Parameterized Verification of Infinite-state Processes with Global Conditions

Parosh Aziz Abdulla[1] parosh@it.uu.se,
Giorgio Delzanno[2] giorgio@disi.unige.it, and
Ahmed Rezine[1] Rezine.Ahmed@it.uu.se

[1] Uppsala University, Sweden
[2] Università di Genova, Italy.

Abstract. We present a simple and effective approximated backward reachability algorithm for parameterized systems with existentially and universally quantified global conditions. The individual processes operate on unbounded local variables ranging over the natural numbers. In addition, processes may communicate via broadcast, rendez-vous and shared variables. We apply the algorithm to verify mutual exclusion for complex protocols such as Lamport's bakery algorithm both with and without atomicity conditions, a distributed version of the bakery algorithm, and Ricart-Agrawala's distributed mutual exclusion algorithm.

1 Introduction

We consider the analysis of safety properties for *parameterized systems*. A parameterized system consists of an arbitrary number of processes. The task is to verify correctness regardless of the number of processes. This amounts to the verification of an infinite family; namely one for each size of the system. Most existing approaches to automatic verification of parameterized systems make the restriction that each process is finite-state. However, there are many applications where the behaviour relies on unbounded data structures such as counters, priorities, local clocks, time-stamps, and process identifiers.

In this paper, we consider parameterized systems where the individual processes operate on Boolean variables, and on numerical variables which range over the natural numbers. The transitions are conditioned by the local state of the process, values of the local variables; and by *global conditions* which check the local states and variables of the other processes. These conditions are stated as propositional constraints on the Boolean variables, and as *gap-order constraints* on the numerical variables. Gap-order constraints [19] are a logical formalism in which we can express simple relations on variables such as lower and upper bounds on the values of individual variables; and equality, and gaps (minimal differences) between values of pairs of variables. A global condition is either *universally* or *existentially* quantified. An example of a universal condition is "variable x of a given process i has a value which is greater than the value of variable y in all other processes inside the system". Process i is then allowed to perform the transition only if this condition is satisfied. In an existential condition we require that *some* (rather than *all*) processes satisfy the condition. In addition to these classes of transitions, processes may communicate through broadcast, rendez-vous, and shared variables.

There are at least two advantages with using gap-order constraints as a language for expressing the enabling conditions of transitions. First, they allow to handle a large class of protocols where the behaviour depends on the relative ordering of values among variables, rather than the actual values of these variables. The second reason is that they define a natural ordering on the set of system configurations. In fact, it can be shown, using standard techniques (such as the ones in [22]), that checking safety properties (expressed as regular languages) can be translated into the reachability of sets of configurations which are upward closed with respect to this ordering.

To check safety properties, we perform backward reachability analysis using gap-order constraints as a symbolic representation of upward closed sets of configurations. In the analysis, we consider a transition relation which is an over-approximation of the one induced by the parameterized system. To do that, we modify the semantics of universal quantifiers by eliminating the processes which violate the given condition. For instance in the above example, process i is always allowed to take the transition. However, when performing the transition, we eliminate each process j where the value of y is smaller than the value of x in i. The approximate transition system obtained in this manner is *monotonic* with respect to the above mentioned ordering, in the sense that larger configurations can simulate smaller ones. In fact, it turns out that universal quantification is the only operation which does not preserve monotonicity and hence it is the only source of approximation in the model. The fact that the approximate transition relation is monotonic, means that upward closedness is maintained under the operation of computing predecessors. Therefore, all the sets which are generated during the backward reachability analysis procedure are upward closed, and can hence be represented by gap-order constraints. A significant aspect of the reachability procedure is that the number of copies of variables (both Boolean and numerical) which appear in the upward closed sets is not bounded a priori. The reason is that there is an arbitrary number of processes each with its own local copy of the variables. The whole verification process is fully automatic since both the approximation and the reachability analysis are carried out without user intervention. Observe that if the approximate transition system satisfies a safety property then we can conclude that the original system satisfies the property, too.

Termination of the approximated backward reachability analysis is not guaranteed in general. However, the procedure terminates on all the examples we report in this paper. Furthermore, termination is guaranteed in some restricted cases such as for systems with existential or universal global conditions but with at most one local integer variable.

In order to test our method we have selected a collection of challenging protocols in which integer variables are used either as identifiers, priorities, local clocks, or time-stamps. Almost all of the examples are outside the class for which termination is guaranteed. In particular, we automatically verify safety properties for parameterized versions of the following algorithms:

– Lamport's bakery algorithm [18] with atomicity conditions;
– A version of Lamport's bakery algorithm with non-atomic computation of tickets;
– A distributed version of Lamport's bakery in which tickets and entry conditions are computed and tested non-atomically by means of a receive-reply protocol run by each process;
– The Ticket mutual exclusion algorithm with a central monitor for distributing tickets [5];
– The Ricart-Agrawala distributed mutual exclusion algorithm based on the use of logical clocks and time-stamps [20].

We also consider a bogus version of the Lamport's bakery without atomicity conditions in the computation of tickets. In this version, the *choosing* flag is simply ignored in the entry section. For this example, our procedure returns symbolic traces (from initial to bad states) that explain the subtle race conditions that may arise when the flag is not tested.

Each one of these examples present challenging problems for parameterized verification methods in the following sense:

– Their underlying logic is already hard for manual or finite-state verification.
– They are all instances of multidimensional infinite-state systems in which processes have unbounded local variables and (apart from Ticket) an order over identifiers is used to break the tie in the entry section. For instance, they cannot be modelled without the use of abstractions in the framework of Regular Model Checking [16, 4, 7, 3].
– In all examples, global conditions are needed to model the communication mechanisms used in the protocols (e.g. broadcasts, update, and entry conditions that depend on the local integer variables of other processes).

Related Work The multi-dimensional parameterized models studied in the present paper cannot be analyzed without use of additional abstractions by methods designed for networks of finite-state processes, e.g., Regular Model Checking [16, 4, 7] and counter abstraction methods [11, 15, 12, 13]. The approximation scheme we apply in our backward reachability procedure works well for a very large class of one-dimensional parameterized systems. In fact, the verification procedure used in [3] is a special case of the current one, where the processes are restricted to be finite-state systems.

Parameterized versions of Lamport's bakery algorithm have been tested using a semi-automated verification method based on *invisible invariants* in [6], with the help of *environmental abstractions* for a formulation with atomicity conditions in [10], and using heuristics to discover *indexed predicates* in [17]. A parameterized formulation of the Ricart-Agrawala algorithm has been verified semi-automatically in [21], where the STeP prover is used to discharge some of the verification conditions needed in the proof. We are not aware of other attempts of fully automatic verification of parameterized versions of the Ricart-Agrawala algorithm or of the distributed version (with no atomicity assumptions) of Lamport's bakery algorithm.

In contrast to the above mentioned methods, our verification procedure is fully automated and it is based on a generic approximation scheme. Furthermore, our method is applicable to versions of Lamport's bakery both with and without atomicity conditions and may return symbolic traces useful for debugging.

A parameterized formulation of an abstraction of the Ticket algorithm has been analyzed in [8]. The verification procedure in [8] does not handle parameterized universally quantified global conditions. Furthermore, in the abstraction of the Ticket algorithm studied in [8] the central monitor may forget tickets (the update of *turn* is defined by a jump to any larger value). Thus, the model does not keep the FIFO order of requests. With the help of universally quantified guards and of our approximation, we verify a more precise model in which the FIFO order of requests is always preserved.

In contrast to symbolic methods for finite, a priori fixed collections of processes with local integer variables, e.g., those in [9, 14], our gap-order constraints are defined over an *infinite* collections of variables. The number of copies of variables needed during the backward search cannot be bounded a priori. This feature allows us to reason about systems with global conditions over any number of processes. Furthermore, the present method covers that of [2] which also uses gap-order constraints to reason about systems with unbounded numbers of processes. However, [2] cannot deal with global conditions which is the main feature of the examples considered here.

Outline In the next two Sections we give some preliminaries and define a basic model for parameterized systems. Section 4 and 5 describe the induced transition system and the coverability (safety) problem. In Section 6 we define the approximated transition system. Section 7 defines the gap-order constraints and presents the backward reachability algorithm, while Section 8 describes the operations on constraints used in the algorithm. Section 9 explains how we extend the basic model to cover features such as shared variables, broadcast and binary communication. In Section 10 we report the results of running our prototypes on a number of examples. Finally, in Section 11, we give conclusions and directions for future work. In the appendix, we give some proofs and detailed descriptions of the case studies.

2 Preliminaries

In this section, we give some preliminary notations and definitions. We use $\mathcal{B}$ to denote the set $\{true, false\}$ of Boolean values; and use $\mathcal{N}$ to denote the set of natural numbers. For a natural number n, let $\overline{n}$ denote the set $\{1, \ldots, n\}$.

For a finite set A, we write a multiset over A as a list $[a_1, a_2, \ldots, a_n]$, where $a_i \in A$ for each $i : 1 \leq i \leq n$. We use $a \in A$ to denote that $a = a_i$ for some $i : 1 \leq$

$i \leq n$. For multisets $M_1 = [a_1, \ldots, a_m]$ and $M_2 = [b_1, \ldots, b_n]$, we use $M_1 \bullet M_2$ to denote the union (sum) of M_1 and M_2 (i.e., $M_1 \bullet M_2 = [a_1, \ldots, a_m, b_1, \ldots, b_n]$).

We will work with sets of variables. Such a set A is often partitioned into two subsets: *Boolean* variables $A_{\mathcal{B}}$ which range over $\mathcal{B}$, and *numerical* variables $A_{\mathcal{N}}$ which range over $\mathcal{N}$. We denote by $\mathbb{B}(A_{\mathcal{B}})$ the set of Boolean formulas over $A_{\mathcal{B}}$. We will also use a simple set of formulas, called *gap formulas*, to constrain the numerical variables. More precisely, we let $\mathbb{G}(A_{\mathcal{N}})$ be the set of formulas which are either of the form $x = y$ or of the form $x <_c y$ where $\sim \in \{<, \leq\}$, $x, y \in A_{\mathcal{N}} \cup \mathcal{N}$, and $c \in \mathcal{N}$. Here $x <_c y$ stands for $x + c < y$. We use $\mathbb{F}(A)$ to denote the set of formulas which has members of $\mathbb{B}(A)$ and of $\mathbb{G}(\mathcal{N})$ as atomic formulas, and which is closed under the Boolean connectives $\wedge, \vee$. For instance, if $A_{\mathcal{B}} = \{a, b\}$ and $A_{\mathcal{N}} = \{x, y\}$ then $\theta = (a \supset b) \wedge (x + 3 < y)$ is in $\mathbb{F}(A)$. Sometimes, we write a formula as $\theta(y_1, \ldots, y_k)$ where $y_1, \ldots, y_k$ are the variables which may occur in θ; so we can write the above formula as $\theta(x, y, a, b)$.

Sometimes, we perform substitutions on logical formulas. A *substitution* is a set $\{x_1 \leftarrow e_1, \ldots, x_n \leftarrow e_n\}$ of pairs, where x_i is a variable and e_i is either a constant or a variable of the same type as x_i, for each $i : 1 \leq i \leq n$. Here, we assume that all the variables are distinct, i.e., $x_i \neq x_j$ if $i \neq j$. For a formula θ and a substitution S, we use $\theta[S]$ to denote the formula we get from θ by simultaneously replacing all occurrences of the variables $x_1, \ldots, x_n$ by $e_1, e_2, \ldots, e_n$ respectively. Sometimes, we may write $\theta[S_1][S_2] \cdots [S_m]$ instead of $\theta[S_1 \cup S_2 \cup \cdots \cup S_m]$. As an example, if $\theta = (x_1 < x_2) \wedge (x_3 < x_4)$ then $\theta[x_1 \leftarrow 3, x_4 \leftarrow 2][x_2 \leftarrow y] = (3 < y) \wedge (x_3 < 2)$.

3 Parameterized Systems

In this section, we introduce a basic model for parameterized systems. The basic model will be enriched by additional features in Section 9.

A parameterized system consists of an arbitrary (but finite) number of identical processes. Each process is modelled as an extended finite-state automaton operating on local variables which range over the Booleans and the natural numbers. The transitions of the automaton are conditioned by the values of the local variables and by *global* conditions in which the process checks, for instance, the local states and variables of all the other processes inside the system. A transition may change the value of any local variable inside the process (possibly deriving the new values from those of the other processes). A parameterized system induces an infinite family of (potentially infinite-state) systems, namely one for each size n. The aim is to verify correctness of the systems for the whole family (regardless of the number n of processes inside the system).

Formally, a *parameterized system* $\mathcal{P}$ is a triple (Q, X, T), where Q is a finite set of *local states*, X is a finite set of *local variables* partitioned into $X_{\mathcal{B}}$ (which range over $\mathcal{B}$) and $X_{\mathcal{N}}$ (which range over $\mathcal{N}$), and T is a finite set of *transition rules*. A transition rule t is of the form

$$t : \left[q \rightarrow q' \;\; \rhd \;\; \theta \right] \tag{1}$$

where $q, q' \in Q$ and θ is either a *local* or a *global condition*. Intuitively, the process which makes the transition changes its local state from q to q'. In the meantime, the values of the local variables of the process are updated according to θ. Below, we describe how we define local and global conditions.

To simplify the definitions, we sometimes regard members of the set Q as Boolean variables. Intuitively, the value of the Boolean variable $q \in Q$ is *true* for a particular process iff the process is in local state q. We define the set $Y = X \cup Q$.

To define local conditions, we introduce the set X^{next} which contains the *next-value* versions of the variables in X. A variable $x^{next} \in X^{next}$ represents the next value of $x \in X$. A *local condition* is a formula in $\mathbb{F}(X \cup X^{next})$. The formula specifies how local variables of the current process are updated with respect to their current values. This operation is performed locally without checking the local variables of the other processes.

Global conditions check the values of local variables of the current process, together with the local states and the values of local variables of the other processes. Consequently, we need to distinguish between a local variable, say x, of the process which is about to perform a transition, and the same local variable x of the other processes inside the system. In order to do that, we introduce for each $x \in Y$, two new variables $\texttt{self}\cdot x$ and $\texttt{other}\cdot x$. We define the sets $\texttt{self}\cdot Y = \{\texttt{self}\cdot x | x \in Y\}$ and $\texttt{other}\cdot Y = \{\texttt{other}\cdot x | x \in Y\}$. The sets $\texttt{self}\cdot X$, $\texttt{other}\cdot X^{next}$, etc, are defined in the obvious manner. A *global condition* θ is of one of the following two forms:

$$\forall\ \texttt{other} \neq \texttt{self} \cdot\ \theta_1 \qquad \exists\ \texttt{other} \neq \texttt{self} \cdot\ \theta_1 \tag{2}$$

where $\theta_1 \in \mathbb{F}(\texttt{self}\cdot X \cup \texttt{other}\cdot Y \cup \texttt{self}\cdot X^{next})$. In other words, the formula checks the local variables of the process which is about to make the transition (through $\texttt{self}\cdot X$), and the local states and variables of the other processes (through $\texttt{other}\cdot Y$). It also specifies how the local variables of the process in transition are updated (through $\texttt{self}\cdot X^{next}$). Notice that the new values are defined in terms of the current values of variables and local states of all the processes inside the system. A global condition is said to be *universal* or *existential* depending on the type of the quantifier appearing in it.

As an example, the following formula

$$\forall\ \texttt{other} \neq \texttt{self} \cdot\ (\texttt{self}\cdot a) \wedge (\texttt{self}\cdot x^{next} > \texttt{other}\cdot x) \wedge \texttt{other}\cdot q_1$$

states that the transition may be performed only if variable a of the current process has the value *true*, and all the other processes are in local state q_1. When the transition is performed, variable x of the current process is assigned a value which is greater than the value of x in all the other processes.

4 Transition System

We describe the transition system induced by a parameterized system.

A *transition system* $\mathcal{T}$ is a pair $(D, \Longrightarrow)$, where D is an (infinite) set of *configurations* and $\Longrightarrow$ is a binary relation on D. We use $\overset{*}{\Longrightarrow}$ to denote the reflexive transitive closure of $\Longrightarrow$. Let $\preceq$ be an ordering on D. We say that $\mathcal{T}$ is *monotonic* with respect to $\preceq$ if the following property is satisfied: for all $c_1, c_2, c_3 \in D$ with $c_1 \Longrightarrow c_2$ and $c_1 \preceq c_3$, there is a $c_4 \in D$ such that $c_3 \Longrightarrow c_4$ and $c_2 \preceq c_4$. We will consider several transition systems in this paper.

First, a parameterized system $\mathcal{P} = (Q, X, T)$ induces a transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$ as follows. A configuration is defined by the local states and the values of the local variables in the processes. Formally, a *local variable state* v is a mapping from X to $\mathcal{B} \cup \mathcal{N}$ which respects the types of the variables. A *process state* u is a pair (q, v) where $q \in Q$ and v is a local variable state. As mentioned in Section 3, we sometimes regard members of the set Q as Boolean variables. Thus, we can view a process state (q, v) as a mapping $u : Y \mapsto \mathcal{B} \cup \mathcal{N}$, where $u(x) = v(x)$ for each $x \in X$, $u(q) = \mathit{true}$, and $u(q') = \mathit{false}$ for each $q' \in Q - \{q\}$. The process state thus agrees with v on the values of local variables, and maps all elements of Q, except q, to *false*. A *configuration* is a multiset $[u_1, u_2, \ldots, u_n]$ of process states. Intuitively, the above configuration corresponds to an instance of the system with n processes. Each pair $u_i = (q_i, v_i)$ gives the local state and the values of local variables of one process in the system. Notice that if c_1 and c_2 are configurations then so is their union $c_1 \bullet c_2$.

We define the transition relation $\longrightarrow$ on the set of configurations as follows. We start by describing the semantics of local conditions. Recall that a local condition corresponds to one process changing state without checking states of the other processes. Therefore, the semantics is defined in terms of two local variable states v, v' corresponding to the current resp. next values of the local variables of the process; and a formula $\theta \in \mathbb{F}(X \cup X^{next})$ (representing the local condition). We write $(v, v') \models \theta$ to denote the validity of the formula $\theta\,[\rho]\,[\rho']$ where the substitutions are defined by $\rho := \{x \leftarrow v(x) |\, x \in X\}$ and $\rho' := \{x^{next} \leftarrow v'(x) |\, x \in X\}$. In other words, we check the formula we get by replacing the current- resp. next-value variables in θ by their values as defined by v resp. v'. The formula is evaluated using the standard interpretations of the Boolean connectives, and the arithmetical relations $<, \leq, =$. For process states $u = (q, v)$ and $u' = (q', v')$, we use $(u, u') \models \theta$ to denote that $(v, v') \models \theta$.

Next, we describe the semantics of global conditions. The definition is given in terms of two local variable states v, v', a process state u_1, and a formula $\theta \in \mathbb{F}\,(\mathtt{self}{\cdot}X \cup \mathtt{other}{\cdot}Y \cup \mathtt{self}{\cdot}X^{next})$ (representing a global condition). The roles of v and v' are the same as for local conditions. We recall that a global condition also checks states of all (or some) of the other processes. Here, u_1 represents the local state and variables of one such a process. We write $(v, v', u_1) \models \theta$ to denote the validity of the formula $\theta\,[\rho]\,[\rho']\,[\rho_1]$ where the substitutions are defined by $\rho := \{\mathtt{self}{\cdot}x \leftarrow v(x) |\, x \in X\}$, $\rho' := \{\mathtt{self}{\cdot}x^{next} \leftarrow v'(x) |\, x \in X\}$, and $\rho_1 := \{\mathtt{other}{\cdot}x \leftarrow u_1(x) |\, x \in Y\}$. The substitutions on v and v' are analogous to the case of local conditions. In addition, we replace the local states and variables of the other processes by their values as defined by v_1. The relation $(u, u', u_1) \models \theta$ is interpreted in a similar manner to the case of local conditions.

Now, we are ready to define the transition relation $\longrightarrow$. Let t be a transition rule of the form of (1). Consider two configurations $c = c_1 \bullet [u] \bullet c_2$ and $c' = c_1 \bullet [u'] \bullet c_2$. We denote by $c \xrightarrow{t} c'$ that one of the following conditions is satisfied:

1. θ is a local condition and $(u, u') \models \theta$.
2. θ is a universal global condition of the form of (2), and $(u, u', u_1) \models \theta_1$ for each $u_1 \in c_1 \bullet c_2$.
3. θ is an existential global condition of the form of (2), and $(u, u', u_1) \models \theta_1$ for some $u_1 \in c_1 \bullet c_2$.

We use $c \longrightarrow c'$ to denote that $c \xrightarrow{t} c'$ for some $t \in T$.

5 Safety Properties

In this section, we introduce an ordering on configurations, and use it to define the safety problem. Given a parameterized system $\mathcal{P} = (Q, X, T)$, we assume that, prior to starting the execution of the system, each process is in an (identical) *initial* process state $u_{init} = (q_{init}, v_{init})$. In the induced transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$, we use *Init* to denote the set of *initial* configurations, i.e., configurations of the form $[u_{init}, \dots, u_{init}]$. Notice that this set is infinite.

We define an ordering on configurations as follows. Consider two configurations, $c = [u_1 \cdot \dots \cdot u_m]$ and $c' = [u'_1 \cdot \dots \cdot u'_n]$, where $u_i = (q_i, v_i)$ for each $i : 1 \le i \le m$, and $u'_i = (q'_i, v'_i)$ for each $i : 1 \le i \le n$. We write $c \preceq c'$ to denote that there is an injection $h : \overline{m} \to \overline{n}$ such that the following four conditions are satisfied for each $i, j : 1 \le i, j \le m$:

1. $q_i = q'_{h(i)}$.
2. $v_i(x) = \mathit{true}$ iff $v'_{h(i)}(x) = \mathit{true}$ for each $x \in X_{\mathcal{B}}$.
3. $v_i(x) = v_j(y)$ iff $v'_{h(i)}(x) = v'_{h(j)}(y)$, for each $x, y \in X_{\mathcal{N}}$.
4. $v_i(x) <_c v_j(y)$ implies that there is a $d \ge c$ with $v'_{h(i)}(x) <_d v'_{h(j)}(y)$, for each $x, y \in X_{\mathcal{N}}$.

In other words, for each process in c there is a corresponding process in c'. The local states and the values of the Boolean variables coincide in the corresponding processes (Conditions 1 and 2). Regarding the numerical variables, the ordering preserves equality of variables (Condition 3), while gaps between variables in c' are larger than gaps between the corresponding variables in c (Condition 4).

A set of configurations $D \subseteq C$ is *upward closed* (with respect to the ordering $\preceq$) if $c \in D$ and $c \preceq c'$ implies $c' \in D$. For sets of configurations $D, D' \subseteq C$ we use $D \longrightarrow D'$ to denote that there are $c \in D$ and $c' \in D'$ with $c \longrightarrow c'$.

The *coverability problem* for parameterized systems is defined as follows:

PAR-COV
Instance
– A parameterized system $\mathcal{P} = (Q, X, T)$.
– An upward closed set C_F of configurations.

Question *Init* $\xrightarrow{*} C_F$?

It can be shown, using standard techniques (see e.g. [22]), that checking safety properties (expressed as regular languages) can be translated into instances of the coverability problem. Therefore, checking safety properties amounts to solving PAR-COV (i.e., to the reachability of upward closed sets).

6 Approximation

In this section, we introduce an over-approximation of the transition relation of a parameterized system. The aim of the over-approximations is to derive a new transition system which is *monotonic* with respect to the ordering $\preceq$ defined on configurations in Section 5. The only transitions which do not preserve monotonicity are those involving universal global conditions. Therefore, the approximate transition system modifies the behavior of universal quantifiers in such a manner that monotonicity is maintained. Roughly speaking, in the new semantics, we remove all processes in the configuration which violate the condition of the universal quantifier. Below we describe how this is done.

In Section 4, we mentioned that each parameterized system $\mathcal{P} = (Q, X, T)$ induces a transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$. A parameterized system $\mathcal{P}$ also induces an *approximate* transition system $\mathcal{A}(\mathcal{P}) = (C, \rightsquigarrow)$; the set C of configurations is identical to the one in $\mathcal{T}(\mathcal{P})$. We define $\rightsquigarrow = (\longrightarrow \cup \rightsquigarrow_1)$, where $\longrightarrow$ is defined in Section 4, and $\rightsquigarrow_1$ (which reflects the approximation of universal quantifiers) is defined as follows. For a configuration c, a formula $\theta \in \mathbb{F}(\texttt{self}\cdot X \cup \texttt{other}\cdot Y \cup \texttt{self}\cdot X^{next})$, and process states u, u', we use $c \ominus (\theta, u, u')$ to denote the configuration derived from c by deleting all process states u_1 such that $(u, u', u_1) \not\models \theta$. To explain this operation intuitively, we recall that a universal global condition requires that the current and next states of the current process (described by u resp. u') together with the state of each other process (described by u_1) should satisfy the formula θ. The operation then removes from c each process whose state u_1 does not comply with this condition.

Consider two configurations $c = c_1 \bullet u \bullet c_2$ and $c' = c'_1 \bullet u' \bullet c'_2$, where $u = (q, v)$ and $u' = (q', v')$. Let t be a transition rule of the form of (1), such that θ is a universal global condition of the form of (2). We write $c \overset{t}{\rightsquigarrow}_1 c'$ to denote that $c'_1 = c_1 \ominus (\theta_1, u, u')$ and $c'_2 = c_2 \ominus (\theta_1, u, u')$. We use $c \rightsquigarrow c'$ to denote that $c \overset{t}{\rightsquigarrow} c'$ for some $t \in T$; and use $\overset{*}{\rightsquigarrow}$ to denote the reflexive transitive closure of $\rightsquigarrow$.

Lemma 1. *The approximate transition system $(C, \rightsquigarrow)$ is monotonic with respect to $\preceq$.*

We define the coverability problem for the approximate system as follows.

<u>APRX-PAR-COV</u>

Instance

- A parameterized system $\mathcal{P} = (Q, X, T)$.
- An upward closed set C_F of configurations.

Question *Init* $\overset{*}{\rightsquigarrow} C_F$?

Since $\longrightarrow \subseteq \rightsquigarrow$, a negative answer to APRX-PAR-COV implies a negative answer to PAR-COV.

7 Backward Reachability Analysis

In this section, we present a scheme based on backward reachability analysis for solving APRX-PAR-COV. For the rest of this section, we fix an approximate transition system $\mathcal{A}(\mathcal{P}) = (C, \rightsquigarrow)$.

Constraints The scheme operates on *constraints* which we use as a symbolic representation for sets of configurations. A constraint ϕ denotes an upward closed set $[\![\phi]\!] \subseteq C$ of configurations. The constraint ϕ represents minimal conditions on configurations. More precisely, ϕ specifies a minimum number of processes which should be in the configuration, and then imposes certain conditions on these processes. The conditions are formulated as specifications of the control states of the processes, and as restrictions on values of the local variables. The restrictions on values are stated as combinations of Boolean formulas (on the Boolean variables) and gap formulas (on the numerical variables). A configuration c which satisfies ϕ should have at least the number of processes specified by ϕ. Furthermore, the local states and the values of the local variables should satisfy the conditions imposed by ϕ. In such a case, c may have any number of additional processes (whose local states and local variables are then irrelevant for the satisfiability of ϕ by c). Notice that this definition implies that the interpretation $[\![\phi]\!]$ of a constraint ϕ is upward closed (a fact proved in Lemma 2). Below, we define the notion of a constraint formally.

For each natural number $i \in \mathcal{N}$ we make a copy Y^i such that $x^i \in Y^i$ if $x \in Y$. A *constraint* ϕ is a pair (m, ψ), where $m \in \mathcal{N}$ is a natural number, and $\psi \in \mathbb{F}(Y^1 \cup Y^2 \cup \cdots \cup Y^m)$. Intuitively, a configuration satisfying ϕ should contain at least m processes (indexed by $1, \ldots, m$). The constraint ϕ uses the elements of the set Y^i to refer to the local states and variables of process i. The values of these states and variables are constrained by the formula ψ. Formally, consider a configuration $c = [u_1, u_2, \ldots, u_n]$ and a constraint $\phi = (m, \psi)$. Let $h : \overline{m} \mapsto \overline{n}$ be an injection. We write $c \models_h \phi$ to denote the validity of the formula $\psi[\rho]$ where $\rho := \{x^i \leftarrow u_{h(i)}(x) | \; x \in Y \text{ and } 1 \leq i \leq m\}$. In other words, there should be at least m processes inside c whose local states and variables have values which satisfy ψ. We write $c \models \phi$ to denote that $c \models_h \phi$ for some h; and define $[\![\phi]\!] = \{c | \; \phi \models c\}$. For a (finite) set of constraints Φ, we define $[\![\Phi]\!] = \bigcup_{\phi \in \Phi} [\![\phi]\!]$. The following lemma follows from the definitions.

Lemma 2. *For each constraint ϕ, the set $[\![\phi]\!]$ is upward closed. Conversely, for any upward closed set U of configurations there is a finite set of constraints Φ such that $U = [\![\Phi]\!]$.*

This means that the set C_F in the definition of APRX-PAR-COV can be represented by a finite set Φ_F of constraints. The coverability question can then be answered by checking whether $Init \xrightarrow{*} [\![\Phi_F]\!]$.

Entailment and Predecessors To define our scheme we will use two operations on constraints; namely *entailment*, and *computing predecessors*, defined below. We define an *entailment relation* $\sqsubseteq$ on constraints, where $\phi_1 \sqsubseteq \phi_2$ iff $[\![\phi_2]\!] \subseteq [\![\phi_1]\!]$. For sets Φ_1, Φ_2 of constraints, abusing notation, we let $\Phi_1 \sqsubseteq \Phi_2$ denote that for each $\phi_2 \in \Phi_2$ there is a $\phi_1 \in \Phi_1$ with $\phi_1 \sqsubseteq \phi_2$. Observe that $\Phi_1 \sqsubseteq \Phi_2$ implies that $[\![\Phi_2]\!] \subseteq [\![\Phi_1]\!]$.

For a constraint ϕ, we let $Pre(\phi)$ be a finite set of constraints, such that $[\![Pre(\phi)]\!] = \{c | \; \exists c' \in [\![\phi]\!] \, . \, c \rightsquigarrow c'\}$. In other words $Pre(\phi)$ characterizes the set of configurations from which we can reach a configuration in ϕ through the application of a single rule in the approximate transition relation. In the definition of *Pre* we rely on the fact that, in any monotonic transition system, upward-closedness is preserved under the computation of the set of predecessors (see e.g. [1]). From Lemma 2 we know that $[\![\phi]\!]$ is upward closed; and from Lemma 1, we know that $(C, \rightsquigarrow)$ is monotonic. It follows from Lemma 2 that the finite set $Pre(\phi)$, mentioned above, indeed exists. In Section 8, we show that this set is in fact computable. For a set Φ of constraints, we let $Pre(\Phi) = \bigcup_{\phi \in \Phi} Pre(\phi)$.

Scheme Given a finite set Φ_F of constraints, the scheme checks whether $Init \overset{*}{\Longrightarrow} [\![\Phi_F]\!]$. We perform a backward reachability analysis, generating a sequence $\Phi_0 \sqsupseteq \Phi_1 \sqsupseteq \Phi_2 \sqsupseteq \cdots$ of finite sets of constraints such that $\Phi_0 = \Phi_F$, and $\Phi_{j+1} = \Phi_j \cup Pre(\Phi_j)$. Since $[\![\Phi_0]\!] \subseteq [\![\Phi_1]\!] \subseteq [\![\Phi_2]\!] \subseteq \cdots$, the procedure terminates when we reach a point j where $\Phi_j \sqsubseteq \Phi_{j+1}$. Notice that the termination condition implies that $[\![\Phi_j]\!] = (\bigcup_{0 \le i \le j} [\![\Phi_i]\!])$. Consequently, Φ_j characterizes the set of all predecessors of $[\![\phi_F]\!]$. This means that $Init \xrightarrow{*} [\![\Phi_F]\!]$ iff $(Init \bigcap [\![\Phi_j]\!]) \neq \emptyset$.

Observe that, in order to implement the scheme (i.e., transform it into an algorithm), we need to be able to (i) compute *Pre*; (ii) check for entailment between constraints; and (iii) check for emptiness of $(Init \bigcap [\![\phi]\!]) \neq \emptyset$ for a constraint ϕ.

8 Constraint Operations

In this section, we show how to perform the three operations on constraints which are used in the scheme presented in Section 7. In the rest of the section, we fix a parameterized systems $\mathcal{P} = (Q, X, T)$. Recall that $Y = X \cup Q$.

Entailment Consider two constraints $\phi = (m, \psi)$, and $\phi' = (m', \psi')$. Let $\mathcal{H}(\phi, \phi')$ be the set of injections $h : \overline{m} \mapsto \overline{m'}$. We use ψ^h to denote the formula $\psi\,[\rho]$, where $\rho := \left\{x^i \leftarrow x^{h(i)} | \; x \in Y \text{ and } 1 \le i \le m\right\}$. The following lemma gives a logical characterization which allows the computation of the entailment relation.

Lemma 3. *Given two constraints $\phi = (m, \psi)$, and $\phi' = (m', \psi')$, we have $\phi \sqsubseteq \phi'$ iff*

$$\forall y_1 \cdots y_k. \left(\psi'(y_1, \ldots, y_k) \supset \bigvee_{h \in \mathcal{H}(\phi, \phi')} \psi^h(y_1, \ldots, y_k) \right)$$

Pre The following lemma describes the computation of the function *Pre*. The proof of the lemma can be found in the appendix.

Lemma 4. *For a constraint ϕ, we can compute $Pre(\phi)$ as a finite set of constraints.*

One important aspect of the *Pre* function is that it can potentially increase the size of the constraint (the number of processes inside the constraint). This means that there is no bound a priori on the sizes of constraints which may arise in the reachability analysis scheme.

Intersection with Initial States For a constraint $\phi = (m, \psi)$, we have $(Init \bigcap [\![\phi]\!]) \neq \emptyset$ iff $[u_{init}, \ldots, u_{init}] \models \phi$, where the multiset $[u_{init}, \ldots, u_{init}]$ is of size m.

9 Additional Features

In this section, we add a number of features to the model of Section 2. These features are modelled by generalizing the guards which are allowed in the transitions. For all the new features, we can use the same constraint system as in Section 7; consequently checking entailment and intersection with initial states need not be modified. Also, as shown in the appendix, the definition of the *Pre* operator can be extended to cope with the new classes of guards.

Binary Communication In *binary communication* two processes perform a *rendez-vous*, changing states simultaneously. Such a transition can be encoded by considering a more general form of existential global conditions than the one allowed in Section 3. More precisely we take θ_1 in the definition of an existential global conditions (see (2)), to be a formula in the set

$$\mathbb{F}\left(\texttt{self}{\cdot}X \cup \texttt{other}{\cdot}Y \cup \texttt{self}{\cdot}X^{next} \cup \texttt{other}{\cdot}Y^{next}\right)$$

In other words, the formula θ_1 may also constrain variables in the set $\texttt{other}Y^{next}$. Here, `self` and `other` represent the two processes involved in the rendez-vous. For instance, the transition

$$\left[\, idle \rightarrow busy \;\; \rhd \;\; \exists\, \texttt{other} \neq \texttt{self} \cdot \; \texttt{other}{\cdot}wait \wedge \texttt{other}{\cdot}use' \,\right]$$

represents a rendez-vous between a process in state *idle* and a process in state *wait*. The first moves to *busy* while the second one moves to *use*.

Shared Variables We assume the presence of a finite set X_s of Boolean and numerical *shared variables* that can be read and written by all processes in the system. A transition may both modify and check X_s together with the local variables of the processes. Shared variables can be modeled as special processes. The updating of the value of a shared variable by a process can be modeled as a rendez-vous between the process and the variable.

Broadcast A *broadcast* transition is initiated by a process, called the *initiator*. Together with the initiator, each other process inside the system responds simultaneously changing its local state and variables. We can model broadcast transitions by generalizing universally quantified conditions. The generalization is similar to the case of binary communication, i.e., we allow variables in $\texttt{other}{\cdot}Y^{next}$ to occur in the quantified formula. For instance, the transition

$$\left[idle \rightarrow wait \ \ \triangleright \ \ \forall\, \texttt{other} \neq \texttt{self} \cdot \ \texttt{other}{\cdot}wait \supset \texttt{other}{\cdot}x^{next} = \texttt{self}{\cdot}x \right]$$

models the broadcasting of the value of variable x in the initiator to all processes which are in state *wait*.

10 Experimental Results

We have built two different prototypes that implement our approximated backward reachability procedure, based on an integer resp. a real solver for handling the constraints over the process variables. The results are summarized in Figure 1. For each protocol we give the number of iterations and the time needed for performing the verification. The experiments are performed using a Pentium M 1.6 Ghz with 1G of memory (see the appendix for the details).

Model	**Iterations**		**Time**		**Safe**		**Trace**	
	R	I	R	I	R	I	R	I
Simplified Bakery Alg.	6	6	0.8s	1s	√	√		
Lamport's Bakery Alg.	9	9	2.1s	5s	√	√		
Bogus Bakery	10	6	0.8s	9s			√	√
Ticket Mutex Alg.	9	8	0.3s	3.7s	√	√		
Ricart-Agrawala Distr. Mutex Alg.	11	11	15s	80s	√	√		
Lamport's Distr. Mutex Alg.	21	27	20m	146mn	√	√		

Fig. 1. Experimental results. R and I stand for the real resp. integer solver. Safe and Trace stand for checking safety properties resp. generating a counter-example.

11 Conclusion and Future Work

We have presented a method for approximate reachability analysis of systems which consist of an arbitrary number of processes each of which is infinite-state. Based on the method, we have implemented a prototype and automatically verified several non-trivial mutual exclusion protocols. The Bakery example describes a distributed protocol without atomicity assumptions on the transitions. One direction for future research is to develop a methodology for automatic verification of general classes of parameterized systems with non-atomic global conditions. Furthermore, our algorithm relies on an abstract ordering which can be naturally extended to several different types of data structures. We are currently developing similar algorithms for systems with more complicated topologies such as trees and general forms of graphs.

References

1. P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.
2. P. A. Abdulla and G. Delzanno. On the coverability problem for constrained multiset rewriting. In *Proc. AVIS'06*, 2006.
3. P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Regular model checking without transducers In *Proc. TACAS '07*, 2007. To appear.
4. P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Regular model checking made simple and efficient. In *Proc. CONCUR 2002*, volume 2421 of *LNCS*, 2002.
5. G. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison Wesley, 2000.
6. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In *Proc. CAV 2001*, volume 2102 of *LNCS*, 2001.
7. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc. CAV 2003*, volume 2725 of *LNCS*, 2003.
8. M. Bozzano and G. Delzanno. Beyond parameterized verification. In *Proc. TACAS '02*, volume 2280 of *LNCS*, 2002.
9. T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables. *ACM TOPLAS*, 21(4):747–789, 1999.
10. E. Clarke, M. Talupur, and H. Veith. Environment abstraction for parameterized verification. In *Proc. VMCAI '06*, volume 3855 of *LNCS*, pages 126–141, 2006.
11. G. Delzanno. Automatic verification of cache coherence protocols. In *Proc. CAV 2000*, volume 1855 of *LNCS*, 2000.
12. E. Emerson and K. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proc. LICS' 98*, 1998.
13. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. LICS' 99,* 14th *IEEE Int. Symp. on Logic in Computer Science*, 1999.
14. L. Fribourg and J. Richardson. Symbolic verification with gap-order constraints. In *Proc. LOPSTR'96*, volume 1207 of *LNCS*, 1997.
15. S. M. German and A. P. Sistla. Reasoning about systems with many identical processes. *Journal of the ACM*, 39(3):675–735, 1992.
16. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *TCS*, 256:93–112, 2001.
17. S. K. Lahiri and R. E. Bryant. Indexed predicate discovery for unbounded system verification. In *Proc. CAV 2004*, pages 135–147, 2004.
18. L. Lamport. A new solution of dijkstra's concurrent programming problem. *Commun. ACM*, 17(8):453–455, 1974.
19. P. Revesz. A closed form evaluation for datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.
20. G. Ricart and A. K. Agrawal. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9–17, 1981.
21. E. Sedletsky, A. Pnueli, and M. Ben-Ari. Formal verification of the ricart-agrawala algorithm. In *Proc. FSTTCS'00*, 2000.
22. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86*, June 1986.