Multi-Clock Timed Networks

(Extended Abstract)

Parosh Aziz Abdulla
Uppsala University

Johann Deneux
Uppsala University

Pritha Mahata
Uppsala University

Abstract

We consider verification of safety properties for parameterized systems of timed processes, so called timed networks. A timed network consists of a finite state process, called a controller, and an arbitrary set of identical timed processes. In a previous work, we showed that checking safety properties is decidable in the case where each timed process is equipped with a single real-valued clock. It was left open whether the result could be extended to multi-clock timed networks. We show that the problem becomes undecidable when each timed process has two clocks.

On the other hand, we show that the problem is decidable when clocks range over a discrete time domain. This decidability result holds when processes have any finite number of clocks.

1 Introduction

One of the main current challenges in model checking is to extend its applicability to *parameterized systems*. The description of such a system is parameterized by the number of components, and the challenge is to check correctness of all instances in one verification step. Most existing methods for model checking of parameterized systems consider the case where each individual component is modelled as a finite-state process.

In this paper we study parameterized systems of *timed* processes, so called *Timed Networks (TNs)*. A TN represents a family of systems, each consisting of a finite-state *controller*, together with finitely, but arbitrarily many *timed processes* (timed automata). A timed process operates on a finite number of real-valued clocks. This means that a TN operates on an unbounded number of clocks, and therefore its behaviour cannot be captured by that of a timed automaton [AD94].

In [AJ03], we show decidability of the *controller state reachability problem* for TNs: given a state of the controller, is there a computation from an initial configuration leading to that state? This problem is relevant since it can be shown, using standard techniques, that checking large classes of safety properties can be reduced to controller state reachability. The decidability result in [AJ03] is given subject to the restriction that each timed process has a single clock. As an example, this allows automatic verification of a parameterized version of *Fischer's protocol* (see e.g. [KLL$^+$97]). This protocol achieves mutual exclusion by defining timing constraints on an arbitrary set of processes each with one clock. We can show correctness of the protocol regardless of the number of participating processes. The paper [AJ03] leaves open the case of *multi-clock* TNs, i.e., TNs where each timed process may have several clocks.

In the literature, there are many applications where a number of timed automata [AD94] run in parallel and where each of the timed automata has more than one clock. For instance, the Phillips audio control protocol with bus collision [BGK$^+$96] has two clocks per sender of audio signals. Also, the system described in [MT01] consists of an arbitrary number of nodes, each of which is connected to a set of LANs. Each node maintains timers to keep track of sending and receiving of messages from other nodes connected to the same set of LANs. In a similar way to Fischer's protocol, it is clearly relevant to ask whether we can verify correctness of the protocol in [BGK$^+$96] regardless of the number of senders, or the protocol in [MT01] regardless of the number of nodes.

The question is then whether the decidability result of [AJ03] can be extended to multi-clock systems. In this paper we answer this question negatively. In fact, we show that it is sufficient to allow two clocks per process in order to get undecidability. The undecidability result is shown through a reduction from the classical reachability problem for 2-counter machines. The main ingredient in the undecidability proof is an encoding of counters which allows testing for zero. The encoding represents each counter by a linked list of processes, where ordering on elements of the list is reflected by ordering on clock values of the relevant processes, and where the link between two elements in the list is encoded by whether two clocks belong to the same process. The value of a counter is reflected by the length of the corresponding list.

We also consider *Discrete Timed Networks (DTNs)*: a variant of timed networks where clocks are interpreted over a discrete time domain rather than a dense one. Surprisingly, it turns out that the controller state reachability problem now becomes decidable. The decidability result holds regardless of the number of clocks allowed inside each timed process. We show decidability using the theory introduced in [AČJYK00] for verification of transition systems which are monotonic with respect to a well quasi-ordering. More precisely, we define a *counter abstraction* for DTNs. This is an exact abstraction of the system where we only count the number of processes which have certain states and certain clock values. We show that such an abstraction induces a well quasi-ordering, and that the behaviour of a DTN is monotonic with respect to that ordering.

Related Work Most works on verification of parameterized systems consider the case where each component is a *finite-state system*. Applications include cache coherence protocols [Del00, EK03], broadcast protocols [EFM99], mutual exclusion protocols with linear topologies [KMM$^+$01], etc.

In [AMC02] a method is given for translating a timed automaton with several clocks into the parallel composition of a finite number of automata each operating on a single clock. This may give the impression that reachability problems for multi-clock TNs can in a similar way be reduced to corresponding problems for single-clock TNs. However, the construction given in [AMC02] will not work in our case. The reason is that, due to the unbounded number of timed processes, it is not possible to keep track of clocks belonging to the same process.

The works in [AAB00, AHV93] consider timed automata which are parameterized in the the following sense: transitions are guarded with predicates which compare clocks (and counters) with parameters possibly ranging over infinite domains. The models used in these papers assume a finite number of clocks and are therefore orthogonal to the models considered in this paper.

A work related to our result on DTNs is [GS92] where counter abstraction is used to obtain a Petri net model for parameterized systems. However, a process in [GS92] is assumed to be finite-state. Furthermore, counter abstraction in the case of DTNs yields a model with a different behaviour than that of Petri nets.

Outline: Section 2 gives the definition of timed networks. Section 3 recalls the classical model of 2-counter machines. Section 4 shows how a configuration of a 2-counter machine can be encoded by a configuration of a timed network, while Section 5 shows how the transitions of a 2-counter machine can be simulated by transitions of a timed network. We give an overview of the correctness proof for our encoding in Section 6. In Section 7 we give an algorithm for deciding the controller-state reachability problem for DTNs. Finally in Section 8, we conclude and give some directions for future work.

2 Definitions

In this section, we define *timed networks*: families of (infinitely many) systems each consisting of a *controller* and an arbitrary number of identical *timed processes*. The controller is a finite state automaton while each process is a timed automaton [AD94], i.e., a finite-state automaton which operates on a finite number of real-valued clocks $x_1, \ldots, x_K$. The values of all clocks are incremented continuously at the same rate. In addition, 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 a fixed number of processes synchronize and simultaneously change their states. A rule may be conditioned on the local state of the controller, together with the local states and clock values of the processes. If the conditions for a rule are satisfied, then a transition may be performed where the controller and each participating process changes its state. Also, during a transition, a process may reset some of its clocks to 0.

We use $\mathbb{N}$ and $\mathbb{R}^{\geq 0}$ for the set of natural numbers and set of non-negative real numbers respectively.

Timed Networks A *family of timed networks* (*timed network* for short) $\mathcal{N}$ with K clocks is a pair $(Q, \mathcal{R})$, where:

- Q is a finite set of *states*. The set Q is the union of two disjoint sets; the set Q^{ctrl} of *controller states*, and the set Q^{proc} of *process states*. These sets contain two distinguished *initial (idle)* states, namely $idle^c \in Q^{ctrl}$ and $idle^p \in Q^{proc}$.

- $\mathcal{R}$ is a finite set of *rules* where each rule is of the form

$$\begin{bmatrix} q_0 \\ \rightarrow \\ q_0' \end{bmatrix} \quad \begin{bmatrix} q_1 \\ g_1 \rightarrow R_1 \\ q_1' \end{bmatrix} \quad \cdots \quad \begin{bmatrix} q_n \\ g_n \rightarrow R_n \\ q_n' \end{bmatrix}$$

  such that $q_0, q_0' \in Q^{ctrl}$, and for all $i : 1 \leq i \leq n$ we have: $q_i, q_i' \in Q^{proc}$, and $g_i \rightarrow R_i$ is a guarded command where g_i is a boolean combination of predicates of the form $k \triangleright x$ for $k \in \mathbb{N}$, $\triangleright \in \{=, <, \leq, >, \geq\}$, $x \in \{x_1, \ldots, x_K\}$ and $R_i \subseteq \{x_1, \ldots, x_K\}$.

Intuitively, the set Q^{ctrl} represents the states of the controller and the set Q^{proc} represents the states of the processes. A rule of the above form describes a set of transitions of the network. The rule is enabled if the state of the controller is q_0 and if there are n processes with states

$q_1, \cdots, q_n$ whose clock values satisfy the corresponding guards. The rule is executed by simultaneously changing the state of the controller to q'_0 and the states of the n processes to $q'_1, \cdots, q'_n$, and resetting the clocks belonging to the sets $R_1, \ldots, R_n$.

For a guard g_i we write $g_i(y_1, \ldots, y_K)$ to denote the Boolean expression which results from substituting the occurrences of $x_1, \ldots, x_K$ in g_i by $y_1, \ldots, y_K$ respectively.

Configurations A configuration γ of a timed network $(Q, \mathcal{R})$ with K clocks is a tuple of the form $(I, q, \mathcal{Q}, X)$, where I is a finite *index set*, $q \in Q^{ctrl}$, $\mathcal{Q} : I \rightarrow Q^{proc}$, and $X : \{1, \ldots, K\} \rightarrow I \rightarrow \mathbb{R}^{\geq 0}$.

Intuitively, the configuration γ refers to the controller whose state is q, and to $|I|$ processes, whose states are defined by $\mathcal{Q}$. The clock values of the processes are defined by X. More precisely, for $k : 1 \leq k \leq K$ and $i \in I$, $X(k)(i)$ gives the value of clock x_k in the process with index i.

We use $|\gamma|$ to denote the number of processes in γ, i.e., $|\gamma| = |I|$. Also, we shall use X_k to denote the mapping $I \rightarrow \mathbb{R}^{\geq 0}$ such that $X_k(i) = X(k)(i)$.

Example 1 *Figure 1 shows graphical representation of a configuration in a timed network with two clocks, given by* $(\{1, 2, 3\}, q, \mathcal{Q}, X)$ *where* $\mathcal{Q}(1) = q_1, \mathcal{Q}(2) = q_2, \mathcal{Q}(3) = q_3$ *and* $X_1(1) = 0.1, X_1(2) = 0.5, X_1(3) = 5.0, X_2(1) = 2.3, X_2(2) = 1.4, X_2(3) = 0.6$.

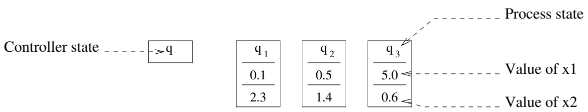

Figure 1. Graphical representation of a configuration in a timed network with two clocks.

Transition Relation The timed network $\mathcal{N}$ above induces a transition relation $\longrightarrow$ on the set of configurations. The relation $\longrightarrow$ is the union of a *discrete* transition relation $\longrightarrow_D$, representing transitions induced by the rules, and a *timed* transition relation $\longrightarrow_T$ which represents passage of time.

The discrete relation $\longrightarrow_D$ is the union $\bigcup_{r \in \mathcal{R}} \longrightarrow_r$, where $\longrightarrow_r$ represents a transition performed according to rule r. Let r be a rule of the form described in the above definition of timed networks. Consider two configurations $\gamma = (I, q, \mathcal{Q}, X)$ and $\gamma' = \langle I, q', \mathcal{Q}', X' \rangle$. We use $\gamma \longrightarrow_r \gamma'$ to denote that there is an injection $h : \{1, \ldots, n\} \rightarrow I$ such that for each $i : 1 \leq i \leq n$ and $k : 1 \leq k \leq K$ we have:

1. $q = q_0$, $\mathcal{Q}(h(i)) = q_i$, and $g_i(X_1(h(i)), \ldots, X_K(h(i)))$ holds. That is, the rule r is enabled.
2. $q' = q'_0$, and $\mathcal{Q}'(h(i)) = q'_i$. The states are changed according to r.
3. If $x_k \in R_i$ then $X'_k(h(i)) = 0$, while if $x_k \notin R_i$ then $X'_k(h(i)) = X_k(h(i))$. In other words, a clock is reset to 0 if it occurs in the corresponding set R_i. Otherwise its value remains unchanged.
4. $\mathcal{Q}'(j) = \mathcal{Q}(j)$ and $X'_k(j) = X_k(j)$, for $j \in I \setminus \text{range}(h)$, i.e., the process states and the clock values of the non-participating processes remain unchanged.

For a configuration $\gamma = (I, q, \mathcal{Q}, X)$ and $t \in \mathbb{R}^{\geq 0}$, we use γ^{+t} to denote the configuration $(I, q, \mathcal{Q}, X')$ where $X'_k(j) = X_k(j) + t$ for each $j \in I$ and $k : 1 \leq k \leq K$. A *timed transition* is of the form $\gamma \longrightarrow_{T=t} \gamma'$ where $\gamma' = \gamma^{+t}$. Such a transition lets time pass by t. We use $\gamma \longrightarrow_T \gamma'$ to denote that $\gamma \longrightarrow_{T=t} \gamma'$ for some $t \in \mathbb{R}^{\geq 0}$.

We define $\longrightarrow$ to be $\longrightarrow_D \cup \longrightarrow_T$ and use $\xrightarrow{*}$ to denote the reflexive transitive closure of $\longrightarrow$. Notice that if $\gamma \longrightarrow \gamma'$ then the index sets of γ and γ' are identical and therefore $|\gamma| = |\gamma'|$. For a configuration γ and a controller state q, we use $\gamma \xrightarrow{*} q$ to denote that there is a configuration γ' of the form $(I', q', \mathcal{Q}', X')$ such that $\gamma \xrightarrow{*} \gamma'$ and $q' = q$.

Reachability A configuration $\gamma_{init} = (I, q, \mathcal{Q}, X)$ is said to be *initial* if $q = idle^c$, $\mathcal{Q}(i) = idle^p$, and $X_k(i) = 0$ for each $i \in I$ and $k : 1 \leq k \leq K$. This means that an execution of a timed network starts from a configuration where the controller and all the processes are in their initial states, and the clock values are all equal to 0. Notice that there is an infinite number of initial configurations, namely one for each index set I.

Controller State Reachability Problem (TN(K)-Reach)

Instance A timed network $(Q, \mathcal{R})$ with K clocks and a controller state q_F.

Question Is there an initial configuration γ_{init} such that $\gamma_{init} \xrightarrow{*} q_F$?

Controller state reachability is relevant, since it can be shown, using standard techniques [VW86, GW93], that checking safety prperties (expressed as regular languages) can be translated into instances of the problem. In [AJ03] we show that TN(1)-Reach is decidable. In this paper we show

Theorem 1 *TN*(2)*-Reach is undecidable.*

3 2-Counter Machines

In this section we recall the standard definition of counter machines. Here, we assume that such a machine operates on two counters which we call c_1 and c_2.

A *two-counter machine* $\mathcal{C}$ is a tuple $(S, \mathcal{I})$ where S is a finite set of *local states* with a distinguished *initial local state* $s_{init} \in S$, and $\mathcal{I}$ is a finite set of *instructions*. An instruction θ is a triple (s_1, op, s_2), where $s_1, s_2 \in S$ and op is either an *increment* (of the form $c_1{+}{+}$ or $c_2{+}{+}$); a *decrement* (of the form $c_1{-}{-}$ or $c_2{-}{-}$); or a *zero testing* (of the form $c_1 = 0?$ or $c_2 = 0?$). A *configuration* β of a two-counter machine is a triple (s, m_1, m_2), where $s \in S$ represents the local state, and $m_1, m_2 \in \mathbb{N}$ represent the values of the counters c_1 and c_2 respectively. The counter machine $\mathcal{C}$ induces a transition relation $\rightsquigarrow$ on the set of configurations, which is defined as usual using the standard interpretations of counter operations. We use $\overset{*}{\rightsquigarrow}$ to denote the reflexive transitive closure of $\rightsquigarrow$. In a similar manner to timed networks, we use $\beta \overset{*}{\rightsquigarrow} s$ to denote that there is a configuration $\beta' = (s', m_1', m_2')$ such that $\beta \overset{*}{\rightsquigarrow} \beta'$ and $s' = s$. We define the *initial configuration* β_{init} to be $(s_{init}, 0, 0)$. The *control state reachability problem* for a 2-counter machines (CM-Reach) is: given local state s_F check whether $\beta_{init} \overset{*}{\rightsquigarrow} s_F$. The following result [Min61] is well-known.

Theorem 2 *CM-Reach is undecidable.*

In our correctness proof (Section 6), we use the relation $\overset{n}{\rightsquigarrow}$, with $n \geq 0$, on configurations, where $\beta \overset{n}{\rightsquigarrow} \beta'$ iff there is a sequence $\beta_0 \rightsquigarrow \beta_1 \rightsquigarrow \cdots \rightsquigarrow \beta_n$ with $\beta_0 = \beta$ and $\beta_n = \beta'$. The relation $\overset{n}{\rightsquigarrow}$ is extended to local states in a similar manner to $\overset{*}{\rightsquigarrow}$. Notice that $\overset{*}{\rightsquigarrow} = \cup_n \overset{n}{\rightsquigarrow}$.

4 Encoding of Configurations

We show undecidability of TN(2)-Reach through a reduction from CM-Reach. Given a counter machine $\mathcal{C} = (S, \mathcal{I})$, we shall derive a timed network $\mathcal{N}_\mathcal{C} = (Q_\mathcal{C}, \mathcal{R}_\mathcal{C})$ with two clocks. In this section, we perform the first step in the reduction; namely we describe how to construct the set $Q_\mathcal{C}$. Also, we describe how configurations of $\mathcal{C}$ are encoded as configurations of $\mathcal{N}_\mathcal{C}$. Finally, we introduce a special type of encodings, called *proper encodings*, which we use in our simulation of $\mathcal{C}$.

States According to the model described in Section 2, the set $Q_\mathcal{C}$ will consist of two disjoint sets of states: the set $Q_\mathcal{C}^{ctrl}$ of controller states and the set $Q_\mathcal{C}^{proc}$ of process states. The set $Q_\mathcal{C}^{ctrl}$ contains three types of states:

1. The initial controller state $idle^c$.
2. *Local states of* $\mathcal{C}$: all members of S have copies in $Q_\mathcal{C}^{ctrl}$.
3. *Temporary states*: the set $Q_\mathcal{C}^{ctrl}$ contains a state tmp^θ for each increment instruction $\theta \in \mathcal{I}$. These states are used as intermediate states during the simulation of increments (Section 5). The set $Q_\mathcal{C}^{ctrl}$ also contains the state s'_{init} (recall that s_{init} is the initial local state of $\mathcal{C}$). This state is used as an intermediate state in the initialization phase of the the simulation (Section 5).

The set $Q_\mathcal{C}^{proc}$ contains two types of states:

1. The initial process state $idle^p$.
2. Six states fst_1, mid_1, $last_1$, fst_2, mid_2, and $last_2$, used for encoding the two counters (as described below).

Encodings Each configuration β of $\mathcal{C}$ will be encoded by a set of configurations in $\mathcal{N}_\mathcal{C}$. The local state of β will be encoded by the controller state. Each counter will be modelled by a *counter encoding*. A counter encoding arranges a set of processes as a circular list. The ordering among elements of the list is defined by the clock values. The length of the list reflects to the value of the counter. To define counter encodings, we shall use the six process states fst_1, mid_1, $last_1$ (used for encoding of c_1), and fst_2, mid_2, $last_2$ (used for encoding of c_2). The states fst_1 and $last_1$ are the states of the first and last processes in the list encoding the value of c_1. All processes in the middle of the list will be in state mid_1. The states fst_2, mid_2, and $last_2$ play similar roles in the encoding of c_2. Formally, a configuration $\gamma = (I, q, \mathcal{Q}, X)$ is said to be a c_1-*encoding* of *value* m if there is an injection from the set $\{0, \ldots, m+1\}$ to I such that the following conditions are satisfied

- $\mathcal{Q}(h(0)) = fst_1$, $\mathcal{Q}(h(m+1)) = last_1$, and $\mathcal{Q}(h(i)) = mid_1$ for each $i : 1 \leq i \leq m$.
- $\mathcal{Q}(j) \in \{idle^p, fst_2, mid_2, last_2\}$ if $j \in I \setminus \mathrm{range}(h)$.
- $X_1(h(i)) < X_1(h(i+1))$ for each $i : 0 \leq i \leq m$.
- $X_2(h(i)) = X_1(h((i+1) \mod (m+2)))$, for each $i : 0 \leq i \leq m+1$.

The first condition states that the processes which are part of a c_1-encoding are in one of the local states fst_1, mid_1, or $last_1$. The second condition states that the processes which are not part of a c_1-encoding are in one of the local states $idle^p$, fst_2, mid_2, or $last_2$. The third and the fourth conditions show how the processes which are part of a c_1-encoding are ordered as a circular list. The position of each process in the list is reflected by values of its clocks x_1 and x_2. More precisely, the ordering among the x_1 clocks reflects the positions of the processes in the list. Also, clock x_2 of each process (except the last process) is equal to clock x_1 of the next process. Finally, clock x_2 of the last process is equal to clock x_1 of the first process (giving the list a "circular" form). We use $Val_1(\gamma)$ to denote the value m of a c_1-encoding γ.

Example 2 *Figure 2(b) shows a c_1-encoding of value 2. Figure 2(a) shows a graphical representation of the ordering among clock values. In Section 5 we shall use such a graphical representation to explain the different steps in the simulation of $\mathcal{C}$. Each ellipse contains clocks of equal values. Clocks in successive ellipses have increasing values, i.e., they are ordered from left (lower clock values) to right (higher clock values). The upper halves of the ellipses represent the x_1-clocks, while the lower halves represent the x_2-clocks. Each process is denoted by an edge whose end points are its two clocks. Such an edge is labelled by the current state of the process.*

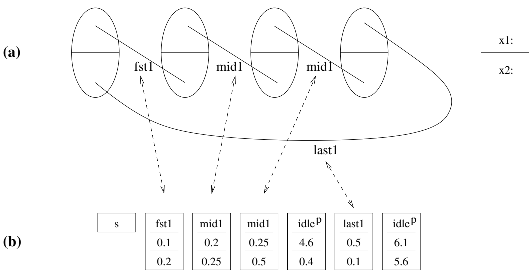

Figure 2. (a) Graphical representation of ordering among clocks in a c_1-encodings. (b) c_1-encodings satisfying the ordering on clocks described in (a).

A c_2-*encoding* and its value $Val_2(\gamma)$ are defined in a similar manner (replacing the states fst_1, mid_1, and $last_1$ by fst_2, mid_2, and $last_2$ in the encoding).

A configuration $\gamma = (I, q, \mathcal{Q}, X)$ is said to be an *encoding* if the following two conditions are satisfied:

- $q = s$ for some $s \in S$, i.e., q is the copy of a local state of $\mathcal{C}$.
- γ is both a c_1- and a c_2-encoding.

If γ satisfies the above conditions (i.e. if γ is an encoding), we define the *signature* $sig(\gamma)$ of γ to be the triple (s, m_1, m_2), where $m_1 = Val_1(\gamma)$ and $m_2 = Val_2(\gamma)$. Intuitively, the triple (s, m_1, m_2) will correspond to a configuration of $\mathcal{C}$. Notice that several (in fact infinitely many) configurations may have the same signature. However, all such configurations will have the same local states and the same orderings on clock values, and therefore will correspond to the same configuration in $\mathcal{C}$.

Proper Encodings In our simulation of $\mathcal{C}$ we shall rely on a particular kind of encodings, called *proper encodings*. An encoding γ of the form $(I, q, \mathcal{Q}, X)$ is said to be *proper* if it satisfies the following condition:

- For each $i \in I$ with $\mathcal{Q}(i) \neq idle^p$, $0 < X_1(i), X_2(i) < 1$.

In other words, all clocks participating in the encoding have values between (not including) zero and one. Certain steps of the simulation (see the decrementing operation in Section 5) are not possible to carry out without an upper bound on clock values of the processes. Working with proper encodings guarantees such an upper bound (namely an upper bound of one).

5 Encoding of Transitions

In this section, we perform the second step in deriving the timed network $\mathcal{N}_\mathcal{C} = (Q_\mathcal{C}, \mathcal{R}_\mathcal{C})$ from the counter machine $\mathcal{C} = (S, \mathcal{I})$. More precisely, we describe the set of rules $\mathcal{R}_\mathcal{C}$. The set $\mathcal{R}_\mathcal{C}$ contains the following rules:

Incrementing For each instruction $\theta = (s_1, c_1{+}{+}, s_2)$ in $\mathcal{C}$ there are two rules in $\mathcal{R}_\mathcal{C}$, namely

$$inc_1^\theta : \begin{bmatrix} s_1 \\ \rightarrow \\ tmp^\theta \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_1 \rightarrow \{x_1\} \\ mid_1 \end{bmatrix} \begin{bmatrix} idle^p \\ true \rightarrow \{x_2\} \\ fst_1 \end{bmatrix}$$

and the rule

$$inc_2^\theta : \begin{bmatrix} tmp^\theta \\ \rightarrow \\ s_2 \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_2 \rightarrow \{x_1\} \\ fst_1 \end{bmatrix} \begin{bmatrix} last_1 \\ true \rightarrow \{x_2\} \\ last_1 \end{bmatrix}$$

The total effect of the two rules is to increment the value of a c_1-encoding by adding one more process to the list. The rule inc_1^θ changes the state of the process which is currently first in the list to mid_1. This process will be placed in the second position in the new encoding. At the same time a new process is picked form the set of idle processes, and its state is changed to fst_1. The new process will be placed first in the list. Furthermore, the rule resets (and therefore equates) clock x_1 and x_2 respectively of the two above mentioned processes. This is done in order to maintain the invariant that clock x_2 of each process (except the last process) is equal to clock x_1 of the next process (recall the definition of an encoding from Section 4). The result of applying rule inc_1^θ on a c_1-encoding of value 2 is shown in Figure 3(b).

Rule inc_2^θ resets clock x_1 of the process which is now in state fst_1 and clock x_2 of the process which is last in the list. This is done in order to maintain (i) the invariant that clock x_1 of a process (here the first process) is smaller than clock x_1 of the next process; and (ii) the invariant that clock x_2 of the last process is equal to clock x_1 of the first process. The result of applying rule inc_2^θ is shown in Figure 3(c). Some remarks about rules inc_1^θ and inc_2^θ:

- After execution of inc_1^θ, the controller will be in state tmp^θ and therefore inc_2^θ is the only rule which may eventually be enabled after execution of inc_1^θ.
- The guard $0 < x_1$ in the definition of inc_1^θ is to guarantee that all clocks have positive values before the rule is applied. This makes sure that we avoid the scenario where we "accidentally" equate some clocks

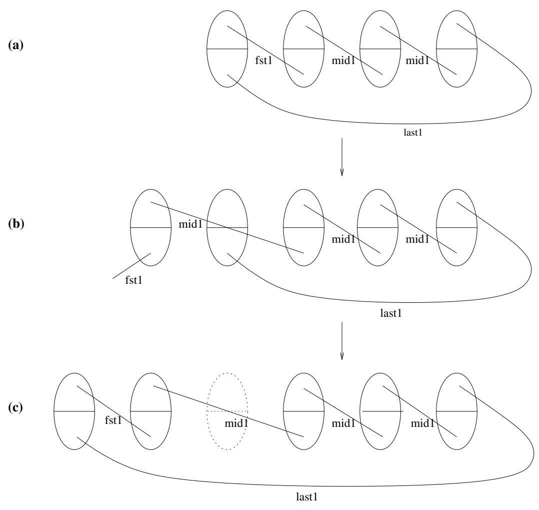

Figure 3. Simulating $(s_1, c_1{+}{+}, s_2)$ on a c_1-encoding

with the ones which are reset during the application of inc_1^θ. The same reasoning applies to the guard $0 < x_2$ in the definition of the rule inc_2^θ. Similar guards exist in the rest of the rules described in this section.

- After application of inc_2^θ, the resulting encoding will not be proper, since clocks have just been reset and their values are now zero. We can re-create a proper encoding by letting time pass through a timed transition. Again, a similar reasoning is applicable to the rest of the rules described in this section.

Also, for each instruction of the form $(s_1, c_2{+}{+}, s_2)$, there are two rules similar to the rules described above (replacing the states fst_1, mid_1, and $last_1$ by fst_2, mid_2 and $last_2$, respectively).

Decrementing For each instruction $\theta = (s_1, c_1{-}{-}, s_2)$ in $\mathcal{C}$ there is a rule in $\mathcal{R}_\mathcal{C}$, namely

$$dec^\theta : \begin{bmatrix} s_1 \\ \rightarrow \\ s_2 \end{bmatrix} \begin{bmatrix} last_1 \\ x_1 = 1 \rightarrow \emptyset \\ idle^p \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_1 \rightarrow \{x_1\} \\ fst_1 \end{bmatrix} \begin{bmatrix} mid_1 \\ x_2 = 1 \rightarrow \{x_2\} \\ last_1 \end{bmatrix}$$

The rule dec^θ decrements the value of a c_1-encoding by removing the last process of the list. More precisely, it changes the state of the last process to $idle^p$ (i.e. removes that process from the list), and changes the state of the process which is next last from mid_1 to $last_1$. In order to do that, we have to be able to identify the process which is next last in the list. Since all processes in the middle of the list are in state mid_1, we cannot identify the next last process simply by checking process states. Instead, we wait until the value of clock x_1 of the last process is equal to one. At that point of time, the process with clock x_2 equal to one is the next last process. Also, the rule resets (and therefore equates) clock x_1 of the first process and clock x_2 of the next last process (which will now become last in the list). Figure 4 shows the effect of applying the rule to a c_1-encoding.

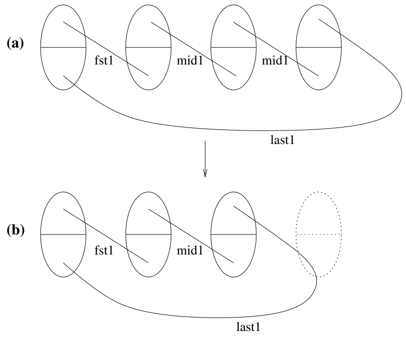

Figure 4. Simulating $(s_1, c_1{-}{-}, s_2)$ on a c_1-encoding

Some remarks about the rule dec^θ:

- Identifying the next last process (by waiting until some clocks are equal to one) uses the assumption that we start from a proper encoding. This implies that clocks of processes participating in the encoding have all values which are less than one. If this property is violated then the rule is not enabled (and will not become enabled through passage of time).

- The rule is not enabled in case the value of the c_1-encoding is equal to zero, since there will be no processes in state mid_1.

- Waiting for clock x_1 of the last process in the c_1-encoding to become equal to one may enforce clocks of processes in the c_2-encoding to become greater than one. More precisely, this happens if some clock in a process which is part of the c_2-encoding has a greater value than clock x_1 of the process which is currently in state $last_1$. After applying dec^θ, the value of such clocks will be greater than one, and therefore the resulting configuration will not be a *proper* encoding.

  Figure 5 illustrates this scenario. We consider a proper encoding (shown in Figure 5(a)) with signature $(s_1, 1, 1)$ such that clock x_1 of the process in state $last_1$ (0.35) is smaller than that of the process in state $last_2$ (0.65). In order to enable the rule dec^θ, we let time pass until clock x_1 of the process $last_1$ becomes equal to one (shown in Figure 5(b)). However, at this point of time, both clock x_2 of a process in state mid_2

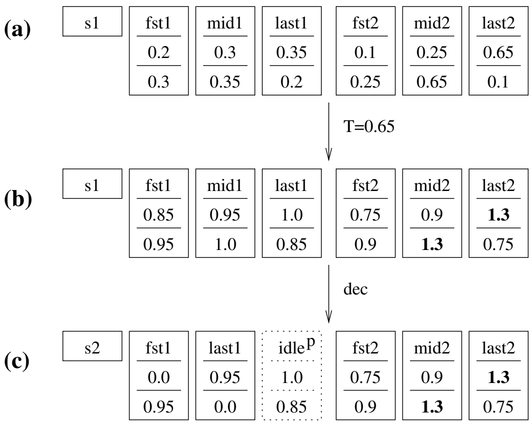

Figure 5. Decrementing may result in an improper encoding.

and x_1 of the process in state $last_2$ have become larger than one (1.3). Therefore, after applying the dec^θ, we get an encoding (of value $(s_2, 0, 1)$) shown in Figure 5(c), which is not proper. This prevents any later application of decrementing and zero-testing rules.

In order, to maintain the possibility of maintaining proper encodings in our simulation, we combine the the rule dec^θ with the *rotation* rules described below.

In a similar way to incrementing, there is also a rule corresponding to an instruction of the form $(s_1, c_2{-}{-}, s_2)$.

Rotation To make it always possible to obtain a proper encoding after decrementing the value of a c_1- or a c_2-encoding (see the *decrementing* rule above) we add a set of *rotation* rules. More precisely, for each state $s \in S$, the set $\mathcal{R}_\mathcal{C}$ contains the following two rules

$$rot_2^s : \begin{bmatrix} s \\ \rightarrow \\ s \end{bmatrix} \begin{bmatrix} fst_2 \\ 0 < x_1 \rightarrow \emptyset \\ mid_2 \end{bmatrix} \begin{bmatrix} last_2 \\ x_1 = 1 \rightarrow \{x_1\} \\ fst_2 \end{bmatrix} \begin{bmatrix} mid_2 \\ x_2 = 1 \rightarrow \{x_2\} \\ last_2 \end{bmatrix}$$

$$rotz_2^s : \begin{bmatrix} s \\ \rightarrow \\ s \end{bmatrix} \begin{bmatrix} fst_2 \\ (0 < x_1) \wedge (x_2 = 1) \rightarrow \{x_2\} \\ last_2 \end{bmatrix} \begin{bmatrix} last_2 \\ x_1 = 1 \rightarrow \{x_1\} \\ fst_2 \end{bmatrix}$$

Let us first explain the rule rot_2^s. The rule does not correspond to any instruction in $\mathcal{C}$; nor does it change the signature of the encoding. In simulating $\mathcal{C}$, we use the rotation rules in connection with decrementing. Recall that if $\theta = (s_1, c_1{-}{-}, s_2)$ then applying a rule dec^θ will not give a proper encoding in case the c_2-encoding has clocks with greater values than clock x_1 of the last process in the c_1-encoding (see Figure 5). The role of rot_2^s then is to decrement clock values of processes which are part of a c_2-encoding while preserving the signature of the whole encoding. More precisely, the rule rot_2^s moves the process which is in state $last_2$ and makes it first in the c_2-encoding. This amounts to a rotation of the the list corresponding to the c_2-encoding. The rotation can be repeated until sufficiently many processes in the c_2-encoding have been moved. When there are no clocks in the c_2-encoding with greater clock values than clock x_1 of the last process in the c_1-encoding, the rotation stops and dec^θ can now safely be applied.

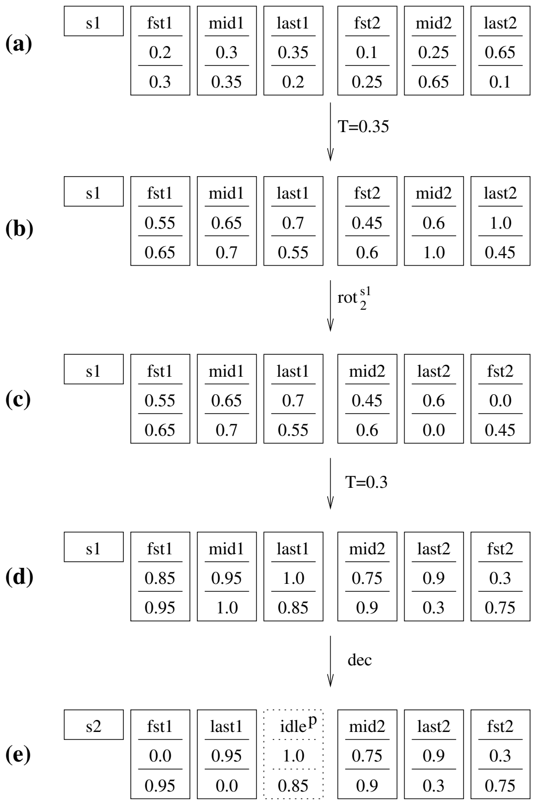

Figure 6. Decrementing preceded by rotation

We illustrate the role of rot_2^s through Figure 6. In a similar manner to Figure 5 we are interested in simulating a decrement instruction. However, instead of following the scenario of Figure 5, we now perform the following steps:

1. Wait for clock x_1 of the process in state $last_2$ to become equal to one (Figure 6(b)).

2. Apply the rule $rot_2^{s_1}$. This results in an encoding (shown in Figure 6(c)) where clock x_1 of the process in state $last_2$ (0.6) is smaller than that of the process

in state $last_1$ (0.7).

3. Wait until clock x_1 of the process in state $last_1$ becomes one (Figure 6(d)).

4. Apply the *decrementing* rule. Notice that the resulting encoding (shown in Figure 6(e)) has all clock values less than one.

After the last step, we can perform a timed transition and obtain a proper encoding.

Also if, before applying dec^θ, there is a clock in the c_2-encoding of the same value as clock x_1 of the process in state $last_1$, then we need to apply $rot_2^{s_2}$ once more after decrementing (this scenario does not occur in Figure 6, but is considered in the correctness proof).

Notice that we cannot apply the rotation rule in case the the value of the c_2-encoding is zero. This is due to the fact that the rule requires at least one process in state mid_2. The rule $rotz_2^s$ has the same role as rot_1^s with the difference that it can be applied when the value of the c_2-encoding is zero.

There are also similar rules rot_1^s and $rotz_1^s$ which are used to rotate a c_1-encoding and which are used in connection with rules of the form dec^θ with $\theta = (s_1, c_2{-}{-}, s_2)$.

Zero Testing For each instruction $\theta = (s_1, c_1 = 0?, s_2)$ in $\mathcal{C}$ there is a rule in $\mathcal{R}_\mathcal{C}$, namely

$$tst^\theta : \begin{bmatrix} s_1 \\ \rightarrow \\ s_2 \end{bmatrix} \begin{bmatrix} fst_1 \\ (0 < x_1) \wedge (x_2 = 1) \rightarrow \{x_2\} \\ last_1 \end{bmatrix} \begin{bmatrix} last_1 \\ x_1 = 1 \rightarrow \{x_1\} \\ fst_1 \end{bmatrix}$$

The rule checks that the value of the encoding is zero by testing that there are no processes in state mid_1. This is done by verifying that the process which is next last in the list is the same as the process which is first in the list. We identify the next last process in a similar manner to the case with decrementing. More precisely, we wait until the value of clock x_1 of the last process is equal to one. At that moment, we check the process with clock x_2 equal to one, and check whether that process has a state equal to fst_1. Notice that, clock x_2 of the first process and clock x_1 of the last process are now both equal to one and the encoding is no more proper. In order to be able to obtain a proper encoding again, we reset both these clocks and interchange the states of the processes in states fst_1 and $last_1$ respectively.

Notice the similarity between the rules tst^θ and $rotz_2^s$.

Some remarks about the rule tst^θ:

- The rule tst^θ is not enabled from an encoding γ with $sig(\gamma) = (s, m_1, m_2)$ and $m_1 \neq 0$, since, in such an encoding, values of clocks x_1 of the process in state $last_1$ and x_2 of the process in state fst_1 are different.

- Sometimes the rule tst^θ must be combined with the rotation rules according to the same scenarios explained for the *decrementing* rule.

In a similar way to the previous rules, there is also a rule corresponding to an instruction of the form $\theta = (s_1, c_2 = 0?, s_2)$.

Initialization The initial phase consists of the following two rules.

$$init_1 : \begin{bmatrix} idle^c \\ \rightarrow \\ s'_{init} \end{bmatrix} \begin{bmatrix} idle^p \\ true \rightarrow \{x_2\} \\ fst_1 \end{bmatrix} \begin{bmatrix} idle^p \\ true \rightarrow \{x_2\} \\ fst_2 \end{bmatrix} \begin{bmatrix} idle^p \\ true \rightarrow \{x_1\} \\ last_1 \end{bmatrix} \begin{bmatrix} idle^p \\ true \rightarrow \{x_1\} \\ last_2 \end{bmatrix}$$

$$init_2 : \begin{bmatrix} s'_{init} \\ \rightarrow \\ s_{init} \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_2 \rightarrow \{x_1\} \\ fst_1 \end{bmatrix} \begin{bmatrix} fst_2 \\ true \rightarrow \{x_1\} \\ fst_2 \end{bmatrix} \begin{bmatrix} last_1 \\ true \rightarrow \{x_2\} \\ last_1 \end{bmatrix} \begin{bmatrix} last_2 \\ true \rightarrow \{x_2\} \\ last_2 \end{bmatrix}$$

The role of the initialization rules is to bring $\mathcal{N}_\mathcal{C}$ from its initial configuration (where the controller and all processes are idle) into a configuration which is an encoding of the initial configuration β_{init} of $\mathcal{C}$. The rule $init_1$ takes the controller into the temporary state s'_{init}. It also picks four processes such that two processes become the first and last processes in the c_1-encoding (with value zero) and the other two processes become the first and last processes in the c_2-encoding (also with value zero). Clock x_2 of the first process and clock x_1 of the last process are reset. Rule $init_2$ changes the controller state to s_{init} and completes the creation of the c_1-encoding and c_2-encoding. This is done by first checking that some time has passed (through the guard $0 < x_2$), and then reseting both clock x_1 of the process which is now in state fst_1 (fst_2), and clock x_2 of the process which is now in state $last_1$ ($last_2$).

6 Correctness

In this section we show the correctness of the construction described in Section 4 and Section 5.

Let $\mathcal{C} = (S, \mathcal{I})$ be a counter machine and let $\mathcal{N}_\mathcal{C} = (Q_\mathcal{C}, \mathcal{R}_\mathcal{C})$ be a timed network derived from $\mathcal{C}$ as described in Section 4 and Section 5. Let $\rightsquigarrow$ and $\longrightarrow$ be the transition relations induced by $\mathcal{C}$ and $\mathcal{N}_\mathcal{C}$ respectively.

If s_F is a control state in $\mathcal{C}$ then the following holds

Theorem 3 *$\beta_{init} \overset{*}{\rightsquigarrow} s_F$ iff $\gamma_{init} \overset{*}{\longrightarrow} s_F$ for some initial configuration γ_{init} of $\mathcal{N}_\mathcal{C}$.*

The if-direction follows immediately from the the following lemma (shown in the appendix).

Lemma 4 *For any configuration $\gamma = (I, q, \mathcal{Q}, X)$ and initial configuration γ_{init} in $\mathcal{N}_\mathcal{C}$, if $\gamma_{init} \stackrel{*}{\longrightarrow} \gamma$ then one of the following holds.*

1. *q is not a member of S, (i.e. q is either a temporary state or the state $idle^c$).*
2. *γ is an encoding such that $\beta_{init} \stackrel{*}{\rightsquigarrow} sig(\gamma)$.*

The only-if-direction follows from the following lemma. The proof is shown in the appendix.

Lemma 5 *If $\beta_{init} \stackrel{n}{\rightsquigarrow} s_F$ then $\gamma_{init} \stackrel{*}{\longrightarrow} s_F$, for each $n \geq 0$ and initial configuration γ_{init} of $\mathcal{N}$ with $|\gamma_{init}| \geq n + 4$.*

The reason for the condition $|\gamma_{init}| \geq n + 4$ is that the sum of counter values never exceeds n in the path from β_{init} to s_F. Furthermore, each c_1- (or c_2)-encoding uses $m + 2$ processes for representing a counter value m. The lemma then states that the initial configuration, from which we start the simulation of the path from β_{init} to s_F, should be sufficiently large to incorporate all counter values which arise along that path.

The proofs (in the appendix) of Lemma 4 and Lemma 5 reflect the informal arguments provided together with each rule in Section 5.

7 Discrete Timed Networks

In this section, we show decidability of the controller state reachability problem for *Discrete Timed Networks (DTNs)*: timed networks in which the clocks assume values from the set of natural numbers. The idea of the proof is to define an ordering on configurations of the DTN. The ordering amounts to *counter abstraction*: for each configuration we count the number of processes which are in a given state and whose clocks are equal to some given values.

Discrete Timed Networks (DTN) The syntax of a DTN is the same as that of a TN (see Section 2). A configuration is also of the same form as in a TN. The behaviour of a DTN differs from that of a TN in two aspects, namely

- In a configuration $(I, q, \mathcal{Q}, X)$, the type of X is $\{1, \ldots, K\} \to I \to \mathbb{N}$, i.e., clocks have values which are natural numbers rather than reals.
- Timed transitions take only discrete steps, i.e., $\gamma_1 \longrightarrow_{T=t} \gamma_2$ if $\gamma_2 = \gamma_1^{+t}$ where $t \in \mathbb{N}$. Discrete transitions are defined in a similar manner to TN.

The Problem DTN(K)-Reach is defined in the same manner as TN(K)-Reach except that the timed network $\mathcal{N}$ in the definition of the problem is now given a discrete interpretation as described above.

In this section we show

Theorem 6 *DTN(K)-Reach is decidable for each $K \in \mathbb{N}$.*

To prove Theorem 6, we rely on the theory introduced in [AČJYK00].

Monotonic Transition Systems (MTS) A *monotonic transition system (MTS)* is a tuple $(\Gamma, \mathit{Init}, \preceq, \hookrightarrow, U)$, where

- Γ is a (potentially infinite) set of *configurations*.
- $\mathit{Init} \subseteq \Gamma$ is a set of *initial* configurations.
- $\preceq$ is a computable ordering on Γ, i.e., for each $\gamma_1, \gamma_2 \in \Gamma$, we can check whether $\gamma_1 \preceq \gamma_2$. Furthermore, $\preceq$ is a *well quasi-ordering*, i.e., for each infinite sequence $\gamma_0, \gamma_1, \gamma_2, \ldots$ there are i and j with $i < j$ and $\gamma_i \preceq \gamma_j$.
- $\hookrightarrow$ is a binary *transition relation* on Γ. Furthermore, $\hookrightarrow$ is monotonic with respect to $\preceq$, i.e., given configurations $\gamma_1, \gamma_2, \gamma_3$ such that $\gamma_1 \hookrightarrow \gamma_2$ and $\gamma_1 \preceq \gamma_3$, there is a configuration γ_4 such that $\gamma_3 \hookrightarrow \gamma_4$ and $\gamma_2 \preceq \gamma_4$.
- U is defined as the upward closure $\Gamma_1 \uparrow$ of a finite set $\Gamma_1 \subseteq \Gamma$, where $\Gamma_1 \uparrow = \{\gamma' \in \Gamma \mid \exists \gamma \in \Gamma_1.\, \gamma \preceq \gamma'\}$.

We use $\stackrel{*}{\hookrightarrow}$ to denote the the reflexive transitive closure of $\hookrightarrow$. For sets $\Gamma_1, \Gamma_2 \subseteq \Gamma$, we say that Γ_2 is *reachable* from Γ_1 if there are $\gamma_1 \in \Gamma_1$ and $\gamma_2 \in \Gamma_2$ such that $\gamma_1 \stackrel{*}{\hookrightarrow} \gamma_2$.

The reachability problem for MTS (*MTS-Reach*) is defined as follows:

Instance An MTS $(\Gamma, \mathit{Init}, \preceq, \hookrightarrow, U)$.

Question Is U reachable from *Init*?

In [AČJYK00] we give sufficient conditions for decidability of MTS-Reach as follows. For $\Gamma_1 \subseteq \Gamma$, we define $\mathit{Pre}(\Gamma_1)$ to be the set $\{\gamma \mid \exists \gamma_1 \in \Gamma_1.\, \gamma \hookrightarrow \gamma_1\}$. For $\Gamma_1 \subseteq \Gamma$, we say that $M \subseteq \Gamma_1$ is *minor* set of Γ_1 if

- for each $\gamma_1 \in \Gamma_1$ there is $\gamma_2 \in M$ such that $\gamma_2 \preceq \gamma_1$.
- If $\gamma_1, \gamma_2 \in M$ and $\gamma_1 \preceq \gamma_2$ then $\gamma_1 = \gamma_2$.

Since $\preceq$ is a wqo, it follows that each minor set is finite. However, for the same set, there may be several minor sets. We use *min* to denote a function which, given $\Gamma_1 \subseteq \Gamma$, returns a minor set of Γ_1. We use $\mathit{minpre}(\gamma)$ to denote the set $\mathit{min}(\mathit{Pre}(\{\gamma\} \uparrow))$.

In [AČJYK00] we show that the following conditions are sufficient for decidability of MTS-Reach.

Theorem 7 *MTS-Reach is decidable if for each $\gamma \in \Gamma$*

- *we can check whether $\gamma \in \mathit{Init}$.*
- *the set $\mathit{minpre}(\gamma)$ is finite and computable.*

From DTN to MTS A DTN $\mathcal{N} = (Q, \mathcal{R})$ with K clocks, together with a controller state q_F induces an MTS $(\Gamma, \mathit{Init}, \preceq, \hookrightarrow, U)$ as follows

- Γ is the set of configurations of $\mathcal{N}$.
- *Init* is the set of initial configurations of $\mathcal{N}$.
- To define $\preceq$, we first introduce a function

  $$\# \;:\; Q^{proc} \times \{0, \ldots, max+1\}^K \;\rightarrow\; \Gamma \;\rightarrow\; \mathbb{N}$$

  where max is the maximum natural number which occurs in the definitions of the rules in the DTN. Given $p \in Q^{proc}$, $m_1, \ldots, m_K \in \{0, \ldots, max+1\}$, and $\gamma = (I, q, \mathcal{Q}, X)$ we define $\#(p, m_1, \ldots, m_K)(\gamma)$ to be the size of the set $I_1 \subseteq I$ such that for each index $i \in I_1$, $\mathcal{Q}(i) = p$ and for each $k : 1 \leq k \leq K$, one of the following conditions holds

  - $X(k)(i) = m_k$ and $m_k \leq max$.
  - $X(k)(i) > m_k$ and $m_k = max + 1$.

  In other words, $\#(p, m_1, \ldots, m_K)(\gamma)$ counts the number of processes in γ whose states are p and whose clock values are given by $m_1, \ldots, m_K$ respectively (we identify clock values larger than max). For $\gamma = (I, q, \mathcal{Q}, X)$ and $\gamma' = (I', q', \mathcal{Q}', X') \in \Gamma$, we use $\gamma \preceq \gamma'$ to denote that the following two conditions hold:

  - $q = q'$
  - $\#(p, m_1, \ldots, m_K)(\gamma) \leq \#(p, m_1, \ldots, m_K)(\gamma')$ for each $p \in Q^{proc}$, $m_1, \ldots, m_K \in \{0, \ldots, max+1\}$.

  The ordering $\preceq$ is trivially computable. Well quasi-ordering of $\preceq$ follows from Dickson's Lemma [Dic13]: according to the ordering, we can represent a configuration by a vector of natural numbers, where each entry of the vector is indexed by a tuple of the form $(p, m_1, \ldots, m_K)$.

- $\hookrightarrow$ is the relation $\longrightarrow$ defined in Section 2 adapted to DTN as described above.

  Lemma 8 *$\longrightarrow$ is monotonic wrt $\preceq$.*

- $U = \{(\emptyset, q_F, \mathcal{Q}, X)\} \uparrow$, i.e., U is the set of all configurations with controller state q_F.

This means that a DTN indeed induces an MTS. Notice that it is trivial to check whether a given configuration is initial. The following lemma (shown in the appendix) states that the induced transition system also satisfies the second sufficient condition for decidability (see Theorem 7).

Lemma 9 *Consider the MTS induced by a DTN. Then, for each configuration γ we can compute $minpre(\gamma)$ as a finite set of configurations.*

This, together with Theorem 7, proves Theorem 6.

8 Conclusions, Discussion, and Future Work

We have shown undecidability of controller state reachability for multi-clock timed networks. We have also shown decidability of the problem when clocks are interpreted over a discrete time domain.

In this paper, we assume a lazy behaviour for TNs. This means that we may choose to let time pass instead of performing discrete transitions, even if that makes these transitions disabled, due to some of the clocks becoming "too old". In fact, we can use the techniques in [JLL77] to show that, in the case of urgent behaviour, the controller state reachability is undecidable even for single-clock TNs. Also, in this paper we only consider safety properties. Liveness properties have been shown to be undecidable for single-clock TNs in [AJ03].

The ordering we provide for proving decidability of DTN corresponds to an abstraction of configurations where we count the number of processes which are in a certain state and which have certain clock values. In a similar manner to [GS92] we can view this abstraction as a "Petri net"-like model where each place corresponds to one combination of process states and clock values. In contrast to [GS92], the transitions in the abstract model do not correspond to those of a Petri net. The main difference is that a timed transition simultaneously moves all tokens from each place, corresponding to a certain clock value, to the place corresponding to the next clock value. Comparing to the model of *Transfer Nets* [FRSB02], a timed transition here corresponds to "parallel transfers", i.e. a set of transfers which are performed simultaneously. An alternative way to prove our decidability result would be to simulate a DTN by a transfer net. One ingredient in such a simulation is to simulate parallel transfers by sequences of transfer operations.

There are several classes of protocols which can be modelled as multi-clock TNs, such as the parameterized versions of the protocols in [BGK+96] and [MT01]. This means that, despite our undecidability result, it is interesting to design semi-algorithms for multi-clock TNs. One direction for future work is to design acceleration techniques which are sufficiently powerful to handle such classes of protocols.

References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Proc. 12^{th} Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, 2000.

[AČJYK00] Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160:109–127, 2000.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AHV93] R. Alur, T. Henzinger, and M. Vardi. Parametric real-time reasoning. In *Proc.* 25^{th} *ACM Symp. on Theory of Computing*, Lecture Notes in Computer Science, pages 592–601, 1993.

[AJ03] Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 290(1):241–264, 2003.

[AMC02] E. Asarin, O. Maler, and P. Caspi. Timed regular expressions. *The Journal of ACM*, 2002.

[BGK+96] J. Bengtsson, W. O. D. Griffioen, K.J. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Verification of an audio protocol with bus collision using UPPAAL. In *Proc.* 8^{th} *Int. Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 244–256, 1996.

[Del00] G. Delzanno. Automatic verification of cache coherence protocols. In Emerson and Sistla, editors, *Proc.* 12^{th} *Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 53–68. Springer Verlag, 2000.

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. J. Math.*, 35:413–422, 1913.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. LICS' 99* 14^{th} *IEEE Int. Symp. on Logic in Computer Science*, 1999.

[EK03] E.A. Emerson and V. Kahlon. Model checking guarded protocols. In *Proc. LICS' 03* 19^{th} *IEEE Int. Symp. on Logic in Computer Science*, Lecture Notes in Computer Science, 2003.

[FRSB02] A. Finkel, J.-F. Raskin, M. Samuelides, and L. Van Begin. Monotonic extensions of petri nets: Forward and backward search revisited. In *Proc. Infinity'02*, 2002.

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

[GW93] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, 1993.

[JLL77] N. D. Jones, L. H. Landweber, and Y. E. Lyen. Complexity of some problems in Petri nets. *Theoretical Computer Science*, (4):277–299, 1977.

[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, 1997.

[KMM+01] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.

[Min61] M. Minsky. Recursive unsolvabitity of post's problem of tag and other topics in the theory of turing machines. *Ann. of Math.*, 74:437–455, 1961.

[MT01] I. A. Mason and C. L. Talcott. Simple network protocol simulation within maude. In Kokichi Futatsugi, editor, *Electronic Notes in Theoretical Computer Science*, volume 36. Elsevier, 2001.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86,* 1^{st} *IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.

Appendix : Proofs of Lemmas

In the proofs we need some definitions. Let $\mathcal{N} = (Q, \mathcal{R})$ be a timed network. We define $\Longrightarrow_r$ to denote $\longrightarrow_T \circ \longrightarrow_r \circ \longrightarrow_T$, i.e. $\Longrightarrow_r$ corresponds to performing a discrete transition according to the rule r, preceded and followed by a timed transition. We define $\Longrightarrow$ to be $\bigcup_{r\in\mathcal{R}} \Longrightarrow_r$. For a set $\mathtt{R} \subseteq \mathcal{R}$ of rules, we let $\gamma_1 \Longrightarrow_{\mathtt{R}} \gamma_2$ denote that $\gamma_1 \Longrightarrow_r \gamma_2$ for some $r \in \mathtt{R}$. We use $\stackrel{*}{\Longrightarrow}$, $\stackrel{*}{\Longrightarrow}_r$ and $\stackrel{*}{\Longrightarrow}_{\mathtt{R}}$ to denote the reflexive transitive closure of the respective relations.

We introduce a new type of encodings to describe the effect of the rule inc_1^θ. For $m \geq 1$, a configuration $\gamma = (I, q, \mathcal{Q}, X)$ is said to be a *semi-c_1-encoding* of *value* m if there is an injection h from $\{0, \ldots, m+1\}$ to I such that the following conditions are satisfied

- $\mathcal{Q}$ is defined as in a c_1-encoding.
- $X_2(h(0)) = X_1(h(1)) < X_2(h(m+1)) < X_2(h(1))$.
- $X_1(h(i)) < X_1(h(i+1))$ for each $i : 2 \leq i \leq m$.
- $X_2(h(i)) = X_1(h(i+1))$, for each $i : 1 \leq i \leq m$.

A graphical representation of a semi-c_1-encoding is shown in Figure 3(b).

In a similar manner to a c_1-encoding, we use $Val_1(\gamma)$ to denote the value m of a semi-c_1-encoding γ.

A configuration $\gamma = (I, q, \mathcal{Q}, X)$ is said to be a *Type* 1 *semi-encoding* if it satisfies the following two conditions:

- $q = tmp^\theta$ for some increment (of the form $\theta = (s_1, c_1{+}{+}, s_2)$ or of the form $\theta = (s_1, c_2{+}{+}, s_2)$).
- γ is both a c_2-encoding and a semi-c_1-encoding.

In such a case, we define $sig(\gamma)$ of γ to be the triple $\left(tmp^\theta, m_1, m_2\right)$, where $m_1 = Val_1(\gamma)$ and $m_2 = Val_2(\gamma)$. Also, we define $next(\gamma)$ to be (s_2, m_1, m_2). Intuitively, $next(\gamma)$ is the signature of the configuration which occurs next time we perform a discrete transition (by rule inc_2^θ) in our simulation.

The notion of a semi-encoding of Type 1 can be extended to a *proper* semi-encoding in the same manner as before, i.e. we require clocks of all processes which are not idle to have values strictly between zero and one. A (proper) semi-encoding of Type 2 is defined in a similar manner.

Proof of Lemma 4

Suppose that $\gamma_{init} \stackrel{*}{\longrightarrow} \gamma$. If $\gamma_{init} \longrightarrow_T \gamma$ then the result follows immediately. Otherwise, $\gamma_{init} \stackrel{*}{\Longrightarrow} \gamma$, i.e., there is a sequence

$$\gamma_{init} = \gamma_0 \Longrightarrow_{r_0} \gamma_1 \Longrightarrow_{r_1} \gamma_2 \Longrightarrow_{r_2} \cdots \Longrightarrow_{r_{n-1}} \gamma_n = \gamma$$

Let $\gamma_i = (I, q_i, \mathcal{Q}_i, X_i)$ for $i : 0 \leq i \leq n$. We notice that $q_0 = idle^c$. By definition of the rules, it must be the case that $r_0 = init_1$ and therefore $q_1 = s'_{init}$. In other words, both γ_0 and γ_1 satisfy the claim of the Lemma. Lemma 4 follows from the following property:

For each $2 \leq i \leq n$, it is the case that γ_i is either

- an encoding with $\beta_{init} \stackrel{*}{\leadsto} sig(\gamma_i)$; or
- a semi-encoding with $\beta_{init} \stackrel{*}{\leadsto} next(\gamma_i)$.

This property is shown using an induction on i. For the base case we observe that, by definition of the rules, it follows that $r_1 = init_2$ and therefore $sig(\gamma_2) = \beta_{init}$. For the induction step, we observe that, for each $i : 2 \leq i < n$, it follows from the rule definitions that one of the following cases is satisfied:

1. $r_i = inc_1^\theta$ for some $\theta = (s_1, c_1{+}{+}, s_2)$, γ_i is an encoding with $sig(\gamma_i) = (s_1, m_1, m_2)$, and γ_{i+1} is a Type 1 semi-encoding with $sig(\gamma_{i+1}) = \left(tmp^\theta, m_1+1, m_2\right)$.
2. $r_i = inc_2^\theta$ for some $\theta = (s_1, c_1{+}{+}, s_2)$, γ_i is a Type 1 semi-encoding with $sig(\gamma_i) = \left(tmp^\theta, m_1, m_2\right)$, and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s_2, m_1, m_2)$.
3. $r = dec^\theta$ for some $\theta = (s_1, c_1{-}{-}, s_2)$, γ_i is an encoding with $sig(\gamma_i) = (s_1, m_1, m_2)$, $m_1 > 0$, and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s_2, m_1 - 1, m_2)$.
4. $r = rot_1^s$ for some $s \in S$ and $sig(\gamma_i) = sig(\gamma_{i+1})$.
5. $r = rotz_1^s$ for some $s \in S$ and $sig(\gamma_i) = sig(\gamma_{i+1})$.
6. $r = tst^\theta$ for some $\theta = (s_1, c_1 = 0?, s_2)$, with $sig(\gamma_i) = (s_1, 0, m_2)$ and $sig(\gamma_{i+1}) = (s_2, 0, m_2)$.
7. Similar cases corresponding to instructions which change counter c_2.

Proof of Lemma 5

To show Lemma 5 we use some definitions. Let $\gamma = (I, q, \mathcal{Q}, X)$ be a (semi-)encoding. Let $i \in I$ be the (unique) index such that $\mathcal{Q}(i) = last_1$. We define $Latest_1(\gamma) = X_1(i)$. In other words, $Latest_1(\gamma)$ is the highest among values of clocks belonging to processes which are part of the (semi-)c_1-encoding. We define $Latest_2(\gamma)$ in a similar manner, and define $Latest(\gamma) = max(Latest_1(\gamma), Latest_2(\gamma))$.

Let $Delay_1(\gamma)$ be the size of the set $J \subseteq I$ such that $j \in J$ iff $\mathcal{Q}(j) \in \{fst_2, mid_2, last_2\}$ and $Latest_1(\gamma) < X_1(j)$. In other words, $Delay_1(\gamma)$ is the number of processes which are part of the c_2-encoding and which have clocks with values higher than any clock of a process which is part of the

c_1-encoding. We define $Delay_2(\gamma)$ in a similar manner. Notice that it may be the case that both $Delay_1(\gamma) = 0$ and $Delay_2(\gamma) = 0$ (if the maximum clock values are equal in the c_1- and the c_2-encoding).

Lemma 5 follows immediately from the following lemma

Lemma 10 *For each $n \geq 0$ and initial configuration γ_{init}, if $\beta_{init} \overset{n}{\rightsquigarrow} \beta$ and $|\gamma_{init}| \geq n+4$, then there exists a proper encoding γ such that $\gamma_{init} \overset{*}{\Longrightarrow} \gamma$ and $sig(\gamma) = \beta$.*

Proof We prove this lemma by induction on n.

In the base case ($n = 0$), we have $\beta = \beta_{init}$ and $|\gamma_{init}| \geq 4$. By the definition of $init_1$, this rule is enabled. Let γ_1 be such that $\gamma_{init} \longrightarrow_{init_1} \gamma_1$. Define $\gamma_2 = \gamma_1^{+t_1}$ with $0 < t_1 < 1$. We have $\gamma_1 \longrightarrow_{T=t_1} \gamma_2$. Rule $init_2$ is now enabled. Let γ_3 be such that $\gamma_2 \longrightarrow_{init_2} \gamma_3$. By definition of $init_2$, γ_3 is an encoding and $sig(\gamma_3) = \beta_{init}$. Let t_2 be such that $0 < t_2 < 1 - Latest(\gamma_3)$. t_2 exists by the definition of t_1, $init_1$ and $init_2$. Let $\gamma_4 = \gamma_3^{+t_2}$. γ_4 is a proper encoding with $sig(\gamma_4) = \beta_{init}$. Notice that the transitions $\longrightarrow_{init_1}$ and $\longrightarrow_{init_2}$ are enabled only because $|\gamma_{init}| \geq 4$.

For the induction step, assume that $\beta_{init} \overset{n+1}{\rightsquigarrow} \beta$ and $|\gamma_{init}| \geq n+5$. We know that there is a β_1 with $\beta_{init} \overset{n}{\rightsquigarrow} \beta_1 \rightsquigarrow \beta$. By the induction hypothesis, it follows that there is a proper encoding γ_1 such that $sig(\gamma_1) = \beta_1$ and $\gamma_{init} \overset{*}{\Longrightarrow} \gamma_1$. We need to show that there is a proper encoding γ with $sig(\gamma) = \beta$ and $\gamma_1 \overset{*}{\Longrightarrow} \gamma$. This follows from the following lemma.

Lemma 11 *Let β_1 and β_2 be configurations of $\mathcal{C}$, where $\beta_1 \rightsquigarrow \beta_2$ and β_2 is of the form (s, m_1, m_2). Let γ_1 be a proper encoding such that $sig(\gamma_1) = \beta_1$ and $|\gamma_1| \geq m_1 + m_2 + 4$. There is a proper encoding γ_2 such that $sig(\gamma_2) = \beta_2$ and $\gamma_1 \overset{*}{\Longrightarrow} \gamma_2$.*

The proof of Lemma 11 follows from Lemma 12, Lemma 13, Lemma 16, and Lemma 17:

- Lemma 12 and Lemma 13 state that an *increment* can be simulated by an application of the rule inc_1^θ followed by an application of the rule inc_2^θ.
- Lemma 16 states that a *decrement* can be simulated by the rule dec^θ possibly preceded and followed by a number of rotations. This lemma follows from Lemma 14 and Lemma 15.
- Lemma 17 deals with zero testing and is similar to Lemma 16.

The condition $|\gamma_1| \geq m_1+m_2+4$ in the claim of Lemma 11 is relevant only in Lemma 12, since this is the only case where the value of a counter is increased.

Lemma 12 *Consider an instruction $\theta = (s_1, c_1{+}{+}, s_2)$. Let γ_1 be a proper encoding with $sig(\gamma_1) = (s_1, m_1, m_2)$ and $|\gamma_1| \geq m_1 + m_2 + 5$. There is a proper semi-encoding γ_2 of Type 1 such that $sig(\gamma_2) = \left(tmp^\theta, m_1+1, m_2\right)$ and $\gamma_1 \Longrightarrow_{inc_1^\theta} \gamma_2$.*

A similar result holds in case θ is of the form $(s_1, c_2{+}{+}, s_2)$.

Proof Since $|\gamma_1| \geq m_1 + m_2 + 5$, there is at least one process in γ_1 whose state is $idle^p$ (we need $m_1 + 2$ processes for the c_1-encoding and $m_2 + 2$ processes for the c_2-encoding, which means that we have at least one process left to be in state $idle^p$). This together with the fact that γ_1 is a proper encoding implies that inc_1^θ is enabled from γ_1, i.e., there is configuration γ_3 with $\gamma_1 \longrightarrow_{inc_1^\theta} \gamma_3$. Define $\gamma_2 = \gamma_3^{+t}$ where $0 < t < 1 - Latest(\gamma_3)$. Such a t exists by definition of inc_1^θ and since γ_1 is a proper encoding. By the definitions it follows that γ_2 is a proper semi-encoding of Type 1 with $sig(\gamma_2) = \left(tmp^\theta, m_1+1, m_2\right)$ and $\gamma_1 \longrightarrow_{inc_1^\theta} \gamma_3 \longrightarrow_{T=t} \gamma_2$.

Lemma 13 *Consider an instruction $\theta = (s_1, c_1{+}{+}, s_2)$. Let γ_1 be a proper semi-encoding of Type 1 with $sig(\gamma_1) = \left(tmp^\theta, m_1, m_2\right)$. There is a proper encoding γ_2 such that $sig(\gamma_2) = (s_2, m_1, m_2)$ and $\gamma_1 \Longrightarrow_{inc_2^\theta} \gamma_2$.*

A similar result holds in case θ is of the form $(s_1, c_2{+}{+}, s_2)$.

Proof Since γ_1 is a proper semi-encoding of Type 1, it follows that inc_2^θ is enabled from γ_1, i.e., there is a configuration γ_3 with $\gamma_1 \longrightarrow_{inc_2^\theta} \gamma_3$. Define $\gamma_2 = \gamma_3^{+t}$ where $0 < t < 1 - Latest(\gamma_3)$. Such a t exists by definition of inc_2^θ and since γ_1 is proper semi-encoding. By the definitions it follows that γ_2 is a proper encoding with $sig(\gamma_2) = (s_2, m_1, m_2)$ and $\gamma_1 \longrightarrow_{inc_2^\theta} \gamma_3 \longrightarrow_{T=t} \gamma_2$.

Lemma 14 *Let γ_1 be a proper encoding with $Delay_1(\gamma_1) > 0$ and $sig(\gamma_1) = (s, m_1, m_2)$. There is a proper encoding γ_2 such that $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$, $sig(\gamma_1) = sig(\gamma_2)$, and either $\gamma_1 \Longrightarrow_{rot_2^s} \gamma_2$ or $\gamma_1 \Longrightarrow_{rotz_2^s} \gamma_2$.*

A similar result holds in case $Delay_2(\gamma_1) > 0$.

Proof We distinguish between two cases, namely when $m_2 > 0$ and when $m_2 = 0$.

First, we assume that $m_2 > 0$. Define $\gamma_3 = \gamma_1^{+t_1}$ where $t_1 = 1 - Latest_2(\gamma_1)$. Such a t_1 exists since γ_1 is a proper encoding. From the definition of t_1 and the fact that $m_2 > 0$ it follows that rot_2^s is enabled from γ_3, i.e., there is a γ_4 with $\gamma_3 \longrightarrow_{rot_2^s} \gamma_4$. Define $\gamma_2 = \gamma_4^{+t_2}$ where $0 < t_2 < 1 - Latest(\gamma_4)$. Existence of t_2 follows from the manner in which t_1 is chosen, definition of the rule rot_2^s, and since γ_1 is a proper encoding. By the definitions it follows that γ_2 is a proper encoding with $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$, $sig(\gamma_2) = sig(\gamma_1)$.

The case when $m_2 = 0$ is similar. Here we replace the rule $rot_2^{s_2}$ by the rule $rotz_2^{s_2}$, and obtain $\gamma_1 \longrightarrow_{T=t_1} \gamma_3 \longrightarrow_{rotz_2^s} \gamma_4 \longrightarrow_{T=t_2} \gamma_2$.

Lemma 15 *Consider an instruction $\theta = (s_1, c_1{-}{-}, s_2)$. Let γ_1 be a proper encoding with $sig(\gamma_1) = (s_1, m_1, m_2)$ and $m_1 > 0$. If $Delay_1(\gamma_1) = 0$ then there is a proper encoding γ_2 such that $sig(\gamma_2) = (s_2, m_1 - 1, m_2)$ and one of the following holds.*

1. *If $Latest_1(\gamma_1) > Latest_2(\gamma_1)$ then $\gamma_1 \Longrightarrow_{dec^\theta} \gamma_2$.*
2. *If $Latest_1(\gamma_1) = Latest_2(\gamma_1)$ and $m_2 > 0$ then $\gamma_1 \Longrightarrow_{dec^\theta} \circ \Longrightarrow_{rot_2^{s_2}} \gamma_2$.*
3. *If $Latest_1(\gamma_1) = Latest_2(\gamma_1)$ and $m_2 = 0$ then $\gamma_1 \Longrightarrow_{dec^\theta} \circ \Longrightarrow_{rotz_2^{s_2}} \gamma_2$.*

A similar result holds in case θ is of the form $(s_1, c_2{-}{-}, s_2)$.

Proof Define $\gamma_3 = \gamma_1^{+t_1}$ where $t_1 = 1 - Latest_1(\gamma_1)$. Such a t_1 exists since γ_1 is a proper encoding. From the definition of t_1 and the fact that $m_1 > 0$ it follows that dec^θ is enabled from γ_3, i.e., there is a γ_4 with $\gamma_3 \longrightarrow_{dec^\theta} \gamma_4$. Now there are three cases depending on the values of $Latest_1(\gamma_1)$ and $Latest_2(\gamma_1)$ as follows:

1. If $Latest_1(\gamma_1) > Latest_2(\gamma_1)$ then define $\gamma_2 = \gamma_4^{+t_2}$ where $0 < t_2 < 1 - Latest(\gamma_4)$. Existence of t_2 follows from the manner in which t_1 is chosen, definition of the rule dec^θ, and since γ_1 is a proper encoding. By the definitions it follows that γ_2 is a proper encoding with $sig(\gamma_2) = (s_2, m_1 - 1, m_2)$, and $\gamma_1 \longrightarrow_{T=t_1} \gamma_3 \longrightarrow_{dec^\theta} \gamma_4 \longrightarrow_{T=t_2} \gamma_2$.
2. If $Latest_1(\gamma_1) = Latest_2(\gamma_1)$ and $m_2 > 0$. It follows that clock x_1 of the last process in the c_2-encoding has value 1 in γ_4. Therefore, the rule $rot_2^{s_2}$ is enabled from γ_4, i.e., there is a γ_5 with $\gamma_4 \longrightarrow_{rot_2^{s_2}} \gamma_5$. Define $\gamma_2 = \gamma_5^{+t_2}$ where $0 < t_2 < 1 - Latest_1(\gamma_5)$. Existence of t_2 follows from the manner in which t_1 is chosen, definitions of the rules dec^θ and $rot_2^{s_2}$, and since γ_1 is a proper encoding. By the definitions it follows that γ_2 is a proper encoding with $sig(\gamma_2) = (s_2, m_1 - 1, m_2)$, and $\gamma_1 \longrightarrow_{T=t_1} \gamma_3 \longrightarrow_{dec^\theta} \gamma_4 \longrightarrow_{rot_2^{s_2}} \gamma_5 \longrightarrow_{T=t_2} \gamma_2$.
3. If $Latest_1(\gamma_1) = Latest_2(\gamma_1)$, but $m_2 = 0$. The proof is similar to the previous case. Here, we use the rule $rotz_2^{s_2}$ instead of the rule $rot_2^{s_2}$ and obtain $\gamma_1 \longrightarrow_{T=t_1} \gamma_3 \longrightarrow_{dec^\theta} \gamma_4 \longrightarrow_{rotz_2^{s_2}} \gamma_5 \longrightarrow_{T=t_2} \gamma_2$.

From Lemma 14 and Lemma 15 we get the following.

Lemma 16 *Consider an instruction $\theta = (s_1, c_1{-}{-}, s_2)$. Let γ_1 be a proper encoding with $sig(\gamma_1) = (s_1, m_1, m_2)$ and $m_1 > 0$. There is a proper encoding γ_2 such that $sig(\gamma_2) = (s_2, m_1 - 1, m_2)$ and*

$$\gamma_1 \overset{*}{\Longrightarrow}_{\{rot_2^{s_1}, rotz_2^{s_1}\}} \circ \Longrightarrow_{dec^\theta} \circ \overset{*}{\Longrightarrow}_{\{rot_2^{s_2}, rotz_2^{s_2}\}} \gamma_2$$

A similar result holds in case θ is of the form $(s_1, c_2{-}{-}, s_2)$.

The proof of the following lemma is similar to the proof of Lemma 16.

Lemma 17 *Consider an instruction $\theta = (s_1, c_1 = 0?, s_2)$. Let γ_1 be a proper encoding with $sig(\gamma_1) = (s_1, 0, m_2)$. Then there is a proper encoding γ_2 such that $sig(\gamma_2) = (s_2, 0, m_2)$ and*

$$\gamma_1 \overset{*}{\Longrightarrow}_{\{rot_2^{s_1}, rotz_2^{s_1}\}} \circ \Longrightarrow_{tst^\theta} \circ \overset{*}{\Longrightarrow}_{\{rot_2^{s_2}, rotz_2^{s_2}\}} \gamma_2$$

A similar result holds in case θ is of the form $(s_1, c_2 = 0?, s_2)$.

Proof of Lemma 8

To show that $\longrightarrow$ is monotonic with respect to $\preceq$, we show that both $\longrightarrow_r$ and $\longrightarrow_T$ are monotonic with respect to $\preceq$.

Monotonicity of $\longrightarrow_r$ is straightforward. So we only show the monotonicity of $\longrightarrow_T$. Given γ_1, γ_2 such that $\gamma_1 \preceq \gamma_2$, we show that $\gamma_1^{+t} \preceq \gamma_2^{+t}$.

- Given $p \in Q^{proc}$, $m'_1, \ldots, m'_K \in \{1, \ldots, max + 1\}$, the definition of $\longrightarrow_T$ and the fact that the clocks with values larger than max are identified, it follows that $\#(p, m'_1, \ldots, m'_K)(\gamma^{+t}) = \Sigma_{m_1,\ldots,m_K} \#(p, m_1, \ldots, m_K)(\gamma)$
  such that one of the following holds for each $k : 1 \leq k \leq K$.
  - if $m'_k \leq max$, then $m_k = m'_k - t$.
  - if $m'_k = max + 1$, then $max + 1 - t \leq m_k \leq max + 1$.
- Given $\gamma_1 \preceq \gamma_2$, for each $p \in Q^{proc}$, $m_1, \ldots, m_K \in \{0, \ldots, max + 1\}$, we have
  $$\#(p, m_1, \ldots, m_K)(\gamma_1) \leq \#(p, m_1, \ldots, m_K)(\gamma_2)$$
- Above two conditions imply that for each $p \in Q^{proc}$, $m'_1, \ldots, m'_K \in \{0, \ldots, max + 1\}$, we have
  $$\#(p, m'_1, \ldots, m'_K)(\gamma_1^{+t}) \leq \#(p, m'_1, \ldots, m'_K)(\gamma_2^{+t})$$
  i.e, $\gamma_1^{+t} \preceq \gamma_2^{+t}$.

Proof of Lemma 9

Given a configuration γ', we compute $minpre(\gamma')$ as follows. Since, $\longrightarrow = \longrightarrow_T \cup \bigcup_{r\in\mathcal{R}} \longrightarrow_r$, we compute $minpre(\gamma') = minpre_T(\gamma') \cup \bigcup_{r\in\mathcal{R}} minpre_r(\gamma')$ where $minpre_T(\gamma')$ characterizes the set of configurations from which we reach $\gamma' \uparrow$ through passage of time and $minpre_r(\gamma')$ does the same through execution of a rule r.

For a configuration $\gamma = (I, q, \mathcal{Q}, X)$, we define $minc(\gamma)$ to be the minimum value in the set $\{X_k(i) \mid 1 \leq k \leq K \ \wedge\ i \in I\}$, i.e, $minc(\gamma)$ denotes the minimum clock value which occurs in γ. We compute $minpre_T(\gamma)$ as follows.

$$minpre_T(\gamma) = \left\{\gamma^{-t} \mid 0 \leq t \leq minc(\gamma)\right\}$$

where γ^{-t} is defined in a manner similar to the definition of γ^{+t}.

Next, we show how to compute $minpre_r$. Given a configuration $\gamma' = \langle I', q_0', \mathcal{Q}', X' \rangle$ and a rule $r \in \mathcal{R}$ of the form

$$\begin{bmatrix} q_0 \\ \rightarrow \\ q_0' \end{bmatrix} \quad \begin{bmatrix} q_1 \\ g_1 \rightarrow R_1 \\ q_1' \end{bmatrix} \quad \cdots \quad \begin{bmatrix} q_n \\ g_n \rightarrow R_n \\ q_n' \end{bmatrix}$$

we compute a finite set of configurations Γ such that $minpre_r(\gamma') = min(\Gamma)$. We define Γ to be the set of all configurations of the form $\langle I, q_0, \mathcal{Q}, X \rangle$ such that there are three pairwise disjoint sets $changing$, $unchanged$ and $guarding$ and the following holds.

- $I' = changing \cup unchanged$,
- $I = changing \cup unchanged \cup guarding$, and
- there is a bijection $h : changing \cup guarding \rightarrow \{1, \ldots, n\}$, which satisfies the following conditions.
  1. $\mathcal{Q}(j) = q_{h(j)}$, and the guard $g_{h(j)}(X_1(j), \ldots, X_K(j))$ holds. Furthermore, $X(k)(j) \in \{0, \ldots, max+1\}$, for each $j \in (changing \cup guarding)$ and for each $k : 1 \leq k \leq K$.
  2. $\mathcal{Q}'(j) = q'_{h(j)}$ for $j \in changing$.
  3. $\mathcal{Q}'(j) = \mathcal{Q}(j)$ for $j \in unchanged$.
  4. For each $k : 1 \leq k \leq K$, $X'_k(j) = 0$ if $j \in changing$ and $x_k \in R_{h(j)}$.
  5. For each $k : 1 \leq k \leq K$ and for each j such that either $j \in changing$ and $x_k \notin R_{h(j)}$, or $j \in unchanged$, $X_k(j) = min(max + 1, X'_k(j))$.

Observe that

- The set $guarding$ of indices with size m can correspond to infinitely many sets of same size. We equate all of them modulo renaming. Therefore, the sets $changing, unchanged, guarding$ are finite and effectively constructible.
- The conditions on the controller states are implicitly included by our notation, which requires the controller states of γ and γ' to be the controller states of r.
- The correctness of the above algorithm can be shown in a similar manner to the proofs in [AJ03].