TCS 4354
pp: 1-23 (col.fig.: Nil)
PROD. TYPE: COM
ED: SG
PAGN: VD – SCAN: Padma

ELSEVIER
Theoretical Computer Science ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮
Theoretical Computer Science
www.elsevier.com/locate/tcs

Verification in loosely synchronous queue-connected discrete timed automata[☆]

Oscar H. Ibarra[a,*], Zhe Dang[b], Pierluigi San Pietro[c]

[a]*Department of Computer Science, University of California, 93106-5110 Santa Barbara, CA 93106, USA*
[b]*School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164, USA*
[c]*Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy*

Received 15 May 2001; received in revised form 6 December 2001; accepted 16 January 2002
Communicated by A. Salomaa

Abstract

We look at a model of a queue system that consists of the following components:
1. Two discrete timed automata W (the "writer") and R ("the reader").
2. One unrestricted queue that can be used to send messages from W to R. There is no bound on the length of the queue.

W and R do not share a global clock and operate in a loosely synchronous way. That is, the absolute value of the difference between the local time of W and the local time of R is always bounded by a positive constant. We show that the binary reachability for these systems is effectively computable, and this result is generalized to the case when there are two queues (one from W to R and the other from R to W) that operate in half-duplex. We then present some properties (e.g., safety, invariance, etc.) that can be verified for loosely synchronous queue-connected discrete timed automata and give an example of a system composed of a sensor and a controller that is verifiable using our results. © 2002 Published by Elsevier Science B.V.

Keywords: Discrete timed automata; Queue-connected; Reachability; Safety; Invariance

1. Introduction

Model checking techniques [29] have received much attention in recent years, due to the success of automatic techniques for verifying finite-state systems describing

[☆] Supported in part by NSF grants IRI-9700370 and IIS-0101134.
[*] Corresponding author. Tel.: +1-805-893-41; Fax: +1-805-893-85.
E-mail address: ibarra@cs.ucsb.edu (O.H. Ibarra).

protocols, hardware devices, and reactive systems [12]. The limited expressiveness of finite automata has recently sparked much research to define and study infinite-state models that can verify "interesting" properties such as reachability, safety, liveness, etc. The infinite-state models that have been investigated include timed automata [2], pushdown automata [8,18], various versions of counter machines [13,17], and process calculi [28,6].

A timed automaton is basically a finite automaton with finitely many unbounded clocks that can be tested and reset. Since their introduction and the development of appropriate model checking algorithms [4,3,21], timed automata have become a standard model for investigating verification problems of real-time systems (see [1,31] for surveys). However, the expressive power of timed automata has many limitations in modeling, since many real-time systems are simply not finite-state, even when time is ignored.

One of the ways to extend a timed automaton is to augment it with an unbounded storage device, e.g., a pushdown stack. It has been shown very recently that the binary reachability of a timed pushdown automaton is decidable [16] when clocks are discrete. This result immediately implies that a number of nonregion properties (e.g., a Presburger formula over clocks) can be verified. This is in contrast to the classical result [2] that region reachability of timed automata is decidable. However, *queues*, not stacks, are a good model for many interesting systems, such as protocols and schedulers.

Queues are usually regarded hopeless for verification, since it is well known that a finite-state automaton equipped with one unbounded queue can simulate a Turing Machine. However, there are restricted models with queues for which reachability is decidable. These models mostly focus on restricted versions of communicating finite state machines (connecting finite state machines with a number of FIFO queues) [7,9], such as a version when the queues contain only one type of message and form a single cycle [30], and a version when the queues are lossy [5]. In this paper, instead of considering finite state machines, we consider discrete timed automata (i.e., clocks are discrete). That is, we study a new queue system, called queue-connected timed automaton, by connecting two discrete timed automata (one "writer" W and one "reader" R) with a FIFO queue. The queue is unrestricted and it is used to send messages from W to R. This model is inspired by the recent work [23] that considers systems with two reversal-bounded counter machines (thus, each counter can be incremented or decremented by 1 and tested for zero, but the number of alternations between nondecreasing mode and nonincreasing mode is bounded by a fixed constant) connected by a FIFO queue. It was shown in [23] that (binary, forward, backward) reachability, safety, and invariance for these systems are solvable when the machines operate synchronously. In this paper, we present similar results for a more complex system of queue-connected discrete timed automata that do not share a global clock and operate in a loosely synchronous way. Our results remain valid when the discrete timed automata are augmented with reversal-bounded counters.

We treat the reader and the writer as running in a distributed environment. Even though our technique is still valid in dealing with the case when the reader and the writer share a global clock, we prefer to consider a harder and more reasonable case. That is, the reader and the writer do not share a global clock.

We also allow the reader and the writer to be loosely independent. This means that the local time of the reader and the local time of the writer always stay close: the variation between them is bounded by a constant. This assumption is reasonable. For instance, in a distributed network, time protocols can be used to control clock drifting, even though a global clock is not usually assumed. Notice that if the reader and the writer are completely independent instead of loosely independent (i.e., synchronizations are made only at queue operations), the technique presented in this paper can still be used (and more easily).

In our model, the queue is essentially one-way: the model can describe, for instance, a time-dependent communication protocol such that party A can only send messages (through a queue) to party B. In this case, A is the writer and B is the reader. Of course, many protocols involve two-way communications instead of only one-way communications. However, it can be easily shown that adding another queue to our model (from B to A) gives it the power of a Turing machine, even when A and B are finite-state machines (not timed automata). But one-way protocols do exist. For instance, there is a number of well-known producer/consumer models such as the modeling of TCP using an unbounded buffer [10], where the producer (the writer) and the consumer (the reader) operate on an unbounded buffer. There is no pre-assumption on the relative speed of the reader and the writer—they can be synchronous, asynchronous, or loosely synchronous. The results presented in this paper can be easily used to automatically verify safety properties expressed as Presburger formulas over the clock readings in both the consumer and the producer, as well as over the number of symbols in the buffer.

Our model can also handle two-way communications by adding a second queue under some restrictions. One of the restrictions is to make the two queues *half-duplex* [11]. That is, at any moment, at least one of the queues is empty. It is known that two finite automata connected by two half-duplex queues have a recognizable reachable set [11]. In this paper, we show that our verifications results still hold for loosely synchronous QTAs with two half-duplex queues. This result opens the door for verifying a restricted class of two-way timed communication protocols.

Another point that needs mentioning is that, since we are characterizing binary reachability and doing verification over a class of Presburger properties, the traditional region technique [11] is not applicable; i.e., discrete clocks here cannot be simply treated as bounded counters [14]). Therefore, constructing the region graph of the model of interest is not enough to deduce the binary reachability [14,16].

The paper has seven sections, in addition to this section. Section 2 briefly recalls the definition of a discrete timed automaton and its extensions. Section 3 discusses some results in [22,23] that are used in the paper and cites some recent results in [16,26] concerning the binary reachability of discrete timed automata (including those with a pushdown stack and/or reversal-bounded counters). Section 4 formally defines queue-connected timed automata and shows that binary reachability is effectively computable. Section 5 extends the verification results to loosely synchronous QTAs with two half-duplex queues. Section 6 presents some properties that can be verified for queue-connected timed automata, including safety and invariance. Section 7 gives an example of a system composed of a sensor and a controller. Section 8 is a brief conclusion.

2. Discrete timed automata and extensions

A timed automaton [2] is a finite-state machine augmented with a number of real-valued clocks. All the clocks progress synchronously with rate 1, except that a clock can be reset to 0 at some transition. Here, we only consider integer-valued clocks. A *clock constraint* is a Boolean combination of *atomic clock constraints* in the following form: $x\#c, x-y\#c$ where # denotes $\leqslant$, $\geqslant$, $<$, $>$, or $=$, c is an integer, x, y are integer-valued clocks. Let $\mathscr{L}_X$ be the set of all clock constraints on clocks X. Let $\mathbb{Z}$ be the set of integers and $\mathbb{N}$ be the set of nonnegative integers. Formally, a *discrete timed automaton* (TA) A is a tuple $\langle S, X, E\rangle$ where S is a finite set of (*control*) *states*, X is a finite set of *clocks* with values in $\mathbb{N}$, and $E \subseteq S \times 2^X \times \mathscr{L}_X \times S$ is a finite set of *edges* or *transitions*. Each edge $\langle q, \lambda, l, q'\rangle$ in E denotes a transition from state q to state q' with an *enabling condition* $l \in \mathscr{L}_X$ and a set of clock resets $\lambda \subseteq X$. Note that λ may be empty. A configuration is a tuple $(q, \mathbf{X})$ where q is a state and $\mathbf{X}$ is an array of clock values (we use $\mathbf{X}_i$ to denote the clock value of x_i). The meaning of a one-step transition along an edge $\langle q, \lambda, l, q'\rangle$ that sends a configuration $(q, \mathbf{X})$ to $(q', \mathbf{X}')$ is as follows:

- The enabling condition l is satisfied on the configuration $(q, \mathbf{X})$; i.e., $l(\mathbf{X})$ holds.
- The state q is set to a new state q'.
- Each clock changes according to the following: If there are no clock resets on the edge, i.e., $\lambda = \emptyset$, then each clock $x_i \in X$ progresses by one time unit; i.e., $\mathbf{X}'_i = \mathbf{X}_i + 1$. If $\lambda \neq \emptyset$, then each clock $x_i \in \lambda$ is reset to 0 ($\mathbf{X}'_i = 0$), while each $x_j \notin \lambda$ remains unchanged ($\mathbf{X}'_j = \mathbf{X}_j$). Thus, clock resets do not take time.

Timed automata have been extended with various unbounded memory structures, such as stacks and reversal-bounded counters. When a TA is augmented with an unrestricted pushdown stack, the one-step transition will be of the form $\langle q, \lambda, l, q', Z, w\rangle$, where Z denotes the top of the stack, and w is the string that replaces Z ($w = \varepsilon$ means pop/erase). We call this model a (discrete) pushdown timed automaton (PTA).

A PTA can further be generalized by equipping it with k reversal-bounded counters (i.e., each counter can be incremented or decremented by one and tested for zero, but the number of times it changes mode from nondecreasing to nonincreasing and vice-versa is bounded by a constant, independent of the computation). The one-step transition is of the form $\langle q, \lambda, l, q', Z, w, b_1, \ldots, b_k, d_1, \ldots, d_k\rangle$, where b_i is the status of counter i (zero or nonzero), and d_i is 1, 0, or -1 (representing increasing by 1, staying unchanged, or decreasing by 1, respectively). This model is called a reversal-bounded multicounter pushdown timed automaton (CPTA). A CPTA without a pushdown will be called a CTA. Note that TAs, PTAs, CPTAs, and CTAs have no input tapes.

A configuration of a CPTA A is a 4-tuple $\alpha = (q, \mathbf{X}, \mathbf{Y}, w)$, where q is the state (from a finite set), $\mathbf{X}$ is the array of clock values, $\mathbf{Y}$ is the array of counter values, and w is the content of the pushdown stack. Note that α can be represented as a string where the components of the tuple are separated by markers and the state clock values, and counter values are written in unary. Note also that $\mathbf{Y}$ (resp. w) is not present in the configuration if there are no reversal-bounded counters (resp. no stack). The *binary reachability* of A is the set $R(A) = \{(\alpha, \beta) :$ configuration α can reach β in 0 or moreone $-$ step transitions}. Section 3 reports on the main decidability results for PTAs, CPTAs and CTAs.

3. Pushdown acceptors with reversal-bounded counters and reachability

A pushdown acceptor with reversal-bounded counters (PCA), first studied in [22], is a nondeterministic one-way (input) pushdown automaton augmented with finitely many reversal-bounded counters. Without loss of generality, we assume that the counters can only store nonnegative integers, since the finite-state control can remember the signs of the numbers. Though not necessary (since it is one-way), we assume, for convenience, that the one-way read-only input to the PCA has left and right delimiters. A PCA without a pushdown stack is called a CA. PCAs, even CAs, are quite powerful and can recognize rather complex languages. Decidability/complexity results concerning PCAs (CAs) have been obtained in [22,20]. Some of the results were recently used to show the decidability/complexity of some decision problems (containment, equivalence, disjointness, etc.) for database queries with linear constraints [25,27]. A fundamental result in [22] is the following:

Theorem 1. *The emptiness problem for PCAs* (*i.e.*, *given a PCA M*, *is the language*, *$L(M)$*, *accepted by M empty*?) *is decidable.*

PCAs can be generalized to have multiple one-way input tapes (one head per tape). Thus, a k-tape PCA M accepts a set $L(M)$ of k-tuples of strings. A 1-tape PCA will simply be called a PCA. The following corollary is easily shown using the above theorem (see [23]).

Corollary 1. *The emptiness problem for multitape PCAs is decidable.*

The decision questions (reachability, safety, etc.) investigated in this paper are reducible to the emptiness problem for multitape PCAs.

Theorem 2. (i) *Let A be a TA. Then $R(A)$ is Presburger* [14,16] *and can be accepted by a* 2*-tape CA* [16]. (*This means that we can effectively construct from A a* 2*-tape CA that*, *when given (α, β) on its two input tapes*, *accepts if and only if (α, β) is in $R(A)$*).

(ii) *Let A be a PTA. Then $R(A)$ can be accepted by a* 2*-tape PCA* [16].

(iii) *Let A be a CPTA. Then $R(A)$ can be accepted by a 2-tape PCA* [26].

(iv) *Let A be a CTA. Then $R(A)$ is Presburger and can be accepted by a* 2*-tape CA* [26].

4. Queue-connected timed automata

The model of a synchronized queue-connected reversal-bounded multicounter machines was introduced and investigated in [23]. We now study the model in which the two machines connected by the queue are discrete timed automata that operate in a loosely synchronous manner. Intuitively, a queue-connected discrete timed automaton (QTA) M can be described as follows.

- Two TAs W (the "writer") and R ("the reader").
- One unrestricted queue that can be used to send messages from W to R. There is no bound on the length of the queue.
- Each transition of the timed automaton W (resp. R) is augmented by a write (resp. read) operation to (resp. from) the queue.

Formally, a QTA M is a tuple $\langle S_R, S_W, X, Y, E_R, E_W, \Gamma, d\rangle$ where

- S_R and S_W are two finite sets of (control) states of the reader R and the writer W, respectively.
- X and Y are two sets of clocks for the reader R and the writer W, respectively, with $X \cap Y = \emptyset$.
- Γ is a queue alphabet, and ε and $\#$ are two special symbols not in Γ.
- $E_R \subseteq S_R \times 2^X \times \mathscr{L}_X \times S_R \times (\{\varepsilon, \#\} \cup \Gamma)$ is a finite set of transitions for the reader R. We use $T_R = \langle q, \lambda, l, q', a\rangle$ to denote a transition. A transition T_R is *progressable* if $\lambda = \emptyset$ (i.e, all the clocks in X must progress along the edge).
- $E_W \subseteq S_W \times 2^Y \times \mathscr{L}_Y \times S_W \times (\{\varepsilon\} \cup \Gamma)$ is a finite set of transitions for the writer W. We use $T_W = \langle p, \lambda, l, p', a\rangle \in E_W$ to denote a transition. T_W is *progressable* if $\lambda = \emptyset$.
- $d > 0$, an integer constant, will be made clear in a moment.

A configuration of a QTA is a tuple $\alpha = (q, \mathbf{X}, p, \mathbf{Y}, w)$ where q is a state of R and $\mathbf{X}$ is an array of clock values of R, p is a state of W, $\mathbf{Y}$ is an array of clock values of W, and $w \in \Gamma^*$ is the content of the queue (the leftmost is the head, i.e., the reader end, and the rightmost is the tail, i.e., the writer end).

Before we formally define the semantics of a QTA M, we first intuitively describe what the intended executions of M are. The writer (resp. the reader) can be thought of as a timed automaton with an output (resp. input) tape. In fact, the queue can be regarded as an output tape when writing, and as an input tape when reading from it. The writer and the reader operate independently—the only restriction is that, when the reader reads a symbol a from the queue, a must have been "previously" written by the writer. Therefore, we should have some way to define a (casual) ordering of read/write events in M. We introduce two special clocks now_W (for the writer) and now_R (for the reader). They measure the local time for the reader and the writer, i.e., the total amount of progressable transitions executed so far. Initially, they both start from 0. Clocks now_W and now_R are not necessarily synchronous (imagining that the reader and the writer are located in a distributed environment), but they will not stay away too far from each other. That is, we assume now_W and now_R are *loosely synchronous*, i.e., the difference between them is bounded: $|now_W - now_R| \leqslant d$ for some pre-assigned constant $d > 0$ in the definition of M. With the two local times included, a configuration $\alpha = (q, \mathbf{X}, p, \mathbf{Y}, w)$ is then expanded to an *extended* configuration $\bar{\alpha} = (q, \mathbf{X}, p, \mathbf{Y}, w, now_R, now_W)$. Such $\bar{\alpha}$ is *initial* if $now_R = now_W = 0$, and it is *consistent* if $|now_W - now_R| \leqslant d$. Now, the semantics is defined on extended configurations, as follows.

Let $\alpha = (q, \mathbf{X}, p, \mathbf{Y}, w)$ and $\beta = (q', \mathbf{X}', p', \mathbf{Y}', w')$ be two configurations with their extended configurations $\bar{\alpha} = (q, \mathbf{X}, p, \mathbf{Y}, w, now_R, now_W)$ and $\bar{\beta} = (q', \mathbf{X}', p', \mathbf{Y}', w', now'_R, now'_W)$.

A *read-transition* $T_R \in E_R$ *sends* $\bar{\alpha}$ to $\bar{\beta}$, written $\bar{\alpha} \xrightarrow{T_R} \bar{\beta}$, if

- $T_R = \langle q, \lambda, l, q', a \rangle$ for some $\lambda \subseteq X$, $l \in \mathscr{L}_X$, and $a \in \{\varepsilon, \#\} \cup \Gamma$.
- The state and the clock values of the writer are unaffected. That is, $p = p'$, $\mathbf{Y} = \mathbf{Y}'$ and $now_W = now'_W$.
- When R is regarded as a TA with queue operations ignored, configuration $(q, \mathbf{X})$ reaches configuration $(q', \mathbf{X}')$ along the edge $\langle q, \lambda, l, q' \rangle$. In particular, if the edge is progressable, then $now'_R = now_R + 1$, else $now'_R = now_R$.
- The queue is updated according to symbol a in T_R. That is,
  - If $a = \varepsilon$, then R does not read from the queue; i.e., $w = w'$. In this case, the transition is called internal.
  - If $a \in \Gamma$, then a is the head of the queue and R reads from the queue; i.e., $w = aw'$. The result is that a is deleted from the queue.
  - If $a = \#$, then both w and w' must be the empty string, i.e., the queue is empty. In this case, the transition is called an *empty-queue transition*, which is also internal.

Similarly, a *write-transition* $T_W \in E_W$ *sends* $\bar{\alpha}$ to $\bar{\beta}$, written $\bar{\alpha} \xrightarrow{T_W} \bar{\beta}$, if

- $T_W = \langle p, \lambda, l, p', a \rangle$ for some $\lambda \subseteq Y$, $l \in \mathscr{L}_Y$, and $a \in \{\varepsilon\} \cup \Gamma$.
- The state and the clock values of the reader are unaffected. That is, $q = q'$, $\mathbf{X} = \mathbf{X}'$ and $now_R = now'_R$.
- When W is regarded as a TA with queue operations ignored, configuration $(p, \mathbf{Y})$ reaches configuration $(p', \mathbf{Y}')$ along the edge $\langle p, \lambda, l, p' \rangle$. In particular, if the edge is progressable, then $now'_W = now_W + 1$, else $now'_W = now_W$.
- The queue is updated according to symbol a in T_W. That is,
  - If $a = \varepsilon$, then W does not write on the queue; i.e., $w = w'$. In this case, the transition is called internal.
  - If $a \in \Gamma$, then W writes a on the queue; i.e., $w' = wa$.

Notice that internal transitions do not change the queue content. Both $\xrightarrow{T_R}$ and $\xrightarrow{T_W}$ only characterize local changes with respect to the reader and the writer. The global changes of M are defined through interleavings of $\xrightarrow{T_R}$ and $\xrightarrow{T_W}$. *M sends $\bar{\alpha}$ to $\bar{\beta}$ in a one-step transition*, written $\bar{\alpha} \rightarrow^M \bar{\beta}$, if both $\bar{\alpha}$ and $\bar{\beta}$ are consistent and, either $\bar{\alpha} \xrightarrow{T_R} \bar{\beta}$ for some T_R or $\bar{\alpha} \xrightarrow{T_W} \bar{\beta}$ for some T_W. Define $\rightarrow^*_M$ to be the transitive closure of $\rightarrow^M$. The notation $\bar{\alpha} \rightarrow^*_M \bar{\beta}$ says that $\bar{\alpha}$ can reach $\bar{\beta}$ through a sequence of read-transitions and write-transitions such that each intermediate configuration is consistent (i.e., the local times do not stay too far away from each other). In particular, we write $\alpha \rightsquigarrow^M \beta$ if there are extended configurations $\bar{\alpha}$ and $\bar{\beta}$ such that $\bar{\alpha}$ is initial and $\bar{\alpha} \rightarrow^*_M \bar{\beta}$. The *binary reachability* $R(M)$ of M is the set $\{(\alpha, \beta) : \alpha \rightsquigarrow^M \beta\}$. This paper will show a language property for the binary reachability (when the components of α (resp. β) are represented as strings, separated by markers with the states and the clock values written in unary).

From now on, we assume that $\bar{\alpha}$ and $\bar{\beta}$ are consistent, with $\bar{\alpha}$ being the initial.

Suppose $\bar{\alpha}$ can reach $\bar{\beta}$ through a sequence τ of read- and write-transitions. This particular sequence may not satisfy the condition that each intermediate configuration is consistent, i.e., the sequence is witnessing $\bar{\alpha} \rightarrow^*_M \bar{\beta}$. However, it can be easily observed that, if the sequence is *internal*, i.e., it contains only internal transitions (thus, the queue content will not change by firing the sequence of transitions), then the sequence can be re-organized such that $\bar{\alpha} \rightarrow^*_M \bar{\beta}$.

Lemma 1. *If $\bar{\alpha}$ can reach $\bar{\beta}$ through an internal sequence of read- and write-transitions, then $\bar{\alpha} \to_M^* \bar{\beta}$.*

Assume $\tau = \tau_1 \cdots \tau_m$ and $\eta_i \xrightarrow{\tau_{i+1}} \eta_{i+1}$, for each $0 \leqslant i \leqslant m-1$ with $\eta_0 = \bar{\alpha}$ and $\eta_m = \bar{\beta}$. For now, we assume that, for each i, τ_i is not an empty-queue transition. Later we shall see how to incorporate empty-queue transitions into the sequence τ. Let $\tau_{k_1}, \ldots, \tau_{k_n}$ be all external (i.e., not internal) transitions in τ, with $1 \leqslant k_1 < \cdots < k_n \leqslant m$. From Lemma 1, if, for all $1 \leqslant i \leqslant n$, $\eta_{k_i - 1}$ and η_{k_i} are both consistent, then $\bar{\alpha} \to_M^* \bar{\beta}$. The reason is that transitions between τ_{k_i} and $\tau_{k_{i+1}}$ can be re-organized such that the intermediate configurations are consistent. Therefore, we will focus on considering the external sequence $\tau_{k_1}, \ldots, \tau_{k_n}$. For simplicity, we write this sequence as $\theta_1, \ldots, \theta_n$.

If, for all $1 \leqslant i \leqslant n$, the configuration of M immediately before firing θ_i and the configuration of M immediately after firing θ_i are both consistent, then obviously $\bar{\alpha} \to_M^* \bar{\beta}$. Each θ_i may be associated with a number, or *timestamp*:

- If θ_i is a write-transition writing a symbol a (to the queue), then its timestamp is the value of the local time of the writer after θ_i. This timestamp is also called the write-timestamp for the symbol a.
- If θ_i is a read-transition reading a symbol a (from the queue), then its timestamp is the value of the local time of the reader before θ_i. This timestamp is also called the read-timestamp for the symbol a.

Each individual symbol that appears in the queue during the sequence of transitions $\tau = \tau_1 \cdots \tau_m$ is associated with a pair of a read-timestamp and a write-timestamp (the read-timestamp is ∞ if the symbol is not read from the queue before reaching $\bar{\beta}$; the write-timestamp is 0 if the symbol is originally in the starting configuration $\bar{\alpha}$). In the following, we will deduce a condition on these timestamp pairs. This condition is equivalent to the existence of the required sequence of external transitions $\theta_1, \ldots, \theta_n$ witnessing $\bar{\alpha} \to_M^* \bar{\beta}$. The condition will later allow us to use an alternating simulation technique to show that $\bar{\alpha} \to_M^* \bar{\beta}$.

To derive this condition, we may look at the sequence τ in a different way. Let w be the queue content in α and v be the queue content in β. Consider a string wuv in Γ^*. The string wu consists of all the symbols that will be read by the reader from the queue; while uv consists of all the symbols that will be written by the writer to the queue. wuv is associated with two sequences of (nonnegative) numbers: t_{W} and t_{R}. The ith symbol a in wuv is therefore associated with $t_{\mathrm{W}}(i)$ (the write-timestamp) and $t_{\mathrm{R}}(i)$ (the read-timestamp). When the index is clear, we simply write $t_{\mathrm{W}}(a)$ and $t_{\mathrm{R}}(a)$. Given w, u, v and t_{W}, t_{R}, we assume that:

- Each symbol in w has write-timestamp 0.
- Each $t_{\mathrm{R}}(i), t_{\mathrm{W}}(i) < \infty$ except that each symbol in v has read-timestamp ∞ (indicating that these symbols will not be read).
- $t_{\mathrm{W}}(i) \leqslant t_{\mathrm{W}}(i+1)$ and $t_{\mathrm{R}}(i) \leqslant t_{\mathrm{R}}(i+1)$, for all $1 \leqslant i \leqslant |wuv| - 1$.

Now the QTA can be thought of working in the following way. The reader scans the string wu from left to right while executing its transitions and updating its clocks. Each symbol read from the queue by the reader is exactly the symbol currently being scanned: the reader will make sure that its local time is the same as the read-timestamp of the symbol (provided by t_{R}). The writer, on the other hand, scans the string uv

from left to right. While executing its transitions and updating its clocks, the writer may write a symbol into the queue. The writing is simulated by reading the symbol currently being scanned (by the writer). The writer makes sure that its local time (after the write) is the same as the write-timestamp of the symbol provided by t_W. The reader R and the writer W work independently. Each reading (writing) of R (resp. W) corresponds to an external read-transition (resp. write-transition) θ_i. We have to make sure that (1)

- R and W start from clock values indicated in $\bar{\alpha}$.
- R and W stop at the end of wu and uv, respectively, with clock values indicated in $\bar{\beta}$.

and (2) the *loosely-synchronous conditions* hold:

- Each symbol currently being scanned by R must be already scanned by W. That is, by looking at the positions of R and W, R is never ahead of W.
- The local time difference between R and W is bounded by d. That is, R and W are loosely synchronous.

It is clear that the existence of a sequence of transitions of R and W satisfying the above two conditions will guarantee that $\bar{\alpha} \rightarrow^*_M \bar{\beta}$, since this is equivalent to the existence of an external sequence $\theta_1, \ldots, \theta_n$ mentioned before. But we are interested in finding a condition on t_R and t_W such that, whenever there is a sequence of moves for which condition (1) holds, then the sequence can be modified such that both (1) and (2) hold, i.e., $\bar{\alpha} \rightarrow^*_M \bar{\beta}$. That is, we could use the condition on t_R and t_W to control the pace of R and W to make them loosely synchronous. The condition is:

(3) for each index i in string wu, $t_W(i) - t_R(i) \leqslant d$.

That is, at each position in string wu, the write-timestamp cannot be larger than the read-timestamp by more than d.

Before we justify condition (3), we need a technical lemma.

Lemma 2. *For any two nondecreasing sequences $n_1, \ldots, n_k$ and $m_1, \ldots, m_k$, $k > 1$, of nonnegative integers, the following conditions* (A) *and* (B) *are equivalent*:

(A) *$m_i - n_i \leqslant d$ for each i,*

(B) *there are two nondecreasing sequences $m'_1, \ldots, m'_k$ and $n'_1, \ldots, n'_k$ of nonnegative integers such that*

(a) *$m'_i \leqslant n_i$, for each i,*

(b) *$m_i \leqslant n'_i$, for each i, and*

(c) *$|m'_i - m_i| \leqslant d$ and $|n'_i - n_i| \leqslant d$, for each i.*

Proof. (A) $\Rightarrow$ (B). Take $m'_i = \min(n_i, m_i)$ and $n'_i = \max(n_i, m_i)$. Both $m'_1, \ldots, m'_k$ and $n'_1, \ldots, n'_k$ are two nondecreasing sequences. Clearly, (a) and (b) hold. (A) implies either $m_i \leqslant n_i$ or $m_i \geqslant n_i$ with $|m_i - n_i| \leqslant d$. Both of the cases imply $|\min(n_i, m_i) - m_i| \leqslant d$ and $|\max(n_i, m_i) - n_i| \leqslant d$ (i.e. (c)). Hence, (A) $\Rightarrow$ (B).

(B) $\Rightarrow$ (A). If $m_i < n_i$, (A) trivially holds. If $n_i \leqslant m_i$, then, from (a) and (b), we have $m'_i \leqslant n_i \leqslant m_i \leqslant n'_i$. From (c), we have (A). □

Lemma 3. *Conditions* (1) *and* (3) *are equivalent to the existence of a sequence of transitions of R and W satisfying conditions* (1) *and* (2).

Proof. In order to use Lemma 2, let k be the length of string wu. Values $n_1, \dots, n_k$ are for read-timestamps t_R, and values $m_1, \dots, m_k$ are for write-timestamps t_W. The integer m_i' stands for the local time of the reader when the ith symbol in wu is written by the writer; n_i' stands for the local time of the writer when the ith symbol in wu is read by the reader. Conditions (1) and (2) imply conditions (a), (b) and (c). This is because

- condition (a) requires that this writing is ahead of reading this symbol by the reader,
- condition (b) requires that this reading is after this symbol was written, and
- condition (c) guarantees that the reader and the writer are loosely synchronous at each external transition (i.e., a read or a write of a symbol).

Therefore, from Lemma 2, conditions (1) and (2) also imply conditions (1) and (3). The other direction (i.e., from (1) + (3) to (1) + (2)) of the lemma is obvious, by choosing a proper interleaving of R and W such that m_i' and n_i' are $\min(m_i, n_i)$ and $\max(m_i, n_i)$, respectively, according to the proof of Lemma 2. □

Hence, in order to show $\bar{\alpha} \to_M^* \bar{\beta}$, we need only to show there is a sequence of transitions such that (3) and (1) are satisfied. This essentially says we need only construct two sequences t_R and t_W satisfying (3). This will greatly simplify our discussions, since the construction can be realized by simulating the reader and writer alternately and by checking whether (3) holds, as shown in the following proof. The simulation itself does not guarantee that the reader and the writer are loosely synchronous. However, it will guarantee that there is a sequence of transitions that makes them loosely synchronous.

Let $\alpha = (q, \mathbf{X}, p, \mathbf{Y}, w)$ and $\beta = (q', \mathbf{X}', p', \mathbf{Y}', w')$ be two configurations. Suppose we want to check if β is reachable from α; i.e., $(\alpha, \beta) \in R(M)$. The pair of configurations (α, β) is a tuple $(q, \mathbf{X}, p, \mathbf{Y}, w, q', \mathbf{X}', p', \mathbf{Y}', w')$. For now, to simplify the proofs, we use an equivalent representation of the two configurations:

$$(q, \mathbf{X}, p, \mathbf{Y}, w, q', \mathbf{X}', p', \mathbf{Y}', i, u),$$

where $(q, \mathbf{X}, p, \mathbf{Y}, w)$ is configuration α; $q', \mathbf{X}', p', \mathbf{Y}'$ are the states and clock values of configuration β; u is the string W wrote during the computation from α to β; i is the length of the prefix of wu that was read by R in reaching β. Thus, w' is the suffix of wu starting at position $i+1$. In $(q', \mathbf{X}', p', \mathbf{Y}', i, u)$, we shall refer to $(q', \mathbf{X}', i)$ as the R-component of configuration β and $(p', \mathbf{Y}')$ as the W-component of β.

Based on the discussion of condition (3), we now show that if M is a QTA, then the binary reachability $R(M)$ is computable and can be accepted by a 2-tape PCA.

First we note that we can view the clocks in R and W of a QTA M as counters, which we shall also refer to as *clock-counters*. In a reversal-bounded multicounter machine, only *standard tests* (comparing a counter against 0) and *standard assignments* (increment or decrement a counter by 1, or simply nochange) are allowed. But clock-counters in either R or W do not have standard tests nor standard assignments, for the following reasons. A clock constraint allows comparison between two clocks like $x_1 - x_2 > 5$. Note that using only standard tests we cannot directly compare the difference of two clock-counter values against an integer constant such as 5 just by storing $x_1 - x_2$ in another counter, since each time this "storing" is done, it will cause at least a counter reversal, and the number of such tests during a computation can be unbounded. On the other hand, a clock progress $x := x + 1$ is standard, but a clock reset $x := 0$ is not. Since there is no bound on the number of clock resets, clock-counters may not be reversal-bounded (each reset causes a counter reversal). Besides this obvious obstacle in relating clock-counters to reversal-bounded counters, we have another difficulty in handling the queue in QTA M. It is well known that a finite-state machine augmented with a queue has already the computing power of a Turing machine. In the following intermediate result, we will show an alternating simulation technique to simulate the queue using two one-way input tapes.

Define a semi-PCA as a PCA that, in addition to a stack and reversal-bounded counters, has clock-counters that use nonstandard tests and assignments as described in the above paragraph. The proof of the following theorem uses ideas in [23].

Theorem 3. *We can effectively construct, given a loosely synchronous QTA M, a 2-tape semi-PCA A accepting $R(M)$.*

Proof. Let d be an integer constant associated with M. Suppose that A is given a pair of configurations (α, β) represented by

$$(q, \mathbf{X}, p, \mathbf{Y}, w, q', \mathbf{X}', p', \mathbf{Y}', i, u),$$

where $(q, \mathbf{X}, p, \mathbf{Y}, w)$ is on tape1 and $(q', \mathbf{X}', p', \mathbf{Y}', i, u)$ is on tape2. A first reads $q, \mathbf{X}, p, \mathbf{Y}$ from tape1 and $q', \mathbf{X}', p', \mathbf{Y}', i$ from tape2, using counters to record these values. At this point, tape1 head is at the beginning of w and tape2 head is at the beginning of u. There are two cases to consider. The first case is when R reads only symbols from w and does not read any symbol from u during the computation from α to β. The second case is when R reads past w. The PCA A begins by guessing which case to simulate. We describe only the operation of A for the latter case (which is harder).

The technique is that A alternately simulates R and W with the following procedure. Note that when R (resp. W) is being simulated, W (resp. R) is suspended (without changing state and clock values). The PCA A simulates the computation of R starting in state q and with counter values $\mathbf{X}$, and using tape1 (which contains w). Also, A uses a counter now_{R} to keep track of the running time of R (i.e., now_{R}, initially being 0, counts the number of progressable transitions that R has executed so far; note that now_{R} is nondecreasing.) When R attempts to read past w on tape1, A suspends the simulation (A records the current values) and begins the simulation of W starting in state p and counter values $\mathbf{Y}$. Moreover, A uses a nondecreasing counter now_{W} to keep track of the running time of W (i.e., now_{W} counts the number of progressable transitions W has executed so far). When W writes the first symbol, say a, of u (writing is simulated by reading a symbol of u on tape2), A continues the simulation until W attempts to write the next symbol. When this happens, A then suspends the simulation of W and resumes the simulation of R until R attempts to read past a. Then A resumes the simulation of W. The process of switching the simulations between W and R continues, while a pushdown stack is used to keep track of the difference between the time $t_{\mathrm{R}}(j)$, when the jth symbol of u was read by R, and the time $t_{\mathrm{W}}(j)$, when the jth symbol of u was written by W. The stack makes sure that $t_{\mathrm{W}}(j) - t_{\mathrm{R}}(j) \leqslant d$. This is possible since the pushdown stack can be used as an unrestricted counter (i.e., there is no bound on the reversal). As we assumed before, empty-queue transitions of the reader were not allowed in the process of $\alpha \rightarrow_M^* \beta$. This assumption is not necessary, since whenever an empty-queue transition is executed in the simulation of R, the emptiness of the queue can be justified by $|t_W(j) - t_R(j)| \leqslant d$. At some point during the simulation, after R has read the kth symbol of u (for some k), A guesses that R has read its last symbol from the queue. A continues the simulation of R and at some point guesses that R has reached the R-component of configuration β: A can verify this (also by checking that i is equal to the length of $w+k$, and that the current state and counter values are equal to what were recorded before the start of the simulations). Then A resumes the simulation of W until at some point it guesses that W has reached the W-component of β: again, A can verify this (by checking that W has written the last symbol of u, etc.). The PCA A accepts if and only if $|now_R - now_W| \leqslant d$. □

Theorem 4. *We can effectively construct, given a loosely synchronous QTA M, a 2-tape PCA M' accepting $R(M)$.*

Proof. We now show that the 2-tape semi-PCA A above can be converted to a 2-tape PCA M' accepting $R(M)$, the emptiness problem for the latter being decidable by Corollary 1.

First we show that a 2-tape PCA B can simulate A without using nonstandard tests, but still uses the nonstandard assignments like clock resets (we shall show how to handle this later). In the construction above, A simulates R and W alternately. Each of these machines has clock-counters. We show how B can simulate R without using nonstandard tests. B can then be modified (using a similar construction) to remove nonstandard tests in the simulation of W. The following technique is a modification from the one in [16].

Let R have clock-counters $x_1, \ldots, x_k$. Let m be one plus the maximal absolute value of all the integer constants that appear in the tests (i.e., the clock constraints on the edges of R in the form of Boolean combinations of $x_i \# c$, $x_i - x_j \# c$ with c an integer). Denote the finite set $[m] = \{-m, \ldots, 0, \ldots, m\}$. Define two finite tables with entries a_{ij} and b_i for $1 \leqslant i, j \leqslant k$. Each entry can be regarded as a finite state variable with states in $[m]$. Intuitively, a_{ij} is used to record the difference between two clock values of x_i and x_j, and b_i is used to record the clock value of x_i. During the computation of R, when the difference $x_i - x_j$ (or the value x_i) goes above m or below $-m$, a_{ij} (or b_i) stays the same as m or $-m$. The procedure for updating the entries is given below, where "$\oplus 1$" means adding one if the result does not exceed m, else it keeps the same value. "$\ominus 1$" means subtracting one if the result is not less than $-m$, else it keeps the same value. We will modify R as follows. Let T_R be an edge in E_R. If on the edge the set of clock resets $\lambda = \emptyset$, the entries are updated by adding the following instructions to T_R, for each $1 \leqslant i \leqslant k$:

- $a_{ij} := a_{ij}$ for each $1 \leqslant j \leqslant k$. Recall that all the clocks progress after this edge; thus, the difference is unchanged.
- $b_i := b_i \oplus 1$. That is, clocks progress by one time unit.

If the set of clock resets is $\lambda \neq \emptyset$, the entries are updated by adding the following instructions to T_R, for each $1 \leqslant i, j \leqslant k$:

- $a_{ij} := 0$ if $i \in \lambda$ and $j \in \lambda$. In this case, both the clocks x_i and x_j reset to 0.
- $a_{ij} := -b_j$ if $i \in \lambda$ and $j \notin \lambda$. In this case, x_i resets but x_j does not. So the difference should be $-x_j$.
- $a_{ij} := b_i$ if $i \notin \lambda$ and $j \in \lambda$.
- $a_{ij} := a_{ij}$ if $i \notin \lambda$ and $j \notin \lambda$.

followed by adding the following instructions:

- $b_i := b_i$ if $x_i \notin \lambda$.
- $b_i := 0$ if $x_i \in \lambda$.

The initial values of a_{ij} and b_i can be constructed directly from the values α_{x_i} of clocks x_i in configuration α, for each $1 \leqslant i, j \leqslant k$:

- $a_{ij} := \alpha_{x_i} - \alpha_{x_j}$ if $|\alpha_{x_i} - \alpha_{x_j}| \leqslant m$,
- $a_{ij} := m$ if $\alpha_{x_i} - \alpha_{x_j} > m$,
- $a_{ij} := -m$ if $\alpha_{x_i} - \alpha_{x_j} < -m$,

and, noticing that clocks are nonnegative,

- $b_i := \alpha_{x_i}$ if $\alpha_{x_i} \leqslant m$,
- $b_i := m$ if $\alpha_{x_i} > m$.

B then simulates R exactly except using $a_{ij}\#c$ for a test $x_i - x_j\#c$ and using $b_i\#c$ for $x_i\#c$, with $-m < c < m$. Completely analogous to the proof in [16], one can prove that doing this is valid:

Claim. *Each time after B updates the entries by executing a transition, then the following two conditions hold, for all $1 \leqslant i, j \leqslant k$ and for each integer $c \in [m-1]$:*

$$x_i - x_j\#c \text{ iff } a_{ij}\#c,$$

and

$$x_i\#c \text{ iff } b_i\#c.$$

Thus, clock counter comparisons are replaced by finite table look-up and, therefore, nonstandard tests are eliminated in B. Finally, we show how nonstandard assignments of the form $x_i := 0$ (clock resets) in machine B can be avoided. We only show how these assignments can be avoided in the simulation of R (the same construction applies for W).

After eliminating the clock comparisons, the clock counters in B become blind, i.e., they do not participate in any test except when:

- using the initial values of x_i to compute the initial values of a_{ij} and b_i as shown in the entry update procedure above,
- using the final value of x_i to check whether they match those in β.

Thus, the actual value of each x_i is useless during the simulation of R, but before the very last reset of x_i, since x_i is blind. We describe how to construct a 2-tape PCA C from B such that in the simulation of R, no nonstandard assignment is used. For each clock x_i in R, there are two cases. The first case is that x_i will not be reset during the entire simulation of B. The second case is that x_i will be reset. C guesses the correct case for each x_i. In the first case, x_i is already reversal-bounded and without using nonstandard assignment $x_i := 0$. In the second case, C first decrements x_i to 0. Then C simulates B. Whenever a clock progress $x_i := x_i + 1$ or a clock reset $x_i := 0$ is being executed by R, C keeps x_i as 0. But, at some point when a clock reset $x_i := 0$ is being executed by R, C guesses that this is the last clock reset for x_i. After this point, C faithfully simulates a clock progress $x_i := x_i + 1$ executed by R, and a later execution of a clock reset $x_i := 0$ in R will cause C to abort abnormally (since the guess of the last reset of x_i was wrong.). Thus C uses only standard assignments $x_i := x_i + 1, x_i := x_i$ and $x_i := x_i - 1$ (initially bring x_i to 0 for the second case above).

It follows from the constructions of A, B, and C above that we can effectively construct, given a QTA M, a 2-tape PCA M' accepting $R(M)$. □

Recall that a pair of configurations (α, β) is represented by

$$(q, \mathbf{X}, p, \mathbf{Y}, w, q', \mathbf{X}', p', \mathbf{Y}', i, u),$$

where $(q, \mathbf{X}, p, \mathbf{Y}, w)$ is on tape1 of M' and $(q', \mathbf{X}', p', \mathbf{Y}', i, u)$ is on tape2. Thus tape2 is not the original (but equivalent) representation of β. We can easily construct from M', a 3-tape PCA M'' which when given $(q, \mathbf{X}, p, \mathbf{Y}, w)$ on tape1, $(q', \mathbf{X}', p', \mathbf{Y}', i, u)$ on tape2, and $(q', \mathbf{X}', p', \mathbf{Y}', w')$ on tape3, accepts if and only if M' accepts $(q, \mathbf{X}, p, \mathbf{Y}, w, q', \mathbf{X}', p', \mathbf{Y}', i, u)$, and w' is the suffix of wu starting at position $i + 1$. Clearly, the third tape of M'' is the "original" representation of β. Then from M'', we can construct a 2-tape PCA M''' without the second tape (M''' simulates M'' by guessing the string on tape2 symbol-by-symbol). M''' then accepts $R(M)$, where $\alpha = (q, \mathbf{X}, p, \mathbf{Y}, w)$ is on tape1 and $\beta = (q', \mathbf{X}', p', \mathbf{Y}', w')$ on the other tape. Thus, we can assume that the original representations of the configurations appear on the tapes.

We can augment the W and R of a QTA with a finite number of reversal-bounded counters. Since the counters can be incremented or decremented by 1 and tested for 0, the QTA now have richer enabling conditions in addition to clock constraints, i.e., tests on the counters. (Note that a configuration α will now have the values of the counters.) Clearly, these counters can be faithfully simulated by extra reversal-bounded counters in the 2-tape semi-PCA and 2-tape PCA in the proofs of Theorems 3 and 4. Hence, we have:

Corollary 2. *We can effectively construct, given a loosely synchronous QTA with reversal-bounded counters M, a 2-tape PCA M' accepting $R(M)$.*

5. Loosely synchronous QTA with two queues

A QTA M can be easily modified to add a second queue from the reader back to the writer. That is, M consists of two timed automata A and B working on two queues $queue_1$ and $queue_2$. More precisely, A is able to write to $queue_1$ and read from $queue_2$; B is able to write to $queue_2$ and read from $queue_1$, as shown in Fig. 1.

Obviously, with two queues, M is able to simulate any Turing machine. Therefore, we have to restrict the behavior of M in order to get decidable verification results.

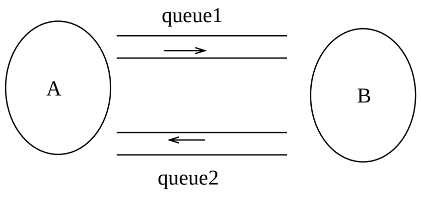

Fig. 1. A QTA with two queues.

One restriction is to make M *half-duplex* [11]. That is, each intermediate configuration during an execution must satisfy: at least one of the two queues is empty. [11] shows that the reachable set of a half-duplex system with two finite state machines is recognizable. Now, we point out that the binary reachability of loosely synchronous half-duplex QTAs still satisfies Theorem 4.

We use $A \Rightarrow B$ to indicate that A is taking the role of a writer and B is of a reader, and both of them working on $queue_1$. The notation $B \Rightarrow A$ is defined symmetrically. In other words, M's execution can be considered as a sequence of phases

$$A \Rightarrow B \Rightarrow A \Rightarrow B \cdots$$

such that when M is engaged in phase $A \Rightarrow B$ (i.e., A is the writer and B is the reader) using $queue_1$, the other queue ($queue_2$ from B to A) is always empty; when M is engaged in phase $B \Rightarrow A$ (i.e., A is the reader and B is the writer) using $queue_2$, the other queue ($queue_1$ from A to B) is always empty. We use A^{W} to denote the result of dropping all read-transitions from A, i.e., A^{W} is a writer. We use A^{R} to denote the result of dropping all write-transitions from A, i.e., A^{W} is a reader. B^{W} and B^{R} are defined similarly. Therefore, an execution of M is considered as an alternation between running a QTA $M_{A\Rightarrow B}$ (which consists of writer A^{W} and reader B^{R}) on $queue_1$, when $queue_2$ is empty, and running a QTA $M_{B\Rightarrow A}$ (which consists of writer B^{W} and reader A^{R}) on $queue_2$, when $queue_1$ is empty. Obviously, both $R(M_{A\Rightarrow B})$ and $R(M_{B\Rightarrow A})$ can be accepted by 2-tape PCAs, as shown in Theorem 4.

Let α and β be two/em half-duplex configurations i.e., one of the two queues in each configuration is empty. Without loss of generality, we assume that $queue_2$ in α and $queue_1$ in β are empty. A sequence of transitions witnessing $\alpha \leadsto^M \beta$ therefore consists of phases

$$A \Rightarrow B \Rightarrow A \cdots \Rightarrow B \Rightarrow A.$$

Hence, we have an execution sequence

$$\eta_0 \leadsto^{M_{A\Rightarrow B}} \eta_1 \leadsto^{M_{B\Rightarrow A}} \eta_2 \cdots \leadsto^{M_{B\Rightarrow A}} \eta_m$$

for some m such that $\eta_0 = \alpha$ and $\eta_m = \beta$, and, $queue_1$ and $queue_2$ are both empty in each η_i with $0 < i < m$. According to Theorem 4, $\eta_0 \leadsto^{M_{A\Rightarrow B}} \eta_1$ can be simulated by the 2-tape PCA for $R(M_{A\Rightarrow B})$; $\eta_1 \leadsto^{M_{B\Rightarrow A}} \eta_2$ can be simulated by the 2-tape PCA for $R(M_{B\Rightarrow A})$. The simulations continue for the rest of the execution sequence. Doing this shows that the above execution sequence can be simulated by a "concatenation" of the two 2-tape PCAs running alternately. This is because:

- both queues in each intermediate configuration η_i are empty, for $0<i<m$.
- When a phase is switched to the other one, clock values in both A and B do not change.

The result of the concatenation is still a 2-tape PCA. Therefore, Theorem 4 still holds for M.

Theorem 5. *We can effectively construct, given a loosely synchronous half-duplex QTA M, a 2-tape PCA M' accepting $R(M)$.*

As in Corollary 2, we have:

Corollary 3. *We can effectively construct, given a loosely synchronous half-duplex QTA with reversal-bounded counters M, a 2-tape PCA M' accepting $R(M)$.*

6. Verification of safety properties

The results of Theorems 4 and 5 allow us to formulate a set of Presburger safety properties that can be automatically verified for loosely synchronous (half-duplex) QTAs as follows.

Given a loosely synchronous (half-duplex) QTA M, let $\alpha, \beta \ldots$ denote variables ranging over configurations. Let $\alpha_{\mathbf{q}}$, α_{x_i}, $\alpha_{\mathbf{p}}$, α_{y_j} and $\alpha_{\mathbf{w}}$ be the state variable (understood as a bounded integer variable) for R, the clock value variables for x_i in R, the state variable for W, the clock value variables for y_j in W, and the queue content variable (when M is half-duplex, this is for the queue other than the empty one), respectively. We use a count variable $\#_\gamma(\alpha_{\mathbf{w}})$ to denote the number of occurrences of a symbol $\gamma \in \Gamma$ in the content of the queue. A QTA-term t is defined as follows:

$$t ::= n \mid \alpha_{\mathbf{q}} \mid \alpha_{x_i} \mid \alpha_{\mathbf{p}} \mid \alpha_{y_j} \mid \#_\gamma(\alpha_{\mathbf{w}}) \mid t - t \mid t + t,$$

where n is an integer, $\gamma \in \Gamma$, $x_i \in X$ and $y_j \in Y$. A QTA-formula f is defined as follows:

$$f ::= t > 0 \mid t \bmod n = 0 \mid \neg f \mid f \vee f,$$

where $n \neq 0$ is an integer. Thus, f is a quantifier-free Presburger formula over state variables, clock value variables and count variables.

For $m \geqslant 1$, let F be a formula in the following format: $\bigvee_{1 \leqslant i \leqslant m}(f_i \wedge \alpha^i \leadsto^M \beta^i)$, where each f_i is a QTA-formula and all i, α^i and β^i are configuration variables. Let $\exists F$ be a closed formula such that each free variable in F is existentially quantified.

The following theorem states that $\exists F$ is verifiable.

Theorem 6. *The truth value of $\exists F$ with respect to a loosely synchronous (half-duplex) QTA M is decidable for any QTA-formula F.*

Proof. Without loss of generality, we assume F is $f \wedge \alpha \rightsquigarrow^M \beta$. The solutions to f can be accepted by a deterministic CA [22], since f is Presburger. On the other hand, from Theorems 4 and 5, $R(M)$ (i.e., the solutions to formula $\alpha \rightsquigarrow^M \beta$ when α and β are understood as configuration variables) can be accepted by a 2-tape PCA. Therefore, the solutions to F can be accepted by a PCA by intersecting the two machines. The theorem follows from Theorem 1. □

From Theorem 6, the following property:

for all configurations α and β with $\alpha \rightsquigarrow^M \beta$, clock x_2 in β is the sum of clocks y_1 and x_2 in α, and symbol γ_1 appears in the queue in β twice as many times as symbol γ_2 does in the queue in α.

can be verified. This is because it can be expressed as,

$$\forall\alpha\forall\beta(\alpha \rightsquigarrow^M \beta \rightarrow (\beta_{x_2} = \alpha_{y_1} + \alpha_{x_2} \wedge \#_{\gamma_1}(\beta_{\mathbf{w}}) = 2\#_{\gamma_2}(\alpha_{\mathbf{w}}))).$$

The negation of this property is equivalent to $\exists F$ for some QTA-formula F. Thus, it can be verified.

Forward reachability and backward reachability are also useful in analyzing safety properties. More precisely, we define the forward reachability set

$$Forward_M(P) = \{\beta\colon\ \exists\alpha \in P\alpha \rightsquigarrow^M \beta\}$$

with respect to a set of configuration P. Similarly, the backward reachability set is

$$Back_M(P) = \{\alpha\colon\ \exists\beta \in P\alpha \rightsquigarrow^M \beta\}.$$

When P can be accepted by a CA (for instance, P is definable by a QTA-formula), we can show that both the forward and the backward reachability sets can be accepted by PCAs.

Theorem 7. *Let M be a loosely synchronous (half-duplex) QTA and P be a set of configurations accepted by a CA. Then, both $Forward_M(P)$ and $Back_M(P)$ can be accepted by PCAs.*

Proof. We only prove the case for $Forward_M(P)$. The case for $Back_M(P)$ is similar. We construct a PCA M' that accepts $Forward_M(P)$. Given a configuration β on the input tape, M' guesses a configuration α while it simulates the CA accepting P. At the end of the simulation, M' verifies that α is accepted (i.e., $\alpha \in P$). In parallel to this simulation, M' also simulates the PCA M'' accepting $R(M)$ (Theorems 4 and 5) using another set of counters (sharing the same starting values in α) and the pushdown stack. When M'' accesses the second tape, M' guesses the tape content for it. At the end, M' also verifies configuration β is reached when M'' does so. Clearly, M' accepts $Forward_M(P)$. □

We conclude this section by noting that Theorems 6 and 7 remain valid when M has reversal-bounded counters.

7. An example

In this section, we illustrate the use of loosely synchronous half-duplex QTAs with reversal-bounded counters (see Corollary 3). Consider a system used in a physics experiment, composed of a set of actuators (for controlling the experiment) and of a set of sensors (for measuring and recording various experimental data such as the speed and number of subatomic particles). A controller is in charge of controlling the actuators and the sensors, and of elaborating the data detected by the sensors. It is crucial for the experiment that the sensors collect data only in a precise interval, which may vary depending on various conditions, and send the data to the controller upon request.

We model only one sensor and one controller, and ignore all other components. Data are read by the sensor with a variable speed, depending on the environment. We assume that the sensor needs one time unit is needed to read a datum and another time unit to communicate it to the controller. Hence, incoming data have a maximum rate of one datum The sensor is associated with a cheap embedded processor, with small computation power and very little memory; hence, it cannot store the data it reads, but it must send them immediately to the controller.

The controller is a powerful processor, with a large memory. However, it has other tasks to perform and it cannot continuously elaborate the data coming from the sensor. Incoming data are then put in a queue and read when the controller is ready to make use of them. The protocol in charge of correctly exchanging data between the sensor and the controller (such as the acknowledgement of packet arrival, etc.) is considered to be at a lower level and is not modeled here. We just assume that when data are sent from one end to another, they are correctly put in the queue.

The sensor is required to read data only within a precise time interval, whose integer length $k>0$ is communicated by the controller at the beginning of the experiment. After this, the controller may decide nondeterministically when it is the moment to start reading data, by sending a signal “begin” to the sensor. Upon receipt of the signal, the sensor must read data for exactly k time units. Only data read after the “begin”, but before the end of the interval of length k, may be sent to the controller.

The clocks of the controller and of the sensor are allowed not to be perfectly synchronized: they may drift away within a small positive integer constant d. Hence, the actual reading interval, measured in the local time of the sensor, may end earlier or later than the instant intended by the controller. To reduce this effect, it is required that $k>d$.

The problem is modeled with a loosely synchronous QTA M with two queues and reversal-bounded (r.b.) counters. M is composed of two timed automata with r.b. counters, S (the sensor) and C (the controller). The property to be verified is that all and only the data communicated to the controller are read in the correct time interval. We model only the case of just one session, i.e., a stream of data is collected only once by S upon request from C, and then both subsystems halt. However, a one-session system is also a model of a multi-session system where all sessions are far apart enough so as not to interfere with each other.

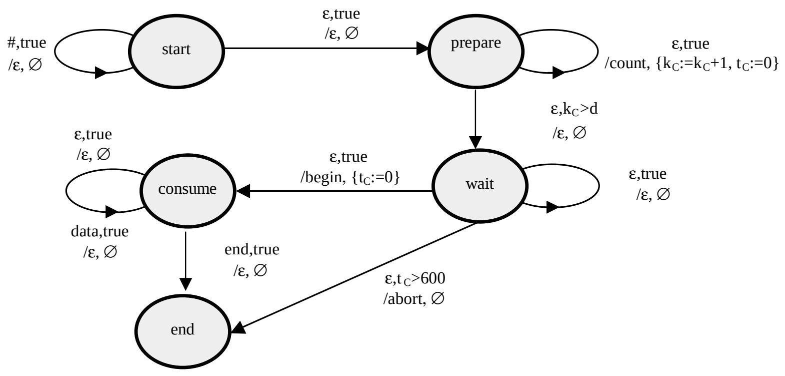

Fig. 2. The controller.

S may write to $queue_1$ and C to $queue_2$. The alphabet of $queue1$ is: $\{data, end\}$, and the alphabet of $queue2$ is: $\{count, begin, abort\}$.

The length of the interval, communicated by C to S, is stored in a one-reversal counter k_S of S. This is a value $k > d$, nondeterministically chosen by C and also stored in a counter k_C of C. In S, there is another counter j_S, introduced only for the purpose of verification, which is used to count the data read by S and subsequently sent to C. In C (and in S as well), there is just one clock, t_C (resp. t_S), which is incremented by one at each transition, unless it is explicitly reset.

The automaton C can be in one of the states:

$$\{start, prepare, wait, consume, end\}$$

and has a clock t_C. Its transition graph is described in Fig. 2. Each label of an edge has four components: the first is the symbol read from the queue, the second is the enabling condition on both clocks and reversal-bounded counters, the third (denoted after a slash) is the symbol written on the other queue, and the fourth is the set of assignments to clocks and counters. For the sake of readability, the set of clock resets of an edge is denoted by assigning the value 0 to each clock in the set (e.g. $t_C := 0$).

For instance, the edge labeled $\varepsilon, true/count, \{k_C := k_C + 1, t_C := 0\}$ can be taken without reading from the queue and with the enabling condition being just *true*; the result of taking that edge is that *count* is written on the other queue, the clock t_C is reset and the counter k_C is incremented by 1.

The automaton S can be in one of the states:

$$\{start, load, wait, sense, write, last, end\}.$$

Its transition graph is described in Fig. 3.

A run of the system begins with both automata in state *start*, with all clocks and r.b. counters starting at 0. Both may stay in *start* as long as the queues are empty (i.e., they read the special symbol #). Nondeterministically, C may decide to move to state *prepare* (causing S to move to state *load*), where a self-loop increases k_C and sends

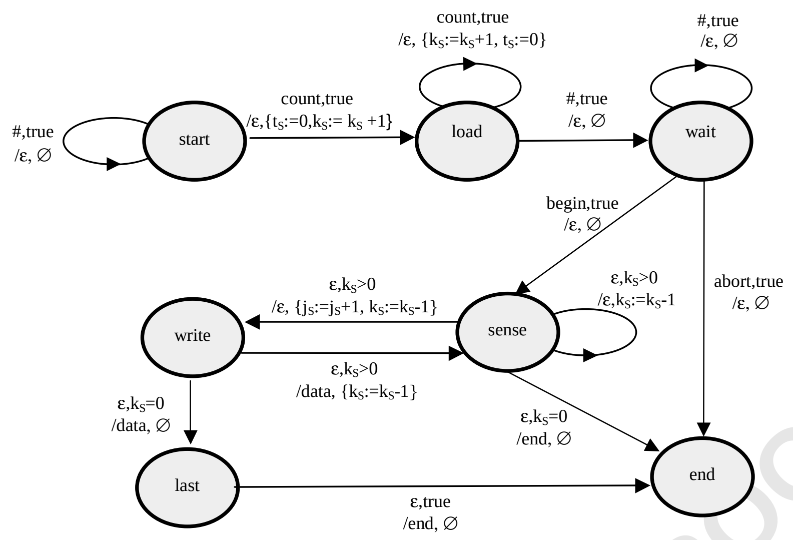

Fig. 3. The sensor.

a *count* signal to S, that in response increments k_S. The final value k of k_C (hence, also of k_S) is chosen nondeterministically beyond the constant d. In both automata, the clocks t_S and t_C are initialized to zero at each transition: hence, both self-loops are in zero-time. This zero-time assumption is reasonable, since we may imagine that the time it takes to transfer the interval length is much smaller than the time to transfer data packets.

When $k_C > d$, C may keep increasing k_C or nondeterministically go to state *wait* (imitated by S), where time passes until C decides that S must start reading. When the latter case occurs, a signal *begin* is sent to S and C enters state *consume*. However, the controller has only a limited time to send a *begin*: a timeout transition, going from state *wait* to state *end*, aborts operations if the controller stays in state *wait* longer than, say, 600 instants, sending an *abort* signal to the sensor. Correspondingly, S waits in state *load* until the queue is empty: if S receives an *abort* in state *load*, it goes to state *end*.

In state *consume*, time may progress while C is idle or reading, at any speed, data coming from S. Upon receipt of event *begin*, S immediately goes to state *sense* and starts reading data: when a datum is detected, counter j_S is increased, counter k_S is decreased and S goes to state *write* to send *data* to C. As already remarked, the actions of reading a datum and sending a datum take one time unit each. The counter k_S must be decremented whenever time progresses. Hence, it is decremented while S is waiting for data in the state *sense* and also when a datum is read and then written to C. Notice that, since there can be at most one datum each two time units, no data are lost while making the transitions from *sense* to *write* and back. When k_S reaches zero, the reading interval is over and S goes to its final state *end*, emitting a signal *end* towards C. The automaton S may also go through state *last*, in order to write also data that arrived at the very last instant, again followed by an *end* signal. When an *end* signal is found on the queue, C goes to its final state.

Verification of the example: In what follows, α, β, γ, and δ represent configurations. Let $initial(\alpha)$ be a formula stating that α is the initial configuration, i.e. a configuration where both S and C are in the initial state and each clock and each counter is set to 0; let $state(C,\beta)$ and $state(S,\beta)$ be the state of C and the state of S, respectively, in a configuration β; let $begin(\beta)$ be the formula $state(C,\beta)$ is $consume) \wedge \beta_{t_C} = 0$, i.e., C has just made the transition signaling that data reading should start; let $expired(\beta,\gamma)$ be the formula $\gamma_{now_S} - \beta_{now_S} = \beta_{k_C} \wedge d < \beta_{k_C}$, which holds if in γ exactly $k_C > d$ instants have passed since β; let $newData(\gamma,\delta)$ be the formula $\delta_{now_S} > \gamma_{now_S} \wedge \delta_j > \gamma_j$, which holds if in δ time has progressed since γ and more data were read. Then let P_1 be the formula:

$$\forall \alpha,\beta,\gamma,\delta(\alpha \rightsquigarrow^{\mathcal{M}} \beta \rightsquigarrow^{\mathcal{M}} \gamma \rightsquigarrow^{\mathcal{M}} \delta \wedge initial(\alpha) \wedge begin(\beta) \wedge expired(\beta,\gamma) \\ \rightarrow \neg newData(\gamma,\delta))$$

Formula P_1 states that no data are read after the interval has expired. The negation of P_1 is a formula where all variables are quantified existentially, which can be decided automatically. If $\neg P_1$ is true, then the above specification does not enforce the basic requirement on the system, since a "wrong" configuration δ is reachable.

If $\neg P_1$ is false, we may erroneously be confident that the system behaves correctly. However, $\neg P_1$ may be false not only when no configuration δ as above is reachable, but also when there are no reachable configurations β or γ as above. In the latter case, there would be a different modeling error, namely the two automata are not even able to reach a configuration where S can start reading data. Let P_2 be the formula

$$\exists \alpha,\beta,\gamma(\alpha \rightsquigarrow^{\mathcal{M}} \beta \rightsquigarrow^{\mathcal{M}} \gamma \wedge initial(\alpha) \wedge begin(\beta) \wedge expired(\beta,\gamma)).$$

It is not necessary to verify whether P_2 holds when $\neg P_1$ is true, but it is enough to execute the verification of P_2 only when $\neg P_1$ is false.

If both P_1 and P_2 hold, the verification is not complete yet, since we also need the property P_3 that no data is read before the reading interval has started. Since $j_S > 0$ if some data have been read, and since the reading interval may only start when C has gone through the state *consume*, the negation of P_3 can be written as:

$$\exists \alpha,\beta(\alpha \rightsquigarrow^{\mathcal{M}} \beta \wedge initial(\alpha) \wedge \beta_{j_S} > 0 \wedge state(C,\beta) \neq consume \\ \wedge state(C,\beta) \neq end).$$

Again, this is a formula whose truth can be decided: if it does not hold, then property P_3 is violated.

8. Conclusions

We introduced a generalization of a discrete timed automaton, i.e., two TAs connected by a unidirectional queue (each of the TAs may also be augmented with reversal-bounded counters) and analyzed the solvability of verification problems such as (binary, forward, and backward) reachability. The two automata operate in a loosely synchronous way, though our results also hold for the case when they are synchronous (i.e., sharing a global clock with $d = 0$) and the case when they are asynchronous (i.e., $d = \infty$). Using an easier (but slightly different) argument, we can show that the proof of Theorem 3 still holds when

- if the QTA is synchronous, the test $t_W(j) - t_R(j) \leqslant d$ in the proof is replaced by $t_W(j) - t_R(j) \leqslant 0$;
- if the QTA is asynchronous, the test $t_W(j) - t_R(j) \leqslant d$ in the proof is replaced by *true* (i.e., no tests).

Under both cases, all the results for the QTAs still hold. The QTA models can be used to reason about a number of timed producer/consumer applications involving only one-way communications. We are able to extend the results to a restricted form of QTA with two half-duplex queues. This opens the door for verification of a restricted form of two-way timed communication protocols.

A special case of a QTA is one where W and R have *no* clocks and no reversal-bounded counters, i.e., they are nondeterministic finite-state machines connected by a queue. We call such a model finite-state QTA. It has been shown in [23] that binary reachability is not computable (i.e., not recursive) for the following models: (i) Finite-state QTA with another (second) queue that can be used to send messages from W to R; (ii) Finite-state QTA with a second queue that can be used to send messages from R to W (thus, there is now two-way communication between the machines); (iii) Finite-state QTA where each of R and W is augmented with a one-turn pushdown stack (i.e., after popping, the stack can no longer push); (iv) Finite-state QTA where each of R and W is augmented with an unrestricted counter.

It would be interesting to further consider the QTA model with dense time. However, the technical difficulties forbidding us to do so are the lack of theoretical tool to handle both dense variables and unbounded discrete data structures in one system. Recent results in [15] show some hope in this direction, by introducing an infinite partition on the dense clock space. We may investigate the dense time version of QTAs in the future. We also leave the work of complexity analysis of the decision procedures presented in this paper as future work. We note, however, that by using recent techniques developed in [24], we can, in fact, strengthen our results in that the 2-tape PCA in Theorems 4 and 5 and their corollaries can be reduced to a 2-tape CA (i.e., the stack is not necessary).

References

[1] R. Alur, Timed automata. CAV'99, Lecture Notes in Computer Science, Vol. 1633, Springer, Berlin, pp. 8–22.
[2] R. Alur, D. Dill, A theory of timed automata, Theoret. Comput. Sci. 126 (2) (1994) 183–236.
[3] R. Alur, T.A. Henzinger, A really temporal logic, J. Assoc. Comput. Mach. 41 (1) (1994) 181–204.
[4] R. Alur, C. Courcoibetis, D. Dill, Model-checking in dense real time, Informat. Comput. 104 (1) (1993) 2–34.
[5] P. Abdulla, B. Jonsson, Verifying programs with unreliable channels, Inform. Comput. 127 (2) (1996) 91–101.

[6] D. Beauquier, A. Slissenko, The railroad crossing problem: towards semantics of timed algorithms and their model-checking in high-level languages, Lecture Notes in Computer Science, Vol. 1214, Springer, Berlin, 1997, pp. 202–212.
[7] G. von Bochman, Finite state descriptions of communicating protocols, Comput. Networks 2 (1978) 361–372.
[8] A. Bouajjani, J. Esparza, O. Maler, Reachability analysis of pushdown automata: application to model—Checking, CONCUR'97, Lecture Notes in Computer Science, Vol. 1243, Springer, Berlin, pp. 135–150.
[9] D. Brand, P. Zafiropulo, On communicating finite-state machines, J. Assoc. Comput. Mach. 30 (2) (1983) 323–342.
[10] T. Bultan, R. Gerber, W. Pugh, Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results, TOPLAS 21 (4) (1999) 747–789.
[11] G. Cece, A. Finkel, Programs with quasi-stable channels are effectively recognizable, CAV'97, Lecture Notes in Computer Science, Vol. 1254, Springer, Berlin, pp. 304–315.
[12] E. Clarke, J. Wing, Formal methods: state of the art and future directions, Assoc. Comput. Mach. Comput. Surveys 28 (4) (1996) 626–643.
[13] H. Comon, Y. Jurski, Multiple counters automata, safety analysis and Presburger arithmetic, CAV'98, Lecture Notes in Computer Science, Vol. 1427, Springer, Berlin, pp. 268–279.
[14] H. Comon, Y. Jurski, Timed automata and the theory of real numbers, CONCUR'99, Lecture Notes in Computer Science, Vol. 1664, Springer, Berlin, pp. 242–257.
[15] Z. Dang, Binary reachability analysis of pushdown timed automata with dense clocks, CAV'01, Lecture Notes in Computer Science, Vol. 2102, Springer, Berlin, pp. 506–517.
[16] Z. Dang, O.H. Ibarra, T. Bultan, R.A. Kemmerer, J. Su, Binary reachability analysis of discrete pushdown timed automata, CAV'00, Lecture Notes in Computer Science, Vol. 1855, Springer, Berlin, pp. 69–84.
[17] A. Finkel, G. Sutre, Decidability of reachability problems for classes of two counter automata, STACS'00, Lecture Notes in Computer Science, Vol. 1770, Springer, Berlin, pp. 346–357.
[18] A. Finkel, B. Willems, P. Wolper, A direct symbolic approach to model checking pushdown systems, Proceedings of the INFINITY'97.
[19] S. Ginsburg, E. Spanier, Bounded algol-like languages, Trans. Amer. Math. Soc. 113 (1964) 333–368.
[20] E.M. Gurari, O.H. Ibarra, The complexity of decision problems for finite-turn multicounter machines, J. Comput. System Sci. 22 (1981) 220–229.
[21] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine, Symbolic model checking for real-time systems, Inform. Comput. 111 (2) (1994) 193–244.
[22] O.H. Ibarra, Reversal-bounded multicounter machines and their decision problems, J. Assoc. Comput. Mach. 25 (1978) 116–133.
[23] O.H. Ibarra, Reachability and safety in queue systems with counters and pushdown stack, Proceedings of the International Conference on Implementation and Application of Automata, 2000.
[24] O.H. Ibarra, Z. Dang, On removing the pushdown stack in reachability constructions. Proceedings of the International Symposium on Algorithms and Computation (ISAAC), 2001.
[25] O.H. Ibarra, J. Su, A technique for the containment and equivalence of linear constraint queries, J. Comput. System Sci. 59 (1) (1999) 1–28.
[26] O.H. Ibarra, J. Su, Generalizing the discrete timed automaton, Proceedings of the International Conference on Implementation and Application of Automata, 2000.
[27] O.H. Ibarra, J. Su, C. Bartzis, Counter machines and the safety and disjointness problems for database queries with linear constraints, in: Words, Sequences, Languages: Where Computer Science, Biology and Linguistics Meet, Kluwer Academic Publishers, Dordrecht, 2000, to appear.
[28] R. Mayr, Decidability and complexity of model-checking problems for infinite state systems, Ph.D. Thesis, Inst. für Informatik, Techn. Universität München, 1998.
[29] K.L. McMillan, Symbolic model-checking—an approach to the state explosion problem, Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University, 1992.
[30] W. Peng, S. Purushothaman, Analysis of a class of communicating finite state machines, Acta Informat. 29 (6/7) (1992) 499–522.
[31] S. Yovine, Model-checking timed automata, Embedded Systems, Lecture Notes in Computer Science, Vol. 1494, Springer, Berlin, 1998, pp. 114–152.