Replace this file with `prentcsmacro.sty` for your meeting,
or with `entcsmacro.sty` for your meeting. Both can be
found at the ENTCS Macro Home Page.

Closed, Open, and Robust Timed Networks

Parosh Aziz Abdulla, Johann Deneux, and Pritha Mahata [1]

Dept. of Information Technology, Uppsala University, Sweden

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, we study two 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.

Key words: Model Checking, Reachability, Parameterized Timed Systems, Undecidability.

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.

[1] Email:{parosh,johannd,pritha}@it.uu.se

©2005 Published by Elsevier Science B. V.

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 may wonder what happens if we consider timed networks with clocks over dense-timed domain, but restrict their excessive expressive power due to their ability to differentiate points in time with infinite precision. In fact, this complaint of excessive expressive power has already been raised against the model of timed automata [AD94] by [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 allows 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. However, it is shown in [OW03a] that expressive powers of the timed automata under standard and robust semantics are incomparable.

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 ones 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 of clock values.

Finally, we show undecidability of controller state reachability for both OTNs and CTNs under the robust semantics. This is achieved by reducing the problem for OTNs under the standard semantics to the problem for both OTNs and CTNs under the robust semantics. The undecidability result for CTNs under robust semantics is surprising, since we already show that the problem is decidable for CTNs under standard semantics. However, it was already pointed out by [OW03a] that the robust semantics for timed automata is less tractable than its standard semantics. Undecidability of controller state reachability problem for CTNs show that the intractability of robust semantics for timed automata even prevails for timed networks.

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 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 $(Q, \Re)$, 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 $idle^p \in Q^{proc}$.

- $\mathfrak{R}$ 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.

Configurations A configuration γ of a timed network $(Q, \mathfrak{R})$ 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 2.1 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 $X_1(1) = 0.1, X_1(2) = 0.5, X_1(3) = 5.0, X_2(1) = 2.3, X_2(2) = 1.4, X_2(3) = 0.6$.

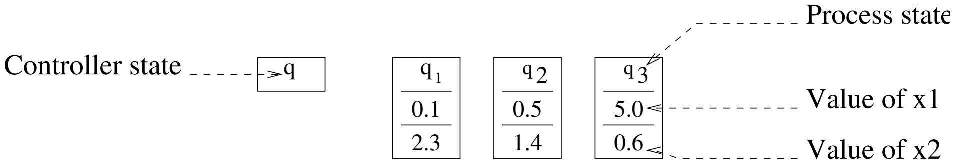

Fig. 1. Graphical representation of a configuration in a timed network with two clocks.

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 $\longrightarrow_{Disc}$, representing transitions induced by the rules, and a *timed* transition relation $\longrightarrow_{Timed}$ which represents passage of time.

The discrete relation $\longrightarrow_{Disc}$ is the union $\bigcup_{r\in\Re} \longrightarrow_r$, 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:

(i) $q = q_0$, $\mathcal{Q}(h(i)) = q_i$, and $g_i(X_1(h(i)), \ldots, X_K(h(i)))$ holds. That is, the rule r is enabled.

(ii) $q' = q'_0$, and $\mathcal{Q}'(h(i)) = q'_i$. The states are changed according to r.

(iii) 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.

(iv) $\mathcal{Q}'(j) = \mathcal{Q}(j)$ and $X'_k(j) = X_k(j)$, for $j \in I \setminus \mathrm{range}(h)$, 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 $\gamma \longrightarrow_{Timed} \gamma'$ to denote that $\gamma \longrightarrow_{T=t} \gamma'$ for some $t \in \mathbb{R}^{\geq 0}$.

We define $\longrightarrow$ to be $\longrightarrow_{Disc} \cup \longrightarrow_{Timed}$ and use $\stackrel{*}{\longrightarrow}$ 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 $\gamma \stackrel{*}{\longrightarrow} q$ to denote that there is a configuration γ' of the form $(I', q', \mathcal{Q}', X')$ such that $\gamma \stackrel{*}{\longrightarrow} \gamma'$ and $q' = q$.

We say that $\pi = \gamma_{init} \longrightarrow_{Timed}\longrightarrow_{Disc} \gamma_1 \longrightarrow_{Timed}\longrightarrow_{Disc} \cdots \longrightarrow_{Timed}\longrightarrow_{Disc} \gamma_n = \gamma$ is a γ_{init}-computation of TN.

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 $(Q, \Re)$ with K clocks and a controller state q_F.
Question Is there an initial configuration γ_{init} such that $\gamma_{init} \stackrel{*}{\longrightarrow} q_F$?

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 2.2

(i) *TN(1)-Reach is decidable [AJ03].*

(ii) *TN(2)-Reach is undecidable [ADM04].*

(iii) *DTN(K)-Reach is decidable [ADM04].*

3 Closed Timed Networks

In this section, we show that the controller state reachability problem for a subclass of timed networks is decidable.

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 $x \in \{x_1, \ldots, x_K\}$.
We define the controller state reachability problem for closed timed networks with K clocks (CTN(K)-Reach) in the obvious manner. We show that

Theorem 3.1 *CTN(K)-Reach is decidable.*

The rest of this section is devoted to the proof of Theorem 3.1.

First we recall the digitization technique introduced in [HMP92]. Let $\delta \in \mathbb{R}^+$ and let $0 \leq \varepsilon < 1$ be real numbers. If $fract(\delta) < \varepsilon$, let $[\delta]_\varepsilon = \lfloor \delta \rfloor$, otherwise $[\delta]_\varepsilon = \lceil \delta \rceil$. The $[.]_\varepsilon$ operator therefore shifts the value of a real number δ to the preceding or the following integer, depending on whether the fractional part of δ is less than ε or not.
From Theorem 2.2(iii), 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 $\mathcal{N}_1 = (Q, \Re)$, we shall consider a discrete timed network $\mathcal{N}_2 = (Q, \Re)$, which are syntactically identical. We show that 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 $\mathcal{N}_2$.
The direction from right to left is straightforward.
We give the proof for the other direction. Suppose $\gamma_{init} \xrightarrow{*} s_F$, i.e, there is a γ_{init}-computation π given by $\gamma_{init} = \gamma_0 \longrightarrow_{T=\delta_1} \gamma_1' \longrightarrow_{r_1} \gamma_1 \longrightarrow_{T=\delta_2} \gamma_2' \longrightarrow_{r_2}$

$\gamma_2 \ldots \longrightarrow_{T=\delta_n} \gamma'_n \longrightarrow_{r_n} \gamma_n = \gamma$ in $\mathcal{N}_1$, where $\delta_i \geq 0$ for $i : 1 \leq i \leq n$. Let γ_j be of the form $(I, q_j, \mathcal{Q}_j, X_j)$.

Given any $\varepsilon : 0 \leq \varepsilon < 1$, we define a χ_0-computation in $\mathcal{N}_2$ such that $\chi_0 \longrightarrow_{T=\delta'_1} \chi'_1 \longrightarrow_{r_1} \chi_1 \longrightarrow_{T=\delta'_2} \chi'_2 \longrightarrow_{r_2} \chi_2 \ldots \longrightarrow_{r_n} \chi_n$ as follows.

Define $\chi_0 = \gamma_0$, and for $j : 1 \leq j \leq n$, define $\chi_j = \left(I, q_j, \mathcal{Q}_j, X'_j\right)$:

(i) $X'_j(k)(i) = [\delta_1 + \cdots + \delta_j]_\varepsilon - [\delta_1 + \cdots + \delta_l]_\varepsilon$, where l is the largest natural number $\leq j$ such that the rule r_l resets the clock x_k.

(ii) $X'_j(k)(i) = [\delta_1 + \cdots + \delta_j]_\varepsilon$ if the clock x_k is never reset.

We define χ'_j in a similar manner to γ'_j. Furthermore, we define $\delta'_1 = [\delta_1]_\varepsilon$ and $\delta'_j = [\delta_1 + \cdots + \delta_j]_\varepsilon - [\delta_1 + \cdots + \delta_{j-1}]_\varepsilon$ for $j > 1$.

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

- From the definition of χ_{j-1}, χ'_j and δ'_j and the fact that $\gamma_{j-1} \longrightarrow_{T=\delta_j} \gamma'_j$, it is clear that $\chi_{j-1} \longrightarrow_{T=\delta'_j} \chi'_j$ for $j : 1 \leq j < n$. Notice that $\delta'_j \geq 0$.
- To show $\chi'_j \longrightarrow_{r_j} \chi_j$, we conclude first that r_j is enabled from χ'_j. This follows from the definition of χ'_j, the fact that r_j is enabled from γ'_j and the fact that, given $\xi_1, \xi_2, \varepsilon \in \mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$:
  · $\xi_1 - \xi_2 \leq k \implies [\xi_1]_\varepsilon - [\xi_2]_\varepsilon \leq k$.
  · $\xi_1 - \xi_2 \geq k \implies [\xi_1]_\varepsilon - [\xi_2]_\varepsilon \geq k$.
  From the definition of χ'_j, χ_j, enabling of the rule r_j from χ'_j and the fact that $\gamma'_j \longrightarrow_{r_j} \gamma_j$, it is clear that $\chi'_j \longrightarrow_{r_j} \chi_j$ for $j : 1 \leq j \leq n$.

Therefore π' is a computation in $\mathcal{N}_2$. Theorem 3.1 follows from this, the fact that each γ_j (γ'_j) has the same controller state as χ_j (χ'_j) and Theorem 2.2(iii).

Example 3.2 Figure 2 shows graphical representation of a computation in a closed timed network with two clocks starting from a configuration 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 $X_k(j) = 0$ for $j \in \{1,2,3\}$ and $k \in \{1,2\}$. Notice that r_0, r_1 and r_2 resets the clocks $(X_1(2), X_2(3))$, $(X_2(1))$ and $(X_1(1), X_2(1))$ respectively. Also, given $\varepsilon = 0.8$, we have $\delta'_0 = [1.5]_\varepsilon = 1$, $\delta'_1 = [1.8]_\varepsilon - [1.5]_\varepsilon = 2 - 1 = 1$ and $\delta'_2 = [3.9]_\varepsilon - [1.8]_\varepsilon = 4 - 2 = 2$. From this, it is easy to see the effect of the discrete transitions.

Example 3.3 As in Figure 2, Figure 3 shows graphical representation of a computation in a closed timed network with two clocks starting from the same configuration given by $(\{1,2,3\}, q, \mathcal{Q}, X)$ with $q, \mathcal{Q}, X$ as before. We also consider the same time lapses and the same set of rules. However, in this case $\varepsilon = 0.2$, we have $\delta'_0 = [1.5]_\varepsilon = 2$, $\delta'_1 = [1.8]_\varepsilon - [1.5]_\varepsilon = 2-2 = 0$ and $\delta'_2 = [3.9]_\varepsilon - [1.8]_\varepsilon = 4-2 = 2$. Again, it is easy to see the effect of the discrete transitions.

4 Open Timed Networks

In this section, we strengthen the undecidability result of Theorem 2.2(ii) by showing undecidability of the controller state reachability problem for a subclass of

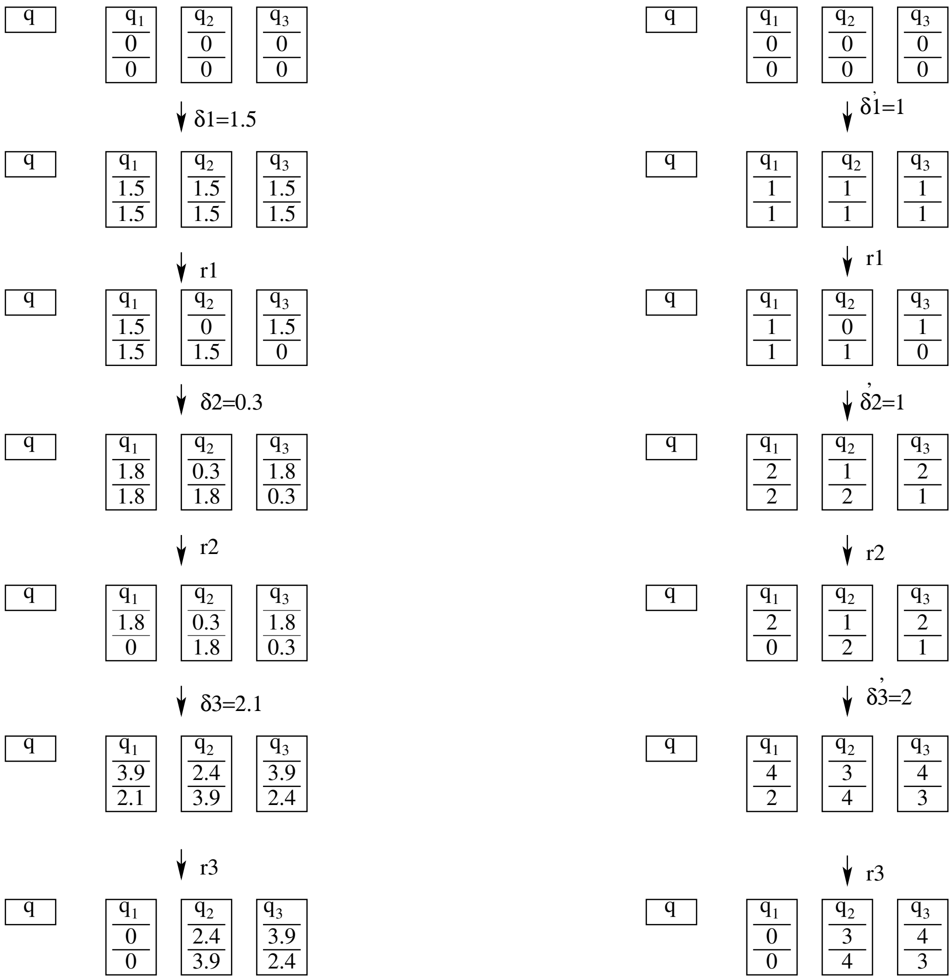

Fig. 2. Simulating a computation of a CTN by a computation in a DTN for $\varepsilon = 0.8$.

timed networks, namely *open timed networks*.

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 $x \in \{x_1, \ldots, x_K\}$.

We define the controller state reachability problem for open timed networks with K clocks (OTN(K)-Reach) in the obvious manner.

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

Theorem 4.1 *OTN*(2)*-Reach is undecidable.*

Notice that Theorem 4.1 implies Theorem 2.2(ii). However, the encoding of transitions in 2-counter machine is more involved for OTNs.

2-Counter Machines

First 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.

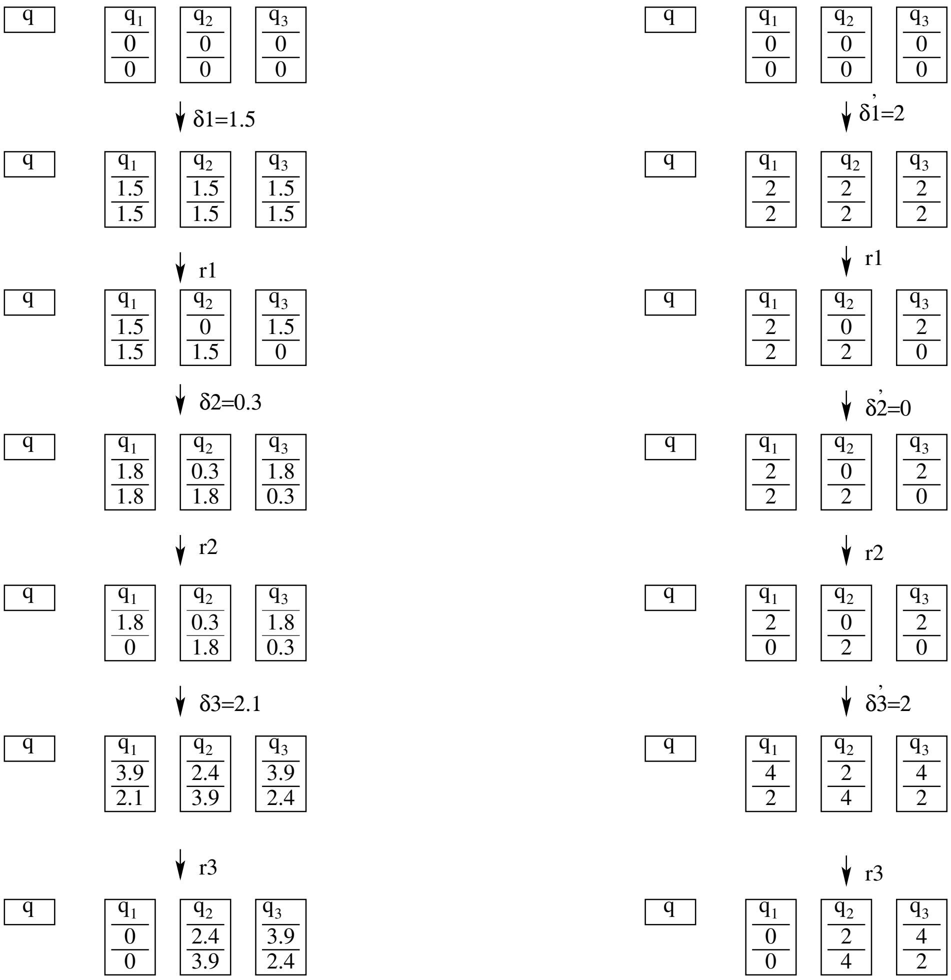

Fig. 3. Simulating a computation of a CTN by a computation in a DTN for $\varepsilon = 0.2$.

A *two-counter machine* C is a tuple $(S, s_{init}, \{c_1, c_2\}, \mathtt{I})$ where S is a finite set of *local states* with a distinguished *initial local state* $s_{init} \in S$, and $\mathtt{I}$ is a finite set of *instructions*. An instruction $\imath$ 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 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 4.2 *CM-Reach is undecidable.*

4.1 *Encoding of Configurations*

We show undecidability of OTN(2)-Reach through a reduction from CM-Reach. Given a counter machine $\mathsf{C} = (S, s_{init}, \{c_1, c_2\}, \mathtt{I})$, we shall derive an open timed network $\mathcal{O}_\mathsf{C} = (Q_\mathsf{C}, \Re_\mathsf{C})$ with two clocks. First we describe how to construct the set Q_C. Then, we describe how configurations of C are encoded as configurations of $\mathcal{O}_\mathsf{C}$. Finally, we introduce a special type of encodings, called *proper encodings*, which we use in our simulation of C.

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

(i) The initial controller state $idle^c$.

(ii) *Local states of* C: all members of S have copies in Q_C^{ctrl}.

(iii) *Temporary states*: the set Q_C^{ctrl} contains
- three states $tmp_1^\imath$, $tmp_2^\imath$, $tmp_3^\imath$ for each increment instruction $\imath \in \mathtt{I}$,
- two states $tmp_1^\imath$, $tmp_2^\imath$ for each zero-testing instruction $\imath \in \mathtt{I}$,
- four states tmp_{11}^s, tmp_{12}^s, tmp_{21}^s and tmp_{22}^s, for each controller state $s \in \mathsf{C}$, and
- three states $s_{init}^1, s_{init}^2, s_{init}^3$ (recall that s_{init} is the initial local state of C). These three states are used as intermediate states in the initialization phase of the simulation (Section 4.2).

The set Q_C^{proc} contains three types of states:

(i) The initial process state $idle^p$.

(ii) Six states fst_1, mid_1, $last_1$, fst_2, mid_2, and $last_2$, used for encoding the two counters (as described below).

(iii) A temporary state $fst_\imath$ for each increment instruction $\imath$. This state is used as an intermediate state in the simulation of incrementing instructions.

Encodings Each configuration β of C will be encoded by a set of configurations in $\mathcal{N}_\mathsf{C}$. 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 $j \in I \setminus \mathrm{range}(h)$.
- $X_1(h(i)) < X_2(h(i-1))$ 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 circular 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 of 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 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.

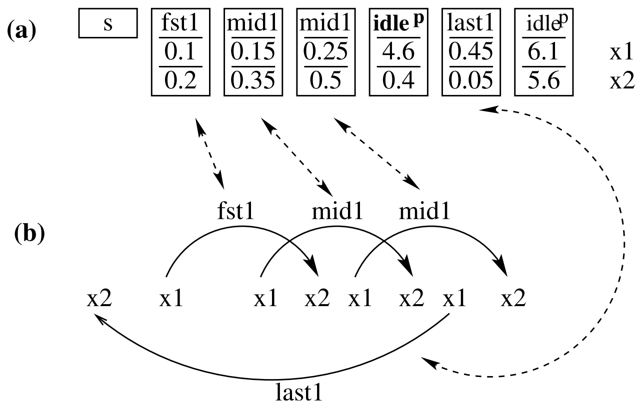

Fig. 4. (a) a c_1-encoding. (b) Graphical representation of ordering among clocks in c_1-encodings.

Example 4.3 Figure 4(a) shows a c_1-encoding of value 2. Figure 4(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 C. Each process is denoted by an edge whose end points are the values of two clocks of the process and the arrow is at clock x_2. Such an edge is labelled by the current state of the process. Clock values in the list of clocks in Figure 4(b) are strictly increasing from left to right.

A c_2-*encoding* and its value $Val_2(\gamma)$ are defined in a similar manner.
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 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 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 C.

Proper Encodings In our simulation of 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).

The difference of the above encoding with the encoding in the proof of Theroem 2.2(ii) is roughly as follows. The encoding in the proof of Theorem 2.2(ii) needs to have clock x_2 of each process (except the last one) exactly equal to clock x_1 of the next process. For OTNs, we need clock x_2 of each process to be strictly larger than clock x_1 of the next process (again, except the last process).

4.2 Encoding of Transitions

Next we perform the second step in deriving the open timed network $\mathcal{O}_{\mathsf{C}} = (Q_{\mathsf{C}}, \Re_{\mathsf{C}})$ from the counter machine $\mathsf{C} = (S, s_{init}, \{c_1, c_2\}, \mathtt{I})$. More precisely, we describe the set of rules $\Re_{\mathsf{C}}$. The set $\Re_{\mathsf{C}}$ contains the following rules:

Incrementing For each instruction $\imath = (s_1, c_1{+}{+}, s_2)$ in $\mathtt{I}$ there are four rules in $\Re_{\mathsf{C}}$, namely

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

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

$$inc_3^{\imath} : \begin{bmatrix} tmp_2^{\imath} \\ \rightarrow \\ tmp_3^{\imath} \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_1 \rightarrow \emptyset \\ mid_1 \end{bmatrix} \begin{bmatrix} fst_{\imath} \\ true \rightarrow \{x_1\} \\ fst_1 \end{bmatrix}$$

$$inc_4^\imath : \begin{bmatrix} tmp_3^\imath \\ \rightarrow \\ s_2 \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_1 \rightarrow \emptyset \\ fst_1 \end{bmatrix} \begin{bmatrix} last_1 \\ true \rightarrow \{x_2\} \\ last_1 \end{bmatrix}$$

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 $inc_1^\imath$ picks a process in state $idle^p$ and changes its state to $fst_\imath$. The new process will be placed first in the list. Furthermore, the rule resets clock x_2 of the new process in state $fst_\imath$. The result of applying rule $inc_1^\imath$ on a c_1-encoding of value 0 is shown in Figure 5(b).

Rule $inc_2^\imath$ 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 $inc_2^\imath$ is shown in Figure 5(c).

Rule $inc_3^\imath$ resets clock x_1 of the process which is now in state $fst_\imath$. It also changes the states of the processes in fst_1 and $fst_\imath$ to states mid_1 and fst_1 respectively. 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 $inc_3^\imath$ is shown in Figure 5(d). Rule $inc_4^\imath$ 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). The result of applying rule $inc_4^\imath$ is shown in Figure 5(e).

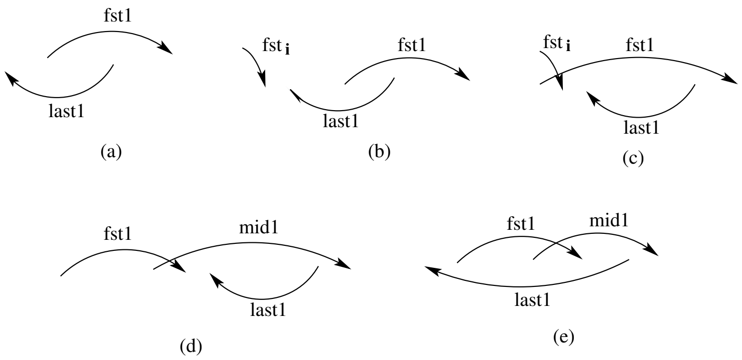

Fig. 5. Simulating $(s_1, c_1{+}{+}, s_2)$ on a c_1-encoding

Some remarks about rules $inc_1^\imath$,$inc_2^\imath$,$inc_3^\imath$ and $inc_4^\imath$:

- After execution of $inc_1^\imath$ ($inc_2^\imath$, $inc_3^\imath$ resp.), the controller will be in state $tmp_1^\imath$ ($tmp_2^\imath$, $tmp_3^\imath$ resp.) and therefore $inc_2^\imath$ ($inc_3^\imath$, $inc_4^\imath$ resp.) is the only rule which may eventually be enabled after execution of $inc_1^\imath$ ($inc_2^\imath$, $inc_3^\imath$ resp.).
- The guard $0 < x_2$ in the definition of $inc_1^\imath$ and $inc_2^\imath$ 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 during the application of $inc_1^\imath$ ($inc_2^\imath$). The same reasoning applies

to the guard $0 < x_1$ in the definition of the rules $inc^{\imath}_3$ and $inc^{\imath}_4$. Similar guards exist in the rest of the rules described in this section.

- After application of $inc^{\imath}_4$, the resulting encoding will not be proper. We can re-create a proper encoding by letting time pass through a timed transition.

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 $\imath = (s_1, c_1{-}{-}, s_2)$ in $\mathtt{I}$ there is a rule in $\Re_{\mathsf{C}}$, namely

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

The rule $dec^{\imath}$ 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. Figure 6 shows the effect of applying the rule to a c_1-encoding of value 2. Some remarks about the rule $dec^{\imath}$:

Fig. 6. Simulating $(s_1, c_1{-}{-}, s_2)$ on a c_1-encoding

- 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 $dec^{\imath}$, the value of such clocks will be greater than one, and therefore the resulting configuration will not be a *proper* encoding.

  Figure 7 illustrates this scenario. We consider a proper encoding (shown in Figure 7(a)) with signature $(s_1, 1, 1)$ such that largest clock value (0.7) in c_1-

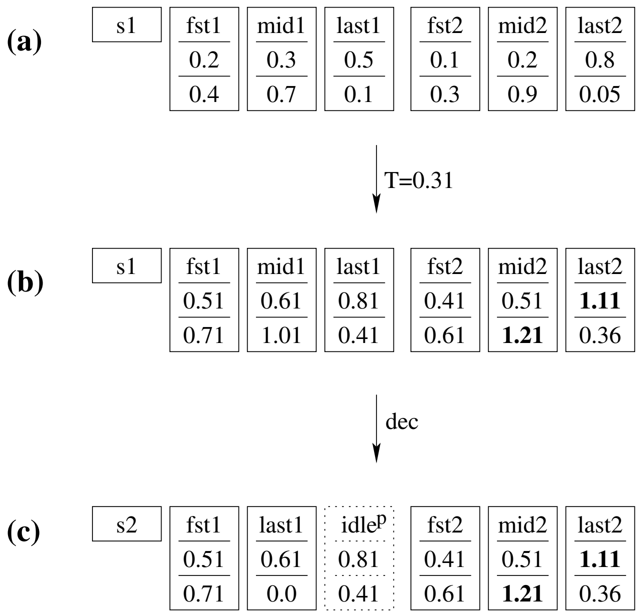

Fig. 7. Decrementing may result in an improper encoding.

encoding is less than the largest clock value (0.9) in c_2-encoding. In order to enable the rule $dec^\imath$, we let time pass until clock x_2 of the process mid_1 (with largest clock value) becomes larger than one (shown in Figure 7(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 $dec^\imath$, we get an encoding (of value $(s_2, 0, 1)$) shown in Figure 7(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 $dec^\imath$ 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 $\Re_C$ contains the following four rules

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

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

$$rot^s_{2,3}: \begin{bmatrix} tmp^s_{21} \\ \to \\ tmp^s_{22} \end{bmatrix} \begin{bmatrix} fst_2 \\ (0 < x_1) \wedge (1 < x_2) \to \emptyset \\ last_2 \end{bmatrix} \begin{bmatrix} last_2 \\ x_1 < 1 \to \{x_1\} \\ fst_2 \end{bmatrix}$$

$$rot^s{}_{2,4}: \begin{bmatrix} tmp^s_{22} \\ \to \\ s \end{bmatrix} \begin{bmatrix} fst_2 \\ 0 < x_1 \to \emptyset \\ fst_2 \end{bmatrix} \begin{bmatrix} last_2 \\ true \to \{x_2\} \\ last_2 \end{bmatrix}$$

These rules do not correspond to any instruction in C; nor does it change the signature of the encoding. In simulating C, we use the rotation rules in connection with decrementing. Recall that if $\imath = (s_1, c_1--, s_2)$ then applying a rule $dec^\imath$ 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. The role of $rot^s{}_{2,1}, rot^s{}_{2,2}, rot^s{}_{2,4}$ then is to decrement clock values of processes which are part of a c_2-encoding of positive value while preserving the signature of the whole encoding. More precisely,

- $rot^s_{2,1}$ resets the clock x_1 of the process in state fst_2. This process will be made second in the list by the next executed rule, i.e. $rot^s_{2,2}$.
- $rot^s_{2,2}$ resets the clock x_1 of the process in $last_2$ and makes this process first in the list. This rule also changes the state of the next last process to $last_1$ and the state of the second process (previously in fst_1) to mid_1. The identification of the next last process is the same as that in case of decrementing. The explanation of $rot^s_{2,3}$ is similar, but $rot^s_{2,3}$ is only applied if the c_2 encoding has value 0.
- $rot^s_{2,4}$ resets the clock x_2 of the process currently in state $last_2$.

This again amounts to a rotation of the list corresponding to the c_2-encoding of non-zero positive value. Figures 8(b), (c) and (d) graphically show the effect of applying these rules to a c_2-encoding of value 1 in Figure 8(a). After the last step, we can perform a timed transition and obtain a proper encoding.

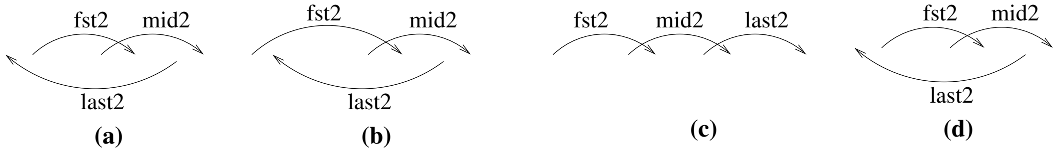

Fig. 8. Rotation of c_2-encoding of value 1.

This amounts to a rotation of 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 $dec^\imath$ can now be safely applied. We illustrate the role of $rot^s{}_{2,1}$, $rot^s{}_{2,2}$ and $rot^s{}_{2,4}$ through Figure 9. Also if, before applying $dec^\imath$, the largest clock value in a c_1-encoding is same as that in a c_2-encoding, then we need to apply $rot^{s_2}{}_{2,1}, rot^{s_2}{}_{2,2}, rot^{s_2}{}_{2,4}$ in sequence once more after decrementing (this scenario does not occur in Figure 9, but is considered in the correctness proof).

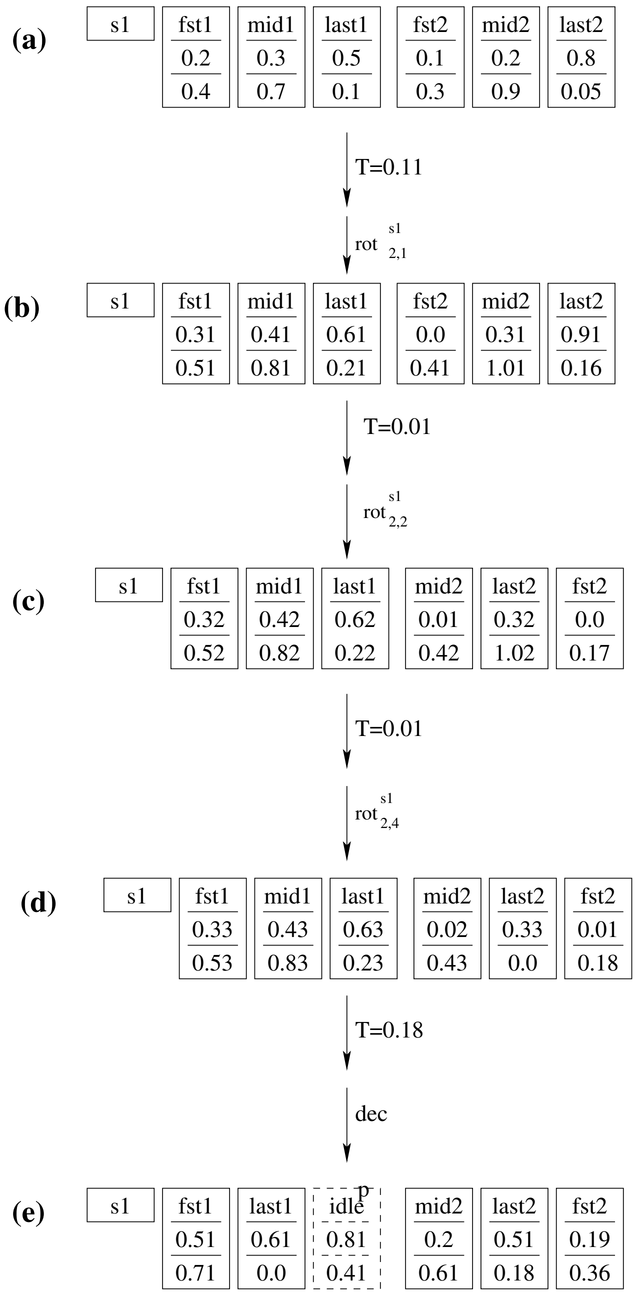

Fig. 9. Decrementing preceded by rotation

The rule $rot^s{}_{2,3}$ is used instead of the rule $rot^s{}_{2,2}$ when we use the rotation of a c_2-encoding of value 0.

There are also similar rules $rot^s{}_{1,1}$, $rot^s{}_{1,2}$, $rot^s{}_{1,3}$, and $rot^s{}_{1,4}$, which are used to rotate a c_1-encoding and which are used in connection with rules of the form $dec^\imath$ with $\imath = (s_1, c_2{-}{-}, s_2)$.

Zero Testing For each instruction $\imath = (s_1, c_1 = 0?, s_2)$ in $\mathtt{I}$ there are rules in $\Re_{\mathsf{C}}$, namely $tst_1^\imath$, $tst_2^\imath$ and $tst_3^\imath$. These three rules check 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 in a manner similar to the decrementing rule.
For each instruction $\imath = (s_1, c_1 = 0?, s_2)$ in C there are three rules in $\Re_{\mathsf{C}}$, namely

$$tst_1^\imath : \begin{bmatrix} s_1 \\ \rightarrow \\ tmp_1^\imath \end{bmatrix} \begin{bmatrix} fst_1 \\ true \rightarrow \{x_1\} \\ fst_1 \end{bmatrix} \begin{bmatrix} last_1 \\ 0 < x_2 \rightarrow \emptyset \\ last_1 \end{bmatrix}$$

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

$$tst_3^\imath : \begin{bmatrix} tmp_2^\imath \\ \rightarrow \\ s_2 \end{bmatrix} \begin{bmatrix} fst_1 \\ 0 < x_1 \rightarrow \emptyset \\ fst_1 \end{bmatrix} \begin{bmatrix} last_1 \\ true \rightarrow \{x_2\} \\ last_1 \end{bmatrix}$$

The rules interchange the processes in states fst_1 and $last_1$ and reset the appropriate clocks to preserve the invariants of an encoding in a manner similar to the rotation rules described above. The explanations of the rules $tst_1^\imath$, $tst_2^\imath$ and $tst_3^\imath$ are in fact, similar to those for the rules $rot_{1,1}^s$, $rot_{1,3}^s$ and $rot_{1,4}^s$ respectively. 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.
Figure 10 shows the effect of applying the rule to a c_1-encoding of value 0.

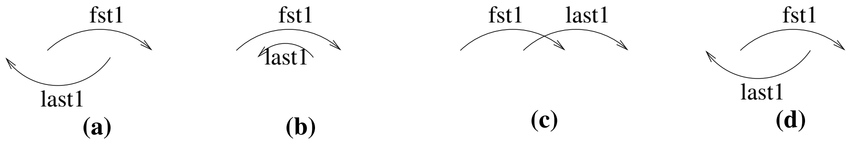

Fig. 10. Simulating $(s_1, c_1?0, s_2)$ on a c_1-encoding of value zero.

Also, there are similar rules in $\Re_{\mathsf{C}}$ for each instruction of the form $\imath = (s_1, c_2 = 0?, s_2)$.

Initialization The initial phase consists of the following four rules.

$$init_1 : \begin{bmatrix} idle^c \\ \rightarrow \\ s_{init}^1 \end{bmatrix} \begin{bmatrix} idle^p \\ true \rightarrow \{x_2\} \\ fst_1 \end{bmatrix} \begin{bmatrix} idle^p \\ true \rightarrow \{x_2\} \\ fst_2 \end{bmatrix}$$

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

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

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

The role of the initialization rules is to bring $\mathcal{O}_{\mathsf{C}}$ 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 C.

The first rule $init_1$ takes the controller into the temporary state s^1_{init}. It also picks two idle processes to be the first processes in the c_1-encoding and c_2-encoding (each with value zero). Clock x_2 of both the processes are reset. The second rule $init_2$ changes the controller state to s^2_{init} and picks two more processes to be the last processes in the c_1-encoding and c_2-encoding. This rule is enabled if the clock x_2 of the process fst_1 is strictly larger than 0. Also, clock x_1 of both the new processes are reset. The third rule $init_3$ is enabled when the clock x_1 of the process $last_1$ is greater than 0. and it changes the controller state to s^3_{init}. It also resets the clock x_1 of the processes in state fst_1 and fst_2 respectively. The fourth rule $init_4$ changes the controller state to s_{init}. It also resets the clock x_2 of the processes in state $last_1$ and $last_2$ respectively and completes the creation of the c_1-encoding and c_2-encoding of value zero. Finally, applying a timed transition yields a proper encoding. The effect of the rules $init_1$, $init_2$, $init_3$ and $init_4$ are illustrated through Figure 11 (only the c_1-encoding is shown).

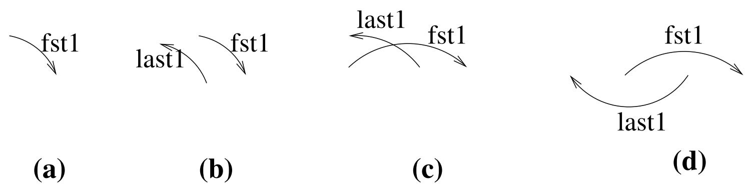

Fig. 11. Initialization. Figure (d) shows a graphical representation of a c_1-encoding of value 0.

Correctness

Let $\mathsf{C} = (S, s_{init}, \{c_1, c_2\}, \mathtt{I})$ be a counter machine and let $\mathcal{O}_{\mathsf{C}} = (Q_{\mathsf{C}}, \Re_{\mathsf{C}})$ be an open timed network derived from C as described in Section 4.1 and Section 4.2. Let $\rightsquigarrow$ and $\longrightarrow$ be the transition relations induced by C and $\mathcal{O}_{\mathsf{C}}$ respectively. Also, let $\overset{*}{\rightsquigarrow}$ and $\overset{*}{\longrightarrow}$ be their respective reflexive, transitive closures.

If s_F is a control state in C then the following holds

Theorem 4.4 *$\beta_{init} \overset{*}{\rightsquigarrow} s_F$ iff $\gamma_{init} \xrightarrow{*} s_F$ for some initial configuration γ_{init} of $\mathcal{O}_{\mathsf{C}}$.*

We show the proof of Theorem 4.4 in Appendix. Theorem 4.1 directly follows from Theorem 4.4.

5 Robust Timed Networks

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

First we define a *timed event* to be a pair (ξ, r) where ξ is the timestamp of the rule $r \in \Re$. A *timed trace* is a finite sequence of timed events with non-decreasing timestamps.

Given a computation $\gamma_{init} \longrightarrow_{T=\delta_1} \longrightarrow_{r_1} \gamma_1 \ldots \gamma_{n-1} \longrightarrow_{T=\delta_n} \longrightarrow_{r_n} \gamma_n$ of a timed network, there is an associated timed trace $tt(\pi) = \langle(\xi_1, r_1), \ldots, (\xi_n, r_n)\rangle$ where $\xi_1 = \delta_1$ and $\xi_i = \Sigma_{1 \leq j \leq i}\delta_j$. 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 consisting of timed traces $tt(\pi)$ where π is a γ_{init}-computation with $\gamma_{init} \in \Gamma_{init}$ and π leads to some final controller state in S_F.

Next we define a metric D on the set of all timed traces of a timed network as follows. Given two timed traces $w = \langle(\xi_1, r_1), \ldots, (\xi_n, r_n)\rangle$ and $w' = \langle(\xi'_1, r'_1), \ldots, (\xi'_n, r'_n)\rangle$, let

- $D(w, w') = \infty$ if there is a $j : 1 \leq j \leq n$ such that $r_j \neq r'_j$.
- $D(w, w') = max\left\{|\xi_j - \xi'_j| \ : \ 1 \leq j \leq n\right\}$.

As argued in [GHJ97], any other 'reasonable' metric on timed traces of timed network yield the same topology as the metric D.

For the metric D, a timed trace w, and a positive real $\varepsilon \in \mathbb{R}^+$, we define the D-tube around w of diameter ε to be the set $\mathcal{T}(w, \varepsilon) = \{w' \mid D(w, w') < \varepsilon\}$ of all timed traces at a distance less than ε from w. A D-open set Op is a set of timed traces such that for all timed trace $w \in Op$, there is a positive real $\varepsilon \in \mathbb{R}^+$ with $\mathcal{T}(w, \varepsilon) \subseteq Op$. 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 'open' to mean a D-open set.

Let the set of all timed traces be called TT. A set tts of timed traces is closed if its complement $TT - tts$ is open. The closure $\overline{tts}$ of a set tts of timed traces is the least closed set containing tts and the interior tts^{int} is the greatest open set contained in tts. Given a set O, we use $[O]$ to denote the set $(\overline{O})^{int}$.

The language L of a timed network induces a *robust language* $[L] = (\overline{L})^{int}$. The set $[L]$ represents the set of timed traces of N under the robust semantics. Notice that a computation π is in $[L]$ if and only if there is some open neighbourhood $\mathcal{T}(\pi, \varepsilon)$

around π within some distance ε such that each timed trace in the neighbourhood is also included in the closure of the set L of computations of TN.

5.1 Open Robust Timed Networks

In the following, we consider timed networks where clock constraints are negation-free and disjunction-free. For any given timed network, one can easily obtain such an equivalent timed network. Next we show the following.

OPEN ROBUST REACHABILITY ($OTN(K)$-ROBUST-REACH)

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

Question: Is $[L(\mathcal{O})] = \emptyset$?

We show that the above problem is undecidable (Theorem 5.4). The undecidability proof in Theorem 5.4 uses the following three lemmas.

Lemma 5.1 *For every open timed network $\mathcal{O}$, $L(\mathcal{O})$ is an open set.*

Proof. This proof is adapted from [GHJ97], where they show this lemma for timed automata. Consider an arbitrary timed trace $w \in L(\mathcal{O})$. Let $w = \langle(\xi_1, r_1), \ldots, (\xi_n, r_n)\rangle$. Since $w \in L(\mathcal{O})$, there is a computation $\pi = \gamma_{init} \longrightarrow_{T=\delta_1} \gamma'_1 \longrightarrow_{r_1} \gamma_1 \ldots \longrightarrow_{T=\delta_n} \gamma'_n \longrightarrow_{r_n} \gamma_n$, where $\delta_i = \xi_i - \xi_{i-1}$ for $i : 1 < i \leq n$ and $\delta_1 = \xi_1$. Let γ'_i be of the form $(I, q^i, \mathcal{Q}^i, X^i)$.

We will show that there is a positive ε such that $\mathcal{T}(w, \varepsilon) \subseteq L(\mathcal{O})$.

For each $i : 1 \leq i \leq n$,

- let ε_i be a real number strictly smaller than the minimum of the distances $|K - X^i_k(j)|$ (where $j \in I$) such that there is a guard $x_k < K$ or $x_k > K$ in the rule r_i and $g(X^i_k(j))$ is satisfied. (since all clock constraints are strict, these distances are strictly positive).

We define $\varepsilon := min\, \{\varepsilon_i/2 \mid 1 \leq i \leq n\}$.

Consider any timed trace $w' = \langle(\xi''_1, r_1), \ldots, (\xi''_n, r_n)\rangle$ where $D(w, w') < \varepsilon$. This means that $|\xi''_i - \xi_i| < \varepsilon$ for each i. We show that there is in fact a γ_{init}-computation $\gamma_{init} \longrightarrow_{T=\delta'_1} \beta'_1 \longrightarrow_{r_1} \beta_1 \ldots \longrightarrow_{T=\delta'_n} \beta'_n \longrightarrow_{r_n} \beta_n$, i.e., $w' \in L(\mathcal{O})$.

Let β'_i be of the form $(I, q^i, \mathcal{Q}^i, X''^i)$. Also, let $\varepsilon'_i = |\xi''_i - \xi_i|$ for each i. Recall that $\varepsilon'_i < \varepsilon$. In π, consider a clock value $X^i_k(j)$. We show that if the clock participated in rule r_i at global time ξ_i, with its new valuation at time ξ''_i, it can still participate in the rule r_i. Either $X^i_k(j) = \xi_i$ if it was never reset or $X^i_k(j) = \xi_i - \xi_\ell$ if it was last reset by the rule r_ℓ. Now, consider $\xi''_i - \xi''_\ell$. We know that $|\xi''_i - \xi_i| < \varepsilon$ and $|\xi''_\ell - \xi_\ell| < \varepsilon$. Then $\xi''_i - \xi''_\ell < (\xi_i + \varepsilon) - (\xi_\ell - \varepsilon) = \xi_i - \xi_\ell + 2 * \varepsilon = \xi_i - \xi_\ell + min(\varepsilon_i : 1 \leq i \leq n)$. From definition of ε and the fact that $X''^i_k(j) = \xi''_i - \xi''_\ell$, we know that the guard $g(X''^i_k(j))$ is still true and we have $\beta'_i \longrightarrow_{r_i} \beta_i$.

The case when the clock is never reset before is handled in a similar manner.

Let $\delta'_i = \xi''_i - \xi''_{i-1}$ and $\delta'_1 = \xi''_1$. Define $\beta'_i = \beta^{+\delta'_i}_{i-1}$ for $i > 1$ and $\beta'_1 = \gamma^{\delta'_1}_{init}$. From the assumption that w' is a timed trace, i.e., $\xi''_{i-1} \leq \xi''_i$, it is clear that $\beta'_{i-1} \longrightarrow_{T=\delta'_i} \beta'_i$ for $i > 1$ and $\gamma_{init} \longrightarrow_{T=\delta'_1} \beta'_1$.

This means that $w' \in L(\mathcal{O})$. This implies that $\mathcal{T}(w, \varepsilon) \subseteq L(\mathcal{O})$ and thus $L(\mathcal{O})$ is an open set. □

Lemma 5.2 *For an open set Op, $Op = \emptyset$ iff $[Op] = \emptyset$.*

Proof. Assume that $Op = \emptyset$. Then $\overline{Op} = \emptyset$. Then $(\overline{Op})^{int} = \emptyset$, i.e., $[Op] = \emptyset$.
Now we show the proof for the other direction. $Op \subseteq \overline{Op}$ and $Op^{int} \subseteq (\overline{Op})^{int}$. Since Op is open, $Op = Op^{int}$. Thus, $Op \subseteq (\overline{Op})^{int}$. Now, if $(\overline{Op})^{int} = \emptyset$, then from the above, it follows that $Op = \emptyset$. □

Lemma 5.3 *For every open timed network $\mathcal{O}$, $L(\mathcal{O}) = \emptyset$ iff $[L(\mathcal{O})] = \emptyset$.*

Proof. Since $L(\mathcal{O})$ is an open set by Lemma 5.1, the proof follows from Lemma 5.2. □

Now we show the following.

Theorem 5.4 *OTN(2)-Robust-Reach is undecidable.*

Proof. The undecidability of OTN(2)-Reach (Theorem 4.1) means that it is undecidable for an open timed network $\mathcal{O}$ whether $L(\mathcal{O}) = \emptyset$. From this and Lemma 5.3, the theorem follows. □

5.2 *Closed Robust Timed Networks*

In section 3, we showed that CTN(K)-Reach under standard semantics is decidable. In this section, we show that the problem becomes undecidable under robust semantics.
First, we define *runs*, which are extensions of traces and computations.

Definition 5.5 A run τ is a tuple $(\gamma_{init}, \Delta, H, R)$ where:

- $\gamma_{init} = (I, q, \mathcal{Q}, X)$ is an initial configuration of a timed network.
- $\Delta = \langle \delta_1, \ldots, \delta_n \rangle$ is a finite sequence of delays.
- $R = \langle r_1, \ldots, r_n \rangle$ is a finite sequence of rules. For $i : 1 \leq i \leq n$, let m_i be the number of transitions in rule r_i.
- $H = \langle h_1, \ldots, h_n \rangle$ is a finite sequence of injections. For $i : 1 \leq i \leq n, h_i : \{1, \ldots, m_i\} \rightarrow I$.

We say that a run τ is valid with respect to a timed network $\mathcal{N}$ if $\exists \langle \gamma_0, \gamma_1', \gamma_1 \ldots, \gamma_n', \gamma_n \rangle : \gamma_0 = \gamma_{init}, \gamma_i' \longrightarrow_{r_i} \gamma_i, \gamma_{i-1} \longrightarrow_{T=\delta_i} \gamma_i'$ and h_i is a witness injection for $\gamma_i' \longrightarrow_{r_i} \gamma_i$ for $i \in \{1, \ldots, n\}$. Note that for given H and Δ, there is at most one $\langle \gamma_0, \gamma_1', \gamma_1, \ldots, \gamma_n', \gamma_n \rangle$ satisfying these conditions. Let $comp(\tau)$ denote this sequence of configurations if it exists. We extend tt to runs in the obvious manner. Several computations may correspond to the same timed trace, and several runs may correspond to the same computation.

The next lemma relates traces and runs.

Lemma 5.6 *Let $\mathcal{N}$ be a timed network. If there exists an infinite sequence $\langle w_i \rangle \in L(\mathcal{N})$ where the distance between any two elements is at most $\varepsilon < \infty$, then there exists an infinite sequence $\langle \tau_{i_j} \rangle$ of valid runs of $\mathcal{N}$ of the form $\big(\gamma, \Delta_{i_j}, H, R\big)$ such that $tt(\tau_{i_j}) = w_{i_j}$. In other words, these valid runs differ only in their sequences of delays.*

Proof. Assume $\langle w_i \rangle$ and ε satisfying above conditions. Since the distance between any two timed traces in $\langle w_i \rangle$ is at most $\varepsilon < \infty$, all w_i are of same length n and all w_i share the same sequence of rules $R = \langle r_1, \ldots, r_n \rangle$. There is an infinite sequence $\langle \tau_i \rangle$ of runs such that $tt(\tau_i) = w_i$ since $w_i \in L(\mathcal{N})$. Let each τ_i is of the form $(\gamma_i, \Delta_i, H_i, R)$. Let L be the number of transitions in the largest rule multiplied by n. The number of processes participating in each run of length n is at most L. Without loss of generality, assume the index set of each γ_i is $I = \{1, \ldots, L\}$. From the fact that all γ_i are initial configurations, we conclude that all clock values in γ_i are equal to 0, all processes are in $idle^p$ and the controller is in $idle^c$. As all γ_i share the same index set, all γ_i are identical. Let $\gamma = \gamma_i$. The set of injections from $\{1, \ldots, L\}$ to $I = \{1, \ldots, L\}$ is obviously finite, and so is the set of sequences of length n of such injections. Therefore, there must exist an infinite subsequence $\langle \tau_{i_j} \rangle$ of $\langle \tau_i \rangle$ where all runs share the same sequence of injections H. Each run in this subsequence is of the form $\big(\gamma, \Delta_{i_j}, H, R\big)$. $\square$

Lemma 5.7 *Given a closed timed network $\mathcal{N}$, $L(\mathcal{N})$ is closed.*

Proof. Let $\langle w_i \rangle$ be an infinite sequence of timed traces in $L(\mathcal{N})$ converging to a timed trace w. To prove that $L(\mathcal{N})$ is closed, it is enough to show that $w \in L(\mathcal{N})$.

By Lemma 5.6, there is an infinite sequence $\langle \tau_i \rangle$ of valid runs, where each τ_i is of the form (γ, Δ_i, H, R).

The convergence of $\langle w_i \rangle$ implies the convergence of sequence $\langle \Delta_i \rangle$. Let Δ be the limit of $\langle \Delta_i \rangle$, and let $\tau = (\gamma, \Delta, H, R)$ with $\Delta = \langle \delta_1, \ldots, \delta_n \rangle, H = \langle h_1, \ldots, h_n \rangle, R = \langle r_1, \ldots, r_n \rangle$. Clearly, $tt(\tau) = w$. We will show that τ is a valid run of $\mathcal{N}$.

We show by induction on the length of runs that the sequence $\langle comp(\tau_i) \rangle$ converges. Let $\langle \gamma_0^i, \gamma_1'^i, \gamma_1^i, \ldots, \gamma_n'^i, \gamma_n^i \rangle = comp(\tau_i)$.

Induction hypothesis $IH(k)$: $\langle \gamma_0^i \rangle$ converges, $\forall j : 1 \le j \le k : \langle \gamma_j^i \rangle$ converges and $\langle \gamma_j'^i \rangle$ converges.

Base case $\forall i : \gamma_0^i = \gamma$, so $\langle \gamma_0^i \rangle$ clearly converges to $\gamma_0 = \gamma$.

Induction Step Assume $1 \le k \le n - 1$ and $IH(k)$.

- $\forall i : \gamma_{k+1}'^i = \gamma_k^i + \delta_{k+1}^i$. Since $\langle \delta_{k+1}^i \rangle$ converges to δ_{k+1}, and $\langle \gamma_k^i \rangle$ converges to γ_k by induction hypothesis, $\langle \gamma_{k+1}'^i \rangle$ converges to $\gamma_k + \delta_{k+1}$.
- Let X_{k+1} be the clock mapping defined as follows. For each process p used by h_{k+1} to execute r_{k+1}, $X_{k+1}(p)(l)$ is 0 if l is reset. For all other processes p and clocks l, $X_{k+1}(p)(l) = X_k(p)(l) + \delta_{k+1}$. Note that each clock mapping X_{k+1}^i of γ_{k+1}^i is defined as follows: For each process p used by h_{k+1} to execute r_{k+1}, $X_{k+1}^i(p)(l)$ is 0 if l is reset. For all other processes p and clocks l, $X_{k+1}^i(p)(l) = X_k^i(p)(l) + \delta_{k+1}^i$. Since $\langle X_k^i \rangle$ converges to X_k by induction hypothesis, $\langle X_{k+1}^i \rangle$

converges to X_{k+1}, and $\langle \gamma^i_{k+1} \rangle$ converges.

Let the sequence of configurations $\langle \gamma_0, \gamma'_1, \gamma_1, \ldots, \gamma_n \rangle$ be the limit of $\langle comp(\tau_i) \rangle$. All clock values in these configurations are limits of clock values in $\langle comp(\tau_i) \rangle$. By definition of closed sets, all sequences converging in a closed set do so within the set. This implies that all guards satisfied in each $comp(\tau_i)$ are still satisfied in $\langle \gamma_0, \gamma'_1, \gamma_1, \ldots, \gamma_n \rangle$. Therefore it is the case that $\gamma_0 \longrightarrow_{T=\delta_1} \gamma'_1 \longrightarrow_{r_1} \gamma_1 \ldots \longrightarrow_{r_n} \gamma_n$.

□

For a timed network $\mathcal{N}$, we let $\overline{\mathcal{N}}$ be the closed timed network derived from $\mathcal{N}$ by replacing each strict constraint by its non-strict counter-part.

Lemma 5.8 *For an open timed network $\mathcal{O}$, if there exist w and $\varepsilon > 0$ such that $\mathcal{T}(w, \varepsilon) \subseteq L(\overline{\mathcal{O}})$ then $L(\mathcal{O}) \neq \emptyset$.*

Proof. Consider $w = \langle (\xi_1, r_1) \ldots (\xi_n, r_n) \rangle$ and ε which satisfy the above constraints.

Let ε' such that $0 < \varepsilon' < \frac{1}{n} min \begin{pmatrix} \{1 - \mathit{fract}(\xi_i), 1 \leq i \leq n\} \cup \\ \{1 - \mathit{fract}(\xi_j - \xi_i), 1 \leq i < j \leq n\} \cup \\ \{\varepsilon\} \end{pmatrix}$

Let $w' = \langle (\xi'_1 r_1) \ldots (\xi'_n, r_n) \rangle$, where $\xi'_i = \xi_i + i * \varepsilon'$. It is the case that $D(w', w) < \varepsilon$, hence $w' \in L(\overline{\mathcal{O}})$. Moreover, since $\varepsilon' < \frac{1 - \mathit{fract}(\xi_i)}{n}$ and $i \leq n$, $\mathit{fract}(\xi'_i) \neq 0$.
For $1 \leq i < j \leq n$, let $\Delta(i,j) = \xi'_j - \xi'_i = \xi_j - \xi_i + (j - i - 1) * \varepsilon'$. $\Delta(i,j)$ denotes the amount of time elapsed between events i and j in w'. Since $\varepsilon' < \frac{1 - \mathit{fract}(\xi_j - \xi_i)}{n}$ and $0 \leq (j - i - 1) \leq n$, $\Delta(i,j) \neq 0$.
Consider any computation $\pi = \gamma_0 \longrightarrow_{T=\xi_1+\varepsilon'} \gamma'_1 \longrightarrow_{r_1} \gamma_1 \longrightarrow_{T=\Delta(1,2)} \gamma'_2 \cdots \gamma_{n-1} \longrightarrow_{T=\Delta(n-1,n)} \gamma'_n \longrightarrow_{r_n} \gamma_n$.
Each clock value in each γ'_i is either of the form ξ'_i if the clock was never reset, or $\xi'_i - \xi'_j$ if it was most recently reset in event j. Hence its value is in $\{\Delta(i,j), 1 \leq i < j \leq n\} \cup \{\xi_i + i * \varepsilon', 1 \leq i \leq n\}$, which does not intersect the set of natural numbers. Therefore no clock value is a natural number, which means that all runs of π satisfy all guards of $\overline{\mathcal{O}}$ strictly. Finally, this implies that they satisfy all guards of $\mathcal{O}$, and $w' \in L(\mathcal{O})$.

□

Theorem 5.9 *CTN(2)-Robust-Reach is undecidable.*

Proof. We show the undecidability of CTN(2)-Robust-Reach by reducing OTN(2)-Robust-Reach to CTN(2)-Robust-Reach. Consider an open timed network $\mathcal{O}$ with 2 clocks.
Now we show that $[L(\mathcal{O})] = \emptyset$ iff $[L(\overline{\mathcal{O}})] = \emptyset$.

- $[L(\overline{\mathcal{O}})] = \emptyset \implies [L(\mathcal{O})] = \emptyset$. It is straightforward that $L(\mathcal{O}) \subseteq L(\overline{\mathcal{O}})$, therefore $[L(\mathcal{O})] \subseteq [L(\overline{\mathcal{O}})]$. Thus, if $[L(\overline{\mathcal{O}})] = \emptyset$, then $[L(\mathcal{O})] = \emptyset$.
- $[L(\mathcal{O})] = \emptyset \implies [L(\overline{\mathcal{O}})] = \emptyset$. Since for an open timed network $\mathcal{O}$, $[L(\mathcal{O})] = \emptyset$

iff $L(\mathcal{O}) = \emptyset$ (Lemma 5.3), it is enough to show that $L(\mathcal{O}) = \emptyset \implies [L(\overline{\mathcal{O}})] = \emptyset$. Assume $L(\mathcal{O}) = \emptyset$. By Lemma 5.8, for each $w \in L(\overline{\mathcal{O}})$ and for each $\varepsilon > 0$, $\mathcal{T}(w, \varepsilon) \nsubseteq L(\overline{\mathcal{O}})$. By Lemma 5.7, $L(\overline{\mathcal{O}})$ is closed, therefore $\overline{L(\overline{\mathcal{O}})} = L(\overline{\mathcal{O}})$. Consequently $[L(\overline{\mathcal{O}})] = \overline{(L(\overline{\mathcal{O}}))}^{int} \subseteq L(\overline{\mathcal{O}})$, which implies that $\mathcal{T}(w, \varepsilon) \nsubseteq [L(\overline{\mathcal{O}})]$. However $[L(\overline{\mathcal{O}})]$ is open, therefore it must be empty.

□

Remark 1 OTN(1)-Robust-Reach is decidable due to decidability of TN(1)-Reach [AJ03] and Lemma 5.3.

Remark 2 CTN(1)-Robust-Reach is also decidable due to decidability of OTN(1)-Robust-Reach and the fact that given a CTN $\mathcal{N}$, one can construct an open timed network $\mathcal{N}^{op}$ such that $[L(\mathcal{N})] = \emptyset$ iff $[L(\mathcal{N}^{op})] = \emptyset$. Proof of this is similar to the proof for Theorem 5.9.

6 Conclusion

We have shown that the controller state reachability problem for multi-clock timed networks is decidable if TN is closed, undecidable otherwise. However, semantic removal of equality under robust semantics makes the problem undecidable even for closed TNs. This emphasises the fact that robust semantics is more intractable than the standard semantics of TNs. This fact was already noted by [OW03a] for timed automata.

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.

A Appendix

In the proofs we need some definitions.

In our correctness proof of Theorem 4.4, 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 $\mathcal{N} = (Q, \Re)$ be a timed network. We define $\Longrightarrow_r$ to denote $\longrightarrow_{Timed} \circ \longrightarrow_r \circ \longrightarrow_{Timed}$, 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 $\bigcup_{r \in \Re} \Longrightarrow_r$. For a set $\mathtt{R} \subseteq \Re$ 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.

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

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

A graphical representation of $c_1^{inc_1}$semi-encoding, $c_1^{inc_2}$semi-encoding and $c_1^{inc_3}$semi-encoding is shown in Figure 5(b), 5(c), and 5(d) respectively.

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

- a $c_1^{rot_1}$semi-encoding of *value* m if there is an injection h from $\{0, \ldots, m+1\}$ to I such that the following conditions are satisfied

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

- a $c_1^{rot_2}$ semi-encoding of *value* m if there is an injection h from $\{0, \ldots, m+1\}$ to I such that the following conditions are satisfied
  - $\cdot$ $\mathcal{Q}$ is as defined for a $c_1^{rot_1}$ semi-encoding.
  - $\cdot$ $X_1(h(i)) < X_2(h(i-1))$ for each $i : 1 \leq i \leq m+1$.
  - $\cdot$ $X_2(h(i)) < X_1(h((i+2)))$, for each $i : 0 \leq i \leq m-1$.
  - $\cdot$ $X_1(h(0)) < X_1(h(1))$.
  - $\cdot$ $X_2(h(m)) < X_2(h(m+1))$.

Figure 8 illustrates the rotation of a c_1-encoding graphically and shows a graphical representation of $c_1^{rot_1}$ semi-encoding and $c_1^{rot_2}$ semi-encoding in Figure 8(b) and 8(c) respectively.

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

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

In such a case, we define $sig(\gamma)$ of γ to be the triple $(tmp_1^{\imath}, m_1, m_2)$, 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 $inc_2^{\imath}$, $inc_3^{\imath}$ and $inc_4^{\imath}$ 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:

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

We define the signature of a $c_1^{inc_2}$ semi-encoding as in $c_1^{inc_1}$ semi-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 $inc_3^{\imath}$ and $inc_4^{\imath}$ 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 C or $q = tmp_1^{\imath}$ for some zero-testing instruction (of the form $\imath = (s_1, c_1?0, s_2)$).

- γ is both a c_2-encoding and a $c_1^{rot_1}$ semi-encoding.

The signature for a $c_1^{rot_1}$ semi-encoding is defined in a similar manner to a $c_1^{inc_1}$ semi-encoding. This means that $sig(\gamma) = (tmp_{11}^s, m_1, m_2)$ or $sig(\gamma) = (tmp_1^\imath, m_1, m_2)$ depending on the value of q. Then $next(\gamma)$ 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 C, or $q = tmp_2^\imath$ for some zero-testing instruction (of the form $\imath = (s_1, c_1?0, s_2)$).
- γ is both a c_2-encoding and a $c_1^{rot_2}$ semi-encoding.

The signature and the function $next(\gamma)$ can be defined for a $c_1^{rot_2}$ semi-encoding in a similar manner to a $c_1^{rot_1}$ semi-encoding. In the following, sometimes we use semi-encoding to means semi-encodings of some Type. The notion of a (semi-)encoding can be extended to a *proper* (semi-)encoding in the same manner as before (Section 4.1), i.e. we require clocks of all processes which are not idle to have values strictly between zero and one.

Proof of Theorem 4.4
The if-direction follows immediately from the the following lemma.

Lemma A.1 *For any configuration $\gamma = (I, q, \mathcal{Q}, X)$ and initial configuration γ_{init} in $\mathcal{O}_{\mathsf{C}}$, if $\gamma_{init} \xrightarrow{*} \gamma$ then one of the following holds.*

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

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

Lemma A.2 *If $\beta_{init} \overset{n}{\rightsquigarrow} s_F$ then $\gamma_{init} \xrightarrow{*} s_F$, for each $n \geq 0$ and initial configuration γ_{init} of $\mathcal{O}_{\mathsf{C}}$ 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 A.1 and Lemma A.2 reflect the informal arguments provided together with each rule in Section 4.2.

Proof of Lemma A.1

Suppose that $\gamma_{init} \xrightarrow{*} \gamma$. If $\gamma_{init} \longrightarrow_{Timed} \gamma$ 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 $i : 0 \le i \le n$. 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 $q_1 = s^1_{init}$, $q_2 = s^2_{init}$ and $q_3 = s^3_{init}$. In other words, $\gamma_0, \ldots, \gamma_3$ satisfy the claim of the Lemma. Lemma A.1 follows from the following property:
For each $4 \le i \le n$, 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 $i : 4 \le i < n$, it follows from the rule definitions that one of the following cases is satisfied:

(i) $r_i = inc^\imath_1$ for some $\imath = (s_1, c_1{+}{+}, s_2)$, 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 $sig(\gamma_{i+1}) = (tmp^\imath_1, m_1 + 1, m_2)$.

(ii) $r_i = inc^\imath_2$ for some $\imath = (s_1, c_1{+}{+}, s_2)$, γ_i is a Type $1a$ semi-encoding with $sig(\gamma_i) = (tmp^\imath_1, m_1, m_2)$, and γ_{i+1} is a Type $1b$ semi-encoding with $sig(\gamma_{i+1}) = (tmp^\imath_2, m_1, m_2)$.

(iii) $r_i = inc^\imath_3$ for some $\imath = (s_1, c_1{+}{+}, s_2)$, γ_i is a Type $1b$ semi-encoding with $sig(\gamma_i) = (tmp^\imath_2, m_1, m_2)$, and γ_{i+1} is a Type $1c$ semi-encoding with $sig(\gamma_{i+1}) = (tmp^\imath_3, m_1, m_2)$.

(iv) $r_i = inc^\imath_4$ for some $\imath = (s_1, c_1{+}{+}, s_2)$, γ_i is a Type $1c$ semi-encoding with $sig(\gamma_i) = (tmp^\imath_3, m_1, m_2)$, and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s_2, m_1, m_2)$.

(v) $r_i = dec^\imath$ for some $\imath = (s_1, c_1{-}{-}, s_2)$, 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 $sig(\gamma_{i+1}) = (s_2, m_1 - 1, m_2)$.

(vi) $r_i = rot^s_{1,1}$ 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 $sig(\gamma_{i+1}) = (tmp^s_{11}, m_1, m_2)$.

(vii) $r_i \in \{rot^s_{1,2}, rot^s_{1,3}\}$ for some $s \in S$ and γ_i is a Type $1d$ semi-encoding with $sig(\gamma_i) = (tmp^s_{11}, m_1, m_2)$, and γ_{i+1} is a Type $1e$ semi-encoding with $sig(\gamma_{i+1}) = (tmp^s_{12}, m_1, m_2)$.

(viii) $r_i = rot^s_{1,4}$ for some $s \in S$ and γ_i is a Type $1e$ semi-encoding with $sig(\gamma_i) = (tmp^s_{12}, m_1, m_2)$, and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s, m_1, m_2)$.

(ix) $r_i = tst^\imath_1$ for some $\imath = (s_1, c_1 = 0?, s_2)$, 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 $sig(\gamma_{i+1}) = (tmp^\imath_1, 0, m_2)$.

(x) $r_i = tst^\imath_2$ for some $\imath = (s_1, c_1 = 0?, s_2)$, and γ_i is a Type $1d$ semi-encoding with $sig(\gamma_i) = (tmp^\imath_1, 0, m_2)$ is and γ_{i+1} is a Type $1e$ semi-encoding with $sig(\gamma_{i+1}) = (tmp^\imath_2, 0, m_2)$.

(xi) $r_i = tst^\imath_3$ for some $\imath = (s_1, c_1 = 0?, s_2)$, and γ_i is a Type $1e$ semi-encoding with $sig(\gamma_i) = (tmp_2^i, 0, m_2)$ is and γ_{i+1} is an encoding with $sig(\gamma_{i+1}) = (s_2, 0, m_2)$.

(xii) Similar cases corresponding to instructions which change counter c_2.

Proof of Lemma A.2

To show Lemma A.2 we use some definitions.

Let $\gamma = (I, q, \mathcal{Q}, X)$ be a configuration in our simulation. 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. We define $Latest_2(\gamma)$ in a similar manner, and define $Latest(\gamma) = max(Latest_1(\gamma), Latest_2(\gamma))$. We also define $Next2Latest_1(\gamma) = max(X_k(i) : k \in \{1,2\} \wedge i \in I \wedge \mathcal{Q}(i) \in \{fst_1, mid_1, last_1\} \wedge X_k(i) < Latest_1(\gamma))$. In other words, $Next2Latest_1(\gamma)$ is the next highest among values of clocks belonging to processes which are part of the (semi-)c_1-encoding. We define $Next2Latest_2(\gamma)$ in a similar manner.

Let $Delay_1(\gamma)$ be the size of the set consisting of clock values of the form $X_i(j)$ where $i \in \{1,2\}, j \in I, \mathcal{Q}(j) \in \{fst_2, mid_2, last_2\}$ and $Latest_1(\gamma) < X_i(j)$. In other words, $Delay_1(\gamma)$ is the number of clocks which are part of the c_2-encoding and which have 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* c_1-encoding which is a 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 $0 < X_1(j), X_2(j) < 1$. Also, $0 < X_1(i) < 1$. An almost proper encoding of Type 1 is an almost proper c_1-encoding and a proper c_2-encoding. Similarly, we define an almost proper c_2-encoding and an almost proper encoding of Type 2. We also extend this notion of almost proper encodings to almost proper semi-encodings.

Lemma A.2 follows immediately from the following lemma.

Lemma A.3 *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 $\gamma_6 = \gamma_5^{+\delta_3}$ with $0 < \delta_3 < 1 - Latest(\gamma_5)$. We have $\gamma_5 \longrightarrow_{T=\delta_3} \gamma_6$. 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 δ_4 be such that $0 < \delta_4 < 1 - Latest(\gamma_7)$.

δ_4 exists by the definition of δ_1, δ_2, δ_3, $init_1$, $init_2$, $init_3$ and $init_4$. Let $\gamma_8 = \gamma_7^{+\delta_4}$. 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. $\square$

Lemma A.4 *Let β_1 and β_2 be configurations of 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 A.4 follows from Lemma A.5, Lemma A.6, Lemma A.7, Lemma A.8, Lemma A.14, and Lemma A.15:

- Lemma A.5, Lemma A.6, Lemma A.7 and Lemma A.8 state that an *increment* can be simulated by an application of the rule $inc_1^{\imath}$ followed by an application of the rule $inc_2^{\imath}$, followed by an application of rule $inc_3^{\imath}$, followed by an application of rule $inc_4^{\imath}$.
- Lemma A.14 states that a *decrement* can be simulated by the rule $dec^{\imath}$ preceded and followed by a number of rotations. This lemma follows from Lemma A.9, Lemma A.10, Lemma A.11, Lemma A.12, and Lemma A.13.
- Lemma A.15 deals with zero testing and is similar to Lemma A.14.

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

Lemma A.5 *Consider an instruction $\imath = (s_1, c_1{+}{+}, s_2)$. 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 $sig(\gamma_2) = (tmp_1^{\imath}, m_1+1, m_2)$ and $\gamma_1 \Longrightarrow_{inc_1^{\imath}} \gamma_2$. A similar result holds in case $\imath$ 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 $inc_1^{\imath}$ is enabled from γ_1, i.e., there is configuration γ_3 with $\gamma_1 \longrightarrow_{inc_1^{\imath}} \gamma_3$. Define $\gamma_2 = \gamma_3^{+\delta}$ where $0 < \delta < 1 - Latest(\gamma_3)$. Such a δ exists by definition of $inc_1^{\imath}$ 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 $sig(\gamma_2) = (tmp_1^{\imath}, m_1+1, m_2)$ and $\gamma_1 \longrightarrow_{inc_1^{\imath}} \gamma_3 \longrightarrow_{T=\delta} \gamma_2$. $\square$

Lemma A.6 *Consider an instruction $\imath = (s_1, c_1{+}{+}, s_2)$. Let γ_1 be a proper semi-encoding of Type $1a$ with $sig(\gamma_1) = (tmp_1^{\imath}, m_1, m_2)$ and $|\gamma_1| \geq m_1+m_2+4$. There is a proper semi-encoding γ_2 of Type $1b$ such that $sig(\gamma_2) = (tmp_2^{\imath}, m_1, m_2)$ and $\gamma_1 \Longrightarrow_{inc_2^{\imath}} \gamma_2$.*

A similar result holds in case $\imath$ 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 $inc_2^\imath$ is enabled from γ_1, i.e., there is configuration γ_3 with $\gamma_1 \longrightarrow_{inc_2^\imath} \gamma_3$. Define $\gamma_2 = \gamma_3^{+\delta}$ where $0 < \delta < 1 - Latest(\gamma_3)$. Such a δ exists by definition of $inc_2^\imath$ 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 $sig(\gamma_2) = (tmp_2^\imath, m_1, m_2)$ and $\gamma_1 \longrightarrow_{inc_2^\imath} \gamma_3 \longrightarrow_{T=\delta} \gamma_2$. □

Lemma A.7 *Consider an instruction $\imath = (s_1, c_1{+}{+}, s_2)$. Let γ_1 be a semi-encoding of Type $1b$ with $sig(\gamma_1) = (tmp_2^\imath, m_1, m_2)$. There is a proper semi-encoding γ_2 of Type $1c$ such that $sig(\gamma_2) = (tmp_3^\imath, m_1, m_2)$ and $\gamma_1 \Longrightarrow_{inc_3^\imath} \gamma_2$.*
A similar result holds in case $\imath$ is of the form $(s_1, c_2{+}{+}, s_2)$.

Proof is similar to that of Lemma A.6.

Lemma A.8 *Consider an instruction $\imath = (s_1, c_1{+}{+}, s_2)$. Let γ_1 be a proper semi-encoding of Type $1c$ with $sig(\gamma_1) = (tmp_3^\imath, m_1, m_2)$. There is a proper encoding γ_2 such that $sig(\gamma_2) = (s_2, m_1, m_2)$ and $\gamma_1 \Longrightarrow_{inc_3^\imath} \gamma_2$.*
A similar result holds in case $\imath$ is of the form $(s_1, c_2{+}{+}, s_2)$.

Proof is similar to that of the above.

Lemma A.9 *Let γ_1 be an (almost) proper c_2-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 $Delay_1(\gamma_1) - Delay_1(\gamma_2) \in \{0, 1\}$, $sig(\gamma_2) = (tmp_{21}^s, m_1, m_2)$, and $\gamma_1 \Longrightarrow_{rot_{2,1}^s} \gamma_2$.*
A similar result holds in case $Delay_2(\gamma_1) > 0$.

Proof.
Now, γ_1 is an (almost) proper c_2-encoding and by definition of the rule $rot_{2,1}^s$, $rot_{2,1}^s$ is enabled from γ_1 and there is a γ_3 with $\gamma_1 \longrightarrow_{rot_{2,1}^s} \gamma_3$. Define $\gamma_2 = \gamma_3^{+\delta}$ where $0 < \delta < 1 - Latest(\gamma_3)$ if γ_1 is a proper encoding, and $0 < \delta < 1 - max(Next2Latest_2(\gamma_3), Latest_1(\gamma_3))$ otherwise. Existence of δ follows from the definition of the rule $rot_{2,1}^s$, and the fact that γ_1 is an (almost) proper c_2-encoding. By the definitions it follows that γ_2 is an (almost) proper semi-encoding of Type $2d$ with

- $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$ if $Latest_2(\gamma_1)$ is strictly smaller than the value of the clock x_1 of the process in state fst_2.
- $Delay_1(\gamma_2) = Delay_1(\gamma_1)$ otherwise.

with $sig(\gamma_2) = (tmp_{21}^s, m_1, m_2)$ and $\gamma_1 \longrightarrow_{rot_{2,1}^s} \gamma_3 \longrightarrow_{T=\delta} \gamma_2$. □

Lemma A.10 *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 $Delay_1(\gamma_1) - Delay_1(\gamma_2) \in \{0, 1\}$, $sig(\gamma_1) = (tmp_{22}^s, m_1, m_2)$, and either $\gamma_1 \Longrightarrow_{rot_{2,2}^s} \gamma_2$ or $\gamma_1 \Longrightarrow_{rot_{2,3}^s} \gamma_2$.*
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 $m_2 > 0$. Now there are two cases.

- γ_1 is a proper semi-conding of Type 2d. To apply $rot^s_{2,2}$, we need to first define δ_1 such that $1 - Latest(\gamma_1) < \delta_1 < 1 - max(Next2Latest_2(\gamma_1), Latest_1(\gamma_1))$ and $\gamma_3 = \gamma_1^{+\delta_1}$. δ_1 exists due to the fact that γ_1 is a proper semi-encoding of Type 2d and by definition of the rule $rot^s_{2,1}$. From the value of δ_1 and the fact that $m_2 > 0$ it follows that $rot^s_{2,2}$ is enabled from γ_3, i.e., there is a γ_4 with $\gamma_3 \longrightarrow_{rot^s_{2,2}} \gamma_4$.
- γ_1 is an almost proper semi-conding of Type 2d. Now, $rot^s_{2,2}$ is enabled from γ_1, i.e., there is a γ_4 with $\gamma_1 \longrightarrow_{T=0}\longrightarrow_{rot^s_{2,2}} \gamma_4$.

Define $\gamma_2 = \gamma_4^{+\delta_2}$ where $0 < \delta < 1 - max(Next2Latest_2(\gamma_4), Latest_1(\gamma_4))$. Existence of δ_2 follows from the definition of the rule $rot^s_{2,2}$, existence of δ_1 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 $last_2$ is smaller than or equal to $Latest_1(\gamma_1)$, in other words, if $Delay_1(\gamma_1) = 1$,
- $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$, otherwise.

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 $rot^s_{2,3}$, and obtain $\gamma_1 \longrightarrow_{T=\delta_1} \gamma_3 \longrightarrow_{rot^s_{2,3}} \gamma_4 \longrightarrow_{T=\delta_2} \gamma_2$. □

Lemma A.11 *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 $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$, with $sig(\gamma_1) = (s, m_1, m_2)$, and $\gamma_1 \Longrightarrow_{rot^s_{2,4}} \gamma_2$.*
A similar result holds in case $Delay_2(\gamma_1) > 0$.

Proof. 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 $rot^s_{2,4}$ is enabled from γ_1, i.e., there is a γ_3 with $\gamma_1 \longrightarrow_{rot^s_{2,4}} \gamma_3$. Define $\gamma_2 = \gamma_3^{+\delta_1}$ where $0 < \delta_1 < 1 - Latest(\gamma_3)$. Existence of δ_1 follows from the definition of the rule $rot^s_{2,4}$, 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 $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$ (due to the fact that the largest clock in the semi-encoding of Type 2e is reset and $Delay_1(\gamma_1) > 0$), and $sig(\gamma_2) = (s, m_1, m_2)$. □

Lemma A.12 *Let γ_1 be an almost proper semi-encoding of Type $2e$ with $Delay_1(\gamma_1) > 0$ such that $sig(\gamma_1) = (tmp^s_{23}, m_1, m_2)$. There is a proper encoding γ_2 such that $Delay_1(\gamma_2) = Delay_1(\gamma_1) - 1$, with $sig(\gamma_1) = (s, m_1, m_2)$, and $\gamma_1 \Longrightarrow_{rot^s_{2,4}} \gamma_2$.*
A similar result holds in case $Delay_2(\gamma_1) > 0$.

Proof of this lemma is similar to that of the previous one.

Lemma A.13 *Consider an instruction $\imath = (s_1, c_1{-}{-}, s_2)$. 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.*

(i) *If* $Latest_1(\gamma_1) > Latest_2(\gamma_1)$ *then* $\gamma_1 \Longrightarrow_{dec^\imath} \gamma_2$.

(ii) *If* $Latest_1(\gamma_1) = Latest_2(\gamma_1)$ *and* $m_2 > 0$ *then* $\gamma_1 \Longrightarrow_{dec^\imath} \circ \Longrightarrow_{rot^{s_2}_{2,1}} \circ \Longrightarrow_{rot^{s_2}_{2,2}} \circ \Longrightarrow_{rot^{s_2}_{2,4}} \gamma_2$.

(iii) *If* $Latest_1(\gamma_1) = Latest_2(\gamma_1)$ *and* $m_2 = 0$ *then* $\gamma_1 \Longrightarrow_{dec^\imath} \circ \Longrightarrow_{rot^{s_2}_{2,1}} \circ \Longrightarrow_{rot^{s_2}_{2,3}} \circ \Longrightarrow_{rot^{s_2}_{2,4}} \gamma_2$.

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

Proof. Define $\gamma_3 = \gamma_1^{+\delta_1}$ where

- $1 - Latest_1(\gamma_1) < \delta_1 < 1 - max(Next2Latest_1(\gamma_1), Latest_2(\gamma_1))$ if $Latest_1(\gamma_1) > Latest_2(\gamma_1)$.
- $1 - Latest_1(\gamma_1) < \delta_1 < 1 - max(Next2Latest_1(\gamma_1), Next2Latest_2(\gamma_1))$ if $Latest_1(\gamma_1) = Latest_2(\gamma_1)$.

Such a δ_1 exists since γ_1 is a proper encoding. From the definition of δ_1 and the fact that $m_1 > 0$ it follows that $dec^\imath$ is enabled from γ_3, i.e., there is a γ_4 with $\gamma_3 \longrightarrow_{dec^\imath} \gamma_4$. Now there are three cases depending on the values of $Latest_1(\gamma_1)$ and $Latest_2(\gamma_1)$ as follows:

(i) If $Latest_1(\gamma_1) > Latest_2(\gamma_1)$ then define $\gamma_2 = \gamma_4^{+\delta_2}$ where $0 < \delta_2 < 1 - Latest(\gamma_4)$. Existence of δ_2 follows from the manner in which δ_1 is chosen, definition of the rule $dec^\imath$, 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 $\gamma_1 \longrightarrow_{T=\delta_1} \gamma_3 \longrightarrow_{dec^\imath} \gamma_4 \longrightarrow_{T=\delta_2} \gamma_2$.

(ii) If $Latest_1(\gamma_1) = Latest_2(\gamma_1)$ and $m_2 > 0$. γ_1 is a proper c_2-encoding, but γ_4 is an almost proper c_2-encoding with $sig(\gamma_4) = (s_2, m_1 - 1, m_2)$, From this fact, it is clear that the rule $rot^{s_2}_{2,1}$ is enabled from γ_4, i.e., there is a γ_5 with $\gamma_4 \longrightarrow_{rot^{s_2}_{2,1}} \gamma_5$. Define $\gamma_6 = \gamma_5^{+\delta_2}$ where $0 < \delta_2 < 1 - max(Next2Latest_2(\gamma_5), Latest_1(\gamma_5))$. Existence of δ_2 follows from the manner in which δ_1 is chosen, definitions of the rules $dec^\imath$ and $rot^{s_2}_{2,1}$, 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 $sig(\gamma_6) = \left(tmp^{s_2}_{2,2}, m_1 - 1, m_2\right)$. From the definition of $dec^\imath$ and the condition that $Latest_1(\gamma_1) = Latest_2(\gamma_1)$, it follows that the largest clock value of the processes in the c_2-encoding has value larger than 1 in γ_6. Therefore, the rule $rot^{s_2}_{2,2}$ is enabled from γ_6, i.e., there is a γ_7 with $\gamma_6 \longrightarrow_{rot^{s_2}_{2,2}} \gamma_7$. Define $\gamma_8 = \gamma_7^{+\delta_3}$ where $0 < \delta_3 < 1 - max(Next2Latest_2(\gamma_7), Latest_1(\gamma_7))$. Existence of δ_3 follows from the manner in which δ_1, δ_2 are chosen, definitions of the rules $rot^{s_2}_{2,2}$, 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 $sig(\gamma_8) = \left(tmp^{s_2}_{2,2}, m_1 - 1, m_2\right)$. Therefore, the rule $rot^{s_2}_{2,4}$ is enabled from γ_8, i.e., there is a γ_9 with $\gamma_8 \longrightarrow_{rot^{s_2}_{2,4}} \gamma_9$. Define $\gamma_2 = \gamma_9^{+\delta_4}$ where

$0 < \delta_4 < 1 - \mathit{Latest}(\gamma_9)$. By the definitions it follows that γ_2 is a proper encoding with $\mathit{sig}(\gamma_2) = (s_2, m_1 - 1, m_2)$. and $\gamma_1 \longrightarrow_{T=\delta_1} \gamma_3 \longrightarrow_{dec^\imath} \gamma_4 \longrightarrow_{rot_{2,1}^{s_2}} \gamma_5 \longrightarrow_{T=\delta_2} \gamma_6 \longrightarrow_{rot_{2,2}^{s_2}} \gamma_7 \longrightarrow_{T=\delta_3} \gamma_8 \longrightarrow_{rot_{2,4}^{s_2}} \gamma_9 \longrightarrow_{T=\delta_4} \gamma_2$.

(iii) If $\mathit{Latest}_1(\gamma_1) = \mathit{Latest}_2(\gamma_1)$, but $m_2 = 0$. The proof is similar to the previous case. Here, we use the rule $rot_{2,3}^{s_2}$ instead of the rule $rot_{2,2}^{s_2}$ in the above and obtain $\gamma_1 \longrightarrow_{T=\delta_1} \gamma_3 \longrightarrow_{dec^\imath} \gamma_4 \longrightarrow_{rot_{2,1}^{s_2}} \gamma_5 \longrightarrow_{T=\delta_2} \gamma_6 \longrightarrow_{rot_{2,3}^{s_2}} \gamma_7 \longrightarrow_{T=\delta_3} \gamma_8 \longrightarrow_{rot_{2,4}^{s_2}} \gamma_9 \longrightarrow_{T=\delta_4} \gamma_2$.

□

From Lemma A.9, Lemma A.10, Lemma A.11, Lemma A.12, and Lemma A.13 we get the following.

Lemma A.14 *Consider an instruction* $\imath = (s_1, c_1 - -, s_2)$. *Let* γ_1 *be a proper encoding with* $\mathit{sig}(\gamma_1) = (s_1, m_1, m_2)$ *and* $m_1 > 0$. *There is a proper encoding* γ_2 *such that* $\mathit{sig}(\gamma_2) = (s_2, m_1 - 1, m_2)$ *and*

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

where for a controller state s, *we define* $rot_2^s = \left\{rot_{2,1}^s, rot_{2,2}^s, rot_{2,3}^s, rot_{2,4}^s\right\}$.
A similar result holds in case $\imath$ *is of the form* $(s_1, c_2 - -, s_2)$.

Lemma A.15 *Consider an instruction* $\imath = (s_1, c_1 = 0?, s_2)$. *Let* γ_1 *be a proper encoding with* $\mathit{sig}(\gamma_1) = (s_1, 0, m_2)$.
Then there is a proper encoding γ_2 *such that* $\mathit{sig}(\gamma_2) = (s_2, 0, m_2)$ *and*

$$\gamma_1 \overset{*}{\Longrightarrow}_{rot_2^{s_1}} \circ \Longrightarrow_{tst_1^\imath} \circ \Longrightarrow_{tst_2^\imath} \circ \Longrightarrow_{tst_3^\imath} \circ \overset{*}{\Longrightarrow}_{rot_2^{s_2}} \gamma_2$$

where for a controller state s, rot_2^s *is as defined in Lemma A.14.*
A similar result holds in case $\imath$ *is of the form* $(s_1, c_2 = 0?, s_2)$.

The proof is similar to that for Lemma A.14.