Beyond Parameterized Verification

Marco Bozzano and Giorgio Delzanno

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Via Dodecaneso 35, 16146 Genova, Italy
{bozzano,giorgio}@disi.unige.it

Abstract. We present a sound and fully automated method for the verification of safety properties of parameterized systems with unbounded local data variables, a new class of infinite-state systems parametric in several dimensions. The method builds upon a specification and an assertional language based on the combination of *multiset rewriting* and *constraints*. We introduce new classes of parameterized systems for which verification of safety properties is decidable, and we introduce abstractions, defined at the level of constraints, to handle examples outside these classes. As case-study, we apply the method to verify fully automatically mutual exclusion properties for formulations of the *ticket mutual exclusion algorithm* parametric in the number of clients, servers, and in which both clients and servers have unbounded local data.

1 Introduction

In recent years several attempts have been made in order to develop tools for the automated verification of *infinite state systems*. Interesting results have been specifically obtained for the class of *parameterized systems*. A typical parameterized system consists of a collection of an *arbitrary* but *finite* number of *finite-state* components interacting via synchronous or asynchronous communication [5, 7, 20]. In many practical cases verification problems for this kind of systems can be reduced to problems related to Petri Nets by applying a *counting* abstraction that simply forgets local data while keeping track of the number of processes in a given state. Reachability procedures can then be used to verify the original property on the resulting Petri Net like model [10, 14, 15, 19]. New results have also been obtained for the verification of another class of infinite-state systems, i.e., concurrent systems with a fixed number of components but *unbounded data*. As an example, in [8, 12, 18], *constraints* are used to symbolically represent and manipulate infinite collections of states for these systems.

In this paper we address the definition of techniques for the automatic verification of systems and protocols parametric in *several dimensions*. As an example, we mention *mutual exclusion protocols* for *multi-client systems* that make use of global and local variables, like, e.g., the *ticket* and *bakery* protocols [8]. In these case-studies the *counting abstraction* turns out to be too rough to prove correctness. Inspired by the seminal paper [2] of Abdulla and Jonsson, in [11], we introduced a specification language and a corresponding assertional language for systems parametric in several dimensions. The specification language is based on *multiset rewriting over first order formulas* as proposed by Cervesato et al. in [9] enriched, however, with *constraints*. This way, we keep separate the structure of processes from the relations over their local data. The assertional language combines symbolic reasoning (unification and subsumption of multisets of first order atomic formulas) and constraint programming (satisfiability, entailment, and variable elimination). Symbolic operations like predecessor operators and comparison tests exploit the operations of the underlying *constraint system*. In this paper we extend the approach presented in [11] as follows.

We have isolated classes of multiset rewriting rules with constraints for which the verification of *safety properties* of practical interest is decidable. We consider a subclass of linear integer constraints, called NC, which allows us to handle an infinite collection of discrete values (e.g. integers) and to compare them using relations like $>$, $\geq$, and $=$. This domain can be used to represent local variables, process identifiers, priorities, and so on. The safety properties we can handle are such that the corresponding *unsafe states* consists of *upward closed sets* of configurations. Our algorithm follows the general approach of backward reachability proposed in [1] and exploits the theory of well and better quasi orderings for proving termination [1, 4, 17]. To attack verification problems for systems defined over *linear integer constraints* that lay outside the class for which termination is guaranteed, we use an *automated abstraction* that maps linear integer constraints into NC-constraints. This abstraction always returns a *conservative approximation* of the original property. Furthermore, it can be viewed as the counterpart of the *counting abstraction* used in [10] for systems properties that are data-sensitive. This way, we obtain a *fully-automatic* and *sound* algorithm for checking *safety properties* for a wide class of systems parametric in several dimensions.

We have implemented our automated verification method and applied it to analyze *mutual exclusion* for different formulations of the *ticket protocol*. As shown in [8], this protocol has an infinite-state space even for system configurations with only 2 processes. In this paper we extend the results of [8], where safety properties were proved for this special case, as follows. We have considered both a *multi-client*, *single-server* formulation, i.e., with an arbitrary but finite number of *dynamically* generated clients but a single shared resource, and a *multi-client*, *multi-server* system in which both clients and servers are created *dynamically*. Both examples have been modeled using *multiset rewriting rules* with *difference constraints*, that support arithmetic operations like *increment and decrement* of data variables. Our models are faithful to the original formulation of the algorithm, in that we do not abstract away global and local integer variables attached to individual clients, that in fact can still grow unboundedly. Using our symbolic backward reachability procedure combined with the dynamic use of abstractions, we have automatically verified that both models are safe for *any number of clients and servers* and for *any values of local and global variables*. To our knowledge both the techniques and the practical results are original.

THE SYSTEM WITH n PROCESSES

```
Program
  global var $s, t : integer$;
  begin
    $t := 0$;
    $s := 0$;
    $P_1 \mid \ldots \mid P_n$;
  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. 1. The Ticket Protocol: n is a parameter of the protocol.

Plan of the Paper In Section 2, we present the case-study. In Section 3, we introduce our specification language. In Section 4, we introduce the assertional language used to reason about safety properties. In Section 5, we describe the verification algorithm. In Section 6, we discuss decidability issues. In Section 7, we discuss experimental results. In Section 8 and 9, we discuss related works and address future works.
We leave the proofs of all results for a long version of the paper.

2 The Case-study: the Ticket Protocol

The ticket protocol is a *mutual exclusion* 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. 1 (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 of n processes. As remarked in [8], 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. 1 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.

3 MSR($\mathcal{C}$): Multiset Rewriting with Constraints

The ticket protocol presented in the previous section is a practical example of protocol parametric in *several dimensions*: the *number* of processes and the *value* of their local variables (denoted by a in Fig. 1). In this paper we will adopt the specification language proposed in [11] that combines aspects peculiar of High Level and Colored Petri Nets and of Constraint Programming. The framework called MSR is based on *multiset rewriting systems* defined over first-order atomic formulas and it has been introduced by Cervesato et al. [9] for the formal specification of *cryptographic protocols*. In [11], the basic formalism (without existential quantification) has been extended to allow for the specification of relations over data variable using *constraints*, i.e., a logic language interpreted over a fixed domain. Multiset rewriting rules allow one to *locally* model *rendez-vous* and *internal* actions of processes, and constraints to symbolically represent the *relation between the data of different processes*, thus achieving a clear separation between process structure and data paths. This formalism can be viewed as a first-order extension of Petri Nets in which tokens carry along structured data. In this section we will review the definitions of [11] and call the resulting formalism MSR($\mathcal{C}$). Lets us start from the formal definition of constraint system.

Definition 1 (Constraint System). 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 with *equality* and closed with respect to $\exists$ and $\wedge$; we call an open formula $\varphi \in \mathcal{L}$ with free variables in $\mathcal{V}$ a *constraint*; $\mathcal{D}$ is the *domain* of interpretation of the variables in $\mathcal{V}$; $Sol(\varphi)$ is the set of solutions (mappings $\mathcal{V} \rightsquigarrow \mathcal{D}$) for φ; $\sqsubseteq^c$ is a relation such that $\varphi \sqsubseteq^c \psi$ implies $Sol(\psi) \subseteq Sol(\varphi)$.

We say that a *constraint* $\varphi \in \mathcal{L}$ is *satisfiable* whenever $Sol(\varphi) \neq \emptyset$. An example of constraint systems is given below.

Definition 2 (DC-constraints). We call *difference constraints* the subclass of *linear arithmetic constraints* having the form

$$\psi \ ::= \ \psi \ \wedge \ \psi \mid x = y + c \mid x > y + c \mid x \geq y + c \mid true, \quad c \in \mathbb{Z}.$$

Given $\mathcal{D} = \mathbb{Z}$, Sol maps constraints into sets of variable evaluations from $\mathcal{V}$ to $\mathbb{Z}$; by definition *true* is always satisfiable.

For instance, let φ be $x \geq y \wedge x \geq z$, then $\sigma = \langle x \mapsto 2, y \mapsto 1, z \mapsto 0, \ldots \rangle \in Sol(\varphi)$. Furthermore, φ is satisfiable and entails $x \geq y$, and $\exists y.\varphi$ is equivalent to the constraint $x \geq z$.

Let $\mathcal{C} = \langle \mathcal{V}, \mathcal{L}, \mathcal{D}, Sol, \sqsubseteq^c \rangle$ be a constraint system, and $\mathcal{P}$ be a set of predicate symbols. An *atomic formula* $p(x_1, \ldots, x_n)$ is such that $p \in \mathcal{P}$, and $x_1, \ldots, x_n$ are *distinct* variables in $\mathcal{V}$. A *multiset* of atomic formulas is indicated as $A_1 \mid \ldots \mid A_k$, where A_i and A_j have distinct variables, and $\mid$ is the multiset constructor. The *empty* multiset is represented as ϵ. In the rest of the paper will use $\mathcal{M}$, $\mathcal{N}$, ... to denote *multisets* of atomic formulas.

Definition 3 (MSR($\mathcal{C}$) Rules). Let $\mathcal{C} = \langle \mathcal{V}, \mathcal{L}, \mathcal{D}, Sol, \sqsubseteq^c \rangle$ be a constraint system. An MSR($\mathcal{C}$) rule 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 built on predicates in $\mathcal{P}$, and $\varphi \in \mathcal{L}$.

Note that $\mathcal{M} \rightarrow \epsilon : \varphi$ and $\epsilon \rightarrow \mathcal{M} : \varphi$ are possible MSR($\mathcal{C}$) rules. An example of MSR(DC) rule R is $p(x, y) \mid r(u, v) \ \longrightarrow \ t(w) \ : x \geq y, w = v$. Let us call *ground* an atomic formula having the form $p(d_1, \ldots, d_n)$ where $d_i \in \mathcal{D}$ for $i : 1, \ldots, n$. A *configuration* is a multiset of *ground atomic formulas*. The ground instances of a multiset rewriting rule are defined as follows: $Inst(\mathcal{M} \longrightarrow \mathcal{M}' : \varphi) = \{\sigma(\mathcal{M}) \longrightarrow \sigma(\mathcal{M}') \mid \sigma \in Sol(\varphi)\}$, where $\sigma(\mathcal{M})$ denotes the straightforward extension of the mapping σ from variables to multisets. In the previous example we have that $p(1,0) \mid r(0,5) \longrightarrow t(5) \in Inst(R)$.

We are now in the position to define formally an MSR($\mathcal{C}$) specification. In the following we will use $\oplus$ and $\ominus$ to denote *multiset union* and *multiset difference*.

Definition 4 (MSR($\mathcal{C}$) Specification). An MSR($\mathcal{C}$) specification $\mathcal{S}$ is a tuple $\langle \mathcal{P}, \mathcal{C}, \mathcal{I}, \mathcal{R} \rangle$, where $\mathcal{P}$ is a 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 $\mathcal{P}$.

The operational semantics of a specification $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}, \mathcal{I}, \mathcal{R} \rangle$ is defined as follows.

Definition 5 (One-step Rewriting). Given two configurations $\mathcal{M}_1$ and $\mathcal{M}_2$, $\mathcal{M}_1 \Rightarrow \mathcal{M}_2$ if and only if there exists a multiset of ground atomic formulas $\mathcal{Q}$ s.t. $\mathcal{M}_1 = \mathcal{N}_1 \oplus \mathcal{Q}$, $\mathcal{M}_2 = \mathcal{N}_2 \oplus \mathcal{Q}$, and $\mathcal{N}_1 \longrightarrow \mathcal{N}_2 \in Inst(\mathcal{R})$.

Definition 6 (Images and Reachable States). Given a set of configurations S, the *successor* operator is defined as $Post(S) = \{\mathcal{M}' | \mathcal{M} \Rightarrow \mathcal{M}', \mathcal{M} \in S\}$, the *predecessor* operator as $Pre(S) = \{\mathcal{M} | \mathcal{M} \Rightarrow \mathcal{M}', \mathcal{M}' \in S\}$, and the *reachability set* as $Post^*(\mathcal{I})$.

3.1 The MSR(DC) Encoding of the Ticket Protocol

In this section we exemplify the notions introduced in Section 3. Multiset rewriting allows us to give an accurate and flexible encoding of the ticket protocol. We will first consider a *multi-client, single resource* system.

One Server, Many Clients Let us first consider a *single* shared resource controlled via the counters t and s as described in Section 2. 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. 2. 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 behaviour of an individual client is described via the third block of rules of Fig. 2, in which the relation between the local variable and the global counters are represented via DC-constraints. Finally, we allow thinking processes to terminate their execution as specified via the last rule of Fig. 2. 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 out any restrictions on their values. As a consequence, there are runs of our model in which s and t grow unboundedly as in the original protocol. A sample run of a system with 2 clients (as in [8]) is shown in Fig. 3.

Initial States

$$init \longrightarrow count(t) \mid turn(s) \;:\; t = s$$

Dynamic Generation

$$\epsilon \longrightarrow think \;:\; true$$

Individual Behaviour

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

Termination

$$think \longrightarrow \epsilon \;:\; true$$

Fig. 2. 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. 3. Example of run.

Many Servers, Many Clients Let us consider now an *open* system with an *arbitrary* but *finite* number of *shared resources*, each one controlled by two local counters s and t. We specify this scenario by associating a *unique identifier* to each resource and to use it to stamp the corresponding pair of counters. Furthermore, we exploit non-determinism in order to simulate the capability of each client to choose which resource to use. The resulting specification is shown in Fig 4. We have considered an *open* system in which new clients can be generated *dynamically* via a *demon* process. The process $demon(n)$ maintains a local counter n used to generate a new identifier, say id, and to associate it to a newly created resource represented via the pair $count(id, t)$ and $turn(id, s)$. A thinking process non-deterministically chooses which resource to wait for by synchronizing with one of the counters in the system (the first rule of the third block in Fig. 4). After this choice, the algorithm behaves as usual w.r.t. to the chosen resource id. The termination rules can be specified as natural extensions of the single-server case. Note that in this specification the sources of infiniteness are the number of clients, the number of shared resources, the values of resource identifiers, and the values of tickets. An example of run is shown in Fig. 5.

4 Specification of Properties via an Assertional Language

Let us focus for a moment on the single-server ticket protocol. It should ensure mutual exclusion for any number of clients, and for any value of the global and local variables. In our specification mutual exclusion holds if every *reachable configuration* $\mathcal{M} \in Post^*(init)$ contains *at most* one occurrence of the predicate

Initial States

$$init \quad \longrightarrow \quad demon(n) \; : \; true$$

Dynamic Process and Server Generation

$$\epsilon \longrightarrow think \; : \; true$$

$$demon(n) \rightarrow demon(n') \mid count(id,t) \mid turn(id',s) \; : \; n' = n+1 \;\wedge\; t = s \;\wedge\; id = n \;\wedge\; id' = id$$

Individual Behaviour

$$think \mid count(id,t) \quad \longrightarrow \quad think(r) \mid count(id',t') : \; r = id \;\wedge\; id' = id \;\wedge\; t' = t$$

$$think(r) \mid count(id,t) \quad \longrightarrow \quad wait(r',a) \mid count(id',t') \; : \; r = id \;\wedge\; a = t \;\wedge\; t' = t+1 \;\wedge\; r' = r \wedge\; id' = id$$

$$wait(r,a) \mid turn(id,s) \quad \longrightarrow \quad use(r',a') \mid turn(id',s') \; : \; r = id \;\wedge\; a = s \;\wedge\; a' = a \;\wedge\; s' = s \wedge\; r' = r \;\wedge\; id' = id$$

$$use(r,a) \mid turn(id,s) \quad \longrightarrow \quad think \mid turn(id',s') \; : \; r = id \;\wedge\; s' = s+1 \;\wedge\; id' = id$$

Termination

$$think(r) \quad \longrightarrow \quad \epsilon \; : \; true$$
$$think \quad \longrightarrow \quad \epsilon \; : \; true$$

Fig. 4. Ticket protocol for *multi-server*, *multi-client* systems.

$$\begin{gathered}
init \Rightarrow demon(3) \;\Rightarrow count(3,0) \mid turn(3,0) \mid demon(4) \Rightarrow \dots \\
\dots \Rightarrow count(3,0) \mid turn(3,0) \mid think \mid think \mid demon(4) \\
\Rightarrow count(3,0) \mid turn(3,0) \mid count(4,8) \mid turn(4,8) \mid think \mid think \mid demon(5) \Rightarrow \dots \\
\Rightarrow count(3,0) \mid turn(3,0) \mid count(4,8) \mid turn(4,8) \mid think(4) \mid think(3) \mid demon(5) \\
\Rightarrow count(3,0) \mid turn(3,0) \mid count(4,9) \mid turn(4,8) \mid wait(4,8) \mid think(3) \mid demon(5).
\end{gathered}$$

Fig. 5. Example of run.

use. The state-space we have to generate to check the previous condition is infinite both in the size of the generated multisets and in the number of multisets of the same size (the latter due to the unboundedness of ticket variables). The only way to algorithmically check this property is using an adequate *assertional language* to finitely represent infinite collections of configurations. In [11], we proposed to use a special assertional language based on *constrained multisets*. To explain this idea, let us first note that an alternative way of formulating the validity of mutual exclusion is as follows: $init \not\in Pre^*(U)$, where U is the infinite collection of *unsafe* states. U consists of all the configurations in which there are *at least* two occurrences of the predicate *use*. This set can be finitely represented using the following idea. Let us introduce the following ordering between configurations:

$$\mathcal{M} \preccurlyeq \mathcal{N} \text{ if and only if } Occ_A(\mathcal{M}) \leq Occ_A(\mathcal{N}) \text{ for any } \textit{ground} \text{ atom } A,$$

where $Occ_A(\mathcal{M})$ is the number of occurrences of A in $\mathcal{M}$. A set of configurations S *generates* its *upward closure* $Up(S)$ defined as $Up(S) = \{\mathcal{N} \mid \mathcal{M} \preccurlyeq \mathcal{N},\ \mathcal{M} \in S\}$. A set S is *upward-closed* whenever $Up(S) = S$. Let us go back to our case-study. It is easy to verify that the set of unsafe states U of the ticket protocol is indeed *upward closed*. Furthermore, U can be represented as the upward closure of the set S of configurations having the form $use(c_1) \mid use(c_2)$ where c_1 and c_2 are arbitrary integer values. Though S is still an infinite set, we can finitely represent it by re-introducing *constraints* as annotations of a multiset of atomic formulas. Specifically, if we define $M = use(x) \mid use(y) \; : \; true$, S corresponds to the set of instances of M w.r.t. the solutions of the constraint *true* (all possible values for x and y). Similarly, the unsafe states for the multi-server ticket protocol can be expressed via the *constrained configuration* $use(id, x) \mid use(id', y) \; : \; id = id'$ meaning that at least two clients are in the critical section associated to the same resource with identifier id. This observation is at the basis of our verification method. Fixed $\mathcal{S} = \langle \mathcal{P}, \mathcal{C}, \mathcal{I}, \mathcal{R} \rangle$ and $\mathcal{C} = \langle \mathcal{V}, \mathcal{L}, \mathcal{D}, Sol, \sqsubseteq^c \rangle$, we generalize the previous ideas as follows.

Definition 7 (Constrained Configuration). A *constrained configuration* is a multiset of atomic formulas with distinct variables, annotated with a *satisfiable* constraint, i.e., $p_1(x_{11}, \ldots, x_{1k_1}) \mid \ldots \mid p_n(x_{n1}, \ldots, x_{nk_n}) : \varphi$ where $p_1, \ldots, p_n \in \mathcal{P}$, $x_{i1}, \ldots, x_{ik_i} \in \mathcal{V}$ for any $i : 1, \ldots n$ and constraint $\varphi \in \mathcal{L}$.

Given a constrained configuration $\mathcal{M} : \varphi$, the *set* of its *ground instances* is defined as $Inst(\mathcal{M} : \varphi) = \{\sigma(\mathcal{M}) \mid \sigma \in Sol(\varphi)\}$. This definition can be extended to sets of constrained configurations with *disjoint variables* (indicated as $\mathbf{S}, \mathbf{S}', \ldots$) in the natural way. However, instead of taking the set of instances as 'flat' denotation of a set of constrained configuration $\mathbf{S}$, we will choose the following rich denotation.

Definition 8 (Rich Denotation). The denotation of a set $\mathbf{S}$ of constrained configurations is the upward closed set of its ground instances $[\![\mathbf{S}]\!] = Up(Inst(\mathbf{S}))$.

We conclude this section by introducing a comparison test between (sets of) constrained configurations whose definition relies on the operations of the underlying constraints system $\mathcal{C}$. Given the atoms $A = p(x_1, \ldots, x_k)$ and $B = q(y_1, \ldots, y_l)$, we will use $A = B$ as an abbreviation for the constraint $x_1 = y_1 \wedge \ldots \wedge x_k = y_k$, provided $p = q$ and $k = l$.

Definition 9. The entailment relation $\sqsubseteq^m$ between constrained configurations is defined as follows: $(A_1 \mid \ldots \mid A_n : \varphi) \; \sqsubseteq^m \; (B_1 \mid \ldots \mid B_k : \psi)$ provided $n \leq k$, and there exist $j_1, \ldots, j_n$ *distinct* indices in $\{1, \ldots, k\}$, such that $\gamma \equiv \exists x_1 . \ldots \exists x_r . \; \psi \wedge A_1 = B_{j_1} \wedge \ldots \wedge A_n = B_{j_n}$ and γ is satisfiable, where $x_1, \ldots, x_r$ are the variables in $B_1, \ldots, B_k$, and, finally, $\varphi \; \sqsubseteq^c \; \gamma$.

Definition 10. The entailment relation $\sqsubseteq^p$ (p stands for *pointwise* extension of the $\sqsubseteq^m$ relation) between sets of constrained configurations is defined as follows: $\mathbf{S} \sqsubseteq^p \mathbf{S}'$ iff for every $M \in \mathbf{S}'$ there exists $N \in \mathbf{S}$ such that $N \sqsubseteq^m M$.

The entailment relation $\sqsubseteq^p$ provides *sufficient* conditions (that are fully parametric w.r.t. $\mathcal{C}$) for testing containment and equivalence of the denotations of sets of constrained configurations. Let $\mathbf{S}$, $\mathbf{S}'$ be two sets of constrained configurations.

Proposition 1. If $\mathbf{S} \sqsubseteq^p \mathbf{S}'$ then $[\![\mathbf{S}']\!] \subseteq [\![\mathbf{S}]\!]$.

```
Procedure $Pre^*(\mathbf{U}$ : Set of DC-constrained config., $Use_\alpha$ : Bool, $UseInv$ : Bool)
  begin
    $\mathbf{S} := \mathbf{U}$;
    $\mathbf{R} := \emptyset$;
    while $\mathbf{S} \neq \emptyset$ do
      remove $M$ from $\mathbf{S}$;
      if $UseInv$ and $Inv(M) = false$ then $skip$
      else
          if $Use_\alpha$ then $M' = \alpha(M)$
          else $M' = M$;
          if $\nexists\ N \in \mathbf{R}$ s.t. $N \sqsubseteq^m M'$ then
                  $\mathbf{R} := \mathbf{R} \cup \{M'\}$;
                  $\mathbf{S} := \mathbf{S} \cup \mathbf{Pre}(\{M'\})$
  end
```

Fig. 6. Symbolic backward reachability ($Inv(\mathcal{M} : \alpha) = true$ iff the statically computed place invariants hold in $\mathcal{M}$).

5 A Sound Verification Procedure

In order to lift to the symbolic level an ideal verification algorithm based on the computation of Pre^* (*backward reachability*), we need a *symbolic Pre* operator working on *sets* of constrained configurations. We first introduce the notion of *unification* between constrained configurations (with disjoint variables) as follows: $(A_1 \mid \ldots \mid A_n : \varphi) =_\theta (B_1 \mid \ldots \mid B_m : \psi)$ provided $m = n$ and the constraint $\theta = \varphi \wedge \psi \wedge \bigwedge_{i=1}^n A_i = B_{j_i}$ is *satisfiable*, $j_1, \ldots, j_n$ being a permutation of $1, \ldots, n$. The symbolic operator $\mathbf{Pre}$ is defined as follows:

Definition 11 (Symbolic Predecessor Operator). Given a set $\mathbf{S}$ of constrained configurations, $(\mathcal{A} \oplus \mathcal{N} : \gamma) \in \mathbf{Pre}(\mathbf{S})$ if and only if there exist a renamed rule $\mathcal{A} \longrightarrow \mathcal{B} : \psi$ in $\mathcal{R}$, and a renamed constrained configuration $\mathcal{M} : \varphi$ in $\mathbf{S}$ such that $\mathcal{M}' \preccurlyeq \mathcal{M}$, $\mathcal{B}' \preccurlyeq \mathcal{B}$, $(\mathcal{M}' : \varphi) =_\theta (\mathcal{B}' : \psi)$, $\mathcal{N} = \mathcal{M} \ominus \mathcal{M}'$, and $\gamma \equiv \exists x_1 \ldots \exists x_k.\theta$ where $x_1, \ldots, x_k$ are the variables of θ not in $\mathcal{A} \oplus \mathcal{N}$.

The symbolic operator $\mathbf{Pre}$ returns a set of constrained configurations and it is correct and complete with respect to Pre, i.e., $[\![\mathbf{Pre}(\mathbf{S})]\!] = Pre([\![\mathbf{S}]\!])$ for any $\mathbf{S}$.

Based on this definition, we define a *symbolic backward reachability* procedure (see Fig. 6) we can use to check safety properties whose negation can be expressed via an upward closed set of configurations. The algorithm is not complete since it is possible to encode undecidable reachability problems (e.g. for two counter machines) as verification problems of generic MSR($\mathcal{C}$) specifications.

As shown in Fig. 6, the theory of structural invariants for Petri nets can be used to optimize the backward search computation. Namely, we can use the so-called *counting abstraction* [10] to transform any MSR($\mathcal{C}$) specification $\mathcal{S}$ into a Petri Net $N_{\mathcal{S}}$, which can then be used to automatically discover *invariants* that must hold for all reachable configurations of the original specification. For instance, in our case-study we can automatically infer that the number of tokens in place *turn* and *count* are always bounded by one. Since this analysis is conservative w.r.t. the abstraction, it follows that in all reachable configurations for the ticket specification, $Occ_{count(v)}(\mathcal{M}) \leq 1$ and $Occ_{turn(v)}(\mathcal{M}) \leq 1$ for any $v \in \mathbb{Z}$. A similar reasoning can be applied to the multi-server protocol concerning the counters associated to a given identifier (at most one copy of each counter) and the demon process (at most one copy). The invariants can be used during the fixpoint computation to *prune* the *search space*, by discharging every constrained multiset which violates the invariants, without any loss of precision.

6 Sufficient Conditions for Termination

We will now introduce a subclass of DC-constraints, called NC (*name constraints*), and corresponding MSRs for which computation of Pre^* (starting from an upward closed set of configurations) is effective.

Definition 12 (The Constraint System NC**).** The class NC consists of the constraints $\varphi \ ::= \ \varphi \wedge \varphi \mid x > y \mid x = y \mid x \geq y \mid true$, interpreted over the *integers* and ordered with respect to the entailment $\sqsubseteq^c$ of linear constraints.

Being a subclass of linear constraints and closed with respect to existential quantification, any constraint solver for linear constraints can still be used to handle NC-constraints. The class of *monadic* MSR over NC is defined as follows.

Definition 13 (The class MSR$_1$(NC)**).** An MSR$_1$(NC) specification $\mathcal{S} = \langle \mathcal{P}, \mathrm{NC}, \mathcal{I}, \mathcal{R} \rangle$ is such that all predicates in $\mathcal{P}$ have arity *less or equal than one*, i.e., atomic formulas have the form p or $p(x)$ for $p \in \mathcal{P}$ and some variable x.

Let $\mathcal{A}_1$ be the set of constrained configurations with the same restrictions of the class MSR$_1$(NC). As an example, the rule $a \mid p(x) \mid q(y) \rightarrow q(z) : x > y \wedge y = z$ is in MSR$_1$(NC), and $p(x) \mid q(y) : x > y$ is an element of $\mathcal{A}_1$. Then, the following properties hold.

Lemma 1. (i) The class $\mathcal{A}_1$ is closed under applications of **Pre** for an MSR$_1$(NC) specification; (ii) the *entailment relation* $\sqsubseteq^p$ (see Def. 10) between sets of constrained configurations in $\mathcal{A}_1$ is a *well quasi ordering.*

Point (ii) is based on the notion of well and better quasi orderings [1, 4] (the proof is available in the extended version of the paper). As a consequence, we obtain the following result.

Theorem 1. The backward reachability algorithm of Section 5 is guaranteed to terminate when taking as input an MSR$_1$(NC) specification and a set $\mathbf{U} \subseteq \mathcal{A}_1$.

The formulation of the ticket protocol for the multi-server system requires predicates with at least two arguments. Therefore, it seems natural to ask whether it is possible to extend Theorem 1 to the general case of arbitrary arity. However, the entailment relation $\sqsubseteq^m$ (see Def. 9) between NC-constrained multisets in which atoms have arbitrary arity is not a *well quasi ordering.* The counterexample to the well quasi ordering of $\sqsubseteq^m$ is as follows. Consider the sequence of constrained configurations $M_2, \ldots, M_i, \ldots$ such that M_i is defined as $p(x_1, x_2) \mid \ldots \mid p(x_{2*i-1}, x_{2*i}) \ : \ x_2 = x_3, \ldots, x_{2*i-2} = x_{2*i-1}, \ x_{2*i} = x_1$. Every constrained configuration in the sequence implicitly defines a *simple cyclic* relation with i edges. The well quasi ordering would imply for a subgraph of a simple cycle of order i to be isomorphic to a simple cycle of order $j < i$. The key point here is the possibility of using predicates in combination with NC-constraints to form cyclic relations. One possible way of avoiding potential circularities consists in restricting the form of constraints as follows.

Definition 14 (The Class $\mathbf{MSR}_n(\mathrm{NC}_n)$). It consists of predicates with at most arity n, and rules annotated with special NC-constraints having the following form: $\varphi \in \mathrm{NC}_n$ iff φ can be partitioned in the subconstraints $\varphi_1 \ldots \varphi_n$, where φ_i contains only variables that occur in position i (arguments are ordered from left-to-right).

We will call $\mathcal{A}_n$ the class of NC_n-constrained configurations. As an example, the constrained configuration $p(\mathbf{y_1}) \mid p(\mathbf{x_1}, x_2) \mid q(\mathbf{w_1}, w_2) \ : \ \mathbf{y_1} > \mathbf{x_1} \wedge \mathbf{x_1} > \mathbf{w_1} \wedge x_2 > w_2$ is in $\mathcal{A}_2$, whereas $p(\mathbf{x_1}, x_2) \mid q(\mathbf{w_1}, w_2) \ : \ \mathbf{x_1} > w_2$ is not. Then, the following properties hold.

Lemma 2. (i) The class $\mathcal{A}_n$ is closed under applications of **Pre** for an $\mathrm{MSR}_n(\mathrm{NC}_n)$ specification; (ii) the *entailment relation* $\sqsubseteq^p$ (see Def. 10) between sets of constrained configurations in $\mathcal{A}_n$ is a *well quasi ordering.*

Point (ii) is based again on the notion of well and better quasi orderings [1, 4]. As a consequence, we obtain the following result.

Theorem 2. The backward reachability algorithm of Section 5 is guaranteed to terminate when taking as input an $\mathrm{MSR}_n(\mathrm{NC}_n)$ specification and a set $\mathbf{U} \subseteq \mathcal{A}_n$.

6.1 Automated Abstraction Procedures

By exploiting the property that NC-constraints are a subclass of DC-constraints, we can enforce *termination* via the following abstraction.

Let $\# \in \{>, =, \geq\}$. The abstraction α from DC- to NC-constraints is defined on a *satisfiable* DC-constraint as follows: $\alpha(\mathit{true}) = \mathit{true}$; $\alpha(\varphi_1 \wedge \varphi_2) = \alpha(\varphi_1) \wedge \alpha(\varphi_2)$; $\alpha(x \ \# \ y + c) = x \ \# \ y$ if $c = 0$; $\alpha(x \ \# \ y + c) = x > y$ if $c > 0$; $\alpha(x \ \# \ y{+}c) = y > x$ if $c < 0$ and $\#$ is $=$; $\alpha(\varphi) = \mathit{true}$ otherwise. Furthermore, we define $\alpha(\mathcal{M} : \varphi) = \mathcal{M} : \alpha(\varphi)$ and we extend this definition to sets of constrained configurations in the natural way.

Since $\mathit{Sol}(\varphi) \subseteq \mathit{Sol}(\alpha(\varphi))$ holds, it follows that $[\![\mathcal{M} : \varphi]\!] \subseteq [\![\alpha(\mathcal{M} : \varphi)]\!]$, and $[\![\mathbf{S}]\!] \subseteq [\![\alpha(\mathbf{S})]\!]$. If we define $\mathbf{Pre}_\alpha = \alpha \circ \mathbf{Pre}$, we have that $\mathbf{Pre}_\alpha$ gives us a conservative approximation of **Pre** when applied to an abstraction (via α) of a set of DC-constrained configurations, i.e. $[\![\mathbf{Pre}(\mathbf{S})]\!] \subseteq [\![\mathbf{Pre}_\alpha(\alpha(\mathbf{S}))]\!]$. As a consequence, we have the following property.

Proposition 2. *Let* $\mathbf{U}$ *be a set of* DC*-constrained configurations and* $\mathcal{I}$ *be the initial configurations, then* $\mathcal{I} \cap [\![\mathbf{Pre}^*_\alpha(\alpha(\mathbf{U}))]\!] = \emptyset$ *implies* $\mathcal{I} \cap [\![\mathbf{Pre}^*(\mathbf{U})]\!] = \emptyset$.

This observation leads us to a new backward reachability algorithm obtained by *interleaving* every application of **Pre** with the application of the abstraction α. Termination is guaranteed by Theorems 1 and 2.

7 Verification of the Parameterized Ticket Protocol

We have verified *mutual exclusion* for both models of the ticket protocol presented in Section 2. According to Section 4, the set of violations can be represented through the constrained configurations $use(x)|use(y) : true$ for the single-server formulation, and $use(id, x)|use(id', y) : id = id'$ for the multi-server one. Thanks to the results of Section 6 and using the abstraction α of Section 6.1, our procedure is guaranteed to terminate (*symbolic state explosion* permitting). In Fig. 7, we describe the experimental results obtained using both the concrete (based on **Pre**) and abstract (based on $\mathbf{Pre}_\alpha$) backward reachability, on the specifications given in Section 3.1. In Fig. 7, $\surd$ indicates that the abstraction α or the pruning technique based on static analysis has been applied after each application of the symbolic predecessor operator; $\uparrow$ indicates that the procedure was still computing after several hours. Furthermore, **Steps** denotes the number of iterations needed to reach a fixpoint (before stopping the program); **Size** the number of constrained configurations contained in the fixpoint (when the program was stopped); and **Time** the execution time (in seconds). The backward search engine with DC-solver described in Fig. 6 has been implemented in ML and tested on the interpreter for Standard ML of New Jersey, version 110.0.7. All experiments have been executed on a Pentium III 450 Mhz, Linux 2.2.13-0.9. As shown in Fig. 7, using the abstract (theoretically always terminating) backward reachability algorithm we managed to prove all safety properties we were interested in (we prune the search space using the invariants discussed in Section 5). Furthermore, we managed to prove mutual exclusion without pruning the search for the first model, whereas it was necessary to cut the search space using structural invariants in the second example in order to avoid the state explosion problem. Note that pruning techniques do not introduce any kind of approximations, in fact, when used without α the fixpoint computation does not terminate as when executed on the pure symbolic algorithm. In an additional series of experiments, we verified again both models adding the structural invariants to the unsafe states. This technique is perfectly sound and, basically, it has the same effect as dynamic pruning.

8 Related Works

This work is inspired to the approach of [2, 4]. In [2], Abdulla and Jonsson proposed an assertional language for Timed Petri Nets in which they use dedicated data structures to symbolically represent markings parametric in the number of tokens and in the *age* (a real number) associated to tokens. In [4], Abdulla and Nylén formulate a symbolic algorithm using *existential regions* to represent the state-space of Timed Petri Nets. Our approach generalizes the ideas of [2, 4]

Ticket Specification	**Seed**	α	**Prune**	**Steps**	**Size**	**Time**	**Verified?**
Multi-client, Single-server (Fig. 2, Section 3.1)	U_s			$\uparrow$	--	--	--
	U_s		$\surd$	$\uparrow$	--	--	--
	U_s	$\surd$		17	222	$150s$	yes
	U_s	$\surd$	$\surd$	10	32	$< 1s$	yes
	$U_s \oplus I_s$	$\surd$		10	34	$< 1s$	yes
Multi-client, Multi-server (Fig. 4, Section 3.1)	U_m			$\uparrow$	--	--	--
	U_m		$\surd$	$\uparrow$	--	--	--
	U_m	$\surd$		> 18	> 3500	$> 4h$	--
	U_m	$\surd$	$\surd$	19	141	$15s$	yes
	$U_m \oplus I_m$	$\surd$		19	147	$19s$	yes

Unsafe states

$U_s \equiv \{\ use(x) \mid use(y) : true\ \}$

$U_m \equiv \{\ use(id, x) \mid use(id', y) : id = id'\ \}$

Structural invariants

$I_s \equiv \{\ count(x) \mid count(y) : true, \quad turn(x) \mid turn(y) : true\ \}$

$I_m \equiv \{\ count(id, x) \mid count(id', y) : id = id', \ turn(id, x) \mid turn(id', y) : id = id',$
$\quad demon(id) \mid demon(id') : true\ \}$

Fig. 7. Analysis of the ticket protocol.

to problems and constraint systems that do not depend on the notion of *time*. Following [4], we use the technique of better quasi orderings to build new classes of well quasi ordered symbolic representations. In [3], the authors apply similar ideas to (unbounded) channel systems in which messages can vary over an infinite *name* domain and can be stored in a finite (and fixed a priori) number of data variables; however, individual processes have finite-state control structures.

For networks of *finite-state* processes, it is important to mention the automata theoretic approach to parameterized verification followed, e.g., in [7, 6, 20–23]. In this setting the set of possible *local states* of individual processes are abstracted into a *finite alphabet*. Sets of global states are represented then 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. [7, 20, 23]), or with automated abstractions techniques (see e.g. [6]). Differently from the automata theoretic approach, in our setting we handle parameterized systems in which individual components have local variables that range over *unbounded* values. As an example, in our model of the ticket algorithm local variables and tickets range over unbounded integers. Furthermore, note that the abstraction with NC-constraints as target does not make the resulting symbolic representation finite. Nevertheless, termination can be guaranteed by applying the theory of well quasi ordering. This way, we do not have to apply *manual abstractions* to describe individual processes. This is an important aspect to take into account when comparing our results for the single-server model with those obtained in [20, 22], in which an *idealized* version of the *ticket algorithm* has been verified using the regular model checking method (actually, a real comparison is difficult here because the verified model is not described in [20, 22]). The previous features also distinguish our approach from the *verification with invisible invariants* method of [5]. Invisible invariants have been applied to automatically verify a *restricted* version of the parameterized *bakery* algorithm in which a special *reducing process* is needed to force the value of the tickets to stay within a given range. Our ideas are related to previous works connecting Constraint Logic Programming and verification, see e.g. [12, 18]. 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 multiset* extends that of *constrained atom* of [12]. The *locality* of rules allows us to consider *rich* denotations (upward-closures) instead of *flat* ones (instances) like, e.g., in [12]. This way, we can lift the approach to the parameterized case. In [16] a combination of transformation of logic programs and of weak monadic second order logic has been applied to verify a parameterized formulation of the bakery algorithm. The proof however is done manually, furthermore, even if implemented, the method is not guaranteed to terminate. Finally, the use of *constraints*, *backward reachability*, *structural invariants* and *better quasi orderings* seem all ingredients that distinguish our *hybrid* method from classical approaches based on multiset and AC rewriting techniques (see e.g. [9, 24]).

9 Conclusions

In this paper we have presented a sound and fully automated method to attack verification of parameterized systems with unbounded local data. Sufficient conditions for termination are given for new classes of infinite-state systems. The method is powered by using static analysis techniques coming from the structural theory of Petri Nets and by automatic abstractions working on constraints. As a practical application, we have automatically verified (as far as we know for the first time) a very general formulation of the *ticket* mutual exclusion protocol in which we allow many clients, many servers, and unbounded local variables. 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 [8], and a *real arithmetic* one [12]. However, we are not aware of other methods that can *automatically* handle the *parameterized* models of Section 3.1 in their generality.

Acknowledgment We would like to thank Moshe Vardi, Parosh A. Abdulla, Bengt Jonsson, the people at VEPAS 2001, and James Larus and Microsoft Research for having supported our research in the last two years.

References

1. P. A. Abdulla, K. Cerāns, B. Jonsson, and Y.-K. Tsay. General Decidability Theorems for Infinite-State Systems. In *Proc. LICS'96*, pp. 313–321, 1996.

2. P. A. Abdulla and B. Jonsson. Verifying Networks of Timed Processes. In *Proc. TACAS'98*, *LNCS* 1384, pp. 298–312, 1998.
3. P. A. Abdulla and B. Jonsson. Channel Representations in Protocol Verification. In *Proc. CONCUR'2001*, *LNCS* 2154, p. 1–15, 2001.
4. P. A. Abdulla and A. Nylén. Better is Better than Well: On Efficient Verification of Infinite-State Systems. In *Proc. LICS'00*, pp. 132–140, 2000.
5. T. Arons, A. Pnueli, S. Ruah, Y. Xu, and L. D. Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *Proc. CAV'01*, *LNCS* 2102, pp. 221–234, 2001.
6. K. Baukus, S. Bensalem, Y. Lakhnech, and K. Stahl. Abstracting WS1S Systems to Verify Parameterized Networks. In *Proc. TACAS'00*, *LNCS* 1785, pp. 188–203, 2000.
7. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular Model Checking. In *Proc. CAV'00*, *LNCS* 1855, pp. 403–418, 2000.
8. T. Bultan, R. Gerber, and W. Pugh. Symbolic Model Checking of Infinite State Systems Using Presburger Arithmetics. In *Proc. CAV'97*, *LNCS* 1254, pp. 400–411, 1997.
9. I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A Meta-notation for Protocol Analysis. In *Proc. CSFW'99*, p. 55–69, 1999.
10. G. Delzanno. Automatic Verification of Parameterized Cache Coherence Protocols. In *Proc. CAV'00*, *LNCS* 1855, pp. 53–68, 2000.
11. G. Delzanno. An Assertional Language for Systems Parametric in Several Dimensions. In *Proc. VEPAS '01*, *ENTCS* volume 50, issue 4, 2001.
12. G. Delzanno and A. Podelski. Model checking in CLP. In *Proc. TACAS'99*, *LNCS* 1579, pp. 223–239, 1999.
13. G. Delzanno, J.-F. Raskin, and L. Van Begin. Attacking Symbolic State Explosion. In *Proc. CAV'01*, *LNCS* 2102, pp. 298–310, 2001.
14. E.A. Emerson and K.S. Namjoshi. On Model Checking for Non-Deterministic Infinite-State Systems. In *Proc. LICS'98*, pp. 70–80, 1998.
15. J. Esparza, A. Finkel, and R. Mayr. On the Verification of Broadcast Protocols. In *Proc. LICS'99*, pp. 352–359, 1999.
16. F. Fioravanti, A. Pettorossi, M. Proietti. Verifcation of Sets of Infinite State Systems Using Program Transformation. In *Proc. LOPSTR'01*, pp. 55-66, 2001.
17. A. Finkel and P. Schnoebelen. Well-Structured Transition Systems Everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
18. L. Fribourg. Constraint Logic Programming Applied to Model Checking. In *Proc. LOPSTR'99*, *LNCS* 1817, pp. 30–41, 1999.
19. S. M. German and A. P. Sistla. Reasoning about Systems with Many Processes. *Journal of the ACM*, 39(3):675–735, 1992.
20. B. Jonsson and M. Nilsson. Transitive Closures of Regular Relations for Verifying Infinite-State Systems. In *Proc. TACAS'00*, *LNCS* 1785, pp. 220–234, 2000.
21. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. In *Proc. CAV'97*, *LNCS* 1254, pp. 424–435, 1997.
22. M. Nilsson. *Regular Model Checking.* PhD thesis, Department of Information Technology, Uppsala University, 2000.
23. A. Pnueli and E. Shahar. Liveness and Acceleration in Parameterized Verification. In *Proc. CAV'00*, *LNCS* 1855, pp. 328–343, 2000.
24. M. Rusinowitch and L. Vigneron. Automated Deduction with Associative and Commutative Operators. *Applicable Algebra in Engineering, Communication and Computing*, 6:23–56, 1995.