \wedge \ldots \wedge x_k = y_k$ is satisfiable via $\sigma \cup \gamma$).
Since φ and ψ have distinct variables, the mapping $\sigma \cup \gamma$ is also a solution for both constraints. Thus, $\sigma \cup \gamma$ is a solution of the constraint

$$\mu = \varphi \wedge \psi \wedge \bigwedge_{i=1}^{n} A_i = B_{j_i}$$

By definition of unifier, we have that

$$(\mathcal{M}' : \varphi) \ =_\mu \ (\mathcal{B}' : \psi)$$

holds.
Furthermore, the solutions of μ with respect to the set of variables in $\mathcal{A} \oplus \mathcal{N}$ coincide with the solutions of $\mu' \equiv \exists x_1. \ldots x_k.\mu$ if $x_1, \ldots, x_k$ are all variables not in $\mathcal{A} \oplus \mathcal{N}$.
Thus, by definition of $SPre$ we have that

$$(\mathcal{A} \oplus \mathcal{N} : \mu') \in SPre(M)$$

and, since $\sigma \cup \gamma \in Sol(\mu)$ we have that

$$(\mathcal{A}\gamma \oplus \mathcal{N}\sigma) \in Inst(SPre(M))$$

and by definition of upward closure we obtain

$$\mathcal{E} = (\mathcal{A}\gamma \oplus \mathcal{N}\sigma \oplus \mathcal{N}') \in Up(Inst(SPre(M))) = \llbracket SPre(M) \rrbracket$$

□

The following corollary immediately follows from Theorem 1.

Corollary 1. *Let S be a set of constrained configurations. Then,* $\llbracket SPre(S) \rrbracket = Pre(\llbracket S \rrbracket)$.

```
procedure SPre*(Spec, Entails, U) : set of constrained configurations
  Spec : MSR(C) spec.
  Entails : entailment
  U, R, S : set of constrained configurations
begin
  R := U;
  repeat
        S := R;
        R := S ∪ SPre(S);
  until  (Entails^S(R, S) = true)
  return R;
end
```

Fig. 2. Symbolic Backward Reachability

5 The Symbolic Backward Reachability Procedure

We are ready now to define a symbolic backward exploration procedure working on sets of assertions. Given an MSR($\mathcal{C}$) specification *Spec*, an entailment relation *Entails* for constrained configurations, its pointwise extension to sets $Entails^S$, and a set U of constrained configurations (with pairwise distinct variables) representing the set of bad states, the symbolic backward reachability procedure is given in Fig. 2. Starting from U the procedure $SPre^*$ attempts to computes a symbolic representation of $Pre^*(\llbracket U \rrbracket)$, by repeatedly applying $SPre$ to all stored constrained configurations (variable S). In other words, it computes a chain

$$I_0 \subseteq I_1 \subseteq I_2 \ldots I_i \ldots$$

where

$$I_0 = U$$
$$I_{i+1} = I_i \cup SPre(I_i)$$

The procedure stops when $SPre$ produces only redundant information, i.e., $Entails^S(I_{i+1}, I_i)$. Note that $Entails^S(I_i, I_{i+1})$ always holds since $I_i \subseteq I_{i+1}$.

If the fixpoint computation terminates we check that the initial configurations $\mathcal{I}$, representing the *initial states* of the system, are not contained in the denotation of the resulting set of constrained configurations (i.e., $\mathcal{I} \cap \llbracket SPre^*(U) \rrbracket = \emptyset$).

Following from what explained in Section 2.2, there exists a constraint system $\mathcal{C}$ such that, on some input, the symbolic backward exploration procedure does no terminate (for any entailment relation we take). Our procedure enjoys however the following properties.

- If the *Entails* is decidable and $SPre$ is effective (i.e. satisfiability is decidable and variable elimination in $\mathcal{C}$ is computable), then every step in the loop of Fig. 2 is effective.
- Furthermore, from Corollary 1 if the procedure terminates on input U it computes a finite, symbolic representation R of the infinite set $Pre^*(\llbracket U \rrbracket)$.
- Finally, if the test $\mathcal{I} \cap \llbracket R \rrbracket$ is decidable, then the procedure can be used to solve the control reachability problem $\mathcal{I} \cap Pre^*(\llbracket U \rrbracket) = \emptyset$.

It is important to recall that the control reachability problem is linked to verification of safety properties. Suppose that U represents all possible violations to a safety property. If the invocation $SPre^*(U)$ terminates and computes a fixpoint R, then the property holds if and only if $\mathcal{I} \cap [\![R]\!] = \emptyset$. Thus, our method can be applied to verify properties and not only to find errors for specifications of concurrent systems parametric on the number of processes and with data defined over infinite domains.

We will study sufficient conditions for termination in Section 6. However, as shown in [7, 8, 19], even for instances of MSR($\mathcal{C}$) for which theoretical termination is not guaranteed, the procedure equipped with the *GenEntails* entailment relation often terminates on practical examples.

6 A Decidable Fragment: Monadic MSR($\mathcal{N}$)

In this section we will study the properties of a monadic fragment of the instance MSR($\mathcal{N}$) of our language scheme. $\mathcal{N}$-constraints are the subclass of linear arithmetic constraints whose formulas are generated by the following grammar:

$$\varphi \ ::= \ \varphi \wedge \varphi \mid x = y \mid x < y \mid x \leq y \mid x > y \mid x \geq y \mid true \mid false$$

In this paper we will interpret $\mathcal{N}$-constraints over reals or rational numbers.

A Monadic MSR($\mathcal{N}$) specification is such that all predicates have arity *less or equal than one*, i.e. atomic formulas have the form p or $p(x)$ for some variable x. We will call $\mathcal{A}$ the set of constrained configurations of a Monadic MSR($\mathcal{N}$) specification.

As an example, the rule $a \mid p(x) \mid q(y) \rightarrow q(z) : x > y \wedge y = z$ is a Monadic MSR($\mathcal{N}$) rule, and $p(x) \mid q(y) : x > y$ is an element of $\mathcal{A}$.

The following property then holds.

Lemma 1. *$\mathcal{A}$ is closed under applications of $SPre$ to a Monadic MSR($\mathcal{N}$) specification.*

Proof. Under the above hypothesis, the application of the operator $SPre$ cannot introduce n-ary predicates: it manipulates only formulas of the specification and of the considered set of constrained configurations, unification does not introduce new predicates. Furthermore, given an $\mathcal{N}$-constraint φ, $\exists x.\varphi$ is equivalent to an $\mathcal{N}$-constraint without existential quantification. Indeed, when executed on $\mathcal{N}$-constraints standard variable elimination algorithms (e.g. Fourier-Motzkin's procedure) returns a simpler $\mathcal{N}$-constraints (e.g. $\exists y.x < y < z$ is simplified to $x < z$). $\square$

The other property we will prove is that, with an adequate choice of the entailment relation, the procedure $SPre^*$ of Fig. 2 always stops after a finite number of steps.

6.1 Rationale Behind the Termination Proof

The other property we will prove is that, with an adequate choice of the entailment relation, every sequence of sets of constrained configurations built by procedure $SPre^*$ of Fig. 2 is always finite. To prove this property we can resort to the following lemma (instantiated with our notations for entailment relation). [1].

Lemma 2 (Lemma 3.1 of [30]). *Let $\mathcal{C}$ be a constraint system representing upward closed sets of elements in the domain $\mathcal{D}$, and Entails an entailment relation. Entails is a well quasi ordering if and only if for each infinite chain of sets of constraints $I_0 \subseteq I_1 \subseteq \ldots$ there exists i such that $I_{j+1} Entails^S I_j$ for all $j \geq i$.*

This implies that if $Entails$ is a well quasi ordering than $SPre^*$ always terminates. By instantiating the definition in Appendix, $Entails$ is a *well quasi ordering* (wqo) if for each infinite sequence of constrained configurations $M_0\ M_1\ M_2\ \ldots$ there exist indexes $i < j$ such that $Entails(M_j, M_i)$. Classical examples of wqo's are multiset and string embedding (see Appendix B).

To define a well quasi ordered entailment for MSR($\mathcal{N}$), we resort to an encoding of constrained configurations into a special kind of strings so as to apply the properties of well quasi orderings described in Appendix.

Specifically, we will show that monadic constrained configurations can be represented using strings defined over multisets of predicate symbols. We will use this simpler representation to define a well-quasi ordering on constrained configurations by using classical properties of strings and multisets. To illustrate, let us consider the following constrained configuration

$$M = (p(x)|q(y)|r(z) : x > y, y = z)$$

If we partition the variables according to the equalities in the constraint, we obtain two equivalence classes $C_1 = \{x\}$ and $C_2 = \{y, z\}$. Furthermore, if we extend the order on variables to the classes they belong to, we obtain that $C_1 > C_2$. Now, let us define $\mathcal{M}_i$ as the multiset containing all predicate symbols $p \in P$ such that $p(x)$ occurs in M and $x \in C_i$, namely $\mathcal{M}_1 = \{p\}$ and $\mathcal{M}_2 = \{q, r\}$. It is immediate to see that we can represent M as the string $\mathcal{M}_1 \cdot \mathcal{M}_2$, i.e.,

$$\{p\} \cdot \{q, r\}$$

Since we only need to maintain the relation between the parameters of different predicates, such a string retains all information we have in M. As an example, the string

$$\{p, q\} \cdot \{p, r\} \cdot \{q, r\}$$

can be viewed as a compact representation of constrained configurations like

$$p(x) \mid q(y) \mid p(z) \mid r(t) \mid q(u) \mid r(v) \ : \ x = y, z = t, u = v, x > z, z > u$$

Variables can be renamed without changing the denotations of the resulting formula.

Unfortunately, the string representation only works for constraints in which the set variables V occurring in the constrained configuration is totally ordered. However, for any $\mathcal{N}$-constraint φ we can compute a finite set $Comp(\varphi, V)$ of constraints that satisfy this property and such that φ and $Comp(\varphi, V)$ have the same solutions. Thus, given two constrained configuration N and M we can define an entailment by using the pointwise ordering $CompEntails^S$ (see Section 3.4) over the sets $Comp(N)$ and $Comp(M)$ obtained by the linearizations of constraints. The order $CompEntails$ on their elements is defined by using a combination of stringa and multiset embeddings.

This transformation creates however another problem. Suppose we manage to prove that $CompEntails$ is a well quasi ordering. We would like to infer that $CompEntails^S$ is a well quasi ordering too. However, as shown by Rado's counterexample [3, 30], well quasi ordering are not closed under finite set with the pointwise relation. This closure property is satisfied however by better quasi orderings, a stronger notion that has been defined to isolate wqo's that do not suffer from Rado's counterexample. The definition of better quasi ordering is fairly complicated. The intuition together with the properties we will exploit in our proof of termination is shown in Appendix B.

Summarizing, our termination proof consists of the following steps:

- We first define the notion of complete constraint configuration.
- We encode complete constrained configurations as strings over multisets. We order them using string and multiset embedding ($CompEntails$ and show that this gives us an entailment.
- We prove that $CompEntails^S$ form a well quasi ordering by resorting to the properties of better quasi orderings.
- We conclude by applying Lemma 2.

In the rest of this section we will formalize the proof.

6.2 Completed Constrained Configurations

Let $M = (\mathcal{M} : \varphi)$ be a constrained configuration with variables in V and φ be an $\mathcal{N}$-constraint. In general a constraint φ induces a partial order on the variables in V. As an example, in $p(x)|q(y)|p(z) : x < y$ there are no explicit constraints between x and z. In order to transform a constrained configuration into a string we first need to find all linearizations induced by φ on V. For this purpose, we simply have to build the finite set of satisfiable constraints $Comp(\varphi, V)$ obtained by conjoining φ with either $x = y$, $x < y$ or $x > y$ for every pair x, y of variables in V. Since constraints like $x \geq y, \varphi$ can be split into the two constraints $x = y, \varphi$ and $x > y, \varphi$ included in $Comp(\varphi, V)$, we can restrict ourattention to $=$ and $>$ with loss of generality. Clearly, it holds that the solutions of φ coincide with the solutions of the disjunction of all constraints in $Comp(\varphi, V)$, i.e.,

$$Sol(\varphi) = \bigcup_{\psi \in Comp(\varphi, V)} Sol(\psi)$$

This property reflects the total order of the interpretation domain of $\mathcal{N}$-constraints (according to it, we consider all possible total ordering of variables).

In the previous example $Comp(\varphi, \{x, y, z\})$ consists of the constraints $\varphi_1 \equiv x < y = z$, $\varphi_2 \equiv x = z < y$, $\varphi_3 \equiv x < y < z$, $\varphi_4 \equiv x < z < y$, and $\varphi_5 \equiv z < x < y$. Furthermore, we have that $\varphi \equiv \varphi_1 \vee \ldots \vee \varphi_5$. Based on this idea, given $M = (\mathcal{M} : \varphi)$ with variables in V, we define the following set

$$Comp(M) = \{(\mathcal{M} : \psi) \mid \psi \in Comp(\varphi, V)\}$$

Elements of $Comp(M)$ are called completed constrained configurations to indicate that the partial order in φ has been completed into a total order on V. This definition is extended to sets of constrained configurations as follows

$$Comp(S) = \bigcup_{M \in S} Comp(M)$$

Since $Comp(\varphi, V)$ is a finite set and $Sol(\varphi) = Sol(Comp(\varphi, V))$ we have that

$$[\![Comp(S)]\!] = [\![S]\!]$$

We will only work with completed constrained configurations.

6.3 Compact Representation of Constrained Configurations

Let $M = (\mathcal{M} : \varphi)$ be a completed constrained configuration with variables in V.

The symmetric and transitive closure of the relation $=$ induces an equivalence relation on V. Let $V_= = \langle C_1, \ldots, C_r \rangle$ be the resulting set of equivalence classes. By assumption, V is totally ordered by φ. This total order can be extended to $V_=$, i.e., we can find a permutation $i_1, \ldots, i_r$ such that $C_{i_1} > \ldots > C_{i_r}$. Here $C > C'$ means that for their representative elements, say x and y, we have that $x > y$.

Now, for $i : 1, \ldots, r$ let M_i be the maximal *multiset* of predicate symbols $\{p_1, \ldots, p_m\}$ such that $p_k(x)$ occurs in $\mathcal{M}$ and $x \in C_i$ for $k = 1, \ldots, m$. Under all the previous assumptions, we define the compact representation of M as the following string

$$Str(M) = M_{i_1} \cdot \ldots \cdot M_{i_r}$$

where $\cdot$ is a string constructor.

6.4 Entailment of Monadic Constrained Configurations

We are ready now to define an entailment relation over $\mathcal{A}$. We do this in three steps. We first define an ordering *StrEntails* between string representations of completed constrained configurations. Then, we use *StrEntails* to define an entailment *CompEntails* for completed constrained configurations. Finally, we use *CompEntails* and the pointwise ordering to define the entailment relation *CEntails* over constrained configurations of Monadic MSR($\mathcal{N}$). Formally, we have the following definitions.

- Given two *strings* $S = M_1 \cdot M_2 \ldots \cdot M_k$ and $T = N_1 \cdot N_2 \cdot \ldots \cdot N_r$, $StrEntails(T, S)$ holds if and only if there exists an injective and strictly monotone mapping h from $1, \ldots, k$ to $1, \ldots, r$ such that $M_i \preccurlyeq N_{h(i)}$ for $i : 1, \ldots, k$. A mapping h is stictly monotone if $i < j$ implies $h(i) < h(j)$. Thus, *StrEntails* is string entailment built over multiset inclusion on the string alphabet.
- Given two completed constrained configurations N and M,

$$CompEntails(N, M) \equiv StrEntails(Str(N), Str(M))$$

- Given N and M in $\mathcal{A}$,

$$CEntails(N, M) \equiv CompEntails^S(Comp(N), Comp(M))^1$$

Example 3. Consider the following definitions

- $Comp(M) = \{M\}$ and $Comp(N) = \{N_1, N_2\}$,
- $Str(M) = \{p, p\} \cdot \{q\} \cdot \{r, r\}$,
- $Str(N_1) = \{p, p, p\} \cdot \{t, t\} \cdot \{q, q\} \cdot \{r, r, r\}$ and $Str(N_2) = \{p\} \cdot \{p, t, t\} \cdot \{p, q, q\} \cdot \{r, r, r\}$

$CompEntails(N_1, M)$ then holds. Indeed, we can map $\{q\}$ to $\{q, q\}$ ($\{q\} \preccurlyeq \{q, q\}$), $\{r, r\}$ to $\{r, r, r\}$ ($\{r, r\} \preccurlyeq \{r, r, r\}$), and $\{p, p\}$ to $\{p, p, p\}$ ($\{p, p\} \preccurlyeq \{p, p, p\}$).

Furthermore, since there is no way to map $\{p, p\}$ to some of the multisets of N_2, it follows that $CompEntails(N_2, M)$ does not hold. As a consequence $CEntails(N, M)$ does not hold either, i.e., N and M represent distinct (upward closed) sets of configurations.

[1] $CEntails(N, M)$ if and only if for all $N' \in Comp(N)$ there exists $M' \in Comp(M)$ such that $CompEntails(N', M')$, (see Section 3.4).

The following property then holds.

Lemma 3. *The relation $CEntails(N, M)$ is an entailment for $\mathcal{A}$.*

Proof. Following from Corollary 1, it is enough to show that if $CompEntails(N, M)$ holds, then $[\![N]\!] \subseteq [\![M]\!]$. Let $Str(M) = M_1 \cdot M_2 \cdot \ldots \cdot M_k$ and $Str(N) = N_1 \cdot N_2 \cdot \ldots \cdot N_r$, and suppose that $CompEntails(N, M)$ holds, i.e., $StrEntails(Str(N), Str(M))$ holds. Then, for every M_i there exists a distinct $N_{h(i)}$ ($h(i) < h(j)$ for $i < j$) such that $M_i \preccurlyeq N_{h(i)}$.

- If $C \in [\![N]\!]$ and $N = \mathcal{N} : \varphi$, then $C = \sigma(\mathcal{N}) \oplus D$ where $\sigma \in Sol(\varphi)$.
- Since $\sigma \in Sol(\varphi)$, it follows that $\sigma(x) = \sigma(y) = v_i$ for every two variables in the i-th cluster of N, and $v_i > v_j$ for $i < j$.
- Thus,
$$C = C_1 \oplus \ldots \oplus C_r \oplus D,$$
where
$$C_i \;=\; p_{i1}(v_i) \mid p_{i2}(v_i) \mid \;\ldots\; \mid p_{im_i}(v_i),$$
where $N_i = p_{i1}p_{i2}\ldots p_{im_i}$ and $v_i > v_j$ if $i < j$ for $i, j : 1, \ldots, r$
- Now let $E \;= E_1 \oplus \ldots \oplus E_k$ where
$$E_i \;=\; q_{i1}(v_{h(i)}) \mid q_{i2}(v_{h(i)}) \mid \;\ldots\; \mid q_{im_i}(v_{h(i)}),$$
where $M_i = q_{i1}q_{i2}\ldots q_{iz_i}$ for $i : 1, \ldots, k$ and $v_i > v_j$ if $i < j$ for $i, j : 1, \ldots, k$. Then, by the definition of $Str(M)$, $E \in [\![M]\!]$.
- Furthermore, since, by hypothesis, there exists $N_{h(1)}, \ldots, N_{h(k)}$ such that $M_i \preccurlyeq N_{h(i)}$ ($h(i) < h(j)$ for $i < j$) for $i : 1, \ldots, k$, it follows that $E \preccurlyeq C$, i.e. $C \in [\![M]\!]$.

□

Following the methodology proposed in [1, 22], to show that symbolic backward reachability always terminates, we need to prove that the entailment relation $CEntails$ is a well quasi ordering. However, well quasi orderings are not closed under the pointwise relation ^{S} we used to define $CEntails$ (see [3]). However, the stronger notion of better quasi ordering (see Appendix B) satisfies this property (last point of Prop. 4). We will apply this property to $CEntails$ in two steps. We first need the following lemma.

Lemma 4. *The entailment relation $CompEntails$ is a better quasi ordering.*

Proof. We first recall that multiset inclusion $\preccurlyeq$ is a better quasi ordering. We use here the property (iii) of Prop. 4 in Appendix B. Since $CompEntails$ is built over elements ordered with the better quasi ordering $\preccurlyeq$, then $CompEntails$ is a better quasi ordering. □

As an immediate consequence, we obtain the following property.

Proposition 3. *The entailment relation $CEntails$ is a well quasi ordering.*

Proof. From the last property of Prop. 4 in Appendix, the pointwise ordering of sets of elements that are part of a better quasi ordering is still a better quasi ordering. Thus, from Lemma 4, it follows that $CEntails$ is a better quasi ordering, and thus a well quasi ordering.

□

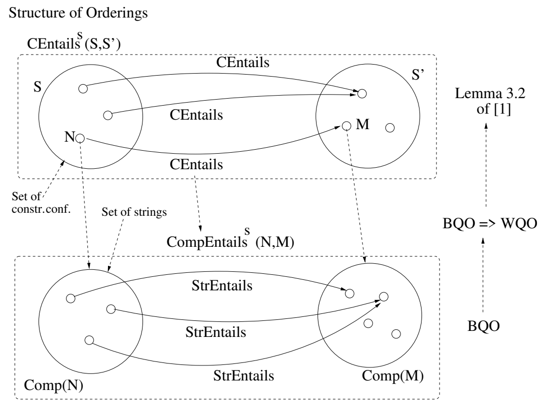

Fig. 3. Structure of the orderings used in the proof of Theorem 2.

These results allows us to prove the termination of symbolic backward reachability for Monadic MSR($\mathcal{N}$) specifications and sets of assertions.

Theorem 2. *The backward reachability algorithm of Section 2 with entailment CEntails is guaranteed to terminate when taking in input a Monadic MSR($\mathcal{N}$) specification and a set of assertions $U \subseteq \mathcal{A}$.*

Proof. It immediately follows from Prop. 3 and Lemma 2. □

In Fig. 3 we summarize once again the structure of the proof. As shown in the figure, we used the pointwise ordering at two different levels: to order sets of configurations via $CEntails^S$, and to order sets of completed configurations when comparing two single constrained configurations via $CEntails$ (defined as $CompEntails^S$). It is important to note that Lemma 2 can be applied to $CEntails^S$ (while it has no use for $CompEntails^S$) because the sets generated by symbolic exploration form a chain (i.e. an increasing sequence) with respect to inclusion. On the other hand, the properties of better quasi orderings are needed to prove that $CompEntails^S$ is a well quasi ordering.

7 Undecidability of Dyadic MSR($\mathcal{N}$)

In this section we will show that Monadic MSR($\mathcal{N}$) is strictly less expressive than Dyadic MSR($\mathcal{N}$), i.e., the fragment of MSR($\mathcal{N}$) in which predicates have arity less of equal than two. To prove this property, we resort again to an encoding of Two Counters Machines. Every counter is represented here via a doubly linked list implemented via a multiset of atomic formulas built on the binary predicate c and a sentinel cell implemented via the predicate s_i. The sentinel points to the head and tail of the list. The formula $s_i(n, n)$ for some value n represents the i-th counter with value zero. The i-th counter with value k is represented via the list

$$s_i(v_0, v_k) \mid c(v_0, v_1) \mid \ \ldots \ \mid c(v_{k-1}, v_k)$$

with $v_0 < v_1 < \ldots < v_k$ represents The current control location p is stored as an atomic formula $l(p)$. The operations on the i-th counter are simulated via insertion and deletion of cells from the list as shown below[2]

$$
\begin{array}{llll}
\text{Zero}: & l(p) \mid s_i(x,y) & \to l(p') \mid s_i(x,y) & : x = y \\
\\
\text{Inc0}: & l(p) \mid s_i(x,y) & \to l(p') \mid s_i(x,z) \mid c(x,z) & : x = y, x > u > z \\
\text{Inc1}: & l(p) \mid s_i(x,y) \mid c(z,y) & \to l(p') \mid s_i(x,y) \mid c(z,u) \mid c(u,y) & : y > u > z > x \\
\\
\text{Dec0}: & l(p) \mid s_i(x,y) \mid c(x,y) & \to l(p') \mid s_i(x,z) & : x > y, x = z \\
\text{Dec1}: & l(p) \mid s_i(x,y) \mid c(z,u) \mid c(u,y) & \to l(p') \mid s_i(x,y) \mid c(z,y) & : y > u > z > x
\end{array}
$$

Every rule updates the control location as prescribed by the counter machine transition relation. In rule Zero we test if the counter c_i is zero by simply testing that an instance of $s_i(x,x)$ is in the current configuration. To ensure that the pointers in the sentinel are always well-defined, we have to distinguish two cases for the increment operation. In rule Inc0 we first check if the counter is zero, i.e., $s_i(n,n)$ is in the current configuration, and then we create a new cell, pick up a value n' such that $n' > n$ and create the subconfiguration $s_i(n,n') \mid c(n,n')$. In rule Inc1 we first check that the counter is greater or equal than one, then we add a new cell in the tail of the list and we adjust the pointers in the sentinel. In rule Dec0 we first check if the counter is one, i.e., $s_i(n,n') \mid c(n,n')$ is in the current configuration, and then we remove the cell and reset the sentinel to $s_i(n,n)$. Finally, in rule Dec1 we first check if the counter is greater than one, then we remove the tail and we adjust the pointers in the sentinel. The initial configuration $\langle Loc = p_0, c_1 = 0, c_2 = 0\rangle$ of the Two Counters Machine is defined by the rule

$$\text{Init}: init \Rightarrow l(p_0) \mid s_1(x,y) \mid s_2(u,v): \ x = 0, y = 0, u = 0, v = 0$$

It is immediate to see that every step of the Two Counters Machine is simulated by a rewriting step of the above described dyadic MSR($\mathcal{N}$) specification.

Since reachability and control state reachability are both undecidable for Two Counters Machine, we conclude that Monadic MSR($\mathcal{N}$) is strictly less expressive than Dyadic MSR($\mathcal{N}$).

8 A Case-study: the Ticket Algorithm

In this section we present one possible application of the decidability result presented in Section 6 to the analysis of a well-known mutual exclusion protocol called the ticket algorithm. The ticket algorithm is a protocol designed for multi-client systems operating on a shared memory. The protocol is based on a first-in first-served access policy. The algorithm is given in Fig. 4 (where we use $P \mid Q$ to denote the *interleaving parallel execution* of P and Q, and $\langle\cdot\rangle$ to denote atomic fragments of code). The protocol works as follows. Initially, all clients are thinking, while t and s store the same initial value. When requesting the access to the critical section, a client stores the value of the current ticket t in its local variable a. A new ticket is then emitted by incrementing t. Clients wait for their turn until the value of their local variable a equals the value of s. After the elaboration inside the critical section, a process releases it and the current turn is updated by incrementing s. During the execution, the *global state* of the protocol consists of the current values of s, t, and of the local variables

[2] For the sake of simplicity we write rules like $q(x,y) \mid \ldots \Rightarrow q(x',y') \mid \ldots : x' = x, y' = y, \ldots$ with multiple occurrences of the same variables, i.e., $q(x,y) \mid \ldots \Rightarrow q(x,y) \mid \ldots : \ldots$.

THE SYSTEM WITH n PROCESSES

```
Program
  global var s, t : integer;
  begin
    t := 0;
    s := 0;
    P_1 | ... | P_n;
  end.
```

THE i-th COMPONENT

```
Process P_i ::=
  local var a : integer;
  repeat forever
    [ think :  ⟨ a := t;
                 t := t + 1; ⟩
      wait :   when ⟨ a = s ⟩ do
      use :         [ CRITICAL SECTION
                    [ ⟨ s := s + 1; ⟩
  end.
```

Fig. 4. The ticket algorithm: n is a parameter of the protocol.

of n processes. As remarked in [11], even for $n = 2$ (only 2 clients), the values of the local variables of individual processes as well as s and t may get *unbounded*. This implies that *any instance* of the scheme of Fig. 4 gives rise to an infinite-state system. The algorithm is supposed to work for any value of n, and it should also work if new clients enter the system at running time.

8.1 Specification of the Protocol

To specify the ticket algorithm we can resort to the instance MSR($\mathcal{L}$) with linear integer arithmetic constraints. Let us consider a *single* shared resource controlled via the counters t and s. The infinite collection of admissible initial states consists of all configurations with an *arbitrary* but finite number of thinking processes and *two* counters having the same initial value ($t = s$). The specification is shown in Fig. 5. The initial configuration is the predicate *init*, the seed of all possible runs of the protocol. The counters are represented here via the atoms $count(t)$ and $turn(s)$. Thinking clients are represented via the propositional symbol *think*, and can be generated *dynamically* via the second rule. The behavior of an individual client is described via the third block of rules of Fig. 5, in which the relation between the local variable and the global counters are represented via linear arithmetic constraints. Finally, we allow thinking processes to terminate their execution as specified via the last rule of Fig. 5. The previous rules are independent of the current number of clients in the system. Note that in our specification we keep an explicit representation of the data variables; furthermore, we do not place any restriction on their values. As a consequence, there are runs of our model in which s and t have no upper bounds as in the original protocol. A sample run of a system with 2 clients (as in [11]) is shown in Fig. 6. The set of violations to mutual exclusion for our case-study can be represented through the constrained configurations

$$U = (use(x) | use(y) : true)$$

where x and y are left unconstrained, i.e., any configuration that contains at least two occurrences of the predicate *use* violates mutual exclusion. In the following we will use $Ticket$ to refer to the MSR($\mathcal{L}$) specification of Fig. 5. The control reachability problem consist then in proving that starting from $\mathcal{M}_0 = \{init\}$ it is not possible to reach a configuration $\mathcal{M} \in [\![U]\!]$ using the rules in $Ticket$. Unfortunately, the application of our symbolic backward search to $Ticket$ and U does not terminate when using a solver for $\mathcal{L}$-constraints.

One possible way to enforce termination is to try to compute an abstract model of the ticket

$$
\begin{array}{c}
\textbf{Initial States} \\
init \;\longrightarrow\; count(t) \mid turn(s) \;:\; t = s \\
\textbf{Dynamic Generation} \\
\epsilon \;\longrightarrow\; think \;:\; true \\
\textbf{Individual Behavior}
\end{array}
$$

$$
\begin{array}{rcll}
think \mid count(t) & \longrightarrow & wait(a) \mid count(t') & : \; a = t \;\wedge\; t' = t + 1 \\
wait(a) \mid turn(s) & \longrightarrow & use \mid turn(s') & : \; a = s \;\wedge\; s' = s \\
use \mid turn(s) & \longrightarrow & think \mid turn(s') & : \; s' = s + 1
\end{array}
$$

$$
\begin{array}{c}
\textbf{Termination} \\
think \;\longrightarrow\; \epsilon \;:\; true
\end{array}
$$

Fig. 5. Ticket protocol for *multi-client*, *single-server* system, with an example of run.

$$
\begin{array}{c}
init \Rightarrow \ldots \Rightarrow think \mid count(8) \mid turn(8) \Rightarrow think \mid think \mid count(8) \mid turn(8) \\
\Rightarrow wait(8) \mid think \mid count(9) \mid turn(8) \Rightarrow wait(8) \mid wait(9) \mid count(10) \mid turn(8) \\
\Rightarrow use \mid wait(9) \mid count(10) \mid turn(8) \Rightarrow use \mid wait(9) \mid count(10) \mid turn(8) \mid think \\
\Rightarrow think \mid wait(9) \mid count(10) \mid turn(9) \mid think
\end{array}
$$

Fig. 6. Example of run.

algorithm in the monadic fragment of MSR($\mathcal{N}$). The specification $Ticket$ is not directly expressible in Monadic MSR($\mathcal{N}$) for two reasons: (1) $\mathcal{L}$-constraints are interpreted over integers and not over reals or rational numbers as $\mathcal{N}$-constraints; (2) we need arithmetic operations like $t' = t + 1$ to update the tickets.

The first abstraction consists in applying a relaxation from integers to reals or rationals of the interpretation of the constraints in $Ticket$. The second abstraction is obtained by replacing the first and last protocol rule with the following ones:

$$
\begin{array}{rcll}
think \mid count(t) & \longrightarrow & wait(a) \mid count(t') & : \; a = t \;\wedge\; t' > t \\
use \mid turn(s) & \longrightarrow & think \mid turn(s') & : \; s' > s
\end{array}
$$

We call $AbsTicket$ the resulting Monadic MSR($\mathcal{N}$) specification. Clearly, the set of reachable configurations of $AbsTicket$ overapproximates that of $Ticket$. However, symbolic backward reachability is guaranteed to terminate when applied to $AbsTicket$ and U. Furthermore, $Pre^*_{AbsTicket}([\![U]\!])$ overapproximates $Pre^*_{Ticket}([\![U]\!])$. Thus, if M_0 does not occur in the set returned by $SPre^*$ applied to $AbsTicket$ and U, then mutual exclusion holds for $Ticket$, too.

The backward search engine has been implemented in SICStus CLP with the CLP(Q,R) constraint solver for linear arithmetics. All experiments have been executed on a Pentium III 450 Mhz, Linux 2.2.13-0.9.

When executed on the MSR($\mathcal{L}$) specification $Ticket$, we had to stop the fixpoint computation after several hours of computation (and we manually verified that the fixpoint computation diverges when constraints are interpreted over integers or reals/rationals).

However, when executed on the Monadic MSR($\mathcal{N}$) specification $AbsTicket$ the procedure terminates after 17 steps. Since $init$ does not occur in the computed fixpoint, we automatically proved mutual exclusion for a version of the ticket algorithm parametric on the number of clients and without bounds on the values of ticket distributed among them.

Acknowledgments I would like to thank Moshe Vardi for several precious suggestions and references, Parosh A. Abdulla, Bengt Jonsson, and Michael Rusinowitch for fruitful discussions, and James Larus and Microsoft Research for having supported this research.

References

1. P. A. Abdulla, K. Cerāns, B. Jonsson, and T. Yih-Kuen. Algorithmic Analysis of Programs with Well Quasi-Ordered Domains. To appera in the Journal of Information and Computation, 2000.
2. P. A. Abdulla and B. Jonsson. Verifying Networks of Timed Processes. In B. Steffen, editor, *Proceedings 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *LNCS*, pages 298–312, Lisbon, Portugal, 1998. Springer-Verlag.
3. P. .A. Abdulla and A. Nylén. Better is Better than Well: On Efficient Verification of Infinite-State Systems. In *Proceedings 15th Annual International Symposium on Logic in Computer Science (LICS'00)*, pages 132–140, Santa Barbara, California, 2000. IEEE Computer Society Press.
4. T. Arons, A. Pnueli, S. Ruah, Y. Xu, and L. D. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 221–234, Paris, France, 2001. Springer-Verlag.
5. K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In S. Graf and M. I. Schwartzbach, editors, *Proceedings 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 188–203, Berlin, Germany, 2000. Springer-Verlag.
6. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular Model Checking. In E. A. Emerson and A. P. Sistla, editors, *Proceedings 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 403–418, Chicago, Illinois, 2000. Springer-Verlag.
7. M. Bozzano, G. Delzanno. Beyond Parameterized Verification. In *Proc. CAV '02*, 2002. In J.-P. Katoen, P.Stevens, editors, *Proceedings 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *LNCS*, pages 221-235, Grenoble, France, 2002. Springer-Verlag.
8. M. Bozzano, G. Delzanno. Automatic Verification of Invalidation-based Protocols. In E. Brinksma, K. G. Larsen, editors, *Proceedings 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 295-308, Copenhagen, Denmark, 2002. Springer-Verlag.
9. M. Bozzano, G. Delzanno, and M. Martelli. Model Checking Linear Logic Specifications. TPLP 4(5-6): 573-619 (2004)
10. M. C. Browne, E. M. Clarke, O. Grumberg. Reasoning about Networks with Many Identical Finite State Processes. *Information and Computation* 81(1): 13–31, 1989.
11. T. Bultan, R. Gerber, and W. Pugh. Symbolic Model Checking of Infinite State Systems Using Presburger Arithmetics. In O. Grumberg, editor, *Proceedings 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 400–411, Haifa, Israel, 1997. Springer-Verlag.
12. I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A Meta-notation for Protocol Analysis. In R. Gorrieri, editor, *12th Computer Security Foundations Workshop (CSFW'99)*, pages 55–69, Mordano, Italy, 1999.

13. H. Cirstea. Specifying Authentication Protocols Using Rewriting and Strategies. In *Proc. of PADL'01*, *LNCS 1990*, pages 138–152, 2001.
14. E. Clarke, O. Grumberg, S. Jha. Verifying Parameterized Networks. *TOPLAS* 19(5): 726–750, 1997.
15. G. Delzanno. Automatic Verification of Parameterized Cache Coherence Protocols. In E. A. Emerson and A. P. Sistla, editors, *Proceedings 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, Chicago, Illinois, 2000. Springer-Verlag.
16. G. Delzanno. An Assertional Language for Systems Parametric in Several Dimensions. In R. Mayr, editor, *Proceedings Workshop on Verification of Parameterized Systems (VEPAS '01)*, volume 50 of *ENTCS*, Crete, Greece, 2001. Elsevier Science.
17. G. Delzanno. Constraint-Based Model Checking for Parameterized Synchronous Systems. In A. Armando, editor, *Proceedings of the 4th International Workshop on Frontiers of Combining Systems (FroCoS 2002)*, volume 2309 of *LNCS*, pages 72-86, 2002. Springer-Verlag.
18. G. Delzanno. A Symbolic Procedure for Control Reachability in the Asynchronous Pi-calculus. Electronic Notes in Theoretical Computer Science 98: 21-33 (2004)
19. G. Delzanno, and P. Ganty. Automatic Verification of Time Sensitive Cryptographic Protocols. In K. Jensen and A. Podelski, editors, *Proceedings of 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, Lecture Notes in Computer Science 2988, pages 342–356, Springer 2004.
20. G. Delzanno and A. Podelski. Model checking in CLP. In R. Cleaveland, editor, *Proceedings 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 223–239, Amsterdam, The Netherlands, 1999. Springer-Verlag.
21. G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In *Proc. of CSFW'98*, 1998.
22. A. Finkel and P. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
23. L. Fribourg. Constraint Logic Programming Applied to Model Checking. In A. Bossi, editor, *Proceedings 9th International Workshop on Logic Program Synthesis and Transformation (LOPSTR'99)*, volume 1817 of *LNCS*, pages 30–41, Venezia, Italy, 1999. Springer-Verlag.
24. S. M. German and A. P. Sistla. Reasoning about Systems with Many Processes. *Journal of the ACM*, 39(3):675–735, 1992.
25. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proc. of LPAR'00*, *LNCS 1955*, pages 131–160, 2000.
26. B. Jonsson and M. Nilsson. Transitive Closures of Regular Relations for Verifying Infinite-State Systems. In S. Graf and M. I. Schwartzbach, editors, *Proceedings 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 220–234, Berlin, Germany, 2000. Springer-Verlag.
27. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In O. Grumberg, editor, *Proceedings 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 424–435, Haifa, Israel, 1997. Springer-Verlag.
28. A. Marcone. Fine and Axiomatic Analysis of the quasi-orderings on $\mathcal{P}(q)$. Dipartimento di Matematica e Informatica dell'Università di Udine. Technical Report 17/99/RR, 1999.
29. M. Nilsson. Regular Model Checking. PhD thesis, Department of Information Technology, Uppsala University, 2000.
30. A. Nylén. Compact Constraints for Verification of Well Quasi-Ordered Programs. PhD thesis. Uppsala University, 2003.
31. M. Rusinowitch and L. Vigneron. Automated Deduction with Associative and Commutative Operators. *Applicable Algebra in Engineering, Communication and Computing*, 6:23–56, 1995.
32. A. P. Sistla and V. Gyuris. Parametrized Verification of Linear Networks using Automata as Invariants. *Formal Aspects of Computing*, 11(4):402–425, 1999.

A Preliminaries

A.1 Petri Nets

A Petri Nets is a tuple $\langle P, T, I, O\rangle$ where $P = \{p_1, \ldots, p_n\}$ is the finite set of places, $T = \{t_1, \ldots, t_m\}$ is the finite set of transitions, $I : P \times T \to \{0, 1\}$ represents the set of edges from places to transitions, and $O : P \times T \to \{0, 1\}$ represents the set of edges from transitions to places. Tokens are associated to places and are redistributed in the net after a transition is fired. A marking is a vector $\mathbf{m} = \langle m_1, \ldots, m_n\rangle$ of dimension n that counts how many tokens are in every place at a given time instant. A firing sequence is a sequence of markings $\mathbf{m_0 m_1} \ldots \mathbf{m_k}$ obtained by firing transitions. A transition t is fireable if there exists at least one token in every place p such that $I(p, t) = 1$. Given a marking $\mathbf{m}$, if t_j is fireable then the new marking $\mathbf{m}'$ generated by the firing of t_j is such that $m'_i = m_i + O(p_i, t_j) - I(p_i, t_j)$ for $i : 1, \ldots, n$. Since reachability is decidable, Petri Nets are not Turing equivalent.

A.2 Two Counters Machine

A Two Counters Machine configurations is a tuple $\langle Loc = p, c_1 = n_1, c_2 = n_2\rangle$ where p is control location taken from a finite set Q, and n_1 and n_2 are natural numbers that represent the values of the counters c_1 and c_2. Each counter can be incremented by one, decremented by one (when it is greater than zero), or tested for zero. Transitions combine operations on individual counters with changes of control locations.

Specifically, the instructions for counter c_i are as follows

Zero: If Loc$=p$ and $c_i = 0$ then $Loc := p'$;
Inc: If Loc$=p$ then $c_i := c_i + 1$; $Loc := p'$;
Dec: If Loc$=p$ and $c_i > 0$ then $c_i := c_i - 1$; $Loc := p'$;

A Two Counter Machines consists then of a list of instructions and of the initial state $\langle Loc = p_0, c_1 = 0, c_2 = 0\rangle$. The operational semantics is defined according to the intuitive semantics of the instructions (the control flow is determined by the value of Loc). Problems like reachability and control state reachability are undecidable for this computational model.

B The Theory of Well and Better Quasi-Orderings

In the following we summarize some basic definitions and results on the theory of *well quasi-orderings* taken from [1, 22, 3]. Following the notation used in the literature the relation $M \sqsubseteq N$ must be read as M is more general than N (i.e. N entails M). A *quasi-order* (preorder) $\sqsubseteq$ on a set A is a binary relation over A which is reflexive and transitive. In the following it will be denoted $(A, \sqsubseteq)$.

A quasi order $(A, \sqsubseteq)$ is a *well quasi ordering* (wqo) if each infinite sequence $a_0\ a_1\ a_2\ \ldots$ of elements in A is *good*. A sequence is $a_0\ a_1\ a_2\ \ldots$ is good if there exist indexes $i < j$ such that $a_i \sqsubseteq a_j$. (in our paper we use $Entails(a_j, a_i)$).

Wqo's are closed under several operations (see Prop. 4 below) like bags with bag embedding, and strings with string embedding. However, hey are not closed under the pointwise ordering of finite sets, a natural operation to extend an entailment to a set. Rado has given a counterexample to this closure property (see [3]).

The notion of *better quasi ordering* (bqo) is stronger than wqo and it has been studied to isolate the subset of wqo's that do not suffer from Rado's counterexample. To obtain this property, instead of reasoning over sequences, i.e., mappings $f : \mathbb{Z} \to A$ in which indexes are ordered with respect to $<$, we reason over an abstract structure $\langle \beta, \lhd \rangle$ called barrier equipped with the generalization $\lhd$ of $<$.

A quasi-order $(A, \sqsubseteq)$ is a *better quasi ordering* if for any barrier $\langle \beta, \lhd \rangle$ and any function $f : \beta \to A$, f is good, i.e., $\exists s \lhd s'.\ f(s) \preccurlyeq f(s')$.

The definition of barrier and of the order $\lhd$ are fairly complicate to explain. The interest reader can found them in [28].

For our purposes, it is enough to recall the closure properties of a bqo(wqo).

In what follows, let $\widehat{n}$ denote the set $\{1, \dots, n\}$. A string or a multiset over A can be represented as a mapping $w : \widehat{|w|} \to A$, where $|w|$ is the length of the string w or the cardinality of the multiset w. We have the following results, according to which a hierarchy of wqos and bqos can be built starting from known ones.

Proposition 4.

1. *Each bqo is a wqo;*
2. *If A is a finite set, then $(A, =)$ is a bqo.*
3. *Let $(A, \preccurlyeq)$ be a bqo, and let A^B denote the set of finite bags (i.e. multisets) over A. Then, $(A^B, \preccurlyeq^B)$ is a bqo, where $\preccurlyeq^B$ is the quasi-order on A^B defined as follows: $w \preccurlyeq^B w'$ iff there exists an injection $h : \widehat{|w|} \to \widehat{|w'|}$ such that $w(j) \preccurlyeq w'(h(j))$ for $1 \leq j \leq |w|$. This property also holds for a wqo.*
4. *Let $(A, \preccurlyeq)$ be a bqo, and let A^* denote the set of finite strings over A. Then, $(A^*, \preccurlyeq^*)$ is a bqo, where $\preccurlyeq^*$ is the quasi-order on A^* defined in the following way: $w \preccurlyeq^* w'$ iff there exists a strictly monotone (meaning that $j_1 < j_2$ iff $h(j_1) < h(j_2)$) injection $h : \widehat{|w|} \to \widehat{|w'|}$ such that $w(j) \preccurlyeq w'(h(j))$ for $1 \leq j \leq |w|$. This property also holds for a wqo, too.*
5. *Let $(A, \preccurlyeq)$ be a bqo. Then, $(\mathcal{P}(A), \preccurlyeq^S)$ is a bqo, where $\mathcal{P}(A)$ is the (finite) power set of A and $\preccurlyeq^S$ is the* pointwise *ordering defined as follows: given $X, Y \subseteq A$, $X \preccurlyeq^S Y$ iff for every $y \in Y$ there exists $x \in X$ s.t. $x \preccurlyeq y$. This property* **does not** *hold for a wqo.*

Examples of well quasi orderings that are also better quasi orderings are

- Multisets inclusion(embedding) (for multisets built over a finite alphabet);
- String embedding (for strings built over a finite alphabet).