What’s Decidable about Perfect Timed Channels?

Parosh Aziz Abdulla¹, Mohamed Faouzi Atig¹, and Shankara Narayanan Krishna²

¹ Uppsala University, Sweden. {parosh,mohamed_faouzi.atig}@it.uu.se
² IIT Bombay, Mumbai, India. krishnas@cse.iitb.ac.in

Abstract. In this paper, 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 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., [3,2,5,1,19,6,13,12,11,15]). In particular, perfect channel systems have been extensively studied as a formal model for communicating protocols [8,18]. Unfortunately, perfect channel systems are in general Turing powerful, and hence all basic decision problems (e.g., the reachability problem) are undecidable for them [8]. To circumvent this undecidability obstacle, several approximate techniques have been proposed in the literature including making the channels lossy [4,10], restricting the communication topology to polyforest architectures [18,16], or using half-duplex communication [9]. 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) [16]. 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 [7].

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 [14] in which the heart and the pacemaker can be modelled using two timed automata communicating through perfect channels and global variables. (A high-level description of the implantable cardiac medical devices can be found in Section 2). 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 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 in say 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., 32,41,19,6,13,21,1,15). The two
closest to ours are those presented in 11,15. 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 11 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 11. The work presented in 15 considers dense clocks with urgent semantics.
In 15, 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.

2 A Motivating Example: Implantable Cardiac Devices

In this section, we provide a high level description of how CTA can be used to
model implantable cardiac medical devices 14. We do not delve into low level
details on how the timed automata are implemented in each case. The electrophysiological functioning of the heart helps in assessing complex arrhythmias
resulting from irregular heartbeats. A healthy heart beats between 60 and 100
times a minute. Arrhythmias can manifest as bradycardia (slower heart rate) or
tachycardia (faster heart rate), both of which happen as a consequence of a lack
of synchronization between the contractions of the left and the right ventricle,
resulting in less blood supply to the body. A pacemaker is a small implantable
device which detects either of the two situations and fires electrical impulses to
the ventricles resulting them to correct their rate.

Consider a CTA consisting of two timed automata modeling the heart and
pacemaker respectively. We do not get into the minute details of these timed
automata and focus on how communication takes place between the heart and
the pacemaker using channels and global variables. There are two channels from
the pacemaker automaton to the heart automaton signifying the leads of the
pacemaker which are connected to the left and right ventricles. The pacemaker
automaton and the heart automaton have their own local clocks which are reset
each time they grow to 60, signifying elapse of one minute. In addition, there

\[ 32,41,19,6,13,21,1,15 \]
are two global variables \( L, R \) in the heart automaton which keep track of the number of times the left/right ventricle contracts in a minute. The pacemaker automaton has a local clock \( X \). If \( L < 60 \) (or \( R < 60 \)) when \( X = 60 \), then it signifies bradycardia and the pacemaker automaton sends impulses through the respective channel connecting it to the heart automaton. This signifies the pacemaker sending impulses to the left/right ventricle. Depending on the heart-rate, the electrical-discharge interval of the pacemaker is adjusted. For instance, if the rate needs to be accelerated very quickly due to a very slow heart rate, the impulses must be delivered almost immediately, while if the rate is closer to the normal rate, then it is delivered after a small delay. This is modeled as the age of the impulses sent through the channel: for instance, if \( L \in (40, 50) \), then the impulses must be delivered between 10 and 20 seconds, while if it is 30, then it must be delivered between 0 and 5 seconds. This is captured as constraints on the ages of the impulses in the channel: if \( L \in (40, 50) \), then the impulses must be received by the heart automaton when their age is in \((10, 20)\). However, if \( L \in (20, 30) \), then the impulses must be read when their age is in \((0, 5)\). For those interested in more details, [14] proposes timed automata to model the heart and pacemaker communicating with each other using channels and global variables.

3 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 \( \infty \). 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 \( \nu : \mathcal{X} \rightarrow \mathbb{N} \). 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 \( \nu \) as a point \((\nu(x_1), \nu(x_2), \ldots, \nu(x_n)) \in \mathbb{N}^{|\mathcal{X}|}\). For a subset of clocks \( \mathcal{X} \subseteq 2^\mathcal{X} \) and valuation \( \nu \in \mathbb{N}^{|\mathcal{X}|} \), we write \( \nu[X:=0] \) for the valuation where \( \nu[X:=0](x) = 0 \) if \( x \notin \mathcal{X} \), 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 \( x \in \mathcal{X} \). The valuation \( 0 \in \mathbb{N}^{|\mathcal{X}|} \) is a special valuation such that \( 0(x) = 0 \) for all \( x \in \mathcal{X} \). A clock constraint over \( \mathcal{X} \) is defined by a (finite) conjunction of constraints of the form \( x \leq k \), where \( k \in \mathbb{N} \), \( x \in \mathcal{X} \), and \( k \in \{<, \leq, =, >, \geq\} \). 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 \( \nu \) satisfies constraint \( g \). For example, \((1, 0, \infty) \models (x_1 < 2) \land (x_2 = 0) \land (x_3 > 1)\). Let \( Act \) denote a finite set called actions.

Timed Automata A timed automaton (TA) is a tuple \( A = (L, L^0, Act, \mathcal{X}, E, F) \) such that (i) \( L \) is a finite set of locations, (ii) \( \mathcal{X} \) is a finite set of clocks, (iii) \( E \subseteq L \times \varphi(\mathcal{X}) \times Act \times 2^\mathcal{X} \times L \) is a finite set of transitions, (iv) \( L^0, F \subseteq L \) 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 \( s = (t, \nu) \in L \times \mathbb{N}^{|\mathcal{X}|} \). A transition \((t, e)\) from a state \( s = (t, \nu) \) to a state \( s' = (t', \nu') \) is written as \( s \xrightarrow{t, e} s' \) if \( e = (t, g, a, Y, t') \in E \), such that \( a \in Act, \nu + t \models g \), and \( \nu' = (\nu + t)[Y:=0](x) \). A run is a finite sequence \( \rho = s_0 \xrightarrow{t_1, e_1} s_1 \xrightarrow{t_2, e_2} s_2 \cdots \xrightarrow{t_n, e_n} s_n \) of states and transitions.
A is non-empty iff there is a run from an initial state $(l_0, 0)$ to some state $(f, \nu)$ where $f \in F$.

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

4 Communicating Timed Automata (CTA)

A communicating timed automata (CTA) is a tuple $N = (A_1, \ldots, A_n, C, \Sigma, T)$ where each $A_i$ is a timed automaton, $C$ is a finite set of FIFO channels, $\Sigma$ is a finite set called the channel alphabet, and $T$ is a network topology. The network topology is a directed graph $\{(A_1, \ldots, A_n), C\}$ comprising of the finite set of timed automata $A_i$ as nodes, and the channels $C$ as edges. $C$ is given as a tuple $(c_{i,j})$; the channel from $A_i$ to $A_j$ is denoted by $c_{i,j}$, with the intended meaning that $A_i$ writes a message from $\Sigma$ to channel $c_{i,j}$ and $A_j$ reads from channel $c_{i,j}$. We assume that there is atmost one channel $c_{i,j}$ from $A_i$ to $A_j$, for any pair $(A_i, A_j)$ of timed automata. Figure 1 illustrates the definition.

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

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 $A_i$ 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 $A_i$ 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 $<a, b>$ with $b \in N$ and $a \in N \setminus \{\infty\}$, “<” stands for left-open or left-closed and “>” for right-open or right-closed. In this case, the timed automaton $A_i$ 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 $I$.

A CTA is said to have global clocks if $g \in \varphi(X)$ (and not necessarily $g \in \varphi(X_i)$) in the transitions $(l_i, g, op, Y, l_i')$ of $E_i$. Thus, in a CTA with global clocks, an
automaton $A_l$ can check guards on clocks from $X_j, j \neq i$; however it cannot update clocks from $X_j, j \neq i$.

**Configurations.** The semantics of $\mathcal{N}$ is given by a labeled transition system $\mathcal{L}_\mathcal{N}$. A configuration $\gamma$ 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 $A_i$, and $\nu_i$ gives the valuations of clocks $X_i$, $1 \leq i \leq n$, where $\nu_i \in \mathbb{N}[X_i]$. $c = (c_{i,j})$, 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 $\gamma_1 = ((l_i, \nu_i), \ldots, (l_n, \nu_n), c)$ and $\gamma_2 = ((l'_i, \nu'_i), \ldots, (l'_n, \nu'_n), c')$ be two configurations. The transitions $\rightarrow$ in $\mathcal{L}_\mathcal{N}$ are of two kinds:

1. Timed transitions $\xrightarrow{\nu}$. These transitions denote the passage of time $t \in \mathbb{N}$.
   
   $\gamma_1 \xrightarrow{\nu} \gamma_2$ if $l_i = l'_i$, and $\nu'_i = \nu_i + t_i$ for all $i$ and $c' = c + t$.

2. Discrete transitions $\xrightarrow{D}$. These are of the following kinds:
   
   (1) $\gamma_1 \xrightarrow{g,c_{i,j}} \gamma_2$ : there is a transition $l_i \xrightarrow{g,c_{i,j}} \gamma_1$ in $E_i$, $\nu_i = g$, $\nu'_i = \nu_i[Y := 0]$ 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) $\gamma_1 \xrightarrow{g,c_{i,j}[a]} \gamma_2$ : 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 $l_i \xrightarrow{g,c_{i,j}[a]} \gamma_1$ in $E_i$, $\nu_i = g$, $\nu'_i = \nu_i[Y := 0]$, $c_{i,j} = w \in (\Sigma \times \mathbb{N})^+$ and $c'_{i,j} = (a, 0).w$.

   (3) $\gamma_1 \xrightarrow{g,c_{i,j}[(a \in l)]Y} \gamma_2$ : 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 $l_i \xrightarrow{g,c_{i,j}[(a \in l)]Y} \gamma_1$ in $E_i$, $\nu_i = g$, $\nu'_i = \nu_i[Y := 0]$, $c_{j,i} = w.(a, t) \in (\Sigma \times \mathbb{N})^+$, $t \in T$ and $c'_{j,i} = w \in (\Sigma \times \mathbb{N})^+$.

**Reachability.** The initial state of $\mathcal{L}_\mathcal{N}$ is defined by $\gamma_0 = ((l_0^0, \nu_0^0), \ldots, (l_n^0, \nu_n^0), c^0)$ where $l_i^0$ is the initial location of $A_i$, $\nu_i^0 = 0$ 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 $\gamma_0$, we can reach some configuration $\gamma$.

## 5 Acyclic CTA

In this section, we look at the reachability problem in CTA whose underlying network topology $T$ is somewhat restrictive. An *acyclic CTA* is a CTA $\mathcal{N} = (A_1, \ldots, A_n, C, \Sigma, T)$ which has no cycles in the underlying undirected graph of $T$. Such topologies are called polyforest topologies in [11]. The next 3 subsections focus on answering the reachability question in acyclic CTA with and without global clocks: we find the thin boundary line which separates decidable and undecidable acyclic CTAs.

---

3 Recall that the network topology $\{\{A_i, \ldots, A_n\}, C\}$ is a directed graph; the underlying undirected graph is obtained by considering all edges as undirected in this graph.
5.1 Reachability in Acyclic CTA 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.

**Proof.** It is known [10] 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 $N$ 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. 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 1: on the left are transitions of $A$; on the right, we depict corresponding transitions in $A_1$ (the red states) and in $A_2$ (yellow states). For $\text{nop}$ and write transitions of $A$, there are no corresponding widgets in $A_2$; read transitions of $A$ have corresponding widgets in $A_1$ and $A_2$.

**Fig. 1.** On the left, we show each transition in $A$ ($\text{nop}$ 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.

**Lemma 1.** Let $A$ be an untimed automaton with the perfect channel $c_{A,A}$ connecting $A$ to itself. Let $\rho$ be a run of $A$ beginning with the initial configuration $(s_0, \epsilon)$, reaching some configuration $(p, w), w \in \Sigma^*$. Then we have a corresponding run $\rho'$ in the constructed CTA $N$ starting with $(s_0, i, \epsilon)$ and reaching configuration $(p, i, w'), w' \in (\Sigma \times N)^*$ such that $\text{untime}(w') = w$. The converse direction simulating a run of $N$ in $A$ holds similarly.

See Appendix B for a detailed proof of Theorem 1.
5.2 Reachability in Acyclic CTA without Global Clocks

**Theorem 2.** The reachability problem is undecidable for acyclic CTA consisting of three timed automata without global clocks.

*Proof.* We prove the undecidability by reducing the halting problem for deterministic two counter machines (see Appendix C.1 for a formal definition). A deterministic two counter machine \( C \) consists of two counters \( c_1, c_2 \) and a finite set of instructions \( \{l_0, l_1, \ldots, l_n\} \). Each instruction either increments/decrements one of the two counters, and switches control to another instruction, or checks if a counter is 0. If the counter is 0, control switches to a chosen instruction, and if non-zero, it switches to another instruction. There is a specific instruction called the HALT instruction, from where nothing happens. The halting problem asks if starting with the initial instruction \( l_0 \) with counter values being 0, whether we can reach the HALT instruction.

Given a two counter machine \( C \), 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 \( l_i : \text{inc or dec } c, \text{ goto } l_j \), or a zero check instruction \( l_i : \text{if } c = 0, \text{ goto } l_j \text{ else goto } l_k \), we have a widget \( W^{A_{m}}_i \) in each \( A_m, m \in \{1, 2, 3\} \). The widgets \( W^{A_{m}}_i \) 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 \( W^{A_{m}}_i \) to the initial location \( l_j \) of widget \( W^{A_{m}}_i \).

We refer to initial/terminal locations (labelled \( p \)) in each \( W^{A_{m}}_i \) using the notation \( (W^{A_{m}}_i, p) \). 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. \( A_1 \) has clocks \( g_{A_1}, x_1 \); \( A_2 \) has clocks \( g_{A_2}, y_1 \); \( A_3 \) has clocks \( g_{A_3}, z_1 \). The clocks \( g_{A_i}, i = 1, 2, 3 \) are never reset. The values of \( g_{A_i} \) represent the total time elapse at any point.

**Encoding Counters.** 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, this means \( c_1^i = (\text{value of } g_{A_2} \text{ at } l_i) - (\text{value of } g_{A_1} \text{ at } l_i) \). Note that \( A_1, A_2 \) are not always in sync while simulating the two counter machine: \( A_1 \) can simulate the \( j \)th instruction \( l_j \) while \( A_2 \) is simulating the \( i \)th 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 \( c_1 \), if \( A_1 \) takes 2 units of time to go from \( l_i \) to \( l_j \) while \( A_2 \) takes just one unit, then the value of \( g_{A_1} \) at \( l_j \) is two more than what it was at \( l_i \); likewise, the value of \( g_{A_2} \)
at \( \ell_j \) is one more than what it was at \( \ell_i \). The channel alphabet is \( \{(\ell_i, c^+, \ell_j) \mid \ell_i : \text{inc } c \text{ goto } \ell_j \} \cup \{(\ell_i, c^-, \ell_j) \mid \ell_i : \text{dec } c \text{ goto } \ell_j \} \cup \{(\ell_i, c=0, \ell_j), (\ell_i, c>0, \ell_k) \mid \ell_i : \text{if } c = 0, \text{ then goto } \ell_j \}, \text{ else goto } \ell_k \} \cup \{\text{zero}_1, \text{zero}_2\} \).

1. Consider an increment instruction \( \ell_i : \text{inc } c \text{ goto } \ell_j \). The widgets \( W_i^{A_m} \) for \( m = 1, 2, 3 \) are described in Figure 2. The one on the left is while incrementing \( c_1 \), while the one on the right is obtained while incrementing \( c_2 \).

![Widgets corresponding to an increment \( c_1, c_2 \) instruction in \( A_1, A_2, A_3 \)](image)

2. The case of a decrement instruction is similar, and is obtained by swapping the speeds of the two automata \( (A_1, A_2 \text{ and } A_2, A_3) \) respectively in reaching \( \ell_j \) from \( \ell_i \).

3. We finally consider a zero check instruction of the form \( \ell_i : \text{if } c_1 = 0, \text{ then goto } \ell_j \), else goto \( \ell_k \). The widgets \( W_i^{A_m} \) for \( m = 1, 2, 3 \) are described in Figure 3. 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. \( \ell_i \) denotes the instruction seen at the \( i \)th step and \( c_1^i, c_2^i \) respectively are the values of counters \( c_1, c_2 \) after \( i \) steps. We represent a block of transitions as \( B_{i,i+1} = \{(W_i^{A_1}, \ell_i, \nu_i^{A_1}), \ldots, (W_i^{A_m}, \ell_i, \nu_i^{A_m})\} \). We obtain a run of each \( A_m \) as \( B_{0,1}, B_{1,2}, \ldots, B_{h-1,h}, \ldots \), where each block \( B_{h-1,h} \) of transitions in the widget \( W_h^{A_m} \) simulate the instruction \( \ell_h \), and shifts control to \( \ell_{h+1} \). For each \( m \), the notation \( (W_i^{A_1}, \ell_i, \nu_i^{A_1}) \) represents that \( A_m \) is at location \( \ell_j \) of widget \( W_i^{A_m} \) with clock valuation \( \nu_i^{A_m}, \nu_j^{A_m} \in \mathbb{N}^2 \) is the tuple giving the current values of the two clocks. Appendix B has proofs of the following lemmas.

**Lemma 2.** Let \( C \) 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 \( h \)th instruction \( \ell_h \). Then there is a run of \( N \) which passes through widgets \( W_0^{A_m}, W_1^{A_m}, \ldots, W_h^{A_m} \) 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 \( W_h^{A_2}, \ell_h \) and the value of clock \( g_{A_1} \) on reaching the initial location \( W_h^{A_1}, \ell_h \).
2. \( c_2^h \) is the difference between the value of clock \( g_{A_3} \) on reaching the initial location \( W_h^{A_3}, \ell_h \) and the value of clock \( g_{A_2} \) on reaching the initial location \( W_h^{A_2}, \ell_h \).
Fig. 3. Widgets corresponding to checking $c_1, c_2$ is 0. Let $\alpha = (\ell_i, c_1 = 0, \ell_j), \beta = (\ell_i, c_2 > 0, \ell_j)$, $\gamma = (\ell_i, c_1 = 0, \ell_j), \zeta = (\ell_i, c_2 > 0, \ell_j)$.

2. If $W_{A_1}^{h_0}$ is a zero check widget for $c_1$ ($c_2$) then $c_1^{h_0}$ ($c_2^{h_0}$) is 0 iff one reaches a terminal location of $W_{A_2}^{h_0}$ reading $\alpha$ ($\gamma$) and $\text{zero}_1$ ($\text{zero}_2$) with age 0.

Likewise, $c_1^{h_0}$ ($c_2^{h_0}$) is greater than 0 iff one reaches a terminal location of $W_{A_2}^{h_0}$ reading $\beta$ ($\zeta$) and $\text{zero}_1$ ($\text{zero}_2$) with age greater than 0.

Lemma 3. Machine $\mathcal{C}$ halts iff the halt widget $W_{halt}^{A_m}$ is reached in $\mathcal{N}, m=1,2$.

See Appendix C for detailed proofs of all lemmas. We now turn to the decidability result with two timed automata.

Theorem 3. The reachability problem is decidable for acyclic CTA consisting of two timed automata without global clocks.

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

Given CTA $\mathcal{N}$ with $A = (L_A, L_0^A, \mathcal{X}_A, \Sigma, E_A, F_A), B = (L_B, L_0^B, \mathcal{X}_B, \Sigma, E_B, F_B)$ and a channel $c_{A,B}$ from $A$ to $B$, we simulate $\mathcal{N}$ using a one counter automaton $\mathcal{O}$ as follows. We start with the region automata $\text{Reg}(A) = (Q_A, Q_0^A, \Sigma \cup \{\checkmark\}, E_A', F_A')$ and $\text{Reg}(B) = (Q_B, Q_0^B, \Sigma \cup \{\checkmark\}, E_B', F_B')$ 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 of $\text{Reg}(A)$ ($\text{Reg}(B)$) are of the form $L_A \times [K]^{|\mathcal{X}_A|} (L_B \times [K]^{|\mathcal{X}_B|})$. A transition $(l, \nu) \rightarrow (l, \nu + 1)$ denotes a time lapse of 1 in both $\text{Reg}(A), \text{Reg}(B)$. If $\nu(x) + 1$ exceeds $K$ for any clock $x$, then it is replaced with $\infty$. For each transition $e = (\ell, g, c_{A,B}!a, Y, \ell')$ in $A$ we have the
transition \( (l, \nu) \xrightarrow{e} (l', \nu') \) in \( \text{Reg}(A) \) if \( \nu \models g \), and \( \nu' = \nu[Y:=0] \). For each transition \( e = (l, g, c_{A,B})? (a \in I), Y, \ell' \) in \( B \) we have the transition \( (l, \nu) \xrightarrow{\nu \in \delta} (l', \nu') \) in \( \text{Reg}(B) \) if \( \nu \models g \), and \( \nu' = \nu[Y:=0] \).

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 \( A \) in our simulation of \( \text{Reg}(A), \text{Reg}(B) \). Internal transitions of \( A, B \) can be simulated by updating the respective control locations in \( \text{Reg}(A), \text{Reg}(B) \). 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.

The state space of \( \mathcal{O} \) consists of \( Q_A \times (Q_B \times (\Sigma \cup \{\epsilon\})) \times ([K]\setminus\{\infty\}) \) along with the unary stack with stack alphabet \( \{1\} \). The \( \Sigma \cup \{\epsilon\} \) in \( Q_B \times (\Sigma \cup \{\epsilon\}) \) is to remember the message (if any) written by \( A \), which has to be read by \( B \). The initial location of \( \mathcal{O} \) is \( \{(q_0^A, 0^{Q_A}), (q_0^B, 0^{Q_B}, \epsilon), 0 \} \mid q_0^A \in L_A, q_0^B \in L_B \) and the unary stack has the bottom of stack symbol \( \bot \) and a special symbol \( \theta \) just above \( \bot \) in the initial configuration. The transitions in \( \mathcal{O} \) are as follows: For \( l, l' \) states of \( \mathcal{O} \), internal transitions \( \Delta_{int} \) consist of transitions of the form \( (l, l') \); push transitions \( \Delta_{push} \) consist of transitions of the form \( (l, a, l') \) for \( a \in \{1, \theta\} \). Finally, we also have pop transitions \( \Delta_{pop} \) of the form \( (l, a, l') \) for \( a \in \{1, \theta\} \). We now describe the transitions.

1. Internal transitions \( \Delta_{int} \): Transitions of \( \Delta_{int} \) simulate internal transitions of \( \text{Reg}(A), \text{Reg}(B) \) as well as \( \checkmark \) transitions as follows:
   (a) For \( l = ((p, \nu_1), (q, \nu_2, \alpha), i) \) and \( l' = ((p', \nu_1'), (q, \nu_2, \alpha), i) \), we have the transition \( (l, l') \in \Delta_{int} \) if \( (p, \nu_1) \rightarrow (p', \nu_1') \) is an internal transition in \( \text{Reg}(A) \). The same can be said of internal transitions in \( \text{Reg}(B) \) updating \( q, \nu_2 \), leaving \( \alpha, i \) and \( (p, \nu_1) \) unchanged.
   (b) For \( l = ((p, \nu_1), (q, \nu_2, \alpha), i) \) with \( 0 < i < K \), \( l' = ((p, \nu_1), (q, \nu_2 + 1, \alpha), i + 1) \), \( (l, l') \in \Delta_{int} \) if \( (q, \nu_2) \xrightarrow{\nu_2} (q, \nu_2 + 1) \) is a \( \checkmark \)-transition in \( \text{Reg}(B) \). Note that \( i + 1 \leq K \).
(c) For $l = ((p, \nu_1), (q, \nu_2, \alpha), i)$ with $0 < i < K$, $l' = ((p, \nu_1 + 1), (q, \nu_2, \alpha), i - 1)$, $(l, l') \in \Delta_{int}$ if $(p, \nu_1) \rightarrow (p, \nu_1 + 1)$ is a $\checkmark$-transition in $Reg(A)$.

(d) For $l = ((p, \nu_1), (q, \nu_2, \epsilon), i), l' = ((p', \nu'_1), (q, \nu_2, \alpha), i)$, $(l, l') \in \Delta_{int}$ if $(p, \nu_1) \rightarrow (p', \nu'_1)$ is a transition in $Reg(A)$ corresponding to a transition from $p$ to $p'$ which writes $\alpha$ 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 $(q, \nu_2) \rightarrow_{a} (q', \nu'_2)$ 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 fact that the stack is empty (top of stack $= \theta$). See 2(c), 3(b), and then use transition $(l, l') \in \Delta_{int}$ for $l = ((p, \nu_1), (q, \nu_2, \epsilon), K), l' = ((p, \nu_1), (q, \nu_2, \alpha), K)$, if $(q, \nu_2) \rightarrow_{m} (r, \nu'_2)$ is a read transition in $Reg(B)$.

(g) To check that a message has age $\geq K$ when read, we need $i = K$, along with the fact that the stack is non-empty (top of stack $= 1$). See 2(d), 3(c), and then use transition $(l, l') \in \Delta_{int}$ for $l = ((p, \nu_1), (q, \nu_2, \epsilon), K), l' = ((p, \nu_1), (q, \nu_2, \alpha), K)$, if $(q, \nu_2) \rightarrow_{m} (r, \nu'_2)$ is a read transition in $Reg(B)$. (age requirements $\geq K$ are checked using this or the above).

2. Pop transitions $\Delta_{pop}$:

(a) If $(p, \nu_1) \rightarrow (p, \nu_1 + 1)$ 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. That is, for $l = ((p, \nu_1), (q, \nu_2, \alpha), K), l' = ((p, \nu_1 + 1), (q, \nu_2, \alpha), K), (l, l') \in \Delta_{pop}$.

(b) If $(p, \nu_1) \rightarrow (p, \nu_1 + 1)$ in $Reg(A)$, and if the counter value as stored in the finite control is $K$, and if the stack is empty, then the top of the stack is the symbol $\theta$. In this case, we pop $\theta$, reduce $K$ in the finite control to $K - 1$, and push back $\theta$ to the stack. We remember that $\theta$ has been popped in the finite control, so that we push it back immediately. For $l = ((p, \nu_1), (q, \nu_2, \alpha), K), l' = ((p, \nu_1 + 1), (q, \nu_2, \alpha), K - 1), (l, \theta, l') \in \Delta_{pop}$.

The location $p_\theta$ tells us that $\theta$ 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 $= \theta$). In this case, we pop $\theta$ and remember it in the finite control, and push it back. For $l = ((p, \nu_1), (q, \nu_2, \alpha), K), l' = ((p, \nu_1), (q, \nu_2, \alpha), K), (l, \theta, l') \in \Delta_{pop}$.

(d) To check that a message has age $\geq K$ 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 $l = ((p, \nu_1), (q, \nu_2, \alpha), K), l' = ((p, \nu_1), (q, \nu_2, \alpha), K), (l, 1, l') \in \Delta_{pop}$.

3. Push transitions $\Delta_{push}$:

(a) Push back $\theta$ to the stack while reducing counter value from $K$ to $K - 1$.

(\(l, \theta, l') \in \Delta_{push} \text{ for } l = ((p, \nu_1), (q, \nu_2, \alpha), K - 1) \text{ and } l' = ((p, \nu_1), (q, \nu_2, \alpha), K - 1)).

(b) Push back $\theta$ to the stack before checking the age of a message is $K$.

(\(l, \theta, l') \in \Delta_{push} \text{ for } l = ((p, \nu_1), (q_\theta, \nu_2, \alpha), K) \text{ and } l' = ((p, \nu_1), (q_\theta, \nu_2, \alpha), K)).
(c) Push back 1 to the stack before checking the age of a message is $> K$. $\Delta_{\text{push}}$ for \( l = ((p, \nu_1), (q^1, \nu_2, \alpha), K) \) and \( l' = ((p, \nu_1), (q^2, \nu_2, \alpha), K) \).

(d) If \((q, \nu_2) \xrightarrow{\nu} (q, \nu_2 + 1)\) in \(\text{Reg}(B)\), 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, \((l, l', l') \in \Delta_{\text{push}}\) for \( l = ((p, \nu_1), (q, \nu_2, \alpha), K) \) and \( l' = ((p, \nu_1), (q, \nu_2 + 1, \alpha), K) \).

Lemma 4. If the configuration in \(O\) has the form \(((l_A, \nu_A), (l_B, \nu_B, a), i)\) along with a stack consisting of \(1^{i+1} \theta \bot\) (top to bottom in the stack) then message \(a\) has age \(i + j\), while \(A\) is at \(l_A\) and \(B\) is at \(l_B\). Also, \(B\) is \(i + j\) time units ahead of \(A\).

Lemma 5. Let \(N\) 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 \(((p_A^0, 0)^{|X_A|}, (p_B^0, 0)^{|X_B|}, \epsilon)\) of \(N\), 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 \((l_B, \nu_2)\). Then in \(O\), starting from \(((p_A^0, 0)^{|X_A|}, (p_B^0, 0)^{|X_B|}, \epsilon), 0)\) and a stack consisting of \(\theta \bot\) in \(O\), we reach a configuration \((i) ((p_A, \nu_A'), (l_B, \nu_2, m), i)\) and a stack consisting of \(\theta \bot\) in \(O\) if \(i \leq K\), or \((ii) ((p_A, \nu_A'), (l_B, \nu_2, m), h)\) and a stack consisting of \(1^j \theta \bot\), \(j > 0\) in \(O\) if \(i > K\). Then \(h + j = i\). Moreover, it is possible to reach \((l_A, \nu_1)\) from \((p_A, \nu_A')\) in \(A\) after elapse of \(i\) units of time. The converse is also true. Finally, we reach a configuration \(((l_1, \nu_1), (l_2, \nu_2, \epsilon), 0)\) in \(N\) iff we reach the configuration \(((l_1, \nu_1), (l_2, \nu_2, \epsilon), 0)\) with a stack consisting of \(\theta \bot\) in \(O\).

6 Regaining Decidability via 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 at most one fixed channel, but possibly writing to many channels that it can write to, except from the one it reads from. The remaining automata can only do internal transitions. A context switch happens when we have two consecutive transitions \(C_{i-1} \rightarrow C_i\) and \(C_i \rightarrow C_{i+1}\) such that (a) or (b) is true. See Figure 4

(a) \(C_{i+1}\) is a configuration obtained by a channel operation in some automaton \(A_k\), and \(C_i\) is the configuration obtained by a channel operation in an automaton \(A_i \neq A_k\), or \(C_i\) is obtained by internal transitions in all automata, but there is a configuration \(C_g, g \leq i - 1\), obtained by a channel operation in an automaton \(A_i \neq A_k\), and there are no channel operations in configurations \(C_{g+1}, \ldots, C_i\).

(b) \(C_{i+1}\) is the configuration obtained when some automaton \(A_k\) reads from a channel \(c\), and \(C_i\) is obtained when \(A_k\) reads from a channel \(c' \neq c\), or 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 \(A_k\) writes to its channels in \(C_{g+1}, \ldots, C_i\).
If a CTA $\mathcal{N}$ is bounded context, then the number of context switches in any run of $\mathcal{N}$ is bounded above by some $B \in \mathbb{N}$.

**Theorem 4.** 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 switches $\leq B$, and let $[K] = \{0, 1, \ldots, K, \infty\}$. For $1 \leq i \leq n$, let $A_i = (L_i, L_i^0, A_i, E_i, F_i)$ 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 for a formal definition. 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 $(p_i, \nu_i)$, keeping all pairs $(p_j, \nu_j)$ 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 $l_i^0 \in L_i^0$, $\nu_i = 0^{\mid X_i\mid}$; 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 $\Delta_{in}$ of $\mathcal{M}$ correspond to any internal transition in any of the $A_i$s and change some $(p, \nu)$ to $(q, \nu')$ where $\nu'$ is obtained by resetting some clocks from $\nu$. These take place irrespective of context switch.

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

When $A_j$ is ready to read from channel $c_{i,j}$ (say), the contents of stack $W_{i,j}$ are shifted to stack $R_{i,j}$ if the stack $R_{i,j}$ is empty. Assuming $R_{i,j}$ is empty, we transfer contents of $W_{i,j}$ to $R_{i,j}$. The stack to be popped is remembered in the finite control of $\mathcal{M}$: the pair $(p, \nu), p \in L_j$ is replaced with $((p^{|W_{i,j}|}, \nu)$. As long as we keep reading symbols $t \in [K]$ from $W_{i,j}$, we remember it in the finite control of $\mathcal{M}$ by adding a tag $t$ to locations $(p^{|W_{i,j}|}, \nu)$ ($p \in L_j$) making it $((p^{|W_{i,j}|}, t, \nu)$. When a message $m$ is seen on top of $W_{i,j}$, with $((p^{|W_{i,j}|}), t, \nu)$ in the finite control of $\mathcal{M}$, we push $(m, t)$ to stack $R_{i,j}$, since $t$ is the indeed the time that elapsed after $m$ was written to channel $c_{i,j}$. When we obtain $t' \in [K]$ as the top of stack $W_{i,j}$,
with \((p_{W_{1,j}}, \nu)\) in the finite control, we add \(t'\) to the finite control obtaining \((p_{W_{1,j}} + t', \nu)\). The next message \(m'\) has age \(t + t'\) and so on, and stack \(R_{t,j}\) is populated. When \(W_{i,j}\) becomes empty, the finite control is updated to \((p_{R_{t,j}}, \nu)\) and \(A_j\) starts reading from \(R_{t,j}\). If \(R_{t,j}\) is already non-empty when \(A_j\) starts reading, it is read off first, and when it becomes empty, we transfer \(W_{i,j}\) to \(R_{t,j}\). 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.

\(\mathcal{M}\) is bounded phase: 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 at most two phase changes (some \(R_{t,j}\) to \(W_{i,j}\) and \(W_{i,j}\) to \(R_{t,j}\)) 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 can be seen in Appendix F. An example can be seen in Appendix F.3.

7 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 a good characterisation for the decidability border of the reachability problem in terms of number of processes and the underlying topology. Our undecidability is obtained for systems with depth 2, since there is a path from \(A_1\) to \(A_2\), and from \(A_2\) to \(A_3\), while our decidability result is obtained for systems with depth 1 containing only two processes. Thus, the general decidability problem for systems with depth 1 is still an open question, in particular for topologies like the star and the broom topology. The star topology is one where a central process writes to many processes, while there is no communication between these processes, and the broom topology is one where many processes write to a central process, with no other communication between processes. We conjecture that CTA reachability will stay decidable for systems with depth 1.

References

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

\[\text{4} \] the graph where each node is associated to a process and a directed edge between two nodes exists if there is a channel between their associated processes

\[\text{5} \] depth is defined as the longest simple path in the underlying topology
2. Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jonathan Cederberg. Timed lossy channel systems. In Deepak D’Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, 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. B. Bérard, F. Cassez, S. Haddad, O. Roux, and D. Lime. Comparison of different semantics for time Petri nets. In *ATVA 2005*, 2005.

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

7. 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.

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

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

10. 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.

11. Lorenzo Clemente, Frédéric Herbreteau, Amélie Stainer, and Grégoire Sutre. Reachability of communicating timed processes. In Frank Pfenning, editor, *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7794 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2013.

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

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

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

15. Pavel Krcáí 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.

16. 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*
17. M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall International, 1967.

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

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

A Illustration of Acyclic CTA and Bounded-context CTA

The leftmost part of Figure 4 illustrates a CTA which is not acyclic, since the underlying undirected graph has a cycle. The second CTA is an acyclic CTA. The right half of the Figure illustrates an acyclic CTA which is not bounded context: there is a run where \( A_1 \) writes an \( a \) every unit interval, 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.

Fig. 4. Left: A non-polyforest and a polyforest topology. Right: An unbounded context CTA.

B Proof of Theorem

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 describes the timed automaton \( A_2 \) of the CTA \( N \). \( A_1 \) 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 \( A_1 \). \( A_2 \) has widgets only corresponding to read transitions in \( A \). The automaton \( A_2 \) is a star-shaped widget joined at a location \( i \); each widget consists of 3 locations. We have a widget in \( A_2 \) for each read of message \( m_i \).

1. Consider a transition \((p, \text{nop}, q)\) in \( A \). Correspondingly, we have in \( A_1 \), a transition from \( p \) to \( q \) that checks if \( x_1 \) is 1 and resets it.
2. Consider a transition \((p, c_{A,A}^A m, q)\) 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 \( c_{1,2} \).
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 \(i\) to its 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 \(q_{m}^\prime\) with no time elapse. From \(q_{m}^\prime\), \(A_1\) 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.

Proof of Lemma 1: Direction from $A$ to $\mathcal{N}$

The proof is by construction. It is clear that corresponding to an initial configuration $(s_0, \epsilon)$ of $A$, we are in an initial configuration $(s_0, i, \epsilon)$ 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 $x_m$ 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 $\mathcal{N}$.

C Proof of Theorem 2

C.1 Counter Machines

A two-counter machine $C$ 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 $\ell_n$ 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 $\ell_k$,
2. (decrement $c$) $\ell_i : c := c - 1$; goto $\ell_k$,
3. (zero-check $c$) $\ell_i :$ if $(c > 0)$ then goto $\ell_k$ else goto $\ell_m$,
4. (Halt) $\ell_n$ : 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 $(k_0, k_1, \ldots)$ 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 $\ell_n$. 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 $\ell_n$. It is well known ([17]) that the halting problem for two-counter machines is undecidable.

We reproduce the widgets here for convenience.

1. Consider an increment instruction $\ell_i : \text{inc } c \text{ goto } \ell_j$. The widgets $\mathcal{W}_{A_i}^m$ for $m = 1, 2, 3$ are described in Figure 7. The one on the left is while incrementing $c_1$, while the one on the right is obtained while incrementing $c_2$.

![Fig. 7. Widgets corresponding to an increment $c_1, c_2$ instruction in each process](image)

2. The case of a decrement instruction is similar, and is obtained by swapping the speeds of the two automata in reaching $\ell_j$ from $\ell_i$. Consider a decrement instruction $\ell_i : \text{dec } c \text{ goto } \ell_j$. The widgets $\mathcal{W}_{d_i}^m$ for $m = 1, 2, 3$ are described in Figure 8. The one on the left is while decrementing $c_1$, while the one on the right is obtained while decrementing $c_2$.

![Fig. 8. Widgets corresponding to a decrement $c_1, c_2$ instruction in each process](image)
3. We finally consider a zero check instruction of the form $\ell_i : \text{if } c_1 = 0, \text{ then goto } \ell_j, \text{ else goto } \ell_k$. The widgets $W_{i}^{A_m}$ for $m = 1, 2, 3$ are described in Figure 9. The one on the left is a zero check of $c_1$, while the one on the right is a zero check of $c_2$.

Fig. 9. Widgets corresponding to checking $c_1, c_2$ is 0

### C.2 Proof of Lemma 2

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 $(W_{0}^{A_1}, \ell_0), (W_{0}^{A_2}, \ell_0), (W_{0}^{A_3}, \ell_0)$. The value of clocks $g_{A_1}, g_{A_2}, g_{A_3}$ are all 0.

1. Handling increment instructions. We start with $\ell_0$. Assume $\ell_0$ is an increment $c_1$ instruction. $A_1$ completes the widget $W_{0}^{A_1}$ in one time unit, while $A_2$ takes two units of time to complete $W_{0}^{A_2}$. It can be seen that $A_1$ reaches $(W_{1}^{A_1}, \ell_1)$ when $g_{A_1} = 1$, while $A_2$ reaches $(W_{1}^{A_2}, \ell_1)$ 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 $(W_{1}^{A_3}, \ell_1)$ when $g_{A_3} = 2, g_{A_3} - g_{A_2} = 0$, the value of $c_2$ after one step. In general, for each $\ell_i : \text{inc } c_1 \text{ goto } \ell_j$ instruction, the widget $W_{i}^{A_1}$ progresses by one time unit, incrementing $g_{A_1}$ by 1, while the widget $W_{i}^{A_2}$ progresses by two time units. This ensures the difference between $g_{A_2}, g_{A_1}$ at $\ell_j$ is one more than the difference at $\ell_i$. Likewise, since widgets $W_{i}^{A_2}, W_{i}^{A_3}$ 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.
\[ \ell_1 : \text{inc} \ c_2 \ \text{goto} \ \ell_j. \] The widgets \( W_{i}^{A_1}, W_{i}^{A_2} \) progress by one unit, preserving the value of \( c_1 \), and \( W_{i}^{A_3} \) progresses by two time units, incrementing \( g_{A_3} - g_{A_2} \) by one.

2. **Handling decrement instructions.** Assume \( \ell_i : \text{dec} \ c_1 \ \text{goto} \ \ell_j \) is a decrement \( c_1 \) instruction. \( A_1 \) completes the widget \( W_{i}^{A_1} \) in two time units, while \( A_2 \) takes one unit of time to complete \( W_{i}^{A_2} \). This ensures the difference between \( g_{A_2}, g_{A_1} \) at \( \ell_j \) is one less than the difference at \( \ell_i \). Likewise, since widgets \( W_{i}^{A_3}, W_{i}^{A_3} \) progress by one time unit, the difference between \( g_{A_3} \) and \( g_{A_2} \) remains constant, preserving the value of counter \( c_2 \). The argument is same for a decrement \( c_2 \) instruction \( \ell_i : \text{dec} \ c_2 \ \text{goto} \ \ell_j \). The widgets \( W_{i}^{A_1}, W_{i}^{A_2} \) progress by two units, preserving the value of \( c_1 \), and \( W_{i}^{A_3} \) 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 \( \ell_i \) to \( \ell_j \) after incrementing \( c_1 \), it writes the tuple \((\ell_i, c_1^+, \ell_j)\) 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 \( \ldots (\ell_i, c_1^+, \ell_j)(\ell_j, c_2^+, \ell_k) \ldots \) 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
\[ \ell_i : \text{if} \ c_1 = 0, \text{then goto} \ \ell_j, \text{else goto} \ \ell_k. \] By the above two cases, the values of counters \( c_1, c_2 \) are correctly encoded when \( A_1, A_2, A_3 \) reach \( \ell_i \) in widget \( W_{i}^{A_m}, m \in \{1, 2, 3\} \).

- 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 \((W_{i}^{A_1}, \ell_i)\) and \((W_{i}^{A_2}, \ell_i)\) and \( g_{A_2} = g_{A_1} \). No time is elapsed in widgets \( W_{i}^{A_1}, W_{i}^{A_2} \). The channel \( c_1,2 \) is empty, and \( A_1 \) writes in a message \text{zero1} 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 \( W_{i}^{A_1} \), 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 \( W_{i}^{A_2}, \ell_i \) to one of the successor locations. If control switches to the upper successor, indeed we get a successful move since the age of \text{zero1} is 0. In this case, \( \alpha \) is read off \( c_1,2 \) and \( \alpha \) is written to \( c_2,3 \). This is to help process \( A_3 \) decide the next instruction \( \ell_j \) correctly. Note that a wrong guess made in \( W_{i}^{A_1} \) 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 \( W_{i}^{A_2} \) since the lower part of \( W_{i}^{A_2} \) will be disabled.

- Assume \( c_1 > 0 \). In this case, we know that \( g_{A_2} - g_{A_1} > 0 \) when control respectively reaches \((W_{i}^{A_1}, \ell_i)\) and \((W_{i}^{A_2}, \ell_i)\). Hence, when \( A_1 \) reaches \((W_{i}^{A_1}, \ell_i)\), \( A_2 \) will be in some widget \( W_{i}^{A_2} \), and \( \ell_d \) is an instruction earlier than \( (\ell_d \text{ comes before } \ell_i) \) \( \ell_i \). Since no time elapse is possible in \((W_{i}^{A_1}, \ell_i)\),
A_2 waits wherever it is, while A_1 completes the widget W_i^{A_1}. Since non-zero time elapse is necessary for A_2 to reach widget 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 W_i^{A_1} is crucial here: A_1 must choose the lower half of the widget and write \beta. This will ensure that A_2 also writes \beta in c_{2,3}, and ensures that all three automata A_1, A_2, A_3 choose the instruction \ell_k.

Note that the value of c_2 is immaterial in the above. If c_2 was zero, then all three automata will be in \ell_i in the respective widget W_i^{A_m}. If c_2 > 0, then A_3 will “catch up” and reach widget 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_1^1, \ell_j), (\ell_i, c_2^1, \ell_j), (\ell_i, c_1^2, \ell_j), (\ell_i, c_2^2, \ell_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.

- First note that if c_1 = 0, then A_1, A_2 are both at \ell_i. If in addition, c_2 = 0, then A_3 is also at \ell_i. Then A_1 guesses if c_2 is zero or not by writing \alpha or \beta 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 \alpha, and if A_2 also made the same guess (A_2 must agree with A_1; otherwise, the computation stops) and reads the \alpha on c_{1,2} and wrote an \alpha on c_{2,3}, then indeed A_3 will proceed smoothly.

- If c_1 > 0, but c_2 = 0, then A_1 will have moved ahead from the widget W_i^{A_1} when A_2, A_3 reach (W_i^{A_2}, \ell_i). (W_i^{A_3}, \ell_i) together. The guesses of A_1 are already made, and one of \alpha, \beta will have been written in c_{1,2}, by the time A_2, A_3 reach W_i^{A_2}, W_i^{A_3}. However, the rest of the computation is smooth only if A_1 wrote \alpha, since A_3 will read zero_2 when its age is 0.

- If c_1 = 0, but c_2 > 0, then A_1, A_2 are together at (W_i^{A_1}, \ell_i), (W_i^{A_2}, \ell_i) respectively, while A_3 is in a widget W_i^{A_3} where \ell_g is an instruction earlier than \ell_i. In this case, a correct computation requires A_1 to take the lower branch of W_i^{A_1} and write a \beta, since the age of zero_2 will be > 0 when A_3 reads it, and then c_{2,3} must have a \beta.

- If c_1 > 0 and c_2 > 0, then A_1 is at the widget W_i^{A_1}, while A_2 is in some widget W_{d_2} for some instruction \ell_d before \ell_i, and A_3 is in some widget W_{d_2} for some instruction \ell_f before \ell_d. In this case again, A_1 must choose the lower branch of W_i^{A_1}, and write a \beta. This \beta will be read by A_2 when it catches up and reaches W_i^{A_2}, and the \beta 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 \beta 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 \alpha, \beta ensures that all three processes are in agreement in their choices of instructions while simulating the two counter machine.
C.3 Proof of Lemma 3

By Lemma 2, we know that in any successful computation of \(N\), 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 \(\ell_{\text{halt}}\), 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 \(N\) also loops through the widgets corresponding to the sequence of instructions visited by the two counter machine.

D Proof of Theorem 3

To prove the correctness of construction of \(O\), we prove lemmas 4 and 5.

D.1 Proof of Lemma 4

Proof. The initial configuration in \(O\) is \(((\ell_0^A, 0^{|X_A|}), (\ell_0^B, 0^{|X_B|}), \epsilon), 0)\). 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 \(O\), 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 no time elapses 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 5

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

We start in \(O\) with \(((\ell_A^0, 0^{|X_A|}), (\ell_B^0, 0^{|X_B|}), \epsilon), 0)\) and a stack consisting of \(\theta\). Till \(A\) writes a message onto the channel, the simulation of \(O\) consists of time elapse and internal transitions of \(A, B\). By construction of \(O\), \(B\) is always ahead of \(A\), or at the same global time. If \(A\) writes its first message say \(a\) when no time elapse
We give an example illustrating Theorem 3. Figure 10 gives a CTA consisting of automata $A, B$, and also the respective region automata $Reg(A), Reg(B)$. Consider elapsed 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 $O$, and we obtain some configuration $((p_A, ν_A'), (l_B, ν_2, a), i)$ and a stack with $θ ⊥$ if $i ≤ K$. Let us assume $i ≤ K$. Once $B$ enables this transition, $a$ is read, and we obtain a configuration $((p_A, ν_A'), (l'_B, ν'_2, ε), i)$. $(p_A, ν_A')$ 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 at least $i$. 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 $j$. It reads it off. The time difference between $B$ and $A$ is now $j$. $A$ can now elapse $h$ 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 $N$, 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 $(l_A, ν_A')$ after $i$ elapse 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 $((l_1, ν_1), (l_2, ν_2, ε), 0)$ in $O$ with a stack consisting of $θ ⊥$ iff in $N$ we reach the configuration $((l_1, ν_1), (l_2, ν_2), ε)$. The same sequence of transitions are taken in $Reg(A), Reg(B)$ in both $O$ and $N$, with the only difference being that in $N$, the two automata move in-sync, while in $O$, $B$ is made to run ahead of $A$ whenever $A$ writes a message. In $O$, we always keep at most 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 $N$ and $O$ is thus that in $O$, $A, B$ are “de-coupled”, while in $N$ they are in-sync.

D.3 Example Illustrating Theorem 3

We give an example illustrating Theorem 3. Figure 10 gives a CTA consisting of automata $A, B$, and also the respective region automata $Reg(A), Reg(B)$. Consider...
the run $\mathcal{N}_0=((s_1,0),(q_1,0),\epsilon) \rightarrow \mathcal{N}_1=((s_2,0),(q_1,0),(a,0)) \rightarrow \mathcal{N}_2=((s_3,1),(q_1,1),(c,0)(a,1)) \rightarrow \mathcal{N}_3=((s_2,0),(q_3,\infty),(b,0)(a,0)(c,2)) \rightarrow \mathcal{N}_4=((s_2,0),(q_4,\infty),(b,0)(a,0)(c,2)) \rightarrow \mathcal{N}_5=((s_2,1),(q_2,\infty),\epsilon)$

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

---

**Fig. 10.** 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.
1. It is easy to see that \( O_0, O_1 \) exactly correspond to \( N_0, N_1 \). \( a \) is read in \( O_1 \), obtaining \( ((s_2, 0), ((q_1, 0), a), 0) \). Neither \( A \) nor \( B \) have elapsed any time, and the stack has \( \emptyset \).

2. If we look at \( N_2 \), 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 \( O \), until the first message is read, we do not write the second message. Thus, \( O_2 \) will be a configuration obtained when \((a, 1)\) is read. Recall that \( a \) was written in \( O_1 \). 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 as \( \emptyset \), 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 \( q_1' \).

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 \( N \). This gives the configuration \( O_2 \) 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 \( \emptyset \). That is, \((s_3, 1), ((q_2, 1), c), 0)\) with the stack holding \( \emptyset \).

3. \( N_3 \) 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 \( N_2 \). In the simulation of \( O \), the message \( c \) will be written first, then 2 time units elapsed, and \( c \) read. We are currently at \((s_3, 1), ((q_2, 1), c), 0)\). \( c \) 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 \((q_3, \infty)\), 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 \( \emptyset \). One more unit time elapse makes the age of \( c \) 2, and a 1 is written on the stack. This makes the configuration \((s_3, 1), ((q_3, \infty), c), 1)\) along with the stack \( 1 \emptyset \). To read the \( c \) from \((q_3, \infty)\), 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 \((q_1', \infty)\). The finite control of \( B \) moves to \((q_2, \infty)\) reading the \( c \) obtaining \((s_3, 1), ((q_2, \infty), c), 1)\) with stack \( 1 \emptyset \). 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 \((s_2, 1), ((q_2, \infty), c), 1)\) with stack \( \emptyset \). The finite control of \( B \) moves from \((q_2, \infty)\) to \((q_1, 0)\), obtaining \((s_2, 1), ((q_1, 0), c), 1)\) with stack \( \emptyset \). 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 \( O_3 \), where we have \((s_2, 1), ((q_1, 0), c), 0)\) with top of stack \( \emptyset \).

4. \( N_4 \) is the configuration where \( c \) has been read, and there are messages \( b, a \) in the channel with age 0. In \( O_3 \) 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 $\theta \bot$. A sequence of transitions as seen in the case of $O_2$ (where $\theta$ 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 $O_4$ with $((s_2, 0), ((q_2, 1), \epsilon), 1)$ with the stack $\theta \bot$.

5. $N_5$ is the configuration where $b$ is read, and the channel is empty, with $A$ at $(s_2, 1)$, $B$ at $(q_2, \infty)$ and an empty channel. In $O$, we have to write $b$ from $O_4$ and read it when its age is $> 1$. This is done in a manner similar to what we did in $O_3$ where the topmost 1 in the stack is read and remembered in the finite control. It can be seen that we obtain $O_5$ with $((s_2, 1), ((q_2, \infty), \epsilon), 0)$ and stack $\theta \bot$.

The main difference between configurations in $N$ and $O$ is thus the fact that in $N$, 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 $O$, 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 $M = (S, S_0, St, \Gamma, \mathcal{X}, \Delta)$ 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, $\Gamma$ 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 $M$ is a tuple $(s, \nu, \{\sigma_{st}\}_{st \in St})$ where $s \in S$ is the current control location, $\nu$ 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 $(s_0, \nu, [\mathcal{X}], \{\sigma_{st}\}_{st \in St})$ with $\sigma_{st} = \epsilon$ for all $st \in St$. The semantics of $M$ is given by defining the transition relation induced by $\Delta$ on the set of configurations of $M$. A transition relation is written as $(s, \nu, \{\sigma_{st}\}_{st \in St}) \rightarrow (s', \nu', \{\sigma'_{st}\}_{st \in St})$ 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 $\nu \models g$, $\nu' = \nu[Y := 0](x)$ 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, $\nu \models g$, $\nu' = \nu[Y := 0](x)$ 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, \( \nu \models g \), \( \nu' = \nu[Y := 0](x) \) and control moves to \( s' \).

A run of \( \mathcal{M} \) is a sequence of transitions \( c_0 \rightarrow c_1 \rightarrow c_2 \cdots \rightarrow c_n \) 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 \( \alpha_i \), 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 \( k \). We prove Lemma 6 reducing it to the bounded-phase reachability problem for untimed multipushdown systems. The proof follows using a standard region construction.

Lemma 6. The reachability problem is decidable for BMPS.

Proof. Let \( \mathcal{M} = (S, s_0, St, \Gamma, X, \Delta) \) be a BMPS. The first step is to convert \( \mathcal{M} \) to \( \text{Reg}(\mathcal{M}) \) by the standard region construction. The states of \( \text{Reg}(\mathcal{M}) \) have the form \( (l, \nu) \) where \( l \in S \) and \( \nu \in \mathbb{N}^{|X|} \). The internal transitions, push and pop transitions are now from locations \( (l, \nu) \) to \( (l', \nu') \). It is easy to see that \( \text{Reg}(\mathcal{M}) \) 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 \( (l, \nu) \) from \( (s_0, 0) \), preserving the stack contents. Using known results \([16]\) we know that the reachability in \( \text{Reg}(\mathcal{M}) \) is decidable. Hence, reachability in \( \mathcal{M} \) is also decidable.

F Proof of Theorem 4

Given a bounded context CTA \( A \), 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 \( A \) 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 \( \Sigma \) is the channel alphabet of \( A \). Let \( A_i = (L_i, L'_i, \text{Act}_i, X_i, E_i, F_i) \) for \( 0 \leq i \leq n \), \( K \) 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 \( A \).

We construct the MPS \( \mathcal{M} = (S, S_0, St, \Gamma, \Delta) \) where

1. \( S \) is a finite set of locations \( (L'_1 \times [K]^{|X_1|}) \times \ldots \times (L'_n \times [K]^{|X_n|}) \times (A_w \times p) \), where \( w \in \{1, \ldots, n\} \) represents the active automaton and \( 0 \leq p \leq B \) is a number that keeps track of context switches in the CTA.
2. \( L'_i = L_i \cup \{ l_i, p^n_i, t^n_i \mid l \in L_i, t \in [K], a \in \Sigma, p \in \{ W_j, R_j, i \mid 1 \leq j \leq n \} \} \).

3. The set of initial locations \( S_0 \) is 
\[
(\{ l_0 \} \times 0^{\lvert X_1 \rvert}) \times \cdots \times (\{ l_0 \} \times 0^{\lvert X_n \rvert}) \times \bigcup_{1 \leq p \leq n} (A_p \times 0),
\]
4. \( St \) is a finite set of stacks: each channel \( c_{i,j} \) of \( A \) 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 \( 0 \leq j \leq B \). We now explain below the transitions in the MPS \( M \). For each run in the CTA \( A \), we show that there is a run in the BMPS \( M \) preserving reachability; moreover, the content of each channel \( c_{i,j} \) is retrieved from stacks \( W_{i,j}, R_{i,j} \) in \( M \).

**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 \( M \), let us start from the location \( (\{ l_0 \}, 0^{\lvert X_1 \rvert}), \ldots, (\{ l_0 \}, 0^{\lvert X_n \rvert}), (A_{i_0}, 0) \) and all stacks empty. Internal transitions in any \( A_i \) are handled by updating the corresponding pair \( (l_i, \nu_i) \) in \( M \), \( l_i \in L_i \) by updating the control locations \( l_i \), and the tuple \( \nu_i \) taking care of resets. These transitions are all in \( \Delta_{int} \).

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 \( M \), we have the transition in \( \Delta_{push} \) which updates \( (p, \nu) \in L_{i_0} \times [K]^{\lvert X \rvert} \) to \( (q, \nu') \), where \( \nu' \) is obtained by resetting clocks \( Y \subset X_{i_0} \), checks guard \( g \) on \( \nu \), and pushes \( m \) to stack \( W_{i_0,j} \). All tuples \( (l, \nu_i) \in L_i \times [K]^{\lvert X_i \rvert}, i \neq i_0 \) are left unchanged. After the first write, any time elapse \( t \in [K] \) is taken care of by transitions in \( \Delta_{push} \) which not only update the clock values, but also push \( t \) to all stacks. 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 \( M \). Subsequent time elapses are pushed to all stacks. To summarize, simulation of context 0 in \( M \) results in stacks \( W_{i_0,j} \) consisting of elements of the form \( \Sigma \cup [K] \) (messages from \( \Sigma \) 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 read from some fixed channel \( c_{k,i_h} \). It can write to several channels \( c_{i_0,j} \), 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 from some channel \( c_{k,i_h} \) and \( A_{i_{h-1}} = A_{i_h} \), but \( A_{i_{h-1}} \) was reading off a channel.

---

\( ^6 \) Note that during a time elapse \( t \), we do two things: (1) update all \( \nu_i \) to \( \nu_i + t \) in all the \( n \) pairs, and (2) push \( t \) onto all stacks. To ensure that all the \( \nu_i \)s are updated to \( \nu_i + t \), we can keep an additional bit in the control location of \( M \) which starts at 1, updates \( \nu_i \), and keeps incrementing the bit till \( n \), when \( \nu_i \) is updated to \( \nu_i + t \), 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.
was active in context 0, and no other automaton has written any message so far. Time elapses made during this context are pushed to all stacks. Assume $A_{i_h}$ is non-empty. In this case, we first read off from $W_{i_h}$ 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.

1. Let $(p, \nu)$ 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 $(p, g, c_{k,i_h} ; m \in I, Y, q)$. We first check if $R_{k,i_h}$ is empty: for this, we first change the control location $(p, \nu)$ 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 $\infty$ 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 $t + t'$. Using the information $t + t'$ from the finite control $((p^{R_{k,i_h}}), t, \nu)$, 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 $((p^{R_{k,i_h}}), t + t', \nu)$, $t'' = \nu[Y := 0]$. 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 $\bot$ 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 $((q^{R_{k,i_h}}), t, \nu')$ or $(q^{R_{k,i_h}}, t', \nu')$ or $(q, \nu')$, $q \in L_{i_h}$).

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 $m \in \Sigma$ appears on top of $W_{k,i_h}$. If the finite control of $\mathcal{M}$ is $((q^{W_{k,i_h}}), t, \nu')$, 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 $\bot$ 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 $R_{k,i_h}$. 

$c_{k',i_{h-1}} \neq c_{k,i_h}$. 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 $c_{i_{h-1}}$ are handled by pushing messages to stack $W_{i_{h-1}}$ and updating the finite control of $\mathcal{M}$ pertaining to $A_{i_h}$. 
5. The context $h$ may finish before $R_{k,ih}$ is empty, in which case, we will continue reading from it when the next context of $A_{ih}$ appears again, assuming $A_{ih}$ still reads from channel $c_{k,ih}$. The other possibility is that $R_{k,ih}$ is emptied in this context.

6. If stack $R_{k,ih}$ is emptied while in context $h$, the finite control of $M$ is updated to $(q, \nu')$ from $(q^{R_{k,ih}}, \nu')$ or $((q^{R_{k,ih}})_1, \nu')$. If $W_{k,ih}$ is empty, then there are no more pops to be done while in this context, since $A_{ih}$ can only write to some of its channels now. If a context switch happens before $R_{k,ih}$ is emptied, then the finite control of $M$ pertaining to $A_{ih}$ is updated to $(q, \nu')$. The finite control $(s, \nu_s)$ of $M$ pertaining to $A_{ih+1} (s \in L_{ih+1})$ may either stay same if $A_{ih+1}$ is enabled to write from $s$, or will be updated to some $(s^{R_{k,ih+1}}, \nu_s)$ if $A_{ih+1}$ is enabled to read from some channel $s_{g,ih+1}$ in the $(h+1)$st context. In the case when $A_{ih+1} = A_{ih}$, then the context switch takes place since $A_{ih}$ is ready to read from another channel $c_{k',ih}$. In this case, we update $(q^{R_{k,ih}}, \nu')$ or $((q^{R_{k,ih}})_1, \nu')$ to $(q^{R_{k',ih}}, \nu')$.

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

### F.2 Correctness of Construction

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

**Lemma 7.** If $A$ is a bounded context CTA with atmost $B$ context switches, then the MPS $M$ 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 $A$.

1. In contexts $i \in \{1, 2, \ldots, B\}$, assume that the active automaton $A_i$ reads from some channel $c_{k,i}$. By construction of $M$, we have stacks $W_{k,i}$, $R_{k,i}$ corresponding to each channel $c_{k,i}$. When we start a new context $i$ of $A$, 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 $i$th context results in pushing $t$ to all stacks. So far, there has been no pop of any stack in $M$ while in context $i$ of $A$. Only when $A_i$ is ready to read from a channel say $c_{k,i}$, do we start popping a stack; first we check if $R_{k,i}$ is non-empty, and if so pop that. This counts as a phase change. If $R_{k,i}$ becomes empty, and we have more read operations of $c_{k,i}$ in context $i$ of $A$, then we pop stack $W_{k,i}$ and transfer contents to $R_{k,i}$. This counts as another phase change. Finally, when $R_{k,i}$ has been populated, we pop $R_{k,i}$ to facilitate reading from $c_{k,i}$. 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 $c_{k,i}$ are already in stack $R_{k,i}$: recall that $A_i$ cannot write to $c_{k,i}$ since she reads from it; if any other automaton writes to $c_{k,i}$, 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, $R_{k,i}$ was non-empty in the beginning of the $i$th context, and does not get emptied (in this case, it is just 1 change of phase), or if $R_{k,i}$ is empty in the beginning of the $i$th context, and we pop $W_{k,i}$ followed by $R_{k,i}$ (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 $\leq 3$ phase changes in $\mathcal{M}$, the maximal number of phase changes in $\mathcal{M}$ is $\leq 3B$.

**Lemma 8.** 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 $A_i$, 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 $c_{k, i, j}$ 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 $A$. The active automaton is $A_{ij}$. Let $A_{ij}$ read from channel $c_{k, i, j}$, in context $j$. At the start of context $j$, by construction of $M$, we have two possibilities for stacks $R_{k, i, j}$, and $W_{k, i, j}$:

(1) either stack $R_{k, i, j}$ contains only symbols from $[K]$ and $W_{k, i, j}$ contains symbols from $\Sigma \cup [K]$, or

(2) $R_{k, i, j}$ contains symbols from $(\Sigma \times [K]) \cup [K]$ and $W_{k, i, j}$ contains symbols from $\Sigma \cup [K]$. If (1), then either channel $c_{k, i, j}$ was never read so far in $A$ and the entire channel content is in $W_{k, i, j}$. The other possibility is that $c_{k, i, j}$ was read in an earlier context, and $A_{ij}$ read all the contents of $c_{k, i, j}$ at that time, and the subsequent writes to $c_{k, i, j}$ are stored in $W_{k, i, j}$.

In case of (2), channel $c_{k, i, j}$ was read in an earlier context, but the channel was not completely read that time; the remaining contents of $c_{k, i, j}$ from that context are in $R_{k, i, j}$, along with possible time elapses since then. All subsequent writes to $c_{k, i, j}$ after that context are stored in $W_{k, i, j}$.

In case of (1), in the $j$th context, the contents of $W_{k, i, j}$ are shifted to $R_{k, i, j}$. At the end of context $j$, if $R_{k, i, j}$ is non-empty, then the contents of $R_{k, i, j}$ top-down is the content of channel $c_{k, i, j}$ (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 $j$th context, we start reading off $R_{k, i, j}$. At the end of the $j$th context, if $R_{k, i, j}$ is over $(\Sigma \times [K]) \cup [K]$ and $W_{k, i, j}$ is over $\Sigma \cup [K]$, then the contents of channel $c_{k, i, j}$ is obtained by first popping $R_{k, i, j}$, 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 $R_{k, i, j}$. Once $R_{k, i, j}$ is empty, we pop $W_{k, i, j}$ in a similar manner. Let $w_1 \in (\Sigma \times [K])^*$ be the string so formed after popping $W_{k, i, j}$. The contents of channel $c_{k, i, j}$ at the end of context $j$ is then obtained as $w_1 w_2$.

It is easy to see that the finite control of $M$ is $((l_1, \mu_1), \ldots, (l_n, \mu_n), (A_t, j))$ iff in $A$ we reach $(l_i, \mu_i)$ 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 $M$. Thus, $M$ preserves reachability, both of control locations as well as channel contents. Finally, the number of phase changes in $M$ depends on the number of context switches in $A$.

F.3 Illustration of Theorem 3: CTA to MPS

We first show a sequence of context switches ($\leq 10$) on the CTA in Figure 11. 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.

![Diagram](image)

Fig. 11. A bounded context CTA.

1. To begin, \( A_2 \) writes several \( a \) 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 \( a \) as when \( y = 1 \).
\( c_{2,1} : (a,0)(a,1) \), \( c_{1,2} : (b,0)(e,0) \)
4. A switch to \( A_1 \) results in reading off the leading \( a \) from \( c_{2,1} \) and 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 \( a \) 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 \( a \) 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 \( a \) 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 \( b, e, a \) 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 \( c_{2,1} : e, c_{1,2} : (e,1) \).

No more context switches are possible.

Consider the following run of the CTA given in Figure 11.
\[
\mathcal{N}_6 = ((p_1,0),(q_1,0),e,\epsilon) \xrightarrow{} \mathcal{N}_1 = ((p_1,0),(q_1,0),e,(a,0)(a,0)) \xrightarrow{} \mathcal{N}_2 = ((p_2,1),(q_2,1),e,(a,1)(a,1)) \xrightarrow{} \mathcal{N}_3 = ((p_1,1),(q_2,2),(b,1)(e,1),(a,2)(a,2)) \xrightarrow{} \mathcal{N}_4 = ((p_1,1),(q_2,2),(b,1)(e,1),(a,2)) \xrightarrow{} \mathcal{N}_5 = ((p_1,1),(q_1,2),e,(a,0)(a,2)) \xrightarrow{} \mathcal{N}_6 = ((p_2,2),(q_3,3),e,(g,0)(a,3)).
\]

In Tables 1, 2, and 3 we show the sequence of locations along with the stack contents of the MPS that correspond to each \( \mathcal{N}_i \). Tables 1, 2, and 3 give a run of the CTA and the corresponding run in the MPS.
### BMPS locations reached

| Location | BMPS stacks |
|----------|-------------|
| $N_0$    | ($p_1, 0), (q_1, 0), (A_2, 0)$ |
| $N_1$    | ($p_1, 0)(q_1, 0), (A_2, 0)$ |
| $N_2$    | ($p_2, 1)(q_2, 1), (A_2, 0)$ |
| $N_3$    | ($p_1^{R_{21}}, 1)(q_2, 2), (A_1, 1)$ |
|          | 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. |
| $N_4$    | (($p_1^{W_{21}}), 1)(q_2, 2), (A_1, 1)$ |
|          | 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 $p_1^{W_{21}}$. |
|          | ($p_1^{W_{21}}, 1)(q_2, 2), (A_1, 1)$ |
|          | ($p_1^{W_{21}}), 2)(q_2, 2), (A_1, 1)$ |
|          | This becomes ($p_1^{W_{21}}), 2)(q_2, 2), (A_1, 1)$ when the $a$ on top of $W_{21}$ is read. $(a, 2)$ is pushed to $R_{21}$ and the control comes back to (($p_1^{W_{21}}), 1)(q_2, 2), (A_1, 1)$ |
|          | 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 $p_1^{R_{21}}$. |
|          | ($p_1^{R_{21}}, 1)(q_2, 2), (A_1, 1)$ |
|          | ($p_1^{R_{21}}), 1)(q_2, 2), (A_1, 1)$ |
|          | ($p_1, 1)(q_2, 2), (A_1, 1)$ |

Table 1.
Table 2.
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 $p_2$.

| CTA | BMPS locations |
|-----|----------------|
| $X_6$ | $(p_2, 2)(q_1, 3), (A_1, 3)$ |

| BMPS stacks |
|----------------|
| $W_{1.2}$ | $R_{1.2}$ | $g$ | $a$ | $R_{2.1}$ |

Table 3.