Communicating Timed Processes with Perfect Timed Channels

Parosh Abdulla[1], M. Faouzi Atig[1], and S. Krishna[2]

2 Dept of CSE, IIT Bombay, India
`krishnas@cse.iitb.ac.in`
2 Uppsala University, Sweden
`parosh,mohamed_faouzi.atig@it.uu.se`

Abstract.

We introduce the model of communicating timed automata (CTA) that extends the classical models of finite-state processes communicating through FIFO perfect channels and timed automata, in the sense that the finite-state processes are replaced by timed automata, and messages inside the perfect channels are equipped with clocks representing their ages. In addition to the standard operations (resetting clocks, checking guards of clocks) each automaton can either (1) append a message to the tail of a channel with an initial age or (2) receive the message at the head of a channel if its age satisfies a set of given constraints. In this paper, we show that the reachability problem is undecidable even in the case of two timed automata connected by one unidirectional timed channel if one allows global clocks (that the two automata can check and manipulate). We prove that this undecidability still holds even for CTA consisting of three timed automata and two unidirectional timed channels (and without any global clock). However, the reachability problem becomes decidable (in EXPTIME) in the case of two automata linked with one unidirectional timed channel and with no global clock. Finally, we consider the bounded-context case, where in each context, only one timed automaton is allowed to receive messages from one channel while being able to send messages to all the other timed channels. In this case we show that the reachability problem is decidable.

1 Introduction

In the last few years, several papers have been devoted to extend classical infinite-state systems such as pushdown systems, (lossy) channel systems and Petri nets with timed behaviors in order to obtain more accurate and precise formal models (e.g., [3, 2, 9, 1, 29, 12, 22, 21, 18, 25, 20, 19, 11, 7, 23, 14, 6, 10]). In particular, *perfect channel systems* have been extensively studied as a formal model for communicating protocols [15, 28]. Unfortunately, perfect channel systems are in general Turing powerful, and hence all basic decision problems (e.g., the reachability problem) are undecidable for them [15]. To circumvent this undecidability obstacle, several approximate techniques have been proposed in the literature including making the channels lossy [4, 17], restricting the communication topology to polyforest architectures [28, 26], or using half-duplex communication [16]. The decidability of the reachability problem can be also obtained by restricting the analysis to only executions performing at most some fixed number of context switches (where in each context only one process is allowed to receive messages from one channel while being able to send messages to all the other channels) [26]. Another well-known technique used in the verification of perfect channel systems is that of loop acceleration where the effect of iterating a loop is computed [13].

In this paper, we introduce the model of *Communicating Timed Automata* (or CTA for short) which extends the classical models of finite-state processes communicating through FIFO perfect channels and discrete timed automata, in the sense that the finite-state processes are replaced by discrete timed automata, and messages inside the perfect channels are equipped with discrete clocks representing their ages. In addition to the standard operations of timed automaton, each automaton can either (1) append a message to the tail of a channel with an initial age or (2) receive the message at the head of a channel if its age satisfies a set of given constraints. In a timed transition, the clock values and the ages of all the messages inside the perfect channels are increased uniformly. Thus, the CTA model subsumes both discrete timed automata and perfect channel systems. More precisely, we obtain the latter if we do not allow the CTA to use the timed information (i.e., all the timing constraints trivially hold); and we obtain the former if we do not use the perfect channels (no message is sent or received from the channels). Observe that a CTA is infinite in multiple dimensions, namely we have a number of channels that may contain an unbounded number of messages each of which is equipped with a natural number.

The CTA model can be used as a formal model for some safety critical devices such as implantable cardiac medical devices [24] in which the heart and the pacemaker can be modelled using two timed automata communicating through perfect channels and global variables. Another application of the CTA model is the modelling of distributed systems consisting of several servers. Each server has its own local clocks. The servers communicate with each other using perfect channels and use their local clocks to timestamp the exchanged messages. In general distributed systems avoid the use of global clocks (for performance reasons) but in certain cases these global clocks are needed to enforce the consistency of the data across the servers. This is the case for instance with ***Spanner***, Google's global SQL database. Spanner time-stamps all data written to it and allows global consistency of reads across the entire database. Data consistency is then achieved in Spanner via the use of TrueTime, a global synchronized clock across the data centres. The global clock helps in ensuring that for two transactions T_1, T_2 taking place, say in Australia and the East Coast respectively, if T_2 starts a commit after T_1 has already committed, then the timestamp for T_2 is greater than the timestamp for T_1.

We show that the reachability problem is undecidable even in the case of two timed automata connected by one unidirectional timed channel if one allows global clocks. We prove that this undecidability still holds even for CTA consisting of three timed automata and two unidirectional timed channels (and without any global clock). However, the reachability problem becomes decidable (in EXPTIME) in the case of two automata linked with one unidirectional timed channel and with no global clock. Finally, we consider the bounded-context case, where in each context only one timed automaton is allowed to receive messages from one channel while being able to send messages to all the other timed channels. In this case we show that the reachability is decidable. This is quite surprising since the reachability problem for unidirectional polyforest architectures can be easily reduced to its corresponding problem in the bounded-context case in the untimed settings.

Related Work

Several extensions of infinite-state systems with time behaviours have been proposed in the literature (e.g., [3, 2, 9, 1, 29, 12, 22, 21, 18, 25, 20, 19, 11, 7, 5, 23, 14, 6, 10]). The two closest to ours are those presented in [18, 25]. Both works extend perfect channel systems with time behaviours but do not associate a clock to each message (i.e., the content of each channel is still a word over a finite alphabet) as in our case. The work presented [18] shows that the reachability problem is decidable if and only if the communication topology is a polyforest while for our model the reachability problem is undecidable for polyforest architectures in general. Furthermore, there is no simple reduction of our results to the results presented in [18]. The work presented in [25] considers dense clocks with urgent semantics. In [25], the authors show (as in our model) that the reachability problem is undecidable for three timed automata and two unidirectional timed channels; while it becomes decidable when considering two automata linked with one unidirectional timed channel. However, the used techniques show that these results are quite different since we do not allow the urgent semantics.

Acyclic CTA	Global clocks	Channels	Reachability	Where
2-CTA, discrete time	Yes (1 global clock)	1	Undecidable	Corollary 2
3-CTA, discrete time	No	2	Undecidable	Theorem 3
2-CTA, discrete time	No	1	Decidable	Theorem 5
*-CTA, discrete time bounded context	Yes	any	Decidable	Theorem 9
2-CTA, dense time	No	1	Open	
*-CTA, dense time bounded context	No	any	Decidable?	

Table 1 Summary of results. k-CTA represents CTA with k timed automata, $k \in \mathbb{N}$. In *-CTA, we do not bound the number of timed automata involved.

2 Preliminaries

In this section, we introduce some notations and preliminaries which will be used throughout the paper. We use standard notation $\mathbb{N}$ for the set of naturals, along with ∞. Let $\mathcal{X}$ be a finite set of variables called *clocks*, taking on values from $\mathbb{N}$. A *valuation* on $\mathcal{X}$ is a function $\nu : \mathcal{X} \to \mathbb{N}$. We assume an arbitrary but fixed ordering on the clocks and write x_i for the clock with order i. This allows us to treat a valuation ν as a point $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n)) \in \mathbb{N}^{|\mathcal{X}|}$. For a subset of clocks $X \in 2^{\mathcal{X}}$ and valuation $\nu \in \mathbb{N}^{|\mathcal{X}|}$, we write $\nu[X{:=}0]$ for the valuation where $\nu[X{:=}0](x) = 0$ if $x \in X$, and $\nu[X{:=}0](x) = \nu(x)$ otherwise. For $t \in \mathbb{N}$, write $\nu + t$ for the valuation defined by $\nu(x) + t$ for all $x \in X$. The valuation $\mathbf{0} \in \mathbb{N}^{|\mathcal{X}|}$ is a special valuation such that $\mathbf{0}(x) = 0$ for all $x \in \mathcal{X}$. A clock constraint over $\mathcal{X}$ is defined by a (finite) conjunction of constraints of the form $x \bowtie k$, where $k \in \mathbb{N}$, $x \in \mathcal{X}$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. We write $\varphi(\mathcal{X})$ for the set of clock constraints. For a constraint $g \in \varphi(\mathcal{X})$, and a valuation $\nu \in \mathbb{N}^{|\mathcal{X}|}$, we write $\nu \models g$ to represent the fact that valuation ν satisfies constraint g. For example, $(1, 0, 10) \models (x_1 < 2) \wedge (x_2 = 0) \wedge (x_3 > 1)$.

Timed automata

Let *Act* denote a finite set called actions. A timed automaton (TA) is a tuple $\mathcal{A} = (L, L^0, Act, \mathcal{X}, E, F)$ such that

- L is a finite set of locations,
- $\mathcal{X}$ is a finite set of clocks,
- *Act* is a finite alphabet called an action set,
- $E \subseteq L \times \varphi(\mathcal{X}) \times Act \times 2^{\mathcal{X}} \times L$ is a finite set of transitions, and
- $L^0, F \subseteq L$ are respectively the sets of initial and final locations and *Act* is a finite set of actions.

A state s of a timed automaton is a pair $s = (\ell, \nu) \in L \times \mathbb{N}^{|\mathcal{X}|}$. A transition (t, e) from a state $s = (\ell, \nu)$ to a state $s' = (\ell', \nu')$ is written as $s \stackrel{t,e}{\to} s'$ if $e = (\ell, g, a, Y, \ell') \in E$, such that $a \in Act$, $\nu + t \models g$, and $\nu' = (\nu + t)[Y{:=}0]$. A run is a finite sequence $\rho = s_0 \stackrel{t_1,e_1}{\to} s_1 \stackrel{t_2,e_2}{\to} s_2 \cdots \stackrel{t_n,e_n}{\to} s_n$ of states and transitions. $\mathcal{A}$ is non-empty iff there is a run from an initial state $(l_0, \mathbf{0})$ to some state (f, ν) where $f \in F$. Note that we have defined discrete timed automata, a subclass of Alur-Dill automata [8], where clocks assume only integral values.

Region Automata

If $\mathcal{A}$ is a timed automaton, the region automaton corresponding to $\mathcal{A}$ denoted by $Reg(\mathcal{A})$ is an untimed automaton defined as follows. Let K be the maximal constant used in the constraints of A and let $[K] = \{0, 1, \ldots, K, \infty\}$. The locations of $Reg(\mathcal{A})$ are of the form $L \times [K]^{|\mathcal{X}|}$. The set of initial locations of $Reg(\mathcal{A})$ is $L_0 \times \mathbf{0}$. The transitions in $Reg(\mathcal{A})$ are of the following kinds: (i) $(l, \nu) \xrightarrow{\checkmark} (l, \nu + 1)$ denotes a time elapse of 1. If $\nu(x) + 1$ exceeds K for any clock x, then it is replaced with ∞. (ii) For each transition $e = (\ell, g, a, Y, \ell')$, we have the transition $(l, \nu) \xrightarrow{a} (l', \nu')$ if $\nu \models g$, $\nu' = \nu[Y{:=}0]$. It is known [8] that $Reg(\mathcal{A})$ is empty iff $\mathcal{A}$ is.

3 Communicating Timed Automata (CTA)

A communicating timed automata (CTA) $\mathcal{N} = (\mathcal{A}_1, \ldots, \mathcal{A}_n, C, \Sigma, \mathcal{T})$ consists of timed automata $\mathcal{A}_i$, a finite set C of FIFO *channels*, a finite set Σ called the *channel alphabet*, and a *network topology* $\mathcal{T}$. The network topology is a directed graph $(\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}, C)$ comprising of the finite set of timed automata $\mathcal{A}_i$ as nodes, and the channels C as edges. C is given as a tuple $(c_{i,j})$; the channel from $\mathcal{A}_i$ to $\mathcal{A}_j$ is denoted by $c_{i,j}$, with the intended meaning that $\mathcal{A}_i$ writes a message from Σ to channel $c_{i,j}$ and $\mathcal{A}_j$ reads from channel $c_{i,j}$. We assume that there is atmost one channel $c_{i,j}$ from $\mathcal{A}_i$ to $\mathcal{A}_j$, for any pair $(\mathcal{A}_i, \mathcal{A}_j)$ of timed automata. Figure 1 illustrates the definition.

Each timed automaton $\mathcal{A}_i = (L_i, L_i^0, Act, \mathcal{X}_i, E_i, F_i)$ in the CTA is as explained before, with the only difference being in the transitions E_i. We assume that $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$ for $i \neq j$. A transition in E_i has the form (l_i, g, op, Y, l_i') where g, Y have the same definition as in that of a timed automaton, while $op \in Act$ is one of the following operation on the channels $c_{i,j}$:

1. nop is an empty operation that does not check or update the channel contents. Transitions having the empty operation nop are called *internal transitions.* Internal transitions of $\mathcal{A}_i$ do not change any channel contents.
2. $c_{i,j}!a$ is a write operation on channel $c_{i,j}$. The operation $c_{i,j}!a$ appends the message $a \in \Sigma$ to the tail of the channel $c_{i,j}$, and sets the age of a to be 0. The timed automaton $\mathcal{A}_i$ moves from location l_i to l_i', checking guard g, resetting clocks Y and writes message a on channel $c_{i,j}$.
3. $c_{j,i}?(a{\in}I)$ is a read operation on channel $c_{j,i}$. The operation $c_{j,i}?(a{\in}I)$ removes the message a from the head of the channel $c_{j,i}$ if its age lies in the interval I. The interval I has the form $<\ell, u>$ with $u \in \mathbb{N}$ and $\ell \in \mathbb{N}\backslash\{\infty\}$, "$<$" stands for left-open or left-closed and "$>$" for right-open or right-closed. In this case, the timed automaton $\mathcal{A}_i$ moves from location l_i to l_i', checking guard g, resetting clocks Y and reads off the oldest message a from channel $c_{j,i}$ if its age is in interval I.

Global Clocks. A clock x is said to be global in a CTA if it can be checked any of the timed automata in the CTA, and can also be reset by any of them on a transition. Note that if a clock x is not global, then it can be checked and reset only by the automata which "owns" it. The automaton A_i owns x iff $x \in \mathcal{X}_i$ (recall that $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$). The convention $\mathcal{X}_i \cap \mathcal{X}_j = \emptyset$ applies to non-global (or local) clocks. Thus, if a CTA consisting of automata $A_1, \ldots, A_n$ has global clocks, then its set of clocks can be thought of as $\biguplus \mathcal{X}_i \uplus \mathcal{G}$ where $\mathcal{G}$ is a set of global clocks, which are accessed by all of $A_1, \ldots, A_n$, while clocks of $\mathcal{X}_i$ are accessible only to A_i.

Configurations

The semantics of $\mathcal{N}$ is given by a labeled transition system $\mathcal{L}_{\mathcal{N}}$. A configuration γ of $\mathcal{N}$ is a tuple $((l_i, \nu_i)_{1 \leq i \leq n}, c)$ where l_i is the current control location of $\mathcal{A}_i$, and ν_i gives the valuations of clocks $\mathcal{X}_i$, $1 \leq i \leq n$, where $\nu_i \in \mathbb{N}^{|\mathcal{X}_i|}$. $c = (c_{i,j})$, and each channel $c_{i,j}$ is represented as a monotonic timed word $(a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$ where $a \in \Sigma$ and $t_i \leq t_{i+1}$, and $t_i \in \mathbb{N}$. Given a word $c_{i,j}$ and a time $t \in \mathbb{N}$, $c_{i,j} + t$ is obtained by adding t to the ages of all messages in channel $c_{i,j}$. For $c = (c_{i,j})$, $c + t$ denotes the tuple $(c_{i,j} + t)$. The states of $\mathcal{L}_{\mathcal{N}}$ are the configurations.

Transition Relation of $\mathcal{L}_\mathcal{N}$

Let $\gamma_1 = ((l_1, \nu_1), \ldots, (l_n, \nu_n), c)$ and $\gamma_2 = ((l'_1, \nu'_1), \ldots, (l'_n, \nu'_n), c')$ be two configurations. The transitions $\rightarrow$ in $\mathcal{L}_\mathcal{N}$ are of two kinds:

1. Timed transitions $\xrightarrow{t}$: These transitions denote the passage of time $t \in \mathbb{N}$. $\gamma_1 \xrightarrow{t} \gamma_2$ iff $l_i = l'_i$, and $\nu'_i = \nu_i + t$, for all i and $c' = c + t$.
2. Discrete transitions $\xrightarrow{D}$. These are of the following kinds:
   (1) $\gamma_1 \xrightarrow{g, \mathsf{nop}, Y} \gamma_2$: there is a transition $l_i \xrightarrow{g, \mathsf{nop}, Y} l'_i$ in E_i, $\nu_i \models g$, $\nu'_i = \nu_i[Y := 0]$, for some i. Also, $l_k = l'_k$, $\nu_k = \nu'_k$ for all $k \neq i$, and $c_{d,h} = c'_{d,h}$ for all d, h. None of the channel contents are changed.
   (2) $\gamma_1 \xrightarrow{g, c_{i,j}!a, Y} \gamma_2$: Then, $l_k = l'_k$, $\nu_k = \nu'_k$ for all $k \neq i$, and $c_{d,h} = c'_{d,h}$ for all $(d, h) \neq (i, j)$. The transition $l_i \xrightarrow{g, c_{i,j}!a, Y} l'_i$ is in E_i, $\nu_i \models g$, $\nu'_i = \nu_i[Y := 0]$, $c_{i,j} = w \in (\Sigma \times \mathbb{N})^*$ and $c'_{i,j} = (a, 0).w$.
   (3) $\gamma_1 \xrightarrow{g, c_{j,i}?(a \in I), Y} \gamma_2$: Then, $l_k = l'_k$, $\nu_k = \nu'_k$ for all $k \neq i$, and $c_{d,h} = c'_{d,h}$ for all $(d, h) \neq (j, i)$. The transition $l_i \xrightarrow{g, c_{j,i}?(a \in I), Y} l'_i$ is in E_i, $\nu_i \models g$, $\nu'_i = \nu_i[Y := 0]$, $c_{j,i} = w.(a, t) \in (\Sigma \times \mathbb{N})^+$, $t \in I$ and $c'_{j,i} = w \in (\Sigma \times \mathbb{N})^*$.

The Reachability Problem

The initial location of $\mathcal{L}_\mathcal{N}$ is given by the tuple $\gamma_0 = ((l_1^0, \nu_1^0), \ldots, (l_n^0, \nu_n^0), c^0)$ where l_i^0 is the initial location of A_i, $\nu_i^0 = \mathbf{0}$ for all i, and c^0 is the tuple of empty channels $(\epsilon, \ldots, \epsilon)$. A control location $l_i \in L_i$ is reachable if $\gamma_0 \xrightarrow{*} ((s_i, \nu_i)_{1 \leq i \leq n}, c)$ such that $s_i = l_i$ (It does not matter what $(\nu_1, \ldots, \nu_n)$ and c are). An instance of the reachability problem asks whether given a CTA $\mathcal{N}$ with initial configuration γ_0, we can reach a configuration γ.

4 Acyclic CTA

In this section, we look at the reachability problem in CTA whose underlying network topology $\mathcal{T}$ is somewhat restrictive. An *acyclic CTA* is a CTA $\mathcal{N} = (A_1, \ldots, A_n, C, \Sigma, \mathcal{T})$ which has no cycles in the underlying undirected graph of $\mathcal{T}$[1]. Such topologies are called polyforest topologies in [26] (left of Figure 1). In this section, we answer the reachability question in acyclic CTA with and without global clocks by finding the thin boundary line which separates decidable and undecidable acyclic CTAs.

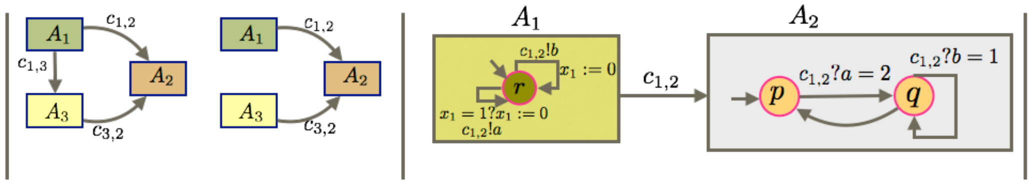


Figure 1 The left half of the figure contains one cyclic and one acyclic topology. The right half of the figure illustrates an acyclic CTA which is not bounded context.

---

[1] Recall that the network topology $(\{A_1, \ldots, A_n\}, C)$ is a directed graph; the underlying undirected graph is obtained by considering all edges as undirected in this graph.

4.1 Undecidable Reachability with Global Clocks

Theorem 1. *In the presence of global clocks, reachability is undecidable for CTA consisting of two timed automata A_1, A_2 connected by a single channel.*

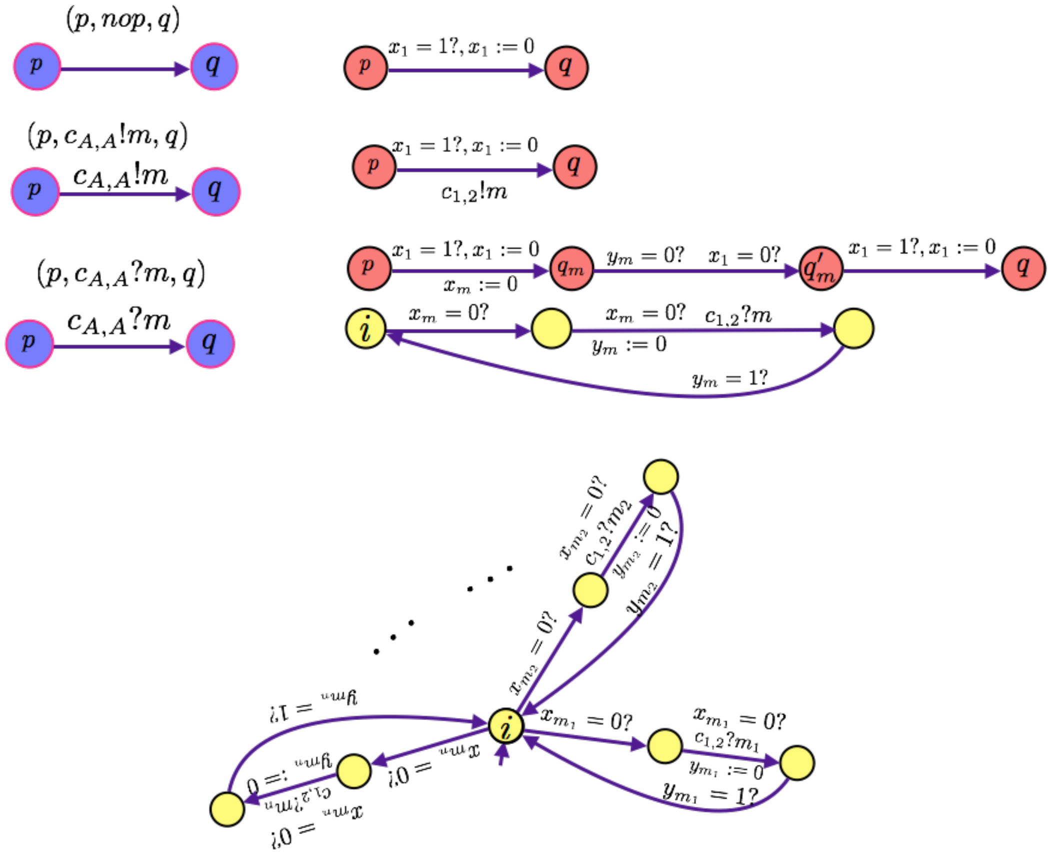


Figure 2 Above left, we show each transition in A (*nop* and write transitions) and the corresponding widget in A_1. A read transition in A has widgets in A_1, A_2. The timed automata A_1, A_2 are obtained by connecting all these widgets. Below, is the automaton A_2 of the CTA, assuming the message alphabet is $\{m_1, \ldots, m_n\}$.

Proof. It is known [26] that if one considers a single untimed automaton A communicating to itself via a perfect, FIFO channel, the reachability is undecidable. Our undecidability result is built via a reduction from this problem. We show that global clocks can simulate the "self-loop" channel which behaves like a pump.

Given an untimed automaton A communicating to itself using channel $c_{A,A}$, we build a CTA $\mathcal{N}$ consisting of two timed automata A_1, A_2 with a channel $c_{1,2}$ from A_1 to A_2. Each time A writes into $c_{A,A}$, A_1 writes into channel $c_{1,2}$. Assume that A reads message m from $c_{A,A}$. Since A_1 cannot read message m from channel $c_{1,2}$, A_1 sets a special clock say x_m to 0 (note that x_m is not zero otherwise, since any other transition is guarded by $x_1 = 1$). A read transition is triggered in A_2 when x_m is 0; A_2 reads off the message m from the head of the channel, and sets a clock y_m to 0, signifying that it has read m. A_1 checks if y_m is 0, and if so, proceeds to the next transition. See Figure 2 : on the top left are transitions of A; on the top right, we depict corresponding transitions in A_1 (the red states) and in A_2 (yellow states). For nop and write transitions of A, there are no corresponding widgets in A_2; read transitions of A have corresponding widgets in both A_1 and A_2.

See Appendix A for a detailed proof of Theorem 1.

□

Corollary 2. *The number of global clocks used in the above proof is twice the size of the channel alphabet. However, we can see that a single global clock suffices for undecidability. We retain the above proof since it is easier. The single global clock undecidability can be seen in Appendix B.*

4.2 Undecidable Reachability with no Global Clocks

Theorem 3. *Reachability is undecidable for acyclic CTA consisting of three one-clock timed automata without global clocks.*

Proof. We prove the undecidability by reducing the halting problem for deterministic two counter machines. We consider the case of a CTA consisting of timed automata A_1, A_2, A_3 with channels $c_{1,2}$ from A_1 to A_2 and $c_{2,3}$ from A_2 to A_3. The undecidability for the other possible topologies are discussed in Appendix C.3.

4.2.1 Counter Machines

A two-counter machine $\mathcal{C}$ is a tuple $(L, \{c_1, c_2\})$ where $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$ is the set of instructions—including a distinguished terminal instruction ℓ_n called HALT—and $\{c_1, c_2\}$ are the two *counters*. The instructions in L are one of: (i) (increment c by 1) ℓ_i:inc c; goto ℓ_k, (ii) (decrement c by 1) ℓ_i:dec c; goto ℓ_k, (iii) (zero-check c) ℓ_i: if ($c>0$) then goto ℓ_k else goto ℓ_m, (iv) (Halt) ℓ_n : HALT, where $c \in \{c_1, c_2\}$, $\ell_i, \ell_k, \ell_m \in L$. A configuration of a two-counter machine is a tuple (l, c, d) where $l \in L$ is an instruction, and c, d are natural numbers that specify the value of counters c_1 and c_2, respectively. The initial configuration is $(\ell_0, 0, 0)$. The transition relation is the standard one for Minsky machines. The *halting problem* for a two-counter machine asks whether its unique run starting at $(\ell_0, 0, 0)$ ends at (ℓ_n, n_1, n_2) for some $n_1, n_2 \in \mathbb{N}$. It is well known ([27]) that this problem is undecidable.

4.2.2 The Encoding

Given a two counter machine $\mathcal{C}$, we build a CTA $\mathcal{N}$ consisting of timed automata A_1, A_2, A_3 with channels $c_{1,2}$ from A_1 to A_2 and $c_{2,3}$ from A_2 to A_3. Corresponding to each increment, decrement and zero check instruction, we have a widget in each A_i. A widget is a "small" timed automaton, consisting of some locations and transitions between them. Corresponding to each increment/decrement instruction ℓ_i : inc or dec c, goto ℓ_j, or a zero check instruction ℓ_i : if $c = 0$, goto ℓ_j else goto ℓ_k, we have a widget $\mathcal{W}_i^{A_m}$ in each $A_m, m \in \{1, 2, 3\}$. The widgets $\mathcal{W}_i^{A_m}$ begin in a location labelled ℓ_i, and terminate in a location ℓ_j for increments/decrements, while for zero check, they begin in a location labelled ℓ_i, and terminate in a location ℓ_j or ℓ_k. Each A_m is hence obtained by superimposing (one of) the terminal location ℓ_j of a widget $\mathcal{W}_i^{A_m}$ to the initial location ℓ_j of widget $\mathcal{W}_j^{A_m}$.

We refer to initial/terminal locations (labelled p) in each $\mathcal{W}_i^{A_m}$ using the notation $(\mathcal{W}_i^{A_m}, p)$. Note that an instruction ℓ_i can appear as initial location in a widget and a terminal location in another; thus, it is useful to remember the location along with the widget we are talking about. x_1, y_1, z_1 respectively denote the clocks used in A_1, A_2, A_3. To argue the proof of correctness, we use clocks $g_{A_1}, g_{A_2}, g_{A_3}$ respectively in A_1, A_2, A_3 which are never used in any transitions (hence g_{A_i} represent the total time elapse at any point in A_i).

4.2.2.1 Counter Values.

The value of counter c_1 after i steps, denoted c_1^i is stored as the difference between the value of clock g_{A_2} after i steps and the value of clock g_{A_1} after i steps. Denoting l_i to be the instruction reached after i steps, and thanks to the fact that we have locations l_i in each of A_1, A_2, A_3 corresponding to the instruction l_i, the value c_1^i=(value of clock g_{A_2} at location l_i of A_2) - (value of clock g_{A_1} at location l_i of A_1). Note that A_1, A_2 are not always in sync while simulating the two counter machine : A_1 can simulate the jth instruction l_j while A_2 is simulating the ith instruction l_i for $j \geq i$, thanks to the invariant maintaining the value of c_1. When they are in sync, the value of c_1 is 0. Thus, A_1 is always ahead of A_2 or at the same step as A_2 in the simulation. The value of counter c_2 is maintained in a similar manner by A_2 and A_3. To maintain the values of c_1, c_2 correctly, the speeds of A_1, A_2, A_3 are adjusted while doing increments/decrements. For instance, to increment c_1, A_2 takes 2 units of time to go from ℓ_i to ℓ_j while A_1 takes just one unit; then the value of g_{A_2} at ℓ_j is two more than what it was at ℓ_i; likewise, the value of g_{A_1} at ℓ_j is one more than what it was at ℓ_i. The channel alphabet is $\{(\ell_i, c^+, \ell_j) \mid \ell_i : \mathsf{inc}\ c\ \mathsf{goto}\ \ell_j\} \cup \{(\ell_i, c^-, \ell_j) \mid \ell_i : \mathsf{dec}\ c\ \mathsf{goto}\ \ell_j\} \cup \{(\ell_i, c{=}0, \ell_j), (\ell_i, c{>}0, \ell_k) \mid \ell_i : \mathsf{if}\ c = 0,\ \mathsf{then\ goto}\ \ell_j,\ \mathsf{else\ goto}\ \ell_k\} \cup \{zero_1, zero_2\}$.

1. Consider an increment instruction ℓ_i:inc c goto ℓ_j. The widgets $\mathcal{W}_i^{A_m}$ for $m = 1, 2, 3$ are described in Figure 3. The one on the left is while incrementing c_1, while the one on the right is obtained while incrementing c_2.

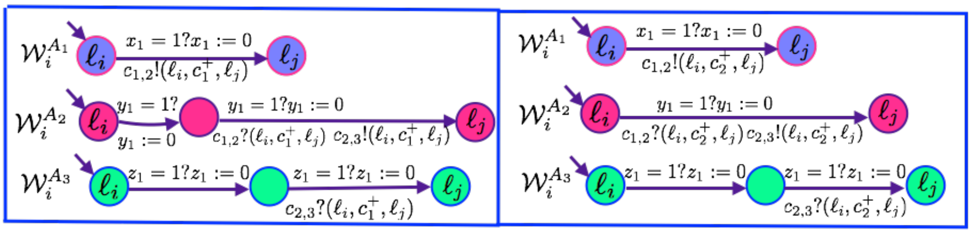


Figure 3 Widgets corresponding to an increment c_1, c_2 instruction in A_1, A_2, A_3

2. The case of a decrement instruction is similar, and is obtained by swapping the speeds of the two automata (A_1, A_2 and A_2, A_3 respectively) in reaching ℓ_j from ℓ_i (see Figure 10). Note that we preserve the invariant that A_1 is ahead of (or same as) A_2 which is ahead of (or same as) A_3 in the simulation of the two counter machine.
3. We finally consider a zero check instruction of the form ℓ_i:if c_1=0, then goto ℓ_j, else goto ℓ_k. The widgets $\mathcal{W}_i^{A_m}$ for m=1, 2, 3 are described in Figure 4. The one on the left is a zero check of c_1, while the one on the right is a zero check of c_2.

Let $(\ell_0, 0, 0), (\ell_1, c_1^1, c_2^1), \ldots, (\ell_h, c_1^h, c_2^h) \ldots$ be the run of the two counter machine. ℓ_i denotes the instruction seen at the ith step and c_1^i, c_2^i respectively are the values of counters c_1, c_2 after i steps. Denote a block of transitions in A_m leading from the ith to the $(i{+}1)$st instruction as $\mathcal{B}_{i,i+1} = [((\mathcal{W}_i^{A_m}, \ell_i), \nu_i^{A_m}), \ldots, ((\mathcal{W}_i^{A_m}, \ell_{i+1}), \nu_{i+1}^{A_m})]$. A run in each A_m is $\mathcal{B}_{0,1}, \mathcal{B}_{1,2}, \ldots, \mathcal{B}_{h,h+1}, \ldots$, where each block $\mathcal{B}_{h,h+1}$ of transitions in the widget $\mathcal{W}_h^{A_m}$ simulate the instruction ℓ_h, and shifts control to ℓ_{h+1}. For each m, $((\mathcal{W}_i^{A_m}, \ell_j), \nu_j^{A_m})$ represents A_m is at location ℓ_j of widget $\mathcal{W}_i^{A_m}$ with clock valuation $\nu_j^{A_m}$.

Lemma 4. *Let $\mathcal{C}$ be a two counter machine. Let c_1^h, c_2^h be the values of counters c_1, c_2 at the end of the hth instruction ℓ_h. Then there is a run of $\mathcal{N}$ which passes through widgets $\mathcal{W}_0^{A_m}, \mathcal{W}_1^{A_m}, \ldots, \mathcal{W}_h^{A_m}$ in $A_m, m \in \{1, 2, 3\}$ such that*

1. c_1^h is the difference between the value of clock g_{A_2} on reaching the initial location $(\mathcal{W}_h^{A_2}, \ell_h)$ and the value of clock g_{A_1} on reaching the initial location $(\mathcal{W}_h^{A_1}, \ell_h)$. c_2^h is the difference

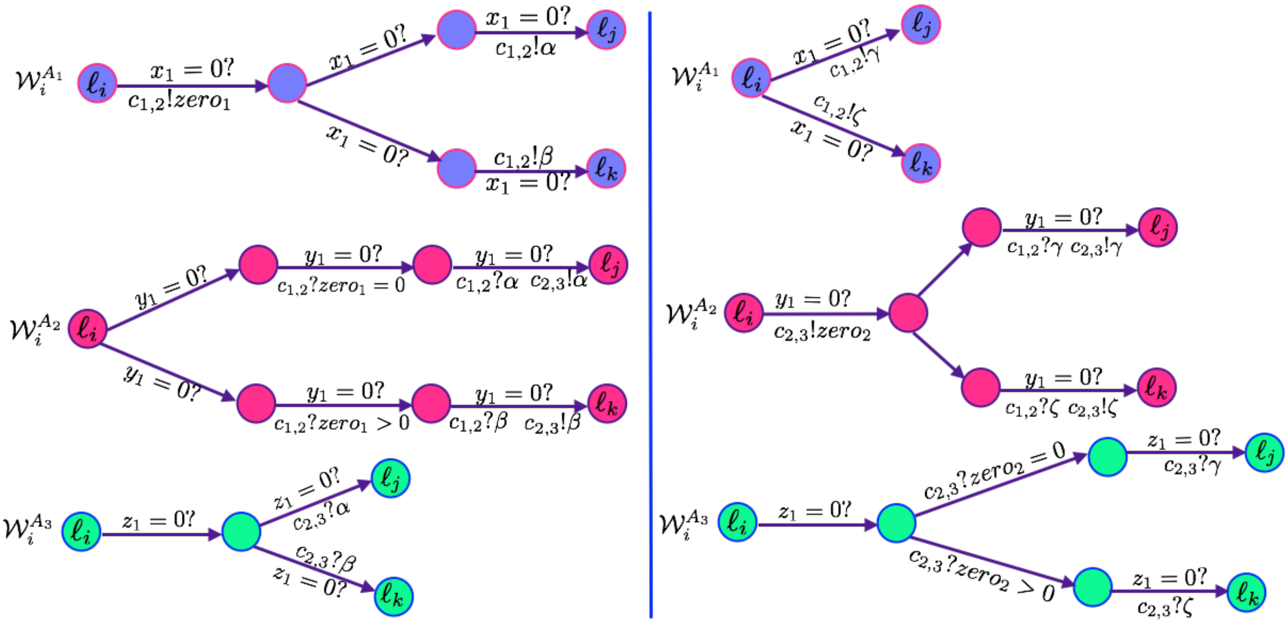


Figure 4 Widgets corresponding to checking c_1, c_2 is 0. Let $\alpha{=}(\ell_i, c_1{=}0, \ell_j), \beta{=}(\ell_i, c_1{>}0, \ell_k), \gamma{=}(\ell_i, c_2{=}0, \ell_j), \zeta{=}(\ell_i, c_2{>}0, \ell_k)$.

between the value of clock g_{A_3} on reaching the initial location $(\mathcal{W}_h^{A_3}, \ell_h)$ and the value of clock g_{A_2} on reaching the initial location $(\mathcal{W}_h^{A_2}, \ell_h)$.

2. *If $\mathcal{W}_h^{A_1}$ is a zero check widget for c_1 (c_2) then c_1^h (c_2^h) is 0 iff one reaches a terminal location of $\mathcal{W}_h^{A_2}$ reading α (γ) and $zero_1$ ($zero_2$) with age 0. Likewise, $c_1^h(c_2^h)$ is > 0 iff one reaches a terminal location of $\mathcal{W}_h^{A_2}$ reading β (ζ) and $zero_1$ ($zero_2$) with age > 0.*

Machine $\mathcal{C}$ halts iff the halt widget $\mathcal{W}_{halt}^{A_m}$ is reached in $\mathcal{N}$, $m{=}1, 2, 3$: Appendix C has the full proof. ◻

4.3 Decidable Reachability

Theorem 5. *The reachability problem is decidable (in* EXPTIME*) for acyclic CTA consisting of two timed automata without global clocks.*

The proof proceeds by a reachability preserving reduction of the CTA to a one counter automaton. We give the proof idea here, correctness arguments and an example can be found in Appendix D.

Given CTA $\mathcal{N}$ consisting of $A = (L_A, L_A^0, \mathcal{X}_A, \Sigma, E_A, F_A)$ and $B = (L_B, L_B^0, \mathcal{X}_B, \Sigma, E_B, F_B)$, with a channel $c_{A,B}$ from A to B, we simulate $\mathcal{N}$ using a one counter automaton $\mathcal{O}$ as follows.

Intermediate Notations

We start with $Reg(A)$ and $Reg(B)$, the corresponding region automata, and run them in an interleaved fashion. Let K be the maximal constant used in the guards of A, B. Let $[K] = \{0, 1, \ldots, K, \infty\}$. The locations Q_A (Q_B) of $Reg(A)$ ($Reg(B)$) are of the form $L_A \times [K]^{|\mathcal{X}_A|}$ ($L_B \times [K]^{|\mathcal{X}_B|}$).

Transitions in $Reg(A), Reg(B)$

(i) A transition $(l,\nu) \xrightarrow{\checkmark} (l,\nu{+}1)$ denotes a time elapse of 1 in both $Reg(A), Reg(B)$. If $\nu(x){+}1$ exceeds K for any clock x, then it is replaced with ∞. (ii) For each transition $e = (\ell, g, c_{A,B}!a, Y, \ell')$ in A we have the transition $(l,\nu) \xrightarrow{a} (l',\nu')$ in $Reg(A)$ if $\nu \models g$, and $\nu' = \nu[Y{:=}0]$. (iii) For each transition $e = (\ell, g, c_{A,B}?(a \in I), Y, \ell')$ in B we have the transition $(l,\nu) \xrightarrow{a \in I} (l',\nu')$ in $Reg(B)$ if $\nu \models g$, and $\nu' = \nu[Y{:=}0]$. (iv) For each internal transition $e = (\ell, g, \mathsf{nop}, Y, \ell')$ in A, B we have the transition $(l,\nu) \xrightarrow{\mathsf{nop}} (l',\nu')$ in $Reg(A), Reg(B)$ if $\nu \models g$, and $\nu' = \nu[Y{:=}0]$. Note that the above is an intermediate notation which will be used in the construction of the one-counter automaton $\mathcal{O}$. There is no channel between $Reg(A), Reg(B)$, and we have symbolically encoded all transitions of A, B in $Reg(A), Reg(B)$ as above.

Construction of $\mathcal{O}$

In the reduction from CTA $\mathcal{N}$ to the one counter automaton $\mathcal{O}$, the global time difference between A and B is stored in the counter, such that B is always ahead of A, or at the same time as A. Thus, a counter value $i \geq 0$ means that B is i units of time ahead of A. The state space of $\mathcal{O}$ is constructed using the locations of $Reg(A), Reg(B)$, and the transitions of $\mathcal{O}$ will make use of the transitions described above of $Reg(A), Reg(B)$. Internal transitions of A, B are simulated by updating the respective control locations in $Reg(A), Reg(B)$. Each unit time elapse in B results in incrementing the counter by 1, while each unit time elapse in A results in decrementing the counter. Consider a transition in A where a message m is written on the channel. The counter value when m is written tells us the time difference between B, A, and hence also the age of the message as seen from B. Assume the counter value is $i \geq 0$. If indeed m must be read in B when its age is exactly i, then B can move towards a transition where m is read, without any further time elapse. In case m must be read when its age is $j > i$, then B can execute internal transitions as well a time elapse $j - i$ so that the transition to read m is enabled. However, if m must have been read when its age is some $k < i$, then B will be unable to read m. By our interleaved execution, each time A writes a message, we make B read it before A writes further messages, and proceed. Note that this does not disallow A writing multiple messages with the same time stamp.

Counter values $\leq K$ are kept as part of the finite control of $\mathcal{O}$, and when the value exceeds K, we use a unary stack with stack alphabet $\{1\}$ to keep track of the exact value $> K$. Note that we have to keep track of the exact time difference between B, A since otherwise we will not be able to check age requirements of messages correctly.

State Space of $\mathcal{O}$

Let $\hat{Q}_x = \{q_\perp, q_1, q'_\perp, q'_1 \mid q \in Q_x, x \in \{A, B\}\}$. Let $O_x = Q_x \cup Q_x^\perp$ for $x \in \{A, B\}$. $O_A \times (O_B \times (\Sigma \cup \{\epsilon\})) \times ([K] \backslash \{\infty\})$ is the state space of $\mathcal{O}$, where the $\Sigma \cup \{\epsilon\}$ in $(O_B \times (\Sigma \cup \{\epsilon\}))$ is to remember the message (if any) written by A, which has to be read by B, and the last entry in the triple denotes the counter value. The stack alphabet is $\{\perp, 1\}$. The initial location of $\mathcal{O}$ is $\{((l_A^0, 0^{|\mathcal{X}_A|}), (l_B^0, 0^{|\mathcal{X}_B|}, \epsilon), 0) \mid l_A^0 \in L_A^0, l_B^0 \in L_B^0\}$ and the unary stack has the bottom of stack symbol $\perp$ in the initial configuration.

Transitions in $\mathcal{O}$

The transitions in $\mathcal{O}$ are as follows : For l, l' states of $\mathcal{O}$, internal transitions Δ_{int} consist of transitions of the form (l, l'); push transitions Δ_{push} consist of transitions of the form (l, a, l') for $a \in \{1, \perp\}$. Finally, we also have pop transitions Δ_{pop} of the form (l, a, l') for $a \in \{1, \perp\}$. We now describe the transitions.

1. Pop transitions Δ_{pop} : Pop transitions simulate time elapse in $Reg(A)$ as well as checking the age of a symbol being K or $> K$ while it is read from the channel.

(a) If $(p, \nu_1) \xrightarrow{\checkmark} (p, \nu_1 + 1)$ in $Reg(A)$, and if the counter value as stored in the finite control is K, and if the stack is non-empty, then we pop the top of the stack to decrement the counter. For $l = ((p, \nu_1), (q, \nu_2, \alpha), K)$, $l' = ((p, \nu_1 + 1), (q, \nu_2, \alpha), K)$, $(l, 1, l') \in \Delta_{pop}$.

(b) If $(p, \nu_1) \xrightarrow{\checkmark} (p, \nu_1 + 1)$ in $Reg(A)$, and if the counter value as stored in the finite control is K, and if the stack is empty, we pop $\perp$, reduce K in the finite control to $K - 1$, and push back $\perp$ to the stack. We remember that $\perp$ has been popped in the finite control, so that we push it back immediately. For $l = ((p, \nu_1), (q, \nu_2, \alpha), K)$, $l' = ((p_\perp, \nu_1 + 1), (q, \nu_2, \alpha), K - 1)$, $(l, \perp, l') \in \Delta_{pop}$. The location $p_\perp$ tells us that $\perp$ has to be pushed back immediately.

(c) To check that a message has age K when read, we need $i = K$, along with the fact that the stack is empty (top of stack=$\perp$). In this case, we pop $\perp$ and remember it in the finite control, and push it back. For $l = ((p, \nu_1), (q, \nu_2, \alpha), K)$, $l' = ((p, \nu_1), (q_\perp, \nu_2, \alpha), K)$, $(l, \perp, l') \in \Delta_{pop}$.

(d) To check that a message has age $> K$ when read, we need $i = K$, along with the fact that the stack is non-empty (top of stack=1). In this case, we pop 1 and remember it in the finite control, and push it back. For $l = ((p, \nu_1), (q, \nu_2, \alpha), K)$, $l' = ((p, \nu_1), (q_1, \nu_2, \alpha), K)$, $(l, 1, l') \in \Delta_{pop}$.

2. Push transitions Δ_{push} : Push transitions simulate time elapse in $Reg(B)$, and also aid in simulating checking the age of a symbol being K or $> K$ while being read from the channel.

(a) Push $\perp$ to the stack while reducing counter value from K to $K - 1$ (1(b)). For $l = ((p_\perp, \nu_1), (q, \nu_2, \alpha), K-1)$ and $l' = ((p, \nu_1), (q, \nu_2, \alpha), K-1)$, $(l, \perp, l') \in \Delta_{push}$.

(b) Push $\perp$ to the stack before checking the age of a message is K (1(c)). For $l = ((p, \nu_1), (q_\perp, \nu_2, \alpha), K)$ and $l' = ((p, \nu_1), (q'_\perp, \nu_2, \alpha), K))$, $(l, \perp, l') \in \Delta_{push}$.

(c) Push 1 to the stack before checking the age of a message is $> K$ (1(d)). For $l = ((p, \nu_1), (q_1, \nu_2, \alpha), K)$ and $l' = ((p, \nu_1), (q'_1, \nu_2, \alpha), K)$, $(l, 1, l') \in \Delta_{push}$.

(d) If $(q, \nu_2) \xrightarrow{\checkmark} (q, \nu_2 + 1)$ in $Reg(B)$, and if the counter value as stored in the finite control is K, then we push a 1 on the stack to represent the counter value is $> K$. That is, $(l, 1, l') \in \Delta_{push}$ for $l = ((p, \nu_1), (q, \nu_2, \alpha), K)$ and $l' = ((p, \nu_1), (q, \nu_2 + 1, \alpha), K)$.

3. Internal transitions Δ_{int}: Transitions of Δ_{int} simulate internal transitions of $Reg(A), Reg(B)$ as well as $\checkmark$- transitions as follows:

(a) Let $l = ((p, \nu_1), (q, \nu_2, \alpha), i)$, $l' = ((p', \nu'_1), (q, \nu_2, \alpha), i)$ be states of $\mathcal{O}$. $(l, l') \in \Delta_{int}$ if $(p, \nu_1) \xrightarrow{\mathsf{nop}} (p', \nu'_1)$ is an internal transition in $Reg(A)$. The same can be said of internal transitions in $Reg(B)$ updating q, ν_2, leaving α, i and (p, ν_1) unchanged.

(b) For $l = ((p, \nu_1), (q, \nu_2, \alpha), i)$ with $0 < i < K$, and $l' = ((p, \nu_1), (q, \nu_2 + 1, \alpha), i + 1)$, $(l, l') \in \Delta_{int}$ if $(q, \nu_2) \xrightarrow{\checkmark} (q, \nu_2 + 1)$ is a $\checkmark$-transition in $Reg(B)$. Note that $i + 1 \leq K$.

(c) For $l = ((p, \nu_1), (q, \nu_2, \alpha), i)$ with $0 < i < K$, and $l' = ((p, \nu_1 + 1), (q, \nu_2, \alpha), i - 1)$, $(l, l') \in \Delta_{int}$ if $(p, \nu_1) \xrightarrow{\checkmark} (p, \nu_1 + 1)$ is a $\checkmark$-transition in $Reg(A)$.

(d) For $l = ((p, \nu_1), (q, \nu_2, \epsilon), i)$, $l' = ((p', \nu'_1), (q, \nu_2, a), i)$, $(l, l') \in \Delta_{int}$ if $(p, \nu_1) \xrightarrow{a} (p', \nu'_1)$ is a transition in $Reg(A)$ corresponding to a transition from p to p' which writes a onto the channel $c_{A,B}$.

(e) For $i < K$, and $i \in I$, $l = ((p, \nu_1), (q, \nu_2, a), i)$, $l' = ((p, \nu_1), (q', \nu'_2, \epsilon), i)$, $(l, l') \in \Delta_{int}$ if $(q, \nu_2) \xrightarrow{a \in I} (q', \nu'_2)$ is a transition in $Reg(B)$ corresponding to a transition from q to q' which reads a from the channel $c_{A,B}$ and checks its age to be in interval I.

(f) To check that a message has age K when read, we need the counter value i to be K, along with the top of stack=$\perp$. See 1(c), 2(b), and then use transition $(l, l') \in \Delta_{int}$ for $l = ((p, \nu_1), (q'_\perp, \nu_2, m), K), l' = ((p, \nu_1), (r, \nu'_2, \epsilon), K)$, if $(q, \nu_2) \stackrel{m \in [K,K]}{\rightarrow} (r, \nu'_2)$ is a read transition in $Reg(B)$.

(g) To check that a message has age $> K$ when read, we need $i = K$, along with the fact that the stack is non-empty (top of stack=1). See 1(d), 2(c), and then $(l, l') \in \Delta_{int}$ for $l = ((p, \nu_1), (q'_1, \nu_2, m), K), l' = ((p, \nu_1), (r, \nu'_2, \epsilon), K)$, if $(q, \nu_2) \stackrel{m \in (K,\infty)}{\rightarrow} (r, \nu'_2)$ is a read transition in $Reg(B)$. (age requirements $\geq K$ are checked using this or the above).

The correctness of the construction is proved in Appendix D using Lemmas 6 and 7.

Lemma 6. *If $((l_A, \nu_A), (l_B, \nu_B, a), i)$ is a configuration in $\mathcal{O}$, along with a stack consisting of $1^j\perp$, then message a has age $i + j$, A is at l_A, B is at l_B, and B is $i + j$ time units ahead of A.*

Lemma 7. *Let $\mathcal{N}$ be a CTA with timed automata A, B connected by a channel $c_{A,B}$ from A to B. Assume that starting from an initial configuration $((l^0_A, 0^{|\mathcal{X}_A|}), (l^0_B, 0^{|\mathcal{X}_B|}), \epsilon)$ of $\mathcal{N}$, we reach configuration $((l_A, \nu_1), (l_B, \nu_2), w.(m, i))$ such that $w \in (\Sigma \times \{0, 1, \ldots, i\})^*$, and $(m, i) \in \Sigma \times [K]$ is read off by B from (l_B, ν_2). Then, from the initial configuration $((l^0_A, 0^{|\mathcal{X}_A|}), (l^0_B, 0^{|\mathcal{X}_B|}, \epsilon), 0)$ with stack contents $\perp$ of $\mathcal{O}$, we reach one of the following configurations*

(i) $((p_A, \nu'_A), (l_B, \nu_2, m), i)$ with stack contents $\perp$ if $i \leq K$,

(ii) $((p_A, \nu'_A), (l_B, \nu_2, m), h)$ with stack contents $1^j\perp$, $j > 0$ if $i > K$ and $h + j = i$.

Moreover, it is possible to reach (l_A, ν_1) from (p_A, ν'_A) in A after elapse of i units of time. The converse is also true.

Complexity : Upper and Lower bounds

The EXPTIME upper bound is easy to see, thanks to the exponential blow up incurred in the construction of $\mathcal{O}$ using the regions of A and B, and the fact that reachability in a push down automaton is linear. The best possible lower bound we can achieve as of now is NP-hardness, as described below.

The proof is by reduction from the subset sum problem. An instance of the subset sum problem consists of a set S of positive integers $S = \{a_1, a_2, \ldots, a_n\}$ and a number c. The question to be solved is whether there exists a subset T of S such that the sum of the elements of T is equal to c. Given S, we construct a CTA with processes A, B as follows. There is a channel $c_{A,B}$ from A to B, and the channel alphabet is S. A consists of locations s_{a_i} for $i = 1, \ldots n$ and hence has $|S|$ locations. There are no clocks in A. s_{a_1} is the unique initial location. The transitions of A are as follows. For all $1 \leq i \leq n - 1$, A writes a_i to the channel $c_{A,B}$ and goes from location s_{a_i} to location $s_{a_{i+1}}$. The final location is s_{a_n}. B has two clocks x, y, and has locations r_{a_i} for $i = 1, \ldots, n$ and a final location r_f. The initial location is r_{a_1}. Transitions in B are as follows. In location r_{a_i}, for $1 \leq i \leq n - 1$, B has the following transitions:

1. B reads a_i from the channel $c_{A,B}$ and checks if clock x is equal to a_i, and if so resets x, and proceeds to location $r_{a_{i+1}}$ for $1 \leq i \leq n - 1$,
2. B reads a_i from the channel $c_{A,B}$ and checks if clock x is equal to 0, and proceeds to location $r_{a_{i+1}}$ for $1 \leq i \leq n - 1$.

On reaching location r_{a_n}, we check if $x = 0$ and $y = c$, and if so, go to the final location r_f. It is clear that B spends time a_i at a location r_{a_i} if it wishes to add a_i to the sum. The clock y which is never reset, holds the sum. The final location is reached iff $y = c$.

5 Bounded Context Switching

In this section, we show that if one considers bounded context CTA, then the reachability problem is decidable even when having global clocks.

Given a CTA, a *context* is a sequence of transitions in the CTA where only one automaton is *active* viz., reading from atmost one fixed channel, but possibly writing to many channels that it can write to, except from the one it reads from (in case of self-loops in the topology). Thus, (a) a context is simply a sequence of transitions where a single automaton A_i performs channel operations, and (b) in a context, A_i can read from atmost one channel. A *context switch* happens when we have transitions $C_g \xrightarrow{+} C_i$ and $C_i \to C_{i+1}$ such that (a) or (b) is true.

(a) C_{i+1} is a configuration obtained when some automaton A_k performs some channel operation, and C_i is the configuration obtained by a channel operation in an automaton $A_t \neq A_k$, or, there is a configuration $C_g, g \leq i-1$, obtained by a channel operation in an automaton $A_t \neq A_k$, and the only channel operations in configurations $C_{g+1}, \ldots, C_i$ are by A_k when it reads from some fixed channel c or it writes to any channel other than c (if it reads from c). It is important that c is a fixed channel from which A_k reads (if it does) in configurations $C_{g+1}, \ldots, C_i, C_{i+1}$.

(b) In this case, assume there is a unique automaton A_k which is active and involved in channel operations in configurations $C_g, \ldots, C_i, C_{i+1}$. Let C_{i+1} be the configuration obtained when A_k reads from a channel c.
- The first possibility for a context switch is that C_i is obtained when A_k reads from a channel $c' \neq c$.
- The second possibility is that there is a configuration $C_g, g \leq i-1$, where A_k reads from a channel $c' \neq c$ and, configurations $C_{g+1}, \ldots, C_i$ either have no channel operations, or A_k only writes to its channels in $C_{g+1}, \ldots, C_i$.

Definition 8. A CTA $\mathcal{N}$ is bounded context, if the number of context switches in any run of $\mathcal{N}$ is bounded above by some $B \in \mathbb{N}$.

See the right part of Figure 1 for an example of a CTA consisting of two processes A_1, A_2, where A_1 writes on $c_{1,2}$ to A_2. This acyclic CTA is not bounded context. There is a run where A_1 writes an a after every one time unit, and A_2 reads an a once in two time units. There is also a run where A_1 writes b onto the channel whenever it pleases and A_2 reads it one time unit after it is written.

Theorem 9. *Reachability is decidable for bounded context CTA with global clocks and any number of processes.*

The Idea

Let K be the maximal constant used in the CTA with bounded context $\leq B$, and let $[K] = \{0, 1, \ldots, K, \infty\}$. For $1 \leq i \leq n$, let $A_i = (L_i, L_i^0, Act, \mathcal{X}_i, E_i, F_i)$ be the n automata in the CTA. Let $c_{i,j}$ denote the channel to which A_i writes to and A_j reads from. We translate the CTA into a bounded phase, multistack pushdown system (BMPS) $\mathcal{M}$ preserving reachability. A multistack pushdown system (MPS) is a timed automaton with multiple untimed stacks. A *phase* in an MPS is one where a fixed stack is popped, while pushes can happen to any number of stacks. A change of phase occurs when there is a change in the stack which is popped. See Appendix E.1 for a formal definition. We use Lemma 10 (proof in Appendix E.1) to obtain decidability after our reduction.

Lemma 10. *The reachability problem is decidable for BMPS.*

Encoding into BMPS

The BMPS $\mathcal{M}$ uses two stacks $W_{i,j}$ and $R_{i,j}$ to simulate channel $c_{i,j}$. The control locations of $\mathcal{M}$ keeps track of the locations and clock valuations of all the A_i, as n pairs $(p_1, \nu_1), \ldots, (p_n, \nu_n)$

with $\nu_i \in [K]$ for all i; in addition, we also keep an ordered pair (A_w, b) consisting of a bit $b \leq B$ to count the context switch in the CTA and also remember the active automaton $A_w, w \in \{1, 2, \ldots, n\}$. To simulate the transitions of each A_i, we use the pairs (p_i, ν_i), keeping all pairs (p_j, ν_j) unchanged for $j \neq i$. An initial location of $\mathcal{M}$ has the form $((l_1^0, \nu_1), \ldots, (l_n^0, \nu_n), (A_i, 0))$ where $l_i^0 \in L_i^0$, $\nu_i = 0^{|\mathcal{X}_i|}$; the pair $(A_i, 0)$ denotes context 0, and A_i is some automaton which is active in context 0 (A_i writes to some channels).

Transitions of $\mathcal{M}$

The internal transitions Δ_{in} of $\mathcal{M}$ correspond to any internal transition in any of the A_is and change some (p, ν) to (q, ν') where ν' is obtained by resetting some clocks from ν. These take place irrespective of context switch.

The push and pop transitions (Δ_{push} and Δ_{pop}) of $\mathcal{M}$ are more interesting. Consider the kth context where A_j is active in the CTA. In $\mathcal{M}$, this information is stored as (A_j, k). In the kth context, A_j can read from atmost one fixed channel $c_{l,j}$; it can also write to several channels $c_{j,i_1}, \ldots, c_{j,i_k} \neq c_{l,j}$, apart from time elapse/internal transitions. All automata other than A_j participate only in time elapse and internal transitions. When A_j writes a message m to channel c_{j,i_h} in the CTA, it is simulated by pushing message m to stack W_{j,i_h}. All time elapses $t \in [K]$ are captured by pushing t to all stacks. Δ_{push} has transitions pushing a message m on a stack W_{i,j_k}, or pushing time elapse $t \in [K]$ on all stacks.

When A_j is ready to read from channel $c_{l,j}$ (say), the contents of stack $W_{l,j}$ are shifted to stack $R_{l,j}$ if the stack $R_{l,j}$ is empty. Assuming $R_{l,j}$ is empty, we transfer contents of $W_{l,j}$ to $R_{l,j}$. The stack to be popped is remembered in the finite control of $\mathcal{M}$: the pair (p, ν), $p \in L_j$ is replaced with $(p^{W_{l,j}}, \nu)$. As long as we keep reading symbols $t \in [K]$ from $W_{l,j}$, we remember it in the finite control of $\mathcal{M}$ by adding a tag t to locations $(p^{W_{l,j}}, \nu)$ $(p \in L_j)$ making it $((p^{W_{l,j}})_t, \nu)$. When a message m is seen on top of $W_{l,j}$, with $((p^{W_{l,j}})_t, \nu)$ in the finite control of $\mathcal{M}$, we push (m, t) to stack $R_{l,j}$, since t is the indeed the time that elapsed after m was written to channel $c_{l,j}$. When we obtain $t' \in [K]$ as the top of stack $W_{l,j}$, with $((p^{W_{l,j}})_t, \nu)$ in the finite control, we add t' to the finite control obtaining $((p^{W_{l,j}})_{t+t'}, \nu)$. The next message m' has age $t + t'$ and so on, and stack $R_{l,j}$ is populated. When $W_{l,j}$ becomes empty, the finite control is updated to $(p^{R_{l,j}}, \nu)$ and A_j starts reading from $R_{l,j}$. If $R_{l,j}$ is already non-empty when A_j starts reading, it is read off first, and when it becomes empty, we transfer $W_{l,j}$ to $R_{l,j}$. A time elapse t'' between reads and/or reads/writes of A_j is simulated by pushing t'' on all stacks, to reflect the increase in age of all messages stored in all stacks.

Phases of $\mathcal{M}$ are bounded

Each context switch in the CTA results in $\mathcal{M}$ simulating a different automaton, or simulating the read from a different channel. Assume that every context switch of the CTA results in some automaton reading off from some channel. Correspondingly in $\mathcal{M}$, we pop the corresponding R-stack, and if it goes empty, pop the corresponding W-stack filling up the R-stack. Once the R-stack is filled up, we continue popping it. This results in atmost two phase changes (some $R_{i,j}$ to $W_{i,j}$ and $W_{i,j}$ to $R_{i,j}$) for each context in the CTA. An additional phase change is incurred on each context switch (a different stack $R_{k,l}$ is popped in the next context). Note that $\mathcal{M}$ does not pop a stack unless a read takes place in some automaton, and the maximum number of stacks popped is 2 per context. $\mathcal{M}$ is hence a $3B$ bounded phase MPS. A detailed proof of correctness and an example can be seen in Appendices F, F.3.

6 Discussion

In this paper, we have studied the reachability problem for timed processes communicating through perfect timed channels. We have shown that in the absence of global clocks, 3 processes with 2 channels already give the undecidability of the reachability problem, while with 2 processes the reachability problem becomes decidable. Our work gives an exhaustive characterisation for the decidability border of the reachability problem in terms of number of processes and the underlying topology[2] in the case of discrete timed systems. Given our undecidability results, the only question that remains open in the case of dense time is the decidability of reachability for 2 processes connected by a unidirectional channel, where the processes are Alur-Dill style timed automata and the ages of the messages can also be non-integral values. The tightness of the lower bound (NP−hardness) of our decidability result (Theorem 5) is also open.

We mention the possible extensions to the model of CTA as studied in this paper which will preserve the decidability result in Theorem 5.

1. If we allow diagonal constraints of the form $x - y \sim c$ where x, y are clocks and $c \in \mathbb{N}$, Theorem 5 continues to hold. In the proof, given a CTA $\mathcal{N}$ consisting of timed automata A, B connected by the channel $c_{A,B}$ from A to B, we construct a one counter automaton $\mathcal{O}$ using $Reg(A)$ and $Reg(B)$. We can easily track the difference between two clocks x, y in $Reg(A)$ or $Reg(B)$, thereby handling diagonal constraints.
2. The initial age of a newly written message in a channel is set to 0. This can be generalized in two ways : (i) allowing the initial age of a message to be some $j \in \mathbb{N}$, or (ii) assigning the value of some clock x as the initial age. The construction of $\mathcal{O}$ is such that each time A writes a message $m \in \Sigma$ to the channel, m is remembered in the finite control of $\mathcal{O}$ (transition 3(d) in the proof of Theorem 5). While simulating the read by B of the message m (transitions 3(e), (f), (g) in the proof of Theorem 5), the value i in the finite control of $\mathcal{O}$ along with the top of the stack determines whether the age of m is $< K, = K$ or $> K$, where K is the maximal constant used in A, B. This is used to see if the age constraint of m is met; the age of m when it is read is same as the time difference between B, A. We can adapt this for an initial age $j > 0$, by remembering (m, j) in the finite control of $\mathcal{O}$. If the counter value is $i < K$, then the age of the message is $j + i$, while if it is K and the top of stack is $\perp$, then the age of m is $j + K$, and it is $> j + K$ if the top of stack is not $\perp$. Checking the age constraint of m correctly now boils down to using $j + i$ and verifying if the constraint is satisfied.

References

1 P. A. Abdulla and A. Nylén. Timed Petri nets and BQOs. In *ICATPN*, 2001.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jonathan Cederberg. Timed lossy channel systems. In *FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 374–386. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

3 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. Dense-timed pushdown automata. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 35–44. IEEE Computer Society, 2012.

4 Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *LICS*. IEEE Computer Society, 1993.

5 Parosh Aziz Abdulla, Pritha Mahata, and Richard Mayr. Dense-timed petri nets: Checking zenoness, token liveness and boundedness. *Logical Methods in Computer Science*, 3(1), 2007.

6 S. Akshay, Paul Gastin, and Shankara Narayanan Krishna. Analyzing timed systems using tree automata. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[2] the graph where each node is associated to a process and a directed edge between two nodes exists iff there is a channel between their associated processes

7 S. Akshay, Blaise Genest, and Loïc Hélouët. Decidable classes of unbounded petri nets with time and urgency. In *Application and Theory of Petri Nets and Concurrency - 37th International Conference, PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings*, volume 9698 of *Lecture Notes in Computer Science*, pages 301–322. Springer, 2016.

8 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.

9 B. Bérard, F. Cassez, S. Haddad, O. Roux, and D. Lime. Comparison of different semantics for time Petri nets. In *ATVA 2005*, 2005.

10 Devendra Bhave, Vrunda Dave, Shankara Narayanan Krishna, Ramchandra Phawade, and Ashutosh Trivedi. A perfect class of context-sensitive timed languages. In *Developments in Language Theory - 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings*, volume 9840 of *Lecture Notes in Computer Science*, pages 38–50. Springer, 2016.

11 Laura Bocchi, Julien Lange, and Nobuko Yoshida. Meeting Deadlines Together. In Luca Aceto and David de Frutos Escrig, editors, *26th International Conference on Concurrency Theory (CONCUR 2015)*, volume 42 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 283–296, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

12 A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems*, LNCS 999, pages 64–85. Springer, 1994.

13 Ahmed Bouajjani and Peter Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. *Theor. Comput. Sci.*, 221(1-2):211–250, 1999.

14 Florent Bouchy, Alain Finkel, and Arnaud Sangnier. Reachability in timed counter systems. *Electr. Notes Theor. Comput. Sci.*, 239:167–178, 2009.

15 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

16 Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.

17 Pierre Chambart and Philippe Schnoebelen. Mixing lossy and perfect fifo channels. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 340–355. Springer, 2008.

18 Lorenzo Clemente, Frédéric Herbreteau, Amélie Stainer, and Grégoire Sutre. Reachability of communicating timed processes. In *FOSSACS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7794 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2013.

19 Lorenzo Clemente and Slawomir Lasota. Timed pushdown automata revisited. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 738–749. IEEE Computer Society, 2015.

20 Lorenzo Clemente, Slawomir Lasota, Ranko Lazic, and Filip Mazowiecki. Timed pushdown automata and branching vector addition systems. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.

21 Z. Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theor. Comput. Sci.*, 302(1-3):93–121, 2003.

22 M. Emmi and R. Majumdar. Decision problems for the verification of real-time software. In *HSCC*, LNCS 3927, pages 200–211. Springer, 2006.

23 Pierre Ganty and Rupak Majumdar. Analyzing real-time event-driven programs. In *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2009.

24 Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam. Cyber-physical modeling of implantable cardiac medical devices. *Proceedings of the IEEE*, 100(1):122–137, 2012.

25 Pavel Krcál and Wang Yi. Communicating timed automata: The more synchronous, the more difficult to verify. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144, pages 249–262. Springer, 2006.

26 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Context-bounded analysis of concurrent queue systems. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2008.

27 M. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall International, 1967.

28 J. K. Pachl. *Reachability Problems for Communicating Finite State Machines.* PhD thesis, Faculty of Mathematics, University of Waterloo, Ontario, 1982.

29 A. Trivedi and D. Wojtczak. Recursive timed automata. In *ATVA*, pages 306–324, 2010.

Appendix

A Proof of Theorem 1

Given an untimed automaton A with a perfect channel feeding into itself, the reachability problem is known to be undecidable. We reduce reachability of such a system to the reachability in a CTA consisting of two timed automata A_1, A_2 connected by a unidirectional channel, allowing global clocks.

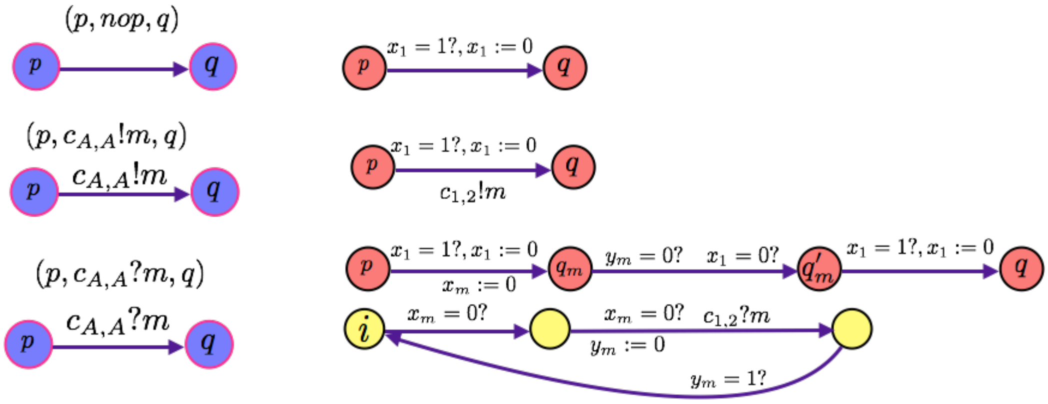


Figure 5 On the left, we show each transition in A (*nop* and write transitions) and on the right, the corresponding widget in A_1. A read transition in A has widgets in both A_1, A_2. A_1, A_2 are obtained by connecting all these widgets.

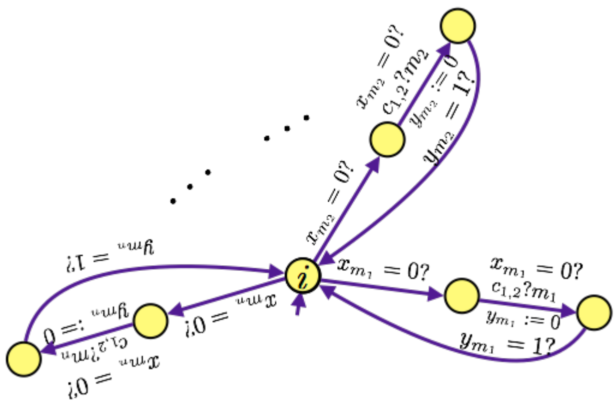


Figure 6 The automaton A_2 of the CTA, assuming the message alphabet is $\{m_1, \ldots, m_n\}$.

Figure 6 describes the timed automaton A_2 of the CTA $\mathcal{N}$. A_1 is obtained by composing all the widgets drawn for each transition in A. Let the channel alphabet of A be $\{m_1, \ldots, m_n\}$. Then A_1 has clocks x_1 and clocks $x_{m_1}, \ldots, x_{m_n}$ while A_2 has clocks $y_{m_1}, \ldots, y_{m_n}$. The clocks x_{m_i}, y_{m_i} will be used while respectively writing/reading message m_i. For each transition in A, we have a widget in A_1 as seen in Figure 5. The initial location of A_1 is the same as A, let it be

s_0. Each transition in A from a location p to q also has a corresponding transition in A_1 from p to q (or a sequence of transitions in A_1 from p to q). A_2 has widgets only corresponding to read transitions in A. The automaton A_2 is star-shaped obtained by joining widgets at a location i (this is the central node in Figure 6). i is also the initial location of A_2. Each read operation of A corresponds to a widget in A_2.

1. Consider a transition (p, nop, q) in A. Correspondingly, we have in A_1, a transition from p to q that checks if x_1 is 1 and resets it. This time elapse ensures that the clocks x_{m_i} and y_{m_i} grow, and are non-zero.
2. Consider a transition $(p, c_{A,A}!m, q)$ in A. Correspondingly, we have in A_1, a transition from p to q that checks if x_1 is 1 and resets it, and writes message m to $c_{1,2}$. This time elapse ensures that the clocks x_{m_i} and y_{m_i} grow, and are non-zero.
3. Consider a transition $(p, c_{A,A}?m, q)$ in A. Correspondingly, we have in A_1, a transition from p to an intermediate location q_m, where x_1 grows to 1 and is reset. The clock x_m is also reset to 0. The automaton A_2 at location i, checks that x_m is 0, and moves from location i into the widget for message m. It reads m from $c_{1,2}$ and sets clock y_m to 0. A_1 checks if y_m is 0 and then moves to location q'_m with no time elapse. From q'_m, A_1 moves to q elapsing a unit of time, resetting x_1. A_2 also goes back to i, elapsing a unit of time.
   Note that A_2 cannot read a message m unless A_1 tells it to; the way A_1 tells A_2 to read m is by setting clock x_m to 0. Note also that every transition involves a time elapse, and so in general, none of the clocks x_m, y_m will be 0. x_m is 0 only when A_1 resets it; A_2 reads m and resets y_m. This is the only time when y_m can be 0.

The correctness of the construction is proved using Lemma 11.

Lemma 11. *Let A be an untimed automaton with the perfect channel $c_{A,A}$ connecting A to itself. Let ρ be a run of A beginning with the initial configuration (s_0, ϵ), reaching some configuration (p, w), $w \in \Sigma^*$. Then we have a corresponding run ρ' in the constructed CTA $\mathcal{N}$ starting with (s_0, i, ϵ) and reaching configuration (p, i, w'), $w' \in (\Sigma \times \mathbb{N})^*$ such that $untime(w') = w$. The converse direction simulating a run of $\mathcal{N}$ in A holds similarly.*

We give here, the proof from A to $\mathcal{N}$. The proof is by construction. It is clear that corresponding to an initial configuration (s_0, ϵ) of A, we are in an initial configuration (s_0, i, ϵ) in $\mathcal{N}$. All internal transitions and write transitions in A from p to q result in a transition in A_1 from p to q. In the case of an internal transition in A, we have an internal transition in A_1; a write in A translates to a write in A_1. In both these cases, A_2 does not move (assume that in the initial configuration, it moves and enters some widget, since all clocks are 0. Then it will get stuck trying to read some message m_i since nothing is written so far. If it tries to read the message at a later time, it will be successful only if A_1 indeed set x_{m_i} to 0 and no time elapse happened after that). Clearly, as long as there are no reads, the contents of channels $c_{A,A}$ and $c_{1,2}$ are the same.

Consider now a read transition from p to q in A, where message m_i is being read. Correspondingly we are at location p in A_1 and at i in A_2. The first transition is a time elapse one, where A_1 moves from p to q_{m_i}. To simulate the read, A_1 resets clock x_{m_i} while going to q_{m_i}. A_2, on checking x_{m_i} as 0, moves from i into the widget corresponding to m_i. It then resets y_{m_i}, and reads m_i with no time elapse. A_1, from q_{m_i}, checks if y_{m_i} is 0, and if so, moves to q'_{m_i}. A unit time elapse takes A_1 to q, while A_2 goes back to i. Note that to move out of i, some x_{m_i} must become 0, and when A_2 returns to i, none of the clocks x_{m_j}, y_{m_j} are zero. Thus, when we reach q in A_1, we have simulated a read of the channel.

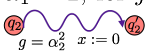
It is clear that $\mathcal{N}$ simulates A, and if we reach some location p of A with some channel contents w, then we reach the same location in A_1, and if we ignore the ages of the messages in channel $c_{1,2}$, we have the same content w. The converse direction from $\mathcal{N}$ to A can be proved similarly by the construction of $\mathcal{N}$.

B Corollary 2: The case of a single global clock

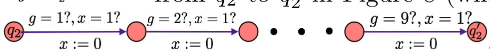
In this section, we show that even if there is only one global clock in the proof of Theorem 1, we obtain undecidability.

Let g denote the global clock and we assume that the messages in the channel alphabet are indexed $m_1, \ldots, m_k$. The proof idea is same as in Theorem 1, namely, to simulate an untimed automaton A with a channel. As in the proof of Theorem 1, we construct a CTA $\mathcal{N}$ with timed automata A_1 and A_2, connected by the channel $c_{1,2}$ from A_1 to A_2. A_1 has all locations of A, and some extra locations to simulate transitions of A. A_2 has $k+1$ locations, of which $init_{A_2}$ is the initial location. The other k locations are used to facilitate the reading of messages m_1 through m_k. The channel alphabet of the CTA is $\{(m_j, j) \mid 1 \le j \le k\}$. A_1 has a local clock x and A_2 has a local clock y.

An internal transition of A is simulated by A_1 by elapsing one unit of time, and both g as well as A_1's local clock x, are reset. Whenever A writes a message m_j to its channel, the first automaton A_1 writes (m_j, j) to the channel $c_{1,2}$. Again, one unit of time elapses, and g, x are set to 0 after that. To simulate a read transition $(p, c_{A,A}?m_j, q)$ in A of the message m_j, A_1 moves to a location q_j from p. From here, it elapses α_j^j units, where α_j is the jth prime number (for $j = 1$, $\alpha_1 = 2$, for $j = 2$, $\alpha_2 = 3$, for $j = 3$, $\alpha_3 = 5$ and so on). See Figure 8. The squiggly transition



from q_2 to q_2' in Figure 8 (when A_1 is simulating the read of m_2) is expanded as



.

A_2 guesses a message it is going to read by choosing a branch and resets its local clock y. Assume A_2 chooses the correct branch guessing that m_j is at the head of the channel. Once a branch is chosen, A_2 will wait to check that g is α_j^j; this time elapse takes place between locations q_j to q_j' of A_1. x is reset to 0. Once $g = \alpha_j^j$, with no time elapse, A_2 moves ahead, and reads message (m_j, j) and resets g. $g = 0$ is the signal for A_1 that the message has been read by A_2.

1. Assume that A_2 guesses a wrong branch. That is, it chooses the branch for message m_j when A_1 was trying to simulate the read of m_i. If indeed m_i is at the head of the channel, then A_2 will get stuck. Note that once A_2 chooses a branch, there is no escape, and the message must be read with no time elapse.
2. Assume now that we have a read transition $(p, c_{A,A}?m_i, q)$ in A, when the head of the channel $c_{A,A}$ actually contains m_j. In this case, A will get stuck. Our construction will be correct if the CTA $\mathcal{N}$ also gets stuck. The transitions of A_1 are obtained from A, so in A_1, we will go from p to location q_i. Below, we check that the simulation gets stuck somewhere in the CTA as well.

   a. The easiest case is when A_2 faithfully guesses that it must read m_i, and chooses that branch. In this case, it gets stuck since the head of the channel is not m_i.
   b. The same holds when A_2 chooses any branch other than m_j. Below we consider what happens when A_2 chooses the branch to read corresponding to m_j.
      - Assume that $j < i$. Then $\alpha_j^j < \alpha_i^i$. Since A_2 has chosen the branch corresponding to m_j, when g becomes equal to α_j^j, A_2 can move forward checking $g = \alpha_j^j$ and $y = 0$ on its chosen branch. At this time, A_1 is somewhere in the path between q_i and q_i', with $g = \alpha_j^j$ and $x = 0$. If A_2 goes inside when $g = \alpha_j^j$ and $y = 0$, it reads (m_j, j) from $c_{1,2}$, and resets g to 0. A_1 will now be stuck : to enable its next transition, it will check $g = \alpha_i^i + 1$ and $x = 1$ simultaneously, which will not be satisfied, since we have $g = 0$ and $x = 0$, and a unit time elapse will make $x = g = 1$.
      - Assume that $j > i$. In this case, A_2 must check $g = \alpha_j^j > \alpha_i^i$ to be able to read m_j. Since A_1 will simulate the transition $(p, c_{A,A}?m_i, q)$, it will go from q_i to q_i', obtaining $g = \alpha_i^i$. This is insufficient for A_2 to read (m_j, j) where it needs g to be α_j^j. A_1 cannot

proceed further since it needs $g = 0$ and $x = 0$. To obtain $g = 0$ in A_1, we need A_2 to read the message and reset g. The latter cannot happen since if A_2 elapses time $\alpha_j^j - \alpha_i^i$ from $init_{A_2}$, then x will be non-zero, disallowing A_1 to move forward to q. Hence, the CTA will get stuck.

The correctness of the construction can be proved in a similar way as done in Lemma 11.

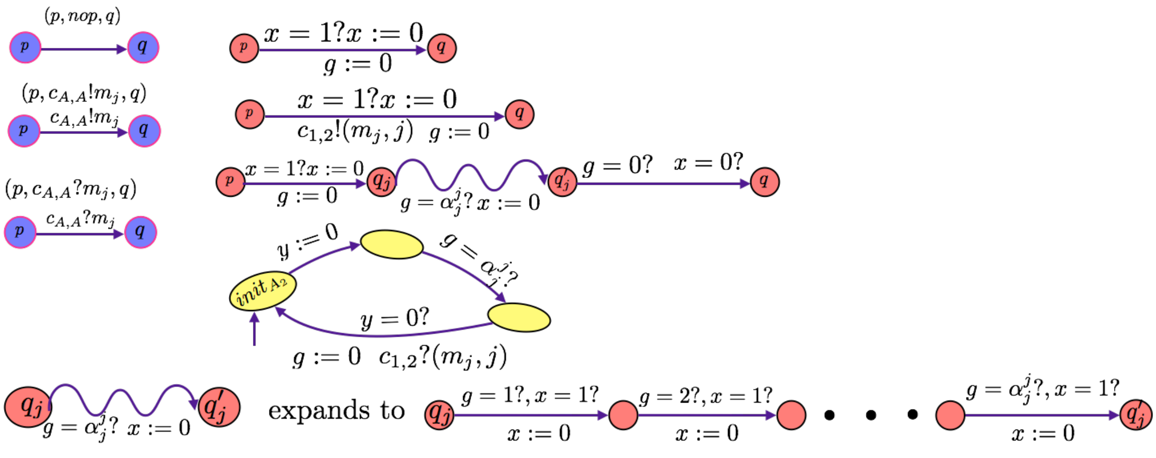


Figure 7 On the left, are the transitions of A. On the right, the red locations are those of A_1, and the yellow ones that of A_2. A_2 is enabled only on read transitions of A. α_j denotes the jth prime number. The squiggly transition from q_j to q_j' is expanded as above, and consists of α_j^j transitions.

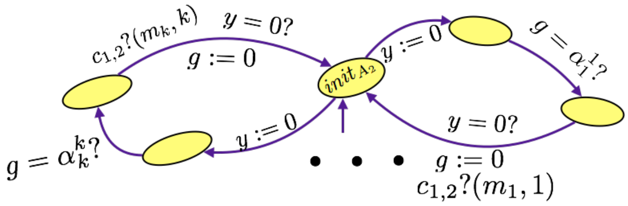


Figure 8 The automaton A_2 consists of widgets for reading messages $m_1, \dots, m_k$. Once a branch is chosen correctly, A_2 can come back to $init_{A_2}$ only after reading the head of the channel.

C Proof of Theorem 3

C.1 Counter Machines

A two-counter machine $\mathcal{C}$ is a tuple $(L, \{c_1, c_2\})$ where $L = \{\ell_0, \ell_1, \dots, \ell_n\}$ is the set of instructions—including a distinguished terminal instruction ℓ_n called HALT—and $\{c_1, c_2\}$ is the set of two *counters*. The instructions L are one of the following types:

1. (increment c) $\ell_i : c := c + 1$; goto ℓ_k,
2. (decrement c) $\ell_i : c := c - 1$; goto ℓ_k,
3. (zero-check c) ℓ_i : if $(c > 0)$ then goto ℓ_k else goto ℓ_m,
4. (Halt) ℓ_n : HALT.

where $c \in \{c_1, c_2\}$, $\ell_i, \ell_k, \ell_m \in L$. A configuration of a two-counter machine is a tuple (l, c, d) where $l \in L$ is an instruction, and c, d are natural numbers that specify the value of counters c_1 and c_2, respectively. The initial configuration is $(\ell_0, 0, 0)$. A run of a two-counter machine is a (finite or infinite) sequence of configurations $\langle k_0, k_1, \ldots\rangle$ where k_0 is the initial configuration, and the relation between subsequent configurations is governed by transitions between respective instructions. The run is a finite sequence if and only if the last configuration is the terminal instruction ℓ_n. Note that a two-counter machine has exactly one run starting from the initial configuration. The *halting problem* for a two-counter machine asks whether its unique run ends at the terminal instruction ℓ_n. It is well known ([27]) that the halting problem for two-counter machines is undecidable.

We reproduce the widgets here for convenience.

1. Consider an increment instruction ℓ_i : inc c goto ℓ_j. The widgets $\mathcal{W}_i^{A_m}$ for $m = 1, 2, 3$ are described in Figure 9. The one on the left is while incrementing c_1, while the one on the right is obtained while incrementing c_2.

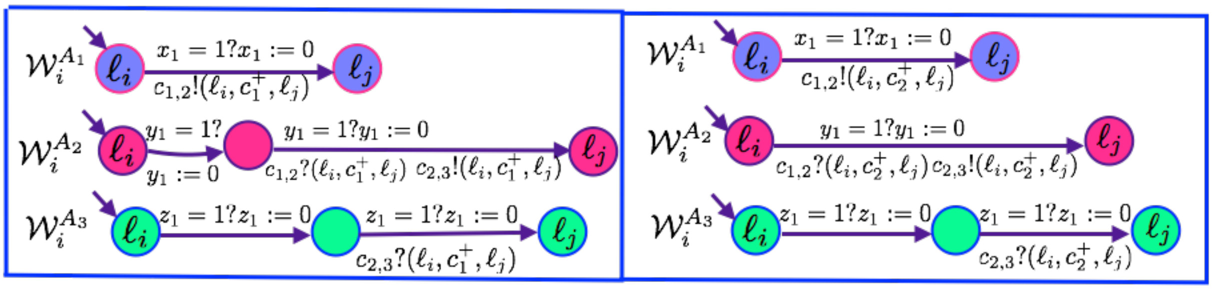


Figure 9 Widgets corresponding to an increment c_1, c_2 instruction in each process. The overload of notation when there is a write and a read on the same transition for A_2 can be easily split into two transitions. We keep it this way for conciseness.

2. The case of a decrement instruction is similar, and is obtained by swapping the speeds of the two automata in reaching ℓ_j from ℓ_i. Consider a decrement instruction ℓ_i : dec c goto ℓ_j. The widgets $\mathcal{W}_i^{A_m}$ for $m = 1, 2, 3$ are described in Figure 10. The one on the left is while decrementing c_1, while the one on the right is obtained while decrementing c_2.

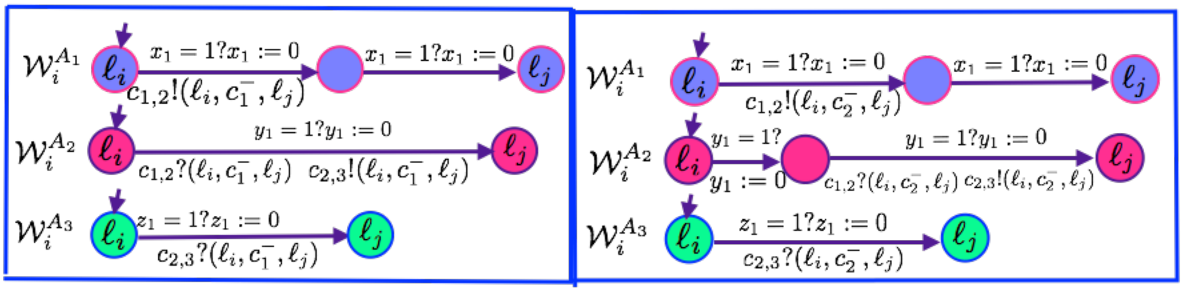


Figure 10 Widgets corresponding to a decrement c_1, c_2 instruction in each process

3. We finally consider a zero check instruction of the form ℓ_i : if $c_1 = 0$, then goto ℓ_j, else goto ℓ_k. The widgets $\mathcal{W}_i^{A_m}$ for $m = 1, 2, 3$ are described in Figure 11. The one on the left is a zero check of c_1, while the one on the right is a zero check of c_2.

C.2 Proof of Lemma 4

Consider a run of the two counter machine $(\ell_0, 0, 0), (\ell_1, c_1^1, c_2^1), \ldots, (\ell_h, c_1^h, c_2^h), \ldots$. The CTA $\mathcal{N}$ is made up of three automata A_1, A_2, A_3, and in the initial configuration, all three automata are respectively in $(\mathcal{W}_0^{A_1}, \ell_0)$, $(\mathcal{W}_0^{A_2}, \ell_0)$, $(\mathcal{W}_0^{A_3}, \ell_0)$. The value of clocks $g_{A_1}, g_{A_2}, g_{A_3}$ are all 0.

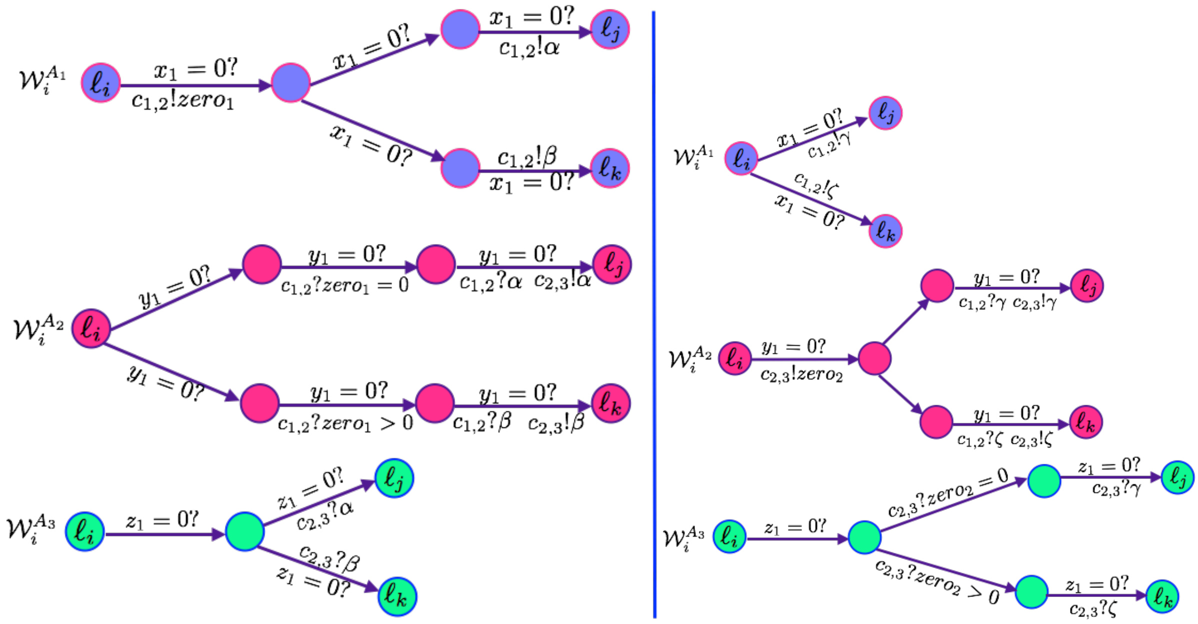


Figure 11 Widgets corresponding to checking c_1, c_2 is 0. $\alpha{=}(\ell_i, c_1{=}0, \ell_j)$, $\beta{=}(\ell_i, c_1{>}0, \ell_k)$, $\gamma{=}(\ell_i, c_2{=}0, \ell_j)$ and $\zeta{=}(\ell_i, c_2{>}0, \ell_k)$.

1. *Handling increment instructions.* We start with ℓ_0. Assume ℓ_0 is an increment c_1 instruction. A_1 completes the widget $\mathcal{W}_0^{A_1}$ in one time unit, while A_2 takes two units of time to complete $\mathcal{W}_0^{A_2}$. It can be seen that A_1 reaches $(\mathcal{W}_1^{A_1}, \ell_1)$ when $g_{A_1} = 1$, while A_2 reaches $(\mathcal{W}_1^{A_2}, \ell_1)$ when $g_{A_2} = 2$. Clearly, $g_{A_2} - g_{A_1} = 1$, the value of c_1 after one step. Likewise, A_3 reaches $(\mathcal{W}_1^{A_3}, \ell_1)$ when $g_{A_3} = 2$. $g_{A_3} - g_{A_2} = 0$, the value of c_2 after one step. In general, for each $\ell_i :$ inc c_1 goto ℓ_j instruction, the widget $\mathcal{W}_i^{A_1}$ progresses by one time unit, incrementing g_{A_1} by 1, while the widget $\mathcal{W}_i^{A_2}$ progresses by two time units. This ensures the difference between g_{A_2}, g_{A_1} at ℓ_j is one more than the difference at ℓ_i. Likewise, since widgets $\mathcal{W}_i^{A_2}$, $\mathcal{W}_i^{A_3}$ progress by two time units, the difference between g_{A_2} and g_{A_3} remains constant, preserving the value of counter c_2. The argument is same for an increment c_2 instruction $\ell_i :$ inc c_2 goto ℓ_j. The widgets $\mathcal{W}_i^{A_1}$, $\mathcal{W}_i^{A_2}$ progress by one unit, preserving the value of c_1, and $\mathcal{W}_i^{A_3}$ progresses by two time units, incrementing $g_{A_3} - g_{A_2}$ by one.
2. *Handling decrement instructions.* Assume $\ell_i :$ dec c_1 goto ℓ_j is a decrement c_1 instruction. A_1 completes the widget $\mathcal{W}_i^{A_1}$ in two time units, while A_2 takes one unit of time to complete $\mathcal{W}_i^{A_2}$. This ensures the difference between g_{A_2}, g_{A_1} at ℓ_j is one less than the difference at ℓ_i. Likewise, since widgets $\mathcal{W}_i^{A_2}$, $\mathcal{W}_i^{A_3}$ progress by one time unit, the difference between g_{A_2} and g_{A_3} remains constant, preserving the value of counter c_2. The argument is same for a decrement c_2 instruction $\ell_i :$ dec c_2 goto ℓ_j. The widgets $\mathcal{W}_i^{A_1}$, $\mathcal{W}_i^{A_2}$ progress by two units, preserving the value of c_1, and $\mathcal{W}_i^{A_3}$ progresses by one time unit, decrementing $g_{A_3} - g_{A_2}$ by one.
3. *The instruction flow in A_1, A_2, A_3.* Each time A_1 shifts control to an instruction, it writes to channel $c_{1,2}$ the instruction switch information. For example, if A_1 moves from ℓ_i to ℓ_j after incrementing c_1, it writes the tuple (ℓ_i, c_1^+, ℓ_j) in $c_{1,2}$. This guides A_2 to follow the same path, and A_2 writes the same in channel $c_{2,3}$ which will be followed by A_3. This is true for each instruction. If we observe the sequence $\dots(\ell_i, c_1^+, \ell_j)(\ell_j, c_2^-, \ell_k)\dots$ of messages written in $c_{1,2}$, it will be the same for $c_{2,3}$. Atleast when considering increment/decrement instructions, we can be sure that A_1, A_2, A_3 follow the same path/run of the two counter machine. The case of zero check is yet to be verified, which we do below.

4. *Handling Zero-Check.* Consider a zero check instruction ℓ_i : if $c_1 = 0$, then goto ℓ_j, else goto ℓ_k. By the above two cases, the values of counters c_1, c_2 are correctly encoded when A_1, A_2, A_3 reach ℓ_i in widget $\mathcal{W}_i^{A_m}$, $m \in \{1, 2, 3\}$.
   - Assume $c_1 = 0$. Then by the correctness of the encoding seen above, we know that the control of A_1, A_2 are respectively at $(\mathcal{W}_i^{A_1}, \ell_i)$ and $(\mathcal{W}_i^{A_2}, \ell_i)$ and $g_{A_2} = g_{A_1}$. No time is elapsed in widgets $\mathcal{W}_i^{A_1}, \mathcal{W}_i^{A_2}$. The channel $c_{1,2}$ is empty, and A_1 writes in a message $zero_1$ in $c_{1,2}$. Control switches non-deterministically, and a guess is made by A_1 whether c_1 is zero or not. If c_1 is guessed to be 0, then control switches to the upper part of $\mathcal{W}_i^{A_1}$, and a message $\alpha = (\ell_i, c_1{=}0, \ell_j)$ is written on the channel $c_{1,2}$. In A_2, control switches non-deterministically from $(\mathcal{W}_i^{A_2}, \ell_i)$ to one of the successor locations. If control switches to the upper successor, indeed we get a successful move since the age of $zero_1$ is 0. In this case, α is read off $c_{1,2}$ and α is written to $c_{2,3}$. This is to help process A_3 decide the next instruction ℓ_j correctly. Note that a wrong guess made in $\mathcal{W}_i^{A_1}$ affects the rest of the computation, since in this case, $\beta = (\ell_i, c_1{>}0, \ell_k)$ is written on $c_{1,2}$, and this cannot be read off in $\mathcal{W}_i^{A_2}$ since the lower part of $\mathcal{W}_i^{A_2}$ will be disabled.
   - Assume $c_1 > 0$. In this case, we know that $g_{A_2} - g_{A_1} > 0$ when control respectively reaches $(\mathcal{W}_i^{A_1}, \ell_i)$ and $(\mathcal{W}_i^{A_2}, \ell_i)$. Hence, when A_1 reaches $(\mathcal{W}_i^{A_1}, \ell_i)$, A_2 will be in some widget $\mathcal{W}_d^{A_2}$, and ℓ_d is an instruction earlier than ℓ_i (ℓ_d comes before ℓ_i). Since no time elapse is possible in $(\mathcal{W}_i^{A_1}, \ell_i)$, A_2 waits wherever it is, while A_1 completes the widget $\mathcal{W}_i^{A_1}$. Since non-zero time elapse is necessary for A_2 to reach widget $\mathcal{W}_i^{A_2}$, the age of $zero_1$ will be > 0 when A_2 reads off from $c_{1,2}$. The guess of A_1 in the widget $\mathcal{W}_i^{A_1}$ is crucial here: A_1 must choose the lower half of the widget and write β. This will ensure that A_2 also writes β in $c_{2,3}$, and ensures that all three automata A_1, A_2, A_3 choose the instruction ℓ_k.

   Note that the value of c_2 is immaterial in the above. If c_2 and c_1 are both zero, then all three automata will be in ℓ_i in the respective widget $\mathcal{W}_i^{A_m}$ at the same time. If $c_2 > 0$, then A_3 will "catch up" and reach widget $\mathcal{W}_i^{A_3}$; however, the guess made by A_1 (which is verified by A_2) guides A_3 to the correct next instruction. The zero-check for c_2 is similar. Note that the sequence consisting of messages $((\ell_i, c_1^+, \ell_j)$, (ℓ_i, c_2^+, ℓ_j), (ℓ_i, c_1^-, ℓ_j), (ℓ_i, c_2^-, ℓ_j), $(\ell_i, c_1{=}0, \ell_j)$, $(\ell_i, c_1{>}0, \ell_j)$, $(\ell_i, c_2{=}0, \ell_j)$ and $(\ell_i, c_2{>}0, \ell_j))$ written in $c_{1,2}$ by A_1 and read by A_2, and written by A_2 on $c_{2,3}$ and read by A_3 ensures that all 3 automata follow the same sequence of instructions of the two counter machine. In particular, if the guesses made by A_1 regarding zero-check go wrong, then the computation stops.

   Some important points regarding checking if c_2 is zero or not.

   (1) If $c_1 = 0 = c_2$ and ℓ_i is an instruction checking if c_2 is zero. Then A_1, A_2 are both at ℓ_i and A_3 is also at ℓ_i. Analogous to α and β, we have $\gamma = (\ell_i, c_2{=}0, \ell_j)$ and $\zeta = (\ell_i, c_2{>}0, \ell_k)$. Then A_1 guesses if c_2 is zero or not by writing γ or ζ in $c_{1,2}$. The guess of A_1 propagates to A_2 and A_3, and the correctness of the guess made by A_1 is verified by A_3. If c_2 was indeed 0, and A_1 chose to write γ, and if A_2 also made the same guess (A_2 must agree with A_1; otherwise, the computation stops) and reads the γ on $c_{1,2}$ and wrote γ on $c_{2,3}$, then indeed A_3 will proceed smoothly, since it expects a γ when the age of $zero_2$ is 0.

   (2) If $c_1 > 0$, but $c_2 = 0$, and ℓ_i is an instruction checking if c_2 is zero. Then A_1 will have moved ahead from the widget $\mathcal{W}_i^{A_1}$ when A_2, A_3 reach $(\mathcal{W}_i^{A_2}, \ell_i)$, $(\mathcal{W}_i^{A_3}, \ell_i)$ together. The guesses of A_1 are already made, and one of ζ, γ will have been written in $c_{1,2}$, by the time A_2, A_3 reach $\mathcal{W}_i^{A_2}$, $\mathcal{W}_i^{A_3}$. The rest of the computation is smooth only if A_1 wrote γ, since A_3 will read $zero_2$ when its age is 0, and will hence expect to read γ.

   (3) If $c_1 = 0$, but $c_2 > 0$ and ℓ_i is an instruction checking if c_2 is zero. Then A_1, A_2 are together at $(\mathcal{W}_i^{A_1}, \ell_i)$, $(\mathcal{W}_i^{A_2}, \ell_i)$ respectively, while A_3 is in a widget $\mathcal{W}_g^{A_3}$ where ℓ_g is an instruction earlier than ℓ_i. In this case, a correct computation requires A_1 to take the

lower branch of $\mathcal{W}_i^{A_1}$ and write a ζ, since the age of $zero_2$ will be > 0 when A_3 reads it, and then $c_{2,3}$ must have a ζ.

(4) If $c_1 > 0$ and $c_2 > 0$, and ℓ_i is an instruction checking if c_2 is zero. Then A_1 is at the widget $\mathcal{W}_i^{A_1}$, while A_2 is in some widget $\mathcal{W}_d^{A_2}$ for some instruction ℓ_d before ℓ_i, and A_3 is in some widget $\mathcal{W}_f^{A_2}$ for some instruction ℓ_f before ℓ_d. In this case again, A_1 must choose the lower branch of $\mathcal{W}_i^{A_1}$, and write a ζ. This ζ will be read by A_2 when it catches up and reaches $\mathcal{W}_i^{A_2}$, and the ζ written by A_2 will be read by A_3 when it catches up a while later after A_2. When A_3 catches up, the age of $zero_2$ is > 0, and it will read the ζ written by A_2.

Note that the check on the age of $zero_1, zero_2$ is useful in checking if c_1, c_2 are 0 or not, and writing α, β ensures that all three processes are in agreement in their choices of instructions while simulating the two counter machine.

Lemma 12. *The two counter machine $\mathcal{C}$ halts iff the halt widget $\mathcal{W}_{halt}^{A_m}$ is reached in $\mathcal{N}$, m=1, 2*

By Lemma 4, we know that in any successful computation of $\mathcal{N}$, all three automata A_1, A_2 and A_3 go through the same sequence of widgets corresponding to the sequence of instructions witnessed by the two counter machine. Hence, if the two counter machine reaches the halt instruction, then all three processes reach the halt widget. The halt widget consists of the single location ℓ_{halt}, with no constraints. Note that when all processes reach this location in the halt widget, the difference between the values of g_{A_2}, g_{A_1} will be the value of counter c_1, while the difference between the values of g_{A_3}, g_{A_2} will be the value of counter c_2.

Likewise, if the two counter machine does not halt, then $\mathcal{N}$ also loops through the widgets corresponding to the sequence of instructions visited by the two counter machine.

C.3 Undecidability with other PolyForest Topologies

The Star Topology. The star topology is one where there is a central timed automaton A_0 which writes to all other timed automata A_i on a channel $c_{0,i}$, and there is no communication between these other automata.

It can be seen that even if we consider a CTA $\mathcal{N}$ with a star-topology with a central node (this central node is a timed automaton A_1) writing to timed automata A_2, A_3 through channels $c_{1,2}$ and $c_{1,3}$, the above undecidability result continues to hold good. In this case, the value of counter c_1 after i steps of the two counter machine will be encoded as the difference of the value of g_{A_2} when at l_i in A_2 and the value of g_{A_1} when at l_i in A_1. Likewise, the value of counter c_2 after i steps of the two counter machine will be encoded as the difference of the value of g_{A_3} when at l_i in A_3 and the value of g_{A_1} when at l_i in A_1. For the zero check instruction, A_1 passes on its guess, that is, whether it is α, β, γ or ζ to both A_2 and A_3 whenever it decides. The choice made if incorrect, will make one of A_2 or A_3 stuck, and that will in turn stop the computation. A correct guess will ensure that there is a smooth simulation of the two counter machine.

The Broom Topology. The broom topology is one where there is a central timed automaton A_0 to which all other timed automata A_i write to, on respective channels $c_{i,0}$, and there is no communication between these other automata. We can similarly encode the value of c_1 after i instructions as the difference between the values of clocks g_{A_1} when at l_i in A_1 and g_{A_2} when at l_i in A_2. Similarly for c_2, the value of c_1 after i instructions as the difference between the values of clocks g_{A_1} when at l_i in A_1 and g_{A_3} when at l_i in A_3. The main challenge is during a zero check. Note that both A_2, A_3 will be ahead of (or equal to) A_1 in the simulation of the two counter machine. Since A_2, A_3 are not communicating with each other, we must ensure that when a zero check instruction ℓ_i is reached, all three automata follow the same sequence of instructions. Assume that ℓ_i is an instruction which checks if c_1 is zero and accordingly, chooses

ℓ_j or ℓ_k. Since A_2 takes care of c_1, it will write the message $zero_1$ on the channel $c_{2,1}$, and follow it up with α or β. The correctness of this guess (age of $zero_1$ being 0 when read by A_1 and α being written, or age of $zero_1$ being > 0 when read by A_1 and β being written) follows as in the existing proof. The issue however is that, when A_3 encounters ℓ_i (it will, before A_1 does, or when A_1 does), it will make a choice of writing one of α, β on the channel $c_{3,1}$. A_3 will not write $zero_1$, since this check is carried out by A_2. If A_3 writes α, it will move to location ℓ_j while if it writes β, it will move to location ℓ_k. If the guess made by A_3 is not the same as made by A_2, then we must stop the computation, since it will mean that the sequence of instructions followed by all three machines are not the same. Note that when A_1 reaches ℓ_i, it will have at the head of channel $c_{2,1}$, the message $zero_1$, followed by one of α, β. Likewise, the head of channel $c_{3,1}$ will be one of α, β. The zero-check widget in A_1 is one with no time elapse. A_1 will first read $zero_1$, check its age, and if the age is 0, it will expect to read α at the head of both channels. Otherwise, it will be stuck. Likewise, if the age of $zero_1$ is > 0, it will expect to read β at the head of both channels. This ensures the correctness of zero check for c_1. The case of zero check for c_2 is similar, with $zero_2$ and γ, ζ playing analogous roles.

D Proof of Theorem 5

To prove the correctness of the construction of $\mathcal{O}$, we prove lemmas 6 and 7.

D.1 Proof of Lemma 6

Proof. The initial configuration in $\mathcal{O}$ is $((l_A^0, 0^{|\mathcal{X}_A|}), (l_B^0, 0^{|\mathcal{X}_B|}, \epsilon), 0)$. All clock values are 0 in A, B; the channel is empty and A, B are at the same global time 0. By construction of $\mathcal{O}$, we allow A to elapse time only when the counter value is $i > 0$. That is, for A to elapse time, B must have already elapsed some time. B is allowed to elapse time whenever it wants, and each such time elapse increases the counter value by 1 till it reaches K; further increase in time is stored in the stack. Thus, if B moves ahead for i units of time from the initial configuration, then the counter value is i, and it does represent the difference in time between B, A. If A elapses k units of time, then the counter value decreases by k. Assume that A writes a message m when we have i in the finite control and there are j 1's in the stack. Then $i + j$ is the time difference between B, A. If there is no time elapse in A after m was written, then it means that in B, $i + j$ time has elapsed since the time m was written, which is the age of m. ◻

D.2 Proof of Lemma 7

Proof. Let $\mathcal{N}$ be a CTA with timed automata A, B connected by a channel $c_{A,B}$ from A to B. Starting from the initial configuration $((l_A^0, 0^{|\mathcal{X}_A|}), (l_B^0, 0^{|\mathcal{X}_B|}), \epsilon)$ of $\mathcal{N}$, assume that we reach configuration $((l_A, \nu_1), (l_B, \nu_2), w.(m, i))$ such that $w \in (\Sigma \times \{0, 1, \ldots, i\})^*$. Also, assume that from (l_B, ν_2), there is an enabled read transition which reads m and checks that the age of m is i.

We start in $\mathcal{O}$ with $((l_A^0, 0^{|\mathcal{X}_A|}), (l_B^0, 0^{|\mathcal{X}_B|}, \epsilon), 0)$ and stack contents $\perp$. Till A writes a message onto the channel, the simulation of $\mathcal{O}$ consists of time elapse and internal transitions of A, B. By construction of $\mathcal{O}$, B is always ahead of A, or at the same global time as A. If A writes its first message say a when no time elapse has happened in A, B, then the age of a is 0 in B. Till B reads this message, we disallow further writes from A. In fact, we disallow any transition in A, and allow time elapse/internal transitions in B until the transition for reading a is enabled. Note that this is fine since there is no clock interference between A, B (if we had global clocks, we cannot do this, since a transition in B may depend on the current value of a clock in A). If a is to be read when its age is some i, then we allow time elapse of i in B after A has written a; at this time, the counter value will be i in $\mathcal{O}$, and we obtain some configuration $((p_A, \nu'_A), (l_B, \nu_2, a), i)$

and a stack with just $\bot$ if $i \leq K$. Let us assume $i \leq K$. Once B enables this transition, a is read, and we obtain a configuration $((p_A, \nu'_A), (l'_B, \nu'_2, \epsilon), i)$. (p_A, ν'_A) is the location reached in $Reg(A)$ after writing a on the channel. In general, if A writes a message when the counter value is i, then it means that the age of the message in B is i.

Assume that the counter value is i, and B just read a message that was written by A. If more messages need be written on the channel with no further time elapse, it can be done, since they can be read off in B only when their age is atleast i. In this case, each message is written, and A waits until it is read by B. If the current message has to be read when its age is $j > i$, and the next message must be read when its age is $j - h$ for some $h < j$, then B moves ahead by $j - i$ units of time, making the age of the message j and reads it off. The time difference between B and A is now j. A can now elapse h units of time and write the message, in which case it will be read by B as soon as it is written. We can continue this till A catches up with B; if none of the messages written in this time duration i need to be read when their ages are bigger than the time difference between B and A.

We know that in $\mathcal{N}$, the two automata A, B are always in-sync; let (l_A, ν_A) be the location of $Reg(A)$ when we are at (l_B, ν_2) in $Reg(B)$, when a is read. Going with the above discussion, indeed it is possible to reach (l_A, ν_A) from (p_A, ν'_A) after elapsing i units of time. In particular, each time A writes a message, B moves ahead exactly by the time needed to read the message satisfying its age requirements.

After A has written its last message and B has read it, A can catch up with B so that the time difference between B, A is 0; this leads to a configuration $((l_1, \nu_1), (l_2, \nu_2, \epsilon), 0)$ in $\mathcal{O}$ with stack contents $\bot$ iff in $\mathcal{N}$ we reach the configuration $((l_1, \nu_1), (l_2, \nu_2), \epsilon)$. The same sequence of transitions are taken in $Reg(A), Reg(B)$ in both $\mathcal{O}$ and $\mathcal{N}$, with the only difference being that in $\mathcal{N}$, the two automata move in-sync, while in $\mathcal{O}$, B is made to run ahead of A whenever A writes a message. In $\mathcal{O}$, we always keep atmost one message in the finite control, and when B has moved ahead and read that one, then we allow A to move ahead. The main difference between $\mathcal{N}$ and $\mathcal{O}$ is thus that in $\mathcal{O}$, A, B are "de-coupled", while in $\mathcal{N}$ they are in-sync. $\square$

D.3 Example Illustrating Theorem 5

We give an example illustrating Theorem 5. Figure 12 gives a CTA consisting of automata A, B, and also the respective region automata $Reg(A), Reg(B)$. Consider the run

$$\mathcal{N}_0{=}((s_1,0),(q_1,0),\epsilon) {\to} \mathcal{N}_1{=}((s_2,0),(q_1,0),(a,0)) \xrightarrow{*} \mathcal{N}_2{=}((s_3,1),(q_1,1),(c,0)(a,1)) \xrightarrow{*}$$
$$\mathcal{N}_3{=}((s_2,0),(q_3,\infty),(b,0)(a,0)(c,2)) \xrightarrow{*} \mathcal{N}_4{=}((s_2,0),(q_2,\infty),(b,0)(a,0)) \xrightarrow{*} \mathcal{N}_5{=}((s_2,1),(q_2,\infty),\epsilon).$$

The table illustrates the sequence of configurations in the counter automaton $\mathcal{O}$.

$\mathcal{O}_0$	$((s_1,0),((q_1,0),\epsilon),0)$	$\bot$	$\mathcal{O}_1$	$((s_2,0),((q_1,0),a),0)$	$\bot$	$\mathcal{N}_0{=}((s_1,0),(q_1,0),\epsilon)$
$\mathcal{O}_2$	$((s_2,0),((q_1,1),a),1)$	$\bot$	$\mathcal{O}_3$	$((s_3,1),((q_3,1),c),0)$	$\bot$	$\mathcal{N}_1{=}((s_2,0),(q_1,0),(a,0))$
	$((s_2,0),((q_{1\bot},1),a),1)$	ϵ		$((s_3,1),((q_3,\infty),c),1)$	$\bot$	$\mathcal{N}_2{=}((s_3,1),(q_1,1),(c,0)(a,1))$
	$((s_2,0),((q'_{1\bot},1),a),1)$	$\bot$		$((s_3,1),((q_3,\infty),c),1)$	$1\bot$	$\mathcal{N}_3{=}((s_2,0),(q_3,\infty),(b,0)(a,0)(c,2))$
	$((s_2,0),((q_2,1),\epsilon),1)$	$\bot$		$((s_3,1),(((q_3)_1,\infty),c),1)$	$\bot$	$\mathcal{N}_4{=}((s_2,0),(q_2,\infty),(b,0)(a,0))$
	$((s_2,1),((q_2,1),\epsilon),0)$	$\bot$		$((s_3,1),(((q'_3)_1,\infty),c),1)$	$1\bot$	$\mathcal{N}_5{=}((s_2,1),(q_2,\infty),\epsilon)$
	$((s_3,1),((q_2,1),\epsilon),0)$	$\bot$		$((s_3,1),((q_2,\infty),\epsilon),1)$	$1\bot$	$\mathcal{N}_i \xrightarrow{*} \mathcal{N}_{i+1}$ $\forall\, 0 \leq i \leq 4$ in the CTA $\mathcal{N}$
				$((s_2,0),((q_2,\infty),\epsilon),1)$	$1\bot$	
				$((s_2,1),((q_1,0),\epsilon),1)$	$\bot$	
				$((s_2,1),((q_1,0),\epsilon),0)$	$\bot$	
$\mathcal{O}_4$	$((s_2,0),((q_1,0),a),0)$	$\bot$	$\mathcal{O}_5$	$((s_2,0),((q_2,1),b),1)$	$\bot$	$\mathcal{O}_i \xrightarrow{*} \mathcal{O}_{i+1}$ forall $0 \leq i \leq 4$ in $\mathcal{O}$
	$((s_2,0),((q_1,1),a),1)$	$\bot$		$((s_2,0),((q_2,1),b),1)$	$\bot$	Each $\mathcal{O}_i$ has several steps
	$((s_2,0),((q_{1\bot},1),a),1)$	ϵ		$((s_2,0),((q_3,\infty),b),1)$	$1\bot$	A message is written and read in each $\mathcal{O}_i$
	$((s_2,0),(((q'_1)_\bot,1),a),1)$	$\bot$		$((s_2,0),((q_2,\infty),\epsilon),1)$	$\bot$	$1 \leq i \leq 5$
	$((s_2,0),((q_2,1),\epsilon),1)$	$\bot$		$((s_2,1),((q_2,\infty),\epsilon),0)$	$\bot$	

1. It is easy to see that $\mathcal{O}_o, \mathcal{O}_1$ exactly correspond to $\mathcal{N}_0, \mathcal{N}_1$. a is read in $\mathcal{O}_1$ obtaining

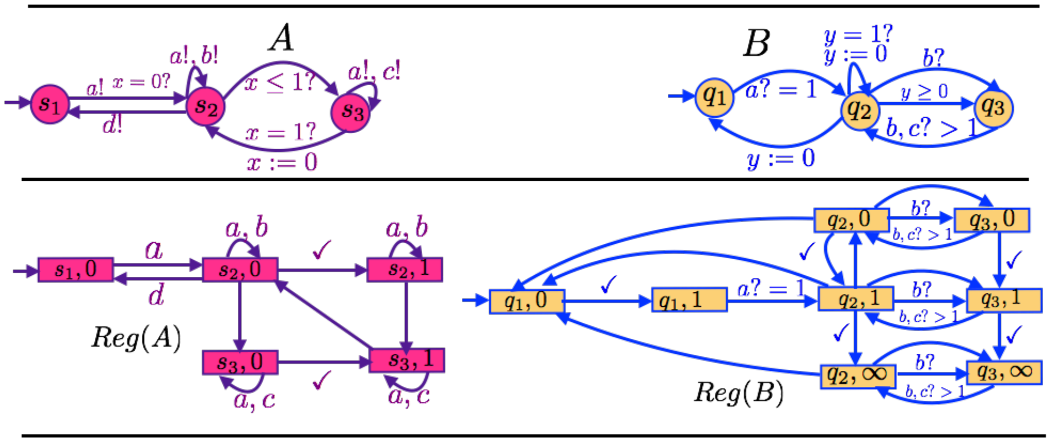


Figure 12 Timed automata A, B in a CTA $\mathcal{N}$. Both have a single clock. The region graphs are below. The checkmark represents unit time elapse.

$((s_2,0),((q_1,0),a),0)$. Neither A nor B have elapsed any time, and the stack is $\perp$.

2. If we look at $\mathcal{N}_2$, there are two messages in the channel, $(c,0)$ and $(a,1)$. This means that A has moved ahead writing two messages, while B has not yet read any. By construction of $\mathcal{O}$, until the first message is read, we do not write the second message. Thus, $\mathcal{O}_2$ will be a configuration obtained when $(a,1)$ is read. Recall that a was written in $\mathcal{O}_1$. Reading $(a,1)$ amounts to elapsing time in B, increasing the counter value and the age of a, and then checking that the age of a is 1. The time elapse of B results in the configuration namely, $((s_2,0),((q_1,1),a),1)$. Since $K=1$, and 1 is remembered in the finite control, checking that the age of a is exactly 1 amounts to checking the top of stack $\perp$, remembering it in the finite control, and then pushing it back. We do this, and once we are sure that the age of 1, we move to q_2 from $q'_{1\perp}$. After reading a, we elapse a unit of time in A, reducing the counter value to 0 from 1. We also move from $(s_2,1)$ to $(s_3,1)$ to read c, the next message read in $\mathcal{N}$. This gives the configuration $\mathcal{O}_2$ where we have $(s_3,1)$ in A, $(q_2,1)$ in B, counter value 0 indicating that B is not ahead of A, and the top of stack being $\perp$. That is, $((s_3,1),((q_2,1),\epsilon),0)$ with the stack holding $\perp$.
3. $\mathcal{N}_3$ is the configuration obtained when $(a,1)$ has been read, the age of c is 2, and in addition, two new messages b,a have been written, making the channel contain 3 messages b,a,c. 2 units of time has elapsed since $\mathcal{N}_2$. In the simulation of $\mathcal{O}$, the message c will be written first, then 2 time units elapsed, and c read. We are currently at $((s_3,1),((q_2,1),\epsilon),0)$. c is written from $(s_3,1)$. This gives $((s_3,1),((q_2,1),c),0)$. B moves from $(q_2,1)$ to $(q_3,1)$ with no time elapse. When B elapses one unit of time, $(q_3,1)$ becomes (q_3,∞), and the counter value becomes 1, the age of c is 1. This gives $((s_3,1),((q_3,\infty),c),1)$, and a stack $\perp$. One more unit time elapse makes the age of c 2, and 1 is pushed on the stack. This makes the configuration $((s_3,1),((q_3,\infty),c),1)$ along with the stack $1\perp$. To read the c from (q_3,∞), we check the age of c by checking if the top of stack is a 1, given that the counter value is 1. The 1 in the counter along with the top of stack 1 ensures that the age of c is >1. This check results in popping 1 from the top of stack, remembering it in the finite control, and then pushing it back, and then simulating the read from $((q'_3)_1,\infty)$. The finite control of B moves to (q_2,∞) reading the c obtaining $((s_3,1),((q_2,\infty),\epsilon),1)$ with stack $1\perp$. Then A moves from $(s_3,1)$ to $(s_2,0)$. A elapses a unit of time obtaining $(s_2,1)$ in the finite control, and the 1 is popped off the stack to keep track of the time difference between B and A. This gives $((s_2,1),((q_2,\infty),\epsilon),1)$ with stack $\perp$. The finite control of B moves from (q_2,∞) to $(q_1,0)$, obtaining $((s_2,1),(q_1,0,\epsilon),1)$ with stack $\perp$. In A, we move from $(s_2,1)$ to $(s_2,1)$ elapsing a

unit of time (for this it moves from $(s_2, 1)$ to $(s_3, 1)$ and back to $(s_2, 0)$, and elapses a unit) reducing the counter value to 0. This results in $\mathcal{O}_3$, where we have $((s_2, 1), ((q_1, 0), \epsilon), 0)$ with top of stack $\perp$.

4. $\mathcal{N}_4$ is the configuration where c has been read, and there are messages b, a in the channel with age 0. In $\mathcal{O}_3$ we read c, but have not yet written a, b. In A, the finite control moves from $(s_2, 1)$ to $(s_2, 0)$, where an a is written (by passing through $(s_3, 1)$). A unit time elapse in B results in the age of a to be 1, the counter value 1, and the finite control as $(q_1, 1)$. This results in $((s_2, 0), ((q_1, 1), a), 1)$ with stack $\perp$. A sequence of transitions as seen in the case of $\mathcal{O}_2$ (where θ is remembered in the finite control) takes place, and eventually, a is 1 after checking its age as 1. The control of B moves to $(q_2, 1)$ reading off a. This results in $\mathcal{O}_4$ with $((s_2, 0), ((q_2, 1), \epsilon), 1)$ with the stack $\perp$.
5. $\mathcal{N}_5$ is the configuration where b is read, and the channel is empty, with A at $(s_2, 1)$, B at (q_2, ∞) and an empty channel. In $\mathcal{O}$, we have to write b from $\mathcal{O}_4$ and read it when its age is > 1. This is done in a manner similar to what we did in $\mathcal{O}_3$ where the topmost 1 in the stack is read and remembered in the finite control. It can be seen that we obtain $\mathcal{O}_5$ with $((s_2, 1), ((q_2, \infty), \epsilon), 0)$ and stack $\perp$.

The main difference between configurations in $\mathcal{N}$ and $\mathcal{O}$ is thus the fact that in $\mathcal{N}$, we can choose to write several messages in the channel and read them later on, as long as their age requirements are met. In the case of $\mathcal{O}$, we write a message, and advance only B to read it, thereby, de-synchronizing A, B. We elapse time in A separately, and write a message only when the message which is written has already been read.

E Timed Multistack Pushdown Systems(MPS)

A timed multipushdown system is a timed automaton equipped with multiple untimed stacks. Formally, it is a tuple $\mathcal{M} = (S, S_0, St, \Gamma, \mathcal{X}, \Delta)$ where S is a finite set of locations, $S_0 \subseteq S$ is the set of initial locations, St is a finite set of stacks, Γ is a finite stack alphabet, $\mathcal{X}$ is a finite set of clocks, $\Delta = \Delta_{int} \cup \Delta_{push} \cup \Delta_{pop}$ is the transition relation with $\Delta_{int} \subseteq S \times \varphi(\mathcal{X}) \times 2^{\mathcal{X}} \times S$, $\Delta_{push} \subseteq S \times \varphi(\mathcal{X}) \times 2^{\mathcal{X}} \times St \times \Gamma \times S$ and $\Delta_{pop} \subseteq S \times \varphi(\mathcal{X}) \times 2^{\mathcal{X}} \times St \times \Gamma \times S$. A configuration of $\mathcal{M}$ is a tuple $(s, \nu, \{\sigma_{st}\}_{st \in St})$ where $s \in S$ is the current control location, ν is the current valuation of all the clocks, and for every $st \in St$, $\sigma_{st} \in \Gamma^*$ denotes the contents of stack St. The initial configuration is $(s_0, 0^{|\mathcal{X}|}, \{\sigma_{st}\}_{st \in St})$ with $\sigma_{st} = \epsilon$ for all $st \in St$. The semantics of $\mathcal{M}$ is given by defining the transition relation induced by Δ on the set of configurations of $\mathcal{M}$. A transition relation is written as $(s, \nu, \{\sigma_{st}\}_{st \in St}) \to (s', \nu', \{\sigma'_{st}\}_{st \in St})$ with one of the following cases:

1. Internal Move : All the stack contents remain unchanged, and we have the transition $(s, g, Y, s') \in \Delta_{int}$. To make the move, we check if $\nu \models g$, $\nu' = \nu[Y := 0]$ and the control moves to s'.
2. Push to stack st_i : The transition has the form $(s, g, Y, st_i, a, s') \in \Delta_{push}$. The contents of stack st_i changes from w to aw (the left most position denotes the top of the stack), all other stack contents stay unchanged, $\nu \models g$, $\nu' = \nu[Y := 0]$ and control moves to s'.
3. Pop from stack st_i: The transition has the form $(s, g, Y, st_i, a, s') \in \Delta_{pop}$. The top of stack st_i is popped. Thus, the contents of st_i changes from aw to w after the pop, all other stack contents stay unchanged, $\nu \models g$, $\nu' = \nu[Y := 0]$ and control moves to s'.

A run of $\mathcal{M}$ is a sequence of transitions $c_0 \to c_1 \to c_2 \cdots \to c_n$ connecting configurations. A state $s \in S$ is reachable iff there is a run with c_0 being the initial configuration, and c_n is a configuration $(s, \nu, \{\sigma_{st}\}_{st \in St})$. A *phase* of a run is part of the run where all the pop moves are from the same stack. A k-phase run is one where the run is composed of atmost k-phases. If a run is k-phase, then we can compose the run as $\alpha_1\alpha_2 \ldots \alpha_k$, where in each subrun α_i, there is a fixed stack $st \in St$ that is popped. Thus, in a k-phase run, there are atmost $k-1$ changes of the stack which is being popped. A MPS is bounded-phase (BMPS) if every run of the MPS is a k-phase run for some k. Reachability in a BMPS is shown decidable by reducing it to the bounded-phase reachability problem for untimed multipushdown systems. The proof (below, section E.1) follows using a standard region construction.

E.1 Proof of Lemma 10

Let $\mathcal{M} = (S, s_0, St, \Gamma, \mathcal{X}, \Delta)$ be a BMPS. The first step is to convert $\mathcal{M}$ to $Reg(\mathcal{M})$ by the standard region construction. The states of $Reg(\mathcal{M})$ have the form (l, ν) where $l \in S$ and $\nu \in \mathbb{N}^{|\mathcal{X}|}$. The internal transitions, push and pop transitions are now from locations (l, ν) to (l', ν'). It is easy to see that $Reg(\mathcal{M})$ is an untimed multistack push down automaton, which is bounded-phase iff $\mathcal{M}$ is. Moreover, given any $l \in S$, we can reach l from some $s_0 \in S^0$ iff we can reach some (l, ν) from $(s_0, \mathbf{0})$, preserving the stack contents. Using known results [26] we know that the reachability in $Reg(\mathcal{M})$ is decidable. Hence, reachability in $\mathcal{M}$ is also decidable.

F Proof of Theorem 9

Given a bounded context CTA $\mathcal{A}$, we first give the construction of an MPS $\mathcal{M}$ in section F.1, and show its correctness (preserves reachability and is bounded phase) in section F.2.

F.1 Construction of BMPS $\mathcal{M}$

Let the bounded context CTA $\mathcal{A}$ consist of n automata $A_1, A_2, \ldots, A_n$. Let $c_{i,j}$ denote the channel from A_i to A_j. Without loss of generality, we assume that there is atmost one channel from any A_i to A_j; our construction will work even when there are many channels from A_i to A_j. Assume Σ is the channel alphabet of $\mathcal{A}$. Let $A_i = (L_i, L_i^0, Act, \mathcal{X}_i, E_i, F_i)$ for $0 \le i \le n$, K be the maximal constant used in any of the A_i, and let $[K] = \{0, 1, 2, \ldots, K, \infty\}$. Let B be the maximal number of context switches in any run of $\mathcal{A}$. We construct the MPS $\mathcal{M} = (S, S_0, St, \Gamma, \Delta)$ where

1. S is a finite set of locations $(L'_1 \times [K]^{|\mathcal{X}_1|}) \times \ldots (L'_n \times [K]^{|\mathcal{X}_n|}) \times (A_w \times p)$, where $w \in \{1, \ldots, n\}$ represents the active automaton and $0 \le p \le B$ is a number that keeps track of context switches in the CTA.
2. $L'_i = L_i \cup \{l_t, l_t^p, l_{ta}^p \mid l \in L_i, t \in [K], a \in \Sigma, p \in \{W_{j,i}, R_{j,i} \mid 1 \le j \le n\}\}$.
3. The set of initial locations S_0 is $(L_1^0 \times 0^{|\mathcal{X}_1|}) \times \cdots \times (L_n^0 \times 0^{|\mathcal{X}_n|}) \times \bigcup_{1 \le p \le n} (A_p \times 0)$,
4. St is a finite set of stacks : each channel $c_{i,j}$ of $\mathcal{A}$ is simulated in the MPS using stacks $W_{i,j}$ and $R_{i,j}$.
5. $\Gamma = \Sigma \cup [K] \cup (\Sigma \times [K])$ is a finite stack alphabet, and $\Delta = \Delta_{int} \cup \Delta_{push} \cup \Delta_{pop}$ is the transition relation.

For $i_0, i_1, \ldots, i_B \in \{1, 2, \ldots, n\}$, let A_{i_j} represent the active automaton in context $0 \le j \le B$. We now explain below the transitions in the MPS $\mathcal{M}$. For each run in the CTA $\mathcal{A}$, we show that there is a run in the BMPS $\mathcal{M}$ preserving reachability; moreover, the content of each channel $c_{i,j}$ is retrieved from stacks $W_{i,j}, R_{i,j}$ in $\mathcal{M}$.

Context 0 in the CTA. In the 0th context of the CTA, A_{i_0} writes into some of the channels to which it can write, and also does some internal transitions. All automata other than A_{i_0} only participate in internal transitions. In $\mathcal{M}$, let us start from the location $((l_0^1, 0^{|\mathcal{X}_1|}) \ldots, (l_0^n, 0^{|\mathcal{X}_n|}), (A_{i_0}, 0))$, and all stacks empty. Internal transitions in any A_i are handled by updating the corresponding pair (l_i, ν_i) in $\mathcal{M}$, $l_i \in L_i$ by updating the control locations l_i, and the tuple ν_i taking care of resets. These transitions are all in Δ_{int}.

Consider the first transition involving a write into some channel $c_{i_0,j}$ by A_{i_0}. Let m be the message written. Let the transition in A_{i_0} be $(p, g, c_{i_0,j}!m, Y, q)$. Then in the MPS $\mathcal{M}$, we have the transition in Δ_{push} which updates $(p, \nu) \in L_{i_0} \times [K]^{|\mathcal{X}_{i_0}|}$ to (q, ν'), where ν' is obtained by resetting clocks $Y \subseteq \mathcal{X}_{i_0}$, checks guard g on ν, and pushes m to stack $W_{i_0,j}$. All tuples $(l, \nu_l) \in L_i \times [K]^{|\mathcal{X}_i|}$, $i \neq i_0$ are left unchanged. After the first write, any time elapse $t \in [K]$ is taken care of by transitions in Δ_{push} which not only update the clock values, but also push t to all stacks.[3] The next write (say to channel $c_{i_0,k}$) is handled similar to the first write, by pushing the message onto stack $W_{i_0,k}$ and updating the finite control of $\mathcal{M}$. Subsequent time elapses are pushed to all stacks. To summarize, simulation of context 0 in $\mathcal{M}$ results in stacks $W_{i_0,j}$ consisting of elements of the form $\Sigma \cup [K]$ (messages from Σ written on channels $c_{i_0,j}$ and time elapses $t \in [K]$ between messages). Stacks $W_{i,j}$ with $i \neq i_0$ and all stacks $R_{i,j}$ contain only symbols from $[K]$ denoting time elapses.

Context h, $h > 0$ in the CTA. In context h, A_{i_h} is the active automaton, and reads from some fixed channel c_{k,i_h}. It can write to several channels $c_{i_h,j}$, all different from c_{k,i_h}. The context switch from $h-1$ to h takes place when A_{i_h} is ready for writing or reading, and $A_{i_{h-1}} \neq A_{i_h}$, or A_{i_h} is ready to read off some channel c_{k,i_h} and $A_{i_{h-1}} = A_{i_h}$, but $A_{i_{h-1}}$ was reading off a channel $c_{k',i_{h-1}} \neq c_{k,i_h}$. This fact is reflected by updating $(A_{i_{h-1}}, h-1)$ in the control of $\mathcal{M}$ to (A_{i_h}, h). Writes made by A_{i_h} to channels $c_{i_h,j}$ are handled by pushing messages to stack $W_{i_h,j}$ and updating the finite control of $\mathcal{M}$ pertaining to A_{i_h}. Time elapses made during this context are pushed to all stacks. Assume A_{i_h} is ready to read a message from some channel c_{k,i_h}. If $h = 1$, k must be i_0 since A_{i_0} was active in context 0, and no other automaton has written any message so far.

If A_{i_h} has never read before from channel c_{k,i_h}, then all messages written into channel c_{k,i_h} so far are stored in stack W_{k,i_h}, along with time elapses after each message. However, the messages are stored in the reverse order in W_{k,i_h}. We pop W_{k,i_h} and store them into R_{k,i_h}, and simulate the read by popping R_{k,i_h}. However, if A_{i_h} has read from c_{k,i_h} in an earlier context, then the stack R_{k,i_h} may be non-empty. In this case, we first read off from R_{k,i_h}, before popping W_{k,i_h}. In any case, we first check if R_{k,i_h} is non-empty before proceeding.

Let (p, ν) be the pair in the control location of $\mathcal{M}$ corresponding to A_{i_h} ($p \in L_{i_h}$). A read is enabled from p in A_{i_h} via the transition $(p, g, c_{k,i_h}?m \in I, Y, q)$.

1. We first check if R_{k,i_h} is empty: for this, we first change the control location (p, ν) to $(p^{R_{k,i_h}}, \nu)$.
2. If the top of the stack R_{k,i_h} is a time $t \in [K]$, we pop it and remember it in the finite control as $((p^{R_{k,i_h}})_t, \nu)$. Consecutive time tags are added and stored in the finite control : if $t' \in [K]$ is the top of stack R_{k,i_h} while in $((p^{R_{k,i_h}})_t, \nu)$, then it is updated to $((p^{R_{k,i_h}})_{t+t'}, \nu)$. Here, $t + t'$ is either $\leq K$ or is ∞ if the sum exceeds K. This is continued until we see some $(m, t'') \in \Sigma \times [K]$ on top of the stack R_{k,i_h}. Then (m, t'') is popped, and we know the age of m to be $t + t' + t''$ using the information $t + t'$ from the finite control $((p^{R_{k,i_h}})_{t+t'}, \nu)$. We simulate the transition $(p, g, c_{k,i_h}?(m \in I), Y, q)$ in A_{i_h} by checking if $\nu \models g$, $t + t' + t'' \in I$, then we update the finite control in $\mathcal{M}$ to $((q^{R_{k,i_h}})_{t+t'}, \nu')$, $\nu' = \nu[Y := 0]$. This is continued until R_{k,i_h} is empty. As usual, if a time elapse happens in between, it is pushed onto all stacks including R_{k,i_h}. When we encounter $\perp$ in R_{k,i_h}, and A_{i_h} is still ready to read from c_{k,i_h} then we have to pop W_{k,i_h}.
3. The first thing before popping W_{k,i_h} is to get the finite control of $\mathcal{M}$ to $(q^{W_{k,i_h}}, \nu')$ (assuming it was some $((q^{R_{k,i_h}})_{t+t'}, \nu')$ or $(q^{R_{k,i_h}}, \nu')$ or (q, ν'), $q \in L_{i_h}$).
4. We start popping W_{k,i_h}; time tags t on top of W_{k,i_h} are remembered in the finite control of $\mathcal{M}$ as usual, by updating it to $((q^{W_{k,i_h}})_t, \nu')$. We accumulate time tags until a message

[3] Note that during a time elapse t, we do two things : (1) update all ν_i to $\nu_i + t$ in all the n pairs, and (2) push t onto all stacks. To ensure that all the ν_is are updated to $\nu_i + t$, we can keep an additional bit in the control location of $\mathcal{M}$ which starts at 1, updates ν_1, and keeps incrementing the bit till n, when ν_n is updated to $\nu_n + t$, and then we push t onto all stacks. We push t to all stacks going in a fixed order. We choose not to dwell on these low level implementation details since it clutters notation.

$m \in \Sigma$ appears on top of W_{k,i_h}. If the finite control of $\mathcal{M}$ is $((q^{W_{k,i_h}})_{t+t'}, \nu')$, then we pop m from W_{k,i_h}, change the finite control to $((q^{W_{k,i_h}})_{t+t',m}, \nu')$ to remember m, and then push $(m, t+t')$ on R_{k,i_h}. After the push, the finite control is again updated to $((q^{W_{k,i_h}})_{t+t'}, \nu')$. Note that $t+t'$ is indeed the time that elapsed after m was written. This is continued until we see a $\perp$ in W_{k,i_h}. Then we have transferred all messages written so far, to the stack R_{k,i_h} in the correct order, along with the ages. Elements in stack R_{k,i_h} have the form $\Sigma \times [K]$ (when transferred from W_{k,i_h}) or $[K]$ (a time elapse which is pushed). The finite control is updated again to $(q^{R_{k,i_h}}, \nu')$ to signify reading from R_{k,i_h}.

5. The context h may finish before R_{k,i_h} is empty, in which case, we will continue reading from it when the next context of A_{i_h} appears again, assuming A_{i_h} still reads from channel c_{k,i_h}. The other possibility is that R_{k,i_h} is emptied in this context.
6. If stack R_{k,i_h} is emptied while in context h, the finite control of $\mathcal{M}$ is updated to (q, ν') from $(q^{R_{k,i_h}}, \nu')$ or $((q^{R_{k,i_h}})_t, \nu')$. If W_{k,i_h} is empty, then there are no more pops to be done while in this context, since A_{i_h} can only write to some of its channels now. If a context switch happens before R_{k,i_h} is emptied, then the finite control of $\mathcal{M}$ pertaining to A_{i_h} is updated to (q, ν'). The finite control (s, ν_s) of $\mathcal{M}$ pertaining to $A_{i_{h+1}}$ $(s \in L_{i_{h+1}})$ may either stay same if $A_{i_{h+1}}$ is enabled to write from s, or will be updated to some $(s^{R_{g,i_{h+1}}}, \nu_s)$ if $A_{i_{h+1}}$ is enabled to read from some channel $s_{g,i_{h+1}}$ in the $(h+1)$st context. In the case when $A_{i_{h+1}} = A_{i_h}$, then the context switch takes place since A_{i_h} is ready to read from another channel c_{k',i_h}. In this case, we update $(q^{R_{k,i_h}}, \nu')$ or $((q^{R_{k,i_h}})_t, \nu')$ to $(q^{R_{k',i_h}}, \nu')$.

It can be seen that the stack alphabet of stacks W_{i_c,i_d} is $\Sigma \cup [K]$ while that of stacks R_{i_c,i_d} is $[K] \cup (\Sigma \times [K])$.

F.2 Correctness of Construction

To show that $\mathcal{M}$ preserves reachability and channel contents, and to show that $\mathcal{M}$ is indeed bounded phase, we use the following lemmas.

Lemma 13. *If $\mathcal{A}$ is a bounded context CTA with atmost B context switches, then the MPS $\mathcal{M}$ constructed as above is bounded phase, with atmost $3B$ phase changes.*

Proof. Let $A_0, A_1, \ldots, A_B$ be the sequence of automata which are active in contexts $0, 1, \ldots B$ in a run of $\mathcal{A}$.

1. In contexts $i \in \{1, 2, \ldots, B\}$, assume that the active automaton A_i reads from some channel $c_{k_i,i}$. By construction of $\mathcal{M}$, we have stacks $W_{k_i,i}, R_{k_i,i}$ corresponding to each channel $c_{k_i,i}$. When we start a new context i of $\mathcal{A}$, we do the following.
   - As long as A_i is writing to channels, we push the respective messages to the respective W-channels. For example, a message m written to channel $c_{i,j}$ is pushed to stack $W_{i,j}$. A time elapse t in the ith context results in pushing t to all stacks. So far, there has been no pop of any stack in $\mathcal{M}$ while in context i of $\mathcal{A}$. Only when A_i is ready to read from a channel say $c_{k_i,i}$, do we start popping a stack; first we check if $R_{k_i,i}$ is non-empty, and if so pop that. This counts as a phase change. If $R_{k_i,i}$ becomes empty, and we have more read operations of $c_{k_i,i}$ in context i of $\mathcal{A}$, then we pop stack $W_{k_i,i}$ and transfer contents to $R_{k_i,i}$. This counts as another phase change. Finally, when $R_{k_i,i}$ has been populated, we pop $R_{k_i,i}$ to facilitate reading from $c_{k_i,i}$. This is the third phase change. There can be no more phase changes while in context i, since all messages written so far in channel $c_{k_i,i}$ are already in stack $R_{k_i,i}$: recall that A_i cannot write to $c_{k_i,i}$ since she reads from it; if any other automaton writes to $c_{k_i,i}$, then the context changes. Thus, we have 3 phase changes in $\mathcal{M}$ corresponding to the context switch i of $\mathcal{A}$. Note that the number of phase changes can be less than 3 if for instance, $R_{k_i,i}$ was non-empty in the beginning of the ith context, and does not get emptied (in this case, it is just 1 change of phase), or if $R_{k_i,i}$ is empty in the beginning of the ith context, and we pop $W_{k_i,i}$ followed by $R_{k_i,i}$ (2 phase changes).

2. If context i of $\mathcal{A}$ involves only writing to channels, then there are no phase changes involved in $\mathcal{M}$ corresponding to context i of $\mathcal{A}$.

Since we know that any run in $\mathcal{A}$ has $\leq B$ context switches, and since each context in $\mathcal{A}$ results in ≤ 3 phase changes in $\mathcal{M}$, the maximal number of phase changes in $\mathcal{M}$ is $\leq 3B$.

□

Lemma 14. *Starting from the initial configuration* $((l_1^0, \nu_1), \ldots, (l_n^0, \nu_n), \epsilon, \ldots, \epsilon)$ *of the CTA* $\mathcal{A}$*, assume that we reach configuration* $((p_1, \nu_1'), \ldots, (p_n', \nu_n'), w_1, \ldots, w_s)$ *in context* $j \leq B$ *in a run of* $\mathcal{A}$*. Let* A_{i_j} *denote the automaton which is active in context* $0 \leq j \leq B$ *of this run. Then, starting from an initial location* $((l_1^0, \nu_1), \ldots, (l_n^0, \nu_n), (A_{i_0}, 0))$ *in* $\mathcal{M}$*, there is a run which leads to the location* $((p_1, \nu_1'), \ldots, (p_n', \nu_n'), (A_{i_j}, j))$*. Moreover, the content* $(\Sigma \times [K])^*$ *of any channel* $c_{k,l}$ *can be obtained from stacks* $R_{k,l}$ *and* $W_{k,l}$*.*

Proof. The proof is by construction of $\mathcal{M}$. Assume we start with an initial location $((l_1^0, \nu_1), \ldots, (l_n^0, \nu_n), (A_{i_0}, 0))$ in $\mathcal{M}$. Then we assume that A_{i_0} writes in context 0 in $\mathcal{A}$. We prove the statement of the theorem for every possible context $0 \leq j \leq B$.

1. As long as we simulate context 0 of $\mathcal{A}$, we push messages $m \in \Sigma$ in stacks $W_{i_0,j}$ for each write of $m \in \Sigma$ on channel $c_{i_0,j}$, and push time elapses t that happened while in context 0, to all stacks. Consider the last configuration of $\mathcal{A}$ in context 0 of the run seen so far; let it be $((l_1, \nu_1'), \ldots, (l_n, \nu_n'), w_1, \ldots, w_s)$. By construction of $\mathcal{M}$, we obtain $((l_1, \nu_1'), \ldots, (l_n, \nu_n'), (A_{i_0}, 0))$. All the R-stacks are populated with elements from $[K]$; while stacks $W_{i_0,j}$ corresponding to channels $c_{i_0,j}$ to which A_{i_0} wrote a message will contain elements from $\Sigma \cup [K]$; finally W-stacks corresponding to channels where A_{i_0} did not write, also has elements from $[K]$.
   Consider a channel $c_{i_0,j}$ to which A_{i_0} wrote messages $m_1, \ldots, m_p$ at times $t_1, t_2, \ldots, t_p$. If t is the current global time, then the age of m_i is $t - t_i$. By construction of $\mathcal{M}$, we will have in stack $W_{i_0,j}$, message m_i, and we have $t_{i+1} - t_i \in [K]$ on top of m_i (we will have $t_{i+1} - t_i$ 1's or a combination of elements from $[K]$ which sums up to $t_{i+1} - t_i \in [K]$). We also have m_{i+1} on top of $t_{i+1} - t_i$, and we have $t_{i+2} - t_{i+1}$ on top of m_{i+1}, and m_{i+2} on top of $t_{i+2} - t_{i+1}$ and so on. The topmost element of $W_{i_0,j}$ is $t - t_p$, and the one below this element is m_p. To retrieve the contents of channel $c_{i_0,j}$, we have to simply pop $W_{i_0,j}$ as follows: remember $t - t_p$ in the finite control. When m_p is popped, tag $t - t_p$ to it obtaining $(m_p, t - t_p)$. Pop $t_p - t_{p-1}$ and add it to the time tag in the finite control, obtaining $t - t_{p-1}$ in the finite control. When m_{p-1} is popped, tag $t - t_{p-1}$ obtaining $(m_{p-1}, t - t_{p-1})$. Continuing like this, we obtain $(m_1, t - t_1)$. The contents of channel $c_{i_0,j}$ at the end of context 0 can be retrieved as $(m_p, t - t_p) \ldots (m_1, t - t_1)$.
2. Assume we are in context j of $\mathcal{A}$. The active automaton is A_{i_j}. Let A_{i_j} read from channel $c_{k_{i_j}, i_j}$ in context j. At the start of context j, by construction of $\mathcal{M}$, we have two possibilities for stacks $R_{k_{i_j}, i_j}$ and $W_{k_{i_j}, i_j}$:
   (1) either stack $R_{k_{i_j}, i_j}$ contains only symbols from $[K]$ and $W_{k_{i_j}, i_j}$ contains symbols from $\Sigma \cup [K]$, or
   (2) $R_{k_{i_j}, i_j}$ contains symbols from $(\Sigma \times [K]) \cup [K]$ and $W_{k_{i_j}, i_j}$ contains symbols from $\Sigma \cup [K]$.
   If (1), then either channel $c_{k_{i_j}, i_j}$ was never read so far in $\mathcal{A}$ and the entire channel content is in $W_{k_{i_j}, i_j}$. The other possibility is that $c_{k_{i_j}, i_j}$ was read in an earlier context, and A_{i_j} read all the contents of $c_{k_{i_j}, i_j}$ at that time, and the subsequent writes to $c_{k_{i_j}, i_j}$ are stored in $W_{k_{i_j}, i_j}$. In case of (2), channel $c_{k_{i_j}, i_j}$ was read in an earlier context, but the channel was not completely read that time; the remaining contents of $c_{k_{i_j}, i_j}$ from that context are in $R_{k_{i_j}, i_j}$, along with possible time elapses since then. All subsequent writes to $c_{k_{i_j}, i_j}$ after that context are stored in $W_{k_{i_j}, i_j}$.
   In case of (1), in the jth context, the contents of $W_{k_{i_j}, i_j}$ are shifted to $R_{k_{i_j}, i_j}$. At the end of context j, if $R_{k_{i_j}, i_j}$ is non-empty, then the contents of $R_{k_{i_j}, i_j}$ top-down is the content of

channel $c_{k_{i_j},i_j}$ (if there are elements from $[K]$ on top, they must be added to the ages of subsequent (m,t) below). In case of (2), in the jth context, we start reading off $R_{k_{i_j},i_j}$. At the end of the jth context, if $R_{k_{i_j},i_j}$ is over $(\Sigma \times [K]) \cup [K]$ and $W_{k_{i_j},i_j}$ is over $\Sigma \cup [K]$, then the contents of channel $c_{k_{i_j},i_j}$ is obtained by first popping $R_{k_{i_j},i_j}$, remembering the topmost elements from $[K]$ in finite control by adding them, and then adding these to the ages of the remaining elements of the form (m,t). Let $w_2 \in (\Sigma \times [K])^*$ be the string so formed after popping $R_{k_{i_j},i_j}$. Once $R_{k_{i_j},i_j}$ is empty, we pop $W_{k_{i_j},i_j}$ in a similar manner. Let $w_1 \in (\Sigma \times [K])^*$ be the string so formed after popping $W_{k_{i_j},i_j}$. The contents of channel $c_{k_{i_j},i_j}$ at the end of context j is then obtained as $w_1 w_2$.

It is easy to see that the finite control of $\mathcal{M}$ is $((l_1,\mu_1),\dots,(l_n,\mu_n),(A_{i_j},j))$ iff in $\mathcal{A}$ we reach (l_i,μ_i) in A_i in context j. Moreover, as seen above, the channel contents at each step of the run can be retrieved from the corresponding stacks in $\mathcal{M}$. Thus, $\mathcal{M}$ preserves reachability, both of control locations as well as channel contents. Finally, the number of phase changes in $\mathcal{M}$ depends on the number of context switches in $\mathcal{A}$. ☐

F.3 Illustration of Theorem 9: CTA to MPS

We first show a sequence of context switches (≤ 10) on the CTA in Figure 13. The maximum number of switches happens when we start with A_2 with clock $y = 0$. It can be seen that for each value of $y = 0,1,2,3,4$ there can be a switch of context. An example run is below.

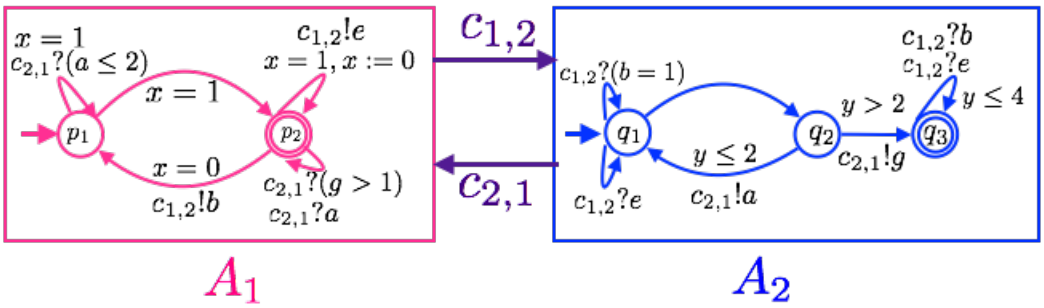


Figure 13 A bounded context CTA.

1. To begin, A_2 writes several as in context 0 in channel $c_{2,1}$ when $y = 0$.
   $c_{2,1} : (a,0)(a,0)$, $c_{1,2} : \epsilon$
2. A switch happens and A_1 writes a e,b in $c_{1,2}$ when $y = 1$.
   $c_{2,1} : (a,1)(a,1)$, $c_{1,2} : (b,0)(e,0)$
3. A_2 again writes some as when $y = 1$.
   $c_{2,1} : (a,0)(a,1)(a,1)$, $c_{1,2} : (b,0)(e,0)$
4. A switch to A_1 results in reading off the leading as (age 2) from $c_{2,1}$ and writing another e,b when $y = 2$ to $c_{1,2}$.
   $c_{2,1} : (a,1)(a,2)(a,2)$, $c_{1,2} : (b,1)(e,1)$ becomes $c_{2,1} : (a,1)$, $c_{1,2} : (b,0)(e,0)(b,1)(e,1)$
5. Now A_2 reads the first e,b (age 1) from $c_{1,2}$ and writes some as when $y = 2$ on $c_{2,1}$.
   $c_{2,1} : (a,0)(a,1)$, $c_{1,2} : (b,0)(e,0)$
6. A_1 takes over, and reads off the as from $c_{2,1}$ writes the e,b when $y = 3$ to $c_{1,2}$.
   $c_{2,1} : (a,1)(a,2)$, $c_{1,2} : (b,1)(e,1)$ becomes $c_{2,1} : (a,1)$, $c_{1,2} : (b,0)(e,0)(b,1)(e,1)$
7. A_2 reads off the e,b of age 1 from $c_{1,2}$ and moves to q_3 writing g.
   $c_{2,1} : (g,0)(a,1)$, $c_{1,2} : (b,0)(e,0)$
8. Back in A_1, the last set of as are read from $c_{2,1}$ and an e is written to $c_{1,2}$ when $y = 4$.
   $c_{2,1} : (g,1)(a,2)$, $c_{1,2} : (b,1)(e,1)$ becomes $c_{2,1} : (g,1)$, $c_{1,2} : (e,0)(b,1)(e,1)$

9. Back in A_2, the b, es are read with $y = 4$.
$c_{2,1} : (g, 1)$, $c_{1,2} : (e, 0)$

10. Switch back to A_1, read the g, $y = 5$.
$c_{2,1} : (g, 2)$, $c_{1,2} : (e, 1)$ becomes $c_{2,1} : \epsilon$, $c_{1,2} : (e, 1)$.

No more context switches are possible. Consider the following run of the CTA given in Figure 13.
$\mathcal{N}_0 = ((p_1, 0), (q_1, 0), \epsilon, \epsilon) \xrightarrow{*} \mathcal{N}_1 = ((p_1, 0), (q_1, 0), \epsilon, (a, 0)(a, 0)) \xrightarrow{*} \mathcal{N}_2 = ((p_2, 1), (q_2, 1), \epsilon, (a, 1)(a, 1)) \xrightarrow{*}$
$\mathcal{N}_3 = ((p_1, 1), (q_2, 2), (b, 1)(e, 1), (a, 2)(a, 2)) \xrightarrow{*} \mathcal{N}_4 = ((p_1, 1), (q_2, 2), (b, 1)(e, 1), (a, 2)) \xrightarrow{*} \mathcal{N}_5 = ((p_1, 1), (q_1, 2), \epsilon, (a, 0)(a, 2)) \xrightarrow{*} \mathcal{N}_6 = ((p_2, 2), (q_3, 3), \epsilon, (g, 0)(a, 3))$. In tables 2, 3 and 4, we show the sequence of locations along with the stack contents of the MPS that correspond to each $\mathcal{N}_i$. Tables 2, 3 and 4 give a run of the CTA and the corresponding run in the MPS.

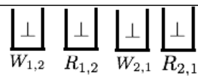
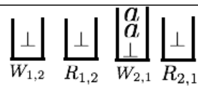
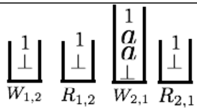
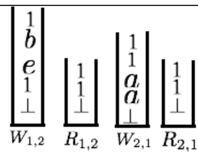
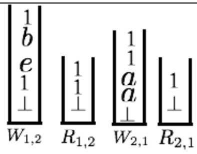
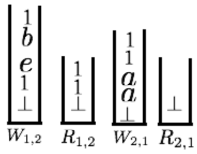
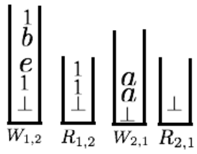
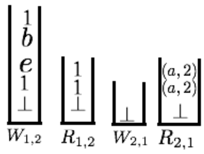
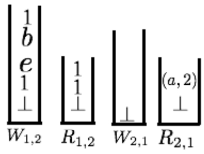
CTA	BMPS locations reached	BMPS stacks
$\mathcal{N}_0$	$(p_1, 0), (q_1, 0), (A_2, 0)$	
$\mathcal{N}_1$	$(p_1, 0)(q_1, 0), (A_2, 0)$	
$\mathcal{N}_2$	$(p_2, 1)(q_2, 1), (A_2, 0)$	
$\mathcal{N}_3$	$(p_1^{R_{21}}, 1)(q_2, 2), (A_1, 1)$ the R_{21} in $p_1^{R_{21}}$ indicates that the next pop is from R_{21}. $(A_2, 0)$ is updated to $(A_1, 1)$ on the switch and now A_1 is ready to read.	
$\mathcal{N}_4$	$((p_1^{R_{21}})_1, 1)(q_2, 2), (A_1, 1)$ The 1 in $()_1$ is the time tag read off from R_{21}. This becomes 2 when the next 1 is read off from R_{21}. On seeing $\perp$ in stack R_{21}, the superscript R_{21} in the location is changed to W_{21} making it $p_1^{W_{21}}$. $(p_1^{W_{21}}, 1)(q_2, 2), (A_1, 1)$ $((p_1^{W_{21}})_2, 1)(q_2, 2), (A_1, 1)$ This becomes $((p_1^{W_{21}})_{2a}, 1)(q_2, 2), (A_1, 1)$ when the a on top of W_{21} is read. $(a, 2)$ is pushed to R_{21} and the control comes back to $((p_1^{W_{21}})_2, 1)(q_2, 2), (A_1, 1)$. This is repeated for the second a in W_{21}, pushing one more $(a, 2)$ to R_{21}. On seeing $\perp$ in W_{21}, $(p_1^{W_{21}})_2$ is changed to $p_1^{R_{21}}$. $(p_1^{R_{21}}, 1)(q_2, 2), (A_1, 1)$ $(p_1, 1)(q_2, 2), (A_1, 1)$	    

Table 2

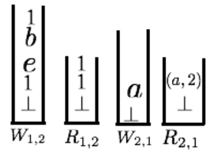
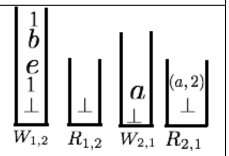
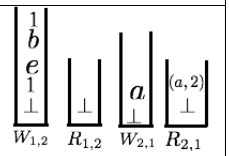
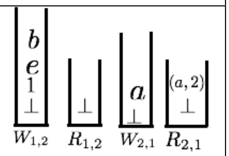
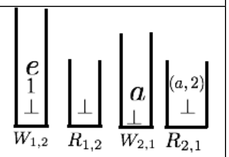
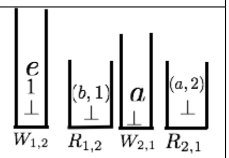
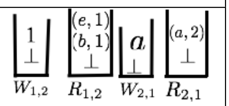
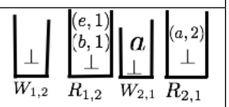
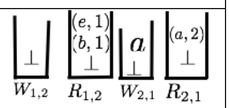
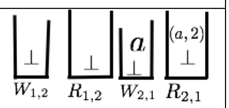
CTA	BMPS locations	BMPS stacks
$\mathcal{N}_5$	$(p_1,1)(q_1,2),(A_2,2)$ $(A_1,1)$ is updated to $(A_2,2)$, and A_2 has written an a	
	$(p_1,1)((q_1^{R_{1,2}})_2,2),(A_2,2)$	
	$(p_1,1)(q_1^{W_{1,2}},2),(A_2,2)$	
	$(p_1,1)((q_1^{W_{1,2}})_1,2),(A_2,2)$	
	$(p_1,1)((q_1^{W_{1,2}})_{1b},2),(A_2,2)$	
	$(p_1,1)((q_1^{W_{1,2}})_1,2),(A_2,2)$	
	$(p_1,1)((q_1^{W_{1,2}})_1,2),(A_2,2)$	
	$(p_1,1)((q_1^{W_{1,2}})_2,2),(A_2,2)$	
	$(p_1,1)(q_1^{R_{1,2}},2),(A_2,2)$	
	$(p_1,1)(q_1,2),(A_2,2)$	

Table 3

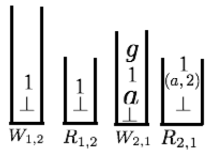
CTA	BMPS locations	BMPS stacks
$\mathcal{N}_6$	$(p_2, 2)(q_3, 3), (A_1, 3)$ While in $(A_2, 2)$ we move from q_1 to q_2 in A_2, and p_1 to p_2 in A_1. Elapse a unit of time at q_2, and goto q_3, writing g. $(A_2, 2)$ is updated to $(A_1, 3)$, since A_1 can read a from p_2.	

Table 4