Verification of Sets of Infinite State Processes Using Program Transformation

Fabio Fioravanti[1], Alberto Pettorossi[2], Maurizio Proietti[1]

(1) IASI-CNR, Viale Manzoni 30, I-00185 Roma, Italy
(2) DISP, University of Roma Tor Vergata, I-00133 Roma, Italy
{fioravanti,adp,proietti}@iasi.rm.cnr.it

Abstract We present a method for the verification of safety properties of concurrent systems which consist of finite sets of infinite state processes. Systems and properties are specified by using constraint logic programs, and the inference engine for verifying properties is provided by a technique based on unfold/fold program transformations. We deal with properties of finite sets of processes of arbitrary cardinality, and in order to do so, we consider constraint logic programs where the constraint theory is the Weak Monadic Second Order Theory of k Successors. Our verification method consists in transforming the programs that specify the properties of interest into equivalent programs where the truth of these properties can be checked by simple inspection in constant time. We present a strategy for guiding the application of the unfold/fold rules and realizing the transformations in a semiautomatic way.

1 Introduction

Model checking is a well established technique for the verification of temporal properties of concurrent systems consisting of a *fixed* number of *finite state* processes [6]. Recently, there have been various proposals to extend model checking for verifying properties of systems consisting of an *arbitrary* number of *infinite state* processes (see, for instance, [18,21,25]). The verification problem addressed by these new proposals can be formulated as follows: given a system S_N consisting of N infinite state processes and a temporal property φ_N, prove that, for all N, the system S_N verifies property φ_N.

The main difficulty of this verification problem is that most properties of interest, such as *safety* and *liveness* properties, are undecidable for that class of concurrent systems, and thus, there cannot be any complete method for their verification. For this reason, all proposed methods resort to semiautomatic techniques, based on either (i) mathematical induction, or (ii) reduction to finite state model checking by abstraction.

This paper describes a method for verifying safety properties of systems consisting of an arbitrary number of processes whose set of states can be either finite or infinite. For reasons of simplicity, throughout this paper we will refer to these processes as infinite state processes. Our method avoids the use mathematical induction by abstracting away from the number N of processes actually present in the system. Indeed, this parameter occurs in the encoding of neither the systems nor the safety properties to be verified. These encodings are expressed as *Constraint Logic Programs* [14], CLP for short, whose constraints are formulas of the *Weak Monadic Second-order Theory of k Successors*, denoted WSkS [28]. These programs will be called CLP(WSkS) programs. By using these encodings, the actual cardinality of the set of processes present in the systems is not required in the proofs of the properties of interest.

Our method uses *unfold/fold transformations* [5,20,26] as inference rules for constructing proofs. There are other verification methods proposed in the literature which use program transformation or are based on CLP [7,12,13,17,19,22,23], but those methods deal either with: (i) finite state systems [19,22], or (ii) infinite state systems where the number N of infinite state processes is fixed in advance [7,12,13,17], or (iii) *parameterized systems*, that is, systems consisting of an arbitrary number of finite state processes [23]. A more detailed discussion of these methods can be found in Section 6.

We assume that in our concurrent systems, every process evolves depending on its local state, called the *process state*, and depending also on the state of the other processes. Correspondingly, the whole system evolves and its global state, called the *system state*, changes. We also assume that each process state consists of a pair $\langle n, s\rangle \in \mathbb{N} \times CS$, where $\mathbb{N}$ denotes the set of natural numbers and CS is a given finite set. n and s are called the *counter* and the *control state* of the process, respectively. Notice that, during the evolution of the system, each process may reach an infinite number of distinct states.

This notion of process state derives from the specification of the Bakery Protocol (see Section 3 below) where a process is viewed as a finite state automaton which at each instante in time, is in a given control state and holds a natural number in a counter. We think, however, that our notion of process state is general enough to allow the specification of a large class of concurrent systems.

Since two distinct processes in a given system may have the same ⟨counter, control state⟩ pair, a system state is a *multiset* of process states.

As usual in model checking, a concurrent system is specified as a *Kripke structure* $\mathcal{K} = \langle S, S_0, R, E\rangle$, where: (i) S is the set of system states, that is, the set of multisets of ⟨control state, counter⟩ pairs, (ii) $S_0 \subseteq S$ is a set of *initial system states*, (iii) $R \subseteq S \times S$ is a *transition relation*, and (iv) $E \subseteq \mathcal{P}(S)$ is a finite set of *elementary properties*.

We also assume that for all $\langle X, Y\rangle \in R$, we have that $Y = (X - \{x\}) \cup \{y\}$ where: (i) x and y are some process states, and (ii) the difference and union operations are to be understood as multiset operations. Thus, a transition from a system state to a new system state consists in replacing a process state by a new process state. This assumption implies that: (i) the number of processes in the concurrent systems does not change over time, and (ii) the concurrent system is asynchronous, that is, the processes of the system do not necessarily synchronize their actions.

We will address the problem of proving safety properties of systems. A safety property is expressed by a formula of the *Computational Tree Logic* [6] (CTL, for short) of the form $\neg EF(\mathit{unsafe})$, where *unsafe* is an elementary property and *EF* is a temporal operator. The meaning of any such formula is given via the satisfaction relation $\mathcal{K}, X_0 \models \neg EF(\mathit{unsafe})$ which holds for a Kripke structure $\mathcal{K}$ and a system state X_0 iff there is no sequence of states $X_0, X_1, \ldots, X_n$ such that: (i) for $i = 0, \ldots, n-1$, $\langle X_i, X_{i+1} \rangle \in R$ and (ii) $X_n \in \mathit{unsafe}$.

We may extend our method to prove more complex properties, besides safety properties. In particular, we may consider those properties which can be expressed by using, in addition to $\neg$ and *EF*, other logical connectives and CTL temporal operators. However, for reasons of simplicity, in this paper we deal with safety properties only, and we do not consider nested temporal operators.

Now we outline our method for verifying that, for all initial system states X of a given Kripke structure $\mathcal{K}$, the safety property φ holds. We use the notions of locally stratified program and perfect model and for them we refer to [2].

Verification Method.

Step 1. (*System and Property Specification*) We introduce: (i) a WSkS formula $\mathit{init}(X)$ which characterizes the initial system states, that is, X is an initial system state iff $\mathit{init}(X)$ holds, and (ii) a locally stratified CLP(WSkS) program $P_{\mathcal{K}}$ which defines a binary predicate *sat* such that for each system state X,

$$\mathcal{K}, X \models \varphi \text{ iff } \mathit{sat}(X, \varphi) \in M(P_{\mathcal{K}}) \qquad (\dagger)$$

where $M(P_{\mathcal{K}})$ denotes the perfect model of the program $P_{\mathcal{K}}$.

Step 2. (*Proof Method*) We introduce a new predicate f defined by the following CLP(WSkS) clause F: $f(X) \leftarrow \mathit{init}(X), \mathit{sat}(X, \varphi)$, where X is a variable. We then apply the transformation rules of Section 4, and from program $P_{\mathcal{K}} \cup \{F\}$ we derive a new program P_f.
If the clause $f(X) \leftarrow \mathit{init}(X)$ occurs in P_f then for all initial system states X, we have that $\mathcal{K}, X \models \varphi$ holds.

The choice of the perfect model as the semantics of the program $P_{\mathcal{K}}$ requires a few words of explanation. By definition, $\mathcal{K}, X \models \neg\varphi$ holds iff $\mathcal{K}, X \models \varphi$ does not hold, and by using ($\dagger$), this fact can be expressed by the clause:

C: $\mathit{sat}(X, \neg\varphi) \leftarrow \neg\, \mathit{sat}(X, \varphi)$

where $\neg$ in the head of C is interpreted as a function symbol, while $\neg$ in the body of C is interpreted as negation by (finite or infinite) failure. Now, since clause C is a locally stratified clause and the other clauses for *sat* do not contain negated atoms (see Section 2.2), the semantics of negation by failure is the one captured by the perfect model (recall that for locally stratified programs the perfect model is identical to the stable model and also to the well-founded model [2]).

The paper is structured as follows. In Section 2 we describe Step 1 of our verification method and we introduce CLP(WSkS) programs, that is, constraint logic programs whose constraints are formulas in the WSkS theory. In Section 3 we illustrate our specification method by considering the case of a system of N processes which use the *Bakery Protocol* for ensuring mutual exclusion [16]. In Section 4 we present Step 2 of our verification method and we see how it is realized by applying suitable rules for program transformation. These rules are adaptations to the case of locally stratified CLP(WSkS) programs of the unfold/fold rules for generic CLP programs presented in [9,12]. We also provide a semiautomatic strategy for guiding the application of the transformation rules and proving the properties of interest. In Section 5, we see our strategy in action for the verification of a safety property the N-process Bakery Protocol. Finally, in Section 6 we compare our paper with the literature in the field and we discuss possible enhancements of our method.

2 System and Property Specification Using Constraint Logic Programs over WSkS

In this section we illustrate Step 1 of our verification method and, in particular, we indicate how to specify: (i) a system consisting of a set of infinite state processes, and (ii) a safety property we want to prove. We specify the given system by a Kripke structure $\mathcal{K} = \langle S, S_0, R, E \rangle$ where: (i) S is the set of finite sets of finite strings, which are ground terms of the WSkS theory, and (ii) S_0, R, and E are specified by suitable WSkS formulas. We specify the given safety property by defining a *sat* relation by means of a CLP program $P_{\mathcal{K}}$ whose constraints are WSkS formulas.

2.1 Constraint Logic Programs over WSkS

The Weak Monadic Second Order Theory of k Successors is a decidable theory which can be used for expressing properties of finite sets of finite strings over an alphabet of k symbols [27,28]. The syntax of WSkS is defined as follows. Let us consider a set $\Sigma = \{s_1, \ldots, s_k\}$ of k symbols, called *successors*, and a set *Ivars* of *individual variables*. An *individual term* is either a string σ or a string $x\sigma$, where $x \in$ *Ivars* and $\sigma \in \Sigma^*$, where Σ^* denotes the set of all finite strings of successor symbols. By ε we denote the *empty string.*

Let us also consider the set *Svars* of *set variables* ranged over by $X, Y, \ldots$

WSkS *terms* are either individual terms or set variables.

Atomic formulas of WSkS are either: (i) equalities between individual terms, written $t_1 = t_2$, or (ii) inequalities between individual terms, written $t_1 \leq t_2$, or (iii) membership atomic formulas, written $t \in X$, where t is an individual term and X is a set variable.

The *formulas* of WSkS are constructed from the atomic formulas by means of the usual logical connectives and the quantifiers over individual variables and set variables. Given any two individual terms, t_1 and t_2, we will also write: (i) $t_1 \neq t_2$ as a shorthand for $\neg(t_1 = t_2)$, and (ii) $t_1 < t_2$ as a shorthand for $t_1 \leq t_2 \wedge \neg(t_1 = t_2)$.

The *semantics* of WSkS formulas is defined by considering the interpretation $\mathcal{W}$ with domain Σ^* such that $=$ is interpreted as string equality, $\leq$ is interpreted as the prefix ordering on strings, and $\in$ is interpreted as membership of a string to a *finite* set of strings. We say that a closed WSkS formula φ holds iff the satisfaction relation $\mathcal{W} \models \varphi$ holds. The relation $\mathcal{W} \models \varphi$ is recursive [27].

A CLP(WSkS) program is a set of many-sorted first order formulas [8]. There are three sorts: *string*, *stringset*, and *tree*, interpreted as finite strings, finite sets of strings, and finite trees, respectively. We use many-sorted logic to avoid the formation of meaningless clauses such as $p(X, s_1) \leftarrow X = s_1$, where X is a set variable of sort *stringset* and s_1 is a constant in Σ of sort *string*.

CLP(WSkS) terms are either WSkS terms or *ordinary* terms (that is, terms constructed out of variables, constants, and function symbols which are all distinct from those used for WSkS terms). The WSkS individual terms are assigned the sort *string*, the WSkS set variables are assigned the sort *stringset*, and ordinary terms are assigned the sort *tree*. Each predicate of arity n is assigned the sort $\langle i_1, \ldots, i_n \rangle$, where for $j = 1, \ldots, n$, i_j is the sort of its j-th argument. For instance, the predicate $\in$ is assigned the sort $\langle string, stringset \rangle$. We assume that CLP(WSkS) programs are constructed by complying with the sorts of terms and predicates.

An *atom* is an atomic formula whose predicate symbol is not in $\{\leq, =, \in\}$. As usual, a *literal* is either an atom or a negated atom. A CLP(WSkS) clause is of the form $A \leftarrow c, L_1, \ldots, L_n$, where A is an atom, c is a formula of WSkS, and $L_1, \ldots, L_n$ are literals. We can extend to CLP(WSkS) programs the definitions of *locally stratified* programs and *perfect models*, by adapting the corresponding definitions which are given for logic programs [2].

2.2 The Specification Method

Now we present our method for specifying systems and their safety properties by using CLP(WSkS) programs. Recall that a system is specified as a Kripke structure $\langle S, S_0, R, E \rangle$ and a system state in S is a *multiset* of process states, that is, a multiset of pairs $\langle n, s \rangle$ where $n \in I\!N$ is a counter and $s \in CS$ is a control state. We assume that CS is a finite set $\{s_1, \ldots, s_h\}$ of symbols.

Now, let us indicate how to specify the four components of the Kripke structure.

(A) The Set S of System States. We consider the following set of successor symbols: $\Sigma = \{1, 2\} \cup CS$. A *process state* is represented as a term of the form $1^n s 2^m$, where: (i) 1^n and 2^m are (possibly empty) strings of 1's and 2's, respectively, and (ii) s is an element of CS. For a process state $1^n s 2^m$ we have that: (i) the string 1^n represents its counter (the empty string ε represents the counter 0), and (ii) the symbol s represents its control state. The string 2^m, with different values of m, is used to allow different terms to represent the same $\langle$counter, control state$\rangle$ pair, so that a *set* of terms each of which is of the form $1^n s 2^m$ can be used to represent a *multiset* of process states. Thus, a *system state* in S, being a multiset of process states, is represented as a *set* of terms, each of which is of the form $1^n s 2^m$.

Now we will show that process states and system states are expressible as formulas in WSkS. First we need the following definitions (for clarifying the reader's ideas, here and in the sequel, we write between parentheses the intended meanings):

- $\mathit{is\text{-}cn}(x) \;\equiv\; \exists X\; ((\forall y\; y \in X \rightarrow (y=\varepsilon \vee \exists z\; (y=z\,1 \wedge z\in X))) \wedge x\in X)$
  (x is a term of the form 1^n for some $n\geq 0$, i.e., x is a counter)
- $\mathit{is\text{-}cs}(x) \;\equiv\; x=s_1 \vee \ldots \vee x=s_h$
  ($x\in CS$, i.e., x is a control state)

Here are the WSkS formulas which define process states and system states:

- $ps(x) \;\equiv\; \exists X\; ((\forall y\; y\in X \rightarrow (\exists n\, \exists s\; y=n\,s \wedge \mathit{is\text{-}cn}(n) \wedge \mathit{is\text{-}cs}(s)) \vee \exists z\; (y=z\,2 \wedge z\in X))) \wedge x\in X)$
  (x is a process state, that is, a term of the form $1^n s 2^m$ for some $n, m \geq 0$ and $s\in CS$)
- $ss(X) \;\equiv \forall x\; (x\in X \rightarrow ps(x))$ (X is a system state, that is, a set of terms of the form $1^n s 2^m$)

(B) The Set S_0 of Initial System States. The set S_0 of initial system states is specified by a WSkS formula $\mathit{init}(X)$ where the set variable X is the only free variable, that is, $X \in S_0$ iff $\mathcal{W} \models \mathit{init}(X)$.

(C) The Transition Relation R. Now we describe the general form of the WSkS formulas which can be used for defining the transition relation R. We need the following two definitions:

- $cn(x,n) \;\equiv\; ps(x) \wedge \mathit{is\text{-}cn}(n) \wedge n\leq x \wedge (\forall y\; (y\leq x \wedge \mathit{is\text{-}cn}(y)) \rightarrow y\leq n)$
  (n is the counter of process state x)
- $cs(x,s) \;\equiv\; ps(x) \wedge \mathit{is\text{-}cs}(s) \wedge (\exists y\; \exists z\; (y\leq x \wedge \mathit{is\text{-}cn}(z) \wedge y=z\,s)$
  (s is the control state of process state x)

We recall that a transition consists in replacing in a system state an old process state by a new process state. This replacement is defined as follows (here and in the sequel the angle brackets $\langle,\rangle$ are used only to improve readability and they should not be considered as belonging to the syntax of WSkS):

- $\mathit{replace}(\langle n_1, s_1\rangle, X, \langle n_2, s_2\rangle, Y) \;\equiv\; ss(X) \wedge ss(\mathrm{Y}) \wedge \exists x\, (x\in X \;\wedge\; cn(x,n_1) \;\wedge\; cs(x,s_1)) \wedge \exists y\, (y\in Y \;\wedge\; cn(y,n_2) \;\wedge\; cs(y,s_2)) \wedge \forall z\, ((z\in X \;\wedge\; z\neq x) \leftrightarrow (z\in Y \;\wedge\; z\neq y))$
  ($Y = (X-\{x\}) \cup \{y\}$ for some process states $x \in X$ and $y \in Y$ such that: (i) x has counter n_1 and control state s_1 and (ii) y has counter n_2 and control state s_2)

We assume that any given transition relation R is specified by a finite disjunction of h formulas, that is, $\langle X, Y\rangle \in R$ iff $\mathcal{W} \models r(X,Y)$, where $r(X,Y) \equiv r_1(X,Y) \vee \ldots \vee r_h(X,Y)$ and, for $i = 1,\ldots,h$:

- $r_i(X,Y) \;\equiv\; \exists n_1\, \exists s_1\, \exists n_2\, \exists s_2\, (\mathit{replace}(\langle n_1, s_1\rangle, X, \langle n_2, s_2\rangle, Y) \wedge \mathit{event}_i(\langle n_1, s_1\rangle, X, \langle n_2, s_2\rangle)$

where $event_i(\langle n_1, s_1\rangle, X, \langle n_2, s_2\rangle)$ is a WSkS formula. In Section 3 we will present some examples of these formulas.

(D) The Set E of Elementary Properties. Each elementary property $\eta \in E$ of the system states is specified by a formula $e(X)$ where the set variable X is the only free variable, that is, $X \in \eta$ iff $\mathcal{W} \models e(X)$.

To end this section we indicate how to specify a safety property of a system by using a CLP(WSkS) program. Let us consider: (i) a system specified by a Kripke structure $\mathcal{K} = \langle S, S_0, R, E\rangle$ whose elementary properties are $\eta_1, \ldots, \eta_m$ specified by the formulas $e_1(X), \ldots, e_m(X)$, respectively, and whose transition relation is specified by $r_1(X,Y) \vee \ldots \vee r_h(X,Y)$, and (ii) a safety property of the form $\neg EF(\eta)$, where η is an elementary property. We introduce the following CLP(WSkS) program $P_\mathcal{K}$:

$$
\begin{array}{l}
sat(X, \eta_1) \leftarrow e_1(X) \\
\ldots \\
sat(X, \eta_m) \leftarrow e_m(X) \\
sat(X, \neg\varphi) \leftarrow \neg\ sat(X, \varphi) \\
sat(X, EF(\varphi)) \leftarrow sat(X, \varphi) \\
sat(X, EF(\varphi)) \leftarrow r_1(X,Y),\ sat(Y, EF(\varphi)) \\
\ldots \\
sat(X, EF(\varphi)) \leftarrow r_h(X,Y),\ sat(Y, EF(\varphi))
\end{array}
$$

which specifies the safety property $\neg EF(\eta)$ in the sense that, for every system state X in S, the following holds [11]:

$$\mathcal{K}, X \models \neg EF(\eta) \ \text{ iff } \ sat(X, \neg EF(\eta)) \in M(P_\mathcal{K})$$

Notice that the program $P_\mathcal{K}$ is locally stratified w.r.t. the size of the second argument of *sat,* and thus, it has a unique perfect model, denoted $M(P_\mathcal{K})$.

3 An Example of the Specification of a System and a Property: The N-Process Bakery Protocol

In this section we illustrate our method for specifying systems and properties in the case of the N-process Bakery Protocol. This protocol ensures mutual exclusion in a system made out of N processes which use a shared resource. Mutual exclusion holds iff the shared resource is used by at most one process at a time.

Let us first give a brief description of the protocol [16]. In this protocol each process state is a $\langle$counter, control state$\rangle$ pair $\langle n, s\rangle$, where the control state s is either $\underline{\mathtt{t}}$ or $\underline{\mathtt{w}}$ or $\underline{\mathtt{u}}$. The constants $\underline{\mathtt{t}}$, $\underline{\mathtt{w}}$, and $\underline{\mathtt{u}}$ stand for *think*, *wait*, and *use*, respectively. Let us denote the set $\{\underline{\mathtt{t}}, \underline{\mathtt{w}}, \underline{\mathtt{u}}\}$ by *CS*. As in the general case, in this protocol a system state is a multiset of process states.

A system state is initial iff each of its process states is $\langle 0, \underline{\mathtt{t}}\rangle$.

The transition relation from a system state X to a new system state Y, is specified as follows (recall that the $-$ and $\cup$ operations refer to multisets):

(T1: from *think* to *wait*) if there exists a process state $\langle n, \underline{\mathtt{t}}\rangle$ in X, then $Y = (X - \{\langle n, \underline{\mathtt{t}}\rangle\}) \cup \{\langle m{+}1, \underline{\mathtt{w}}\rangle\}$, where m is the maximum value of the counters of the processes states in X,

(T2: from *wait* to *use*) if there exists a process state $\langle n, \underline{\mathtt{w}}\rangle$ in X such that, for any process state $\langle m, s\rangle$ in $X - \{\langle n, \underline{\mathtt{w}}\rangle\}$, either $m = 0$ or $n < m$, then $Y = (X - \{\langle n, \underline{\mathtt{w}}\rangle\}) \cup \{\langle n, \underline{\mathtt{u}}\rangle\}$, and

(T3: from *use* to *think*) $Y = (X - \{\langle n, \underline{\mathtt{u}}\rangle\}) \cup \{\langle 0, \underline{\mathtt{t}}\rangle\}$.

The mutual exclusion property is expressed by the CTL formula $\neg EF(\mathit{unsafe})$, where *unsafe* is an elementary property which holds in a system state X iff there are at least two distinct process states in X with control state $\underline{\mathtt{u}}$.

In order to give a formal specification of our N-process Bakery Protocol we use the 5 successor symbols: 1, 2, $\underline{\mathtt{t}}$, $\underline{\mathtt{w}}$, and $\underline{\mathtt{u}}$. Thus, we consider the WS5S theory.

(A) The System States. A system state is a set of terms, each of which is of the form $1^n s 2^m$, where s is an element of $\{\underline{\mathtt{t}}, \underline{\mathtt{w}}, \underline{\mathtt{u}}\}$.

(B) The Initial System States. A system state X is initial iff $\mathcal{W} \models \mathit{init}(X)$, where:

- $\mathit{init}(X) \equiv \forall x\ (x \in X \rightarrow (\mathit{cn}(x, \varepsilon) \wedge \mathit{cs}(x, \underline{\mathtt{t}})))$
  (all process states in X have counter 0 and control state $\underline{\mathtt{t}}$)

(C) The Transition Relation. For specifying the transition relation for the N-process Bakery Protocol we need the following two predicates *max* and *min*:

- $\mathit{max}(X,m) \equiv \exists x\ (x \in X \wedge \mathit{cn}(x, m)) \wedge \forall y\ \forall n\ ((y \in X \wedge \mathit{cn}(y, n)) \rightarrow n \leq m)$
  (m is the maximum counter in the system state X)
- $\mathit{min}(X,m) \equiv \exists x\ (x \in X \wedge \mathit{cn}(x, m)) \wedge$
  $\forall y\ \forall n\ ((y \in X \wedge y \neq x \wedge \mathit{cn}(y, n)) \rightarrow (n = \varepsilon \vee m < n))$
  (In the system state X there exists a process state x with counter m such that the counter of any process state in $X - \{x\}$ is either 0 or greater than m. Recall that the term ε represents the counter 0.)

The transition relation between system states is defined as follows: $\langle X, Y\rangle \in R$ iff $\mathcal{W} \models \mathit{tw}(X, Y) \vee \mathit{wu}(X, Y) \vee \mathit{ut}(X, Y)$, where the predicates *tw*, *wu*, and *ut* correspond to the transition of a process from *think* to *wait*, from *wait* to *use*, and from *use* to *think*, respectively. We have that:

- $\mathit{tw}(X, Y) \equiv \exists n_1 \exists s_1 \exists n_2 \exists s_2\ \mathit{replace}(\langle n_1, s_1\rangle, X, \langle n_2, s_2\rangle, Y) \wedge$
  $s_1 = \underline{\mathtt{t}} \wedge \exists m\ (\mathit{max}(X, m) \wedge n_2 = m\,1) \wedge s_2 = \underline{\mathtt{w}}$
  ($Y = (X - \{x\}) \cup \{y\}$, where x is a process state in X with control state $\underline{\mathtt{t}}$, and y is a process with control state $\underline{\mathtt{w}}$ and counter $m{+}1$ such that m is the maximum counter in X. Notice that the term $m\,1$ represents the counter $m{+}1$)
- $\mathit{wu}(X, Y) \equiv \exists n_1 \exists s_1 \exists n_2 \exists s_2\ \mathit{replace}(\langle n_1, s_1\rangle, X, \langle n_2, s_2\rangle, Y) \wedge$
  $s_1 = \underline{\mathtt{w}} \wedge \mathit{min}(X, n_1) \wedge n_2 = n_1 \wedge s_2 = \underline{\mathtt{u}}$
  ($Y = (X - \{x\}) \cup \{y\}$, where x is a process state in X with counter n_1 and

control state $\underline{\mathtt{w}}$ such that the counter of any process state in $X-\{x\}$ is either 0 or greater than n_1, and y is a process state with counter n_1 and control state $\underline{\mathtt{u}}$)

– $ut(X,Y) \equiv \exists n_1 \exists s_1 \exists n_2 \exists s_2\, replace(\langle n_1, s_1\rangle, X, \langle n_2, s_2\rangle, Y) \wedge$
$s_1 = \underline{\mathtt{u}} \wedge n_2 = \varepsilon \wedge s_2 = \underline{\mathtt{t}}$
($Y = (X-\{x\}) \cup \{y\}$, where x is a process state in X with control state $\underline{\mathtt{u}}$, and y is a process state with counter 0 and control state $\underline{\mathtt{t}}$)

(D) The Elementary Properties. The unsafety property holds in each system state X such that $\mathcal{W} \models unsafe(X)$, where:

– $unsafe(X) \equiv \exists x \exists y\, (x \in X \wedge y \in X \wedge x \neq y \wedge cs(x, \underline{\mathtt{u}}) \wedge cs(y, \underline{\mathtt{u}}))$
(there exist two distinct process states in X with control state $\underline{\mathtt{u}}$)

The following locally stratified CLP(WSkS) program P_{Bakery} defines the predicate *sat* of Step 1 of our verification method.

$sat(X, unsafe) \leftarrow unsafe(X)$
$sat(X, \neg F) \leftarrow \neg\, sat(X, F)$
$sat(X, EF(\varphi)) \leftarrow sat(X, \varphi)$
$sat(X, EF(\varphi)) \leftarrow tw(X,Y),\ sat(Y, EF(\varphi))$
$sat(X, EF(\varphi)) \leftarrow wu(X,Y),\ sat(Y, EF(\varphi))$
$sat(X, EF(\varphi)) \leftarrow ut(X,Y),\ sat(Y, EF(\varphi))$

Thus, in order to verify the safety of the Bakery Protocol we have to prove that, for all system states X,

if $init(X)$ holds then $sat(X, \neg EF(unsafe)) \in M(P_{Bakery})$.

4 Rules and Strategy for Verification

In this section we show how Step 2 of our verification method is performed by using unfold/fold rules for transforming CLP(WSkS) programs. These rules are presented below. They are similar to those introduced in [9,12]. We also present a semiautomatic strategy for guiding the application of these transformation rules.

For presenting the transformation rules we need the following notation and terminology. By $FV(\varphi)$ we denote the set of free variables occurring in φ. By $v, w, \ldots$ (possibly with subscripts), we denote variables in $Ivars \cup Svars$. We say that the atom A is *failed* in program P iff in P there is no clause whose head is unifiable with A. The set of *useless predicates* of a program P is the maximal set U of predicates occurring in P such that the predicate p is in U iff the body of each clause defining p in P contains a positive literal whose predicate is in U. The set of *useless clauses* of a program P is the set of clauses defining useless predicates in P.

The process of transforming a given CLP(WSkS) program P_0 whereby deriving program P_n, can be formalized as a sequence $P_0, \ldots, P_n$ of programs, called a *transformation sequence*, where for $r = 0, \ldots, n-1$, program P_{r+1} is obtained from program P_r by applying one of the following transformation rules.

R1. Constrained Atomic Definition. Let δ be the clause:

$newp(v_1, \ldots, v_n) \leftarrow c, A$

where: (i) $newp$ is a new predicate symbol not occurring in $P_0, \ldots, P_r$, and (ii) $\{v_1, \ldots, v_n\} = FV(c, A)$. Then $P_{r+1} = P_r \cup \{\delta\}$.
Clause δ is called a *definition clause* and for $i \geq 0$, $Defs_i$ is the set of definition clauses introduced during the transformation sequence $P_0, \ldots, P_i$. In particular, $Defs_0 = \emptyset$.

R2. Unfolding. Let $\gamma \in P_r$ be the clause $H \leftarrow c, G_1, L, G_2$.
(R2p) If L is an atom A and $\{A_j \leftarrow c_j, B_j \mid j = 1, \ldots, m\}$ is the set of all renamed apart clauses in P_r such that the atoms A and A_j are unifiable via a most general unifier ϑ_j, then $P_{r+1} = (P_r - \{\gamma\}) \cup \{(H \leftarrow c,\, c_j,\, G_1, B_j, G_2)\vartheta_j \mid j = 1, \ldots, m\}$.

(R2n) If L is a negated atom $\neg A$ and A is failed in P_r, then $P_{r+1} = (P_r - \{\gamma\}) \cup \{H \leftarrow c,\, G_1, G_2\}$.

R3. Constrained Atomic Folding. Let γ be the clause $H \leftarrow c, G_1, L, G_2$ in P_r, where L is either the atom A or the negated atom $\neg A$. Let δ be a definition clause $newp(v_1, \ldots, v_n) \leftarrow d, A$ in $Defs_r$, such that $\mathcal{W} \models \forall w_1, \ldots, w_m(c \rightarrow d)$, where $\{w_1, \ldots, w_m\} = FV(c \rightarrow d)$.
(R3p) If L is A then $P_{r+1} = (P_r - \{\gamma\}) \cup \{H \leftarrow c, G_1, newp(v_1, \ldots, v_n), G_2\}$.
(R3n) If L is $\neg A$ then $P_{r+1} = (P_r - \{\gamma\}) \cup \{H \leftarrow c, G_1, \neg newp(v_1, \ldots, v_n), G_2\}$.

R4. Clause Removal. $P_{r+1} = P_r - \{\gamma\}$ if one of the following two cases occurs.
(R4f) γ is the clause $H \leftarrow c, G$ and c is unsatisfiable, that is, $\mathcal{W} \models \forall v_1, \ldots, v_n\, \neg c$, where $\{v_1, \ldots, v_n\} = FV(c)$.
(R4u) γ is useless in P_r.

R5. Constraint Replacement. Let γ be the clause $H \leftarrow c_1, G$. If for some WSkS formula c_2 we have that $\mathcal{W} \models \forall w_1, \ldots, w_n\, (c_1 \leftrightarrow c_2)$, where $\{w_1, \ldots, w_n\} = FV(c_1 \leftrightarrow c_2)$, then $P_{r+1} = (P_r - \{\gamma\}) \cup \{H \leftarrow c_2, G\}$.

These rules are different from those introduced in the case of general programs by Seki [24]. In particular, Seki's folding rule can be used for replacing a clause γ: $H \leftarrow c, G_1, \neg A, G_2$ by a new clause $\gamma 1$: $H \leftarrow c, G_1, newp(\ldots), G_2$, but not by a new clause $\gamma 2$: $H \leftarrow c, G_1, \neg newp(\ldots), G_2$. The replacement of clause γ by clause $\gamma 2$ is possible by using our folding rule R3n.

It can be shown that, under suitable restrictions, the transformation rules presented above preserve the perfect model semantics [11].

Step 2 of our verification method consists in applying the transformation rules R1–R5 according to a transformation strategy which we describe below. We will see this strategy in action for the verification of a safety property of the N-process Bakery Protocol (see Section 5).

Suppose that we are given a system specified by a Kripke structure $\mathcal{K}$ and a safety formula φ, and we want to verify that $\mathcal{K}, X \models \varphi$ holds for all initial system states X. Suppose also that $\mathcal{K}$ and φ are given by a CLP(WSkS) program $P_{\mathcal{K}}$ as described in Section 2.2. We proceed as follows. First we consider the clause:

F. $f(X) \leftarrow init(X),\ sat(X, \varphi)$

where: (i) f is a new predicate symbol, and (ii) $\mathcal{W} \models \mathit{init}(X)$ iff X is an initial system state.

Then we apply the following transformation strategy which uses a *generalization function* *gen*. Given a WSkS formula c and a literal L which is the atom A or the negated atom $\neg A$, the function *gen* returns a definition clause $\mathit{newp}(v_1, \ldots, v_n) \leftarrow d, A$ such that: (i) *newp* is a new predicate symbol, (ii) $\{v_1, \ldots, v_n\} = FV(d, A)$, and (iii) $\mathcal{W} \models \forall w_1, \ldots, w_n\, (c \rightarrow d)$, where $\{w_1, \ldots, w_n\} = FV(c \rightarrow d)$.

Transformation Strategy

Input: (i) Program $P_{\mathcal{K}}$, (ii) clause F: $f(X) \leftarrow \mathit{init}(X), \mathit{sat}(X, \varphi)$, and (iii) generalization function *gen*.

Output: A program P_f such that for every system state X, $f(X) \in M(P_{\mathcal{K}} \cup \{F\})$ iff $f(X) \in M(P_f)$.

Phase A. $\mathit{Defs} := \{F\}$; $\mathit{NewDefs} := \{F\}$; $P := P_{\mathcal{K}}$;

while $\mathit{NewDefs} \neq \emptyset$ **do**

1. from $P \cup \mathit{NewDefs}$ derive $P \cup C_{\mathit{unf}}$ by unfolding once each clause in *NewDefs*;
2. from $P \cup C_{\mathit{unf}}$ derive $P \cup C_r$ by removing all clauses with unsatisfiable body;
3. $\mathit{NewDefs} := \emptyset$;
   for each clause $\gamma \in C_r$ of the form $H \leftarrow c, G$ and for each literal L in the goal G such that γ cannot be folded w.r.t. L using a clause in *Defs* **do**
   $\mathit{NewDefs} := \mathit{NewDefs} \cup \{\mathit{gen}(c, L)\}$;
4. $\mathit{Defs} := \mathit{Defs} \cup \mathit{NewDefs}$;
5. fold each clause in C_r w.r.t. all literals in its body whereby deriving $P \cup C_{\mathit{fld}}$;
6. $P := P \cup C_{\mathit{fld}}$

end-while

Phase B.

1. from P derive P_u by removing all useless clauses in P;
2. from P_u derive P_f by unfolding the clauses in P_u w.r.t. every failed negative literal occurring in them.

Step 2 of the verification method ends by checking whether or not clause $f(X) \leftarrow \mathit{init}(X)$ occurs in program P_f. If it occurs, then for all initial system states X, we have that $\mathcal{K}, X \models \varphi$.

The correctness of our verification method is a consequence of the following two facts: (i) the transformation rules preserve perfect models, and (ii) perfect models are models of the completion of a program [2].

Theorem 1. [Correctness of the Verification Method] Given a Kripke structure $\mathcal{K}$ and a safety property φ, if $f(X) \leftarrow \mathit{init}(X)$ occurs in P_f then for all initial system states X, we have that $\mathcal{K}, X \models \varphi$.

Proof. Let us assume that $f(X) \leftarrow init(X)$ occurs in P_f and let us consider an initial system state I. Thus, $\mathcal{W} \models init(I)$ and $f(I) \in M(P_f)$. By the correctness of the transformation rules [11], we have that $f(I) \in M(P_{\mathcal{K}} \cup \{F\})$. Since: (i) $M(P_{\mathcal{K}} \cup \{F\})$ is a model of the completion $comp(P_{\mathcal{K}} \cup \{F\})$, (ii) the formula $\forall X\, (f(X) \leftrightarrow (init(X) \wedge sat(X, \varphi))$ belongs to $comp(P_{\mathcal{K}} \cup \{F\})$, and (iii) $\mathcal{W} \models init(I)$ we have that $sat(I, \varphi) \in M(P_{\mathcal{K}} \cup \{F\})$. Now, since no sat atom in $M(P_{\mathcal{K}} \cup \{F\})$ can be inferred by using clause F, we have that $sat(I, \varphi) \in M(P_{\mathcal{K}})$, that is, $\mathcal{K}, I \models \varphi$. □

The automation of our transformation strategy depends on the availability of a suitable generalization function *gen*. In particular, our strategy terminates whenever the codomain of *gen* is a finite set of definition clauses. Suitable generalization functions with finite codomain can be constructed by following an approach similar to the one described in [12]. More on this issue will be mentioned in Section 6.

Finally, let us notice that our verification method is *incomplete*, in the sense that there exist a Kripke structure $\mathcal{K}$, an initial system state X, and a safety property φ, such that $\mathcal{K}, X \models \varphi$ holds, and yet there is no sequence of applications of the transformation rules which leads from the program $P_{\mathcal{K}} \cup \{f(X) \leftarrow init(X), sat(X, \varphi)\}$ to a program P_f containing the clause $f(X) \leftarrow init(X)$. This incompleteness limitation cannot be overcome, because the problem of verifying properties of finite sets of infinite state processes is undecidable and not semidecidable. This is a consequence of the fact that the uniform verification of parameterized systems consisting of finite state processes is undecidable [3].

5 Verifying the N-process Bakery Protocol via Program Transformation

In this section we show how Step 2 of our verification method described in Section 4 is performed for verifying the safety of the N-process Bakery Protocol. We apply the unfold/fold transformation rules to the constraint logic program P_{Bakery} (see end of Section 3) according to the transformation strategy of Section 4.

As already remarked at the end of Section 4, the application of our strategy can be fully automatic, provided that we are given a generalization function which introduces new definition clauses needed for the folding steps (see Point 3 of the transformation strategy). In particular, during the application of the transformation strategy for the verification of the N-process Bakery Protocol which we now present, we have that: (i) all formulas to be checked for applying the transformations rules are formulas of WS5S, and thus, they are decidable, and (ii) the generalization function is needed for introducing clauses d3, d9, and d16 (see below).

We start off by introducing the following new definition clause:

d1. $f(X) \leftarrow init(X),\ sat(X, \neg EF(unsafe))$

Our goal is to transform the program $P_{Bakery} \cup \{\mathrm{d}1\}$ into a program P_f which contains a clause of the form $f(X) \leftarrow init(X)$.
We start Phase A by unfolding clause 1 w.r.t. the *sat* atom, thereby obtaining:

2. $f(X) \leftarrow init(X),\ \neg\ sat(X, EF(unsafe))$

The constraint $init(X)$ is satisfiable and clause 2 *cannot* be folded using the definition clause d1. Thus, we introduce the new definition clause:

d3. $newp1(X) \leftarrow init(X),\ sat(X, EF(unsafe))$

By using clause d3 we fold clause 2, and we obtain:

4. $f(X) \leftarrow init(X),\ \neg\ newp1(X)$

We proceed by applying the unfolding rule to the newly introduced clause d3, thereby obtaining:

5. $newp1(X) \leftarrow init(X) \wedge unsafe(X)$
6. $newp1(X) \leftarrow init(X) \wedge tw(X,Y),\ sat(Y, EF(unsafe))$
7. $newp1(X) \leftarrow init(X) \wedge wu(X,Y),\ sat(Y, EF(unsafe))$
8. $newp1(X) \leftarrow init(X) \wedge ut(X,Y),\ sat(Y, EF(unsafe))$

Clauses 5, 7 and 8 are removed, because their bodies contain unsatisfiable constraints. Indeed, the following formulas hold: (i) $\forall X\ \neg(init(X) \wedge unsafe(X))$, (ii) $\forall X\ \forall Y\ \neg(init(X) \wedge wu(X,Y))$, and (iii) $\forall X\ \forall Y\ \neg(init(X) \wedge ut(X,Y))$.

Clause 6 cannot be folded using either d1 or d3, because $\forall X\, \forall Y\, (init(X) \wedge tw(X,Y) \rightarrow init(Y))$ does not hold. Thus, in order to fold clause 6, we introduce the new definition clause:

d9. $newp2(X) \leftarrow c(X),\ sat(X, EF(unsafe))$

where $c(X)$ is a new constraint defined by the following WS5S formula:

$\forall x\ (x \in X \rightarrow ((cn(x,\varepsilon) \wedge cs(x,\underline{\mathtt{t}})) \vee (\exists c\ (cn(x,c) \wedge \varepsilon < c) \wedge cs(x,\underline{\mathtt{w}}))))$

This formula tells us that every process state in the system state X is either the pair $\langle 0, \underline{\mathtt{t}}\rangle$ or the pair $\langle c, \underline{\mathtt{w}}\rangle$ for some $c > 0$. We have that $\forall X\, \forall Y\, (init(X) \wedge tw(X,Y) \rightarrow c(Y))$ holds and thus, we can fold 6 using d9. We obtain:

10. $newp1(X) \leftarrow init(X) \wedge tw(X,Y),\ newp2(Y)$

By unfolding the definition clause d9 we obtain:

11. $newp2(X) \leftarrow c(X) \wedge unsafe(X)$
12. $newp2(X) \leftarrow c(X) \wedge tw(X,Y),\ sat(Y, EF(unsafe))$
13. $newp2(X) \leftarrow c(X) \wedge wu(X,Y),\ sat(Y, EF(unsafe))$
14. $newp2(X) \leftarrow c(X) \wedge ut(X,Y),\ sat(Y, EF(unsafe))$

Clauses 11 and 14 have unsatisfiable constraints in their bodies and we remove them. Indeed, the following formulas hold: (i) $\forall X\ \neg(c(X) \wedge unsafe(X))$, and (ii) $\forall X\ \forall Y\ \neg(c(X) \wedge ut(X,Y))$.

We fold clause 12 by using the already introduced definition clause d9, because $\forall X\, \forall Y\, (c(X) \wedge tw(X,Y) \rightarrow c(Y))$ holds. We obtain:

15. $newp2(X) \leftarrow c(X) \wedge tw(X,Y),\ newp2(Y)$

However, clause 13 cannot be folded by using a definition clause introduced so far. Thus, in order to fold clause 13, we introduce the following new definition clause:

d16. $newp3(X) \leftarrow d(X),\ sat(X, EF(unsafe))$

where the constraint $d(X)$ is the WS5S formula:

$$\begin{aligned}\forall x\ (x \in X \rightarrow ((&cn(x,\varepsilon) \wedge cs(x,\underline{\mathtt{t}})) \vee \\ &(\exists c\ (cn(x,c) \wedge \varepsilon < c\ ) \wedge cs(x,\underline{\mathtt{w}})) \vee \\ &(\exists n\ (cn(x,n) \wedge min(X,n) \wedge \varepsilon < n) \wedge cs(x,\underline{\mathtt{u}})))\end{aligned}$$

This formula tells us that every process state in the system state X is either $\langle 0,\underline{\mathtt{t}}\rangle$, or $\langle c,\underline{\mathtt{w}}\rangle$ for some $c>0$, or $\langle n,\underline{\mathtt{u}}\rangle$ for some $n>0$ such that no process state in X has a positive counter smaller than n. We have that $\forall X\, \forall Y\, (c(X) \wedge wu(X,Y) \rightarrow d(Y))$ holds, and thus, we can fold clause 13 using clause d16. We obtain:

17. $newp2(X) \leftarrow c(X) \wedge wu(X,Y),\ \ newp3(Y)$

We now proceed by applying the unfolding rule to the definition clause d16 and we get:

18. $newp3(X) \leftarrow d(X) \wedge unsafe(X)$
19. $newp3(X) \leftarrow d(X) \wedge tw(X,Y),\ sat(Y, EF(unsafe))$
20. $newp3(X) \leftarrow d(X) \wedge wu(X,Y),\ sat(Y, EF(unsafe))$
21. $newp3(X) \leftarrow d(X) \wedge ut(X,Y),\ sat(Y, EF(unsafe))$

We remove clause 18 because its body contains an unsatisfiable constraint because $\forall X\, \neg(d(X) \wedge unsafe(X))$ holds. Then, we fold clauses 19, 20, and 21 by using the definition clauses d16, d16, and d9, respectively. Indeed, the following three formulas hold:

$\forall X\, \forall Y\, (d(X) \wedge tw(X,Y) \rightarrow d(Y))$
$\forall X\, \forall Y\, (d(X) \wedge wu(X,Y) \rightarrow d(Y))$
$\forall X\, \forall Y\, (d(X) \wedge ut(X,Y) \rightarrow c(Y))$

We get:

22. $newp3(X) \leftarrow d(X) \wedge tw(X,Y),\ newp3(Y)$
23. $newp3(X) \leftarrow d(X) \wedge wu(X,Y),\ newp3(Y)$
24. $newp3(X) \leftarrow d(X) \wedge ut(X,Y),\ newp2(Y)$

Since these last folding steps were performed without introducing new definition clauses, we terminate Phase A of our transformation process. The program derived so far is $P_{Bakery} \cup \{4, 10, 15, 17, 22, 23, 24\}$.

Now we proceed by performing Phase B of our transformation strategy. We remove the useless clauses 10, 15, 17, 22, 23, and 24, which define the predicates $newp1$, $newp2$, and $newp3$. Therefore, we derive the program $P_{Bakery} \cup \{4\}$. Then we apply the unfolding rule to clause 4 w.r.t. the literal $\neg newp1(X)$, where $newp1(X)$ is a failed atom (see Point R2n of the unfolding rule). We obtain:

25. $f(X) \leftarrow init(X)$

Thus, we derive the final program P_f which is $P_{Bakery} \cup \{25\}$. According to Step 2 of our verification method, the presence of clause 25 in P_f proves, as desired, the mutual exclusion property for the N-process Bakery Protocol.

6 Related Work and Conclusions

Several methods have been recently proposed for the verification of *parameterized systems*, that is, systems consisting of an *arbitrary* number of *finite state* processes. Among them the method described in [23] is closely related to ours, in that it uses unfold/fold program transformations for generating induction proofs of safety properties of parameterized systems. However, our paper differs from [23] because we use constraint logic programs with locally stratified negation to specify concurrent systems and their properties, while [23] uses definite logic programs. Correspondingly, we use a different set of transformation rules. Moreover, we consider systems with an arbitrary number of *infinite state* processes and these systems are more general than parameterized systems.

Now we recall the main features of some verification methods based on (constraint) logic programming, which have been recently proposed in the literature. (i) The method described in [17] uses partial deduction and abstract interpretation of logic programs for verifying safety properties of infinite state systems. (ii) The method presented in [13] uses logic programs with linear arithmetic constraints and Presburger arithmetic to verify safety properties of Petri nets. (iii) The method presented in [7] uses constraint logic programs to represent infinite state systems. This method can be applied to verify CTL properties of those systems by computing approximations of least and greatest fixed points via abstract interpretation. (iv) The method proposed in [22] uses tabulation-based logic programming to efficiently verify μ-calculus properties of finite state transitions systems expressed in a CCS-like language. (v) The method described in [19] uses CLP with finite domains, extended with constructive negation and tabled resolution, for finite state local model checking.

With respect to these methods (i)–(v), the distinctive features of our method are that: (1) we deal with systems consisting of an *arbitrary* number of infinite state processes, (2) we use CLP(WSkS) for their description, and (3) we apply unfold/fold program transformations for the verification of their properties.

Verification techniques for systems with an arbitrary number of infinite state processes have been presented also in the following papers.

In [18] the authors introduce a proof technique which is based on induction and model checking. Proofs are carried out by solving a finite number of model checking problems on a finite abstraction of the given system and they are mechanically checked. The technique is illustrated by proving that the N-process Bakery Protocol is starvation free.

In [21] the author presents a proof of the mutual exclusion for the N-process version of the Ticket Protocol [1] which is uniform w.r.t. N and it is based on the Owicki-Gries assertional method. The proof has been mechanically checked by using the Isabelle theorem prover.

In [25] the author presents a proof of the mutual exclusion for the N-process Bakery Protocol. This proof is based on theorem proving, model checking, and abstraction, so to reduce the protocol itself to the case of two processes only.

Similarly to the techniques presented in the above three papers [18,21,25], each step of our verification method can be mechanized, but the construction of the whole proof requires some human guidance. However, in contrast to [18,21,25] in our approach the parameter N representing the number of processes is *invisible*. Moreover, we do not use induction on N is performed and we do not perform any abstraction on the set of processes.

More recently, in [4] the authors have presented an automated method for the verification of safety properties of parameterized systems with unbounded local data. The method, which is based on multiset rewriting and constraints, is complete for a restricted class of parameterized systems.

The verification method presented in this paper is an enhancement of the *rules + strategies* transformation method proposed in [12] for verifying CTL properties of systems consisting of a fixed number of infinite state processes. In particular, Step 2 of our verification method can be viewed as a strategy for the specialization of program $P_{\mathcal{K}}$ encoding the system and the property of interest w.r.t. the goal $init(X), sat(X, \varphi)$. In [12] we proved the mutual exclusion property for the 2-process Bakery Protocol by using CLP programs with constraints expressed by linear inequations over real numbers. That proof can easily be extended to the case of any fixed number of processes by using CLP programs over the same constraint theory. Here, however, we proved the mutual exclusion property for the N-process Bakery Protocol, *uniformly* for any N, by using CLP programs with constraints over WSkS.

The proof of the mutual exclusion property for the N-process Bakery Protocol presented in Section 5, was done by applying under some human guidance the transformation strategy of Section 4. Notice, however, that our verification method can be made fully automatic by adding to our CLP program transformation system MAP [10]: (i) a solver for checking WSkS formulas, and (ii) suitable generalization functions for introducing new definition clauses. For Point (i) we may use existing solvers, such as MONA [15]. Point (ii) requires further investigation but we believe that one can apply some of the ideas presented in [12] in the case of systems consisting of a fixed number of infinite state processes.

As discussed in the Introduction, the verification method we proposed in this paper, is tailored to the verification of safety properties for *asynchronous* concurrent systems, where each transition is made by one process at a time. This limitation to asynchronous systems is a consequence of our assumption that each transition from a system state X to a new system state Y is of the form $Y = (X - \{x\}) \cup \{y\}$ for some process states x and y. In order to model *synchronous* systems, where transitions may involve more than one process at a time, we may relax this assumption and allow transitions of the form $Y = (X - A) \cup B$ for some multisets of process states A and B. Since these more general transitions whereby the number of processes may change over time, can be defined by WSkS formulas, one might use our method for verifying properties of synchronous systems.

References

1. G. R. Andrews. *Concurrent programming: principles and practice.* Addison-Wesley, 1991.

2. K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19, 20:9–71, 1994.
3. K. R. Apt and D. C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986.
4. M. Bozzano and G. Delzanno. Beyond parameterized verification. In *Proceedings of the Eighth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)"*, Lecture Notes in Computer Science 2280, pages 221–235. Springer, 2002.
5. R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.
6. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
7. G. Delzanno and A. Podelski. Model checking in CLP. In R. Cleaveland, editor, *5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, Lecture Notes in Computer Science 1579, pages 223–239. Springer-Verlag, 1999.
8. H. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.
9. S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.
10. F. Fioravanti. MAP: A system for transforming constraint logic programs. available at `http://www.iasi.rm.cnr.it/~fioravan`, 2001.
11. F. Fioravanti. *Transformation of Constraint Logic Programs for Software Specialization and Verification*. PhD thesis, Università di Roma "La Sapienza", Italy, 2002.
12. F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying CTL properties of infinite state systems by specializing constraint logic programs. In *Proceedings of the ACM Sigplan Workshop on Verification and Computational Logic VCL'01, Florence (Italy)*, Technical Report DSSE-TR-2001-3, pages 85–96. University of Southampton, UK, 2001.
13. L. Fribourg and H. Olsén. A decompositional approach for computing least fixed-points of Datalog programs with z-counters. *Constraints*, 2(3/4):305–335, 1997.
14. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
15. N. Klarlund and A. Møller. *MONA Version 1.4 User Manual*. BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus, January 2001.
16. L. Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications of the ACM*, 17(8):453–455, 1974.
17. M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialization. In A. Bossi, editor, *Proceedings of LOPSTR '99, Venice, Italy*, Lecture Notes in Computer Science 1817, pages 63–82. Springer, 1999.
18. K. L. McMillan, S. Qadeer, and J. B. Saxe. Induction in compositional model checking. In *CAV 2000*, Lecture Notes in Computer Science 1855, pages 312–327. Springer, 2000.
19. U. Nilsson and J. Lübcke. Constraint logic programming for local and symbolic model-checking. In J. W. Lloyd, editor, *CL 2000: Computational Logic*, Lecture Notes in Artificial Intelligence 1861, pages 384–398, 2000.
20. A. Pettorossi and M. Proietti. Transformation of logic programs: Foundations and techniques. *Journal of Logic Programming*, 19,20:261–320, 1994.
21. L. Prensa-Nieto. Completeness of the Owicki-Gries system for parameterized parallel programs. In *Formal Methods for Parallel Programming: Theory and Applications, FMPPTA 2001*. IEEE Computer Society Press, 2001.

22. Y. S. Ramakrishna, C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, T. Swift, and D. S. Warren. Efficient model checking using tabled resolution. In *CAV '97*, Lecture Notes in Computer Science 1254, pages 143–154. Springer-Verlag, 1997.
23. A. Roychoudhury and I.V. Ramakrishnan. Automated inductive verification of parameterized protocols. In *CAV 2001*, pages 25–37, 2001.
24. H. Seki. Unfold/fold transformation of stratified programs. *Theoretical Computer Science*, 86:107–139, 1991.
25. N. Shankar. Combining theorem proving and model checking through symbolic analysis. In *CONCUR 2000: Concurrency Theory*, number 1877 in Lecture Notes in Computer Science, pages 1–16, State College, PA, August 2000. Springer-Verlag.
26. H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-Å. Tärnlund, editor, *Proceedings of the Second International Conference on Logic Programming, Uppsala, Sweden*, pages 127–138. Uppsala University, 1984.
27. J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–82, 1968.
28. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.