Closed, Open, and Robust Timed Networks

Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata

Dept. of Information Technology, Uppsala University, Sweden.
Email :{parosh,johannd,pritha}.it.uu.se

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 [AJ03] it was shown that checking safety properties is decidable in the case where each timed process is equipped with a single real-valued clock. In [ADM04], we showed that this is no longer possible if each timed process is equipped with at least two real-valued clocks. In this paper, first we study two syntactic subclasses of timed networks: *closed* and *open* timed networks. In closed timed networks, all clock constraints are non-strict, while in open timed networks, all clock constraints are strict (thus corresponds to syntactic removal of equality testing). We show that the problem becomes decidable for closed timed network, while it remains undecidable for open timed networks. We also consider *robust* semantics of timed networks by introducing timing fuzziness through semantic removal of equality testing. We show that the problem is undecidable both for closed and open timed networks under the robust semantics.

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 local clocks. Since a TN operates on an unbounded number of clocks, its behaviour cannot be captured by that of a timed automaton [AD94].

The paper [AJ03] showed 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] was given subject to the restriction that each timed process has a single clock. In [ADM04] we show undecidability of the problem for the case of *multi-clock* TNs, i.e., TNs where each timed process may have several clocks.

One complaint which has been raised against the model of timed automata, as defined in [AD94], is their excessive expressive power due to their ability to differentiate points in time with infinite precision (see e.g. [GHJ97,Pur98]). This makes algorithmic analysis hard in many cases. For instances, checking emptiness is PSPACE-complete, while checking universality is undecidable. Two classes of methods have been suggested to remedy this problem:

- The use of *digitization techniques* [HMP92]. The idea is to identify subclasses of timed automata for which verification problems can reduced from the dense time domain to the discrete one. This either yields speeding-up of the verification problem, or implies decidability of problems which are undecidable in the general case. A class of timed automata which allow digitization is *closed timed automata* [GHJ97,OW03b,OW03a] in which only non-strict clock constraints are allowed (of the form $x \leq 3$ or $x \geq 2$).
- To restrict the model of timed automata so that checking exact equality of clock values is prohibited. This restriction can be achieved syntactically through the use of *open timed automata* [GHJ97,OW03b,OW03a]. In an open automaton only strict clock constraints are allowed (of the form $x < 3$ or $x > 2$). The restriction can also be achieved semantically by considering a *robust semantics* [GHJ97,HR00,OW03a] where a computation is accepted to be valid if and only if neighbouring computations are also accepted.

The complaint about the excessive power of timed automata is equally valid in the case of timed networks. In this paper, we consider subclasses of timed networks based on similar restrictions to the one mentioned above.

First, we consider *closed timed networks* (CTNs) and *open timed networks* (OTNs). Using a similar idea to [OW03b], we show that digitization is applicable to CTNs. This reduces controller state reachability for CTNs to the same problem for discrete timed networks. The latter problem is shown to be decidable in [ADM04]. This result is of practical relevance since any timed network can be safely infinitesimally over-approximated by a closed timed network. Furthermore, we show undecidability of controller state reachability for OTNs. The undecidability result is shown through a reduction from the reachability problem for 2-counter machines. The undecidability result strengthens the result in [ADM04], in the sense that (i) it shows undecidability for a subclass of that in [ADM04]; and (ii) it uses an encoding which does not rely on using equality for clock values.

Finally, we show undecidability of controller state reachability for both CTNs and OTNs under the robust semantics. This is achieved by reducing the problem for OTNs under the standard semantics to the problem for both CTNs and OTNs under the robust semantics.

Outline: Section 2 gives the definition of timed networks. Section 3 details the decidability of the controller state reachability problem for closed timed networks. Section 4 sketches the undecidability proof of the problem for open timed networks. Finally, Section 5 shows the undecidability proof of the problem under robust semantics.

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 local 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

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

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.

Next, we mention two important syntactic subclasses of timed network.

Closed Timed Network A *closed timed network* is a timed network in which guarded commands in the rules may only contain a negation-free boolean combination of predicates of the form $k \leq x$ or $k \geq x$ where $x \in \{x_1, \ldots, x_K\}$.

Open Timed Network An *open timed network* is a timed network in which guarded commands in the rules may only contain a negation-free boolean combination of predicates of the form $k < x$ or $k > x$ where $x \in \{x_1, \ldots, x_K\}$.

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 $X_1(1) = 0.1, X_1(2) = 0.5, X_1(3) = 5.0, X_2(1) = 2.3, X_2(2) = 1.4, X_2(3) = 0.6$.

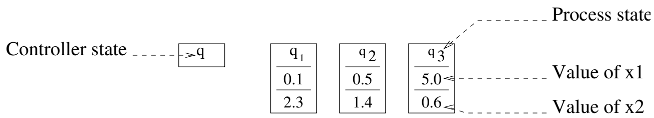

Fig. 1. Graphical representation of a configuration in a timed network with two clocks.

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 $\gamma' = (I, q', \mathcal{Q}', X')$. 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. We also say that $\pi = \gamma_{init} \longrightarrow \gamma_1 \longrightarrow \ldots \longrightarrow \gamma_n = \gamma$ is a γ_{init}-computation of TN.

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 properties (expressed as regular languages) can be translated into instances of the problem.

Discrete Timed Networks (DTNs) are timed networks in which the clocks assume values from the set of natural numbers and timed transitions take only discrete steps. A configuration of a DTN has same form as that of the TN, but the clocks have values which are natural numbers rather than real numbers. Furthermore, 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. We define DTN(K)-Reach in the obvious manner.

The following results are known for timed networks.

Theorem 1. *(1) TN(1)-Reach is decidable [AJ03]. (2) TN(2)-Reach is undecidable [ADM04]. (3) DTN(K)-Reach is decidable for every K [ADM04].*

3 Closed Timed Networks

In this section, we show that the controller state reachability problem for closed timed networks with K clocks (CTN(K) in short) is decidable.

We define *CTN(K)-Reach* in a manner similar to the definition of TN(K)-Reach. We show that

Theorem 2. *CTN(K)-Reach is decidable.*

From Theorem 1(3), we know that DTN(K)-Reach is decidable. To decide CTN(K)-Reach, we reduce the problem CTN(K)-Reach to the problem DTN(K)-Reach. Given a closed timed network $\mathcal{N}_1 = (Q, \mathcal{R})$, we shall consider a discrete timed network $\mathcal{N}_2 = (Q, \mathcal{R})$. Notice that $\mathcal{N}_1$ and $\mathcal{N}_2$ are syntactically identical. Given $\mathcal{N}_1, \mathcal{N}_2$, Theorem 2 follows from the following Lemma.

Lemma 1. *For any initial configuration γ_{init} and a controller state s_F, there is an γ_{init}-computation in $\mathcal{N}_1$ which leads to the final state s_F iff there is a γ_{init}-computation leading to the final state s_F in the derived $\mathcal{N}_2$.*

Rest of the section is devoted to proving Lemma 1.

First we recall the digitization technique introduced in [HMP92]. Let $t \in \mathbb{R}^+$ and let $0 \leq \epsilon < 1$ be real numbers. If $fract(t) < \epsilon$, let $[t]_\epsilon = \lfloor t \rfloor$, otherwise $[t]_\epsilon = \lceil t \rceil$. The $[.]_\epsilon$ operator therefore shifts the value of a real number t to the preceding or the following integer, depending on whether the fractional part of t is less than ϵ or not.

The direction from right to left is straightforward.

We give the proof for the other direction. Suppose $\gamma_{init} \xrightarrow{*} \gamma$, i.e, there is a γ_{init}-computation π given by $\gamma_{init} = \gamma_0 \longrightarrow_{T=t_0} \gamma_0' \longrightarrow_{r_0} \gamma_1 \longrightarrow_{T=t_1} \gamma_1' \longrightarrow_{r_1} \gamma_2 \ldots \longrightarrow_{T=t_{n-1}} \gamma_{n-1}' \longrightarrow_{r_{n-1}} \gamma_n = \gamma$ in $\mathcal{N}_1$. Let γ_j be of the form $(I, q_j, \mathcal{Q}_j, X_j)$.

Given any $\epsilon : 0 \leq \epsilon < 1$, we define a computation $\pi' = \mu_0 \longrightarrow_{T=t_0'} \mu_0' \longrightarrow_{r_0} \mu_1 \longrightarrow_{T=t_1'} \mu_1' \longrightarrow_{r_1} \mu_2 \ldots \longrightarrow_{r_{n-1}} \mu_n$ in $\mathcal{N}_2$ as follows.

Define $\mu_0 = \gamma_0$, and for $j : 1 \leq j \leq n$, define $\mu_j = \left(I, q_j, \mathcal{Q}_j, X_j'\right)$:

1. $X_j'(k)(i) = [t_0 + \cdots + t_{j-1}]_\epsilon - [t_0 + \cdots + t_l]_\epsilon$, where l is the largest natural number $< j$ such that the rule r_l resets the clock x_k.
2. $X_j'(k)(i) = [t_0 + \cdots + t_{j-1}]_\epsilon$ if the clock x_k is never reset.

We define μ_j' in a similar manner to γ_j'. Furthermore, we define $t_0' = [t_0]_\epsilon$ and $t_j' = [t_0 + \cdots + t_j]_\epsilon - [t_0 + \cdots + t_{j-1}]_\epsilon$ for $j > 0$.

We show that π' is a computation in $\mathcal{N}_2$:

- From the definition of μ_j, μ_j' and t_j' and the fact that $\gamma_j \longrightarrow_{T=t_j} \gamma_j'$, it is clear that $\mu_j \longrightarrow_{T=t_j'} \mu_j'$ for $j : 0 \leq j < n$.
- To show $\mu_j' \longrightarrow_{r_j} \mu_{j+1}'$, we conclude first that r_j is enabled from μ_j'. This follows from the definition of μ_j', the fact that r_j is enable from γ_j' and the fact that, given $p, q, \epsilon \in \mathbb{R}^{\geq 0}$ and $N \in \mathbb{N}$:
  - $p - q \leq N \implies [p]_\epsilon - [q]_\epsilon \leq N$.
  - $p - q \geq N \implies [p]_\epsilon - [q]_\epsilon \geq N$.

  From the definition of μ_j', μ_{j+1}', enabling of r_j from μ_j' and the fact that $\gamma_j \longrightarrow_{r_j} \gamma_{j+1}$, it is clear that $\mu_j' \longrightarrow_{r_j} \mu_{j+1}'$ for $j : 0 \leq j < n$.

Therefore π' is a computation in $\mathcal{N}_2$. Theorem 2 follows from this and the fact that each γ_j has the same controller state as μ_j.

4 Open Timed Networks

In this section, we strengthen the undecidability result of [ADM04] by showing undecidability of the controller state reachability problem for open timed networks (OTNs).

The controller state reachability problem for open timed networks (*OTN(K)-Reach*) is defined in the obvious manner.

In the rest of this section, we prove the following theorem.

Theorem 3. *$OTN(2)$-Reach is undecidable.*

2-Counter Machines First we recall the standard definition of counter machines operating on two counters 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 $\leadsto$ on the set of configurations, which is defined as usual using the standard interpretations of counter operations. We use $\overset{*}{\leadsto}$ to denote the reflexive transitive closure of $\leadsto$. In a similar manner to timed networks, we use $\beta \overset{*}{\leadsto} s$ to denote that there is a configuration $\beta' = (s', m_1', m_2')$ such that $\beta \overset{*}{\leadsto} \beta'$ 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 $\beta_{init} \overset{*}{\leadsto} s_F$. It is well-known that CM-Reach is undecidable [Min61].

4.1 Encoding of Configurations

We show undecidability of OTN(2)-Reach through a reduction from CM-Reach. Given a counter machine $\mathcal{C} = (S, \mathcal{I})$, we shall derive an *open* timed network $\mathcal{N}_{\mathcal{C}} = (Q_{\mathcal{C}}, \mathcal{R}_{\mathcal{C}})$ with two clocks. First 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
   - three states tmp_1^θ, tmp_2^θ, tmp_3^θ for each increment instruction $\theta \in \mathcal{I}$,
   - two states tmp_1^θ, tmp_2^θ for each zero-testing instruction $\theta \in \mathcal{I}$, and
   - eight states $tmp_{11}^s, tmp_{12}^s, tmp_{21}^s, tmp_{22}^s, tmp_{z11}^s, tmp_{z12}^s, tmp_{z21}^s, tmp_{z22}^s$ for each controller state $s \in S$.
   - three states $s_{init}^1, s_{init}^2, s_{init}^3$ (recall that s_{init} is the initial local state of $\mathcal{C}$). These three states are used as intermediate states in the initialization phase of the simulation (Section 4.2).

The set $Q_{\mathcal{C}}^{proc}$ contains three 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).
3. A temporary state fst_θ for each increment instruction θ. This state is used as an intermediate state in the simulation of incrementing instructions.

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 h 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 $j \in I \setminus \text{range}(h)$.
- $X_1(h(i)) < X_2(h(i-1))$ for each $i : 1 \leq i \leq m+1$.
- $X_2(h(i)) < X_1(h(i+2))$, for each $i : 0 \leq i \leq m-1$.
- $X_2(h(m+1)) < X_1(h(0)) < X_1(h(1))$.

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 last three conditions show how the processes which are part of a c_1-encoding are ordered as a list. The position of each process in the list is reflected by values of its clocks x_1 and x_2. More precisely, condition three says that except the first process, clock x_1 of each process in the list is strictly smaller than clock x_2 of the previous process. Condition four says that clock x_2 of the each process is less than clock x_1 of the second process to its right (except the last two processes). Finally the last condition states that clock x_2 of the last process is strictly less than the clock x_1 of the first process, which is again strictly less than the clock x_1 of the second process. We use $Val_1(\gamma)$ to denote the value m of a c_1-encoding γ.

Example 2. Figure 2(a) shows a c_1-encoding of value 2. Figure 2(b) shows a graphical representation of the ordering among clock values. In Section 4.2 we shall use such a graphical representation to explain the different steps in the simulation of $\mathcal{C}$. Each process is denoted by an edge whose end points are its two clocks and the arrow is at clock x_2. Such an edge is labelled by the current state of the process.

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:

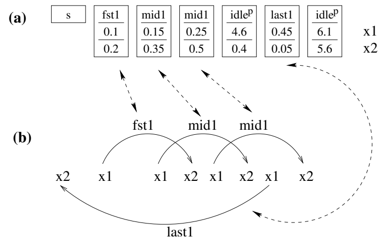

Fig. 2. (a) a c_1-encoding. (b) Graphical representation of ordering among clocks in c_1-encodings.

- $q = s$ 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 conditions:

- 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 4.2) 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).

4.2 Encoding of Transitions

Next we perform the second step in deriving the open 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 four rules in $\mathcal{R}_\mathcal{C}$, namely

$$inc_1^\theta : \begin{bmatrix} s_1 \\ \rightarrow \\ tmp_1^\theta \end{bmatrix} \begin{bmatrix} last_1 \\ 0 < x_2 \rightarrow \emptyset \\ last_1 \end{bmatrix} \begin{bmatrix} idle^p \\ tt \rightarrow \{x_2\} \\ fst_\theta \end{bmatrix}$$

$$inc_2^\theta : \begin{bmatrix} tmp_1^\theta \\ \rightarrow \\ tmp_2^\theta \end{bmatrix} \begin{bmatrix} fst_\theta \\ 0 < x_2 \rightarrow \emptyset \\ fst_\theta \end{bmatrix} \begin{bmatrix} fst_1 \\ tt \rightarrow \{x_1\} \\ fst_1 \end{bmatrix}$$

$$inc_3^\theta : \begin{bmatrix} tmp_2^\theta \\ \rightarrow \\ tmp_3^\theta \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_1 \rightarrow \emptyset \\ fst_1 \end{bmatrix} \begin{bmatrix} fst_\theta \\ tt \rightarrow \{x_1\} \\ fst_\theta \end{bmatrix}$$

$$inc_4^\theta : \begin{bmatrix} tmp_3^\theta \\ \rightarrow \\ s_2 \end{bmatrix} \begin{bmatrix} fst_\theta \\ 0 < x_1 \rightarrow \emptyset \\ fst_1 \end{bmatrix} \begin{bmatrix} fst_1 \\ tt \rightarrow \emptyset \\ mid_1 \end{bmatrix} \begin{bmatrix} last_1 \\ tt \rightarrow \{x_2\} \\ last_1 \end{bmatrix}$$

The total effect of the four rules is to increment the value of a c_1-encoding by adding one more process to the list.

The rule inc_1^θ picks a process in state $idle^p$ and changes its state to fst_θ. The new process will be placed first in the list. Furthermore, the rule resets clock x_2 of the new process in state fst_θ. The result of applying rule inc_1^θ on a c_1-encoding of value 0 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 will be placed second in the list. This is done in order to maintain the invariant that clock x_1 of each process (except the first process) is strictly less than the clock x_2 of the previous process (recall the definition of an encoding from Section 4.1). The result of applying rule inc_2^θ is shown in Figure 3(c).

Rule inc_3^θ resets clock x_1 of the process which is now in state fst_θ. This is done in order to maintain the invariant that clock x_1 of the first process is strictly less than the clock x_1 of the second process in the list (recall the definition of an encoding from Section 4.1). The result of applying rule inc_3^θ is shown in Figure 3(d).

Rule inc_4^θ resets clock x_2 of the process which is now in state $last_1$. This is done in order to maintain the invariant that clock x_2 of the last process is strictly less than the clock x_1 of the first process in the list (recall the definition of an encoding from Section 4.1). Furthermore, it changes the state of the process in fst_1 to mid_1, and thus makes this process second in the list. It also changes the state of the process in fst_θ to fst_1, and thus makes this process first in the list. The result of applying rule inc_4^θ is shown in Figure 3(e).

Some remarks about rules inc_1^θ,inc_2^θ,inc_3^θ and inc_4^θ:

- After execution of inc_1^θ (inc_2^θ, inc_3^θ resp.), the controller will be in state tmp_1^θ (tmp_2^θ, tmp_3^θ resp.) and therefore inc_2^θ (inc_3^θ, inc_4^θ resp.) is the only rule which may eventually be enabled after execution of inc_1^θ (inc_2^θ, inc_3^θ resp.).
- The guard $0 < x_2$ in the definition of inc_1^θ and inc_2^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 with the ones which are reset

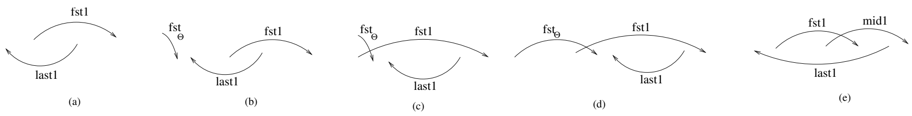

Fig. 3. Simulating $(s_1, c_1{+}{+}, s_2)$ on a c_1-encoding

during the application of inc_1^θ (inc_2^θ). The same reasoning applies to the guard $0 < x_1$ in the definition of the rules inc_3^θ and inc_4^θ. Similar guards exist in the rest of the rules described in this section.

- After application of inc_4^θ, the resulting encoding will not be proper, since a clock has just been reset and its value is 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 four 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

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

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_2 of the process (next last) in the state mid_1 is greater than one, but the clock x_1 of the process in $last_1$ is still less than 1. Some remarks about the rule dec^θ:

- Identifying the next last process 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 process in state mid_1.
- Waiting for clock x_1 of the next last process in the c_1-encoding to become greater than 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 the clocks of each process in c_1-encoding.

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 4 illustrates this scenario. We consider a proper encoding (shown in Fig-

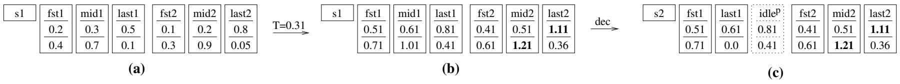

Fig. 4. Decrementing may result in an improper encoding.

ure 4(a)) with signature $(s_1, 1, 1)$ such that largest clock value (0.7) in c_1-encoding is less than the largest clock value (0.9) in c_2-encoding. In order to enable the rule dec^{θ}, we let time pass until clock x_2 of the process mid_1 (with largest clock value) becomes larger than one (shown in Figure 4(b)) while clock x_1 of the process is in state $last_1$ is still less than one. However, at this point of time, both clock x_2 of a process in state mid_2 and clock x_1 of the process in state $last_2$ have become larger than one (1.21 and 1.11 respectively). Therefore, after applying the dec^{θ}, we get an encoding (of value $(s_2, 0, 1)$) shown in Figure 4(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 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 three rules

$$rot^{s}{}_{2,1} : \begin{bmatrix} s \\ \rightarrow \\ tmp^{s}_{21} \end{bmatrix} \begin{bmatrix} fst_2 \\ tt \rightarrow \{x_1\} \\ fst_2 \end{bmatrix} \begin{bmatrix} mid_2 \\ x_2 > 1 \rightarrow \emptyset \\ mid_2 \end{bmatrix} \begin{bmatrix} last_2 \\ 0 < x_2 \wedge x_1 < 1 \rightarrow \emptyset \\ last_2 \end{bmatrix}$$

$$rot^{s}{}_{2,2} : \begin{bmatrix} tmp^{s}_{21} \\ \rightarrow \\ tmp^{s}_{22} \end{bmatrix} \begin{bmatrix} fst_2 \\ 0 < x_1 \rightarrow \emptyset \\ fst_2 \end{bmatrix} \begin{bmatrix} mid_2 \\ x_1 < 1 \wedge x_2 > 1 \rightarrow \emptyset \\ mid_2 \end{bmatrix} \begin{bmatrix} last_2 \\ x_1 < 1 \rightarrow \{x_1\} \\ last_2 \end{bmatrix}$$

$$rot^{s}{}_{2,3} : \begin{bmatrix} tmp^{s}_{22} \\ \rightarrow \\ s \end{bmatrix} \begin{bmatrix} fst_2 \\ tt \rightarrow \emptyset \\ mid_2 \end{bmatrix} \begin{bmatrix} last_2 \\ 0 < x_1 \rightarrow \emptyset \\ fst_2 \end{bmatrix} \begin{bmatrix} mid_2 \\ x_1 < 1 \wedge x_2 > 1 \rightarrow \{x_2\} \\ last_2 \end{bmatrix}$$

These rules do 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 the clocks of the processes in the c_1-encoding (see Figure 4). The role of $rot^{s}{}_{2,1}, rot^{s}{}_{2,2}, rot^{s}{}_{2,3}$ 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, these three rules together move the process which is in state $last_2$ and makes it first in the c_2-encoding. Figure 5 shows the effect of applying the rules to a c_2-encoding. This amounts to a rotation of

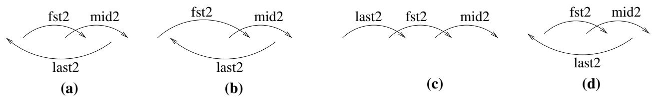

Fig. 5. Rotation of c_2-encoding of value 1.

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 the largest clock value in the c_1-encoding, the rotation stops and dec^θ can now be safely applied.

We illustrate the role of $rot^s{}_{2,1}$, $rot^s{}_{2,2}$ and $rot^s{}_{2,3}$ through Figure 6.

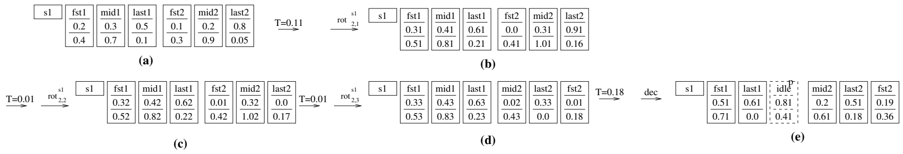

Fig. 6. Decrementing preceded by rotation

In a similar manner to Figure 4 we are interested in simulating a decrement instruction. However, instead of following the scenario of Figure 4, we now perform the following steps:

1. Wait for clock x_2 of the next last process in state mid_2 to become greater than one (1.01) and reset the clock x_1 of the process in state fst_2 by the rule $rot^{s_1}{}_{2,1}$ (shown in Figure 6(b)).
2. Now, $rot^{s_1}{}_{2,2}$ is the only rule we can apply after letting time progress by a positive amount (say, 0.01). The result (shown in Figure 6(c)) shows that the clock x_1 of the process in $last_2$ is reset.
3. Next, $rot^{s_1}{}_{2,3}$ is the only rule we can apply after letting time progress by a positive amount (say, 0.01) again. The result (shown in Figure 6(d)) is an encoding where largest clock value in c_1-encoding (0.83) is larger than the largest clock value in c_2-encoding (0.43). Notice that the process in state $last_2$ changes its states to fst_2. Also the processes in states fst_2 and mid_2 changes its states to mid_2 and $last_2$ respectively.

4. Wait (say be 0.18 again) until clock x_2 of the last process in state mid_1 becomes greater than 1 (1.01) while process $last_1$ has clock x_1 less than 1 (0.81). Then we apply the *decrementing* rule (shown in Figure 6(e)).

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

Also if, before applying dec^{θ}, the largest clock value in a c_1-encoding is same as that in a c_2-encoding, then we need to apply $rot^{s_2}{}_{2,1}, rot^{s_2}{}_{2,2}, rot^{s_2}{}_{2,3}$ in sequence 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 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.

For each state $s \in S$, the set $\mathcal{R}_{\mathcal{C}}$ also contains three more rules. $rotz^s_{2,1}, rotz^s_{2,1}$, and $rotz^s_{2,1}$. These rules are similar to the above rules and are used in connection with decrementing of counter c_1 if counter c_2 has value zero.

$$rotz^s_{2,1} : \begin{bmatrix} s \\ \rightarrow \\ tmp^s_{z21} \end{bmatrix} \begin{bmatrix} fst_2 \\ (x_2 > 1) \rightarrow \{x_1\} \\ fst_2 \end{bmatrix} \begin{bmatrix} last_2 \\ 0 < x_2 \wedge x_1 < 1 \rightarrow \emptyset \\ last_2 \end{bmatrix}$$

$$rotz^s_{2,2} : \begin{bmatrix} tmp^s_{z21} \\ \rightarrow \\ tmp^s_{z22} \end{bmatrix} \begin{bmatrix} fst_2 \\ x_2 > 1 \wedge 0 < x_1 \rightarrow \emptyset \\ fst_2 \end{bmatrix} \begin{bmatrix} last_2 \\ x_1 < 1 \rightarrow \{x_1\} \\ last_2 \end{bmatrix}$$

$$rotz^s_{2,3} : \begin{bmatrix} tmp^s_{z22} \\ \rightarrow \\ s \end{bmatrix} \begin{bmatrix} fst_2 \\ x_2 > 1 \rightarrow \{x_2\} \\ last_2 \end{bmatrix} \begin{bmatrix} last_2 \\ 0 < x_1 \wedge x_2 < 1 \rightarrow \emptyset \\ fst_2 \end{bmatrix}$$

The rules $rotz^s{}_{2,1}, rotz^s{}_{2,2}$ and $rotz^s{}_{2,3}$ have the same role as $rot^s{}_{2,1}$, $rot^s{}_{2,2}$ and $rot^s{}_{2,3}$ respectively, with the difference that they can be applied when the value of the c_2-encoding is zero. Figure 7 shows the effect of applying these rules to a c_2-encoding.

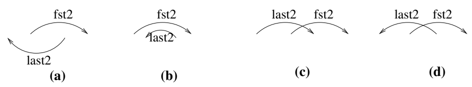

Fig. 7. Rotation of a c_2-encoding of value zero.

There are also similar rules $rot^s{}_{1,1}$, $rot^s{}_{1,2}$, $rot^s{}_{1,3}$, $rotz^s{}_{1,1}$, $rotz^s{}_{1,2}$ and $rotz^s{}_{1,3}$ 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 are rules in $\mathcal{R}_{\mathcal{C}}$, namely $tst^{\theta}_1, tst^{\theta}_2$ and tst^{θ}_3.

$$tst_1^\theta : \begin{bmatrix} s_1 \\ \rightarrow \\ tmp_1^\theta \end{bmatrix} \begin{bmatrix} \mathit{fst_1} \\ x_2 > 1 \rightarrow \{x_1\} \\ \mathit{fst_1} \end{bmatrix} \begin{bmatrix} \mathit{last_1} \\ 0 < x_2 \wedge x_1 < 1 \rightarrow \emptyset \\ \mathit{last_1} \end{bmatrix}$$

$$tst_2^\theta : \begin{bmatrix} tmp_1^\theta \\ \rightarrow \\ tmp_2^\theta \end{bmatrix} \begin{bmatrix} \mathit{fst_1} \\ (0 < x_1) \wedge (x_2 > 1) \rightarrow \emptyset \\ \mathit{fst_1} \end{bmatrix} \begin{bmatrix} \mathit{last_1} \\ x_1 < 1 \rightarrow \{x_1\} \\ \mathit{last_1} \end{bmatrix}$$

$$tst_3^\theta : \begin{bmatrix} tmp_2^\theta \\ \rightarrow \\ s_2 \end{bmatrix} \begin{bmatrix} \mathit{fst_1} \\ (x_2 > 1) \rightarrow \{x_2\} \\ \mathit{last_1} \end{bmatrix} \begin{bmatrix} \mathit{last_1} \\ 0 < x_1 \wedge x_2 < 1 \rightarrow \emptyset \\ \mathit{fst_1} \end{bmatrix}$$

The rule tst_1^θ checks that the value of the encoding is zero by testing that there are no processes in state $\mathit{mid_1}$. 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, and it has the value of clock x_2 greater than one. At this moment, we also check whether the last process has clock x_1 less than one. This check is done in a manner similar to the decrementing rule except that no process goes to $idle^p$. The rules also interchange the processes in states $\mathit{fst_1}$ and $\mathit{last_1}$ and reset the appropriate clocks to preserve the invariants of an encoding in a manner similar to the rotation rules described above. Sometimes before applying the rules for zero-testing, one has to apply the rotation rules according to the same scenarios explained for the *decrementing* rule.

This rule also resets clock x_1 of the process in state $\mathit{fst_1}$.

The rule tst_2^θ resets the clock x_1 of process in state $\mathit{last_1}$, so that it is less than clock x_1 of the process in state $\mathit{fst_1}$. This is done to preserve the invariant that clock x_1 of first process (currently the process in state $\mathit{last_1}$) in the list is less than clock x_1 of the second process (currently the process in state $\mathit{fst_1}$) in the list.

Notice that the clock x_2 of the process in state $\mathit{fst_1}$ is greater than one. So, the rule tst_3^θ resets the clock x_2 of the process in state $\mathit{fst_1}$ and changes the state of the process to $\mathit{last_1}$. It also changes the state of the process in $\mathit{last_1}$ to $\mathit{fst_1}$.

Thus, rule tst_1^θ together with rules tst_2^θ and tst_3^θ rotate the list corresponding to c_1-encoding by interchanging the states of the processes in states $\mathit{fst_1}$ and $\mathit{last_1}$ respectively, to yield a proper encoding. Notice the similarity between the rules tst_1^θ, tst_2^θ, tst_3^θ and $rotz_{1,1}^s$, $rotz_{1,2}^s$ $rotz_{1,3}^s$. The effect of the rules for zero-testing a c_2-encoding is thus captured by Figure 7.

Some remarks about the rules $tst_1^\theta, tst_2^\theta, tst_3^\theta$:

- The rule tst_1^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_2 of the process in state $\mathit{fst_1}$ cannot be greater than one, while the clock x_1 of the process in state $\mathit{last_1}$ is less than 1.
- After applying tst_1^θ (tst_2^θ), only the rule tst_2^θ (tst_3^θ) can be applied.
- Sometimes before applying the rule tst_1^θ, one has to apply 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 four rules.

$$init_1 : \begin{bmatrix} idle^c \\ \to \\ s^1_{init} \end{bmatrix} \begin{bmatrix} idle^p \\ tt \to \{x_2\} \\ fst_1 \end{bmatrix} \begin{bmatrix} idle^p \\ tt \to \{x_2\} \\ fst_2 \end{bmatrix}$$

$$init_2 : \begin{bmatrix} s^1_{init} \\ \to \\ s^2_{init} \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_2 \wedge x_2 < 1 \to \emptyset \\ fst_1 \end{bmatrix} \begin{bmatrix} idle^p \\ tt \to \{x_1\} \\ last_1 \end{bmatrix} \begin{bmatrix} idle^p \\ tt \to \{x_1\} \\ last_2 \end{bmatrix}$$

$$init_3 : \begin{bmatrix} s^2_{init} \\ \to \\ s^3_{init} \end{bmatrix} \begin{bmatrix} fst_1 \\ x_2 < 1 \to \{x_1\} \\ fst_1 \end{bmatrix} \begin{bmatrix} fst_2 \\ tt \to \{x_1\} \\ fst_2 \end{bmatrix} \begin{bmatrix} last_1 \\ 0 < x_1 \to \emptyset \\ last_1 \end{bmatrix}$$

$$init_4 : \begin{bmatrix} s^3_{init} \\ \to \\ s_{init} \end{bmatrix} \begin{bmatrix} fst_1 \\ (0 < x_1) \wedge (x_2 < 1) \to \emptyset \\ fst_1 \end{bmatrix} \begin{bmatrix} last_1 \\ tt \to \{x_2\} \\ last_1 \end{bmatrix} \begin{bmatrix} last_2 \\ tt \to \{x_2\} \\ last_2 \end{bmatrix}$$

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 $\mathcal{C}$. It can be shown that there are rules which pick four processes from state $idle^p$ and change their states to fst_1, $last_1$, fst_2, $last_2$ such that the resulting configuration is an encoding of β_{init}. The effect of the rules $init_1$, $init_2$, $init_3$ and $init_4$ are explained in Figure 8 (only the c_1-encoding is shown).

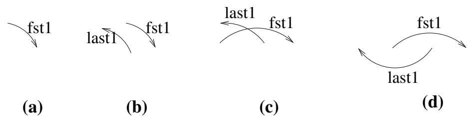

Fig. 8. Initialization.

Correctness 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 4.1 and Section 4.2. 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 4. *$\beta_{init} \overset{*}{\rightsquigarrow} s_F$ 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.

Lemma 2. *For any configuration $\gamma = (I, q, \mathcal{Q}, X)$ and initial configuration γ_{init} in $\mathcal{N}_\mathcal{C}$, if $\gamma_{init} \overset{*}{\longrightarrow} \gamma$ 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 $\beta_{init} \overset{*}{\rightsquigarrow} sig(\gamma)$.*

The only-if-direction follows from the following lemma.

Lemma 3. *If* $\beta_{init} \overset{n}{\rightsquigarrow} s_F$ *then* $\gamma_{init} \overset{*}{\longrightarrow} s_F$*, 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 of Lemma 2 and Lemma 3 reflect the informal arguments provided together with each rule in Section 4.2.

Theorem 3 follows from Theorem 4.

4.3 Proof of Theorem 4

In the proofs we need some definitions.

In our correctness proof of Theorem 4, 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 $\overset{*}{\rightsquigarrow} = \cup_n \overset{n}{\rightsquigarrow}$.

Let $\mathcal{N} = (Q, \mathcal{R})$ be an open 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 $\overset{*}{\Longrightarrow}$, $\overset{*}{\Longrightarrow}_r$ and $\overset{*}{\Longrightarrow}_{\mathtt{R}}$ to denote the reflexive transitive closure of the respective relations.

We introduce five new types of temporary encodings $c_1^{inc_1}$ *encoding*, $c_1^{inc_2}$ *encoding*, $c_1^{inc_3}$ *encoding*, $c_1^{rot_1}$ *encoding* and $c_1^{rot_2}$ *encoding* to describe the effect of the rules inc_1^{θ}, inc_2^{θ}, inc_3^{θ}, $rot^s{}_{11}$ and $rot^s{}_{12}$. For $m \geq 1$, a configuration $\gamma = (I, q, \mathcal{Q}, X)$ is said to be

- a $c_1^{inc_1}$ *encoding* of *value* m if there is an injection h from $\{0, \ldots, m+1\}$ to I such that the following conditions are satisfied
  - $\mathcal{Q}(h(0)) = fst_\theta$ where θ is an increment instruction in $\mathcal{C}$, $\mathcal{Q}(h(1)) = fst_1$, $\mathcal{Q}(h(i)) = mid_1$ for each $i : 2 \leq i \leq m$ and $\mathcal{Q}(h(m+1)) = last_1$.
  - $X_1(h(i)) < X_2(h(i-1))$ for each $i : 2 \leq i \leq m+1$.
  - $X_2(h(i)) < X_1(h(i+2))$, for each $i : 1 \leq i \leq m-1$.
  - $X_2(h(0)) < X_2(h(m+1)) < X_1(h(1)) < X_1(h(2))$.
- a $c_1^{inc_2}$ *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 as defined for a $c_1^{inc_1}$ *encoding*.
  - $X_1(h(i)) < X_2(h(i-1))$ for each $i : 1 \leq i \leq m+1$.
  - $X_2(h(i)) < X_1(h(i+2))$, for each $i : 1 \leq i \leq m-1$.
  - $X_2(h(0)) < X_2(h(m+1)) < X_1(h(2))$.
- a $c_1^{inc_3}$ *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 as defined for a $c_1^{inc_1}$ *encoding*.

- $X_1(h(i)) < X_2(h(i-1))$ for each $i : 1 \leq i \leq m+1$.
- $X_2(h(i)) < X_1(h((i+2)))$, for each $i : 1 \leq i \leq m-1$.
- $X_1(h(0)) < X_1(h(1))$.
- $X_2(h(0)) < X_2(h(m+1)) < X_1(h(2))$.

A graphical representation of $c_1^{inc_1}$ *encoding*, $c_1^{inc_2}$ *encoding* and $c_1^{inc_3}$ *encoding* is shown in Figure 3(b), 3(c), and 3(d) respectively.

For $m \geq 0$, a configuration $\gamma = (I, q, \mathcal{Q}, X)$ is said to be

- a $c_1^{rot_1}$ *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 as defined for a c_1-encoding.
  - $X_1(h(i)) < X_2(h(i-1))$ for each $i : 1 \leq i \leq m+1$.
  - $X_2(h(i)) < X_1(h((i+2)))$, for each $i : 0 \leq i \leq m-1$.
  - $X_1(h(0)) < X_2(h(m+1)) < X_1(h(1))$.
- a $c_1^{rot_2}$ *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 as defined for a $c_1^{rot_1}$ *encoding*.
  - $X_1(h(i)) < X_2(h(i-1))$ for each $i : 1 \leq i \leq m$.
  - $X_2(h(i)) < X_1(h((i+2)))$, for each $i : 0 \leq i \leq m-2$.
  - $X_1(h(m+1)) < X_1(h(0)) < X_2(h(m+1))$.
  - $X_2(h(m+1)) < X_1(h(1))$ and $X_2(h(m-1)) < X_2(h(m))$ if $m > 0$.
  - $X_2(h(m+1)) < X_2(h(0))$ if $m = 0$.

Figure 5 illustrates the rotation of a c_1-encoding graphically and shows a graphical representation of $c_1^{rot_1}$ *encoding* and $c_1^{rot_2}$ *encoding* in Figure 5(b) and 5(c) respectively.

In a similar manner to a c_1-encoding, we use $Val_1(\gamma)$ to denote the value m of a $c_1^{inc_1}$ *encoding* ($c_1^{inc_2}$ *encoding*, $c_1^{inc_3}$ *encoding*, $c_1^{rot_1}$ *encoding*, $c_1^{rot_2}$ *encoding*) γ.

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

- $q = tmp_1^\theta$ for some increment (of the form $\theta = (s_1, c_1{+}{+}, s_2)$).
- γ is both a c_2-encoding and a $c_1^{inc_1}$ *encoding*.

In such a case, we define $sig(\gamma)$ of γ to be the triple $\left(tmp_1^\theta, m_1, m_2\right)$, 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 after performing three discrete transitions (by rule inc_2^θ, inc_3^θ and inc_4^θ in sequence) in our simulation.

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

- $q = tmp_2^\theta$ for some increment (of the form $\theta = (s_1, c_1{+}{+}, s_2)$).
- γ is both a c_2-encoding and a $c_1^{inc_2}$ *encoding*.

We define the signature of a $c_1^{inc_2}$ *encoding* as in $c_1^{inc_1}$ *encoding*. Furthermore, we use $next(\gamma)$ to be (s_2, m_1, m_2). Intuitively, $next(\gamma)$ is the signature of the configuration which occurs next after performing two discrete transitions (by rules inc_3^θ and inc_4^θ in sequence) in our simulation.

Similarly, we define a *Type* $1c$ *semi-encoding*, it signature and the function $next$ for such a semi-encoding.

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

- $q = tmp_{11}^{s}$ for some controller state s in $\mathcal{C}$ or $q = tmp_{1}^{\theta}$ for some zero-testing instruction (of the form $\theta = (s_1, c_1?0, s_2)$).
- γ is both a c_2-encoding and a $c_1^{rot_1}$ *encoding*.

The signature for a $c_1^{rot_1}$ *encoding* is defined in a similar manner to a $c_1^{inc_1}$ *encoding*. Furthermore, the function $next(\gamma)$ is defined as (s, m_1, m_2) if $q = tmp_{11}^{s}$, $(s_2, 0, m_2)$ otherwise.

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

- $q = tmp_{12}^{s}$ for some controller state s in $\mathcal{C}$, or $q = tmp_{2}^{\theta}$ for some zero-testing instruction (of the form $\theta = (s_1, c_1?0, s_2)$).
- γ is both a c_2-encoding and a $c_1^{rot_2}$ *encoding*.

The signature and the function $next(\gamma)$ can be defined for a $c_1^{rot_2}$ *encoding* in a similar manner to a $c_1^{rot_1}$ *encoding*.

The notion of a semi-encoding of Type $1a, 1b, 1c, 1d, 1e$ 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. (Proper) semi-encodings of Type $2a, 2b, 2c, 2d, 2e$ are defined in a similar manner.

Proof of Lemma 2

Suppose that $\gamma_{init} \xrightarrow{*} \gamma$. If $\gamma_{init} \longrightarrow_T \gamma$ then the result follows immediately. Otherwise, $\gamma_{init} \overset{*}{\Longrightarrow} \gamma$, 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 $r_0 = init_1, r_1 = init_2, r_2 = init_3$ and therefore $q_1 = s_{init}^1$, $q_2 = s_{init}^2$ and $q_3 = s_{init}^3$. In other words, $\gamma_0, \ldots, \gamma_3$ satisfy the claim of the Lemma. Lemma 2 follows from the following property:

For each $4 \leq i \leq n$, it is the case that γ_i is either

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

This property is shown using an induction on i. For the base case we observe that, by definition of the rules, it follows that $r_3 = init_4$ and therefore $sig(\gamma_4) = \beta_{init}$. For the induction step, we observe that, for each $i : 4 \leq i < n$, 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 $1a$ semi-encoding with $sig(\gamma_{i+1}) = \left(tmp_1^{\theta}, m_1 + 1, m_2\right)$.

2. $r_i = inc_2^\theta$ for some $\theta = (s_1, c_1{+}{+}, s_2)$, γ_i is a Type $1a$ semi-encoding with $sig(\gamma_i) = \left(tmp_1^\theta, m_1, m_2\right)$, and γ_{i+1} is a Type $1b$ semi-encoding with $sig(\gamma_{i+1}) = \left(tmp_2^\theta, m_1, m_2\right)$.
3. $r_i = inc_3^\theta$ for some $\theta = (s_1, c_1{+}{+}, s_2)$, γ_i is a Type $1b$ semi-encoding with $sig(\gamma_i) = \left(tmp_2^\theta, m_1, m_2\right)$, and γ_{i+1} is a Type $1c$ semi-encoding with $sig(\gamma_{i+1}) = \left(tmp_3^\theta, m_1, m_2\right)$.
4. $r_i = inc_4^\theta$ for some $\theta = (s_1, c_1{+}{+}, s_2)$, γ_i is a Type $1c$ semi-encoding with $sig(\gamma_i) = \left(tmp_3^\theta, m_1, m_2\right)$, and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s_2, m_1, m_2)$.
5. $r_i = dec^\theta$ 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 $sig(\gamma_{i+1}) = (s_2, m_1 - 1, m_2)$.
6. $r_i = rot_{1,1}^s$ for some $s \in S$ and γ_i is an encoding with $sig(\gamma_i) = (s, m_1, m_2)$, and γ_{i+1} is a Type $1d$ semi-encoding with $sig(\gamma_{i+1}) = (tmp_{11}^s, m_1, m_2)$.
7. $r_i = rot_{1,2}^s$ for some $s \in S$ and γ_i is a Type $1d$ semi-encoding with $sig(\gamma_i) = (tmp_{11}^s, m_1, m_2)$, and γ_{i+1} is a Type $1e$ semi-encoding with $sig(\gamma_{i+1}) = (tmp_{12}^s, m_1, m_2)$.
8. $r_i = rot_{1,3}^s$ for some $s \in S$ and γ_i is a Type $1e$ semi-encoding with $sig(\gamma_i) = (tmp_{12}^s, m_1, m_2)$, and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s, m_1, m_2)$.
9. $r_i = rotz_{1,1}^s$ for some $s \in S$ and γ_i is an encoding with $sig(\gamma_i) = (s, 0, m_2)$, and γ_{i+1} is a Type $1d$ semi-encoding with $sig(\gamma_{i+1}) = (tmpz_{11}^s, 0, m_2)$.
10. $r_i = rotz_{1,2}^s$ for some $s \in S$ and γ_i is a Type $1d$ semi-encoding with $sig(\gamma_i) = (tmpz_{11}^s, 0, m_2)$, and γ_{i+1} is a Type $1e$ semi-encoding with $sig(\gamma_{i+1}) = (tmpz_{12}^s, 0, m_2)$.
11. $r_i = rotz_{1,3}^s$ for some $s \in S$ and γ_i is a Type $1e$ semi-encoding with $sig(\gamma_i) = (tmpz_{12}^s, 0, m_2)$, and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s, 0, m_2)$.
12. $r_i = tst_1^\theta$ for some $\theta = (s_1, c_1 = 0?, s_2)$, and γ_i is an encoding with $sig(\gamma_i) = (s_1, 0, m_2)$ is and γ_{i+1} is a Type $1d$ semi-encoding with $sig(\gamma_{i+1}) = \left(tmp_1^\theta, 0, m_2\right)$.
13. $r_i = tst_2^\theta$ for some $\theta = (s_1, c_1 = 0?, s_2)$, and γ_i is a Type $1d$ semi-encoding with $sig(\gamma_i) = \left(tmp_1^\theta, 0, m_2\right)$ is and γ_{i+1} is a Type $1e$ semi-encoding with $sig(\gamma_{i+1}) = \left(tmp_2^\theta, 0, m_2\right)$.
14. $r_i = tst_3^\theta$ for some $\theta = (s_1, c_1 = 0?, s_2)$, and γ_i is a Type $1e$ semi-encoding with $sig(\gamma_i) = \left(tmp_2^\theta, 0, m_2\right)$ is and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s_2, 0, m_2)$.
15. Similar cases corresponding to instructions which change counter c_2.

Proof of Lemma 3

To show Lemma 3 we use some definitions.

Let $\gamma = (I, q, \mathcal{Q}, X)$ be a (semi-)encoding. We define $Latest_1(\gamma) = max(X_2(i) : i \in I \wedge \mathcal{Q}(i) \in \{fst_1, mid_1\})$. 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. From the definition of encodings, clock x_2 of a process in state mid_2 (mid_1) can only have the highest clock value, if the value of the encoding is greater than 0. Otherwise, the clock x_2 of the process in fst_2 (fst_1) can have the largest clock value if value of the encoding is 0. We define $Latest_2(\gamma)$ in a similar manner, and define $Latest(\gamma) = max(Latest_1(\gamma), Latest_2(\gamma))$. We also define $Next2Latest_1(\gamma) = max(X_k(i) : k \in \{1,2\} \wedge i \in I \wedge \mathcal{Q}(i) \in \{fst_1, mid_1, last_1\} \wedge X_k(i) \neq Latest_1(\gamma))$. We define $Next2Latest_2(\gamma)$ in a similar manner and define $Next2Latest(\gamma) = max(Next2Latest_1(\gamma), Next2Latest_2(\gamma)$.

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 either $Latest_1(\gamma) < X_1(j)$ or $Latest_1(\gamma) < X_2(j)$. 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).

We define another temporary encoding: *almost proper* semi-c_1-encoding which is a semi-c_1 encoding with one process having index $i \in I$ s.t $\mathcal{Q}(i) = mid_1$ and $X_2(i) > 1$ while all processes with index $j \in I$, $\mathcal{Q}(j) \neq idle^p$ and $j \neq i$ satisfies $0 < X_1(j), X_2(j) < 1$. Similarly ,we define an almost proper semi-c_2-encoding. An almost proper semi-encoding of Type 1 is an almost proper semi-c_1-encoding of Type 1 and a proper c_2-encoding. Similarly we define an almost proper semi-encoding of Type 2.

Lemma 3 follows immediately from the following lemma.

Lemma 4. *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 $\gamma_2 \longrightarrow_{init_2} \gamma_3$. Define $\gamma_4 = \gamma_3^{+t_2}$ with $0 < t_2 < 1 - Latest(\gamma_3)$. We have $\gamma_3 \longrightarrow_{T=t_2} \gamma_4$. Rule $init_3$ is now enabled. Let γ_5 be such that $\gamma_4 \longrightarrow_{init_3} \gamma_5$. Define $\gamma_6 = \gamma_5^{+t_3}$ with $0 < t_3 < 1 - Latest(\gamma_5)$. We have $\gamma_5 \longrightarrow_{T=t_3} \gamma_6$. Rule $init_4$ is now enabled. Let γ_7 be such that $\gamma_6 \longrightarrow_{init_4} \gamma_7$. By definition of $init_4$, γ_7 is an encoding and $sig(\gamma_7) = \beta_{init}$. Let t_4 be such that $0 < t_4 < 1 - Latest(\gamma_7)$. t_4 exists by the definition of t_1, t_2, t_3, $init_1$, $init_2$, $init_3$ and $init_4$. Let $\gamma_8 = \gamma_7^{+t_4}$. γ_8 is a proper encoding with $sig(\gamma_8) = \beta_{init}$. Notice that the transitions $\longrightarrow_{init_1}$, $\longrightarrow_{init_2}$, $\longrightarrow_{init_3}$ and $\longrightarrow_{init_4}$ 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 5. *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 5 follows from Lemma 6, Lemma 7, Lemma 8, Lemma 9, Lemma 14, and Lemma 15:

- Lemma 6, Lemma 7, Lemma 8 and Lemma 9 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, followed by an application of rule inc_3^θ, followed by an application of rule inc_4^θ.
- Lemma 14 states that a *decrement* can be simulated by the rule dec^θ preceded and followed by a number of rotations. This lemma follows from Lemma 10, Lemma 11, Lemma 12, and Lemma 13.
- Lemma 15 deals with zero testing and is similar to Lemma 14.

The condition $|\gamma_1| \geq m_1+m_2+4$ in the claim of Lemma 5 is relevant only in Lemma 6, since this is the only case where the value of a counter is increased.

Lemma 6. *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 $1a$ such that $sig(\gamma_2) = \left(tmp_1^\theta, m_1+1, m_2\right)$ 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 $1a$ with $sig(\gamma_2) = \left(tmp_1^\theta, m_1+1, m_2\right)$ and $\gamma_1 \longrightarrow_{inc_1^\theta} \gamma_3 \longrightarrow_{T=t} \gamma_2$.

Lemma 7. *Consider an instruction $\theta = (s_1, c_1{+}{+}, s_2)$. Let γ_1 be a proper semi-encoding of Type $1a$ with $sig(\gamma_1) = \left(tmp_1^\theta, m_1, m_2\right)$ and $|\gamma_1| \geq m_1 + m_2 + 4$. There is a proper semi-encoding γ_2 of Type $1b$ such that $sig(\gamma_2) = \left(tmp_2^\theta, m_1, m_2\right)$ 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. The fact that γ_1 is a proper semi-encoding of Type $1a$ implies that inc_2^θ is enabled from γ_1, i.e., there is 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 a proper encoding. By the definitions it follows that γ_2 is a proper semi-encoding of Type $1b$ with $sig(\gamma_2) = \left(tmp_2^\theta, m_1+1, m_2\right)$ and $\gamma_1 \longrightarrow_{inc_2^\theta} \gamma_3 \longrightarrow_{T=t} \gamma_2$.

Lemma 8. *Consider an instruction $\theta = (s_1, c_1{+}{+}, s_2)$. Let γ_1 be a semi-encoding of Type $1b$ with $sig(\gamma_1) = \left(tmp_2^\theta, m_1, m_2\right)$. There is a proper semi-encoding γ_2 of Type $1c$ such that $sig(\gamma_2) = \left(tmp_3^\theta, m_1+1, m_2\right)$ and $\gamma_1 \Longrightarrow_{inc_3^\theta} \gamma_2$.*

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

Proof is similar to that of Lemma 7.

Lemma 9. *Consider an instruction* $\theta = (s_1, c_1{+}{+}, s_2)$*. Let* γ_1 *be a proper semi-encoding of Type* $1c$ *with* $sig(\gamma_1) = \left(tmp_3^\theta, m_1, m_2\right)$*. There is a proper encoding* γ_2 *such that* $sig(\gamma_2) = (s_2, m_1, m_2)$ *and* $\gamma_1 \Longrightarrow_{inc_3^\theta} \gamma_2$*.*

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 $1c$, it follows that inc_4^θ is enabled from γ_1, i.e., there is a configuration γ_3 with $\gamma_1 \longrightarrow_{inc_4^\theta} \gamma_3$. Define $\gamma_2 = \gamma_3^{+t}$ where $0 < t < 1 - Latest(\gamma_3)$. Such a t exists by definition of inc_4^θ and since γ_1 is proper semi-encoding of Type $1c$. By the definitions it follows that γ_2 is a proper encoding with $sig(\gamma_2) = (s_2, m_1, m_2)$ and $\gamma_1 \longrightarrow_{inc_4^\theta} \gamma_3 \longrightarrow_{T=t} \gamma_2$.

Lemma 10. *Let* γ_1 *be a proper encoding with* $Delay_1(\gamma_1) > 0$ *and* $sig(\gamma_1) = (s, m_1, m_2)$*. There is an almost proper semi-encoding of Type* $2d$*,* γ_2 *such that* $Delay_1(\gamma_2) = Delay_1(\gamma_1)$*,* $sig(\gamma_2) = (tmp_{21}^s, m_1, m_2)$*, and either* $\gamma_1 \Longrightarrow_{rot_{2,1}^s} \gamma_2$ *or* $\gamma_1 \Longrightarrow_{rotz_{2,1}^s} \gamma_2$*.*

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 t_1 such that $1 - Latest(\gamma_1) < t_1 < 1 - Next2Latest(\gamma_1)$ and $\gamma_3 = \gamma_1^{+t_1}$. t_1 exists due to the fact that γ_1 is a proper c_2-encoding and the fact that $m_2 > 0$. Now, $rot_{2,1}^s$ is enabled from γ_3 and there is a γ_4 with $\gamma_3 \longrightarrow_{rot_{2,1}^s} \gamma_4$. Define $\gamma_2 = \gamma_4^{+t_2}$ where $0 < t_2 < 1 - Next2Latest(\gamma_4)$. Existence of t_2 follows from the definition of the rule $rot_{2,1}^s$, and the fact that γ_1 is a proper c_2-encoding. By the definitions it follows that γ_2 is an almost proper semi-encoding of Type $2d$ with $Delay_1(\gamma_2) = Delay_1(\gamma_1)$, $sig(\gamma_2) = (tmp_{21}^s, m_1, m_2)$. and $\gamma_1 \longrightarrow_{T=t_1} \longrightarrow_{rot_{2,1}^s} \gamma_3 \longrightarrow_{T=t_2} \gamma_2$.

The case when $m_2 = 0$ is similar. Here we replace the rule $rot_{2,1}^s$ by the rule $rotz_{2,1}^s$, and obtain $\gamma_1 \longrightarrow_{T=t_1} \longrightarrow_{rotz_{2,1}^s} \gamma_3 \longrightarrow_{T=t_2} \gamma_2$.

Lemma 11. *Let* γ_1 *be an almost proper semi-encoding of Type* $2d$ *with* $Delay_1(\gamma_1) > 0$ *such that* $sig(\gamma_1) = (tmp_{21}^s, m_1, m_2)$*. There is an almost proper semi-encoding of Type* $2e$*,* γ_2 *such that* $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$*, (or* $Delay_1(\gamma_2) = Delay_1(\gamma_1)$*),* $sig(\gamma_1) = (tmp_{22}^s, m_1, m_2)$*, and either* $\gamma_1 \Longrightarrow_{rot_{2,2}^s} \gamma_2$ *or* $\gamma_1 \Longrightarrow_{rotz_{2,2}^s} \gamma_2$*.*

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 $m_2 > 0$. From the fact that γ_1 is an almost proper semi-encoding of Type $2d$ and the fact that $m_2 > 0$ it follows that $rot_{2,2}^s$ is enabled from γ_1, i.e., there is a γ_3 with $\gamma_1 \longrightarrow_{rot_{2,2}^s} \gamma_3$. Define $\gamma_2 = \gamma_3^{+t_1}$ where $0 < t_1 < 1 - Next2Latest(\gamma_3)$. Existence of t_1 follows from the definition of the rule $rot_{2,2}^s$, and the fact that γ_1 is an almost proper semi-encoding of Type $2d$. By the definitions it follows that γ_2 is an almost proper semi-encoding of Type $2e$ with

- $Delay_1(\gamma_2) = Delay_1(\gamma_1)$ if the clock x_1 of the process in $last_2$ of γ_1 is smaller than $Latest_1(\gamma_1)$,

– $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$ if the clock x_1 of the process in state *last₁* is larger than $Latest_1(\gamma_1)$, but the clock x_2 of process *last₂* in γ_1 is smaller than $Latest_1(\gamma_1)$.

and $sig(\gamma_2) = (tmp^s_{22}, m_1, m_2)$.

The case when $m_2 = 0$ is similar. Here we replace the rule $rot^s_{2,2}$ by the rule $rotz^s_{2,2}$, and obtain $\gamma_1 \longrightarrow_{rotz^s_{2,2}} \gamma_3 \longrightarrow_{T=t_2} \gamma_2$.

Lemma 12. *Let γ_1 be an almost proper semi-encoding of Type $2e$ with $Delay_1(\gamma_1) > 0$ such that $sig(\gamma_1) = (tmp^s_{22}, m_1, m_2)$. There is a proper encoding γ_2 such that $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$ (or, $Delay_1(\gamma_2) = Delay_1(\gamma_1)$) with $sig(\gamma_1) = (s, m_1, m_2)$, and either $\gamma_1 \Longrightarrow_{rot^s_{2,3}} \gamma_2$ or $\gamma_1 \Longrightarrow_{rotz^s_{2,3}} \gamma_2$.*

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 $m_2 > 0$. From the fact that γ_1 is an almost proper semi-encoding of Type $2e$ and the fact that $m_2 > 0$ it follows that $rot^s_{2,3}$ is enabled from γ_1, i.e., there is a γ_3 with $\gamma_1 \longrightarrow_{rot^s_{2,3}} \gamma_3$. Define $\gamma_2 = \gamma_3^{+t_1}$ where $0 < t_1 < 1 - Latest(\gamma_3)$. Existence of t_1 follows from the definition of the rule $rot^s_{2,3}$, and the fact that γ_1 is an almost proper semi-encoding of Type $2e$. By the definitions it follows that γ_2 is a proper encoding with

– $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$ (if $Latest_1(\gamma_1) > Next2Latest(\gamma_2)$),
– $Delay_1(\gamma_2) = Delay_1(\gamma_1)$ otherwise.

and $sig(\gamma_2) = (s, m_1, m_2)$.

The case when $m_2 = 0$ is similar. Here we replace the rule $rot^s_{2,3}$ by the rule $rotz^s_{2,3}$, and obtain $\gamma_1 \longrightarrow_{rotz^s_{2,3}} \gamma_3 \longrightarrow_{T=t_2} \gamma_2$.

Lemma 13. *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 $\gamma_1 \Longrightarrow_{dec^\theta} \circ \Longrightarrow_{rot^{s2}_{2,1}} \circ \Longrightarrow_{rot^{s2}_{2,2}} \circ \Longrightarrow_{rot^{s2}_{2,3}} \gamma_2$.*
3. *If $Latest_1(\gamma_1) = Latest_2(\gamma_1)$ and $m_2 = 0$ then $\gamma_1 \Longrightarrow_{dec^\theta} \circ \Longrightarrow_{rotz^{s2}_{2,1}} \circ \Longrightarrow_{rotz^{s2}_{2,2}} \circ \Longrightarrow_{rotz^{s2}_{2,3}} \gamma_2$.*

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 $1 - Latest_1(\gamma_1) < t_1 < 1 - Next2Latest(\gamma_1)$.

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 the largest clock value of the processes in the c_2-encoding has value larger than 1 in γ_4. Therefore, the rule $rot_{2,1}^{s_2}$ is enabled from γ_4, i.e., there is a γ_5 with $\gamma_4 \longrightarrow_{rot_{2,1}^{s_2}} \gamma_5$. Define $\gamma_6 = \gamma_5^{+t_2}$ where $0 < t_2 < 1 - Next2Latest(\gamma_5)$. Existence of t_2 follows from the manner in which t_1 is chosen, definitions of the rules dec^θ and $rot_{2,1}^{s_2}$, and since γ_1 is a proper encoding. By the definitions it follows that γ_6 is an almost proper semi-encoding of Type $2d$ with $sig(\gamma_6) = \left(tmp_{2,2}^{s_2}, m_1 - 1, m_2\right)$. Therefore, the rule $rot_{2,2}^{s_2}$ is enabled from γ_6, i.e., there is a γ_7 with $\gamma_6 \longrightarrow_{rot_{2,2}^{s_2}} \gamma_7$. Define $\gamma_8 = \gamma_7^{+t_3}$ where $0 < t_3 < 1 - Next2Latest(\gamma_7)$. Existence of t_3 follows from the manner in which t_1, t_2 are chosen, definitions of the rules $rot_{2,2}^{s_2}$, and since γ_6 is an almost proper semi-encoding of Type $2d$. By the definitions it follows that γ_8 is an almost proper semi-encoding of Type $2e$ with $sig(\gamma_8) = \left(tmp_{2,3}^{s_2}, m_1 - 1, m_2\right)$. Therefore, the rule $rot_{2,3}^{s_2}$ is enabled from γ_8, i.e., there is a γ_9 with $\gamma_8 \longrightarrow_{rot_{2,3}^{s_2}} \gamma_9$. Define $\gamma_2 = \gamma_9^{+t_4}$ where $0 < t_4 < 1 - Latest(\gamma_9)$. By the definitions it follows that γ_2 is a proper encoding with $sig(\gamma_2) = (s, m_1 - 1, m_2)$. and $\gamma_1 \longrightarrow_{T=t_1} \gamma_3 \longrightarrow_{dec^\theta} \gamma_4 \longrightarrow_{rot_{2,1}^{s_2}} \gamma_5 \longrightarrow_{T=t_2} \gamma_6 \longrightarrow_{rot_{2,2}^{s_2}} \gamma_7 \longrightarrow_{T=t_3} \gamma_8 \longrightarrow_{rot_{2,3}^{s_2}} \gamma_9 \longrightarrow_{T=t_4} \gamma_2$.
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 rules $rotz_{2,1}^{s_2}$, $rotz_{2,2}^{s_2}$ and $rotz_{2,3}^{s_2}2$ instead of the rules $rot_{2,1}^{s_2}$, $rot_{2,2}^{s_2}$ and $rot_{2,1}^{s_2}$ respectively and obtain $\gamma_1 \longrightarrow_{T=t_1} \gamma_3 \longrightarrow_{dec^\theta} \gamma_4 \longrightarrow_{rotz_2^{s_2}} \gamma_5 \longrightarrow_{T=t_2} \gamma_6 \longrightarrow_{rotz_{2,2}^{s_2}} \gamma_7 \longrightarrow_{T=t_3} \gamma_8 \longrightarrow_{rotz_{2,3}^{s_2}} \gamma_9 \longrightarrow_{T=t_4} \gamma_2$.

From Lemma 10, Lemma 11, Lemma 12 and Lemma 13 we get the following.

Lemma 14. *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*

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

where for a controller state s, we define $rot_2^s = \left\{rot_{2,1}^s, rot_{2,2}^s, rot_{2,3}^s\right\}$, and $rotz_2^s = \left\{rotz_{2,1}^s, rotz_{2,2}^s, rotz_{2,3}^s\right\}$.

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

Notice that the number of rotations are finite. This is due to the fact that there are only finite number of clocks which are larger than the largest clock of c_1-encoding and these clocks are reset by a finite number of rotations.

Lemma 15. *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

$$\gamma_1 \stackrel{*}{\Longrightarrow}_{rot_2^{s_1} \cup rotz_2^{s_1}} \circ \Longrightarrow_{tst_1^\theta} \circ \Longrightarrow_{tst_2^\theta} \circ \Longrightarrow_{tst_3^\theta} \circ \stackrel{*}{\Longrightarrow}_{rot_2^{s_2} \cup rotz_2^{s_2}} \gamma_2$$

where for a controller state s, rot_2^s, $rotz_2^s$ are as defined in Lemma 14.

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

The proof is similar to that for Lemma 14.

5 Robustness

In this section, we define the *robust* semantics of timed network. We show that the controller state reachability is undecidable for both open and closed robust timed networks. Notice that the problem is decidable for robust timed automata.

First, we define a metric d on the set of computations of a timed network as follows. Given two computations, $\pi = \gamma_{init} \longrightarrow_{T=t_1} \circ \longrightarrow_{r_1} \gamma_1 \cdots \gamma_{n-1} \longrightarrow_{T=t_n} \circ \longrightarrow_{r_n} \gamma_n$, and $\pi' = \gamma_{init} \longrightarrow_{T=t'_1} \circ \longrightarrow_{r'_1} \gamma'_1 \cdots \gamma'_{n-1} \longrightarrow_{T=t'_n} \circ \longrightarrow_{r'_n} \gamma'_n$, we let

- $d(\pi, \pi') = \infty$ if there is a $j : 1 \leq j \leq n$ such that $r_j \neq r'_j$.
- $d(\pi, \pi') = max \left\{ |t_j - t'_j| \;\; | \; 1 \leq j \leq n \right\}$.

As argued in [GHJ97], it can be also argued that any other 'reasonable' metric on computations of timed network yield the same topology as the metric d.

For the metric d, a computation π, and a positive real $\epsilon \in \mathbb{R}^+$, we define the d-tube around π of diameter ϵ to be the set $T(\pi, \epsilon) = \{\pi' \mid d(\pi, \pi') < \epsilon\}$ of all computations at a d-distance less than ϵ from π. A d-open set Op is a set of computations such that for all computations $\pi \in Op$, there is a positive real $\epsilon \in \mathbb{R}^+$ with $T(\pi, \epsilon) \subseteq Op$. Thus, if a d-open set contains a computation π, then it also contains all computations in some neighbourhood of π. From now on, we shall omit reference to d and use the terms 'open', 'closed', etc to mean d-open, d-closed and so on.

Let the set of all computations be called $Comp$. A set S of computations is closed if its complement $Comp - S$ is open. The closure $\overline{S}$ of a set S of computations is the least closed set containing S and the interior S^{int} is the greatest open set contained in S. The set S' of computations is dense in S iff $S \subseteq S'$.

Consider a timed network N. Let Γ_{init} be the (infinite) set of initial configurations (defined as in Section 2) and let S_F be a set of final controller states. We define the *language* $L(N)$ of N as a set of γ_{init}-computations π such that $\gamma_{init} \in \Gamma_{init}$ and π leads to some final controller state in S_F. The language L of a timed network induces a *robust language* $[L] = (\overline{L})^{int}$. The set $[L]$ represents the set of computations of N under the robust semantics. Notice that a computation π is in $[L]$ if and only if the set L of computations of TN is dense in some open neighbourhood of π.

Open Robust Reachability ($OTN(K)$-Robust-Reach)

Instance An open timed network $N = (Q, \mathcal{R})$ with K clocks and a set of controller states S_F.

Question Is $[L(N)] = \emptyset$?

For closed timed networks, we define the problem CTN(K)-Robust-Reach in a similar manner.

We extend the theory of robust semantics for timed automata [GHJ97,OW03a] in a straightforward manner for timed networks and get the following.

Lemma 16.

1. *For any timed network N, there is a closed timed network $\overline{N}$ having the same robust semantics, i.e,* $[L(N)] = [L(\overline{N})]$.
2. *For any timed network N, there is an open timed network N^{int} having the same robust semantics, i.e,* $[L(N)] = [L(N^{int})]$.
3. *For every open timed network N, $L(N) = \emptyset$ iff* $[L(N)] = \emptyset$.

The undecidability of OTN(2)-Reach (Theorem 3) means that it is undecidable whether $L(\text{TN}(2)) = \emptyset$. From this and Lemma 16(3), it follows that

Theorem 5. *OTN(2)-Robust-Reach is undecidable.*

Next, we show that the problem still remains undecidable even if we consider closed timed networks.

Consider an open timed network N with 2 clocks. We can derive a closed timed network $\overline{N}$ by Lemma 16(1) such that $[L(N)] = [L(\overline{N})]$. From Theorem 5, it is already undecidable whether $[L(N)] = \emptyset$. From these two facts, it follows that

Theorem 6. *CTN(2)-Robust-Reach is undecidable.*

Remark 1 OTN(1)-Robust-Reach is decidable due to decidability of TN(1)-Reach (Theorem 1(1)) and Lemma 16(3).

Remark 2 CTN(1)-Robust-Reach is also decidable due to decidability of OTN(1)-Robust-Reach and Lemma 16(2).

References

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

[ADM04] P. Abdulla, J. Deneux, and P. Mahata. Multi-clock timed networks. In *Proc. LICS' 04*, pages 345–354. IEEE Computer Society Press, 2004.

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

[GHJ97] V. Gupta, T. Henzinger, and R. Jagadesan. Robust timed automata. In *In Proc. of HART' 97*, volume 1201 of *Lecture Notes in Computer Science*, pages 331–345, 1997.

[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.

[HMP92] T.A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks. In *Proc. ICALP' 92*, volume 623 of *Lecture Notes in Computer Science*, pages 545–558, 1992.

[HR00] T. Henzinger and J. Raskin. Robust undecidability of timed and hybrid systems. In *Proc. of HSCC' 00*, volume 1790 of *Lecture Notes in Computer Science*, pages 145–159, 2000.

[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.

[OW03a] J. Ouaknine and J. Worrell. Revisiting digitization, robustness and decidability for timed automata. In *Proc. of LICS' 03*, pages 198–207. IEEE Computer Society Press, 2003.

[OW03b] J. Ouaknine and J. Worrell. Universality and language inclusion for open and closed timed automata. In *Proc. of HSCC' 03*, volume 2623 of *Lecture Notes in Computer Science*, 2003.

[Pur98] A. Puri. Dynamical properties of timed automata. In *Proc. FTRTFT'98*, volume 1486 of *Lecture Notes in Computer Science*, pages 210–227, 1998.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS'86*, pages 332–344. IEEE Computer Society Press, 1986.