Verifying Networks of Timed Processes (Extended Abstract)

Parosh Aziz Abdulla and Bengt Jonsson

Dept. of Computer Systems
Uppsala University
P.O. Box 325, S-751 05 Uppsala, Sweden
{parosh,bengt}@docs.uu.se,
WWW: http://www.docs.uu.se/~{parosh,bengt}

Abstract. Over the last years there has been an increasing research effort directed towards the automatic verification of infinite state systems, such as timed automata, hybrid automata, data-independent systems, relational automata, Petri nets, and lossy channel systems. We present a method for deciding reachability properties of networks of timed processes. Such a network consists of an arbitrary set of identical timed automata, each with a single real-valued clock. Using a standard reduction from safety properties to reachability properties, we can use our algorithm to decide general safety properties of timed networks. To our knowledge, this is the first decidability result concerning verification of systems that are infinite-state in "two dimensions": they contain an arbitrary set of (identical) processes, and they use infinite data-structures, viz. real-valued clocks. We illustrate our method by showing how it can be used to automatically verify Fischer's protocol, a timer-based protocol for enforcing mutual exclusion among an arbitrary number of processes.

1 Introduction

The last decade has seen much progress with regard to automated verification of reactive programs. The most dramatic advances have been obtained for finite-state programs. However, methods and algorithms are now emerging for the automatic verification of infinite state programs. There are at least two ways in which a program can be infinite-state. A program can be infinite-state because it operates on data structures from a potentially infinite domain, e.g., integers, stacks, queues, etc. Nontrivial verification algorithms have been developed for several classes of such systems, notably timed automata [ACD90], hybrid automata [Hen95], data-independent systems [JP93,Wol86], relational automata ([Čer94]), Petri nets ([JM95]), pushdown processes ([BS95]) and lossy channel systems [AJ96,AK95]. A program can also be infinite-state because it is intended to run on a network with an arbitrary number of nodes, i.e., the the program is parameterized with respect to the topology of the network of nodes. In this case, one would like to verify correctness for any number of components and any interconnection topology. Verification algorithms have been developed for systems consisting of an unbounded number of similar or identical finite-state processes ([GS92]), and (using a manually supplied induction hypothesis) for more general classes of parameterized systems [CGJ95,KM89].

In this paper, we will present an algorithm for verifying safety properties of a class of programs, which we call *timed networks*. A timed network is a system consisting of an arbitrary set of processes, each of which is a finite-state system operating on a real-valued clock. Each process could roughly be considered as a timed automaton [ACD90] with a single clock. In addition, our model also allows a central finite-state process, called a *controller*. Timed networks embody both of the two reasons for being infinite-state: they use an infinite data structure (namely clocks which can assume values from the set of real numbers), and they are parameterized in allowing an arbitrary set of processes. To our knowledge, this is the first decidability result concerning verification of networks of infinite-state processes.

We present an algorithm for deciding reachability properties of timed networks. Using a standard reduction (described e.g., in [VW86]) from safety properties to reachability properties, we can use this algorithm to decide general safety properties of timed networks. To decide reachability, we adapt a standard symbolic verification algorithm which has been used e.g., in model-checking [CES86]. A rough description of this method is that in order to check whether a state in some set F is reachable, we compute the set of all states from which a state in F is reachable. This computation is performed using a standard fixed-point iteration, where for successively larger j we compute the set of states from which a state in F can be reached by a sequence of transitions of length less than or equal to j. More precisely, we obtain the $(j+1)$st approximation from the jth approximation by adding the *pre-image* of the jth approximation, i.e., the set of states from which a state in the jth approximation can be reached by some transition. If this procedure converges, one checks whether the result intersects the set of initial states of the model. The heart of our result is solving the following three problems:

- finding a suitable representation of infinite sets of states,
- finding a method for computing pre-images, and
- proving that the iteration always converges.

To represent sets of states, we use constraints which generalize the notion of regions used to verify properties of (non-parameterized) timed automata [ACD90]. A constraint represents conditions on a potentially unbounded number of processes and their clocks. In contrast to the situation for timed automata [ACD90], where for each program there are finitely many regions, there is in general an infinite number of constraints that can appear in the analysis of a given timed network. To handle this, we introduce an entailment ordering on contraints. The key step in our proof of decidability consists in proving that this relation is a well quasi-ordering, implying that the the above mentioned fixedpoint iteration converges.

Our results also demonstrate the strength and applicability of the general framework described in [AČJYK96,Fin90]. Using that framework, we can conclude the decidability of eventuality properties (of the form AFp in CTL). for timed networks, and the question of whether or not a timed network simulates or is simulated by a finite-state system. We will not further consider these questions in this paper.

Our model of timed networks is related to other formalisms for timed systems, notably time or timed Petri nets [GMMP91,BD91] and Timed CCS [Yi91]. Our decidability result can be translated to decidability results for variants of these formalisms. It is known that reachability is undecidable for time Petri nets. This is due to the inclusion of *urgency* in the Petri net model. Urgency means that a transition is forced to execute within a specified timeout period. In our model transitions can not be forced to occur; a timeout can only specify that a transition is executed within a specified time period *if* it is executed. Urgency allows the model to test for emptiness of a place, thus leading to undecidability. A similar difference holds in comparison with Timed CCS.

As an illustration of our method, we model Fischer's protocol and show how an automatic verification algorithm would go about verifying mutual exclusion. Several tools for verifying automata with a fixed number of clocks have been used to verify the protocol for an increasing number of processes (e.g., [ACHH92]). Kristoffersen et al. [KLL$^+$97] describes an experiment where the number of processes is 50. In [LSW95], a constraint-based proof methodology is used to perform a manual verification of the protocol.

Outline The rest of the paper is structured as follows. In the next section, we present our model of timed networks. An overview of the reachability algorithm is presented in Section 3. In Section 4, we present our constraint system. In Section 5, we present a procedure for calculating the pre-image of a set of states which are represented by a constraint. In Section 6 we prove that the entailment ordering on the constraint system is a well quasi-ordering, which implies that our algorithm always terminates. An application of the reachability algorithm to the verification of Fischer's protocol is given in Section 7.

2 Timed Networks

In this section, we will define networks of timed processes. Intuitively, a network of timed processes consists of a *controller* and an arbitrarily large set of identical (timed) *processes*. The controller is a finite-state transition system. Each process has a finite-state control part, and an unbounded data structure, namely a real-valued clock[1]. The values of the clocks of the processes are incremented continuously at the same rate. In addition to letting time pass by incrementing the clocks, the network can change its configuration according to a finite number of *rules*. Each rule describes a set of transitions in which the controller and an arbitrary but fixed-size set of processes synchronize and simultaneously change their states. A rule may be conditioned on the control states of the participating

[1] The controller could also be equipped with a timer, but this aspect is not central to our result, so we will omit it.

controller and processes, and on conditions on the clock values of the participating processes. If the conditions for a rule are satisfied, the controller and each participating process may change its state and (optionally) reset its clock to 0.

We are interested in verifying correctness of a network regardless of its size. The actual object of study will therefore be a *family* of networks, where the number of processes is not given. A family merely defines the controller and process states together with a set of rules. The parameter (i.e., size) of the network will be introduced later, when we define configurations.

We use $\mathcal{N}$ and $\mathcal{R}^{\geq 0}$ to denote the sets of natural numbers and nonnegative reals respectively. For $n \in \mathcal{N}$, we use $\widehat{n}$ to denote the set $\{1, \ldots, n\}$. A *guarded command* is of the form $p(x) \longrightarrow op$, where $p(x)$ is a boolean combination of predicates of the form $k < x$, $k \leq x$, $k > x$, or $k \geq x$ for $k \in \mathcal{N}$, and $op \in \{reset, skip\}$.

Definition 1. *A* family of timed networks *(timed network for short) is a triple* $\langle C, Q, R \rangle$*, where:*

C *is a finite set of* controller *states.*
Q *is a finite set of* process *states.*
R *is a finite set of* rules*. A rule r is of the form*

$$\langle \langle c, c' \rangle, \langle q_1, stmt_1, q_1' \rangle, \ldots, \langle q_n, stmt_n, q_n' \rangle \rangle$$

where $c, c' \in C$, $q_i, q_i' \in Q$, and $stmt_i$ is a guarded command.

Intuitively, the set C represents the set of states of the controller. The set Q represents the set of states of each of the identical processes. A rule r describes a set of transitions of the network. The rule is enabled if the state of the controller is c, and if there are n processes with states $q_1, \ldots, q_n$, respectively, whose clock values satisfy the corresponding guards. The rule is executed by simultaneously changing the state of the controller to c', changing the states of the n processes to $q_1', \ldots, q_n'$ respectively, and modifying values of the clocks according to the relevant guarded commands.

Definition 2. *A* configuration γ *of a timed network $\langle C, Q, R \rangle$ is a quadruple of form $\langle I, c, \mathrm{q}, \mathbf{x} \rangle$, where I is a finite* index set*, $c \in C$, $\mathrm{q}: I \to Q$, and $\mathbf{x}: I \to \mathcal{R}^{\geq 0}$.*

Intuitively, I is the set of indices of processes in the network. The index set does not change when performing transitions. Each element in I will be used as an index to represent one particular process in the network. Thus, we can say that a timed network defines a family of networks parametrized by I[2]. The state of the controller is given by c, the states of the processes are given by the mapping q from indices to process states, and the clock values are given by the mapping $\mathbf{x}$ from indices to nonnegative real numbers.

[2] We can extend our model to include dynamic creation and destruction of processes, by allowing the set of indices in a configuration to change dynamically. Our decidability result holds also for such an extension. However, we will not consider that in the present paper.

A timed network changes its configuration by performing transitions. We will define a transition relation $\longrightarrow$ as the union of a *discrete* transition relation $\longrightarrow_D$, representing transitions caused by the rules, and a *timed* transition relation $\longrightarrow_T$ which represents the passage of time. The discrete relation $\longrightarrow_D$ will furthermore be the union of transition relations $\xrightarrow{r}_D$ corresponding to each rule r, i.e., $\longrightarrow_D = \bigcup_{r \in R} \xrightarrow{r}_D$.

Definition 3. *Let $r = \langle \langle c, c' \rangle, \langle q_1, stmt_1, q_1' \rangle, \ldots, \langle q_n, stmt_n, q_n' \rangle \rangle$ be a rule where $stmt_i$ is of form $p_i(x) \longrightarrow op_i$ for $i = 1, \ldots, n$. Consider two configurations $\gamma = \langle I, c, \mathsf{q}, \mathbf{x} \rangle$ and $\gamma' = \langle I, c', \mathsf{q}', \mathbf{x}' \rangle$, with the same index sets, and where the controller states of γ and γ' are the same as the controller states in the rule r. We use $\gamma \xrightarrow{r}_D \gamma'$ to denote that there is an injection $h: \widehat{n} \rightarrow I$ from indices of the rule r to indices of the network such that*

1. $\mathsf{q}(h(i)) = q_i$, *and* $p_i(\mathbf{x}(h(i)))$ *holds for each* $i \in \widehat{n}$,
2. $\mathsf{q}'(h(i)) = q_i'$ *for* $i \in \widehat{n}$,
3. $\mathsf{q}'(j) = \mathsf{q}(j)$ *for* $j \in (I \setminus range(h))$,
4. $\mathbf{x}'(h(i)) = 0$ *for* $i \in \widehat{n}$ *with* $op_i = reset$,
5. $\mathbf{x}'(h(i)) = \mathbf{x}(h(i))$ *for* $i \in \widehat{n}$ *with* $op_i = skip$, *and*
6. $\mathbf{x}'(j) = \mathbf{x}(j)$ *for* $j \in (I \setminus range(h))$.

The first condition asserts that r is enabled, i.e., that the process states $q_1, \ldots, q_n$ are matched by the corresponding process states in the configuration γ and that the corresponding guarded commands are enabled. The second condition means that in the transition from γ to γ', the states of processes that are matched (by h) with indices of r are changed according to r. The third condition asserts that the states of the other processes are unchanged. The fourth condition asserts that in the transition from γ to γ', the clock values of processes that are matched (by h) with indices of r are set to 0 if the corresponding guarded command contains *reset*, the fifth asserts that clocks are unchanged if the guarded command contains *skip*. The last condition asserts that clock values of unmatched processes are unchanged.

Let $\gamma = \langle I, c, \mathsf{q}, \mathbf{x} \rangle$ be configuration. For $\delta \in \mathcal{R}^{\geq 0}$, we use $\gamma^{+\delta}$ to denote the configuration $\langle I, c, \mathsf{q}, \mathbf{x}' \rangle$, where $x'(j) = x(j) + \delta$ for each $j \in I$.We say that γ performs a *timed transition* to a configuration γ', denoted $\gamma \longrightarrow_T \gamma'$, if there is a $\delta \in \mathcal{R}^{\geq 0}$ such that $\gamma' = \gamma^{+\delta}$. We use $\gamma \longrightarrow \gamma'$ to denote that either $\gamma \longrightarrow_D \gamma'$ or $\gamma \longrightarrow_T \gamma'$. We use $\xrightarrow{*}$ to denote the reflexive transitive closure of $\longrightarrow$.

3 Overview of the Reachability Algorithm

In this section we define the reachability problem, and give an overview of our method for solving it. Given a timed network $\langle C, Q, R \rangle$ together with states $c_{init} \in C$ and $q_{init} \in Q$ which we call *the initial controller state* and *the initial process state* respectively, we define an *initial configuration* γ_{init} of the timed network $\langle C, Q, R \rangle$ as a configuration of the form $\langle I, c_{init}, \mathsf{q}_{init}, \mathbf{x}_{init} \rangle$ where

$\mathtt{q}_{init}(j) = q_{init}$ and $\mathtt{x}_{init}(j) = 0$ for each $j \in I$. Thus, there is one initial configuration for each possible index set I. We say that a configuration γ is *reachable* if $\gamma_{init} \xrightarrow{*} \gamma$, for some initial configuration γ_{init}. We say that a set Γ of configurations is *reachable* if there is a reachable $\gamma \in \Gamma$.

We will present an algorithm for deciding whether a set Γ of configurations of a timed network is reachable. Note that in general, Γ will contain configurations of networks with infinitely many different sizes, where the size of a configuration is given by its index set. This means that we ask whether there is *some* size of the network such that a configuration with this size (as given by its index set) is reachable. In a typical situation, we are interested in verifying that Γ is an unreachable set of "bad" configurations, irrespective of the size of the network. If we include in Γ the bad configurations of all possible network sizes, and if our analysis finds Γ to be unreachable, this means that we have verified that the configurations in Γ are unreachable for all possible sizes of the network. For instance, we can verify correctness of an n-process mutual exclusion algorithm for all values of n simultaneously.

In Section 4, we will define a class of constraints for representing sets of configurations. A constraint ϕ denotes a (possibly infinite) set $[\![\phi]\!]$ of configurations. A finite set $\Phi = \{\phi_1, \ldots, \phi_n\}$ of constraints denotes the union of the denotations of its elements, i.e., $[\![\Phi]\!] = \bigcup_{i=1}^{n} [\![\phi_i]\!]$. Formally, the reachability problem is defined as follows.

Instance: A timed network $\langle C, Q, R \rangle$, an initial controller state c_{init}, an initial process state q_{init} and a finite set Φ of constraints.
Question: Does $[\![\Phi]\!]$ contain a reachable configuration?

To check the reachability of Φ we perform a reachability analysis backwards. Let *pre*(ϕ) denote the set $\{\gamma \ : \ \exists\gamma' \in [\![\phi]\!] : \gamma \longrightarrow \gamma'\}$, and *pre*$(\Phi)$ denote the set $\{\gamma \ : \ \exists\gamma' \in [\![\Phi]\!] : \gamma \longrightarrow \gamma'\}$. Note that *pre*$(\Phi)$ is equivalent to $\cup_{\phi\in\Phi} pre(\phi)$. Starting from Φ we define the sequence $\Phi_0, \Phi_1, \Phi_2, \ldots$ of finite sets of constraints by $\Phi_0 = \Phi$ and $\Phi_{j+1} = \Phi_j \cup pre(\Phi_j)$. Intuitively, Φ_j denotes the set of configurations from which Φ is reachable by a sequence of at most j transitions. Note that the sequence is increasing i.e., that $[\![\Phi_0]\!] \subseteq [\![\Phi_1]\!] \subseteq [\![\Phi_2]\!] \subseteq \cdots$. In the next paragraph, we will prove that the iteration converges (using Theorem 4), i.e., that there is a k such that $[\![\Phi_k]\!] = [\![\Phi_{k+1}]\!]$, implying that $[\![\Phi_k]\!] = [\![\Phi_j]\!]$ for all $j \geq k$. It follows that Φ is reachable if and only if there is an initial configuration γ_{init} such that $\gamma_{init} \in [\![\Phi_k]\!]$, which is easily checked since Φ_k is a *finite* set of constraints.

To prove convergence, we introduce, in Definition 6, a quasi-order $\preceq$ on constraints by defining $\phi \preceq \phi'$ to denote that $[\![\phi']\!] \subseteq [\![\phi]\!]$. In Theorem 4, we will show that $\preceq$ is a well quasi-ordering on the set of constraints, i.e., that in any infinite sequence $\phi_0 \ \phi_1 \ \phi_2 \ \cdots$ of constraints, there are indices $i < j$ such that $\phi_i \preceq \phi_j$. This implies that any increasing sequence $[\![\Phi_0]\!] \subseteq [\![\Phi_1]\!] \subseteq [\![\Phi_2]\!] \subseteq \cdots$ of finite sets of constraints will converge, since otherwise we could extract an infinite sequence $\phi_0 \ \phi_1 \ \phi_2 \ \cdots$ of constraints (where ϕ_i is chosen such that $\phi_i \in \Phi_i$ but $[\![\phi_i]\!] \not\subseteq [\![\Phi_{i-1}]\!]$) for which there are no indices $i < j$ such that $\phi_i \preceq \phi_j$.

Summarizing, we have established the following theorem

Theorem 1. *The reachability problem for families of timed networks is decidable*

Proof. Follows from the preceding discussion, using Theorem 2 (decidability of $\preceq$), Theorem 3 (computability of *pre*) and Theorem 4 (well quasi-orderedness of $\preceq$).

The following sections contain the above mentioned definitions and lemmas. In Section 4, we define the constraint system. In Section 5, we show that $pre(\phi)$ can be computed and represented by a finite set of constraints whenever ϕ is a contraint. Finally, in Section 6, we show that the relation $\preceq$ is a well quasi-ordering on the set of constraints.

4 A Constraint System for Timed Networks

In this section we introduce a constraint system for timed networks. Our constraint system generalizes the notion of regions, employed for the analysis of timed automata [ACD90]. We use a representation of constraints, which is similar to a representation of regions used by Godskesen [God94].

For a quasi-order $\sqsubseteq$[3] on some set, we use $a_1 \equiv a_2$ to denote that $a_1 \sqsubseteq a_2$ and $a_2 \sqsubseteq a_1$, and use $a_1 \sqsubset a_2$ to denote that $a_1 \sqsubseteq a_2$ and $a_2 \not\sqsubseteq a_1$. For a real number $x \in \mathcal{R}^{\geq 0}$, let $\lfloor x \rfloor$ denote its integer part, and let $fract(x)$ denote its fractional part.

Definition 4. *Let $\langle C, Q, R\rangle$ be a family of timed networks. Let max be the maximum constant occurring in the guarded commands in R. A* constraint *ϕ of $\langle C, Q, R\rangle$ is a tuple $\langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq\rangle$ where*

- *$c \in C$ is a controller state,*
- *m is a natural number, where $\widehat{m}$ intuitively denotes a set of indices of processes constrained by ϕ,*
- *$\mathbf{q} : \widehat{m} \mapsto Q$ is a mapping from indices to process states,*
- *$\mathbf{k} : \widehat{m} \mapsto \{0, \ldots, max\}$ maps each index to a natural number not greater than max,*
- *$\sqsubseteq$ is a quasi-order on the set $\widehat{m} \cup \{\bot, \top\}$ which satisfies*
  - *the elements $\bot$ and $\top$ are minimal and maximal elements of $\sqsubseteq$, respectively, with $\bot \sqsubset \top$* [4]*,*
  - *$j \equiv \bot$ or $j \equiv \top$ whenever $\mathbf{k}(j) = \max$, for $j \in \widehat{m}$, and*
  - *$\mathbf{k}(j) = \max$ whenever $j \equiv \top$, for $j \in \widehat{m}$.*

Intuitively, a constraint denotes a set of configurations of networks in the family. The constraint $\langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq\rangle$ represents the set of configurations with controller state c in which each index $j \in \widehat{m}$ represents a process which has control state $\mathbf{q}(j)$, for which $\mathbf{k}(j)$ is either *max* or the integer part of its clock, whichever

[3] A quasi-order is a reflexive and transitive relation.

[4] Note that $\bot, \top \not\in \widehat{m}$.

is least, for which $j \equiv \perp$ iff the integer part of the clock is at most *max* and the fractional part of the clock is 0, and for which $j \equiv \top$ iff the clock value is more than *max*. Furthermore, the fractional parts of the clocks corresponding to indices j with $j \sqsubset \top$ are ordered exactly according to $\sqsubseteq$. This implies, among other things, that for clock values that are larger than *max*, a constraint gives no information about the difference between the actual clock value and *max*. The meaning of constraints is made formal in the following definition.

Definition 5. *Let $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq\rangle$ be a constraint and let $\gamma = \langle I, c, \mathbf{q}, \mathbf{x}\rangle$ be a configuration*[5]*. We define $\gamma \in [\![\phi]\!]$ to mean that there is an injection $h : \widehat{m} \mapsto I$ from the indices of ϕ to the indices of γ such that for all $j, j_1, j_2 \in \widehat{m}$*

- $\mathbf{q}(h(j)) = \mathbf{q}(j)$,
- $\min(max, \lfloor \mathbf{x}(h(j)) \rfloor) = \mathbf{k}(j)$,
- *$j \equiv \perp$ if and only if $\mathbf{x}(h(j)) \leq max$ and $fract(\mathbf{x}(h(j))) = 0$,*
- *$j \equiv \top$ if and only if $\mathbf{x}(h(j)) > max$, and*
- *if $j_1, j_2 \not\equiv \top$ then $fract(\mathbf{x}(h(j_1))) \leq fract(\mathbf{x}(h(j_2)))$ if and only if $j_1 \sqsubseteq j_2$.*

Note that a constraint ϕ defines conditions on states and clock values which should be satisfied by *some* set of processes (those represented by indices in $range(h)$) in the configuration γ in order for γ to be included in $[\![\phi]\!]$. The constraint puts no requirements on processes whose indices are outside $range(h)$.

Definition 6. *Define the ordering $\preceq$ on constraints by $\phi \preceq \phi' \overset{def}{=} [\![\phi']\!] \subseteq [\![\phi]\!]$.*

Intuitively, $\phi \preceq \phi'$ means that ϕ' is "stronger" than ϕ, or that ϕ' "entails" ϕ. The following theorem shows how to compute $\preceq$.

Theorem 2. *Let $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq\rangle$ and $\phi' = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq'\rangle$ be constraints. We have $\phi \preceq \phi'$ if and only if there is an injection $g : \widehat{m} \mapsto \widehat{m'}$ such that*

- *$c = c'$,*
- *for all $j \in \widehat{m}$ we have*
  - $\mathbf{q}'(g(j)) = \mathbf{q}(j)$,
  - $\mathbf{k}'(g(j)) = \mathbf{k}(j)$,
  - *$g(j) \equiv' \perp$ iff $j \equiv \perp$,*
  - *$g(j) \equiv' \top$ iff $j \equiv \top$,*
- *if $j_1, j_2 \in \widehat{m}$ then $g(j_1) \sqsubseteq' g(j_2)$ if and only if $j_1 \sqsubseteq j_2$.*

5 Computing *pre*

In this section we show, for a given constraint ϕ, how to compute $pre(\phi)$, defined as $\{\gamma : \exists \gamma' \in [\![\phi]\!] : \gamma \longrightarrow \gamma'\}$. Since the transition relation is the union of a discrete and a timed transition relation, we will compute $pre(\phi)$ as $pre_D(\phi) \cup pre_T(\phi)$, where $pre_D(\phi)$ is the set $\{\gamma : \exists \gamma' \in [\![\phi]\!] : \gamma \longrightarrow_D \gamma'\}$, and where $pre_T(\phi)$ is the set $\{\gamma : \exists \gamma' \in [\![\phi]\!] : \gamma \longrightarrow_T \gamma'\}$.

[5] Observe that the controller states are the same in ϕ and γ.

5.1 Computing pre_D

We will compute $pre_D(\phi')$ as $\cup_{r \in R} pre(r, \phi')$, where $pre(r, \phi')$ denotes the set $\{\gamma \;:\; \exists \gamma' \in [\![\phi']\!] \;:\; \gamma \xrightarrow{r}_D \gamma'\}$ of configurations from which ϕ' is reachable through a single application of r[6].

Let $r = \langle \langle c, c' \rangle, \langle q_1, p_1(x) \longrightarrow op_1, q_1' \rangle, \ldots, \langle q_n, p_n(x) \longrightarrow op_n, q_n' \rangle \rangle$ and let $\phi' = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq' \rangle$. We will compute a representation of $pre(r, \phi')$ as a finite set of constraints. Each constraint ϕ with $[\![\phi]\!] \subseteq pre(r, \phi')$ will be obtained from a particular way of matching indices of ϕ' with indices of r. Each such matching gives rise to a set of constraints ϕ with $[\![\phi]\!] \subseteq pre(r, \phi')$, namely those constraints that are consistent both with the conditions imposed by ϕ' according to Definition 5, and with the conditions imposed by r according to Definition 3.

Let $p(x)$ be a guard and let $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$ be a constraint. For $j \in \widehat{m}$, we use $\langle \phi, j \rangle \models p(x)$ to mean that p *is satisfied at index* j *in* ϕ. For instance, if $p(x)$ is of form $k \leq x$ for some $k \in \{0, \ldots, max\}$, then $\langle \phi, j \rangle \models p(x)$ iff $\mathbf{k}(j) \geq k$. We can derive analogous expressions for other forms of $p(x)$.

A matching will be represented by a *partial* injection g' from $\widehat{m'}$ to $\widehat{n}$: each index $j \in domain(g')$ of ϕ' is matched with a unique index $g'(j)$ of r (note that $domain(g') \subseteq \widehat{m'}$). Indices in $(\widehat{m'} \setminus domain(g'))$ represent processes which are constrained by ϕ' but are not matched with any index of r. In the indices of ϕ, we must also include the $n - |range(g')|$ indices of r which are not matched with any index of ϕ'. Thus, let $m = m' + (n - |range(g')|)$ be the number of indices of ϕ. Define an *extension* g of g' to be a *surjective* partial injection $g : \widehat{m} \mapsto \widehat{n}$, with domain $domain(g) = domain(g') \cup (\widehat{m} \setminus \widehat{m'})$, such that $g(j) = g'(j)$ for each $j \in domain(g')$. [7] It follows from the definition of extension that $\widehat{m'} \setminus domain(g')$ is not in $domain(g)$ and that g in addition maps each $j \in (\widehat{m} \setminus \widehat{m'})$ to a unique $g(j) \in (\widehat{n} \setminus range(g'))$.

Lemma 1. *If* $\phi' = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq' \rangle$ *is a constraint and* r *is a rule, as above, then* $pre(r, \phi')$ *is the denotation of the set of constraints of form* $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$*, for which there is an extension* g *of a partial injection* g' *from* $\widehat{m'}$ *to* $\widehat{n}$*, where* $m = m' + (n - |range(g')|)$*, which satisfies the following conditions*[8].

1. $\mathbf{q}(j) = q_{g(j)}$ *and* $\langle \phi, j \rangle \models p_{g(j)}(x)$ *for each* $j \in domain(g)$,
2. $\mathbf{q}'(j) = q'_{g'(j)}$ *for* $j \in domain(g')$,
3. $\mathbf{q}'(j) = \mathbf{q}(j)$ *for* $j \in \widehat{m'} \setminus domain(g')$,
4. $\mathbf{k}'(j) = 0$ *and* $j \equiv' \bot$ *if* $j \in domain(g')$ *and* $op_{g'(j)} = reset$,
5. *For all* j *such that either* $j \in domain(g')$ *and* $op_{g'(j)} = skip$*, or such that* $j \in \widehat{m'} \setminus domain(g')$*,*
   we have $\mathbf{k}'(j) = \mathbf{k}(j)$*, and* $j \equiv' \bot$ *iff* $j \equiv \bot$*, and* $j \equiv' \top$ *iff* $j \equiv \top$.

[6] In order to be consistent with the notation in Section 2, we use the primed version of the constraint to refer to the constraint after a transition, and an unprimed version of the constraint to refer to the constraint before a transition.

[7] note that $\widehat{m} \setminus \widehat{m'} = \{m' + 1, m' + 2, \ldots, m\}$

[8] Note that we implicitly require the controller states c and c' of r to coincide with the controller states c and c' of ϕ and ϕ', respectively.

6. *For each j_1 and j_2 such that for $i = 1, 2$ either $j_i \in domain(g')$ and $op_{g'(j_i)} =$ skip, or $j_i \in \widehat{m'} \setminus domain(g')$, we have $j_1 \sqsubseteq' j_2$ if and only if $j_1 \sqsubseteq j_2$.*

The above list of conditions captures the semantics of $\xrightarrow{r}_D$, given the correspondences between the indices of r, ϕ' and ϕ which are given by g and g'. Note the close correspondence between the conditions of the lemma and the conditions of transitions in Definition 3. The conditions on controller states are implicitly included by our notation, which requires that the controller states of ϕ and ϕ' be the controller states of r. Condition 1 state that r must be enabled in a configuration satisfying ϕ. Conditions 2 and 3 capture the conditions on states of the processes: after a transition, states of processes with indices in *domain*(g') are constrained by 2; and processes with indices in $\widehat{m'} \setminus domain(g')$ are unaffected by the rule (condition 3). Condition 4 describes the effect of a *reset* statement: the clock value becomes 0 in ϕ'. Finally, conditions 5 and 6 assert that for indices that correspond to a *skip* statement, or for indices not matched by r (and hence unaffected by the transition), the clock values are unchanged by a transition.

5.2 Computing pre_T

First, we define a relation pre_t which we later use (Lemma 3) to compute pre_T.

Definition 7. *For a constarint $\phi' = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq' \rangle$ we define $pre_t(\phi')$ to be the denotation of the set of constraints of form $\phi = \langle c', m', \mathbf{q}', \mathbf{k}, \sqsubseteq \rangle$ satisfying either of the following two conditions.*

1. *for some $j \in \hat{m}$ we have $j \equiv' \bot$, there is no $j \in \hat{m}$ such that $j \equiv' \bot$ and $\mathbf{k}'(j) = 0$, and the following three conditions hold.*
   - $\mathbf{k}(j) = \mathbf{k}'(j) - 1$ *if* $j \equiv' \bot$,
   - $\mathbf{k}(j) = \mathbf{k}'(j)$ *if* $j \not\equiv' \bot$,
   - $j_1 \sqsubseteq j_2$ *if and only if either*
     (a) $j_2 \equiv' \top$, *or*
     (b) $j_2 \equiv' \bot$ *and* $j_1 \not\equiv' \top$, *or*
     (c) $j_1 \sqsubseteq' j_2$ *and* $\bot \sqsubset' j_1, j_2 \sqsubset' \top$.
2. *There is no $j \in \hat{m}$ such that $j \equiv' \bot$ and the following four conditions hold:*
   - $\mathbf{k} = \mathbf{k}'$,
   - *whenever* $\bot \sqsubset' j_1, j_2 \sqsubset' \top$ *we have* $j_1 \sqsubseteq j_2$ *if and only if* $j_1 \sqsubseteq' j_2$,
   - *whenever* $j \equiv' \top$ *we have* $j \equiv \bot$ *or* $j \equiv \top$,
   - $\sqsubset' \neq \sqsubset$.

Intuitively, the first case captures the situation where there are indices with fractional parts of some clocks being 0. The second case captures the situation when no clocks have fractional parts equal to 0.

Lemma 2. *There is no infinite sequence $\phi_0, \phi_1, \phi_2, \ldots$ of constraints, such that $\phi_{i+1} \in pre_t(\phi_i)$.*

Definition 8. *For a set of constraints Φ' and a natural number i, we define $pre_t{}^i(\Phi', i)$ inductively as follows.*

- $pre_t{}^0(\Phi') = \Phi'$; *and*
- $pre_t{}^{i+1}(\Phi') = \{\phi \;:\; \exists \phi' \in pre_t{}^i(\Phi') : \phi \in pre_t(\phi')\}$.

We define $pre_t{}^*(\Phi') = \cup_{i \geq 0} pre_t{}^i(\Phi')$.

Sometimes we write $pre_t{}^*(\phi')$ instead of $pre_t{}^*(\{\phi'\})$

Lemma 3. *If ϕ' is a constraint, then $pre_T(\phi')$ is the denotation of the set of constraints in the set $pre_t{}^*(\{\phi'\})$. In other words*

$$pre_T(\phi') \;=\; [\![pre_t{}^*(\phi)]\!]$$

The computability of pre_T follows from Lemma 3 and Lemma 2.

5.3 Computing *pre*

By combining the rules for computing $pre_D(\phi)$ in Lemma 1 and the rules for computing $pre_T(\phi)$ in Lemma 3, we obtain the following theorem.

Theorem 3. *If ϕ is a constraint, then we can compute a finite set Φ of constraints such that $[\![\Phi]\!] = pre(\phi)$.*

6 The entailment ordering is a well quasi-ordering

In this section, we shall prove that the preorder $\preceq$ on constraints is a well quasi-ordering. We will first review some standard results from the literature concerning well quasi-orderings ([Hig52]), and then apply them to our constraint system.

Definition 9. *Let A be a set. A quasi-order $\preceq$ on A is a binary relation over A which is reflexive and transitive. A quasi-order $\preceq$ is a* well quasi-ordering *(wqo) if in each infinite sequence $a_0\ a_1\ a_2\ a_3 \cdots$ of elements in A, there are indices $i < j$ such that $a_i \preceq a_j$.*

We shall now restate two standard lemmas, which allow us to lift well quasi-orderings from elements to bags and to sequences. Let A^* denote the set of finite strings over A, and let Let A^B denote the set of finite bags over A. An element of A^* and of A^B can be represented as a mapping $w\colon \widehat{|w|} \mapsto A$ where $|w|$ is the size of the bag or the length of the sequence. Given a quasi-order $\preceq$ on a set A, define the quasi-order $\preceq^*$ on A^* by letting $w \preceq^* w'$ if and only if there is a monotone[9] injection $h\colon \widehat{|w|} \mapsto \widehat{|w'|}$ such that $w(j) \preceq w'(h(j))$ for $1 \leq j \leq |w|$. Define the quasi-order $\preceq^B$ on bags of A by $w \preceq^* w'$ if and only if there is a (not necessarily monotonic) injection $h\colon \widehat{|w|} \mapsto \widehat{|w'|}$ such that $w(j) \preceq w'(h(j))$ for $1 \leq j \leq |w|$.

[9] meaning that $h(j_1) \leq h(j_2)$ if and only if $j_1 \leq j_2$

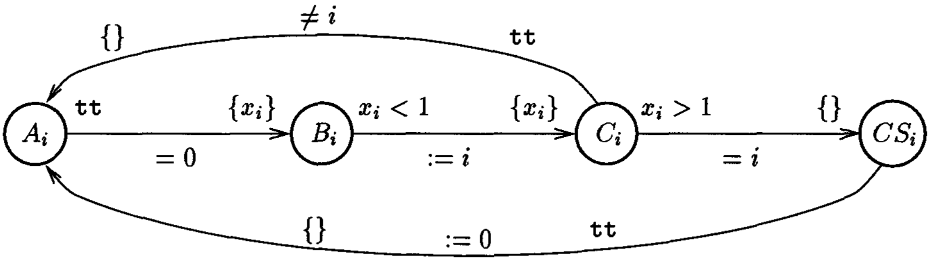

Fig. 1. Fischer's Protocol for Mutual Exclusion.

Lemma 4. *If $\preceq$ is a wqo on A, then $\preceq^*$ is a wqo on A^* and $\preceq^B$ is a wqo on A^B.*

Proof. The proof can be found in [Hig52].

Let ϕ be a constraint $\langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$. For each $j \in \widehat{m} \cup \{\bot, \top\}$, define the *rank* of j in ϕ to be the number of equivalence classes of $\sqsubseteq$ which are less than or equal (wrp. to $\preceq$) to the equivalence class containing j. In other words, the rank of j is the maximum k such that there is a sequence $\bot \sqsubset j_1 \sqsubset \cdots \sqsubset j_k = j$. Note that the rank of $\top$ is equal to the number of equivalence classes of $\equiv$. Define the rank of ϕ as the rank of $\top$ in ϕ.

Let r be the rank of the constraint $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$. For $i \in \widehat{r}$, define $\phi[i]$ to be the bag of pairs of the form $\langle q, k \rangle$ such that $u(j) = q$ and $\mathbf{k}(j) = k$ for some j with rank i in ϕ. Define the ordering $\preceq$ on these pairs to be the identity relation on pairs of the form $\langle q, k \rangle$. Since there are finitely many such pairs, $\preceq$ is trivially a well quasi-ordering.

Lemma 5. *Let $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$ and $\phi' = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq' \rangle$ be constraints with ranks r and r'. We have $\phi \preceq \phi'$ if and only if $c = c'$, $\phi[0] \preceq^B \phi'[0]$, $\phi[r] \preceq^B \phi'[r']$, and there is a monotonic injection $h : (\widehat{r-1}) \mapsto (\widehat{r'-1})$ such that $\phi[i] \preceq^B \phi'[h(i)]$ for all $i \in (\widehat{r-1})$.*

Theorem 4. *The relation $\preceq$ on the set of constraints is a well quasi-ordering.*

Proof. The proof follows from Lemma 5 and repeated application of Lemma 4.

7 Example: Fischer's Protocol

As an illustration of our method, we model Fischer's protocol [SBK92], and show how an automatic verification algorithm would go about verifying mutual exclusion. The purpose of the protocol is to guarantee mutual exclusion in a concurrent system consisting of an arbitrary number of processes, using clocks and a shared variable. Each process has a local clock, and runs a protocol before entering the critical section. Each process has a local control state, which in our model assumes values in the set $\{A, B, C, CS\}$ where A is the initial state and CS represents the critical section. The processes also read from and write to a shared variable whose value is either $\perp$ or the index of one of the processes. A description in a graphical pseudo-code (taken from [KLL$^+$97]) of the behavior of a process with index i is given in Figure 1.

Intuitively, the protocol behaves as follows: A process wishing to enter the critical section starts in state A. If the value of the shared variable is $\perp$, the process can proceed to state B and reset its local clock. From state B, the process can proceed to state C if the clock value is still less than 1. In other words, the clock implements a timeout which guarantees that the process either stays in state B at most one time unit, or gets stuck in B forever. When moving from B to C, the process sets the value of the shared variable to its own index i and again resets its clock. From state C, the process can proceed to the critical section if the clock is strictly more than 1 and if the value of the shared variable is still i, the index of the process. Thus, in state C the clock enforces a delay which is longer than the length of the timeout in state B. Finally, when exiting the critical section, the process resets the shared variable to $\perp$. Processes that get stuck in state C can reenter the protocol by returning to state A. Since we do not intend to model liveness properties, such as e.g., absence of starvation, we do not impose requirements that force processes to change their state[10].

$$\begin{aligned}
&\mathit{initiate} : \langle\langle \mathit{udf}, \mathit{udf}\rangle, \langle A, x \geq 0 \longrightarrow \mathit{reset}, B\rangle\rangle \\[1ex]
&\mathit{choose}_1 : \langle\langle \mathit{udf}, \mathit{df}\rangle, \langle B, x < 1 \longrightarrow \mathit{reset}, C^\dagger\rangle\rangle \\
&\mathit{choose}_2 : \langle\langle \mathit{df}, \mathit{df}\rangle, \langle B, x < 1 \longrightarrow \mathit{reset}, C^\dagger\rangle, \langle q^\dagger, \mathit{skip}, q\rangle\rangle \\
&\mathit{choose}_3 : \langle\langle \mathit{df}, \mathit{df}\rangle, \langle B^\dagger, x < 1 \longrightarrow \mathit{reset}, C^\dagger\rangle\rangle \\[1ex]
&\mathit{enter} : \quad \langle\langle \mathit{df}, \mathit{df}\rangle, \langle C^\dagger, x > 1 \longrightarrow \mathit{skip}, CS^\dagger\rangle\rangle \\[1ex]
&\mathit{fail}_1 : \quad \langle\langle \mathit{udf}, \mathit{udf}\rangle, \langle C, \mathit{skip}, A\rangle\rangle \\
&\mathit{fail}_2 : \quad \langle\langle \mathit{df}, \mathit{df}\rangle, \langle C, \mathit{skip}, A\rangle\rangle \\[1ex]
&\mathit{exit}_1 : \quad \langle\langle \mathit{df}, \mathit{udf}\rangle, \langle CS^\dagger, \mathit{skip}, A\rangle\rangle \\
&\mathit{exit}_2 : \quad \langle\langle \mathit{df}, \mathit{udf}\rangle, \langle CS, \mathit{skip}, A\rangle, \langle q^\dagger, \mathit{skip}, q\rangle\rangle \\
&\mathit{exit}_3 : \quad \langle\langle \mathit{udf}, \mathit{udf}\rangle, \langle CS, \mathit{skip}, A\rangle\rangle
\end{aligned}$$

Fig. 2. Rules for Modeling Fischer's protocol

We can model the protocol in our timed networks formalism. The controller state is either *udf*, indicating that the value of the shared variable is undefined, or *df*, indicating that the value of the shared variable is defined. The set of

[10] In fact, our formalism cannot express such requirements, although they can be added in terms of e.g., fairness constraints.

process states is given by $\{A, B, C, CS, A^\dagger, B^\dagger, C^\dagger, CS^\dagger\}$. The states marked with † correspond to configurations where the value of the shared variable is equal to the index of that particular process.

A straightforward translation of the description in Figure 1 yields the set of rules in Figure 2. We use q to denote an arbitrary process state. We use *skip* to denote the guarded command $0 \leq x \longrightarrow skip$.

Using the method described in this paper, it is possible to verify that there is never more than one process in its critical section, by checking that the set of constraints with $m = 2$ that contain exactly two occurrences of CS or $CS^\dagger$ is unreachable. More details are in the full paper.

References

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proc. 5th IEEE Int. Symp. on Logic in Computer Science*, pages 414–425, Philadelphia, 1990.

[ACHH92] R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verificationof hybrid systems. In Grossman, Nerode, Ravn, and Rischel, editors, *Hybrid Systems*, number 736 in Lecture Notes in Computer Science, pages 209–229, 1992.

[AČJYK96] Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. 11th IEEE Int. Symp. on Logic in Computer Science*, pages 313–321, 1996.

[AJ96] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

[AK95] Parosh Aziz Abdulla and Mats Kindahl. Decidability of simulation and bisimulation between lossy channel systems and finite state systems. In Lee and Smolka, editors, *Proc. CONCUR '95, 6th Int. Conf. on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 333 – 347. Springer Verlag, 1995.

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, 1991.

[BS95] O. Burkart and B. Steffen. Composition, decomposition, and model checking of pushdown processes. *Nordic Journal of Computing*, 2(2):89–125, 1995.

[Čer94] K. Čerāns. Deciding properties of integral relational automata. In Abiteboul and Shamir, editors, *Proc. ICALP '94*, volume 820 of *Lecture Notes in Computer Science*, pages 35–46. Springer Verlag, 1994.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CGJ95] E. M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstraction and regular languages. In Lee and Smolka, editors, *Proc. CONCUR '95, 6th Int. Conf. on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 395–407. Springer Verlag, 1995.

[Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89:144–179, 1990.

[GMMP91] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzè. A unified high-level Petri net formalism for time-critical systems. *IEEE Trans. on Software Engineering*, 17(2):160–172, 1991.

[God94] J.C. Godskesen. *Timed Modal Specifications.* PhD thesis, Aalborg University, 1994.

[GS92] S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

[Hen95] T.A. Henzinger. Hybrid automata with finite bisimulations. In *Proc. ICALP '95*, 1995.

[Hig52] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, 2:326–336, 1952.

[JM95] P. Jančar and F. Moller. Checking regular properties of Petri nets. In *Proc. CONCUR '95, 6th Int. Conf. on Concurrency Theory*, pages 348–362, 1995.

[JP93] B. Jonsson and J. Parrow. Deciding bisimulation equivalences for a class of non-finite-state programs. *Information and Computation*, 107(2):272–302, Dec. 1993.

[KLL$^+$97] K.J. Kristoffersen, F. Larroussinie, K. G. Larsen, P. Pettersson, and W. Yi. A compositional proof of a real-time mutual exclusion protocol. In *TAPSOFT '97 7th International Joint Conference on the Theory and Practice of Software Development*, Lecture Notes in Computer Science, Lille, France, April 1997. Springer Verlag.

[KM89] R.P. Kurshan and K. McMillan. A structural induction theorem for processes. In *Proc. 8th ACM Symp. on Principles of Distributed Computing, Canada*, pages 239–247, Edmonton, Alberta, 1989.

[LSW95] K.G. Larsen, B. Steffen, and C. Weise. Fischer's protocol revisited: a simples proof using modal constraints. In *4th DIMACS Workshop on Verification and Control of Hybrid Systems*, New Brunswick, New Jersey, Oct. 1995.

[SBK92] F. B. Schneider, Bloom B, and Marzullo K. Putting time into proof outlines. In de Bakker, Huizing, de Roever, and Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, 1992.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.

[Wol86] Pierre Wolper. Expressing interesting properties of programs in propositional temporal logic (extended abstract). In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–193, Jan. 1986.

[Yi91] Wang Yi. CCS + Time = an interleaving model for real time systems. In Leach Albert, Monien, and Rodriguez Artalejo, editors, *Proc. ICALP '91*, volume 510 of *Lecture Notes in Computer Science*. Springer Verlag, 1991.