ELSEVIER

Theoretical Computer Science 290 (2003) 241–264

Theoretical Computer Science

www.elsevier.com/locate/tcs

Model checking of systems with many identical timed processes ☆

Parosh Aziz Abdulla, Bengt Jonsson *

Department of Computer Systems, Uppsala University, PO Box 325 S-751 05 Uppsala, Sweden

Received November 1999; accepted June 2001
Communicated by W. Thomas

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, lossy channel systems, context-free and push-down processes. 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.

Finally, we show undecidability of the *recurrent state problem*: given a state in a timed network, check whether there is a computation of the network visiting the state infinitely often. This implies undecidability of model checking for any temporal logic which is sufficiently expressive to encode the recurrent state problem, such as PTL, CTL, etc.

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 atleast two ways in which a program can

☆ An extended abstract of this paper appeared in the proceedings of TACAS'98: Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems.

* Corresponding author.

E-mail addresses: parosh@docs.uu.se (P.A. Abdulla), bengt@docs.uu.se (B. Jonsson).

0304-3975/02/$ - see front matter
PII: S0304-3975(01)00330-9

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 [7, 9, 13], hybrid automata [24], data-independent systems [29, 42], relational automata [10, 14, 15], Petri nets [18, 26–28], pushdown processes [12, 40] and lossy channel systems [4, 5]. 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 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 ([20]), and (using a manually supplied induction hypothesis) for more general classes of parameterized systems [17, 32, 43].

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 [7] 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 [41, 22]) 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 [36, 16] and assertional verification [38]. 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 fixedpoint 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 a single 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 (nonparameterized) timed automata [7]. A constraint rep-

resents conditions on a potentially unbounded number of processes and their clocks. In contrast to the situation for timed automata [7], where for each program there are finitely many regions, there is in general no bound on the number of constraints that can appear in the analysis of a given timed network. To handle this, we introduce an entailment ordering on constraints. The key step in our proof of decidability consists in proving that this relation is a well quasi-ordering, implying that the above mentioned fixedpoint iteration converges.

Our results also demonstrate the strength and applicability of the general framework of *well quasi-orderings* described in [19, 1, 2]. Using that framework, we can also 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 [5, 30, 1]. 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 [35, 21, 11] and Timed CCS [44]. Our decidability result can be translated to decidability results for variants of these formalisms. It is known that reachability is undecidable for timed 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 cannot 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 [37]. Our algorithm can then be used to perform an automatic verification of the fact that mutual exclusion will be satisfied regardless of the number of processes. 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., [8]). Kristoffersen et al. [31] describes an experiment where the number of processes is 50. In [33], a constraint-based proof methodology is used to perform a manual verification of the protocol.

Finally, we show the undecidability of the *recurrent state problem*: given a state in a timed network, check whether there is a computation of the network visiting that state infinitely often. Our proof is based on the observation that timed networks can "simulate" lossy channel systems, for which we have earlier shown the recurrent state problem to be undecidable [3]. Since temporal logics such as PTL and CTL are sufficiently powerful to express the recurrent state problem, it follows that model checking for these logics, interpreted over timed networks, is undecidable.

Outline: The rest of the paper is structured as follows. In the next section, we present our model of timed networks. In Section 3 we describe Fischer's protocol [37] in our model. An overview of the reachability algorithm is presented in Section 4. In Section 5 we present our constraint system. In Section 6 we present a procedure for calculating the pre-image of a set of states which are represented by a constraint. In

Section 7 we prove that the entailment ordering on the constraint system is a well quasi-ordering, which implies that our algorithm always terminates. In Section 8 we show undecidability of the recurrent state problem.

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

Definition 2.1. 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

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

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 (see Definition 2.3).

Definition 2.2. A *configuration* γ of a timed network $\langle C,Q,R\rangle$ is quadruple of form $\langle I,c,\mathtt{q},\mathtt{x}\rangle$, where I is a finite *index set*, $c\in C$, $\mathtt{q}: I\to Q$, and $\mathtt{x}: I\to\mathscr{R}^{\geqslant 0}$.

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[2] by I. The state of the controller is given by c, the states of the processes are given by the mapping $\mathtt{q}$ from indices to process states, and the clock values are given by the mapping $\mathtt{x}$ from indices to nonnegative real numbers.

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

Definition 2.3. 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 $p_i(x)\to op_i$ for $i=1,\ldots,n$. Consider two configurations $\gamma=\langle I,c,\mathtt{q},\mathtt{x}\rangle$ and $\gamma'=\langle I,c',\mathtt{q}',\mathtt{x}'\rangle$, 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 $\gamma\overset{r}{\to}_D\gamma'$ to denote that there is an injection $h:\hat{n}\to I$ from indices of the rule r to indices of the network such that

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

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

[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.

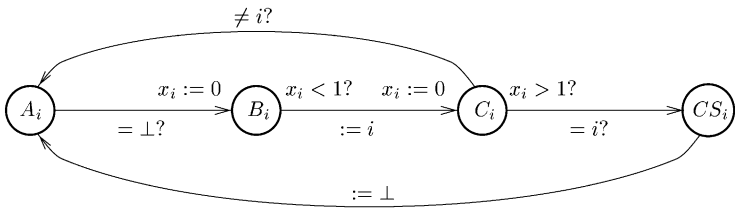

Fig. 1. Fischer's protocol for mutual exclusion.

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 or 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 $\gamma = \langle I, c, \mathrm{q}, \mathrm{x}\rangle$ be a configuration. For $\delta \in \mathscr{R}^{\geqslant 0}$, we use $\gamma^{+\delta}$ to denote the configuration $\langle I, c, \mathrm{q}, \mathrm{x}'\rangle$, where $x'(j) = x(j) + \delta$ for each $j \in I$. We say that γ performs a *timed transition* to a configuration γ', denoted $\gamma \rightarrow_T \gamma'$, if there is a $\delta \in \mathscr{R}^{\geqslant 0}$ such that $\gamma' = \gamma^{+\delta}$. We use $\gamma \rightarrow \gamma'$ to denote that either $\gamma \rightarrow_D \gamma'$ or $\gamma \rightarrow_T \gamma'$. We use $\stackrel{*}{\rightarrow}$ to denote the reflexive transitive closure of $\rightarrow$.

3. Example: Fischer's protocol

As an illustration we describe Fischer's protocol [37] in our model. The protocol has been used as a measure of the performance of tools for verification of timed automata. The example was suggested by Fred Schneider and has been verified manually (e.g., [6]) and using theorem provers (e.g., [39]).

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 [31]) of the behavior of a process with index i is given in Fig. 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. [3]

A rough argument for the correctness of the protocol goes as follows. The conditions on the shared variable ensure that a process cannot reach B if any other process is in C or CS. The timing conditions on the clocks ensure that a process cannot move from C to CS if some other process is still in B. Thus, if a set of processes start the mutual exclusion protocol and all arrive in C, then the process which was the last to enter C will read its own identity in the shared variable and enter the critical section.

We will now model the protocol in our timed networks formalism. We let the local state of each process, in addition to its control state, indicate whether or not the value of the shared variable is equal to the index of the process. As global information, we must record whether the value of the shared variable is $\perp$: this is done using a controller. Summarizing, 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 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 Fig. 1 yields the set of rules in Fig. 2. We use q to denote an arbitrary process state. We use *skip* to denote the guarded command $0 \leqslant x \to skip$.

An example of a configuration is $\gamma = \langle I, c, \mathrm{q}, \mathrm{x}\rangle$, where $I = \{1,2,3\}$, $c = df$, $\mathrm{q}(1) = C$, $\mathrm{q}(2) = C^\dagger$, $\mathrm{q}(3) = A$, $\mathrm{x}(1) = 0.5$, $\mathrm{x}(2) = 0.3$, and $\mathrm{x}(3) = 1.2$. Intuitively, γ is a configuration in an instance of Fischer's protocol, with a controller and three processes. The processes have indices 1, 2, and 3, respectively. To simplify the notation in the following example, we use a tuple notation, so we write e.g., the value of q in the definition of γ above as $\langle C, C^\dagger, A\rangle$. We use a similar notation to describe the value of x.

Consider Definition 2.3, and the configurations

$\gamma_1 = \langle \{1,2,3\}, df, \langle 0.2, 0.5, 0.9\rangle, \langle C^\dagger, B, A\rangle\rangle$, and

$\gamma_2 = \langle \{1,2,3\}, df, \langle 0.2, 0, 0.9\rangle, \langle C, C^\dagger, A\rangle\rangle$. It follows that $\gamma_1 \overset{choose_2}{\to}_D \gamma_2$, since we have an injection $h\colon \{1,2\} \to \{1,2,3\}$ satisfying the conditions of the definition. More

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

$$
\begin{array}{ll}
\textit{initiate}: & \langle\langle udf, udf\rangle, \langle A, x \geqslant 0 \to reset, B\rangle\rangle \\
\textit{choose}_1: & \langle\langle udf, df\rangle, \langle B, x < 1 \to reset, C^\dagger\rangle\rangle \\
\textit{choose}_2: & \langle\langle df, df\rangle, \langle B, x < 1 \to reset, C^\dagger\rangle, \langle q^\dagger, skip, q\rangle\rangle \\
\textit{choose}_3: & \langle\langle df, df\rangle, \langle B^\dagger, x < 1 \to reset, C^\dagger\rangle\rangle \\
\textit{enter}: & \langle\langle df, df\rangle, \langle C^\dagger, x > 1 \to skip, CS^\dagger\rangle\rangle \\
\textit{fail}_1: & \langle\langle udf, udf\rangle, \langle C, skip, A\rangle\rangle \\
\textit{fail}_2: & \langle\langle df, df\rangle, \langle C, skip, A\rangle\rangle \\
\textit{exit}_1: & \langle\langle df, udf\rangle, \langle CS^\dagger, skip, A\rangle\rangle \\
\textit{exit}_2: & \langle\langle df, udf\rangle, \langle CS, skip, A\rangle, \langle q^\dagger, skip, q\rangle\rangle \\
\textit{exit}_3: & \langle\langle udf, udf\rangle, \langle CS, skip, A\rangle\rangle
\end{array}
$$

Fig. 2. Rules for modelling Fischer's protocol.

precisely, we have $h(1)=2$ and $h(2)=1$. Below we give an example of a sequence of transitions.

$$
\begin{array}{l}
\langle\{1,2,3\}, udf, \langle A,A,A\rangle, \langle 0,0,0\rangle\rangle \to_T \\
\langle\{1,2,3\}, udf, \langle A,A,A\rangle, \langle 0.4,0.4,0.4\rangle\rangle \overset{initiate}{\to}_D \\
\langle\{1,2,3\}, udf, \langle A,B,A\rangle, \langle 0.4,0,0.4\rangle\rangle \to_T \\
\langle\{1,2,3\}, udf, \langle A,B,A\rangle, \langle 0.6,0.2,0.6\rangle\rangle \overset{initiate}{\to}_D \\
\langle\{1,2,3\}, udf, \langle B,B,A\rangle, \langle 0,0.2,0.6\rangle\rangle \to_T \\
\langle\{1,2,3\}, udf, \langle B,B,A\rangle, \langle 0.1,0.3,0.7\rangle\rangle \overset{choose_1}{\to}_D \\
\langle\{1,2,3\}, df, \langle C^\dagger,B,A\rangle, \langle 0,0.3,0.7\rangle\rangle \to_T \\
\langle\{1,2,3\}, df, \langle C^\dagger,B,A\rangle, \langle 0.2,0.5,0.9\rangle\rangle \overset{choose_2}{\to}_D \\
\langle\{1,2,3\}, df, \langle C,C^\dagger,A\rangle, \langle 0.2,0,0.9\rangle\rangle \to_T \\
\langle\{1,2,3\}, df, \langle C,C^\dagger,A\rangle, \langle 0.5,0.3,1.2\rangle\rangle \overset{enter}{\to}_D \\
\langle\{1,2,3\}, df, \langle C,CS^\dagger,A\rangle, \langle 0.5,0.3,1.2\rangle\rangle
\end{array}
$$

Observe that the index set $\{1,2,3\}$ does not change during the transitions. As mentioned in the remark in the end of Section 2, we can extend our model to allow creation and destruction of processes. This would mean that the index set might be changed by discrete transitions.

4. 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 $c_{\text{init}} \in C$

and $q_{\text{init}} \in Q$ 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 $\langle I, c_{\text{init}}, \mathsf{q}_{\text{init}}, \mathsf{x}_{\text{init}}\rangle$ where $\mathsf{q}_{\text{init}}(j) = q_{\text{init}}$ and $\mathsf{x}_{\text{init}}(j) = 0$ 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 $\gamma_{\text{init}} \xrightarrow{*} \gamma$, 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 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 5, 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 $pre(\phi)$ denote the set $\{\gamma : \exists \gamma' \in [\![\phi]\!] : \gamma \to \gamma'\}$, and $pre(\Phi)$ denote the set $\{\gamma : \exists \gamma' \in [\![\Phi]\!] : \gamma \to \gamma'\}$. Note that $pre(\Phi)$ is equivalent to $\bigcup_{\phi \in \Phi} pre(\phi)$. 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 7.4), i.e., that there is a k such that $[\![\Phi_k]\!] = [\![\Phi_{k+1}]\!]$, implying that $[\![\Phi_k]\!] = [\![\Phi_j]\!]$ for all $j \geqslant k$. It follows that Φ is reachable if and only if there is an initial configuration γ_{init} such that $\gamma_{\text{init}} \in [\![\Phi_k]\!]$, which is easily checked since Φ_k is a *finite* set of constraints.

To prove convergence, we introduce, in Definition 5.3, a quasi-order $\preccurlyeq$ on constraints by defining $\phi \preccurlyeq \phi'$ to denote that $[\![\phi']\!] \subseteq [\![\phi]\!]$. In Theorem 7.4, we will show that $\preccurlyeq$ 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 $\phi_i \preccurlyeq \phi_j$. This will imply 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 $\phi_i \preccurlyeq \phi_j$.

Summarizing, we will have established the following theorem

Theorem 4.1. *The reachability problem for families of timed networks is decidable*

Proof. Follows from the preceding discussion, using Theorem 5.4 (decidability of $\preccurlyeq$), Theorem 6.14 (computability of *pre*) and Theorem 7.4 (well quasi-orderedness of $\preccurlyeq$). □

The following sections contain the above mentioned definitions and lemmas. In Section 5, we define the constraint system. In Section 6, we show that $pre(\phi)$ can be computed and represented by a finite set of constraints whenever ϕ is a constraint. Finally, in Section 7, we show that the relation $\preccurlyeq$ is a well quasi-ordering on the set of constraints.

5. 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 [7]. We use a representation of constraints, which is similar to a representation of regions used by Godskesen [23].

A *quasi-order* is a reflexive and transitive relation. For a quasi-order $\sqsubseteq$, 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 $x \in \mathscr{R}^{\geq 0}$, let $\lfloor x \rfloor$ denote its integer part, and let $fract(x)$ denote its fractional part.

Definition 5.1. 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 $\hat{m}$ intuitively denotes the set of indices of processes constrained by ϕ,
- $\mathbf{q} : \hat{m} \mapsto Q$ is a mapping from indices to process states,
- $\mathbf{k} : \hat{m} \mapsto \{0, \ldots, max\}$ maps each index to a natural number not greater than *max*,
- $\sqsubseteq$ is a quasi-order on the set $\hat{m} \cup \{\bot, \top\}$ 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 $j \in \hat{m}$, and
  - $\mathbf{k}(j) = \max$ whenever $j \equiv \top$, for $j \in \hat{m}$.

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 $j \in \hat{m}$ represents a process which has control state $\mathbf{q}(j)$, for which

[4] Note that $\bot, \top \notin \hat{m}$.

$\mathbf{k}(j)$ is either *max* or the integer part of its clock, whichever is least, for which $j \equiv \bot$ 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 5.2. Let $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$ be a constraint and let $\gamma = \langle I, c, \mathtt{q}, \mathtt{x} \rangle$ be a configuration.[5] We define $\gamma \in [\![\phi]\!]$ to mean that there is an injection $h : \hat{m} \mapsto I$ from the indices of ϕ to the indices of γ such that for all $j, j_1, j_2 \in \hat{m}$

- $\mathtt{q}(h(j)) = \mathbf{q}(j)$,
- $\min(max, \lfloor \mathtt{x}(h(j)) \rfloor) = \mathbf{k}(j)$,
- $j \equiv \bot$ if and only if $\mathtt{x}(h(j)) \leqslant max$ and $fract(\mathtt{x}(h(j))) = 0$,
- $j \equiv \top$ if and only if $\mathtt{x}(h(j)) > max$, and
- if $j_1, j_2 \not\equiv \top$ then $fract(\mathtt{x}(h(j_1))) \leqslant fract(\mathtt{x}(h(j_2)))$ 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 5.3. Define the ordering $\preccurlyeq$ on constraints by $\phi \preccurlyeq \phi' \stackrel{\text{def}}{=} [\![\phi']\!] \subseteq [\![\phi]\!]$.

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

Theorem 5.4. *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 $\phi \preccurlyeq \phi'$ if and only if there is an injection $g : \hat{m} \mapsto \widehat{m'}$ such that*

- $c = c'$,
- *for all $j \in \hat{m}$ we have*[6]
  - $\mathbf{q}'(g(j)) = \mathbf{q}(j)$,
  - $\mathbf{k}'(g(j)) = \mathbf{k}(j)$,
  - $(g(j)) \equiv' \bot$ *iff* $j \equiv \bot$,
  - $(g(j)) \equiv' \top$ *iff* $j \equiv \top$,
- *if $j_1, j_2 \in \hat{m}$ then $g(j_1) \sqsubseteq' g(j_2)$ if and only if $j_1 \sqsubseteq j_2$.*

Proof (**If**). Assume that the conditions of the theorem hold and that $\langle I, c, \mathtt{q}, \mathtt{x} \rangle \in [\![\phi']\!]$. It follows that there is a mapping $h' : \widehat{m'} \mapsto I$ which satisfies the conditions in Definition 5.2. Define $g : \hat{m} \mapsto I$ by $g \stackrel{\text{def}}{=} h' \circ h$. It is easy to check that g satisfies the conditions of Definition 5.2, implying that $\langle I, c, \mathtt{q}, \mathtt{x} \rangle \in [\![\phi]\!]$.

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

[6] In a similar manner to $\equiv$, we use $\equiv'$ to denote the equivalence relation induced by $\sqsubseteq'$.

(*Only if*). Assume that $\phi \preccurlyeq \phi'$. It is trivial to see that $[\![\phi]\!] \neq \emptyset$ for any constraint ϕ. Let $\gamma \in [\![\phi']\!]$, implying that $\gamma \in [\![\phi]\!]$ and that there is a mapping $h' : \widehat{m'} \mapsto I$ which satisfies the conditions in Definition 5.2. Since $\gamma \in [\![\phi]\!]$, there is also a mapping $h : \hat{m} \mapsto I$ which satisfies the conditions in Definition 5.2. First observe that $\phi \preccurlyeq \phi'$ trivially implies $m \leqslant m'$. Define the mapping $g : \hat{m} \mapsto \widehat{m'}$ by $g \stackrel{\text{def}}{=} (h')^{-1} \circ h$. It is easy to see that g is an injection which satisfies the conditions in the theorem. □

Example. In Fischer's protocol (Section 3), we observe that the value of max is equal to 1. An example of a constraint (Definition 5.1) is given by $\phi_1 = \langle df, 5, \mathbf{q}_1, \mathbf{k}_1, \sqsubseteq_1 \rangle$, where $\mathbf{q}_1$ and $\mathbf{k}_1$ are defined by the table [7]

	1	2	3	4	5
$\mathbf{q}_1$	C	A	CS	$B^\dagger$	A
$\mathbf{k}_1$	0	1	0	1	0

and $\bot \equiv_1 4 \equiv_1 3 \sqsubset_1 5 \sqsubset_1 1 \sqsubset_1 2 \equiv_1 \top$.

A configuration satisfying ϕ_1 is given by $\gamma = \langle \{1,2,3,4,5,6,7\}, df, \mathtt{q}, \mathtt{x} \rangle$, where $\mathtt{q}$ and $\mathtt{x}$ are defined by the table

	1	2	3	4	5	6	7
q	$C^\dagger$	CS	A	B	C	$B^\dagger$	A
x	0.4	0	0.3	2.6	0.8	1.0	3.4

The reason why $\gamma \in [\![\phi_1]\!]$ follows from the fact that there is an injection $h : \{1,2,3,4,5\} \to \{1,2,3,4,5,6,7\}$ which satisfies the conditions of Definition 5.2. The value of h is given by $h(1) = 5$, $h(2) = 7$, $h(3) = 2$, $h(4) = 6$, and $h(5) = 3$.

Another example of a constraint is given by $\phi_2 = \langle df, 7, \mathbf{q}_2, \mathbf{k}_2, \sqsubseteq_2 \rangle$, where

	1	2	3	4	5	6	7
$\mathbf{q}_2$	A	$C^\dagger$	$B^\dagger$	C	A	CS	CS
$\mathbf{k}_2$	1	0	1	0	0	0	0

and $\bot \equiv 3 \equiv_2 7 \sqsubset_2 2 \sqsubset_2 5 \sqsubset_2 4 \equiv_2 6 \sqsubset_2 1 \equiv \top$. According to Theorem 5.4, we have $\phi_1 \preccurlyeq \phi_2$, since we can define $g(1) = 4$, $g(2) = 1$, $g(3) = 7$, $g(4) = 3$, and $g(5) = 5$.

6. Computing *pre*

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

[7] The entry corresponding to $\mathbf{q}_1$ and j defines the value of $\mathbf{q}_1(j)$. For instance, $\mathbf{q}_1(3) = CS$. The value of $\mathbf{k}_1$ is defined in a similar manner.

relation, we will compute $pre(\phi')$ as $pre_D(\phi') \cup pre_T(\phi')$, where $pre_D(\phi')$ denotes the set $\{\gamma : \exists \gamma' \in [\![\phi']\!] : \gamma \rightarrow_D \gamma'\}$, and where $pre_T(\phi')$ denotes the set $\{\gamma : \exists \gamma' \in [\![\phi']\!] : \gamma \rightarrow_T \gamma'\}$.

6.1. Computing pre_D

We start with a preliminary definition. Recall that a guard is a boolean combination of predicates of the form $k < x$, $k \leqslant x$, $k > x$, or $k \geqslant x$. We observe that all negations can be eliminated by pushing them inwards in the standard manner. Therefore we can assume without loss of generality that the only Boolean operators which occur in the guards are those of conjunction and disjunction.

Definition 6.1. Let $p(x)$ be a guard and let $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$ be a constraint. For $j \in \hat{m}$, we define the relation $\langle \phi, j \rangle \models p(x)$, meaning that *$p$ is satisfied at index j in ϕ*, as follows:

- If $p(x)$ is of form $k \leqslant x$ for some $k \in \{0, \ldots, max\}$ then $\langle \phi, j \rangle \models p(x)$ iff $\mathbf{k}(j) \geqslant k$.
- If $p(x)$ is of form $k < x$ for some $k \in \{0, \ldots, max\}$ then $\langle \phi, j \rangle \models p(x)$ iff either $\mathbf{k}(j) > k$ or both $\mathbf{k}(j) = k$ and $\bot \sqsubset j$.
- If $p(x)$ is of form $k \geqslant x$ for some $k \in \{0, \ldots, max\}$ then $\langle \phi, j \rangle \models p(x)$ iff either $\mathbf{k}(j) < k$ or both $\mathbf{k}(j) = k$ and $j \equiv \bot$.
- If $p(x)$ is of form $k > x$ for some $k \in \{0, \ldots, max\}$ then $\langle \phi, j \rangle \models p(x)$ iff $\mathbf{k}(j) < k$.
- If $p(x)$ is of form $p_1(x) \wedge p_2(x)$ then $\langle \phi, j \rangle \models p(x)$ iff $\langle \phi, j \rangle \models p_1(x)$ and $\langle \phi, j \rangle \models p_2(x)$.
- Disjunction is treated analogously as conjunction.

Lemma 6.2. *Let $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$ be a constraint and let $\gamma = \langle I, c, \mathrm{q}, \mathrm{x} \rangle$ be a configuration, such that $\gamma \in [\![\phi]\!]$. Let $h : \hat{m} \mapsto I$ be an injection which satisfies the* 5 *conditions in Definition* 5.2, *i.e., h is an injection which shows why $\gamma \in [\![\phi]\!]$. Then*

$$\langle \phi, j \rangle \models p(x) \quad \textit{iff} \quad p(\mathrm{x}(h(j))) \textit{ holds.}$$

Proof. The proof is structured according to the structure of $p(x)$.

- If $p(x)$ is of form $k \leqslant x$ for some $k \in \{0, \ldots, max\}$ then $\langle \phi, j \rangle \models p(x)$ is equivalent to $\mathbf{k}(j) \geqslant k$, which by Definition 5.2 is equivalent to $\min(max, \lfloor \mathrm{x}(h(j)) \rfloor) \geqslant k$, which, since $k \leqslant max$, is equivalent to $\mathrm{x}(h(j)) \geqslant k$, which is the same as $p(\mathrm{x}(h(j)))$.
- The other cases are analogous. □

We will compute $pre_D(\phi')$ as $\bigcup_{r \in R} pre_D(r, \phi')$, where $pre_D(r, \phi')$, for each rule r, 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.

Let $r = \langle \langle c, c' \rangle, \langle q_1, p_1(x), \rightarrow op_1, q_1' \rangle, \ldots, \langle q_n, p_n(x) \rightarrow op_n, q_n' \rangle \rangle$ and let $\phi' = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq' \rangle$. We will characterize $pre_D(r, \phi')$ by a finite set Φ of constraints such that $\bigcup_{\phi \in \Phi} [\![\phi]\!] = pre_D(r, \phi')$. To obtain a constraint ϕ in Φ, we first choose some partitioning of the indices $\widehat{m'}$ of ϕ' into a set *changing* of indices of processes whose states are changed by the rule r and a set *unchanged* of indices that are not affected by r. These

sets will be contained in the set $\hat{m}$ of indices of ϕ. In addition to the indices of ϕ', the indices $\hat{m}$ of ϕ must also contain a set *guarding*, corresponding to processes which are not at all mentioned by ϕ but which are affected by r. The processes in *guarding* may constrain the application of r by the requirement that they satisfy guards of r. The relationship between the indices of ϕ and the indices of the rule r is represented by a bijection $g : (changing \cup guarding) \mapsto \hat{n}$. Note that g need not relate the indices in *unchanged* since these are not concerned by r. The constraint ϕ must then satisfy the conditions in Definitions 5.2 and 2.3. First, we define a relation $\mathtt{pre}_{\mathtt{D}}$ which we later use (Lemma 6.5) to compute pre_D.

Definition 6.3. For a constraint $\phi' = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq' \rangle$ and a rule r, we define $\mathtt{pre}_{\mathtt{D}}(r, \phi')$ to be the set of constraints of form $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$, for which there are pairwise disjoint sets *changing*, *unchanged* and *guarding* such that

- $\widehat{m'} = changing \cup unchanged$, and
- $\hat{m} = changing \cup unchanged \cup guarding$,

together with a bijection $g : (changing \cup guarding) \mapsto \hat{n}$, which satisfies the following conditions.

1. $\mathbf{q}(j) = q_{g(j)}$ and $\langle \phi, j \rangle \models p_{g(j)}(x)$ for each $j \in (changing \cup guarding)$,
2. $\mathbf{q}'(j) = q'_{g(j)}$ for $j \in changing$,
3. $\mathbf{q}'(j) = \mathbf{q}(j)$ for $j \in unchanged$,
4. $\mathbf{k}'(j) = 0$ and $j \equiv' \bot$ if $j \in changing$ and $op_{g(j)} = reset$,
5. For all j such that either $j \in changing$ and $op_{g(j)} = skip$, or $j \in unchanged$, 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. For each j_1 and j_2 such that for $i = 1, 2$ either $j_i \in changing$ and $op_{g(j_i)} = skip$, or $j_i \in unchanged$, we have $j_1 \sqsubseteq' j_2$ if and only if $j_1 \sqsubseteq j_2$.

Observe that the sets *changing*, *unchanged* and *guarding* are finite and effectively constructible. 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. Note the close correspondence between the conditions of Definition 6.3 and the conditions of transitions in Definition 2.3. 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 states 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 *changing* are constrained by 2; and processes with indices in *unchanged* 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 affected by r (and hence unaffected by the transition), the clock values are unchanged by a transition.

The relation between pre_D and $\mathtt{pre}_{\mathtt{D}}$ is captured by Lemma 6.5. First, we state an auxiliary lemma, which will be used in the proof of Lemma 6.5.

Lemma 6.4. *Let $\gamma = \langle I, c, \mathtt{q}, \mathtt{x}\rangle$ be a configuration, let $m \in \mathcal{N}$ with $m \leqslant |I|$ be a natural number which is less than or equal to the size of I, and let $h : \hat{m} \mapsto I$ be an injection. Then there is a unique constraint $\phi_{\gamma,h}$ which satisfies the 5 conditions in Definition 5.2.*

Proof. It can be seen that the conditions in Definition 5.2 uniquely define the components of $\phi_{\gamma,h}$ under the assumptions in the lemma. □

Lemma 6.5. *If ϕ' is a constraint and r is a rule, then $pre_D(r,\phi')$ is the denotation of the set of constraints in the set $\mathtt{pre}_{\mathtt{D}}(r,\phi')$. In other words*

$$pre_D(r,\phi') = [\![\mathtt{pre}_{\mathtt{D}}(r,\phi')]\!].$$

Proof. First, we show that $pre(r,\phi') \subseteq [\![\mathtt{pre}_{\mathtt{D}}(r,\phi')]\!]$. Assume that $\gamma \in pre(r,\phi')$, i.e., that there is a $\gamma' \in [\![\phi']\!]$ with $\gamma \xrightarrow{r}_D \gamma'$. We must prove that $\gamma \in [\![\phi]\!]$ for some constraint $\phi \in \mathtt{pre}_{\mathtt{D}}(r,\phi')$. Let $\gamma = \langle I, c, \mathtt{q}, \mathtt{x}\rangle$ and $\gamma' = \langle I, c', \mathtt{q}', \mathtt{x}'\rangle$, where I is the set of indices in γ and γ'. By the definition of $\xrightarrow{r}_D$, there is an injection $f : \hat{n} \mapsto I$ satisfying the conditions of Definition 2.3. Since $\gamma' \in [\![\phi']\!]$, there is an injection $h' : \widehat{m'} \mapsto I$ satisfying the conditions in Definition 5.2.

Define the following sets of indices:

- *changing* is the set of indices $j \in \widehat{m'}$ such that $h'(j) \in range(f)$. Let $m = m' + (n - |range(h)|)$;
- $unchanged = \widehat{m'} \backslash changing$;
- *guarding* is defined by letting $m = m' + (n - |f^{-1} \circ h'(changing)|)$, i.e., m is obtained from m' by adding the number of indices in r which do not have a corresponding index in ϕ', and letting $guarding = \hat{m} \backslash \widehat{m'}$.

Define the bijection $g : (changing \cup guarding) \mapsto \hat{n}$ as any bijection whose restriction to *changing* is equal to $f^{-1} \circ h'$. Extend h' to an injection $h : \hat{m} \mapsto I$ by letting $h = h'$ on $\widehat{m'}$ and $h = f \circ g$ on $domain(g)$. Note that h is well-defined since $f \circ g = f \circ ((f)^{-1} \circ h') = h'$ on $domain(g) \cap \widehat{m'}$. We will take the sought constraint ϕ to be $\phi_{\gamma,h}$, which by Lemma 6.4 satisfies the conditions in Definition 5.2, with h as the injection.

We must check that $\phi_{\gamma,h}$ and g satisfies the conditions of Definition 6.3. Let us treat the conditions one by one.

1. Let $j \in (changing \cup guarding)$. We have that $\mathbf{q}(j)$ equals $\mathtt{q}(h(j))$ by the definition of $\phi_{\gamma,h}$, which equals $\mathtt{q}(f(g(j)))$ by the definition of h, which by condition 1 in Definition 2.3 equals $q_{g(j)}$ (noting that $g(j) \in \hat{n}$). We have by condition 1 of Definition 2.3 that $P_{g(j)}(\mathtt{x}(f(g(j))))$ holds, which by Lemma 6.2 implies $\langle \phi, j\rangle \models P_{g(j)}(x)$.
2. Let $j \in changing$. We have that $\mathbf{q}'(j)$ equals $\mathtt{q}'(h'(j))$ since $\gamma' \in [\![\phi']\!]$, which equals $q'(f(g(j)))$ by the definition of h', which by condition 2 in Definition 2.3 equals $q'_{g(j)}$.
3. Let $j \in unchanged$, i.e., $h'(j) \notin range(f)$. By condition 2 in Definition 2.3 we have $\mathtt{q}'(h'(j)) = \mathtt{q}(h'(j))$, which by $\gamma' \in [\![\phi']\!]$ and Definition 5.2 implies $\mathbf{q}'(j) = \mathtt{q}(h'(j))$, which from the definition of $\phi_{\gamma,h}$ (noting that $h = h'$ on *unchanged*) implies $\mathbf{q}'(j) = \mathbf{q}(j)$.

4. Let $j \in \mathit{changing}$ and $\mathit{op}_{g(j)} = \mathit{reset}$. We have by condition 4 of Definition 2.3 that $\mathtt{x}'(f(g(j))) = 0$ holds, which since $h' = f \circ g$ implies $\mathtt{x}'(h'(j)) = 0$, which by $\gamma' \in [\![\phi']\!]$ and Definition 5.2 implies $\mathbf{k}'(j) = 0$ and $j \equiv' \bot$.
5. and 6. Let $j \in \mathit{changing}$ and $\mathit{op}_{g(j)} = \mathit{skip}$. By condition 5 of Definition 2.3 we have that $\mathtt{x}'(f(g(j))) = \mathtt{x}(f(g(j)))$, which since $h' = f \circ g$ implies $\mathtt{x}'(h'(j)) = \mathtt{x}(h'(j))$. If $j \in \mathit{unchanged}$, i.e., $h'(j) \notin \mathit{range}(f)$, then by condition 6 of Definition 2.3 we have $\mathtt{x}'(h'(j)) = \mathtt{x}(h'(j))$. In both cases, the conclusion $\mathtt{x}'(h'(j)) = \mathtt{x}(h'(j))$ implies conditions 5 and 6, using the definition of $\phi_{\gamma,h}$, using $\gamma' \in [\![\phi']\!]$, and using Definition 5.2.

Now, we show that $[\![\mathtt{pre}_{\mathrm{D}}(r, \phi')]\!] \subseteq \mathit{pre}(r, \phi')$. Assume that there is a constraint ϕ, sets *changing*, *unchanged* and *guarding*, and a bijection $g : (\mathit{changing} \cup \mathit{guarding}) \mapsto \hat{n}$, which satisfy the conditions of Definition 6.3 with $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$. We must prove that $[\![\phi]\!] \subseteq \mathit{pre}(r, \phi')$. Assume that $\gamma \in [\![\phi]\!]$. By $\gamma \in [\![\phi]\!]$ there is an injection $h : \hat{m} \mapsto I$ which satisfies the conditions in Definition 5.2. Define the injection $f : \hat{n} \mapsto I$ by $f \stackrel{\mathrm{def}}{=} h \circ g^{-1}$. For $j \in (\mathit{changing} \cup \mathit{guarding})$ we have, by condition 1 of the lemma, $\mathbf{q}(j) = q_{g(j)}$ and $\langle \phi, j \rangle \models p_{g(j)}(x)$. By $\gamma \in [\![\phi]\!]$ and Lemma 6.2 we have $\mathbf{q}(j) = \mathtt{q}(h(j))$ and $p_{g(j)}(\mathtt{x}(h(j)))$, implying $\mathtt{q}(f(i)) = q_i$ and $p_i(\mathtt{x}(h(j)))$. Thus, the first condition in Definition 2.3 is satisfied, implying that there is a unique γ' satisfying the conditions for $\gamma \xrightarrow{r}_D \gamma'$.

Let h' be the restriction of h to $\widehat{m'}$. We will now check that $\gamma' \in [\![\phi']\!]$ by proving that the conditions in Definition 5.2 are satisfied, using h' as the injection. The proof is structured into cases, depending on the index j.

- If $j \in \mathit{changing}$ then by condition 2 we have $\mathbf{q}'(j) = q'_{g(j)}$, and by condition 2 in Definition 2.3 we have $\mathtt{q}'(f(g(j))) = q'_{g(j)}$. We conclude that $\mathtt{q}'(h'(j)) = \mathtt{q}'(f(g(j))) = q'_{g(j)} = \mathbf{q}'(j)$.
  If $j \in \mathit{unchanged}$, then by condition 3 we have $\mathbf{q}'(j) = \mathbf{q}(j)$. By $\gamma \in [\![\phi]\!]$ and condition 1 in Definition 5.2 we have $\mathtt{q}(h(j)) = \mathbf{q}(j)$, and by condition 3 in Definition 2.3 we have $\mathtt{q}(h(j)) = \mathtt{q}'(h(j))$ since $h(j) \notin \mathit{range}(f)$. We conclude $\mathtt{q}'(h(j)) = \mathbf{q}'(j)$.
  This concludes the proof of the first condition in Definition 5.2.
- Let $j \in \mathit{changing}$ and $\mathit{op}_{g(j)} = \mathit{reset}$. Then by condition 4 we have $\mathbf{k}'(j) = 0$ and $j \equiv' \bot$. By condition 4 of Definition 2.3 we have $\mathtt{x}'(h'(j)) = \mathtt{x}'(f(g(j))) = 0$, implying that conditions 2, 3, and 4 in Definition 5.2 are satisfied.
- Let j be such that either $j \in \mathit{changing}$ and $\mathit{op}_{g(j)} = \mathit{skip}$, or such that $j \in \mathit{unchanged}$. By conditions 5 and 6 of Definition 2.3 we have $\mathtt{x}'(h'(j)) = \mathtt{x}'(f(g(j))) = \mathtt{x}(f(g(j))) = \mathtt{x}(h(j))$. Since $\gamma \in [\![\phi]\!]$, the fact that γ satisfies conditions 2–4 of Definition 2.3 implies that γ' also does. The fact that γ satisfies conditions 5 for indices j_1 and j_2 implies that γ' also does in the case that for $i = 1, 2$ we have either $j_i \in \mathit{changing}$ and $\mathit{op}_{g'(j_i)} = \mathit{skip}$ or $j_i \in \mathit{unchanged}$. If $j_i \in \mathit{changing}$ and $\mathit{op}_{g(j_i)} = \mathit{reset}$ this follows by noting that $\mathtt{x}'(h'(j)) = 0$ was shown in the previous case. □

6.2. Computing pre_T

First, we define a relation $\mathtt{pre}_{\mathrm{T}}$ which we later use (Lemma 6.13) to compute pre_T.

Definition 6.6. For a constraint $\phi' = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq' \rangle$ we define $\mathtt{pre}_{\mathrm{T}}(\phi')$ to be 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 $\mathbf{k}(j) = 0$. Furthermore, the following three conditions should 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 $j \equiv' \bot$. Furthermore, the following four conditions should 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. If no such clock has an integer part which is 0, time can move backwards by making these clocks (corresponding to the indices j with $j \equiv' \bot$) decrease their integer parts by one, and by making their fractional parts become the largest (wrp. to $\sqsubset$) among all clocks which are less than *max*. The second case captures the situation when no clocks have fractional parts equal to 0. In this situation, the smallest step backwards in time corresponds to preserving the relative order between the fractional parts of clocks which are less than *max*, and by doing either or both of

- making the fractional parts of all clocks that are minimal wrp. to $\sqsubset'$ become 0,
- making the fractional parts of some clocks with values larger than *max* become 0.

Lemma 6.7. *There is no infinite sequence $\phi_0, \phi_1, \phi_2, \ldots$ of constraints, such that $\phi_{i+1} \in \mathtt{pre}_{\mathrm{T}}(\phi_i)$.*

Proof. For a constraint $\phi = \langle c, m, \mathbf{q}, \mathbf{k}, \sqsubseteq \rangle$, we say that ϕ is *of type* 0 if there is a $j \in \hat{m}$ such that $j \equiv \bot$, otherwise we say that ϕ is *of type* 1.

Suppose that we have an infinite sequence $\phi_0, \phi_1, \phi_2, \ldots$ of constraints, such that $\phi_{i+1} \in \mathtt{pre}_{\mathrm{T}}(\phi_i)$, for $i \geqslant 0$.

Let $\phi_i = \langle c_i, m_i, \mathbf{q}_i, \mathbf{k}_i, \sqsubseteq_i \rangle$, and let $K_i = \sum_{j \in \widehat{m_i}} \mathbf{k}_i(j)$. From Definition 6.6, it follows that for each $i \geqslant 0$, one the following holds.

- ϕ_i is of type 0, ϕ_{i+1} is of type 1, and $K_i > K_{i+1}$.
- ϕ_i is of type 1, ϕ_{i+1} is of type 0, and $K_i = K_{i+1}$.

This implies that $K_i > K_{i+2}$ for each $i \geqslant 0$, which is a contradiction, since each K_i is a natural number. □

Definition 6.8. For a set of constraints Φ' and a natural number i, we define $\mathtt{pre}_{\mathrm{T}}^{i}(\Phi', i)$ inductively as follows.

- $\mathtt{pre}_{\mathrm{T}}^{0}(\Phi') = \Phi'$; and
- $\mathtt{pre}_{\mathrm{T}}^{i+1}(\Phi') = \{\phi : \exists \phi' \in \mathtt{pre}_{\mathrm{T}}^{i}(\Phi') : \phi \in \mathtt{pre}_{\mathrm{T}}(\phi')\}$.

We define $\mathtt{pre}_{\mathrm{T}}^{*}(\Phi') = \bigcup_{i \geqslant 0} \mathtt{pre}_{\mathrm{T}}^{i}(\Phi')$.

Sometimes we write $\mathtt{pre}_{\mathrm{T}}^{*}(\phi')$ instead of $\mathtt{pre}_{\mathrm{T}}^{*}(\{\phi'\})$.

In Lemma 6.13 we describe the relation between pre_T and $\mathtt{pre}_{\mathrm{T}}$. First, we give some auxiliary definitions and lemmas.

Let $\gamma = \langle I, c, \mathtt{q}, \mathtt{x} \rangle$ be a configuration. By $\mathtt{x}_{\min}(\gamma)$ we mean $\min\{\mathtt{x}(j) : j \in I\}$. For $\delta \in \mathscr{R}^{\geqslant 0}$, with $\delta \leqslant \mathtt{x}_{\min}(\gamma)$, we use $\gamma^{-\delta}$ to denote the configuration $\langle I, c, \mathtt{q}, \mathtt{x}' \rangle$, where $x'(j) = x(j) - \delta$ for each $j \in I$.

Lemma 6.9. *For a configuration γ' and a constraint ϕ', with $\gamma' \in [\![\phi']\!]$ and $\mathtt{x}_{\min}(\gamma') > 0$, there is a $\delta' \in \mathscr{R}^{\geqslant 0}$ and a constraint ϕ such that $\phi \in \mathtt{pre}_{\mathrm{T}}(\phi')$, and one of the following three conditions holds*:

1. *$\gamma^{-\delta} \in [\![\phi']\!]$, for all $0 \leqslant \delta \leqslant \mathtt{x}_{\min}(\gamma)$;*
2. *$\gamma^{-\delta} \in [\![\phi']\!]$, for all $\delta : 0 < \delta \leqslant \delta'$; or*
3. *$\gamma^{-\delta} \in [\![\phi']\!]$, for all $\delta : 0 \leqslant \delta < \delta'$, and $\gamma^{-\delta'} \in [\![\phi]\!]$.*

Proof. Let $\gamma' = \langle I', c', \mathtt{q}', \mathtt{x}' \rangle$, and $\phi = \langle c', m', \mathbf{q}', \mathbf{k}', \sqsubseteq' \rangle$. Let $h : \hat{m} \mapsto I$ be an injection satisfying the conditions stated in Definition 5.2. There are three cases:

- If there is a $j \in \hat{m}$ where $fract(x(h(j))) = 0$. Define $\delta_1 = \min\{fract(x(h(j))) : j \in \hat{m}$ and $fract(x(h(j))) > 0\}$. We take δ' such that $0 < \delta' < \delta_1$ if δ_1 is well-defined, and $0 < \delta' < 1$ otherwise.[8] We define ϕ to be the (unique) constraint $\langle c', m', \mathbf{q}', \mathbf{k}, \sqsubseteq \rangle$ satisfying condition 1 in Definition 6.6. It follows that condition 2 in the lemma is satisfied.
- If there is one $j \in \hat{m}$ where $fract(x(h(j))) = 0$, and there is at least no $j \in \hat{m}$ such that $x(h(j)) < \max + 1$. We take $\delta' = \min\{fract(x(h(j))) : j \in \hat{m}$ and $x(h(j)) < \max + 1\}$. If $\delta' > \mathtt{x}_{\min}(\gamma')$ then condition 1 is satisfied, otherwise we define γ to be the (unique) constraint $\langle I', c', \mathtt{q}', \mathtt{x} \rangle$ where
  - whenever $\bot \sqsubset' j_1, j_2 \sqsubset' \top$, we have $j_1 \sqsubseteq j_2$ if and only if $j_1 \sqsubseteq' j_2$.
  - $j \equiv \bot$ if and only if $fract(x(h(j))) = \delta'$ and $x(h(j)) < \max + 1$,
  - $j \equiv \top$ if and only if $x(h(j)) > \max + \delta'$.

  From Definition 6.6 it follows that condition 3 of the lemma is satisfied.
- If $x(h(j)) \geqslant \max + 1$ for each $j \in \hat{m}$. Define $\delta' = \min\{x(h(j)) - \max : j \in \hat{m}\}$. If $\delta' > \mathtt{x}_{\min}(\gamma)$ then condition 1 is satisfied, otherwise we define γ to be the (unique) constraint $\langle I', c', \mathtt{q}', \mathtt{x} \rangle$ where
  - $j \equiv \bot$ if and only if $x(h(j)) = \max + \delta'$,
  - $j \equiv \top$ if and only if $x(h(j)) > \max + \delta'$.

  From Definition 6.6 it follows that condition 3 of the lemma is satisfied. □

[8] δ' will be undefined if there is no $j \in \hat{m}$ such that $fract(x(h(j))) > 0$. We can also take $\delta_1 = \min\{fract(x(h(j))) : j \in \hat{m}$ and $fract(x(h(j))) > 0$ and $x(h(j)) < \max + 1\}$.

Corollary 6.10. *For configurations γ and γ', and a constraint ϕ', if $\gamma' \in [\![\phi']\!]$ and $\gamma \rightarrow_T \gamma'$, then there exists a constraint ϕ such that $\gamma \in [\![\phi]\!]$ and $\phi \in \mathtt{pre}_T^*(\phi')$.*

Lemma 6.11. *For a configuration γ, and constraints ϕ and ϕ', if $\phi \in \mathtt{pre}_T(\phi')$ and $\gamma \in [\![\phi]\!]$, then there is a configuration γ' such that $\gamma' \in [\![\phi']\!]$ and $\gamma \rightarrow_T \gamma'$.*

Proof. Let $\gamma = \langle I, c, \mathtt{q}, \mathtt{x}\rangle$. We define $\gamma' = \gamma^{+\delta}$, where δ is defined according to one of the following two cases.

- If ϕ and ϕ' satisfy the conditions of case 1 in Definition 6.6, then we define $\delta < 1 - \max\{fract(x(j)) : j \in I \text{ and } x(j) \leqslant \max\}$.
- If ϕ and ϕ' satisfy the conditions of case 2 in Definition 6.6. We define $\delta_1 = \max\{fract(x(j)) : j \in I \text{ and } x(j) \leqslant \max\}$. We take $\delta = 1 - \delta_1$ if δ_1 is well-defined, and take δ arbitrarily otherwise.[9] □

Corollary 6.12. *For a configuration γ, and constraints ϕ and ϕ', if $\phi \in \mathtt{pre}_T^*(\phi')$ and $\gamma \in [\![\phi]\!]$, then there is a configuration γ' such that $\gamma' \in [\![\phi']\!]$ and $\gamma \rightarrow_T \gamma'$.*

Now we are ready to give the relation between pre_T and $\mathtt{pre}_T$.

Lemma 6.13. *If ϕ' is a constraint, then $pre_T(\phi')$ is the denotation of the set of constraints in the set $\mathtt{pre}_T^*(\phi')$. In other words*

$$pre_T(\phi') = [\![\mathtt{pre}_T^*(\phi')]\!].$$

Proof. The proof follows from Corollaries 6.10 and 6.12. □

The computability of pre_T follows from Lemmas 6.13 and 6.7.

6.3. Computing pre

By combining the rules for computing $pre_D(\phi)$ in Lemma 6.5 and the rules for computing $pre_T(\phi)$ in Lemma 6.13, we obtain the following theorem.

Theorem 6.14. *If ϕ is a constraint, then we can compute a finite set Φ of constraints such that $[\![\Phi]\!] = pre(\phi)$.*

7. The entailment ordering is a well quasi-ordering

In this section, we shall prove that the preorder $\preccurlyeq$ on constraints is a well quasi-ordering. We will first review some standard results from the literature concerning well quasi-orderings ([25, 34]), and then apply them to our constraint system.

[9] δ' will be undefined if there is no $j \in I$ with $x(j) \leqslant \max$.

Definition 7.1. Let A be a set. A quasi-order $\preccurlyeq$ on A is a binary relation over A which is reflexive and transitive. A quasi-order $\preccurlyeq$ 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 $a_i \preccurlyeq a_j$.

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 $w:\widehat{|w|}\mapsto A$ where $|w|$ is the size of the bag or the length of the sequence. Given a quasi-order $\preccurlyeq$ on a set A, define the quasi-order $\preccurlyeq^*$ on A^* by letting $w\preccurlyeq^* w'$ if and only if there is a monotone [10] injection $h:\widehat{|w|}\mapsto\widehat{|w'|}$ such that $w(j)\preccurlyeq w'(h(j))$ for $1\leqslant j\leqslant|w|$. Define the quasi-order $\preccurlyeq^B$ on bags of A by $w\preccurlyeq^B w'$ if and only if there is a (not necessarily monotonic) injection $h:|\hat{w}|\mapsto\widehat{|w|'}$ such that $w(j)\preccurlyeq w'(h(j))$ for $1\leqslant j\leqslant|w|$.

Lemma 7.2. *If $\preccurlyeq$ is a wqo on A, then $\preccurlyeq^*$ is a wqo on A^* and $\preccurlyeq^B$ is a wqo on A^B.*

Proof. The proof can be found in [25]. □

Let ϕ be a constraint $\langle c,m,\mathbf{q},\mathbf{k},\sqsubseteq\rangle$. For each $j\in\hat{m}\cup\{\bot,\top\}$, define the *rank* of j in ϕ to be the number of equivalence classes induced by $\sqsubseteq$ which are strictly less than (wrp. to $\sqsubseteq$) 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 $i\in\hat{r}$, 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 $\preccurlyeq$ on these pairs to be the identity relation on pairs of the form $\langle q,k\rangle$. Since there are finitely many such pairs, $\preccurlyeq$ is trivially a well quasi-ordering.

Lemma 7.3. *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 $\phi\preccurlyeq\phi'$ if and only if $c=c'$, $\phi[0]\preccurlyeq^B\phi'[0]$, $\phi[r]\preccurlyeq^B\phi'[r']$, and there is a monotonic injection $h:\widehat{(r-1)}\mapsto\widehat{(r'-1)}$ such that $\phi[i]\preccurlyeq^B\phi'[h(i)]$ for all $i\in\widehat{(r-1)}$.*

Proof. The proof follows from the definitions. □

Theorem 7.4. *The relation $\preccurlyeq$ on the set of constraints is a well quasi-ordering.*

Proof. The proof follows from Lemma 7.3 and repeated application of Lemma 7.2. □

[10] Meaning that $h(h_1)\leqslant h(j_2)$ if and only if $j_1\leqslant j_2$.

8. Undecidability of the recurrent state problem

In this section we show undecidability of the *recurrent state problem*. This implies undecidability of model checking of timed networks, with respect to any temporal logic which is sufficiently expressive to encode the recurrent state problem. Examples of such logics include PTL, CTL, etc. The idea of the proof is a reduction from a similar problem for lossy channel systems.

Assume a timed network N. For configurations γ_1 and γ_2, we use $\gamma_1 \Rightarrow \gamma_2$ to denote that there is a configuration γ_3 such that $\gamma_1 \rightarrow_T \gamma_3 \rightarrow_D \gamma_2$. For a configuration γ, a *computation* π of N from γ is an infinite sequence of configurations of the form $\gamma_0, \gamma_1, \gamma_2, \ldots$, where $\gamma_0 = \gamma$ and $\gamma_i \Rightarrow \gamma_{i+1}$ for each $i \geqslant 0$. For a controller state c, we say that π visits c infinitely often if there are infinitely many i such that the controller state component of γ_i is equal to c.

The *Recurrent State Problem* (*RSP-TN*) for timed networks is defined as follows.
Problem. RSP-TN

Instance: A timed network N, a configuration γ, and a controller state c in N.

Question: Is there a computation π of N from γ visiting c infinitely often?

Now, we consider lossy channel systems. A *lossy channel system* (*LCS*) L [4] consists of a finite-state process operating on a channel. The process can send and receive messages belonging to a finite alphabet M through the channel. The channel behaves as a FIFO-buffer which is unbounded in size, and unreliable in the sense that, at any time, it can nondeterministically lose any of its messages. For technical reasons, we assume here that messages are only lost in the head of the channel. A *configuration* β of a lossy channel system is a pair $\langle s, w \rangle$ where s represents the state of the finite state process, called the *local state*, and $w \in M^*$ represents the content of the channel. A *computation* of L is defined in a similar manner to timed networks. The recurrent state problem (RSP-LCS) for lossy channel systems is also defined in a similar manner to RSP-TN. Here, we ask whether a *local state* (rather than a controller state) is visited infinitely often by a computation of the LCS. In [3] we show the following theorem.

Theorem 8.1. *RSP-LCS is undecidable.*

We show undecidability of RSP-TN through a reduction from RSP-LCS. We will not give the technical details of the proof. The main idea is that, given an LCS L, we derive a timed network N such that L and N have essentially the same set of computations. Thus, our derivation preserves computations in which some control state is recurrent. We encode each configuration $\beta = \langle s, w \rangle$ of L by a configuration $\gamma = \langle I, c, \mathrm{q}, \mathrm{x} \rangle$ of N. The local state s is represented by the state of the controller c. Each element in w is represented by a process (an element $k \in I$), where $0 < \mathrm{x}(k) < 1$. The ordering of the elements in w is encoded by the ordering of the clock values of the corresponding processes. More precisely,

- $c = s$;
- $\mathrm{q}(h(j)) = w(j)$ whenever $j \in \widehat{|w|}$;

- $0 < \mathrm{x}(k) < 1$ whenever $k \in range(h)$;
- If $j_1, j_2 \in \widehat{|w|}$ then $\mathrm{x}(h(j_1)) \leqslant \mathrm{x}(h(j_2))$ if and only if $j_1 \leqslant j_2$; and
- $\mathrm{x}(k) > 1$ whenever $k \notin range(h)$.

A send operation in L is simulated by picking a process whose clock value is strictly greater than one and resetting its clock value, thus putting it in the head of the channel. A loss or receive operation can be simulated by letting time pass thus making the clock value of the corresponding process strictly greater than one. It is not difficult to verify that a computation of N visits a controller state infinitely often if and only if there is a computation of L which visits the corresponding local state infinitely often.

The above reduction, together with Theorem 8.1, gives the following theorem.

Theorem 8.2. *RSP-TN is undecidable.*

Acknowledgements

We thank the anonymous referees for many helpful comments. This research was funded in part by the Swedish Research Council for Engineering Sciences (TFR).

References

[1] P.A. Abdulla, K. Čerāns, B. Jonsson, T. Yih-Kuen, General decidability theorems for infinite-state systems, in: Proc. 11th IEEE Int. Symp. on Logic in Computer Science, 1996, pp. 313–321.
[2] P.A. Abdulla, K. Čerāns, B. Jonsson, T. Yih-Kuen, Algorithmic analysis of programs with well quasi-ordered domains, Inform. and Comput. 160 (2000) 109–127.
[3] P.A. Abdulla, B. Jonsson, Undecidable verification problems for programs with unreliable channels, Inform. and Comput. 130 (1) (1996) 71–90.
[4] P.A. Abdulla, B. Jonsson, Verifying programs with unreliable channels, Inform. and Comput. 127 (2) (1996) 91–101.
[5] P.A. Abdulla, M. Kindahl, Decidability of simulation and bisimulation between lossy channel systems and finite state systems, in: I. Lee, S.A. Smolka (Eds.), Proc. CONCUR'95, 6th Internat. Conf. on Concurrency Theory, Lecture Notes in Computer Science, Vol. 962, Springer, Berlin, 1995, pp. 333–347.
[6] M. Abadi, L. Lamport, An old-fashioned recipe for real time, in: J. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg (Eds.), Real-Time: Theory in Practice, Lecture Notes in Computer Science, Vol. 600, Springer, Berlin, 1992, pp. 1–27.
[7] R. Alur, C. Courcoubetis, D. Dill, Model-checking for real-time systems, in: Proc. 5th IEEE Int. Symp. on Logic in Computer Science, Philadelphia, 1990, pp. 414–425.
[8] R. Alur, C. Courcoubetis, T. Henzinger, P.-H. Ho, Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems, in: Grossman, Nerode, Ravn, Rischel (Eds.), Hybrid Systems, Lecture Notes in Computer Science, Vol. 736, Springer, Berlin, 1992, pp. 209–229.
[9] R. Alur, T. Henzinger, A really temporal logic, in: Proc. 30th Annual Symp. Foundations of Computer Science, 1989, pp. 164–169.
[10] J.M. Barzdin, J.J. Bicevskis, A.A. Kalninsh, Automatic construction of complete sample systems for program testing, Proc. IFIP Congress, 1977.
[11] B. Berthomieu, M. Diaz, Modeling and verification of time dependent systems using time Petri nets, IEEE Trans. on Software Eng. 17 (3) (1991) 259–273.
[12] O. Burkart, B. Steffen, Composition, decomposition, and model checking of pushdown processes, Nordic J. Comput. 2 (2) (1995) 89–125.

[13] K. Čerāns, Decidability of bisimulation equivalence for parallel timer processes, in: Proc. Workshop on Computer Aided Verification, Lecture Notes in Computer Science, Vol. 663, Springer, Berlin, 1992, pp. 302–315.

[14] K. Čerāns, Feasibility of finite and infinite paths in data dependent programs, in: Proc. LFCS'92, Lecture Notes in Computer Science, Vol. 620, Springer, Berlin, 1992, pp. 69–80.

[15] K. Čerāns, Deciding properties of integral relational automata, in: S. Abiteboul, E. Shamir (Ed.), Proc. ICALP'94, Lecture Notes in Computer Science, Vol. 820, Springer, Berlin, 1994, pp. 35–46.

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

[17] E.M. Clarke, O. Grumberg, S. Jha, Verifying parameterized networks using abstraction and regular languages, in: I. Lee, S.A. Smolka (Eds.), Proc. CONCUR'95, 6th Internat. Conf. on Concurrency Theory, Lecture Notes in Computer Science, Vol. 962, Springer, Berlin, 1995, pp. 395–407.

[18] J. Esparza, Petri nets, commutative context-free grammars, and basic parallel processes, in: Proc. Fundamentals of Computation Theory, Lecture Notes in Computer Science, Vol. 965, Springer, Berlin, 1995, pp. 221–232.

[19] A. Finkel, Reduction and covering of infinite reachability trees, Inform. and Comput. 89 (1990) 144–179.

[20] S.M. German, A.P. Sistla, Reasoning about systems with many processes, J. ACM 39 (3) (1992) 675–735.

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

[22] P. Godefroid, P. Wolper, Using partial orders for the efficient verification of deadlock freedom and safety properties, Formal Methods System Design 2 (2) (1993) 149–164.

[23] J.C. Godskesen, Timed modal specifications, Ph.D. Thesis, Aalborg University, 1994.

[24] T.A. Henzinger, Hybrid automata with finite bisimulations, in: Proc. ICALP'95, 1995, pp. 324–335.

[25] G. Higman, Ordering by divisibility in abstract algebras, Proc. London Math. Soc. 2 (1952) 326–336.

[26] P. Jančar, Decidability of a temporal logic problem for Petri nets, Theoret. Comput. Sci. 74 (1990) 71–93.

[27] P. Jančar, J. Esparza, Deciding finiteness of Petri nets up to bisimulation, in: Proc. ICALP '96, Lecture Notes in Computer Science, Vol. 1099, Springer, Berlin, 1996, pp. 478–489.

[28] P. Jančar, F. Moller, Checking regular properties of Petri nets, in: Proc. CONCUR'95, 6th Internat. Conf. on Concurrency Theory, Lecture Notes in Computer Science, Vol. 962, Springer, Berlin, 1995, pp. 348–362.

[29] B. Jonsson, J. Parrow, Deciding bisimulation equivalences for a class of non-finite-state programs, Inform. and Comput. 107 (2) (1993) 272–302.

[30] M. Kindahl, Verification of infinite-state systems: decision problems and efficient algorithms, Ph.D. Thesis, Department of Computer Systems, Uppsala University, Sweden, 1999.

[31] K.J. Kristoffersen, F. Larroussinie, K.G. Larsen, P. Pettersson, W. Yi, A compositional proof of a real-time mutual exclusion protocol, in: Proc. TAPSOFT'97, 7th Internat. J. Confer. on the Theory and Practice of Software Development, Lille, France, Lecture Notes in Computer Science, Vol. 1214, Springer, Berlin, 1997, pp. 565–579.

[32] R.P. Kurshan, K. McMillan, A structural induction theorem for processes, in: Proc. 8th ACM Symp. on Principles of Distributed Computing, Canada, Edmonton, Alberta, 1989, pp. 239–247.

[33] K.G. Larsen, B. Steffen, C. Weise, Fischer's protocol revisited: a simple proof using modal constraints, in: Proc. 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, NJ, October 1995.

[34] M. Lothaire, Combinatorics on Words, Encyclopedia of Mathematics and its Applications, Vol. 17, Addison-Wesley, Reading, MA, 1983.

[35] P. Merlin, D.J. Farber, Recoverability of communication protocols — implications of a theoretical study, IEEE Trans. Comput. COM-24 (1976) 1036–1043.

[36] J.P. Queille, J. Sifakis, Specification and verification of concurrent systems in Cesar, in: M. Dezani-Ciancaglini, M. Montanari (Eds.), Proc. Int. Symp. Programming, Lecture Notes in Computer Science, Vol. 137, Springer, Berlin, 1982, pp. 337–351.

[37] F.B. Schneider, B. Bloom, K. Marzullo, Putting time into proof outlines, in: J.W. de Bakker, C. Huizing, W.P. de Roever, G. Rozenberg (Eds.), Real-Time: Theory in Practice, Lecture Notes in Computer Science, Vol. 600, Springer, Berlin, 1992, pp. 618–639.
[38] A.U. Shankar, An introduction to assertional reasoning for concurrent systems, Comput. Surveys 25 (3) (1993) 225–262.
[39] N. Shankar, Verification of real-rime systems using PVS, in: Courcoubetis (Ed.), Proc. 5th Int. Conf. on Computer Aided Verification, Lecture Notes in Computer Science, Vol. 697, Springer, Berlin, 1993, pp. 280–291.
[40] C. Stirling, Decidability of bisimulation equivalence for normed pushdown processes, in: Proc. CONCUR '96, 7th Int. Conf. on Concurrency Theory, Lecture Notes in Computer Science, Vol. 1119, Springer, Berlin, 1996, pp. 217–232.
[41] M.Y. Vardi, P. Wolper, An automata-theoretic approach to automatic program verification, in: Proc. 1st IEEE Int. Symp. on Logic in Computer Science, June 1986, pp. 332–344.
[42] P. Wolper, Expressing interesting properties of programs in propositional temporal logic (extended abstract), in: Proc. 13th ACM Symp. on Principles of Programming Languages, January 1986, pp. 184–193.
[43] P. Wolper, V. Lovinfosse, Verifying properties of large sets of processes with network invariants (extended abstract), in: J. Sifakis (Ed.), Proc. Workshop on Computer Aided Verification, Lecture Notes in Computer Science, Vol. 407, Springer, Berlin, 1989, pp. 68–80.
[44] W. Yi, CCS + Time = an interleaving model for real time systems, in: J. Leach Albert, B. Monien, M. Rodriguez Artalejo (Eds.), Proc. ICALP'91, Lecture Notes in Computer Science, Vol. 510, Springer, Berlin, 1991, pp. 217–228.