Parameterized Verification of Infinite-state Processes with Global Conditions

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

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

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

1 Introduction

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

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

perform the transition only if this condition is satisfied. In an existential condition we require that *some* (rather than *all*) processes satisfy the condition. In addition to these classes of transitions, processes may communicate through broadcast, rendez-vous, and shared variables.

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

To check safety properties, we perform backward reachability analysis using gap-order constraints as a symbolic representation of upward closed sets of configurations. In the analysis, we consider a transition relation which is an over-approximation of the one induced by the parameterized system. To do that, we modify the semantics of universal quantifiers by eliminating the processes which violate the given condition. For instance in the above example, process i is always allowed to take the transition. However, when performing the transition, we eliminate each process j where the value of y is smaller 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, termination is guaranteed in some restricted cases such as for systems with existential or universal global conditions but with at most one local integer variable.

In order to test our method we have selected a collection of challenging protocols in which integer variables are used either as identifiers, priorities, local clocks, or time-stamps. Almost all of the examples are outside the class for

which termination is guaranteed. In particular, we automatically verify safety properties for parameterized versions of the following algorithms:

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

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

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

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

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

Parameterized versions of Lamport's bakery algorithm have been tested using a semi-automated verification method based on *invisible invariants* in [6], with the help of *environmental abstractions* for a formulation with atomicity conditions in [10], and using heuristics to discover *indexed predicates* in [17]. A parameterized formulation of the Ricart-Agrawala algorithm has been verified semi-automatically in [21], where the STeP prover is used to discharge some of the verification conditions needed in the proof. We are not aware of other

attempts of fully automatic verification of parameterized versions of the Ricart-Agrawala algorithm or of the distributed version (with no atomicity assumptions) of Lamport's bakery algorithm.

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

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

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

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

2 Preliminaries

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

For a finite set A, we write a multiset over A as a list $[a_1, a_2, \ldots, a_n]$, where $a_i \in A$ for each $i : 1 \leq i \leq n$. We use $a \in A$ to denote that $a = a_i$ for some $i : 1 \leq i \leq n$. For multisets $M_1 = [a_1, \ldots, a_m]$ and $M_2 = [b_1, \ldots, b_n]$, we use $M_1 \bullet M_2$ to denote the union (sum) of M_1 and M_2 (i.e., $M_1 \bullet M_2 = [a_1, \ldots, a_m, b_1, \ldots, b_n]$).

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

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

3 Parameterized Systems

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

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

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

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

where $q, q' \in Q$ and θ is either a *local* or a *global condition*. Intuitively, the process which makes the transition changes its local state from q to q'. In the

meantime, the values of the local variables of the process are updated according to θ. Below, we describe how we define local and global conditions.

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

To define local conditions, we introduce the set X^{next} which contains the *next-value* versions of the variables in X. A variable $x^{next} \in X^{next}$ represents the next value of $x \in X$. A *local condition* is a formula in $\mathbb{F}(X \cup X^{next})$. The formula specifies how local variables of the current process are updated with respect to their current values.

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 $\texttt{self}{\cdot}x$ and $\texttt{other}{\cdot}x$. We define the sets $\texttt{self}{\cdot}Y = \{\texttt{self}{\cdot}x | x \in Y\}$ and $\texttt{other}{\cdot}Y = \{\texttt{other}{\cdot}x | x \in Y\}$. The sets $\texttt{self}{\cdot}X$, $\texttt{other}{\cdot}X^{next}$, etc, are defined in the obvious manner. A *global condition* θ is of one of the following two forms:

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

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

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

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

4 Transition System

We describe the transition system induced by a parameterized system.

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

First, a parameterized system $\mathcal{P} = (Q, X, T)$ induces a transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$ as follows. A configuration is defined by the local states and

the values of the local variables in the processes. Formally, a *local variable state* v is a mapping from X to $\mathcal{B} \cup \mathcal{N}$ which respects 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 $q' \in Q - \{q\}$. 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 $\theta \in \mathbb{F}(\texttt{self}\cdot X \cup \texttt{other}\cdot Y \cup \texttt{self}\cdot X^{next})$ (representing a global condition). The roles of v and v' are the same as for local conditions. We recall that a global condition also checks states of all (or some) of the other processes. Here, u_1 represents the local state and variables of one such a process. We write $(v, v', u_1) \models \theta$ to denote the validity of the formula $\theta\,[\rho]\,[\rho']\,[\rho_1]$ where the substitutions are defined by $\rho := \{\texttt{self}\cdot x \leftarrow v(x) |\ x \in X\}$, $\rho' := \{\texttt{self}\cdot x^{next} \leftarrow v'(x) |\ x \in X\}$, and $\rho_1 := \{\texttt{other}\cdot x \leftarrow u_1(x) |\ x \in Y\}$. 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 $t \in T$.

5 Safety Properties

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

We define an ordering on configurations as follows. Consider two configurations, $c = [u_1 \cdot \ldots \cdot u_m]$ and $c' = [u'_1 \cdot \ldots \cdot u'_n]$, 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 $h : \overline{m} \rightarrow \overline{n}$ 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. $v_i(x) = \mathit{true}$ iff $v'_{h(i)}(x) = \mathit{true}$ for each $x \in X_{\mathcal{B}}$.
3. $v_i(x) = v_j(y)$ iff $v'_{h(i)}(x) = v'_{h(j)}(y)$, for each $x, y \in X_{\mathcal{N}}$.
4. $v_i(x) <_k v_j(y)$ 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. [22]), that checking safety properties (expressed as regular languages) can be translated into instances of the coverability problem. Therefore, checking safety properties amounts to solving PAR-COV (i.e., to the reachability of upward closed sets).

6 Approximation

In this section, we introduce an over-approximation of the transition relation of a parameterized system. The aim of the over-approximations is to derive a

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

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

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

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

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

APRX-PAR-COV

Instance

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

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

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

7 Backward Reachability Analysis

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

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

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

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

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

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

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

Observe that, in order to implement the scheme (i.e., transform it into an algorithm), we need to be able to (i) compute *Pre*; (ii) check for entailment

between constraints; and (iii) check for emptiness of $(\mathit{Init} \bigcap [\![\phi]\!]) \neq \emptyset$ for a constraint ϕ.

8 Constraint Operations

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

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

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

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

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

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

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

Intersection with Initial States For a constraint $\phi = (m, \psi)$, we have $(\mathit{Init} \bigcap [\![\phi]\!]) \neq \emptyset$ 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 Communication In *binary communication* two processes perform a *rendez-vous*, changing states simultaneously. Such a transition can be encoded by considering a more general form of existential global conditions than the one allowed in Section 3. More precisely we take θ_1 in the definition of an existential global conditions (see (2)), to be a formula in the set

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

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

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

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

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

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

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

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

10 Experimental Results

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

Model	Iterations		Time		Safe		Trace	
	R	I	R	I	R	I	R	I
Simplified Bakery Alg.	6	6	0.8s	0.3s	√	√		
Lamport's Bakery Alg.	9	9	2.1s	2s	√	√		
Bogus Bakery	10	6	0.8s	11s			√	√
Ticket Mutex Alg.	9	8	0.3s	1.6s	√	√		
Ricart-Agrawala Distr. Mutex Alg.	9	11	3.4s	2mn40s	√	√		
Lamport's Distr. Mutex Alg.	21	27	9mn19s	146mn	√	√		

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

11 Conclusion and Future Work

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

References

1. P. A. Abdulla, K. Čerāns, B. Jonsson, and T. Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.
2. P. A. Abdulla and G. Delzanno. On the coverability problem for constrained multiset rewriting. In *Proc. AVIS'06*, 2006.
3. P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Regular model checking without transducers In *Proc. TACAS '07*, 2007. To appear.
4. P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Regular model checking made simple and efficient. In *Proc. CONCUR 2002*, volume 2421 of *LNCS*, 2002.
5. G. Andrews. *Foundations of Multithreaded, Parallel, and Distributed Programming.* Addison Wesley, 2000.
6. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In *Proc. CAV 2001*, volume 2102 of *LNCS*, 2001.
7. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc. CAV 2003*, volume 2725 of *LNCS*, 2003.
8. M. Bozzano and G. Delzanno. Beyond parameterized verification. In *Proc. TACAS '02*, volume 2280 of *LNCS*, 2002.

9. T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables. *ACM TOPLAS*, 21(4):747–789, 1999.
10. E. Clarke, M. Talupur, and H. Veith. Environment abstraction for parameterized verification. In *Proc. VMCAI '06*, volume 3855 of *LNCS*, pages 126–141, 2006.
11. G. Delzanno. Automatic verification of cache coherence protocols. In *Proc. CAV 2000*, volume 1855 of *LNCS*, 2000.
12. E. Emerson and K. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Proc. LICS' 98*, 1998.
13. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. LICS' 99,* 14th *IEEE Int. Symp. on Logic in Computer Science*, 1999.
14. L. Fribourg and J. Richardson. Symbolic verification with gap-order constraints. In *Proc. LOPSTR'96*, volume 1207 of *LNCS*, 1997.
15. S. M. German and A. P. Sistla. Reasoning about systems with many identical processes. *Journal of the ACM*, 39(3):675–735, 1992.
16. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *TCS*, 256:93–112, 2001.
17. S. K. Lahiri and R. E. Bryant. Indexed predicate discovery for unbounded system verification. In *Proc. CAV 2004*, pages 135–147, 2004.
18. L. Lamport. A new solution of dijkstra's concurrent programming problem. *Commun. ACM*, 17(8):453–455, 1974.
19. P. Revesz. A closed form evaluation for datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.
20. G. Ricart and A. K. Agrawal. An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9–17, 1981.
21. E. Sedletsky, A. Pnueli, and M. Ben-Ari. Formal verification of the ricart-agrawala algorithm. In *Proc. FSTTCS'00*, 2000.
22. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86*, June 1986.

A Computing *Pre*

To define *Pre*, we consider existentially quantified formulas. Consider sets $B \subseteq A$ and a formula $\psi \in \mathbb{F}(A)$. We interpret the formula $\exists B.\psi$ in the standard manner. The following follows immediately from closeness of gap formulas under existential quantification (shown in [19]).

Lemma 5. *For each $\psi \in \mathbb{F}(A)$ and $B \subseteq A$, there is a $\psi' \in \mathbb{F}(A)$ such that $\psi' = \exists B.\psi$.*

Recall that $Y = X \cup Q$. For a constraint $\phi = (m, \psi)$, we define $Pre(\phi) = \bigcup_{1 \leq i \leq m} \bigcup_{t \in T} Pre_{t,i}(\phi)$, i.e., we compute the set of predecessor constraints with respect to each process i in the constraint, and with respect to each transition rule $t \in T$. In the following, assume t to be a transition rule of the form (1). Let $Y^{\bullet} = \{x^{\bullet} | \ x \in Y\}$ be a fresh set of variables which do not occur in ϕ or t. We compute $Pre_{t,i}(\phi)$ depending on the condition which appears in t as follows:

- If θ is a local condition. Define

$$\theta' := \theta \wedge q \wedge {q'}^{next} \wedge \bigwedge_{q_1 \neq q} \neg q_1 \wedge \bigwedge_{q_2 \neq q'} \neg q_2^{next} \tag{3}$$

  In other words, we add to θ the information that the original local state of the process is q while its next state is q'. We define $Pre_{t,i}(\phi)$ to be the singleton which contains the constraint (m, ψ') where

$$\psi' = \exists Y^{\bullet}. \left(\psi\left[\rho_1\right] \wedge \theta'\left[\rho_2\right]\left[\rho_3\right]\right) \tag{4}$$

  and the substitutions contain the following elements for each $x \in Y$:

$$\rho_1 : x^i \leftarrow x^{\bullet} \qquad \rho_2 : x^{next} \leftarrow x^{\bullet} \qquad \rho_3 : x \leftarrow x^i$$

  Here, process i is the one performing the transition t, and therefore it is the only process which may change state. Also, we are dealing with a local condition, and hence only the local variables of i are checked during t. Since we are computing predecessors, the constraint represents the local states and values of the local variables after performing t. Therefore we match each variable x^i which occurs in ϕ with the next-value variable x^{next} in t. We do that through replacing both variables by the corresponding fresh variable $x^{\bullet}$ (through the substitutions ρ_1 and ρ_2). To maintain the property that constraints only contain variables in the copy sets Y^i, we replace each variable x (which represents its value before performing t) by the corresponding variable x^i in Y^i (this is done through the substitution ρ_3).
- If θ is an existential global condition of the form of (2). Define

$$\theta_1' := \theta_1 \wedge \mathtt{self}\cdot q \wedge \mathtt{self}\cdot {q'}^{next} \wedge \bigwedge_{q_1 \neq q} \neg \mathtt{self}\cdot q_1 \wedge \bigwedge_{q_2 \neq q'} \neg \mathtt{self}\cdot q_2^{next}$$

  The formula θ_1' plays the same role as θ' in the case of a local condition. The set $Pre_{t,i}(\phi)$ contains the following constraints:

- the constraint (m, ψ_j) for each $j : 1 \leq j \neq i \leq m$, where

$$\psi_j = \exists Y^{\bullet}. \left(\psi\left[\rho_1\right] \wedge \theta_1'\left[\rho_2\right]\left[\rho_3\right]\left[\rho_4\right]\right) \tag{5}$$

  and the substitutions contain the following elements for each $x \in Y$:

$$\begin{array}{ll} \rho_1 : x^i \leftarrow x^{\bullet} & \rho_2 : \mathtt{self} \cdot x^{next} \leftarrow x^{\bullet} \\ \rho_3 : \mathtt{self} \cdot x \leftarrow x^i & \rho_4 : \mathtt{other} \cdot x \leftarrow x^j \end{array}$$

  The substitutions ρ_1, ρ_2, ρ_3, have similar interpretations to the case of local conditions. In addition, we recall that an existential global condition requires that there is another "witness" process which satisfies the existentially quantified condition θ_1. This other process is represented by index j in the constraint. The substitution ρ_4 has a similar role to ρ_3. Notice that we get one constraint for each possible witness process, i.e., for each $j : 1 \leq i \neq j \leq m$.
- the constraint $(m+1, \psi_{m+1})$ where

$$\psi_{m+1} = \exists Y^{\bullet}. \left(\psi\left[\rho_1\right] \wedge \theta_1'\left[\rho_2\right]\left[\rho_3\right]\left[\rho_4\right]\right) \tag{6}$$

  and the substitutions contain the following elements for each $x \in Y$:

$$\begin{array}{ll} \rho_1 : x^i \leftarrow x^{\bullet} & \rho_2 : \mathtt{self} \cdot x^{next} \leftarrow x^{\bullet} \\ \rho_3 : \mathtt{self} \cdot x \leftarrow x^i & \rho_4 : \mathtt{other} \cdot x \leftarrow x^{m+1} \end{array}$$

  The difference compared to the previous sub-case is that the witness process is not one of the processes mentioned in the constraint. Observe that the size of the constraint increases by one in this case. This makes the sizes of constraints which arise in the reachability analysis unbounded in general.

– If θ is a universal global condition, then we define θ_1' as in the case of an existential global condition. We define $Pre_{t,i}(\phi)$ to be the singleton which contains the constraint (m, ψ') where

$$\psi' = \exists Y^{\bullet}. \left(\bigwedge_{1 \leq j \neq i \leq m} \left(\psi\left[\rho_1\right] \wedge \theta_1'\left[\rho_2\right]\left[\rho_3\right]\left[\rho_4^j\right]\right)\right) \tag{7}$$

and the substitutions contain the following elements for each $x \in Y$ and $1 \leq j \neq i \leq m$:

$$\begin{array}{ll} \rho_1 : x^i \leftarrow x^{\bullet} & \rho_2 : \mathtt{self} \cdot x^{next} \leftarrow x^{\bullet} \\ \rho_3 : \mathtt{self} \cdot x \leftarrow x^i & \rho_4^j : \mathtt{other} \cdot x \leftarrow x^j \end{array}$$

All the substitutions have identical roles to the case of an existential global condition. The difference is that we here require that all other processes satisfy the universally quantified condition θ_1, and hence we conjunct over $1 \leq j \neq i \leq m$.

In the following, we use v_i (resp. u_i) to refer to the *local variable state* (resp. *process state*) of process i. In other words, v_i (resp. u_i) is a mapping from X_i (resp. Y_i) to $\mathcal{B} \cup \mathcal{N}$. Given a substitution $\rho : x^i \leftarrow x^j$, we mean by ρ^{-1} the substitution $\rho^{-1} : x^j \leftarrow x^i$. In this context, $u_j\left[\rho^{-1}\right]$ is the process state of process i mapping each $x^i \in Y_i$ to $u_j(x^j)$. Intuitively, this corresponds to the variables of process i taking the same values as the variables of process j.

Let $\Phi = Pre_t(\phi')$, then the following two lemmas hold. We use in the following the same notations as those from the definition of *Pre*.

Lemma 6. $[\![\Phi]\!] \subseteq \left\{c \middle| \, \exists c' \in [\![\phi']\!] \,.\, c \overset{t}{\rightsquigarrow} c'\right\}$.

Proof. We assume t to be a transition rule of form (1). Let c be a configuration in $[\![\Phi]\!]$, i.e., there is $\phi \in \Phi$ such that $c \in [\![\phi]\!]$. By hypothesis, ϕ is obtained from ϕ' via the application of the predecessor operator. In order to show the existence of a configration c' in $[\![\phi']\!]$ such that $c \overset{t}{\rightsquigarrow} c'$, we proceed by case analysis, on the form of the condition in t.

We omit the case where the guard is a local condition. A local condition can be viewed as a special case of universally quantified conditions, where no $\texttt{other}\cdot x$ appears in θ; i.e. the values of the variables belonging to processes other than the one firing the transition are not constrained.

Exists For $\phi' = (m, \psi)$, let ϕ be in $Pre_{t,i}(\phi')$ for some i in $\overline{m}$. According to the definition of *Pre*, there are two possible cases: i) either $\phi = (m, \psi_j)$ for some j in $\overline{m}$ and different from i, or ii) $\phi = (m+1, \psi_{m+1})$.

i) Suppose $\phi = (m, \psi_j)$ for some j in $\overline{m}$ and different from i.
Let c be a configuration in $[\![\phi]\!]$. We can, up to a renaming, assume c to be of the form:

$$c = [u_1, \ldots, u_n]$$

where $u_1, \ldots u_n$ are process states verifying: $[u_1, \ldots, u_m] \models \psi_j$.
From the definition of ψ_j in 5, we deduce the existence of a process state $u^\bullet$ such that:

$$[u_1, \ldots, u_m, u^\bullet] \models \psi\,[\rho_1] \wedge \theta'_1\,[\rho_2]\,[\rho_3]\,[\rho_4] \tag{8}$$

where:

$$\rho_1 : x^i \leftarrow x^\bullet \qquad \rho_2 : \texttt{self}\cdot x^{next} \leftarrow x^\bullet$$
$$\rho_3 : \texttt{self}\cdot x \leftarrow x^i \qquad \rho_4 : \texttt{other}\cdot x \leftarrow x^j.$$

Define the configuration c' to be:

$$c' = [u'_1, \ldots, u'_n]$$

where $u'_i = u^\bullet\left[\rho_1^{-1}\right]$ and $u'_k = u_k$ for all other k in $\overline{n}$. Intuitively, the variables of the processes in c' are mapped to the same values as in c except for the variables of process i which are mapped according to $u^\bullet$.
Observe that, because of (8), we have $[u_1, \ldots, u_m, u^\bullet] \models \psi\,[\rho_1]$. This means $[u'_1, \ldots, u'_m] \models \psi$, in other words, $[u'_1, \ldots, u'_m]$ is in $[\![\phi']\!]$.

Since $[u'_1, \dots, u'_m] \preceq c'$ and because $[\![\phi']\!]$ is by definition upward closed, we get $c' \in [\![\phi']\!]$.
Again because of (8), we have:

$$[u_1, \dots, u_m, u^\bullet] \models \theta'_1 [\rho_2] [\rho_3] [\rho_4]$$

This has two consequences, (i) u_i, resp. u'_i, states process i to be at q, resp. q', and (ii) there is a j in $\overline{m}$ such that $(u_i, u'_i, u_j) \models \theta_1$. In other words, the rule t can be fired by process i in $[u_1, \dots, u_m]$, leading to $[u'_1, \dots, u'_m]$.
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 $c \xrightarrow{t} c'$, and therefore $c \overset{t}{\rightsquigarrow} c'$.

ii) Suppose $\phi = (m+1, \psi_{m+1})$.
Let c be a configuration in $[\![\phi]\!]$. We can, up to a renaming, assume c to be of the form:

$$c = [u_1, \dots, u_n]$$

where $u_1, \dots, u_n$ are process states verifying: $[u_1, \dots, u_{m+1}] \models \psi_{m+1}$.
From the definition of ψ_{m+1} in (5), we deduce the existence of a process state $u^\bullet$ such that:

$$[u_1, \dots, u_{m+1}, u^\bullet] \models \psi [\rho_1] \wedge \theta'_1 [\rho_2] [\rho_3] [\rho_4] \tag{9}$$

where:

$$\rho_1 : x^i \leftarrow x^\bullet \qquad \rho_2 : \mathtt{self} \cdot x^{next} \leftarrow x^\bullet$$
$$\rho_3 : \mathtt{self} \cdot x \leftarrow x^i \qquad \rho_4 : \mathtt{other} \cdot x \leftarrow x^{m+1}.$$

Define the configuration c' to be:

$$c' = [u'_1, \dots, u'_n]$$

where $u'_i = u^\bullet \left[\rho_1^{-1}\right]$ and $u'_k = u_k$ for all other k in $\overline{n}$. Intuitively, the variables of the processes in c' are mapped to the same values as in c except for the variables of process i which are mapped according to $u^\bullet$.
Observe that, because of (9), we have $[u_1, \dots, u_{m+1}, u^\bullet] \models \psi [\rho_1]$. Since ψ does not refer to variables with indice $m+1$, we can write $[u_1, \dots, u_m, u^\bullet] \models \psi [\rho_1]$. This means $[u'_1, \dots, u'_m] \models \psi$, in other words, $[u'_1, \dots, u'_m]$ is in $[\![\phi']\!]$.
Since $[u'_1, \dots, u'_m] \preceq \left[u'_1, \dots, u'_{m+1}\right] \preceq c'$ and because $[\![\phi']\!]$ is by definition upward closed, we get c' in $[\![\phi']\!]$.
Again because of (9), we have:

$$[u_1, \dots, u_{m+1}, u^\bullet] \models \theta'_1 [\rho_2] [\rho_3] [\rho_4]$$

This has two consequences, (i) the local process state stated by u_i is q while the one stated by u'_i is q', and (ii) $(u_i, u'_i, u_{m+1}) \models \theta_1$. In other words, the rule t can be fired by process i with the process $m+1$ as a witness, obtaining $\left[u'_1, \dots, u'_{m+1}\right]$ from $[u_1, \dots, u_{m+1}]$.
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 $c \xrightarrow{t} c'$, and therefore, $c \overset{t}{\rightsquigarrow} c'$.

Forall For $\phi' = (m, \psi)$, let ϕ be in $Pre_{t,i}(\phi')$ for some i in $\overline{m}$. By definition of Pre, ϕ is of the form (m, ψ').

Let c be a configuration in $[\![\phi]\!]$. We can, up to a renaming, assume c to be of the form:

$$c = [u_1, \ldots, u_n] = c_1 \bullet u_i \bullet c_2$$

where c_1 and c_2 are configurations, and $u_1, \ldots u_n$ process states verifying $[u_1, \ldots, u_m] \models \psi'$.

From the definition of ψ' in (7), we deduce the existence of a process state $u^\bullet$ such that:

$$[u_1, \ldots, u_m, u^\bullet] \models \psi\,[\rho_1] \wedge \bigwedge_{1 \leq j \neq i \leq m} \theta_1'\,[\rho_2]\,[\rho_3]\left[\rho_4^j\right] \qquad (10)$$

where:

$$\rho_1 : x^i \leftarrow x^\bullet \qquad \rho_2 : \mathtt{self} \cdot x^{next} \leftarrow x^\bullet$$
$$\rho_3 : \mathtt{self} \cdot x \leftarrow x^i \qquad \rho_4^j : \mathtt{other} \cdot x \leftarrow x^j.$$

We define for every k in $\overline{m}$ the process state u_k' to be equal to u_k, except for $k = i$ in which case $u_i' = u^\bullet\left[\rho_1^{-1}\right]$; i.e. u_i' maps the vriables of process i to the same values as those defined by $u^\bullet$.

Observe that (10) implies $[u_1, \ldots, u_m, u^\bullet] \models \bigwedge_{1 \leq j \neq i \leq m} \theta_1'\,[\rho_2]\,[\rho_3]\left[\rho_4^j\right]$. This has two consequences, (i) u_i, resp. u_i', states process i to be at q, resp. q', and (ii) $(u_i, u_i', u_j) \models \theta_1$ for each $j \neq i$ in $\overline{m}$.

The consequence (ii) implies $c_1 \ominus (\theta_1, u_i, u_i') = [u_1, \ldots, u_{i-1}]$ and $c_2 \ominus (\theta_1, u_i, u_i') = [u_{i+1}, \ldots, u_m] \bullet c''$ for some configuration c''

Define two new configurations $\tilde{c}$ and c':

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

$$c' = (c_1 \ominus (\theta_1, u_i, u_i')) \bullet u_i' \bullet (c_2 \ominus (\theta_1, u_i, u_i')).$$

We get by construction that $\tilde{c} \xrightarrow{t} c'$, and hence $c \stackrel{t}{\rightsquigarrow}_1 c'$. In order to conclude, we show c' in $[\![\phi']\!]$.

Again because of (10), we have that $[u_1, \ldots, u_m, u^\bullet] \models \psi\,[\rho_1]$. This implies $[u_1', \ldots, u_m'] \models \psi$. We therefore deduce $[u_1', \ldots, u_m']$ to be in $[\![\phi']\!]$.

Since $[u_1', \ldots, u_m'] \preceq c'$, and because $[\![\phi']\!]$ is by definition upward closed, we conclude that c' is in $[\![\phi']\!]$.

Lemma 7. $\left\{c |\, \exists c' \in [\![\phi']\!] \,.\, c \stackrel{t}{\rightsquigarrow} c'\right\} \subseteq [\![\Phi]\!] \cup [\![\phi']\!]$.

Proof. We assume t to be a transition rule of form (1). Suppose $\phi' = (m, \psi)$. Up to a renaming, we can assume any configuration c' in $[\![\phi']\!]$ to be of the form:

$$c' = [u_1', \ldots, u_n']$$

for process states $u_1', \ldots u_n'$ with $[u_1', \ldots, u_m'] \models \psi$.

For any configuration c verifying $c \overset{t}{\rightsquigarrow} c'$, we show $c \in [\![\phi']\!]$ or exhibit a constraint ϕ from the set Φ computed by *Pre* such that $c \in [\![\phi]\!]$. We proceed by case analysis on the condition in t. The case where the guard is a local condition is ommited since it can be viewed as a special case of universally quantified conditions.

We use the same notations as those in the definition of *Pre*.

Exist The transitions involving existential conditions are not over-approximated by definition; i.e. $c \overset{t}{\rightsquigarrow} c'$ iff $c \xrightarrow{t} c'$.

Based on the indice i of the process firing the transition, and on the indice j of the witness satisfying the existential condition, we differentiate three cases: (i) i and j in $\overline{m}$, (ii) only i in $\overline{m}$, the witness has index j outside $\overline{m}$, i.e. in $m+1 \leq j \leq n$, and (iii) the process i is outside $\overline{m}$.

i) Suppose $i, j \in \overline{m}$.
By definition of the transitions with an existential global condition, c is of the form :

$$c = [u_1, \ldots, u_n]$$

where $u_1, \ldots u_n$ satisfy the following:
- $u_k = u'_k$ for every k in $\overline{n} - \{i\}$
- u_i, resp. u'_i, states process i to be at q, resp. q'.
- $(u_i, u'_i, u_j) \models \theta_1$ for some $j \in \overline{m}$.

Now, we take a set of fresh copies $Y^\bullet$ of Y, we define $u^\bullet$ to take the same values as u'_i; i.e. $u^\bullet = u'_i\,[\rho_1]$ where $\rho_1 : x^i \leftarrow x^\bullet$.
By definition of $u'_1, \ldots, u'_n$, we have $[u'_1, \ldots, u'_m] \models \psi$, it is then easy to see that:

$$[u_1, \ldots, u_m, u^\bullet] \models \psi\,[\rho_1]$$

Aslo, observe that $(u_i, u'_i, u_j) \models \theta_1$ means θ_1 evaluates to *true* when the occurences of $\texttt{self} \cdot x$ are replaced ($\rho_3 : \texttt{self} \cdot x \leftarrow x^i$) by their value in u_i, the occurences of $\texttt{self} \cdot x^{next}$ are replaced ($\rho_2 : \texttt{self} \cdot x^{next} \leftarrow x^\bullet$) by their value in u'_i and the occurences of $\texttt{other} \cdot x$ are replaced ($\rho_4 : \texttt{other} \cdot x \leftarrow x^j$) by their value in u_j. In other words:

$$[u_1, \ldots, u_m, u^\bullet] \models \theta_1\,[\rho_2]\,[\rho_3]\,[\rho_4]$$

We know u_i, resp. u'_i, states process i to be at local state q, resp. q'. Therefore, $\theta'_1\,[\rho_2]\,[\rho_3]\,[\rho_4]$ evaluates to *true* when applied to $[u_1, \ldots, u_m, u^\bullet]$.
We have now :

$$[u_1, \ldots, u_m, u^\bullet] \models \psi\,[\rho_1] \wedge \theta'_1\,[\rho_2]\,[\rho_3]\,[\rho_4]$$

That is:

$$[u_1, \ldots, u_m] \models \exists Y^\bullet . \psi\,[\rho_1] \wedge \theta'_1\,[\rho_2]\,[\rho_3]\,[\rho_4]$$

We deduce $[u_1, \ldots, u_m]$ is in $[\![\phi]\!]$, where $[\![\phi]\!] = (m, \psi_j)$. Since $[u_1, \ldots, u_m] \preceq c$, and because $[\![\phi]\!]$ is upward closed, we conclude that the configuration c is also in $[\![\phi]\!]$.

– Suppose only i is in $\overline{m}$.
By definition of the transitions with an existential global condition, c is of the form :

$$c = [u_1, \ldots, u_n]$$

where the process states $u_1, \ldots u_n$ satisfy the following:
- $u_k = u'_k$ for every k in $\overline{n} - \{i\}$
- u_i, resp. u'_i, states process i to be at q, resp. q'
- $(u_i, u'_i, u_j) \models \theta_1$ for some witness u_j with $m < j \leq n$.

We define $\tilde{c}$ to be the configuration containing the witness with indice j together with the m first processes of c. Formally, assuming the substitution $\rho : x^j \leftarrow x^{m+1}$:

$$\tilde{c} = [\tilde{u}_1, \ldots, \tilde{u}_m] \bullet \tilde{u}_{m+1}$$

with $\tilde{u}_{m+1} = u_j\,[\rho]$ while $u_k = \tilde{u}_k$ for all k in $\overline{m}$.
Now, let $Y^\bullet$ be a set of fresh copies of Y. We define $u^\bullet$ to take the same values as u'_i; i.e. $u^\bullet = u'_i\,[\rho_1]$ where $\rho_1 : x^i \leftarrow x^\bullet$.
By definition of $u'_1, \ldots, u'_m$, we have $[u'_1, \ldots, u'_m] \models \psi$, it is then easy to see that $[u_1, \ldots, u_m, u^\bullet] \models \psi\,[\rho_1]$, that is to say:

$$[\tilde{u}_1, \ldots, \tilde{u}_{m+1}, u^\bullet] \models \psi\,[\rho_1]$$

Aslo, observe that $(u_i, u'_i, u_j) \models \theta_1$ means θ_1 evaluates to *true* when the occurences of $\texttt{self} \cdot x$ are replaced ($\rho_3 : \texttt{self} \cdot x \leftarrow x^i$) by their value in u_i, the occurences of $\texttt{self} \cdot x^{next}$ are replaced ($\rho_2 : \texttt{self} \cdot x^{next} \leftarrow x^\bullet$) by their value in u'_i and the occurences of $\texttt{other} \cdot x$ are replaced ($\tilde{\rho_4} : \texttt{other} \cdot x \leftarrow x^j$) by their value in u_j.
In other words: $[u_1, \ldots, u_m, u_j, u^\bullet] \models \theta_1\,[\rho_2]\,[\rho_3]\,[\tilde{\rho_4}]$, that is to say:

$$[\tilde{u}_1, \ldots, \tilde{u}_{m+1}, u^\bullet] \models \theta_1\,[\rho_2]\,[\rho_3]\,[\tilde{\rho_4} \circ \rho]$$

where $\tilde{\rho_4} \circ \rho : \texttt{other} \cdot x \leftarrow x^{m+1}$, we write ρ_4 to mean $\tilde{\rho_4} \circ \rho$.
We know u_i, resp. u'_i, states process i to be at local state q, resp. q'. Therefore, $\theta'_1\,[\rho_2]\,[\rho_3]\,[\rho_4]$ evaluates to *true* when applied to $[\tilde{u}_1, \ldots, \tilde{u}_{m+1}, u^\bullet]$.
We have now:

$$[\tilde{u}_1, \ldots, \tilde{u}_{m+1}, u^\bullet] \models \psi\,[\rho_1] \wedge \theta'_1\,[\rho_2]\,[\rho_3]\,[\rho_4]$$

That is:

$$[\tilde{u}_1, \ldots, \tilde{u}_{m+1}] \models \exists Y^\bullet.\psi\,[\rho_1] \wedge \theta'_1\,[\rho_2]\,[\rho_3]\,[\rho_4]$$

We deduce that $[\tilde{u}_1, \ldots, \tilde{u}_{m+1}]$ is in $[\![\phi]\!]$. Where $\phi = (m, \psi_{m+1})$. Since $\tilde{c} \preceq c$, and because $[\![\phi]\!]$ is by definition upward closed, we conclude that c is in $[\![\phi]\!]$.

– In case i is not in $\overline{m}$, then $c = [u_1, \ldots, u_n]$ with $u_k = u'_k$ for each k in $\overline{m}$. The configuration c is therefore also in $[\![\phi']\!]$.

Forall We consider universal conditions of the form $\forall_{LR}$. The cases of $\forall_L$ and $\forall_R$ can be easily deduced.

We distinguish two situations: the process firing the transition has an index i such that (i) i in $\overline{m}$; (ii) i outside $\overline{m}$.

– Case with the process firing the transition in $\overline{m}$.
  We write the configuration $c' \in [\![\phi']\!]$ also of the form:

$$c' = c'_1 \bullet u'_i \bullet c'_2 = [u'_1, \ldots, u'_n]$$

  for configurations $c'_1 = [u_1, \ldots, u_{i-1}]$, and $c'_2 = [u_{i+1}, \ldots, u_n]$.
  By definition of $c \overset{t}{\rightsquigarrow} c'$ we know the configuration c is of the form:

$$c = c_1 \bullet u_i \bullet c_2$$

  Where:
  - u_i, resp. u'_i, states process i to be at q, resp. q'
  - $c'_1 = c_1 \ominus (\theta_1, u_i, u'_i)$ and $c'_2 = c_2 \ominus (\theta_1, u_i, u'_i)$;
  - $\left(u_i, u'_i, u'_j\right) \models \theta_1$ for each process with indice j in c'_1 or c'_2.

  Intuitively, this says that (i) the state of the firing process was q in c and becomes q' in c'. (ii) The processes violating the universal condition are deleted from c_1, resp. c_2; this gives the configuration c'_1, resp. c'_2. (iii) All the processes that are left, i.e. those in c'_1 and c'_2, comply with the universal condition.
  Define the configuration $\tilde{c}$ to be:

$$\tilde{c} = \tilde{c}_1 \bullet u_i \bullet \tilde{c}_2 = [\tilde{u}_1, \ldots, \tilde{u}_m]$$

  with $\tilde{u}_k = u'_k$ for each $k \in \overline{m} - \{i\}$, and $\tilde{u}_i = u_i$.
  In the following we show $\tilde{c} \in [\![\phi]\!]$, where $\phi = (m, \psi')$ is obtained from $Pre_{t,\phi'}$.
  This is sufficient to show c also in $[\![\phi]\!]$ because: (i) $\tilde{c} \preceq c$, and (ii) $[\![\phi]\!]$ is upward closed by definition.
  Assume a set of fresh copies $X^\bullet$ of X. Define $u^\bullet$ to take the same values as u'_i, i.e. $u^\bullet = u'_i\,[\rho_1]$ where $\rho_1 : x^i \leftarrow x^\bullet$.
  By definition of $u'_1, \ldots, u'_n$, we have $[u'_1, \ldots, u'_m] \models \psi$, it is then easy to see that:

$$[\tilde{u}_1, \ldots, \tilde{u}_m, u^\bullet] \models \psi\,[\rho_1]$$

  Aslo, observe that $\left(u_i, u'_i, u'_j\right) \models \theta_1$ for each u'_j in c'_1 and c'_2, that is: $(\tilde{u}_i, u^\bullet, \tilde{u}_j) \models \theta_1$ for each $\tilde{u}_j$ in $\tilde{c}_1$ and $\tilde{c}_2$. In other words:

$$[\tilde{u}_1, \ldots, \tilde{u}_m, u^\bullet] \models \bigwedge_{1 \leq j \neq i \leq m} \theta_1\,[\rho_2]\,[\rho_3]\left[\rho_4^j\right]$$

  where the occurences of $\mathtt{self}\cdot x$ are replaced ($\rho_3 : \mathtt{self}\cdot x \leftarrow x^i$) by their value in $\tilde{u}_i$; the occurences of $\mathtt{self}\cdot x^{next}$ are replaced ($\rho_2 : \mathtt{self}\cdot x^{next} \leftarrow x^\bullet$) by their value in $u^\bullet$; and the occurences of $\mathtt{other}\cdot x$ are replaced ($\rho_4 : \mathtt{other}\cdot x \leftarrow x^j$) by their value in $\tilde{u}_j$ for each $\tilde{u}_j$ in $\tilde{c}_1$ or $\tilde{c}_2$.

We know $\tilde{u}_i$, resp. $u^\bullet$, states process i to be at local state q, resp. q'. Therefore:

$$[\tilde{u}_1, \ldots, \tilde{u}_m, u^\bullet] \models \psi\,[\rho_1] \wedge \bigwedge_{1 \le j \ne i \le m} \theta'_1\,[\rho_2]\,[\rho_3]\,\left[\rho_4^j\right]$$

That is:

$$[\tilde{u}_1, \ldots, \tilde{u}_m] \models \exists Y^\bullet.\psi\,[\rho_1] \wedge \bigwedge_{1 \le j \ne i \le m} \theta'_1\,[\rho_2]\,[\rho_3]\,[\rho_4]$$

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

– Case with the process firing the transition outside $\overline{m}$.
We know c' to be of the form:

$$c' = [u'_1, \ldots, u'_m] \bullet c'_1 \bullet u'_i \bullet c'_2$$

for a process state u'_i and some configurations c'_1, c'_2. By definition of $c \overset{t}{\rightsquigarrow} c'$ we know that c is of the form:

$$c = c_1 \bullet u_i \bullet c_2$$

Where:

- u_i, resp. u'_i, states process i to be at q, resp. q'
- $[u'_1, \ldots, u'_m] \bullet c'_1 = c_1 \ominus (\theta_1, u_i, u'_i)$ and $c'_2 = c_2 \ominus (\theta_1, u_i, u'_i)$;
- $(u_i, u'_i, u_j) \models \theta_1$ for each process with indice j in $[u_1, \ldots, u_m] \bullet c'_1$ or c'_2.

Intuitively, (i) the state of the firing process was q in c and becomes q' in c'; (ii) the processes violating the universal condition are deleted from $[u_1, \ldots, u_m] \bullet c_1$, resp. c_2; and (iii) all the processes that are left, i.e. those in $[u_1, \ldots, u_m] \bullet c'_1$ and c'_2 comply with the universal condition.
Notice that $[u'_1, \ldots, u'_m] \bullet c'_1 = c_1 \ominus (\theta_1, u_i, u'_i)$ means $[u'_1, \ldots, u'_m] \preceq c_1$. Therefore $c_1 \models \psi'$, and $c_1 \in [\![\phi']\!]$. By upward closeness of constraint denotation, $c \in [\![\phi']\!]$.

Now, we explain the computation of *Pre* for both rendez-vous and Broadcast.

Binary Communication Let $Y^\circ = \{x^\circ | x \in Y\}$ be a fresh copy of Y. For existential conditions, there are two types of constraint (m, ψ_j) for each $j : 1 \le j \ne i \le m$ is such that

– In the first case the condition is defined as:

$$\psi_j = \exists (X^\bullet \cup X^\circ)\,(\psi\,[\rho_1]\,[\rho_2] \wedge \theta'_1\,[\rho_3]\,[\rho_4]\,[\rho_5]\,[\rho_6])$$

where the new substitutions contain the following elements for each $x \in Y$:

$$\begin{array}{ll} \rho_1 : x^i \leftarrow x^\bullet & \rho_2 : x^j \leftarrow x^\circ \\ \rho_3 : \mathtt{self} \cdot x \leftarrow x^i & \rho_4 : \mathtt{self} \cdot x^{next} \leftarrow x^\bullet \\ \rho_5 : \mathtt{other} \cdot x \leftarrow x^j & \rho_6 : \mathtt{other} {\cdot} x^{next} \leftarrow x^\circ \end{array}$$

– In the second case the condition is defined as:

$$\psi'_j = \exists (X^\bullet \cup X^\circ) \left(\psi\left[\rho_1\right]\left[\rho_2\right] \wedge \theta'_1\left[\rho_3\right]\left[\rho_4\right]\left[\rho_5\right]\left[\rho_6\right]\right)$$

where the new substitutions contain the following elements for each $x \in Y$:

$$\begin{array}{ll} \rho_1 : x^i \leftarrow x^\bullet & \rho_2 : x^j \leftarrow x^\circ \\ \rho_3 : \mathtt{self} \cdot x \leftarrow x^j & \rho_4 : \mathtt{self} \cdot x^{next} \leftarrow x^\circ \\ \rho_5 : \mathtt{other} \cdot x \leftarrow x^i & \rho_6 : \mathtt{other} {\cdot} x^{next} \leftarrow x^\bullet \end{array}$$

Furthermore, there are two different constraints of the form $(m+1, \psi_{m+1})$.

– In the first case the condition is defined as:

$$\psi_{m+1} = \exists (X^\bullet \cup X^\circ) \left(\psi\left[\rho_1\right] \wedge \theta'_1\left[\rho_2\right]\left[\rho_3\right]\left[\rho_4\right]\left[\rho_5\right]\right)$$

where the new substitution contains the following elements for each $x \in Y$:

$$\begin{array}{lll} \rho_1 : x^i \leftarrow x^\bullet & \rho_2 : \mathtt{self} \cdot x \leftarrow x^i & \rho_3 : \mathtt{self} \cdot x^{next} \leftarrow x^\bullet \\ \rho_4 : \mathtt{other} \cdot x \leftarrow x^{m+1} & \rho_5 : \mathtt{other} {\cdot} x^{next} \leftarrow x^\circ & \end{array}$$

– In the second case the condition is defined as:

$$\psi'_{m+1} = \exists (X^\bullet \cup X^\circ) \left(\psi\left[\rho_1\right] \wedge \theta'_1\left[\rho_2\right]\left[\rho_3\right]\left[\rho_4\right]\left[\rho_5\right]\right)$$

where the new substitution contains the following elements for each $x \in Y$:

$$\begin{array}{lll} \rho_1 : x^i \leftarrow x^\bullet & \rho_2 : \mathtt{self} \cdot x \leftarrow x^{m+1} & \rho_3 : \mathtt{self} \cdot x^{next} \leftarrow x^\circ \\ \rho_4 : \mathtt{other} \cdot x \leftarrow x^i & \rho_5 : \mathtt{other} {\cdot} x^{next} \leftarrow x^\bullet & \end{array}$$

Broadcast For universally quantified formula, we need fresh copies Y_j° of Y for each set Y^j of variables occurring in ψ where $1 \leq i \neq j \leq m$. The new gap formula is defined then as

$$\psi' = \exists (X^\bullet \cup \bigcup_{1 \leq i \neq j \leq m} Y_j^\circ) \left(\bigwedge_{1 \leq j \neq i \leq m} \left(\psi\left[\rho_1\right]\left[\rho_2^j\right] \wedge \theta'_1\left[\rho_3\right]\left[\rho_4\right]\left[\rho_5^j\right]\left[\rho_6^j\right] \right) \right)$$

where the substitutions contain the following elements for each $x \in Y$ and $1 \leq j \neq i \leq m$:

$$\begin{array}{ll} \rho_1 : x^i \leftarrow x^\bullet & \rho_2^j : x^j \leftarrow x_j^\circ \\ \rho_3 : \mathtt{self} \cdot x \leftarrow x^i & \rho_4 : \mathtt{self} \cdot x^{next} \leftarrow x^\bullet \\ \rho_5^j : \mathtt{other} \cdot x \leftarrow x^j & \rho_6^j : \mathtt{other} {\cdot} x^{next} \leftarrow x_j^\circ \end{array}$$

B Details of the Case Studies

The table in Fig. 1 shows a summary of the experimental results we obtained with a prototype implementation of our approximated backward search. In the following section we give a detailed description of each case-study.

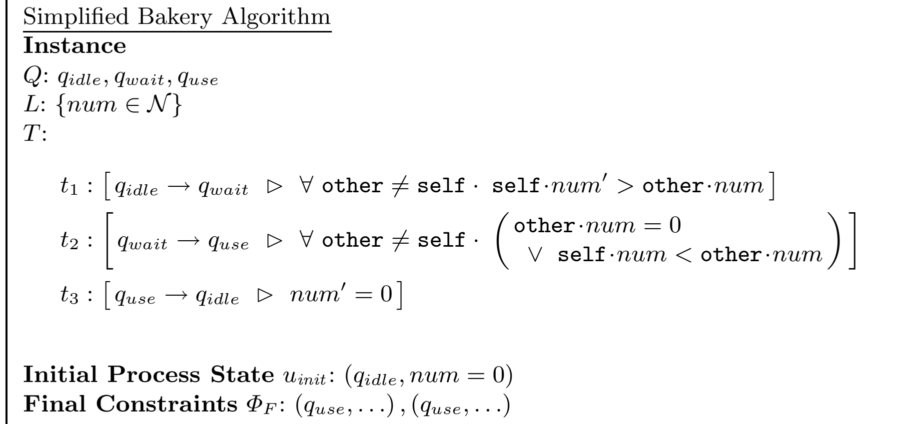

Fig. 2. Simplified Bakery Algorithm.

B.1 Simplified Bakery Algorithm

Lamport's Bakery's algorithm [18] is a well-known solution to the critical section problem for an arbitrary, finite number of processes. The algorithm works as follows. Each process has a local variable *num* in which it stores a ticket. Initially *num* is set to zero. When a process is interested in entering the critical section, it sets *num* to a value strictly greater than the tickets (i.e., value of *num*) of all other processes in the system. A possible way to implement this step is by taking the *max* value over all processes and then incrementing it by one. More in general, we just need to generate a *fresh* ticket. After the choosing step, the process waits until its ticket is less than the tickets of all other processes and then enters the critical section. When it releases the critical section, it resets its ticket to zero.

If we assume that the choosing step and the assignment to the local variable *num* is made in a atomic step, then we can model this protocol as shown in Fig. 2 (this version is known in the literature as the Simplified Bakery algorithm/or Bakery with atomicity conditions).

Since processes have a single local variable, our algorithm is guaranteed to terminate on this verification problem. As shown in Fig. 1, our procedure terminates and it automatically proves mutual exclusion for any number of processes.

B.2 Lamport's Bakery Algorithm

The original Lamport's Bakery algorithm does not take any atomicity assumption on the computation of the fresh number. The entry condition becomes more complicated here. Since two processes may take the same number, the condition compares tickets and identifiers: if two processes have the same number the one

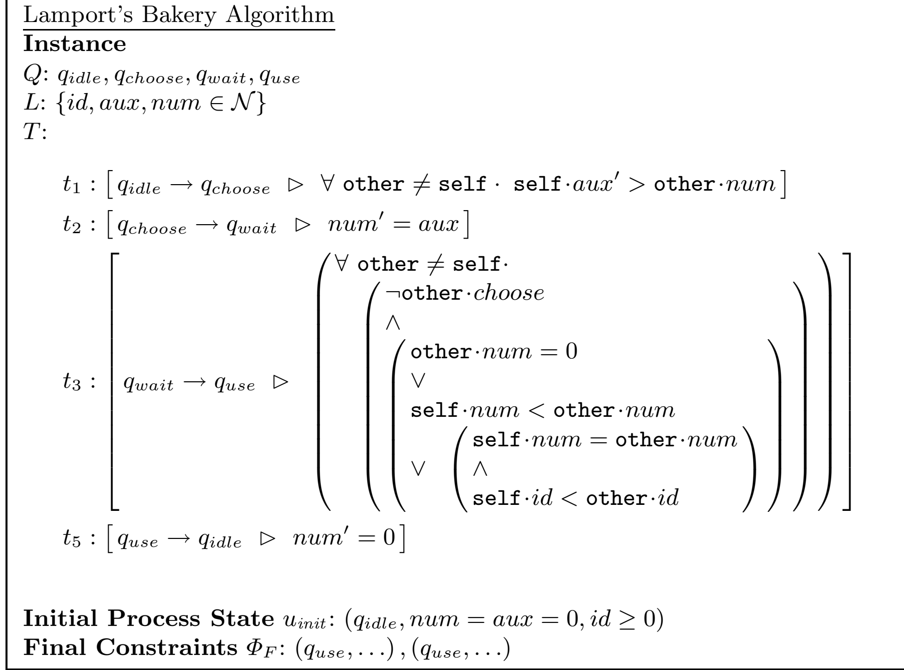

Fig. 3. Lamport mutex algorithm.

with smaller identifier has higher priority. We assume that all identifiers are different. Furthermore, to protect the test in the entry section from race conditions in the choosing phase, Lamport's algorithm uses a Boolean flag to mark *choosing* the assignment to the local variable num. The flag is set to true before starting the assignment and to false after its completion. A process willing to enter the critical section is forced to wait until all processes have concluded their choosing step before comparing its ticket with the other ones. Lamport's protocol is shown in Fig. 3. In our model each process has three local variables. The variable id maintains its identifier. The variable num maintains the current value (≥ 0) of the ticket. The variable aux is used to split in two steps the assignment of the fresh number to num, i.e., to make explicit the non-atomicity of the *choose* step. With rule t_1 we first update aux with a value greater than all numbers. With rule t_2 we assign aux to num. Rule t_3 models the enter condition. By means of a universally quantified formula, the current process compares its number with the number of all other processes. We assume this operation atomic. Notice that the condition fails (i.e. the process waits) whenever another process id' is choosing a number, or 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 $id' < id$.

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

B.3 Bogus Bakery Algorithm

It is interesting to notice that since the algorithm builds a symbolic reachability graph it can also be used for debugging purposes. For instance, consider again the Lamport's bakery algorithm described in the previous section. Let us now remove the condition $\neg\texttt{other}\cdot choose$ from the entry condition in t_3. We obtain a new transition defined as

$$\left[q_{wait} \to q_{use} \;\; \rhd \;\; \forall\, \texttt{other} \neq \texttt{self} \cdot \left(\begin{array}{l} \texttt{other}\cdot num = 0 \\ \vee\; \texttt{self}\cdot num < \texttt{other}\cdot num \\ \vee \left(\begin{array}{l} \texttt{self}\cdot num = \texttt{other}\cdot num \\ \wedge\; \texttt{self}\cdot id < \texttt{other}\cdot id \end{array} \right) \end{array} \right) \right]$$

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 Fig. 1, our procedure terminates after 10 iterations. Among the formulas computed at step 6, 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 traces are shown in Fig. 10: one is indicated by the ★ arrow and the other one by the ◄ arrow (they are symmetric). These traces represent the following type of executions. Process i moves to state *choose* and selects a number m. Process j, $j \neq i$, moves to state *choose*, and, then, selects and assigns to aux a number n s.t. $n < m$ before process i assigns m to num. (assignments are non-atomic). Now, process i enters the critical section. Then process j terminates its assignment and enters the critical section because its number is smaller than that of i. This execution corresponds to real error traces that explain why a process has to wait that the choosing step of all other processes is terminated before entering the critical section. Another interesting error trace is that produced by formula number 35 at step 5 indicated by the symbol ✠ in Fig. 10. At step 6 this formula produces a formula subsumed by 40 and 41 (i.e. an initial state). By looking at the trace from 35 to the bad states we discover another subtle error trace: process i selects a value n and assigns it to aux; process j with $i < j$ chooses a value $m = n$, and assigns it to num; process j enters the critical section. now, when process i completes its assignment, and it enters the critical section since its identifier i is less than that of the process j.

B.4 Ticket Mutual Exclusion Algorithm

The Ticket Algorithm [5] 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. When interested in the 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

Ticket Algorithm

Instance

Q: $q_{idle}, q_{wait}, q_{use}$

L: $\{a \in \mathcal{N}\}$

X_s: $\{t, s \in \mathcal{N}\}$

T:

$$t_1 : \left[q_{idle} \rightarrow q_{wait} \;\triangleright\; a' = t, t' > t \right]$$

$$t_2 : \left[q_{wait} \rightarrow q_{use} \;\triangleright\; a = s \right]$$

$$t_3 : \left[q_{use} \rightarrow q_{idle} \;\triangleright\; \begin{pmatrix} \exists \, \texttt{other} \neq \texttt{self} \cdot \; s' = \texttt{other} \cdot a \\ \wedge \\ \forall \, \texttt{other} \neq \texttt{self} \cdot \; s' \leq \texttt{other} \cdot a \end{pmatrix} \right]$$

Initial State of Shared Vars s=t=0

Initial Process State u_{init}: $(q_{idle}, a = 0)$

Final Constraints Φ_F: $(q_{use}, a = _) \, , (q_{use}, a = _)$

Fig. 4. Ticket mutex algorithm.

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 Fig. 1, our procedure terminates and it automatically proves mutual exclusion for any number of processes. The fixpoint contains formulas with at most 3 process states.

B.5 Lamport's Distributed Mutual Exclusion Algorithm

In this section we study a distributed version of Lamport's bakery algorithm without a central monitor and in which the computation of new tickets as well as the test for entering the critical section are non-atomic. Every process behaves like a server for its own ticket. The value of the ticket is sent upon reception of an explicit request from another process. Thus every step of the original bakery protocol is split into a client-server sub-protocol. Processes communicate via directed one-place channels through which they send requests (to enter cs) and acknowledgments. Each process has local variable *num* ranging on natural numbers. When a process *id* wants to enter, it assigns *num* to an auxiliary variable *aux*, and broadcasts a request req_1 to all other processes (on channels $id \rightarrow id'$) with the current value of *num*. Then it enters in the *choose* state waiting for acknowledgment ack_1 containing the current value of *num* of all other processes. Reply messages contain *num* in a variable v. Upon reception of an acknowledgment with value v, a process stores in its variable *aux* the maximum of v and the current value of *aux*. After having received the numbers from all

Lamport's Distributed Mut-Ex, Part I: Distributed Computation of Number

Instance

Q_P: $q_{idle}, q_{wait}, q_{use}$
X_P: $\{id, num, aux \in \mathcal{N}\}$

Q_C: $q_{empty}, q_{req_1}, q_{ack_1}, q_{ok_1}, q_{req_2}, q_{ack_2}, q_{ok_2}$
X_C: $\{s_id, r_id, v \in \mathcal{N}\}$

Transitions (part I):

$$t_1 : \left[q_{idle} \rightarrow q_{choose} \;\; \rhd \;\; \left(\begin{array}{l} aux' = num \, \wedge \\ \forall \, \texttt{other} \neq \texttt{self} \cdot \\ \left(\begin{array}{l} \texttt{other} \cdot state = empty \;\wedge\; \texttt{other} \cdot s_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = req_1 \;\wedge\; \texttt{other} \cdot v' = \texttt{self} \cdot num \end{array} \right) \end{array} \right) \right]$$

$$t_2 : \left[q_{choose} \rightarrow q_{choose} \;\; \rhd \;\; \left(\begin{array}{l} \exists \, \texttt{other} \neq \texttt{self} \cdot \\ \left(\begin{array}{l} \texttt{other} \cdot state = ack_1 \;\wedge\; \texttt{other} \cdot s_id = \texttt{self} \cdot id \\ \quad \wedge\; \texttt{other} \cdot v > \texttt{self} \cdot aux \\ \supset \\ \texttt{other} \cdot state' = ok_1 \;\wedge\; \texttt{self} \cdot aux' = \texttt{other} \cdot v \end{array} \right) \end{array} \right) \right]$$

$$t_3 : \left[q_{choose} \rightarrow q_{choose} \;\; \rhd \;\; \left(\begin{array}{l} \exists \, \texttt{other} \neq \texttt{self} \cdot \\ \left(\begin{array}{l} \texttt{other} \cdot state = ack_1 \;\wedge\; \texttt{other} \cdot s_id = \texttt{self} \cdot id \\ \quad \wedge\; \texttt{other} \cdot v \leq \texttt{self} \cdot aux \\ \supset \\ \texttt{other} \cdot state' = ok_1 \end{array} \right) \end{array} \right) \right]$$

$$t_4 : \left[q_{choose} \rightarrow q_{wait} \;\; \rhd \;\; \left(\begin{array}{l} num' > aux \\ \wedge \\ \forall \, \texttt{other} \neq \texttt{self} \cdot \\ \;\; \texttt{other} \cdot s_id = \texttt{self} \cdot id \;\; \supset \;\; \texttt{other} \cdot state = ok_1 \end{array} \right) \right]$$

Fig. 5. Distributed computation of ticket.

other processes, a process moves to state *wait* selecting a new value for *num* greater than *aux*, i.e., a fresh ticket. In state *wait* a process broadcasts a new request req_2 for reading again the numbers of all other process, and then waits for a reply. For each acknowledgment ack_2 with value v coming from process id', process id checks the enter-condition $v = 0 \vee num < v \vee (num = v \wedge id < id')$. If the condition is satisfied it checks the condition for the other process, otherwise it sends another req_2 to the same process. When the enter-condition is checked over all other processes, process *id* enters the critical section. The variable *num* is reset when releasing the critical section.

Our model of this algorithm is shown in in Fig. 5, Fig. 6, and Fig. 7. In the model we use processes of two types: processes and channels. Processes have local variables *id*, *num* (ticket), and *aux* (auxiliary variable). Channels have local variables *s_id* (sender id), *r_id* (receiver id), *v* (ticket). Initial states are

Lamport's Distributed Mut-Ex, Part II: Reply

$$t_5 : \left[q_s \to q_s \ \triangleright \ \begin{pmatrix} \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \texttt{other}\cdot state = req_1 \ \wedge \ \texttt{other}\cdot r_id = \texttt{self}\cdot id \\ \supset \\ \texttt{other}\cdot state' = ack_1 \ \wedge \ \texttt{other}\cdot v' = \texttt{self}\cdot num \end{pmatrix} \right]$$
$$for\ any\ s \in \{idle, choose, wait, use\}$$

$$t_6 : \left[q_s \to q_s \ \triangleright \ \begin{pmatrix} \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \texttt{other}\cdot state = req_2 \ \wedge \ \texttt{other}\cdot r_id = \texttt{self}\cdot id \\ \supset \\ \texttt{other}\cdot state' = ack_2 \ \wedge \ \texttt{other}\cdot v' = \texttt{self}\cdot num \end{pmatrix} \right]$$
$$for\ any\ s \in \{idle, wait, use\}$$

Fig. 6. Reply messages.

collections of *idle* processes (each one with unique identifiers and local clock set to zero) and of empty channels (two directed channels $id \to id'$ and $id' \to id$ for each pair of processes id, id' s.t. $id \neq id'$).

The rules in Fig. 5 model the distributed computation of a new ticket (*choosing* step). In rule t_1, process id in state *idle* broadcasts a request for *num* to all other processes, i.e., for any $id1$ we set the state of the channel $id \to id1$ to req_1. Variable *aux* is set to *num*. Here we assume that a process $id1$ can reply with *num* to req_1 in any state (see t_5 in Fig. 6). In rule t_2 and t_3 process id in state *choose* receives an acknowledgment with value v from $id1$: it stores ok_1 in the channel $id \to id1$, and saves the maximum of v and *aux* in *aux*. In rule t_4 process id checks that all channels $id \to id1$ contain ok_1 (all acks have been received). If this holds it moves to *wait* and sets *num* to a value greater than *aux*.

The rules in Fig. 7 model the race for entering the critical section. In rule t_7, process id in state *wait* broadcasts a new request for *num* to all other processes, i.e., for any $id1$ we set the state of channel $id \to id1$ to req_2. Here we assume that $id1$ can reply with *num* to req_2 if and only if it is not in state *choose* (see t_6 in Fig. 6). In rule t_8 and t_9 process id in state *wait* receives an acknowledgment ack_2 with value v from $id1$. If the enter-condition on v and *num* is satisfied, it sets the state of channel $id \to id_1$ to ok_2. Otherwise, the request req_2 to process $id1$ remains pending. In rule t_{10} process id checks that all channels $id \to id1$ contain ok_2 (all acks have been received). If this holds, it enters state *use* (critical section). In rule t_{11} a process id in state *use* goes back to state *idle*, resets *num* to zero, and sets the state of each channel $id \to id1$ back to *empty*.

The bad states form an upward closed set of configurations in which at least two processes id and id' are in state *use* with mutually granted acknowledgments (i.e. channels $id \to id'$ and $id' \to id$ are both in state ok_2). As shown in Fig. 1, our procedure terminates on this example and it automatically proves mutual exclusion for any number of processes.

B.6 Ricart-Agrawala's Distributed Mutual Exclusion Algorithm

The Ricart-Agrawala Algorithm [20] is an algorithm for mutual exclusion on a distributed system. Processes communicate via directed one-place channels through which they send requests (to enter cs) and acknowledgments. When a process *id* wants to enter, it broadcasts a request to all other processes (on channels $id \rightarrow id'$) with its current local clock as time-stamp. This time-stamp is stored in a local variable *last_request*. The process enters when it receives acknowledgments from all other processes. Acknowledgments are sent: (1) in state idle, (2) while waiting if the request has a time-stamp that is either less than *last_request* or equal to *last_request* but the identifier of the requesting process is smaller (this condition breaks the tie in case two processes send requests with the same time-stamps). Each time a process receives a message (ack/req) with time-stamp *ts* it updates its local clock by taking a value strictly greater than *ts* a the old value of the clock (in this manner, local clock are synchronized). The new value is used to stamp outgoing messages. When a process releases the critical section, all outgoing channels are emptied, and acknowledgments are sent to all pending requests (with an updated time-stamp).

Our model of the Ricart-Agrawala algorithm is shown in Fig. 8 and Fig. 9 (reply messages). In the model we use processes of two types: Type P for processes with state in $\{idle, wait, use\}$, and local variables *id*, *clock*, and *last*; Type C for channels with state in $\{empty, req, ack, ok\}$ and local variables *s_id* (sender id), *r_id* (receiver id), *ts* (time-stamp). Initial states are collections of *idle* processes (each one with unique identifier and local clock set to zero) and of empty channels (two directed channels $id \rightarrow id'$ and $id' \rightarrow id$ for each pair of processes (id, id') s.t. $id \neq id'$). The four rules in Fig. 8 model request messages sent by process *id* on channel $id \rightarrow id1$ as well as entry and exit conditions. In rule t_1, process *id* in state *idle* broadcasts a request to all other processes, i.e., for any id' we set the empty channel $id \rightarrow id'$ to *req*. The local clock is refreshed and the new value stored in *last*. In rule t_2 process *id* in state *wait* receives an acknowledgment from id' and stores *ok* in the channel $id \rightarrow id'$. Both the local clock and the time-stamp of the channel are refreshed. In rule t_3 a process *id* enters in critical section provided that all channels $id \rightarrow id'$ are in state *ok* (i.e. *id* has received an acknowledgment from each other process $id1$). In rule t_4 a process *id* releases the critical section: it empties every channel $id \rightarrow id'$, and sends acknowledgments for every pending request on channel $id' \rightarrow id$. Acknowledgments are sent with a fresh time-stamp.

The three rules in Fig. 9 model reply messages sent by process *id* on channel $id1 \rightarrow id$. In rule t_5 process *id* in state *idle* acknowledges a request coming from process id' on channel $id' \rightarrow id$. The local clock is refreshed (taking a value greater than both the old clock and the time-stamp of the incoming request). The new clock is used as a time-stamp for the acknowledgment. In rule t_6, process *id* in state *wait* immediately sends an acknowledgment to a request coming from process $id1$ on channel $id1 \rightarrow id$ with time stamp *ts* less than *last*. The same happens when the time stamp *ts* is equal to *last* and $id1 < id$ as modeled in rule t_7. The local clock and the time-stamp are refreshed in both cases. In the

remaining cases the request remains pending in the channel $id1 \to id$ (we need no rules for this case).

The bad states are upward closed sets of configurations in which at least two processes are in state *use*. In order to exploit our approximation we also insert information on channels, i.e., we consider two processes id and id' in state *use* with mutually granted acknowledgments (i.e. channels $id \to id'$ and $id' \to id$ are both in state *ok*).

Processes of both types have three local variables of type integer. On this type of systems our procedure is not guaranteed to terminate. However, as shown in Fig. 1, our procedure terminates on this example and it automatically proves mutual exclusion for any number of processes.

Lamport's Distributed Mut-Ex, Part III: Entry and Exit

$$t_7 : \left[q_{wait} \rightarrow q_{wait} \;\; \triangleright \;\; \left(\begin{array}{l} \forall\ \texttt{other} \neq \texttt{self}\cdot \\ \quad \texttt{other}\cdot state = ok_1 \ \wedge \ \texttt{other}\cdot s_id = \texttt{self}\cdot id \\ \quad \supset \\ \quad \texttt{other}\cdot state' = req_2 \end{array} \right) \right]$$

$$t_8 : \left[q_{wait} \rightarrow q_{wait} \;\; \triangleright \;\; \left(\begin{array}{l} \exists\ \texttt{other} \neq \texttt{self}\cdot \\ \left(\begin{array}{l} \texttt{other}\cdot state = ack_2\ \wedge \\ \texttt{other}\cdot s_id = \texttt{self}\cdot id \ \wedge \ \texttt{other}\cdot v > 0 \wedge \\ \left(\begin{array}{l} \texttt{self}\cdot num > \texttt{other}\cdot v\ \vee \\ \left(\texttt{other}\cdot v = \texttt{self}\cdot num \wedge \ \texttt{self}\cdot id > r_id\right) \end{array} \right) \end{array} \right) \\ \quad \supset \\ \quad \texttt{other}\cdot state' = req_2 \end{array} \right) \right]$$

$$t_9 : \left[q_{wait} \rightarrow q_{wait} \;\; \triangleright \;\; \left(\begin{array}{l} \exists\ \texttt{other} \neq \texttt{self}\cdot \\ \left(\begin{array}{l} \texttt{other}\cdot state = ack_2\ \wedge \\ \texttt{other}\cdot s_id = \texttt{self}\cdot id\ \wedge \\ \left(\begin{array}{l} \texttt{other}\cdot v = 0\ \vee \\ \texttt{self}\cdot num < \texttt{other}\cdot v\ \vee \\ (\texttt{other}\cdot v = \texttt{self}\cdot num \ \wedge \ \texttt{self}\cdot id < r_id) \end{array} \right) \end{array} \right) \\ \quad \supset \\ \quad \texttt{other}\cdot state' = ok_2 \end{array} \right) \right]$$

$$t_{10} : \left[q_{wait} \rightarrow q_{use} \;\; \triangleright \;\; \left(\begin{array}{l} \forall\ \texttt{other} \neq \texttt{self}\cdot \\ \quad s_id = \texttt{self}\cdot id \quad \supset \quad \texttt{other}\cdot state = ok_2 \end{array} \right) \right]$$

$$t_{11} : \left[q_{use} \rightarrow q_{idle} \;\; \triangleright \;\; \left(\begin{array}{l} num' = 0 \\ \wedge \\ \forall\ \texttt{other} \neq \texttt{self}\cdot \\ \quad \texttt{other}\cdot state = ok_2 \ \wedge \ \texttt{other}\cdot s_id = \texttt{self}\cdot id \\ \quad \supset \\ \quad \texttt{other}\cdot state' = empty \end{array} \right) \right]$$

Fig. 7. Core protocol

Ricart-Agrawala's Algorithm, Part I: Request, Entry, and Exit

Instance

Q_P: $q_{idle}, q_{wait}, q_{use}$ X_P: $\{id, clock, last \in \mathcal{N}\}$

Q_C: $q_{empty}, q_{req}, q_{ack}, q_{ok}$

X_C: $\{s_id, r_id, ts \in \mathcal{N}\}$

Transitions (Part I):

$$t_1 : \left[q_{idle} \to q_{wait} \;\; \rhd \;\; \begin{pmatrix} clock' > clock \\ \wedge \\ \forall\ \texttt{other} \neq \texttt{self} \cdot \\ \quad \begin{pmatrix} \texttt{other} \cdot state = empty \;\wedge\; \texttt{other} \cdot s_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = req \;\wedge\; \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{pmatrix} \end{pmatrix} \right]$$

$$t_2 : \left[q_{wait} \to q_{wait} \;\; \rhd \;\; \begin{pmatrix} clock' > clock \\ \wedge \\ \exists\ \texttt{other} \neq \texttt{self} \cdot \\ \quad \begin{pmatrix} \texttt{other} \cdot state = ack \;\wedge\; \texttt{other} \cdot s_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = ok \;\wedge\; \texttt{self} \cdot clock' > \texttt{other} \cdot ts \\ \quad \wedge\; \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{pmatrix} \end{pmatrix} \right]$$

$$t_3 : \left[q_{wait} \to q_{use} \;\; \rhd \;\; \begin{pmatrix} \forall\ \texttt{other} \neq \texttt{self} \cdot \\ \quad \texttt{other} \cdot s_id = \texttt{self} \cdot id \;\; \supset \;\; \texttt{other} \cdot state = ok \end{pmatrix} \right]$$

$$t_4 : \left[q_{use} \to q_{idle} \;\; \rhd \;\; \begin{pmatrix} clock' > clock \\ \wedge \\ \forall\ \texttt{other} \neq \texttt{self} \cdot \\ \quad \begin{pmatrix} \texttt{other} \cdot state = ok \;\wedge\; \texttt{other} \cdot s_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = empty \end{pmatrix} \\ \wedge \\ \forall\ \texttt{other} \neq \texttt{self} \cdot \\ \quad \begin{pmatrix} \texttt{other} \cdot state = req \;\wedge\; \texttt{other} \cdot r_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = ack \;\wedge\; \texttt{self} \cdot clock' > \texttt{other} \cdot ts \\ \quad \wedge\; \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{pmatrix} \end{pmatrix} \right]$$

Fig. 8. Request messages, entry and exit section.

Ricart-Agrawala's Algorithm, Part II: Reply

$$t_5 : \left[q_{idle} \rightarrow q_{idle} \;\; \triangleright \;\; \begin{pmatrix} clock' > clock \\ \wedge \\ \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad \begin{pmatrix} \texttt{other}\cdot state = req \;\wedge\; \texttt{other}\cdot r_id = \texttt{self}\cdot id \\ \supset \\ \texttt{other}\cdot state' = ack \;\wedge\; \texttt{self}\cdot clock' > \texttt{other}\cdot ts \\ \quad\quad \wedge\; \texttt{other}\cdot ts' = \texttt{self}\cdot clock' \end{pmatrix} \end{pmatrix} \right]$$

$$t_6 : \left[q_{wait} \rightarrow q_{wait} \;\; \triangleright \;\; \begin{pmatrix} clock' > clock \\ \wedge \\ \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad \begin{pmatrix} \texttt{other}\cdot state = req \;\wedge\; \texttt{other}\cdot r_id = \texttt{self}\cdot id \\ \quad\quad \wedge\; \texttt{other}\cdot ts < \texttt{self}\cdot last \\ \supset \\ \texttt{other}\cdot state' = ack \;\wedge\; \texttt{self}\cdot clock' > \texttt{other}\cdot ts \\ \quad\quad \wedge\; \texttt{other}\cdot ts' = \texttt{self}\cdot clock' \end{pmatrix} \end{pmatrix} \right]$$

$$t_7 : \left[q_{wait} \rightarrow q_{wait} \;\; \triangleright \;\; \begin{pmatrix} clock' > clock \\ \wedge \\ \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad \begin{pmatrix} \texttt{other}\cdot state = req \;\wedge\; \texttt{other}\cdot r_id = \texttt{self}\cdot id \\ \quad\quad \wedge\; \texttt{other}\cdot ts = \texttt{self}\cdot last \;\wedge \\ \quad\quad\quad \wedge\; \texttt{other}\cdot s_id < \texttt{self}\cdot id \\ \supset \\ \texttt{other}\cdot state' = ack \;\wedge\; \texttt{self}\cdot clock' > \texttt{other}\cdot ts \\ \quad\quad \wedge\; \texttt{other}\cdot ts' = \texttt{self}\cdot clock' \end{pmatrix} \end{pmatrix} \right]$$

Fig. 9. Reply messages.

...

Step 6

$(\mathit{think}, A, _, _), (\mathit{think}, B, _, C), A < B, C >= 0, \underline{41}, \underline{36}, T_1 \quad \bigstar$

$(\mathit{think}, A, _, _), (\mathit{think}, B, _, C), B < A, C >= 0, \underline{40}, \underline{37}, T_1 \quad \blacktriangleleft$

Step 5

$(\mathit{think}, A, _, _), (\mathit{choose}, B, _, _), A < B, \underline{37}, \underline{25}, T_1 \quad \blacktriangleleft$

$(\mathit{think}, A, _, _), (\mathit{choose}, B, _, _), B < A, \underline{36}, \underline{27}, T_1 \quad \bigstar$

$(\mathit{think}, A, _, _), (\mathit{choose}, B, C, _), C > 0, A < B, \underline{35}, \underline{28}, T_1 \quad \maltese$

...

Step 4

$(\mathit{choose}, A, B, _), (\mathit{choose}, C, D, _), C < A, D = B, \underline{28}, \underline{20}, T_2 \quad \maltese$

$(\mathit{choose}, A, B, _), (\mathit{choose}, C, D, _), D < B, C < A, \underline{27}, \underline{21}, T_2 \quad \bigstar$

...

$(\mathit{choose}, A, B, _), (\mathit{choose}, C, D, _), D < B, A < C, \underline{25}, \underline{24}, T_2 \quad \blacktriangleleft$

...

Step 3

$(\mathit{wait}, A, _, B), (\mathit{choose}, C, D, _), D < B, A < C, \underline{24}, \underline{7}, T_3 \quad \blacktriangleleft$

$(\mathit{wait}, A, _, B), (\mathit{choose}, C, D, _), D < B, C < A, \underline{21}, \underline{9}, T_3 \quad \bigstar$

$(\mathit{wait}, A, _, B), (\mathit{choose}, C, D, _), C < A, D = B, \underline{20}, \underline{10}, T_3 \quad \maltese$

...

Step 2

$(\mathit{wait}, A, _, 0), (\mathit{wait}, B, _, C), C > 0, A < B, 13, 5, T_3$

$(\mathit{wait}, A, _, 0), (\mathit{wait}, B, _, 0), A < B, 12, 5, T_3$

$(\mathit{wait}, A, _, B), (\mathit{wait}, C, _, 0), B > 0, A < C, 11, 6, T_3$

$(\mathit{choose}, A, B, _), (\mathit{use}, C, _, D), D = B, A < C, \underline{10}, \underline{3}, T_2 \quad \maltese$

$(\mathit{choose}, A, B, _), (\mathit{use}, C, _, D), B < D, A < C, \underline{9}, \underline{4}, T_2 \quad \bigstar$

$(\mathit{choose}, A, _, _), (\mathit{use}, B, _, 0), B < A, 8, 5, T_2$

$(\mathit{choose}, A, B, _), (\mathit{use}, C, _, D), C < A, B < D, \underline{7}, \underline{6}, T_2 \quad \blacktriangleleft$

Step 1

$(\mathit{wait}, A, _, B), (\mathit{use}, C, _, D), C < A, B < D, \underline{6}, \underline{1}, T_3 \quad \blacktriangleleft$

$(\mathit{wait}, A, _, _), (\mathit{use}, B, _, 0), B < A, 5, 1, T_3$

$(\mathit{wait}, A, _, B), (\mathit{use}, C, _, D), B < D, A < C, \underline{4}, \underline{1}, T_3 \quad \bigstar$

$(\mathit{wait}, A, _, B), (\mathit{use}, C, _, D), D = B, A < C, \underline{3}, \underline{1}, T_3 \quad \maltese$

$(\mathit{wait}, A, _, _), (\mathit{use}, B, _, 0), A < B, 2, 1, T_3$

Step 0 : **Bad states**

$(\mathit{use}, A, _, _), (\mathit{use}, B, _, _), A < B, \underline{1}, 0, 0 \quad \bigstar \quad \maltese \quad \blacktriangleleft$

Fig. 10. 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.