Noname manuscript No.
(will be inserted by the editor)

Approximated Parameterized Verification of Infinite-state Processes with Global Conditions

Parosh Aziz Abdulla · Giorgio Delzanno · Ahmed Rezine

Received: date / Accepted: date

Abstract We present a simple and effective approximated backward reachability procedure 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, rendezvous and shared variables. The procedure operates on an over-approximation of the transition system induced by the parameterized system. We verify mutual exclusion for complex protocols such as atomic, non-atomic and distributed versions of Lamport's bakery algorithm.

Keywords Parameterized systems · unbounded processes · over-approximation

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

P. Abdulla
Uppsala University
Tel.: +46-18-4713163
Fax: +46-18-511925
E-mail: parosh@it.uu.se

G. Delzanno
Università di Genova
Tel.: +39-10-3536638
Fax: +39-10-3536699
E-mail: giorgio@disi.unige.it

A. Rezine
Uppsala University
Tel.: +46-18-4713159
Fax: +46-18-511925
E-mail: rezine.ahmed@it.uu.se

to automatic verification of parameterized systems make the restriction that each process is finite-state. However, there are many applications where the behavior 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 [22] 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.

To simplify the presentation, we introduce the class of systems we consider in a stepwise manner. First, we consider a basic model where we only allow transitions with local conditions (only the local state and local variables of the process are conditioned) and global conditions. We describe how to derive the approximate transition relation and how to analyze safety properties for the basic model. Then, we introduce the additional features one by one. This includes using shared variables, broadcast and rendez-vous communication and composition of global conditions. For each new feature, we describe how to extend the approximate transition relation and the reachability algorithm in a corresponding manner.

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 behavior 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 [24]), 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, consider again the universal condition "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". In the approximated semantics, 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 or equal to 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. A significant aspect of the reachability procedure is that the number of copies of variables (both Boolean and numerical) which appear in constraints whose denotations are 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, from the results in [2] 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.

Case-studies In this paper we discuss in detail the application of our approximated parameterized verification method to the analysis of several formulations of Lamport's bakery algorithm. The bakery algorithm [21] is a well-known solution to Dijkstra's critical section problem that works for any number of processes. The algorithm presents three main phases. In the *choosing* phase, a process that is interested in entering the critical section selects a new ticket (a number greater than zero). A new ticket is supposed to have a value strictly greater than the tickets of all other processes. Tickets may be requested concurrently by processes in the system. In the *entry* phase, the process waits until the value of its ticket becomes smaller than the tickets of all other interested processes and then enters the critical section. In the *exit* section, a process releases the critical section and resets its ticket to zero. These phases can be formulated in several different ways depending on the assumptions on the atomicity of the statements used to generate and compare tickets.

In our experiments we automatically verify safety properties for parameterized versions of the following formulations of Lamport's bakery algorithm:

- A version with atomic computation of new tickets and an atomic entry phase.
- A version with race conditions in the computation of new tickets, but with an atomic entry phase.
- A version where both choosing and entry phases are non-atomic. Non-atomic phases are implemented by means of a marking sub-protocol. A process p applies the sub-protocol to test the global condition process-by-process (p marks process q as visited if the condition is verified for p and q).
- A distributed version of bakery, called the ticket algorithm in [7], that makes use of a central monitor for distributing tickets and checking the entry conditions.

We also consider a bogus version of the second formulation of the bakery algorithm (the version with race conditions on the computation of tickets). In this version, the choosing flag used to control race conditions in the original version of the algorithm is simply ignored. For the bogus version, 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.

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. Also the protocols (apart from Ticket) use an order over identifiers to break ties in the entry section. These features cannot be modeled without the use of abstractions in the framework of Regular Model Checking [19, 6, 9, 5].
- In all examples, global conditions are needed to model the communication mechanisms used in the protocols.

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 [19, 6, 9] and counter abstraction methods [14, 18, 15, 16]. 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 [5] 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 [8], with the help of *environment abstraction* for a formulation with atomicity conditions in [13], and using heuristics to discover *indexed predicates* in [20]. We are not aware of other attempts of fully automatic verification of parameterized versions of Lamport's bakery algorithm without atomic conditions.

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 in which the FIFO order of requests is not preserved is verified in [11]. 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. The verification procedure in [11] does not handle parameterized universally quantified global conditions.

In contrast to symbolic methods for finite, a priori fixed collections of processes with local integer variables, e.g., those in [12, 17], 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. By viewing processes as atomic formulas and configurations as multisets of atomic formulas, we can exploit the decidability result in [2] as a termination argument for symbolic backward reachability for a restricted class of parameterized systems in which processes have at most one integer local variable. The symbolic representation and the entailment operation we use here are a natural generalization of those used for monadic atomic formulas in [2]. Termination for this class of parameterized systems follows from the well-quasi ordering of entailment for monadic constraint multiset rewriting systems. Unfortunately, this

property does not hold for most of the examples considered here in which processes have two or more integer local variables. For these systems, termination cannot be guaranteed a priori, however our procedure often terminates in practical cases.

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 extends the basic model to cover features such as shared variables, rendez-vous and broadcast. In Section 10 we give a detailed description of our case studies and report the results of running our prototypes on them. Finally, in Section 11, we give conclusions and directions for future work. In the appendix, we give some proofs.

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 $x \sim_k y$ where $\sim \in \{<, \leq\}$, $x, y \in A_{\mathcal{N}}$, and $k \in \mathcal{N}$. Here $x \sim_k y$ stands for $x + k \sim y$[1]. 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 $\wedge$ and $\vee$ only (gap-order formulas are not closed under negation). 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 $\theta(x, y, a, b)$.

For a set $B = \{x_1, \ldots, x_n\} \subseteq A$, and a formula $\theta \in \mathbb{F}(A)$, we use $\exists B.\ \theta$ to denote the existentially quantified formula $\exists x_1, \cdots x_n.\ \theta$.

In [22], it is shown that gap-order formulas are effectively closed under existential quantification. It is easy to verify that the same property holds for the subset of gap-formulas we consider here. Since Boolean formulas are also effectively closed under existential quantification, it follows that, for any finite set B with $B \subseteq A$, and any formula $\theta \in \mathbb{F}(A)$, we can compute a formula $\theta' \in \mathbb{F}(A)$ which is equivalent to $\exists B.\ \theta$.

Sometimes, we perform substitutions on logical formulas. A *substitution* is a set $\{x_1 \leftarrow y_1, \ldots, x_n \leftarrow y_n\}$ of pairs, where x_i and y_i are variables of the same type, for

[1] $\mathbb{G}(A_{\mathcal{N}})$ is a subset of the gap formulas defined in [22].

each $i : 1 \leq i \leq n$. Here, we assume that $x_i \neq x_j$ 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 $y_1, y_2, \ldots, y_n$ 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 we have $\theta = (x_1 < x_2) \wedge (x_3 < x_4)$ then $\theta[x_1 \leftarrow y_2, x_4 \leftarrow y_3][x_2 \leftarrow y] = (y_2 < y) \wedge (x_3 < y_3)$.

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

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

where $q, q' \in Q$ and θ is either a *local* or a *global condition*. Intuitively, the process which takes 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 if and only if 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.

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. 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. We do that by introducing, for each $x \in Y$, two new variables self$\cdot x$ and other$\cdot x$. `self` denotes the process that is about to perform the transition and `other` any other process different from `self`.

We define the sets self$\cdot Y = \{\text{self}\cdot x | x \in Y\}$ and other$\cdot Y = \{\text{other}\cdot x | x \in Y\}$. The sets self$\cdot X$, 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 $\theta_1 \in \mathbb{F}\left(\text{self}{\cdot}X \cup \text{other}{\cdot}Y \cup \text{self}{\cdot}X^{next}\right)$. In other words, the formula checks the local variables of the process which is about to make the transition (through self$\cdot X$), and the local states and variables of the other processes (through other$\cdot Y$). It also specifies how the local variables of the process in transition are updated (through self$\cdot X^{next}$). 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 $\left(D, \Longrightarrow\right)$, 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 $\mathcal{T}(\mathcal{P}) = \left(C, \longrightarrow\right)$ as follows. C is the set of configurations. 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 variables' types. 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 may regard members of 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 $q' \in Q \setminus \{q\}$. 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. 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 θ in the set $\mathbb{F}\left(\text{self}{\cdot}X \cup \text{other}{\cdot}Y \cup \text{self}{\cdot}X^{next}\right)$ (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 $\rho := \{\text{self}{\cdot}x \leftarrow v(x) \mid x \in X\}$, $\rho' := \{\text{self}{\cdot}x^{next} \leftarrow v'(x) \mid x \in X\}$, and $\rho_1 := \{\text{other}{\cdot}x \leftarrow u_1(x) \mid x \in Y\}$. 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 $c' = c_1 \bullet [u'] \bullet c_2$ where $u = (q, v)$ and $u' = (q', v')$. 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 $t \in T$.

Example 1 Consider a parameterized system $\mathcal{P} = (Q, X, T)$, where $Q = \{idle, wait, use\}$, $X = \{num \in \mathcal{N}\}$, and $T = \{t, t'\}$. Both transitions t and t' are defined below.

Consider a configuration $c = [u_1, u_2, u_3]$. In this configuration, the process states u_1, u_2 and u_3 are associated with three processes, say p_1, p_2 and p_3. We suppose $u_i = (idle, v_i)$ for each $i : 1 \leq i \leq 3$, $v_1(num) = 1$, $v_2(num) = 10$, and $v_3(num) = 11$. In other words, all three processes are in state *idle*, the variable *num* belonging to process p_1 is mapped to 1, the one belonging to process p_2 is mapped to 10, and the one belonging to process p_3 is mapped to 11.

Let t be the local transition defined as

$$t = \left[\, idle \rightarrow wait \;\; \triangleright \;\; num' > num \,\right]$$

Define the process state u_1' as $(wait, v_1')$, where the local variable state v_1' sends the variable *num* to 4. Since the condition $v_1'(num) > v_1(num)$ is satisfied, we can associate u_1' with the process p_1, and state that $c \xrightarrow{t} c'$ where $c' = [u_1', u_2, u_3]$.

Now let t' be the universal global transition defined as

$$t' = \left[\, wait \rightarrow use \;\; \triangleright \;\; \forall\, \texttt{other} \neq \texttt{self} \cdot \; (\texttt{self}{\cdot}num < \texttt{other}{\cdot}num) \,\right]$$

In order for process p_1 to take the transition t', it needs to be in state *wait*. Furthermore, the value associated with its variable *num* should be smaller than the value associated with the *num* variable belonging to each of the other processes. Observe that this is the case in the configuration $c' = [u_1', u_2, u_3]$. In case process p_1 takes the transition t', it changes state to *use* and updates its variable *num* to some arbitrary numerical value, for instance 1. We can therefore write that $c' \xrightarrow{t'} c''$ where $c'' = [u_1'', u_2, u_3]$ and $u_1'' = (use, v_1)$.

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 $\mathcal{T}(\mathcal{P}) = \big(C, \longrightarrow\big)$, we use *Init* to denote the set of *initial* configurations, i.e., configurations of the form $[u_{init}, \ldots, u_{init}]$. Notice that this set is infinite.

We define an ordering on configurations as follows. Consider two configurations, $c = [u_1, \ldots, u_m]$ and $c' = [u'_1, \ldots, u'_n]$, where $u_i = (q_i, v_i)$ for each $i : 1 \leq i \leq m$, and $u'_i = (q'_i, v'_i)$ for each $i : 1 \leq i \leq n$. 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 $i, j : 1 \leq i, j \leq m$:

1. $q_i = q'_{h(i)}$.
2. $v_i(x) = v'_{h(i)}(x)$ 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. $v_i(x) <_k v_j(y)$ implies that there is a $l \geq k$ with $v'_{h(i)}(x) <_l v'_{h(j)}(y)$, 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 at least as large as the 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 $\mathit{Init} \xrightarrow{*} C_F$?

It can be shown, using standard techniques (see e.g. [24]), 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-approximation 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 $\theta \in \mathbb{F}\left(\texttt{self}{\cdot}X \cup \texttt{other}{\cdot}Y \cup \texttt{self}{\cdot}X^{next}\right)$, 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 $\rightsquigarrow$.

Example 2 Consider a parameterized system $\mathcal{P} = (Q, X, T)$, where $Q = \{idle, wait, use\}$ and $X = \{num \in \mathcal{N}\}$.

Let t be the transition defined as

$$t = \left[\, idle \rightarrow use \;\; \triangleright \;\; \forall\, \texttt{other} \neq \texttt{self} \cdot \;\; (\texttt{self}{\cdot}num < \texttt{other}{\cdot}num) \,\right]$$

Consider a configuration $c = [u_1, u_2, u_3]$, where the process states u_1, u_2 and u_3 are associated with processes p_1, p_2 and p_3. We suppose $u_1 = (idle, v_i)$ for $i : 1 \leq i \leq 3$, and $v_1(num) = 3$, $v_2(num) = 1$, $v_3(num) = 5$.

Process p_1 cannot perform the transition t from *idle* to *use* since process p_2 has the minimal value for *num*. Thus, the test of the global condition fails.

In the approximated transition relation, the process p_1 can take the transition, for example $c \overset{t}{\rightsquigarrow} c'$ with $c' = [u'_1, u_3]$ and $u'_1 = (use, v_1)$. This is because we delete all processes that violate the global condition (p_2 here). Notice that the resulting configuration has a size that is smaller than the one of the configuration c.

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

Proof A detailed proof of this lemma is given in [23].

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

APRX-PAR-COV
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 $\mathcal{A}(\mathcal{P}) = (C, \rightsquigarrow)$.

7.1 Constraints

The scheme operates on *constraints* which we use as a symbolic representation for sets of configurations. 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 $\rho := \left\{ x^i \leftarrow u_{h(i)}(x) \middle|\ x \in Y \text{ and } 1 \leq i \leq m \right\}$. 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 $[\![\phi]\!] = \{c \mid c \models \phi\}$. For a (finite) set of constraints $\boldsymbol{\Phi}$, we define $[\![\boldsymbol{\Phi}]\!] = \bigcup_{\phi \in \boldsymbol{\Phi}} [\![\phi]\!]$. The following lemma follows from the definitions.

Lemma 2 *For each constraint ϕ, the set $[\![\phi]\!]$ is upward closed.*

Proof A detailed proof of this lemma is given in [23].

In all the examples we consider, the set C_F in the definition of APRX-PAR-COV can be represented by a finite set $\boldsymbol{\Phi}_F$ of constraints.The coverability question can then be answered by checking whether $\mathit{Init} \xrightarrow{*} [\![\boldsymbol{\Phi}_F]\!]$.

7.2 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 $\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2$ of constraints, abusing notation, we let $\boldsymbol{\Phi}_1 \sqsubseteq \boldsymbol{\Phi}_2$ denote that for each $\phi_2 \in \boldsymbol{\Phi}_2$ there is a $\phi_1 \in \boldsymbol{\Phi}_1$ with $\phi_1 \sqsubseteq \phi_2$. Observe that $\boldsymbol{\Phi}_1 \sqsubseteq \boldsymbol{\Phi}_2$ implies that $[\![\boldsymbol{\Phi}_2]\!] \subseteq [\![\boldsymbol{\Phi}_1]\!]$.

For a constraint ϕ, we let $\mathit{Pre}(\phi)$ be a set of constraints, such that $[\![\mathit{Pre}(\phi)]\!] = \left\{ c \middle|\ \exists c' \in [\![\phi]\!] \,.\, c \rightsquigarrow c' \right\}$. In other words $\mathit{Pre}(\phi)$ 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; by Lemma 1, $(C, \rightsquigarrow)$ is monotonic, we therefore know that $[\![Pre(\phi)]\!]$ is upward closed. In fact, we show in Section 8 that $Pre(\Phi)$ is finite and computable. For a set Φ of constraints, we let $Pre(\Phi) = \bigcup_{\phi \in \Phi} Pre(\phi)$.

7.3 Scheme

Given a finite set Φ_F of constraints, the scheme checks whether $Init \stackrel{*}{\Longrightarrow} [\![\Phi_F]\!]$. 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 $[\![\Phi_j]\!] = (\bigcup_{0 \leq i \leq j} [\![\Phi_i]\!])$. Consequently, Φ_j characterizes the set of all predecessors of $[\![\phi_F]\!]$. This means that $Init \stackrel{*}{\longrightarrow} [\![\Phi_F]\!]$ iff $\left(Init \bigcap [\![[\![\Phi_j]\!]]\!]\right) \neq \emptyset$.

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.

7.4 Termination

The termination of the fixpoint computation is not guaranteed in general. However, by using the theoretical analysis of monadic multiset rewriting systems with gap-order constraints in [2], we can show that termination is guaranteed for processes with at most one integer local variable (and any number of local Boolean variables).

The intuition however is as follows. In the framework of [2] a configuration of a parameterized system is modeled as a multiset of atomic formulas. An atomic formula $p(d_1, \ldots, d_n)$ represents a process in state p and values $d_1, \ldots, d_n$ for its n local variables. An atomic formula (predicate) with a single parameter is called monadic. Thus, for monadic predicates, an atomic formula $p(d)$ can be viewed as a process with state p and value d (a natural number) for its unique integer local variable.

The symbolic representation and the corresponding entailment relation used in the present paper are a natural generalization (to the case of n-ary predicates) of the operations on multisets of monadic atomic formulas in [2].

Furthermore, since in our definition of parameterized systems we only allows gap-order constraints as conditions, we can easily express transitions of parameterized systems with one integer local variable as rules in monadic constrained multiset rewriting with gap-order constraints. For instance, rendez-vous is a multiset rewriting rule that simultaneously rewrite two atomic formulas into two new ones thar correspond to the next state of the communicating processes. We can then exploit the well-quasi ordering of entailment for symbolic configurations of monadic constrained multiset rewriting systems in [2] as a termination argument for symbolic backward reachability of parameterized systems with (at most) one integer local variable.

We omit a detailed proof of this encoding here since most of the examples considered in the present paper are outside the decidable class.

8 Constraint Operations

In this section, we show how to perform the three operations on constraints used in the scheme presented in Section 7, namely computing *Pre*, entailment, and intersection with initial states. In the rest of the section, we fix a parameterized systems $\mathcal{P} = (Q, X, T)$. Recall that $Y = X \cup Q$.

8.1 Computing Pre

We show, for a constraint ϕ', how to compute the set $Pre(\phi')$. We will compute $Pre(\phi')$ as $\bigcup_{t \in T} Pre_t(\phi')$, where $[\![Pre_t(\phi')]\!] = \left\{ c \middle| \exists . c' \in [\![\phi']\!].\ c \overset{t}{\rightsquigarrow} c' \right\}$. Consider a transition rule $t : \left[q \to q' \ \rhd \ \theta \right]$. Consider also a constraint $\phi' = \left(m, \psi'\right)$ (where $\psi' \in \mathbb{F}\left(Y^1 \cup \ldots \cup Y^m\right)$). Below, we describe how to compute $Pre_t(\phi')$ as a finite set of constraints. The definition has several cases depending on the condition θ (local, existential, or global condition) as follows.

Local Conditions. If θ is a local condition. By definition, we know that $\theta \in \mathbb{F}\left(X \cup X^{next}\right)$. First, we recall that t gives rise to a set of transitions, where a single process changes state while the other ones remain passive. To capture this property, we will define the formula $\theta^{\bullet}$ as:

$$\theta \wedge q \wedge \left(\bigwedge_{p \neq q} \neg p \right) \wedge q'^{next} \wedge \left(\bigwedge_{p' \neq q'} \neg p'^{next} \right)$$

Intuitively, the first conjunct θ corresponds to the condition of the transition rule. The second and third conjuncts encode the fact that the control state of the configuration before performing the transition is equal to q, while the last two conjuncts encode that the control state after performing the transition is equal to q'.

We define $Pre_t(\phi')$ to be the smallest set containing the constraints:

- A constraint $\phi_i = (m, \psi_i)$ where ψ_i is given by[2]

$$\left(\exists Y^i . \left(\theta^{\bullet}\left[\rho_1^i\right] \wedge \psi'\right)\right)\left[\rho_2^i\right]$$

  for each $i : 1 \leq i \leq m$. The substitutions are defined by

$$\rho_1^i = \left\{ x^{next} \leftarrow x^i \middle| \ x^i \in Y^i \right\} \qquad \rho_2^i = \left\{ x \leftarrow x^i \middle| \ x^i \in Y^i \right\}$$

  The constraint ϕ_i corresponds to the case where the process performing the transition is part of the definition of the constraint ϕ' (namely, it is the process represented by index i in ϕ'). The role of the substitution ρ_1^i is to guarantee that we match each variable $x^i \in Y^i$ which occurs in ψ' with the next-value variable x^{next} in t. After eliminating the renamed next-value variables by projection (existential quantification), we rename each x to the corresponding x^i with the substitution ρ_2^i.

[2] Recall that the existential quantifier can be effectively eliminated (as described in Section 2).

– A constraint $\phi = (m+1, \psi)$ where ψ is given by

$$\left(\exists Y^{m+1}. \left(\theta^{\bullet}\left[\rho_1^{m+1}\right] \wedge \psi'\right)\right)\left[\rho_2^{m+1}\right]$$

where

$$\rho_1 = \left\{x^{next} \leftarrow x^{m+1} \middle|\, x^{m+1} \in Y^{m+1}\right\} \qquad \rho_2 = \left\{x \leftarrow x^{m+1} \middle|\, x^{m+1} \in Y^{m+1}\right\}$$

Here, the process taking the transition is not part of ϕ'. The process is therefore represented by an index $m+1$ which is outside the range m of ϕ'.

The constraint $(m+1, \psi)$ is needed in the definition of Pre_t in order to ensure that the denotation of $Pre_t(\phi')$ contains all predecessors of the denotation of ϕ'. Notice however that, when computing the transitive closure Pre_t^* of Pre_t the constraint $(m+1, \psi)$ becomes redundant. Indeed, it does not add new information with respect to that represented by the original constraint ϕ'.

Example 3 (Pre of a local rule) Consider a parameterized system with states $\{p, q\}$, a local Boolean variable f and a local rule $t : \left[\, p \to q \;\rhd\; \theta \,\right]$ where θ equals $\left(f^{next} = f\right)$. Now consider the constraint $\phi' = \left(1, q^1 \wedge f^1\right)$. If the transition t is performed by the process represented in ϕ', we obtain the constraint $\phi_1 = \left(1, p^1 \wedge f^1\right)$. If the process that performed the transition is not the one represented in the constraint ϕ', we obtain the constraint $\phi_2 = (2, q^1 \wedge f^1 \wedge p^2)$ with size two. The latter constraint is derived by introducing in ϕ' a process with state p and no constraints on f. Notice that the denotation of ϕ_1 is not a subset of the one of ϕ_2, and vice versa. We need both constraints to symbolically represent the predecessors of the denotation of ϕ'. Also, notice that $[\![\phi_1]\!] \not\subseteq [\![[\![\phi']\!]]\!]$, whereas $[\![\phi_2]\!] \subseteq [\![[\![\phi']\!]]\!]$, i.e., the configurations denoted by ϕ_2 are already denoted by the set $\left\{\phi'\right\} \cup Pre_t(\phi')$.

Existential Conditions. If θ is of the form $\exists \mathrm{other} \neq \mathrm{self} \cdot \theta_1$, where the condition θ_1 is in $\mathbb{F}\left(\mathrm{self}{\cdot}X \cup \mathrm{self}{\cdot}X^{next} \cup \mathrm{other}{\cdot}Y\right)$. The main difference compared to the case of local conditions is that we have to consider two processes. More precisely, in addition to the process (with index i) which performs the transition, we have also to consider a witness process (with index j) which enables the transition. The definition of $Pre_t(\phi')$ consists of several cases corresponding to whether or not the processes i and j are parts of ϕ'. We define $Pre_t(\phi')$ to be the smallest set containing the following constraints:

– A constraint $\phi_{i,j} = \left(m, \psi_{ij}\right)$ where $\psi_{i,j}$ is given by

$$\left(\exists Y^{i}. \left(\theta_1^{\bullet}\left[\rho_1^{i}\right]\left[\rho_2^{j}\right] \wedge \psi'\right)\right)\left[\rho_3^{i}\right]$$

for each $i : 1 \leq i \neq j \leq m$. The constraint ϕ_{ij} corresponds to the case where both the process (with index i) performing the transition, and the witness process (with index j) are parts of the definition of the constraint ϕ'. The formula $\theta_1^{\bullet}$ is defined in a similar manner to $\theta^{\bullet}$ above (with the difference that θ is replaced by θ_1). The substitutions are defined by

$$\rho_1^i = \left\{\mathrm{self}{\cdot}x^{next} \leftarrow x^i \middle|\, x^i \in X^i\right\} \cup \left\{q^{next} \leftarrow q^i \middle|\, q \in Q\right\}$$
$$\rho_2^j = \left\{\mathrm{other}{\cdot}x \leftarrow x^j \middle|\, x^j \in Y^j\right\}$$
$$\rho_3^i = \left\{\mathrm{self}{\cdot}x \leftarrow x^i \middle|\, x^i \in X^i\right\} \cup \left\{q \leftarrow q^i \middle|\, q \in Q\right\}$$

The substitutions ρ_1^i and ρ_3^i here respectively play the same role as the substitutions ρ_1^i and ρ_2^i in the case of local conditions (taking into consideration that local variables are preceded by the prefix self in the case of global conditions and that θ_1 is in $\mathbb{F}\left(\text{self}{\cdot}X \cup \text{self}{\cdot}X^{next} \cup \text{other}{\cdot}Y\right)$,i.e., $\text{self}{\cdot}q$ cannot occur in θ_1 for $q \in Q \cup Q^{next}$). The substitution ρ_2^j encodes the conditions imposed on the witness process (with index j).

– A constraint $\phi_{i,m+1} = \left(m+1, \psi_{i,m+1}\right)$ where $\psi_{i,m+1}$ is given by

$$\left(\exists Y^i.\left(\theta_1^{\bullet}\left[\rho_1^i\right]\left[\rho_2^{m+1}\right] \wedge \psi'\right)\right)\left[\rho_3^i\right]$$

for each $i : 1 \leq i \leq m$. The substitutions ρ_1^i and ρ_3^i are identical to the previous case. The substitution ρ_2^{m+1} is defined by

$$\rho_2^{m+1} = \left\{\text{other}{\cdot}x \leftarrow x^{m+1} \mid x^{m+1} \in Y^{m+1}\right\}$$

Here, the process (with index i) taking the transition is part of ϕ', while the witness process is not. The witness process is therefore represented by an index $m+1$ which is outside the range m of ϕ'.
Notice that this case gives rise to a constraint ϕ_i whose size is larger than the size of the original constraint ϕ'. Furthermore, in contrast to the case of local conditions, the new constraint $\phi_{i,m+1}$ is not necessarily subsumed by ϕ'. This makes the sizes of constraints which arise in the reachability analysis unbounded in general.

– We have two cases where the process taking the transition is not part of ϕ', and where the witness process may or may not be part of ϕ'. In a similar manner to the case of local conditions, the generated constraints are all subsumed by ϕ' and can therefore be safely discarded in the reachability analysis. The details of these cases are given in the appendix.

Example 4 (*Pre* of an existential rule) Add to the parameterized system of example 3 the rule $t : \left[\, p \rightarrow q \;\rhd\; \exists\, \text{other} \neq \text{self} \cdot \theta_1 \,\right]$ where θ_1 is $\left(\text{self}{\cdot}f^{next} = \text{self}{\cdot}f \wedge \text{other}{\cdot}f\right)$. Consider the constraint $\phi' = \left(2, q^1 \wedge \neg f^1 \wedge p^2\right)$. If both the process that performed the transition t and the witness process are represented in ϕ', we get the constraint $\phi_{12} = \left(2, p^1 \wedge \neg f^1 \wedge p^2 \wedge f^2\right)$. If the witness process is not represented in ϕ' we insert it, and obtain the constraint $\phi_{13} = \left(3, p^1 \wedge \neg f^1 \wedge p^2 \wedge f^3\right)$. If the witness process is represented in ϕ', but not the process that performed the transition, we obtain the constraint $\phi_{31} = \left(3, q^1 \wedge \neg f^1 \wedge p^2 \wedge f^2 \wedge p^3\right)$. Finally, if both the process that performed the transition and the witness process are not represented in ϕ', we obtain the constraint $\phi_{34} = \left(4, q^1 \wedge \neg f^1 \wedge p^2 \wedge p^3 \wedge f^4\right)$. Observe that $[\![\phi_{31}]\!] \subseteq [\![[\![\phi']\!]]$, and $[\![\phi_{34}]\!] \subseteq [\![[\![\phi']\!]]$. Observe also that $[\![\phi_{13}]\!] \not\subseteq [\![[\![\phi']\!]]$, meaning that ϕ_{13} has to be added during the fixpoint computation.

Universal Conditions. If θ is of the form $\forall\, \text{other} \neq \text{self} \cdot \theta_1$ where the condition θ_1 is in $\mathbb{F}\left(\text{self}{\cdot}X \cup \text{self}{\cdot}X^{next} \cup \text{other}{\cdot}Y\right)$. We define $Pre_t(\phi')$ to be the smallest set containing the constraints:

– A constraint $\phi_i = (m, \psi_i)$ where ψ_i is given by

$$\left(\exists Y^i.\ \left(\bigwedge_{1\le j\neq i\le m} \theta_1^{\bullet}\left[\rho_1^i\right]\left[\rho_2^j\right] \wedge \psi'\right)\right)\left[\rho_3^i\right]$$

for each $i : 1 \le i \le m$.
The constraint ϕ_i corresponds to the case where the process (with index i) performing the transition is part of the definition of the constraint ϕ'. The formula $\theta_1^{\bullet}$ and the substitutions ρ_1^i, ρ_2^j, and ρ_3^i are defined like for the case of existential conditions. The difference is that we apply $\theta_1^{\bullet}$ on all the other processes, and not only on one witness (as encoded by the conjunction operator).
– In a similar manner to the local condition case, we have one more constraint where the process taking the transition is not part of ϕ'. The generated constraint is subsumed by ϕ' and can therefore be safely discarded in the reachability analysis. The details of this case is given in the appendix.

Example 5 (*Pre* of a universal rule) Add to the parameterized system of examples 3, 4 the rule $t : \left[\, p \to q \ \rhd\ \forall \text{other} \neq \text{self} \cdot \theta_1 \,\right]$ with $\theta_1 = \left(\text{self}\cdot f^{next} = \text{self}\cdot f \wedge \text{other}\cdot f\right)$. Consider the constraint $\phi' = \left(2, f^1 \wedge p^2\right)$. If the process that performed the transition t is represented in ϕ', we obtain the constraint $\phi_1 = \left(2, p^1 \wedge f^1 \wedge p^2 \wedge f^2\right)$. In the case where the process that performed the transition is not represented in the constraint ϕ', we obtain the constraint $\phi_3 = \left(3, f^1 \wedge p^2 \wedge f^2 \wedge p^3\right)$. Observe that $[\![\phi_3]\!] \subseteq [\![[\![\phi']\!]]\!]$, but that $[\![\phi_1]\!] \not\subseteq [\![[\![\phi']\!]]\!]$.

Lemma 3 *We have* $[\![Pre_t(\phi')]\!] = \left\{ c \mid \exists . c' \in [\![\phi']\!].\ c \overset{t}{\rightsquigarrow} c' \right\}$ *for any transition* t *with a local condition, a global existential condition, or a global universal condition.*

Proof see appendix.

Entailment Consider two constraints $\phi = (m, \psi)$, and $\phi' = \left(m', \psi'\right)$. 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 $\rho := \left\{ x^i \leftarrow x^{h(i)} \mid x \in Y \text{ and } 1 \le i \le m \right\}$. The following lemma gives a logical characterization which allows the computation of the entailment relation.

Lemma 4 *Given two constraints* $\phi = (m, \psi)$, *and* $\phi' = \left(m', \psi'\right)$, *we have* $\phi \sqsubseteq \phi'$ *iff*

$$\forall y_1 \cdots y_k.\ \left(\psi'(y_1, \ldots, y_k) \supset \bigvee_{h \in \mathcal{H}(\phi,\phi')} \psi^h(y_1, \ldots, y_k)\right) \tag{3}$$

Proof see appendix.

Example 6 (Entailment) Consider a parameterized system (Q, X, T) with $Q = \{p, q\}$, and $X = \{x \in \mathcal{N}\}$. We compare the constraints $\phi = (4, \psi)$ and $\phi' = \left(4, \psi'\right)$ with

$$\psi = x^1 \le x^2 \le x^3 \le x^4$$

and

$$\psi' = x^1 \le x^2 \le x^4 \wedge x^1 \le x^3 \le x^4$$

Among all the possible renamings, let us consider $h_1 = \{1 \to 1, 2 \to 2, 3 \to 3, 4 \to 4\}$ and $h_2 = \{1 \to 1, 2 \to 3, 3 \to 2, 4 \to 4\}$. When restricting our attention to renamings h_1 and h_2 we obtain the formula

$$\forall x_1, x_2, x_3, x_4.\ \left(\psi' \supset \begin{pmatrix} x^1 \leq x^2 \leq x^3 \leq x^4 \\ \vee \\ x^1 \leq x^3 \leq x^2 \leq x^4 \end{pmatrix}\right)$$

which is valid. Thus, we have that $\phi \sqsubseteq \phi'$ holds. Let us fix now a configuration $c = [u_1, u_2, u_3, u_4]$. Clearly, if there is an injection h such that $c \models_h \phi$, then the relative order of the values v_2 and v_3 of the variable x for the processes that are mapped to position 2 and 3 in ϕ is either $v_2 \leq v_3$ or $v_3 < v_2$. Thus, from h we can build injections h_1 and h_2 such that either $c \models_{h_2} \phi$ or $c \models_{h_2} \phi$. Thus, entailment corresponds to containment of denotations.

Intersection with Initial States For a constraint $\phi = (m, \psi)$, we have $(\mathit{Init} \bigcap [\![\phi]\!]) \neq \emptyset$ 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 and N-ary 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

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

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

$$\left[\, \mathit{idle} \to \mathit{busy} \ \rhd\ \exists\, \mathtt{other} \neq \mathtt{self} \cdot\ \mathtt{other}{\cdot}\mathit{wait} \wedge \mathtt{other}{\cdot}\mathit{use}^{next} \,\right]$$

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*.

We can naturally generalize the binary communication scheme by introducing the use of existentially quantified conditions of the form

$$\exists\, \mathtt{other}_1, \ldots, \mathtt{other}_n \neq \mathtt{self} \cdot \theta_1$$

where $\texttt{other}\cdot_1, \ldots, \texttt{other}\cdot_n$ represent n distinct processes (all different from self) and θ_1 is defined over variables and next-variables associated to the qualifiers $\texttt{self}$, $\texttt{other}\cdot_1, \ldots, \texttt{other}\cdot_n$, i.e. θ_1 is in the set

$$\mathbb{F}\left(\texttt{self}\cdot X \cup \texttt{self}\cdot X^{next} \cup \bigcup_{i=1}^{n} \texttt{other}_i \cdot Y \cup \texttt{other}_i \cdot Y^{next}\right)$$

This kind of quantification can be used to specify synchronization between $n+1$ distinct processes. For instance, the transition

$$\left[busy \rightarrow busy_p \;\; \rhd \;\; \left(\begin{array}{l} \exists\, \texttt{other}_1, \texttt{other}_2 \neq \texttt{self}\cdot \\ \quad \texttt{other}_1 \cdot wait_h \wedge \texttt{other}_1 \cdot use_h^{next} \wedge \\ \quad \texttt{other}_2 \cdot use_l \wedge \texttt{other}_2 \cdot preempted_l^{next} \end{array} \right) \right]$$

represents a rendez-vous between a process in state *busy* and two processes, one in state $wait_h$ (h=high priority) and the other one in state use_l (l=low priority). The first one moves to $busy_p$ (p=preemption), the second moves to use_h and the third one moves to $preempted_l$.

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

$$\left[idle \rightarrow wait \;\; \rhd \;\; \left(\begin{array}{l} \forall\, \texttt{other} \neq \texttt{self}\cdot \\ \quad (\texttt{other}\cdot wait \wedge \texttt{other}\cdot x^{next} = \texttt{self}\cdot x) \vee \\ \quad (\neg \texttt{other}\cdot wait \wedge \texttt{other}\cdot x^{next} = \texttt{other}\cdot x) \end{array} \right) \right]$$

models the broadcasting of the value of variable x in the initiator to all processes which are in state *wait*. In the rest of the paper we often omit to mention assignments of the form $x^{next} = x$ (i.e. variables remain unchanged unless we specify the update in the condition).

Composite Conditions As a final possible extension, we allow conditions obtained as conjunctions and disjunctions of local and global conditions. As an example, we may consider conditions like

$$(\exists\, \texttt{other} \neq \texttt{self}\cdot\ \texttt{self}\cdot x^{next} = \texttt{other}\cdot x) \wedge (\forall\, \texttt{other} \neq \texttt{self}\cdot\ \texttt{other}\cdot x \geq \texttt{self}\cdot x^{next})$$

to specify that the local variable x is updated with the minimum of the current values of variable x in all other processes.

```
var number: shared array [0..n-1] of integer;

Process P[i] :=
  loop forever
    number[i] := max(number[0],number[1],...,number[n-1])+1;
    for j := 0 to n-1 do begin
      while number[j] <> 0 and (number[j], j) << (number[i],i) do skip;
    end;
    (* critical section *)
    number[i] := 0;
    (* non-critical section *);
```

Fig. 1 Pseudo-code for the bakery mut-ex algorithm with atomic assignment. The relation $(a,b) << (c,d)$ holds iff $a < c$ or ($a = c$ and $b < d$).

10 Case-studies

In this section we discuss the application of our verification method. For this purpose, we consider four different formulations of Lamport's bakery algorithm. The algorithm is a well-known solution to the critical section problem where an arbitrary and finite number of processes compete for a shared resource. A simple version of the algorithm can be written in the pseudo-code of Figure 1. The rationale behind the algorithm is as follows. Each process has a local variable *number* in which it stores a ticket. Initially *number* equals zero. When a process is interested in entering the critical section, it sets its *number* to a value strictly greater than the tickets (i.e., value of *number*) of all other processes in the system. We will refer to this step as the choosing phase. A possible way to implement this phase is by choosing a value strictly larger than the maximum value of *number* in all processes. After the choosing phase, a process waits until its ticket is less than the tickets of all other interested processes. We will refer to this step as the entry phase. Once the process finishes its entry phase, it enters the critical section. When a process releases the critical section, it resets its ticket to zero (releasing phase).

In the following, we specify different models (formulations) for the above algorithm. These models differ on the assumptions concerning two phases, namely the *choosing phase* and the *entry phase*. We use parameterized systems with global conditions to specify the different formulations.

Notation: To simplify the presentation we use the following abbreviations: s$\cdot$ instead of self$\cdot$, o$\cdot$ instead of other$\cdot$, and, for any variable x, we use the primed variable x' as an abbreviation for x^{next}.

Also, the only primed variables we explicitly specify in the conditions are those updated during the corresponding transition. In other words, all non mentioned primed variables are assumed to equal their non-primed counterpart.

10.1 Simplified Bakery Algorithm

Let us assume in Figure 1 that both the *choosing* and the *entry phases* are atomic operations. Under these assumptions, we model the bakery algorithm as shown in Figure 2. Individual processes have a local state in the set $Q = \{idle, wait, use\}$, and a local variable *num* used to maintain the current ticket. In rule t_1 a process moves

States: $Q = \{idle, wait, use\}$

Local variables: $X = \{num \in \mathcal{N}\}$

Transitions:

$t_1 : \left[\, idle \to wait \;\; \triangleright \;\; \forall\, \mathtt{o} \neq \mathtt{s} \cdot \;\; \mathtt{s}{\cdot}num' > \mathtt{o}{\cdot}num \,\right]$

$t_2 : \left[\, wait \to use \;\; \triangleright \;\; \forall\, \mathtt{o} \neq \mathtt{s} \cdot \;\; \left(\, \mathtt{o}{\cdot}num = 0 \;\vee\; \mathtt{s}{\cdot}num < \mathtt{o}{\cdot}num \,\right) \,\right]$

$t_3 : \left[\, use \to idle \;\; \triangleright \;\; num' = 0 \,\right]$

Fig. 2 Simplified Bakery Algorithm.

from *idle* to *wait* and assigns to *num* a value strictly greater than the value of *num* in all other processes. This operation is done in an atomic phase and models the choosing phase. In rule t_2 a process moves from *wait* to *use* only if the value of *num* in any other process, i.e., $\mathtt{o}{\cdot}num$, is either zero (not interested in entering the critical section) or larger than the ticket stored in the local variable *num* ($\mathtt{s}{\cdot}num < \mathtt{o}{\cdot}num$). We model the entry phase using a universally quantified formula, i.e., as an atomic step. In the last rule t_3, a process moves from *use* to *idle* and resets *num* to zero. Initial configurations consist of processes in state *idle* with their local variables *num* set to *zero*. Thus, a process uses rule t_3 to jump to its initial state. Our model corresponds to the model with atomicity assumptions of [13].

In this case-study we are interested in verifying the safety of the mutual exclusion algorithm for any number of processes. Unsafe states are configurations in which at least two processes are inside the critical section. This infinite set of configurations can be modelled by the single constraint

$$\phi_F = \left(2, use^1 \wedge use^2\right)$$

As shown in Table 1, when running our procedure on this model with ϕ_F, the analysis terminates and in the resulting fixpoint there are no constraints whose denotations contain initial states. This proves that mutual exclusion holds for the model in Figure 2 for any number of processes.

10.2 Bakery with Race Conditions

The original formulation of the bakery algorithm does not take any atomicity assumption on the computation of fresh tickets. The entry condition becomes more complicated if we drop this assumption. Such a formulation can be written in the pseudo-code of Figure 3. Since two processes may choose the same ticket, the condition compares the tickets and the identifiers: if two processes choose the same ticket, the one with the smaller identifier gets higher priority. We assume that all identifiers are different. Furthermore, in order to protect the test in the entry phase from race conditions in the *choosing phase*, Lamport's algorithm uses one Boolean flag for each process (hence the array *choosing*). The flag is set to true before starting the assignment to the local variable *number*, and to false after its completion. A process performing the *entry phase*

```
var choosing: shared array [0..n-1] of boolean;
    number:   shared array [0..n-1] of integer;

Process P[i] :=
 loop forever
  choosing[i] := true;
  number[i] := max(number[0],number[1],...,number[n-1])+1;
  choosing[i] := false;
  for j := 0 to n-1 do begin
    while choosing[j] do skip;
    while number[j] <> 0 and (number[j], j) << (number[i],i) do skip;
  end;
  (* critical section *)
  number[i] := 0;
  (* non-critical section *);
```

Fig. 3 Pseudo code for bakery algorithm with race conditions on assignments. The relation $(a,b) << (c,d)$ holds iff $a < c$ or ($a = c$ and $b < d$).

States: $Q = \{idle, choose, wait, use\}$

Local variables: $X = \{id, aux, num \in \mathcal{N}\}$

Transitions:

$t_1 : \left[\, idle \to choose \;\triangleright\; \forall\, \mathtt{o} \neq \mathtt{s} \cdot\ \mathtt{s}{\cdot}aux' > \mathtt{o}{\cdot}num \,\right]$

$t_2 : \left[\, choose \to wait \;\triangleright\; num' = aux \,\right]$

$$t_3 : \left[\begin{array}{l} wait \to use \;\triangleright \\ \left(\begin{array}{l} \forall\, \mathtt{o} \neq \mathtt{s}\cdot \\ \quad \neg \mathtt{o}{\cdot}choose \wedge \\ \quad (\mathtt{o}{\cdot}num = 0 \vee \mathtt{s}{\cdot}num < \mathtt{o}{\cdot}num \;\vee\; (\mathtt{s}{\cdot}num = \mathtt{o}{\cdot}num \wedge \mathtt{s}{\cdot}id < \mathtt{o}{\cdot}id)) \end{array}\right) \end{array}\right]$$

$t_4 : \left[\, use \to idle \;\triangleright\; num' = 0 \,\right]$

Fig. 4 Bakery with Race Conditions.

and willing to compare its ticket with another process in its choosing phase, is forced to wait for the other process to conclude its choosing phase.

In Figure 4 we present a parameterized system that models the bakery algorithm with race conditions in the choosing phase. Here, each process has a local state from the set $Q = \{idle, choose, wait, use\}$ and three local variables. The variable id maintains the identifier of the process. The variable num maintains the current value of the ticket. Finally, the variable aux is used to split in two phases the assignment of the fresh number to num, i.e., to make explicit the non-atomicity of the choosing phase. The choosing phase spans over two rules. In rule t_1 we first update aux with a value greater than the tickets of all other processes. In rule t_2 we assign aux to num. Rule t_3 models the entry phase by means of a universally quantified formula. The current process compares its number with the number of all other processes. We assume this phase to be atomic. Notice that the condition fails (i.e. the process waits) in each one of the following cases: another process id' is choosing a number, id' has chosen a number that is strictly smaller than the value of num or id' has chosen a number that is equal to the value of num but $id' < id$. Rule t_4 models the release of the critical section and

Step 6
$(2, idle^1, idle^2, id^1 < id^2) : \underline{12}, \underline{11}, t_1$ ★
Step 5
$(2, idle^1, choose^2, id^1 < id^2) : \underline{11}, \underline{9}, t_1$ ★
$(2, idle^1, choose^2, aux^2 > 0, id^1 < id^2) : \underline{10}, \underline{8}, t_1$ ✠
Step 4
$(2, choose^1, choose^2, id^1 < id^2, aux^1 < aux^2) : \underline{9}, \underline{7}, t_2$ ★
$(2, choose^1, choose^2, id^1 < id^2, aux^1 = aux^2) : \underline{8}, \underline{6}, t_2$ ✠
Step 3
$(2, choose^1, wait^2, id^1 < id^2, aux^1 < num^2) : \underline{7}, \underline{5}, t_3$ ★
$(2, choose^1, wait^2, id^1 < id^2, aux^1 = num^2) : \underline{6}, \underline{4}, t_3$ ✠
Step 2
$(2, choose^1, use^2, id^1 < id^2, aux^1 < num^2) : \underline{5}, \underline{3}, t_2$ ★
$(2, choose^1, use^2, id^1 < id^2, aux^1 = num^2) : \underline{4}, \underline{2}, t_2$ ✠
Step 1
$(2, wait^1, use^2, id^1 < id^2, num^1 < num^2) : \underline{3}, \underline{1}, t_3$ ★
$(2, wait^1, use^2, id^1 < id^2, num^1 = num^2) : \underline{2}, \underline{1}, t_3$ ✠
Step 0 : **Bad states**
$(2, use^1, use^2, id^1 < id^2) : \underline{1}$ ★,✠

Fig. 5 Excerpt from the fixpoint for Bogus bakery: the data after each formula are tracking information: number for the formula, number of formula from which it derives from, and transition used to derived it.

the return to the initial state. As in the simplified version of bakery, we prove that mutual exclusion holds for the model in Figure 4 for any number of processes starting from ϕ_F. We show the results in Table 1.

10.3 A Bogus Formulation of the Bakery Algorithm

Our verification procedure builds a symbolic reachability graph that can be used for debugging purposes. To illustrate, let us consider again the bakery algorithm described in the previous section. If we remove the condition $\neg \circ \cdot choose$ from the entry condition in t_3, we obtain a new transition defined as

$$\left[wait \rightarrow use \ \triangleright \ \forall\, \mathtt{o} \neq \mathtt{s} \cdot \begin{pmatrix} \mathtt{o} \cdot num = 0 \ \vee \ \mathtt{s} \cdot num < \mathtt{o} \cdot num \\ \vee \ \left(\mathtt{s} \cdot num = \mathtt{o} \cdot num \ \wedge \ \mathtt{s} \cdot id < \mathtt{o} \cdot id \right) \end{pmatrix} \right]$$

This modified version of Lamport's bakery algorithm is a classical example used in textbooks to explain why the *choosing* flag is needed.

As shown in Table 1, when executed on this model our verification procedure terminates after 10 iterations. Among the formulas computed in the fixpoint, we find two formulas representing initial configurations (two *idle* processes). By exploiting some tracking information inserted during the search it is possible to reconstruct a symbolic trace from these formulas to the formula representing bad states. The resulting error trace is indicated as ★ in Figure 5 and contains constraints 12, 11, 9, 7, 5, 3, and 1. This trace represents the following type of execution. Process 2 moves to state *choose*, selects a number m, and assigns it to the local variable *aux*. Process 1 moves to state *choose*, and, then selects and assigns to the local variable *aux* a number n s.t. $n < m$. Process 2 assigns *aux* to *num*. Since process 1 has not assigned the new value stored in *aux* to *num*, process 2 can enter the critical section. Process 1 now assigns *aux* to *num* and enters the critical section because its number is smaller than that of process

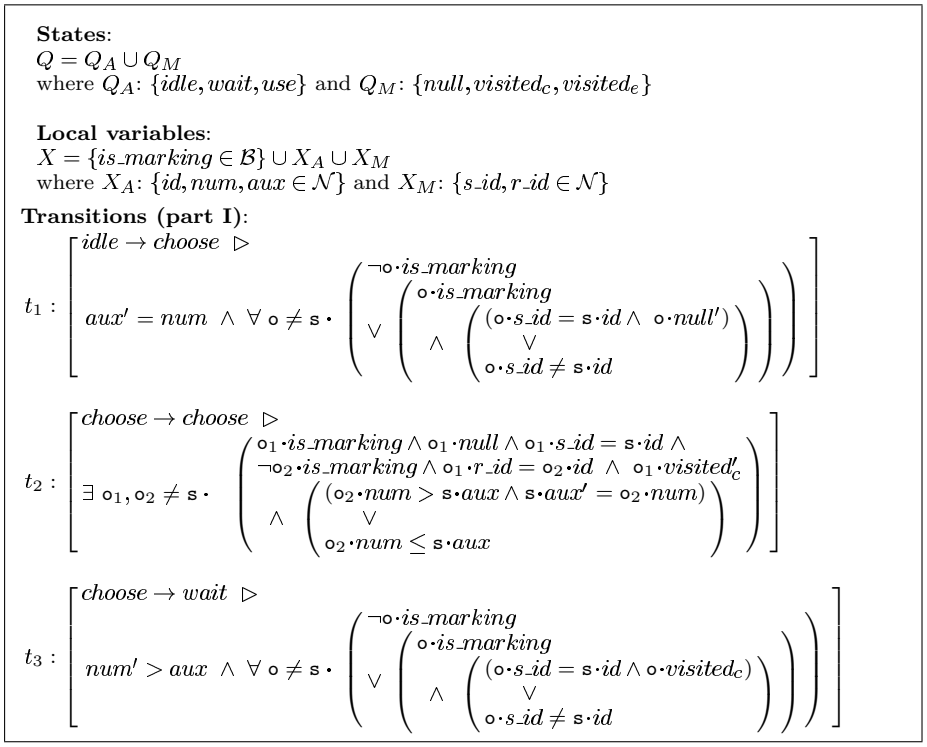

Fig. 6 Non-atomic computation of ticket.

2. This execution corresponds to a real error trace that explain why a process has to wait that the choosing phase of all other processes is terminated before entering the critical section.

Another interesting error trace is that indicated by the symbol ✠ in Figure 5 that contains constraints 10, 8, 6, 4, 2, and 1. When computing predecessors, the constraint 10 produces a formula subsumed by the formula 12. This implies that it is possible to reach an instance of 10 from an initial state. By looking at the trace from 10 to the bad states, we discover another subtle error trace. Process 2 selects a value n greater than the value of num for process 1 and assigns it to aux. Process 1 chooses a value $m = n$, and assigns it to aux. Process 2 can enter the critical section because process 1 has not terminated the assignment. Process 1 completes its assignment and enters the critical section since its identifier is less than that of process 2.

10.4 Bakery with Non-atomic Conditions

In this section we present a formal model for a version of the bakery algorithm in which choosing and entry phase are both non-atomic. Testing a condition non-atomically on a set of processes means that we admit any possible interleaving between the test of the condition over each other processes and any other enabled transition.

Transitions (part II) :

$$t_4 : \left[\begin{array}{l} wait \rightarrow wait \ \rhd \\ \forall\, \mathtt{o} \neq \mathtt{s} \cdot \ \neg \mathtt{o}{\cdot}is\_marking \vee \left(\mathtt{o}{\cdot}is\_marking \wedge \left(\begin{array}{l} (\mathtt{o}{\cdot}s\_id = \mathtt{s}{\cdot}id \wedge \ \mathtt{o}{\cdot}null') \\ \quad \vee \\ \mathtt{o}{\cdot}s\_id \neq \mathtt{s}{\cdot}id \end{array} \right) \right) \end{array} \right]$$

$$t_5 : \left[\begin{array}{l} wait \rightarrow wait \ \rhd \\ \exists\, \mathtt{o}_1, \mathtt{o}_2 \neq \mathtt{s} \cdot \left(\begin{array}{l} \mathtt{o}_1{\cdot}is\_marking \wedge \neg \mathtt{o}_2{\cdot}is\_marking \wedge \mathtt{o}_1{\cdot}null \wedge \mathtt{o}_1{\cdot}s\_id = \mathtt{s}{\cdot}id \\ \wedge \mathtt{o}_2{\cdot}id = \mathtt{o}_1{\cdot}r\_id \wedge \neg \mathtt{o}_2{\cdot}choose \ \wedge \mathtt{o}_1{\cdot}visited'_e \\ \wedge \left(\begin{array}{l} \mathtt{o}_2{\cdot}num = 0 \ \vee \ \mathtt{s}{\cdot}num < \mathtt{o}_2{\cdot}num \\ \quad \vee \\ (\mathtt{o}_2{\cdot}num = \mathtt{s}{\cdot}num \wedge \mathtt{s}{\cdot}id < \mathtt{o}_1{\cdot}r\_id) \end{array} \right) \end{array} \right) \end{array} \right]$$

$$t_6 : \left[\begin{array}{l} wait \rightarrow use \ \rhd \\ \forall\, \mathtt{o} \neq \mathtt{s} \cdot \ \neg \mathtt{o}{\cdot}is\_marking \vee \left(\mathtt{o}{\cdot}is\_marking \wedge \left(\begin{array}{l} (\mathtt{o}{\cdot}s\_id = \mathtt{s}{\cdot}id \wedge \mathtt{o}{\cdot}visited_e) \\ \quad \vee \\ \mathtt{o}{\cdot}s\_id \neq \mathtt{s}{\cdot}id \end{array} \right) \right) \end{array} \right]$$

$$t_7 : \left[use \rightarrow idle \ \rhd \ num' = 0 \right]$$

Fig. 7 Non-atomic entry phase.

Our non-atomic model of bakery is based on the following idea. Let us first consider the computation of new tickets. Each process uses an auxiliary local variable *aux* used to compute the maximal value among the set of values of *num* in other processes. The computation of a new ticket is performed by visiting the other processes in an arbitrary order. Fix one of the other processes, the maximum value of *aux* and *num* is assigned to *num*, and the process is marked as visited. When all processes have been visited, a value strictly greater than *aux* is assigned to *num*.

The entry condition is tested in a similar way. Specifically, we compares the value of *num* in the current process with the value of *num* in each other process. If the entry condition is satisfied by a process, we mark it as visited. When all processes have been visited, we can enter the critical section.

The formal model is shown in Figure 6 (non-atomic choosing phase), and Figure 7 (non-atomic entry phase). We use here processes of two types: *agents* and *marking* processes. Agents model processes competing for the critical section. Marking processes are used to mark visited agents. We associate a marking process to each pair of process identifiers. A Boolean local variable *is_marking* is used to distinguish between the two types of processes, i.e., we assume that *is_marking* is set to true only for marking processes. Agents and marking are defined on the set of states Q_A and Q_M, respectively (see Figure 6). Agents have three local variables, namely *id* (agent identifier), *num* (ticket), and *aux* (auxiliary variable). Marking processes have two local variables, namely *s_id* (the process that computes a ticket or tests a condition), and *r_id* (the index of the process to be visited). The flag *is_marking* is used to select the subset of local variables that are relevant for the two types of processes (agents ignore *s_id* and *r_id*, whereas marking processes ignore *id*, *num* and *aux*). Since we need to mark processes in two distinct phases (choosing and entry), marking processes assume three different states: *null*, $visited_c$ (choosing phase) and $visited_e$ (entry phase). Initial states are collections of *idle* agents (each one with unique identifiers) and of marking

processes in state *null*. Notice that, for a fixed process identifier id, the collection of marking processes such that $s\_id = id$ can be viewed as an unbounded array indexed on all other process identifiers whose cells, say $marking(id, id')$, contain a value in $\{null, visited_c, visited_e\}$.

We are ready now to explain the meaning of the rules in Figure 6. In transition t_1, when a process id wants to enter the critical section, it assigns num to an auxiliary variable aux, and sets the state of the marking process $marking(id, id')$ to $null$ for each other processes id'. Then, it moves to state *choose*. Notice that with conditions of the form

$$\forall\, \mathtt{o} \neq \mathtt{s} \cdot\ \neg \mathtt{o} \cdot is\_marking\ \vee (\mathtt{o} \cdot is\_marking\ \wedge\ \varphi)$$

like the condition used in t_1, we can require φ to hold only on processes that represent marking processes. In transitions t_2, process id compares aux with the value of other processes. Specifically, for each process id' the maximum value between aux in id and num in id' is assigned to aux' and id' is marked as visited, i.e., the state of process $marking(id, id')$ is updated to $visited_c$. These rules are specified via a synchronization of three processes (two agents and a marking process). In transition t_3, the process id moves to state $wait$ when all other processes have been visited (i.e. $marking(id, id')$ is in state $visited_c$ for any id') and updates num by taking a value strictly greater than aux. This ensures the freshness of the generated ticket.

We now describe the rules in Figure 7. In transition t_4, process id sets the state of $marking(id, id')$ to $null$, and starts comparing its current ticket with the ticket of all other processes. In transition t_5, process id tests the entry condition on process id'. If the condition succeeds, the state of $marking(id, id')$ is updated to $visited_e$ meaning that id' has been successfully visited. This transition employs an existential quantification that involves three processes. In transition t_6, process id checks that all marking processes are in state $visited_e$ and then enters critical section (state use). In transition t_7, process id releases the critical section, and sets num back to zero.

In this model unsafe states are configurations in which at least two processes are inside the critical section. They can be modeled by the single constraint

$$\phi_F \;=\; \left(4,\ \begin{array}{l} \neg is\_marking^1 \wedge \neg is\_marking^2 \wedge is\_marking^3 \wedge is\_marking^4 \wedge \\ use^1 \wedge use^2 \wedge id^1 = s\_id^3 \wedge id^2 = r\_id^3\ \wedge\ id^2 = s\_id^4 \wedge id^1 = r\_id^4 \end{array}\right)$$

With this formula, we assert the presence of at least two distinct processes in state use (process 1 and 2) and of two marking processes $marking(id^1, id^2)$ and $marking(id^2, id^1)$ with any possible content.

As shown in Table 1, in this case the symbolic fixpoint computation terminates. Furthermore, initial configurations (i.e. in which all processes are in state $idle$ and all marking processes have state $null$) do not occur in the denotation of the resulting fixpoint. This proves that mutual exclusion holds for any number of processes.

10.5 Distributed Bakery: Ticket Mutual Exclusion Algorithm

As a final experiment, we analyze a variation of the bakery algorithm used in distributed systems. The Ticket Algorithm [7] is a distributed version of the bakery algorithm in which a central monitor is used to generate tickets and to maintain the ticket of the process to be served next. The central monitor can be modeled by means of two global variables: t maintains the next fresh ticket, and s the ticket of the next process to serve.

Global variables:
$X = \{t, s \in \mathcal{N}\}$
Local variables:
$X = \{a \in \mathcal{N}\}$

States:
Q: $\{idle, wait, use\}$

Transitions:

$t_1 : [\, idle \to wait \;\triangleright\; a' = t \wedge t' > t \,]$

$t_2 : [\, wait \to use \;\triangleright\; a = s \,]$

$t_3 : [\, use \to idle \;\triangleright\; (\exists\, \mathtt{o} \neq \mathtt{s} \cdot\; \mathtt{s}{\cdot}s' = \mathtt{o}{\cdot}a) \;\wedge\; (\forall\, \mathtt{o} \neq \mathtt{s} \cdot\; \mathtt{s}{\cdot}s' \leq \mathtt{o}{\cdot}a) \,]$

Fig. 8 Ticket mutex algorithm.

Model	N-R	N-I	E-R	E-I	S-R	S-I	T-R	T-I
Atomic conditions	6	6	0.8s	0.01s	√	√		
Race conditions	9	4	2.1s	0.01s	√	√		
Bogus	10	6	0.8s	0.22s			√	√
Non-atomic conditions	14	11	18s	0.3s	√	√		
Ticket algorithm	9	19	0.3s	1.4s	√	√		

Table 1 Experimental results on different formulations of the bakery algorithm.

When interested in entering the critical section, a process gets a fresh ticket and stores it in a local integer variable a. Then it waits until its turn comes, i.e., until $s = a$ and then enters. When a process leaves the critical section, the monitor assigns to s the smallest value among all processes in the system. In other words, the central monitor maintains the processes waiting to enter the critical section in a FIFO queue. Since processes have a single local integer variable, our algorithm is guaranteed to terminate when applied to this model.

As shown in Table 1, our procedure terminates and it automatically proves mutual exclusion for any number of processes.

A parameterized formulation of an abstraction of the ticket algorithm with central monitor in which rules have no universally quantified guards is studied in [11]. In [11] 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.

10.6 Summary of Experiments

Table 1 summarizes the experimental results for the different formulations of the bakery algorithm described in this section. The later example is discussed in [3]. For the experiments we have used two different implementations of the verification procedure. The first implementation is based on the general purpose clp(Q,R) solver for real arithmetic, while the second implementation is based on a specialized solver we implement

by adapting difference bound matrices on integers to handle gap-order constraints. In Table 1, **R** and **I** indicate the use of the real and integer solver respectively. **N** indicates the number of iterations needed for computing the fixpoint. **E** indicates the execution time. Finally, **S** and **T** indicate if the safety properties has been proved or if an error trace has been detected.

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. In this paper we have discussed in detail verification of mutual exclusion for several formulations of Lamport's bakery algorithm including formulations without atomicity assumptions on global conditions.

Our method has been successfully applied to several other examples of parameterized systems like Burns', Dijkstra and Szymanski's mutual exclusion protocols with atomicity conditions [5], and to distributed protocols like Ricart-Agrawala [4]. Descriptions of these case-studies are given in [23]. Our approximation scheme cannot directly be applied to protocols in which processes in a given state are used to control race conditions among processes competing for the critical section. One such example is a non-atomic formulation of Szymanski's mutual exclusion algorithm. For this protocol, our approximation returns a spurious trace. We are currently working at a refinement method to eliminate this kind of false positives.

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. This would allow us to extend our method in order to verify systems such as those in [10].

References

1. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.K.: Algorithmic analysis of programs with well quasi-ordered domains. Information and Computation **160**, 109–127 (2000)
2. Abdulla, P.A., Delzanno, G.: On the coverability problem for constrained multiset rewriting. In: Proc. AVIS'06, 5^{th} Int. Workshop on on Automated Verification of Infinite-State Systems (2006)
3. Abdulla, P.A., Delzanno, G., Rezine, A.: Parameterized verification of infinite-state processes with global conditions. In: Proc. 19^{th} Int. Conf. on Computer Aided Verification, *Lecture Notes in Computer Science*, vol. 4590, pp. 145–157 (2007)
4. Abdulla, P.A., Delzanno, G., Rezine, A.: Monotonic abstraction in action (automatic verification of distributed mutex algorithms). In: ICTAC (2008). To appera.
5. Abdulla, P.A., Henda, N.B., Delzanno, G., Rezine, A.: Regular model checking without transducers (on efficient verification of parameterized systems). In: Proc. TACAS '07, 13^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (2007)
6. Abdulla, P.A., Jonsson, B., Nilsson, M., d'Orso, J.: Regular model checking made simple and efficient. In: Proc. CONCUR 2002, 13^{th} Int. Conf. on Concurrency Theory, *Lecture Notes in Computer Science*, vol. 2421, pp. 116–130 (2002)
7. Andrews, G.: Foundations of Multithreaded, Parallel, and Distributed Programming. Addison Wesley (2000)

8. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.: Parameterized verification with automatically computed inductive assertions. In: Berry, Comon, Finkel (eds.) Proc. 13^{th} Int. Conf. on Computer Aided Verification, *Lecture Notes in Computer Science*, vol. 2102, pp. 221–234 (2001)
9. Boigelot, B., Legay, A., Wolper, P.: Iterating transducers in the large. In: Proc. 15^{th} Int. Conf. on Computer Aided Verification, *Lecture Notes in Computer Science*, vol. 2725, pp. 223–235 (2003)
10. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract tree regular model checking of complex dynamic data structures. In: Proc. 13^{th} Int. Symp. on Static Analysis (2006)
11. Bozzano, M., Delzanno, G.: Beyond parameterized verification. In: Proc. TACAS '02, 8^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, *Lecture Notes in Computer Science*, vol. 2280, pp. 221–235 (2002)
12. Bultan, T., Gerber, R., Pugh, W.: Model-checking concurrent systems with unbounded integer variables. ACM Trans. on Programming Languages and Systems **21**(4), 747–789 (1999)
13. Clarke, E., Talupur, M., Veith, H.: Environment abstraction for parameterized verification. In: Proc. VMCAI '06, 7^{th} Int. Conf. on Verification, Model Checking, and Abstract Interpretation, *Lecture Notes in Computer Science*, vol. 3855, pp. 126–141 (2006)
14. Delzanno, G.: Automatic verification of cache coherence protocols. In: Emerson, Sistla (eds.) Proc. 12^{th} Int. Conf. on Computer Aided Verification, *Lecture Notes in Computer Science*, vol. 1855, pp. 53–68. Springer Verlag (2000)
15. Emerson, E., Namjoshi, K.: On model checking for non-deterministic infinite-state systems. In: Proc. LICS '98, 13^{th} IEEE Int. Symp. on Logic in Computer Science, pp. 70–80 (1998)
16. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: Proc. LICS '99, 14^{th} IEEE Int. Symp. on Logic in Computer Science (1999)
17. Fribourg, L., Richardson, J.: Symbolic verification with gap-order constraints. In: LOPSTR'96, *Lecture Notes in Computer Science*, vol. 1207 (1997)
18. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. Journal of the ACM **39**(3), 675–735 (1992)
19. Kesten, Y., Maler, O., Marcus, M., Pnueli, A., Shahar, E.: Symbolic model checking with rich assertional languages. Theoretical Computer Science **256**, 93–112 (2001)
20. Lahiri, S.K., Bryant, R.E.: Indexed predicate discovery for unbounded system verification. In: CAV 2004, pp. 135–147 (2004)
21. Lamport, L.: A new solution of Dijkstra's concurrent programming problem. Commun. ACM **17**(8), 453–455 (1974)
22. Revesz, P.: A closed form evaluation for datalog queries with integer (gap)-order constraints. Theoretical Computer Science **116**(1), 117–149 (1993)
23. Rezine, A.: Parameterized systems: Generalizing and simplifying automatic verification. Ph.D. thesis, Uppsala University (2008)
24. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Proc. LICS '86, 1^{st} IEEE Int. Symp. on Logic in Computer Science, pp. 332–344 (1986)

A Computing *Pre*

In the following we use u, u' and $\tilde{u}$ to denote *process states*, that is, mappings from Y to $\mathcal{B} \cup \mathcal{N}$. Unless otherwise specified, a process state indexed by i, for instance u_i, is a mapping from Y^i to $\mathcal{B} \cup \mathcal{N}$. Given a substitution $\rho = \{x^j \leftarrow x^i \mid x^i \in Y^i\}$, we write $(\rho)_{inv}$ to mean the substitution $\{x^i \leftarrow x^j \mid x^j \in Y^j\}$. We sometimes view a process state $u_i : Y^i \to \mathcal{B} \cup \mathcal{N}$ as the substitution $\{x^i \leftarrow u(x^i) \mid x^i \in Y^i\}$. For a substitution $\rho = \{x^i \leftarrow x^j \mid x^j \in Y^j\}$, and a process state $u_i : Y^i \to \mathcal{B} \cup \mathcal{N}$, we write $u_i\,[\rho]$ to mean the process state that sends each variable x^j in Y^j to $u(x^i)$.

Proof (lemma 3) Recall $\phi' = (m, \psi')$ and θ is the condition in the transition t. We distinguish three cases depending on whether θ is a local, a global existential or a global universal condition. We show $Pre_t(\phi') = \left\{ c \in [\![\phi']\!] \mid c \overset{t}{\rightsquigarrow} c' \right\}$ for each case by considering both directions.

- Local conditions. Can be easily deduced from the existential and the universal cases.

- Existential conditions.

  * First direction: $[\![Pre_t(\phi')]\!] \subseteq \left\{c \middle|\, \exists c' \in [\![\phi']\!] \,.\, c \xrightarrow{t} c'\right\}$.
  For $\phi' = (m, \psi')$, let $\phi = (n, \psi_{i,j})$ be in $Pre_t(\phi')$. The following four cases are possible by definition of Pre_t. Either i) $(n,i,j) = (m,k,l)$ for some k,l in $\overline{m}$ and $k \neq l$, or ii) $(n,i,j) = (m+1,k,m+1)$ for some k in $\overline{m}$, or iii) $(n,i,j) = (m+1,m+1,l)$ for some l in $\overline{m}$, or iv) $(n,i,j) = (m+2,m+1,m+2)$. Intuitively, the process taking the transition is indexed by i, while the witness is indexed by j.
  We let c be a configuration in $[\![\phi]\!]$ and show the existence of a configuration c' in $[\![\phi']\!]$ such that $c \xrightarrow{t} c'$. We assume c to equal $[u_1, \ldots, u_{|c|}]$ where, up to a renaming, u_k for k in $\overline{n}$ are mappings from Y^k to $\mathcal{B} \cup \mathcal{N}$ such that $\psi_{i,j}\,[u_1, \ldots, u_n]$ holds. That is to say
  $$\left(\left(\exists Y^i . \psi' \wedge \theta_1^\bullet\,[\rho_1^i]\,\left[\rho_2^j\right]\right)\,[\rho_3^i]\right)\,[u_1, \ldots, u_n] \tag{4}$$
  where $\theta_1^\bullet$ is defined as
  $$\theta_1 \wedge \texttt{self}\cdot q \wedge \texttt{self}\cdot q'^{next} \wedge \bigwedge_{p \neq q} \neg \texttt{self}\cdot p \wedge \bigwedge_{p' \neq q'} \neg \texttt{self}\cdot q'^{next}$$
  while $\rho_1^i = \left\{\text{self}\cdot x^{next} \leftarrow x^i \middle|\, x^i \text{ in } Y^i\right\}$, $\rho_2^j = \left\{\text{other}\cdot x \leftarrow x^j \middle|\, x^j \text{ in } Y^j\right\}$, and $\rho_3^i = \left\{\text{self}\cdot x \leftarrow x^i \middle|\, x^i \text{ in } Y^i\right\}$. Notice that u_i, respectively u_j, is the only process state in $u_1, \ldots, u_n$ mapping variables in Y^i, respectively Y^j.
  Observe that (4) implies the existence of a mapping from Y^i to $\mathcal{B} \cup \mathcal{N}$ (that we denote u_i') and such that
  $$\left(\psi' \wedge \theta_1^\bullet\,[\rho_1^i]\,\left[\rho_2^j\right]\right)\,[u_1, \ldots, u_i\,[(\rho_3^i)_{inv}], \ldots, u_n, u_i'] \tag{5}$$
  Intuitively, u_i' corresponds to the next-values of the process that took the transition t. Define c' to be the configuration $\left[u_1', \ldots, u_{|c|}'\right]$ where $u_k' = u_k$ for each k in $\overline{(|c|)} \setminus \{i\}$. Notice that, except for the variables of process i, the variables of processes in c' are mapped to the same values as those of c.
  We show $c' \in [\![\phi']\!]$. Observe that ψ' in (5) does not refer to any element of self$\cdot Y$, or Y^o for $o \notin \overline{m}$. We therefore discard u_i and u_o' for o outside $\overline{m}$, and deduce $\psi'\,[u_1', \ldots, u_m']$. In other words, the configuration $[u_1', \ldots, u_m']$ is in $[\![\phi']\!]$. Clearly, $[u_1', \ldots, u_m'] \preceq c'$ (take the injection to be the identity on $\overline{m}$). Lemma (2) says the set $[\![\phi']\!]$ is upward closed, and hence $c' \in [\![\phi']\!]$.
  Now, we show $c \xrightarrow{t} c'$. It is sufficient to prove $c \xrightarrow{t} c'$. Observe (5) also implies $\left(\theta_1^\bullet\,[\rho_1^i]\,\left[\rho_2^j\right]\right)\,[u_1, \ldots, u_i\,[(\rho_3^i)_{inv}], \ldots, u_n, u_i']$. Because $\theta_1^\bullet\,[\rho_1^i]\,\left[\rho_2^j\right]$ only depends on variables in self$\cdot Y$, Y^i or Y^j, we can ignore process states that map other variables, meaning we can write $\left(\theta_1^\bullet\,[\rho_1^i]\,\left[\rho_2^j\right]\right)\,[u_i\,[(\rho_3^i)_{inv}], u_i', u_j]$. This has two consequences. i) u_i, respectively u_i', ensures the first process is at state q, respectively q', and ii) $(u_i, u_i', u_j) \models \theta_1$. In other words, $[u_1, \ldots, u_n] \xrightarrow{t} [u_1', \ldots, u_n']$. Furthermore, since the transition modifies only the local state of process i, and because the existential quantification remains valid in any larger configuration, we have that $c \xrightarrow{t} c'$.

  * Second direction: $\left\{c \middle|\, \exists c' \in [\![\phi']\!] \,.\, c \xrightarrow{t} c'\right\} \subseteq [\![Pre_t(\phi')]\!]$.
  The transitions involving existential conditions are not over-approximated, hence $c \xrightarrow{t} c'$ iff $c \xrightarrow{t} c'$. Assume $c = [u_1, \ldots, u_{|c|}]$ and $c' = \left[u_1', \ldots, u_{|c'|}'\right]$. Observe that $c \longrightarrow c'$ says $|c| = |c'|$, and implies the existence of $i \neq j$, both in $\overline{|c'|}$, such that the three following statements hold. First, the process i is at q in c, and at q' in c'; second $(u_i, u_i', u_j) \models \theta_1$; and third $u_k = u_k'$ for every k in $\overline{|c'|} \setminus \{i\}$.
  Also, recall $\phi' = (m, \psi')$. We can assume, up to a renaming, that process states $u_1' \ldots, u_{|c'|}'$ verify $\psi'\,[u_1', \ldots, u_m']$, $i \in \overline{m+1}$ and $j \in \overline{m+2} \setminus \{i\}$. Define n to take the maximum on $\{m, i, j\}$.

The condition ψ' only depends on control states and variables in $Y^1 \dots Y^m$. Define ρ_3^i to be the substitution $\{\text{self}{\cdot}x \leftarrow x^i \mid x^i \in Y^i\}$. ψ' does not depend on elements of $\text{self}{\cdot}Y$, and we clearly get

$$\psi' \left[u'_1, \dots, u_i \left[(\rho_3^i)_{inv}\right], \dots, u'_n\right] \tag{6}$$

Also, recall $(u_i, u'_i, u_j) \models \theta_1$ means θ_1 holds when occurrences of $\text{self} \cdot x$, $\text{self} \cdot x^{next}$, and $\text{other} \cdot x$ are respectively replaced by the values stated in u_i, u'_i and u_j. Furthermore, u_i and u'_i respectively state that process i is at q and q'. Define $\rho_1^i = \{\text{self}{\cdot}x^{next} \leftarrow x^i \mid x^i \text{ in } Y^i\}$, and $\rho_2^j = \{\text{other}{\cdot}x \leftarrow x^j \mid x^j \text{ in } Y^j\}$. We can therefore write $\left(\theta_1^\bullet \left[\rho_1^i\right] \left[\rho_2^j\right]\right) \left[u_i \left[(\rho_3^i)_{inv}\right], u'_i, u_j\right]$. Since $\theta_1^\bullet$ only depends on $\text{self}{\cdot}x$, $\text{self}{\cdot}x^{next}$, and $\text{other} \cdot x$, we deduce

$$\left(\theta_1^\bullet \left[\rho_1^i\right] \left[\rho_2^j\right]\right) \left[u_1, \dots u_i \left[(\rho_3^i)_{inv}\right], \dots u_n, u'_i\right] \tag{7}$$

By combining (6) and (7), we get

$$\left(\exists Y^i . \left(\psi' \wedge \theta_1^\bullet \left[\rho_1^i\right] \left[\rho_2^j\right]\right)\right) \left[u_1, \dots u_i \left[(\rho_3^i)_{inv}\right], \dots u_n\right]$$

We deduce $[u_1, \dots, u_n]$ is in $[\![(n, \psi_{i,j})]\!]$. Lemma (2) says $[\![(n, \psi_{i,j})]\!]$ is upward closed. Combined with $[u_1, \dots, u_n] \preceq c$ (identity on $\overline{n}$), we conclude that the configuration c is also in $[\![Pre_t(\phi')]\!]$.

- Universal conditions

  * First direction $[\![Pre_t(\phi')]\!] \subseteq \left\{c \mid \exists c' \in [\![\phi']\!] \,.\, c \xrightarrow{t} c'\right\}$
  For $\phi' = (m, \psi')$, let $\phi = (n, \psi_i)$ be in $Pre_t(\phi')$. Two cases are possible according to the definition of Pre_t. Either i) $(n, i) = (m, k)$ for some k in $\overline{m}$, or ii) $(n, i) = (m+1, m+1)$. Intuitively, the process taking the transition is indexed by i, This process may or may not be represented in ψ'.
  We let c be a configuration in $[\![\phi]\!]$ and show the existence of a configuration c' in $[\![\phi']\!]$ such that $c \xrightarrow{t} c'$. Up to a renaming, we write c both as $c_1 \bullet [u_i] \bullet c_2$ and as $\left[u_1, \dots, u_{|c|}\right]$, where $\psi_i\,[u_1, \dots, u_n]$. That is to say

$$\left(\exists Y^i . \left(\psi' \wedge \bigwedge_{j \in \overline{n} \setminus \{i\}} \theta_1^\bullet \left[\rho_1^i\right] \left[\rho_2^j\right]\right) \left[\rho_3^i\right]\right) [u_1, \dots, u_n] \tag{8}$$

  where $\theta_1^\bullet$ is defined as in the existential case.

$$\theta_1 \wedge \texttt{self}{\cdot}q \wedge \texttt{self}{\cdot}q'^{next} \wedge \bigwedge_{p \neq q} \neg \texttt{self}{\cdot}p \wedge \bigwedge_{p' \neq q'} \neg \texttt{self}{\cdot}p'^{next}$$

  where $\rho_1^i = \{\text{self}{\cdot}x^{next} \leftarrow x^i \mid x^i \text{ in } Y^i\}$, $\rho_2^j = \{\text{other}{\cdot}x \leftarrow x^j \mid x^j \text{ in } Y^j\}$, and $\rho_3^i = \{\text{self}{\cdot}x \leftarrow x^i \mid x^i \text{ in } Y^i\}$. Notice that u_i is the only process state in $u_1, \dots, u_n$ mapping variables in Y^i. Notice also that all variables in Y^j for $j \in \overline{n} \setminus \{i\}$ are constrained in (8).
  We deduce the existence of a process state u'_i mapping variables in Y^i to $\mathcal{B} \cup \mathcal{N}$ such that the following holds

$$\left(\psi' \wedge \bigwedge_{j \in \overline{n} \setminus \{i\}} \theta_1^\bullet \left[\rho_1^i\right] \left[\rho_2^j\right]\right) \left[u_1, \dots, u_i \left[(\rho_3^i)_{inv}\right], \dots, u_n, u'_i\right] \tag{9}$$

  Define both configurations $\tilde{c}$ and c' as:

$$\tilde{c} = (c_1 \ominus (\theta_1, u_i, u'_i)) \bullet [u_i] \bullet (c_2 \ominus (\theta_1, u_i, u'_i));$$

$$c' = (c_1 \ominus (\theta_1, u_i, u'_i)) \bullet [u'_i] \bullet (c_2 \ominus (\theta_1, u_i, u'_i)).$$

Notice that $\left(\bigwedge_{j\in\overline{n}\setminus\{i\}} \theta_1^{\bullet}\left[\rho_1^i\right]\left[\rho_2^j\right]\right)\left[u_1,\ldots,u_i\left[(\rho_3^i)_{inv}\right],\ldots,u_n,u_i'\right]$ implies each process state in $\{u_1,\ldots,u_n\}\setminus\{u_i\}$ is in $(c_1\bullet c_2)\ominus(\theta_1,u_i,u_i')$.
By construction, we have that u_i and u_i', respectively ensure process i is at state q and q'. Moreover, we have $(u_i,u_i',u)\models\theta_1$ for each u in $(c_1\bullet c_2)\ominus(\theta_1,u_i,u_i')$. That is to say, $\tilde{c}\xrightarrow{t} c'$, and $c\overset{t}{\rightsquigarrow}_1 c'$. In order to conclude, we show c' in $[\![\phi']\!]$.
The statement (9) ensures $\psi'\left[u_1,\ldots,u_i\left[(\rho_3^i)_{inv}\right],\ldots,u_n,u_i'\right]$. Since ψ' does not mention variables in self$\cdot Y$, we get $\psi'\left[u_1,\ldots,u_i',\ldots,u_n\right]$. Recall that each process state in $\{u_1,\ldots,u_n\}\setminus\{u_i\}$ is in $(c_1\bullet c_2)\ominus(\theta_1,u_i,u_i')$. Clearly $\left[u_1,\ldots,u_j',\ldots,u_n\right]\preceq c'$. Since $[\![\phi']\!]$ is upward closed by lemma (2), we conclude c' to be in $[\![\phi']\!]$.

* Second direction $\left\{c\middle|\ \exists c'\in[\![\phi']\!]\,.\,c\overset{t}{\rightsquigarrow}c'\right\}\subseteq[\![Pre_t(\phi')]\!]$

Let c and c' be two configurations such that $c\overset{t}{\rightsquigarrow}c'$ and c' in $[\![\phi']\!]$. We show c is in $[\![\phi]\!]$ for some ϕ in $Pre_t(\phi')$.
Recall that $\phi'=(m,\psi')$. Assume $c'=\left[u_1',\ldots,u_{|c'|}'\right]$, where up to a renaming the process that took the transition has index i in $\overline{m+1}$ and $\psi'\left[u_1',\ldots,u_m'\right]$. Intuitively, the case $i=m+1$ corresponds to the case where the process that took the transition is not represented in ψ'.
Let u be the process state of the process in $c=c_1\bullet[u]\bullet c_2$ taking the transition t. By definition of $c\overset{t}{\rightsquigarrow}c'$ we have, $c'=c_1'\bullet[u_i']\bullet c_2'$ where i) u in c is at q and u_i' in c' is at q', and ii) $c_1'=c_1\ominus(\theta_1,u,u_i')$ while $c_2'=c_2\ominus(\theta_1,u,u_i')$.
We show c in $[\![\phi]\!]$ for $\phi=(n,\psi_i)$ with $i\in\overline{n}$ and $n\in\{m,m+1\}$.
Define the configuration $\tilde{c}$ to be $\tilde{c}=\tilde{c}_1\bullet[\tilde{u}_i]\bullet\tilde{c}_2=[\tilde{u}_1,\ldots,\tilde{u}_n]$ with $\tilde{u}_k=u_k'$ for each $k\in\overline{n}\setminus\{i\}$, and $\tilde{u}_i$ maps $x^i\in Y^i$ to the same value u maps $x\in Y$ to. Intuitively $\tilde{c}$ corresponds to a predecessor, without approximation, of c'. Consider the injection sending each process state in $\tilde{c}$ to the first equal process state in c, this injection is well defined since $\tilde{c}$ is obtained through deletion of processes from c. This means $\tilde{c}\preceq c$. In the following we show $\tilde{c}\in[\![\phi]\!]$. This is sufficient to show c also in $[\![\phi]\!]$ because the later is upward closed by lemma (2).
The condition ψ' does not refer to elements of self$\cdot Y$. Define the substitution ρ_3^i as $\left\{\text{self}\cdot x\leftarrow x^i\middle|\ x^i\in Y^i\right\}$. By construction

$$\psi'\left[\tilde{u}_1,\ldots,\tilde{u}_i\left[(\rho_3^i)_{inv}\right],\ldots,\tilde{u}_n,u_i'\right]$$

Define substitutions ρ_1^i and ρ_2^j to respectively equal $\left\{\text{self}\cdot x^{next}\leftarrow x^i\middle|\ x^i\in Y^i\right\}$ and $\left\{\text{self}\cdot x\leftarrow x^j\middle|\ x^j\in Y^j\right\}$. Recall that $\left(u,u_i',u_j'\right)\models\theta_1$ for each u_j' in c_1' and c_2'. This implies $\left(\theta_1\left[\rho_1^i\right]\left[\rho_2^j\right]\right)\left[\tilde{u}_i\left[(\rho_3^i)_{inv}\right],u_i',\tilde{u}_j\right]$ for each $\tilde{u}_j$ in $\tilde{c}_1$ and $\tilde{c}_2$. Also recall $\tilde{u}_i$, respectively u_i', states process i to be at local state q, respectively q'. Therefore:

$$\left(\psi'\wedge\bigwedge_{j\in\overline{n}\setminus\{i\}}\theta_1^{\bullet}\left[\rho_1^i\right]\left[\rho_2^j\right]\right)\left[\tilde{u}_1,\ldots,\tilde{u}_i\left[(\rho_3^i)_{inv}\right],\ldots,\tilde{u}_n,u_i'\right]$$

That is:

$$\left(\exists Y^i.\left(\psi'\wedge\bigwedge_{j\in\overline{m}\setminus\{i\}}\theta_1^{\bullet}\left[\rho_2\right]\left[\rho_3\right]\left[\rho_4^j\right]\right)\right)\left[\tilde{u}_1,\ldots,\tilde{u}_i\left[(\rho_3^i)_{inv}\right],\ldots,\tilde{u}_n\right]$$

Hence, $\tilde{c}\models\psi$ and $\tilde{c}\in[\![\phi]\!]$.

Proof (lemma 4) We show both directions

– First direction (if). Given c in $[\![\phi']\!]$, we show c in $[\![\phi]\!]$. since c is in $[\![\phi']\!]$, we get that $c=[u_1,\ldots,u_n]$ and (up to renaming) $\psi'\left[u_1,\ldots,u_{m'}\right]$ holds. From (3) we deduce the existence of an injection h in $\mathcal{H}(\phi,\phi')$ such that $\psi^h\left[u_1,\ldots,u_{m'}\right]$ holds. In other words, the configuration $\left[u_{h(1)},\ldots,u_{h(m)}\right]$ is in $[\![\phi]\!]$. Clearly $\left[u_{h(1)},\ldots,u_{h(m)}\right]\preceq c$, which results in $c\in[\![\phi]\!]$.

– Second direction (only if)
Suppose (3) does not hold. We find a configuration c in $[\![\phi']\!]$ but not in $[\![\phi]\!]$. The fact that (3) does not hold entails the existence of at least a substitution to $y_1,\ldots,y_k$, for which $\psi'(y_1,\ldots,y_k)$ holds but not $\bigvee_{h\in\mathcal{H}(\phi,\phi')}\psi^h(y_1,\ldots,y_k)$. Reformulate such a substitution into the configuration $c=[u_1,\ldots,u_m]$ of size m. Clearly, $\psi^h\,[u_1,\ldots,u_m]$ does not hold for any injection $h\in\mathcal{H}(\phi,\phi')$. The configuration c is therefore outside $[\![\phi]\!]$.

Now, we explain the computation of *Pre* for both rendez-vous and Broadcast. In the following, we assume a constraint $\phi'=(m,\psi')$.

Binary Communication We assume Y^{m+1} and Y^{m+2} to be fresh copies of Y. We define the following four substitutions

$$\rho_1^i=\{\text{self}\cdot x^{next}\leftarrow x^i \mid x^i\in Y^i\} \qquad \rho_2^j=\{\text{other}\cdot x^{next}\leftarrow x^j \mid x^j\in Y^j\}$$
$$\rho_3^i=\{\text{self}\cdot x\leftarrow x^i \mid x^i\in Y^i\} \qquad \rho_4^j=\{\text{other}\cdot x\leftarrow x^j \mid x^j\in Y^j\}$$

The set $Pre_t(\phi')$ contains the constraints $\phi=(n,\psi_{i,j})$ with

$$\psi_{i,j}=\left(\exists(Y^i\cup Y^j).\ \psi'\wedge\theta_1^\bullet\,[\rho_1^i]\left[\rho_2^j\right]\right)[\rho_3^i]\left[\rho_4^j\right]$$

The tuple (n,i,j) can take the following values:

– $(n,i,j)=(m,i,j)$ and $i,j\in\overline{m}$ and $i\neq j$.
– $(n,i,j)=(m+1,i,m+1)$ and $i\in\overline{m}$
– $(n,i,j)=(m+1,m+1,j)$ and $j\in\overline{m}$
– $(n,i,j)=(m+2,m+1,m+2)$.

Broadcast We let $Y^{k^{next}}$, for $k:1\leq k\leq m+1$, be $m+1$ fresh copies of Y. We define the following five substitutions

$$\rho_1^i=\left\{\text{self}\cdot x^{next}\leftarrow x^{i^{next}} \mid x^{i^{next}}\in Y^{i^{next}}\right\} \qquad \rho_2^j=\left\{\text{other}\cdot x^{next}\leftarrow x^{j^{next}} \mid x^{j^{next}}\in Y^{j^{next}}\right\}$$
$$\rho_3^i=\{\text{self}\cdot x\leftarrow x^i \mid x^i\in Y^i\} \qquad \rho_4^j=\{\text{other}\cdot x\leftarrow x^j \mid x^j\in Y^j\}$$
$$\rho_5=\left\{x^k\leftarrow x^{k^{next}} \mid x^{k^{next}}\in Y^{k^{next}},\ k:1\leq k\leq n\right\}$$

The set $Pre_t(\phi')$ contains the constraints $\phi=(n,\psi_i)$ with

$$\psi_i=\exists(\bigcup_{k\in\overline{n}}Y^{k^{next}}).\ \left(\psi'\,[\rho_5]\wedge\bigwedge_{j\in\overline{n}\setminus\{i\}}\theta_1^\bullet\,[\rho_1^i]\left[\rho_2^j\right][\rho_3^i]\left[\rho_4^j\right]\right)$$

The tuple (n,i) can take the following values:

– $(n,i)=(m,i)$ and $i\in\overline{m}$.
– $(n,i)=(m+1,m+1)$.

For a a binary communication or a broadcast t, one can prove $[\![Pre_t(\phi')]\!]=\left\{c\overset{t}{\rightsquigarrow}c' \mid c'\in[\![\phi]\!]\right\}$ by using straightforward generalizations of the proofs for transitions with existential and universal conditions.