Time-Bounded Reachability Problem for Recursive Timed Automata Is Undecidable

Shankara Narayanan Krishna, Lakshmi Manasa(✉), and Ashutosh Trivedi

Indian Institute of Technology Bombay, Mumbai, India
{krishnas,manasa,trivedi}@cse.iitb.ac.in

Abstract. Motivated by the success of bounded model checking framework for finite state machines, Ouaknine and Worrell proposed a time-bounded theory of real-time verification by claiming that restriction to bounded-time recovers decidability for several key decision problem related to real-time verification. In support of this theory, the list of undecidable problems recently shown decidable under time-bounded restriction is rather impressive: language inclusion for timed automata, emptiness problem for alternating timed automata, and emptiness problem for rectangular hybrid automata. The objective of our study was to recover decidability for general recursive timed automata(RTA)—and perhaps for recursive hybrid automata—under time-bounded restriction in order to provide an appealing verification framework for powerful modeling environments such as Stateflow/Simulink. Unfortunately, however, we answer this question in negative by showing that time-bounded reachability problem stays undecidable for RTA with 5 clocks.

Keywords: Recursive State Machines · Timed Automata · Reachability

1 Introduction

Recursive state machines (RSMs), as introduced by Alur, Etessami, and Yannakakis [2], are a variation on various visual notations to represent hierarchical state machines, notably Harel's statecharts [8] and Object Management Group supported UML diagrams [11], that permits recursion while disallowing concurrency. RSMs closely correspond [2] to pushdown systems [6], context-free grammars, and Boolean programs [4], and provide a natural specification and verification framework to reason with sequential programs with recursive procedure calls. The two fundamental verification questions for RSM, namely reachability and Büchi emptiness checking, are known to be decidable in polynomial time [2,7].

Timed automata [3] extend finite state machines with continuous variables called clocks that permit a natural modeling of timed systems. In a timed automaton the variables continuously flow with uniform rates within each discrete state, while they are allowed to have discontinuous jumps during transitions between states that are guarded by constraints over variables. It is well

A.-H. Dediu et al. (Eds.): LATA 2015, LNCS 8977, pp. 237–248, 2015.
DOI: 10.1007/978-3-319-15579-1_18

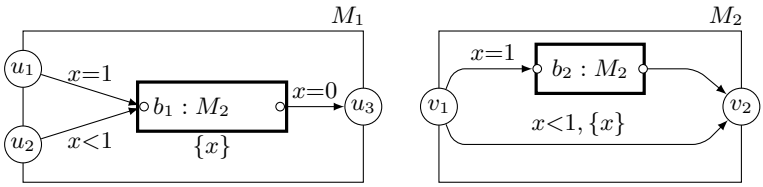

Fig. 1. An example of recursive timed automata with one clock and two components

known that the reachability problem is decidable (PSPACE-complete) for timed automata [3]. Trivedi and Wojtczak [12] introduced *recursive timed automata* (RTAs) as an extension of timed automata with recursion to model real-time software systems. Formally, an RTA is a finite collection of components where each component is a timed automaton that in addition to making transitions between various states, can have transitions to "boxes" that are mapped to other components modeling a potentially recursive call to a subroutine. During such invocation a limited information can be passed through clock values from the "caller" component to the "called" component via two different mechanism: a) *pass-by-value*, where upon returning from the called component a clock assumes the value prior to the invocation, and b) *pass-by-reference*, where upon return clocks reflect any changes to the value inside the invoked procedure.

Example 1 (Visual Presentation). The visual presentation of a recursive timed automaton with two components M_1 and M_2, and one clock variable x is shown in Figure 1 (example taken from [12]), where component M_1 calls component M_2 via box b_1 and component M_2 recursively calls itself via box b_2. Components are shown as thinly framed rectangles with their names written next to upper right corner. Various control states, or "nodes", of the components are shown as circles with their labels written inside them, e.g. see node u_1. Entry nodes of a component appear on the left of the component (see u_1), while exit nodes appear on the right (see u_3).

Boxes are shown as thickly framed rectangles inside components labeled $b : M$, where b is the label of the box, M is the component it is mapped to. We write the set of clocks passed to M by value just below the box, while we omit this notation if all the clocks are passed by reference. The rest of the clocks are assumed to be passed by reference. For the sake of clarity of presentation, we often abuse the notation and write $\overline{Y}$ to denote the set $\mathcal{X} \setminus Y$. For instance, in the component Inc c shown in Figure 3, we pass the clocks $\{x, z_1, z_2, b\}$ by value to the component mapped to the box F_1:Up$_2^y$.

Call ports of boxes are drawn as small circles on the left of the box, while return ports are on the right. We omit labeling the call and return ports as these labels are clear from their position on the boxes. For example, call port (b_1, v_1) is the top small circle on the left-hand side of box b_1, since box b_1 is mapped to M_2 and v_1 is the top node on its left-hand side. Each transition is labeled with

a guard and the set of reset variables, (e.g. transition from node v_1 to v_2 can be taken only when variable $x<1$, and after this transition is taken, x is reset).

Related Work. Trivedi and Wojtczak [12] showed that the reachability and termination (reachability with empty calling context) problem is undecidable for RTAs with three or more clocks. Moreover, they considered the so-called glitch-free restriction of RTAs—where at each invocation either all clocks are passed by value or all clocks are passed by reference— and showed that the reachability (and termination) is EXPTIME-complete for RTAs with two or more clocks. In the model of [12] it is compulsory to pass all the clocks at every invocation with either mechanism. Abdulla, Atig, and Stenman [1] studied a related model called timed pushdown automata where they disallowed passing clocks by value. They allowed clocks to be passed either by reference or not passed at all (in that case they are stored in the call context and continue to tick with the uniform rate). It is shown in [1] that the reachability problem for this class remains decidable (EXPTIME-complete). In this paper we restrict ourselves to the recursive timed automata model as introduced in [12]. In particular, we consider time-bounded reachability problem for RTA and show that the problem stays undecidable for RTA with 5 or more clocks. We have also studied two player reachability games on RTA in [9], and showed that time-bounded reachability games are undecidable for RTA with 3 or more clocks.

For a survey of models related to recursive timed automata and dense-time pushdown automata we refer the reader to [12]and [1]. Another closely related model is introduced in [5] where pushdown automata is extended with an additional stack used to store clock valuations. The reachability problem is known to be undecidable for this model. We do not consider this model in the current paper, but we conjecture that time-bounded reachability problem for this model is also undecidable.

2 Preliminaries

2.1 Labeled Transition System

A *labeled transition system* (LTS) is a tuple $\mathcal{L} = (S, A, X)$ where S is the set of *states*, A is the set of *actions*, and $X : S \times A \to S$ is the *transition function*. We say that an LTS $\mathcal{L}$ is *finite* (*discrete*) if both S and A are finite (countable). We write $A(s)$ for the set of actions available at $s \in S$, i.e., $A(s) = \{a : X(s,a) \neq \emptyset\}$.

We say that $(s, a, s') \in S \times A \times S$ is a transition of $\mathcal{L}$ if $s' = X(s,a)$ and a *run* of $\mathcal{L}$ is a sequence $\langle s_0, a_1, s_1, \ldots\rangle \in S \times (A \times S)^*$ such that (s_i, a_{i+1}, s_{i+1}) is a transition of $\mathcal{L}$ for all $i \geq 0$. We write $Runs^{\mathcal{L}}$ ($FRuns^{\mathcal{L}}$) for the sets of infinite (finite) runs and $Runs^{\mathcal{L}}(s)$ ($FRuns^{\mathcal{L}}(s)$) for the sets of infinite (finite) runs starting from state s. For a set $F \subseteq S$ and a run $r = \langle s_0, a_1, \ldots\rangle$ we define $Stop(F)(r) = \inf\{i \in \mathbb{N} : s_i \in F\}$. Given a state $s \in S$ and a set of final states $F \subseteq S$ we say that a final state is reachable from s_0 if there is a run $r \in Runs^{\mathcal{L}}(s_0)$ such that $Stop(F)(r) < \infty$. Given an LTS, an initial state, and a set of final states, the *reachability problem* for LTS is to decide whether a final state is reachable from the given initial state.

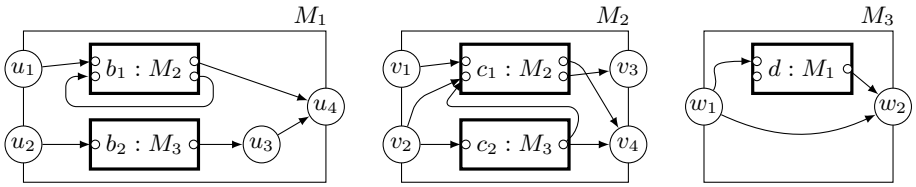

Fig. 2. Example recursive state machine taken from [2]

2.2 Recursive State Machines

Definition 2. *A* recursive state machine *[2]* $\mathcal{M}$ *is a tuple* $(\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k)$ *of components, where each component* $\mathcal{M}_i = (N_i, \mathrm{EN}_i, \mathrm{EX}_i, B_i, Y_i, A_i, X_i)$ *for each* $1 \leq i \leq k$ *is such that:*

- N_i *is a finite set of* nodes *including a distinguished set* EN_i *of* entry nodes *and a set* EX_i *of* exit nodes *such that* EX_i *and* EN_i *are disjoint sets;*
- B_i *is a finite set of* boxes*;*
- $Y_i : B_i \rightarrow \{1, 2, \ldots, k\}$ *is a mapping that assigns every box to a component. We associate a set of* call ports *Call*(b) *and return ports Ret*(b) *to each box* $b \in B_i$*: Call*$(b) = \{(b, en) : en \in \mathrm{EN}_{Y_i(b)}\}$ *and Ret*$(b) = \{(b, ex) : ex \in \mathrm{EX}_{Y_i(b)}\}$*. Let* $Call_i = \cup_{b \in B_i} Call(b)$ *and* $Ret_i = \cup_{b \in B_i} Ret(b)$ *be the set of call and return ports of component* $\mathcal{M}_i$*.*
  We define the set of vertices Q_i *of component* $\mathcal{M}_i$ *as the union of the set of nodes, call ports and return ports, i.e.* $Q_i = N_i \cup Call_i \cup Ret_i$*;*
- A_i *is a finite set of* actions*; and*
- $X_i : Q_i \times A_i \rightarrow Q_i$ *is the transition function with a condition that call ports and exit nodes do not have any outgoing transitions.*

For the sake of simplicity, we assume that the set of boxes $B_1, \ldots, B_k$ *and the set of nodes* $N_1, N_2, \ldots, N_k$ *are mutually disjoint. We use symbols* N, B, A, Q, X*, etc. to denote the union of the corresponding symbols over all components.*

An example of a RSM is shown in Figure 2. An execution of a RSM begins at the entry node of some component and depending upon the sequence of input actions the state evolves naturally like a labeled transition system. However, when the execution reaches an entry port of a box, this box is stored on a stack of pending calls, and the execution continues naturally from the corresponding entry node of the component mapped to that box. When an exit node of a component is encountered, and if the stack of pending calls is empty, then the run terminates; otherwise, it pops the box from the top of the stack, and jumps to the exit port of the just popped box corresponding to the just reached exit of the component. We formalize the semantics of a RSM using a discrete LTS, whose states are pairs consisting of a sequence of boxes, called the context, mimicking the stack of pending calls and the current vertex.

Definition 3 (RSM semantics). *Let $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k)$ be an RSM where the component $\mathcal{M}_i$ is $(N_i, En_i, Ex_i, B_i, Y_i, A_i, X_i)$. The semantics of $\mathcal{M}$ is the discrete labelled transition system $[\![\mathcal{M}]\!] = (S_{\mathcal{M}}, A_{\mathcal{M}}, X_{\mathcal{M}})$ where:*

- *$S_{\mathcal{M}} \subseteq B^* \times Q$ is the set of states;*
- *$A_{\mathcal{M}} = \cup_{i=1}^{k} A_i$ is the set of actions;*
- *$X_{\mathcal{M}} : S_{\mathcal{M}} \times A_{\mathcal{M}} \to S_{\mathcal{M}}$ is the transition relation such that for $s = (\langle \kappa \rangle, q) \in S_{\mathcal{M}}$ and $a \in A_{\mathcal{M}}$, we have that $s' = X_{\mathcal{M}}(s, a)$ if and only if one of the following holds:*
  1. *the vertex q is a call port, i.e. $q = (b, en) \in Call$, and $s' = (\langle \kappa, b \rangle, en)$;*
  2. *the vertex q is an exit node, i.e. $q = ex \in \mathrm{EX}$ and $s' = (\langle \kappa' \rangle, (b, ex))$ where $(b, ex) \in Ret(b)$ and $\kappa = (\kappa', b)$;*
  3. *the vertex q is any other vertex, and $s' = (\langle \kappa \rangle, q')$ and $q' \in X(q, a)$.*

Given $\mathcal{M}$ and a subset $Q' \subseteq Q$ of its nodes we define the set $[\![Q']\!]_{\mathcal{M}}$ as the set $\{(\langle \kappa \rangle, v') : \kappa \in B^* \text{ and } v' \in Q'\}$. We also define the set of terminal configurations $Term_{\mathcal{M}}$ as the set $\{(\langle \varepsilon \rangle, ex) : ex \in \mathrm{EX}\}$ with the empty context $\langle \varepsilon \rangle$. Given a recursive state machine $\mathcal{M}$, an initial node v, and a set of *final vertices* $F \subseteq Q$ the *reachability problem* on $\mathcal{M}$ is defined as the reachability problem on the LTS $[\![\mathcal{M}]\!]$ with the initial state $(\langle \varepsilon \rangle, v)$ and final states $[\![F]\!]$. We define *termination problem* as the reachability of one of the exits with the empty context. The following is a well known result.

Theorem 4 ([2]). *The reachability and the termination problem for recursive state machines can be solved in polynomial time.*

3 Recursive Timed Automata

Recursive timed automata (RTAs) extend classical timed automata (TAs) with recursion in a similar way RSMs extend LTSs.

3.1 Syntax

Let $\mathbb{R}$ be the set of real numbers. Let $\mathcal{X}$ be a finite set of real-valued clocks. A *valuation* on $\mathcal{X}$ is a function $\nu : \mathcal{X} \to \mathbb{R}$. We assume an arbitrary but fixed ordering on the clocks and write x_i for the variable with order i. This allows us to treat a valuation ν as a point $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n)) \in \mathbb{R}^{|\mathcal{X}|}$. For a subset of clocks $X \subseteq \mathcal{X}$ and a valuation $\nu' \in \mathcal{X}$, we write $\nu[X{:=}\nu']$ for the valuation where $\nu[X{:=}\nu'](x) = \nu'(x)$ if $x \in X$, and $\nu[X{:=}\nu'](x) = \nu(x)$ otherwise. The valuation $\mathbf{0} \in \mathbb{R}^{|\mathcal{X}|}$ is a special valuation such that $\mathbf{0}(x) = 0$ for all $x \in \mathcal{X}$.

We define a constraint over a set $\mathcal{X}$ as a subset of $\mathbb{R}^{|\mathcal{X}|}$. We say that a constraint is *rectangular* if it is defined as the conjunction of a finite set of constraints of the form $x \bowtie k$, where $k \in \mathbb{Z}$, $x \in \mathcal{X}$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. For a constraint G, we write $[\![G]\!]$ for the set of valuations in $\mathbb{R}^{|\mathcal{X}|}$ satisfying the constraint G. We write $\top$ (resp., $\bot$) for the special constraint that is true (resp., false) in all the valuations, i.e. $[\![\top]\!] = \mathbb{R}^{|\mathcal{X}|}$ (resp., $[\![\bot]\!] = \emptyset$). We write $\mathrm{rect}(\mathcal{X})$ for the set of rectangular constraints over $\mathcal{X}$ including $\top$ and $\bot$.

Definition 5. *A* recursive timed automaton $\mathcal{H} = (\mathcal{X}, (\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k))$ *is a pair of set of clocks* $\mathcal{X}$ *and a collection of components* $(\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k)$ *where every* $\mathcal{H}_i = (N_i, \mathrm{EN}_i, \mathrm{EX}_i, B_i, Y_i, A_i, X_i, P_i, Inv_i, E_i, J_i, F_i)$ *is such that:*

- N_i *is a finite set of* nodes *including a distinguished set* EN_i *of* entry nodes *and a set* EX_i *of* exit nodes *such that* EX_i *and* EN_i *are disjoint sets;*
- B_i *is a finite set of* boxes*;*
- $Y_i : B_i \to \{1, 2, \ldots, k\}$ *is a mapping that assigns every box to a component. (Call ports* $Call(b)$ *and return ports* $Ret(b)$ *of a box* $b \in B_i$*, and call ports* $Call_i$ *and return ports* Ret_i *of a component* $\mathcal{H}_i$ *are defined as before. We set* $Q_i = N_i \cup Call_i \cup Ret_i$ *and refer to this set as the set of locations of* $\mathcal{H}_i$*.)*
- A_i *is a finite set of* actions*.*
- $X_i : Q_i \times A_i \to Q_i$ *is the transition function with a condition that call ports and exit nodes do not have any outgoing transitions.*
- $P_i : B_i \to 2^{\mathcal{X}}$ *is pass-by-value mapping that assigns every box the set of clocks that are passed by value to the component mapped to the box; (The rest of the clocks are assumed to be passed by reference.)*
- $Inv_i : Q_i \to rect(\mathcal{X})$ *is the* invariant condition*;*
- $E_i : Q_i \times A_i \to rect(\mathcal{X})$ *is the* action enabledness function*;*
- $J_i : A_i \to 2^{\mathcal{X}}$ *is the* variable reset function*;*

We assume that the sets of boxes, nodes, locations, etc. are mutually disjoint across components and we write $(N, B, Y, Q, P, X,$ *etc.) to denote corresponding union over all components.*

We say that a recursive timed automaton is *glitch-free* if for every box either all clocks are passed by value or none is passed by value, i.e. for each $b \in B$ we have that either $P(b) = \mathcal{X}$ or $P(b) = \emptyset$. Any general RTA with one clock is trivially glitch-free. We say that a RTA is *hierarchical* if there exists an ordering over components s.t. a component never invokes another component of higher order or same order.

3.2 Semantics

A *configuration* of an RTA $\mathcal{H}$ is a tuple $(\langle\kappa\rangle, q, \nu)$, where $\kappa \in (B \times \mathbb{R}^{|\mathcal{X}|})^*$ is sequence of pairs of boxes and variable valuations, $q \in Q$ is a location and $\nu \in \mathbb{R}^{|\mathcal{X}|}$ is a variable valuation over $\mathcal{X}$ such that $\nu \in Inv(q)$. The sequence $\langle\kappa\rangle \in (B \times \mathbb{R}^{|\mathcal{X}|})^*$ denotes the stack of pending recursive calls and the valuation of all the variables at the moment that call was made, and we refer to this sequence as the context of the configuration. Technically, it suffices to store the valuation of variables passed by value, because other variables retain their value after returning from a call to a box, but storing all of them simplifies the notation. We denote the the empty context by $\langle\epsilon\rangle$. For any $t \in \mathbb{R}$, we let $(\langle\kappa\rangle, q, \nu)+t$ equal the configuration $(\langle\kappa\rangle, q, \nu+t)$.

Informally, the behavior of an RTA is as follows. In configuration $(\langle\kappa\rangle, q, \nu)$ time passes before an available action is triggered, after which a discrete transition occurs. Time passage is available only if the invariant condition $Inv(q)$ is satisfied while time elapses, and an action a can be chosen after time t elapses only if

it is enabled after time elapse, i.e., if $\nu+t \in E(q,a)$. If the action a is chosen then the successor state is $(\langle\kappa\rangle, q', \nu')$ where $q' \in X(q,a)$ and $\nu' = (\nu+t)[J(a) := \mathbf{0}]$. Formally, the semantics of an RTA is given by an LTS which has both an uncountably infinite number of states and transitions.

Definition 6 (RTA semantics). *Let $\mathcal{H} = (\mathcal{X}, (\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k))$ be an RTA where each component is of the form $\mathcal{H}_i = (N_i, \mathrm{EN}_i, \mathrm{EX}_i, B_i, Y_i, A_i, X_i, P_i, Inv_i, E_i, J_i, F_i)$. The semantics of $\mathcal{H}$ is a labelled transition system $[\![\mathcal{H}]\!]=(S_{\mathcal{H}},A_{\mathcal{H}},X_{\mathcal{H}})$ where:*

- $S_{\mathcal{H}} \subseteq (B \times \mathbb{R}^{|\mathcal{X}|})^* \times Q \times \mathbb{R}^{|\mathcal{X}|}$, *the set of states, is s.t.* $(\langle\kappa\rangle, q, \nu) \in S_{\mathcal{H}}$ *if* $\nu \in Inv(q)$.
- $A_{\mathcal{H}} = \mathbb{R}_{\oplus} \times A$ *is the set of* timed actions, *where* $\mathbb{R}_{\oplus}$ *is the set of non-negative reals;*
- $X_{\mathcal{H}} : S_{\mathcal{H}} \times A_{\mathcal{H}} \to S_{\mathcal{H}}$ *is the transition function such that for* $(\langle\kappa\rangle, q, \nu) \in S_{\mathcal{H}}$ *and* $(t,a) \in A_{\mathcal{H}}$, *we have* $(\langle\kappa'\rangle, q', \nu') = X_{\mathcal{H}}((\langle\kappa\rangle, q, \nu), (t,a))$ *if and only if the following condition holds:*
  1. *if the location q is a call port, i.e. $q = (b, en) \in Call$ then $t = 0$, the context $\langle\kappa'\rangle = \langle\kappa, (b,\nu)\rangle$, $q' = en$, and $\nu' = \nu$.*
  2. *if the location q is an exit node, i.e. $q = ex \in Ex$, $\langle\kappa\rangle = \langle\kappa'', (b,\nu'')\rangle$, and let $(b, ex) \in Ret(b)$, then $t = 0$; $\langle\kappa'\rangle = \langle\kappa''\rangle$; $q' = (b, ex)$; and $\nu' = \nu[P(b) := \nu'']$.*
  3. *if location q is any other kind of location, then $\langle\kappa'\rangle = \langle\kappa\rangle$, $q' \in X(q,a)$, and*
     (a) $\nu + t' \in Inv(q)$ for all $t' \in [0,t]$;
     (b) $\nu + t \in E(q,a)$;
     (c) $\nu' = (\nu+t)[J(a) := \mathbf{0}]$.

3.3 Reachability and Time-Bounded Reachability Problems

For a subset $Q' \subseteq Q$ of states of a recursive time automaton $\mathcal{H}$ we define the set $[\![Q']\!]_{\mathcal{H}}$ as the set $\{(\langle\kappa\rangle, q, \nu) \in S_{\mathcal{H}} : q \in Q'\}$. We define the terminal configurations as $Term_{\mathcal{H}} = \{(\langle\varepsilon\rangle, q, \nu) \in S_{\mathcal{H}} : q \in \mathrm{EX}\}$. Given a recursive timed automaton $\mathcal{H}$, an initial node q and valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, and a set of *final locations* $F \subseteq Q$, the *reachability problem* on $\mathcal{H}$ is to decide the existence of a run in the LTS $[\![\mathcal{H}]\!]$ staring from the initial state $(\langle\varepsilon\rangle, q, \nu)$ to some state in $[\![F]\!]_{\mathcal{H}}$. As with RSMs, we also define *termination problem* as reachability of one of the exits with the empty context. Hence, given an RTA $\mathcal{H}$ and an initial node q and a valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, the termination problem on $\mathcal{H}$ is to decide the existence of a run in the LTS $[\![\mathcal{H}]\!]$ from initial state $(\langle\varepsilon\rangle, q, \nu)$ to a final state in $Term_{\mathcal{H}}$.

Given a run $r = \langle s_0, (t_1, a_1), s_2, (t_2, a_2), \ldots, (s_n, t_n)\rangle$ of an RTA, its time duration $\mathsf{time}(r)$ is defined as $\sum_{i=1}^{n} t_i$. Given a recursive timed automaton $\mathcal{H}$, an initial node q, a bound $T \in \mathbb{N}$, and valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, and a set of *final locations* $F \subseteq Q$, the *time-bounded reachability problem* on $\mathcal{H}$ is to decide the existence of a run r in the LTS $[\![\mathcal{H}]\!]$ staring from the initial state $(\langle\varepsilon\rangle, q, \nu)$ to some state in $[\![F]\!]_{\mathcal{H}}$ such that $\mathsf{time}(r) \leq T$. Time-bounded termination problem is defined in an analogous manner. The following is the key result of the paper which is proved in the rest of the paper.

Theorem 7. *Time-Bounded Reachability problem is undecidable for unrestricted RTAs with at least 5 clocks.*

4 Undecidability of Time-Bounded Reachability Problem

In this section, we provide a complete proof of Theorem 7 by reducing the halting problem for two counter machines to the reachability problem in an RTA.

A *two-counter machine* M is a tuple (L, CTR) where $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$ is the set of instructions including a distinguished terminal instruction ℓ_n called HALT, and the set $CTR = \{C, D\}$ of two *counters*. The instructions L are of the type:

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

where $c_j \in CTR$, $\ell_i, \ell_k, \ell_m \in L$. A configuration of a two-counter machine is a tuple (ℓ, c, d) where $\ell \in L$ is an instruction, and $c, d \in \mathbb{N}$ are the values of counters C and D, resp. A run of a two-counter machine is a (finite or infinite) sequence of configurations $\langle k_0, k_1, \ldots \rangle$ where $k_0 = (\ell_0, 0, 0)$ and the relation between subsequent configurations is governed by transitions between respective instructions. The *halting problem* for a two-counter machine asks whether its unique run ends at the terminal instruction ℓ_n. It is well known ([10]) that the halting problem for two-counter machines is undecidable.

In order to prove the results of Theorem 7, we construct a recursive timed automaton whose main components simulate various instructions. In these constructions the reachability of the exit node of each component corresponding to an instruction is due to a faithful simulation of various increment, decrement and zero check instructions of the machine by choosing appropriate delays to adjust the clocks, to reflect changes in counter values.

We specify a main component for each instruction of the two counter machine. The entry node and exit node of a main component corresponding to an increment instruction $\langle \ell_i : c_j := c_j + 1;$ goto $\ell_m \rangle$ are respectively ℓ_i and ℓ_m. Similarly, a main component corresponding to a zero check instruction $\langle \ell_i :$ if $(c_j > 0)$ then goto ℓ_m else goto $\ell_n \rangle$, has a unique entry node ℓ_i, and two exit nodes corresponding to ℓ_m and ℓ_n respectively. We get the complete RTA for the two-counter machine when we connect these main components in the same sequence as the corresponding machine. We prove that the problem of reaching a chosen vertex in an RTA within 18 units of total elapsed time is undecidable. In order to get the undecidability result, we use a reduction from the halting problem for two counter machines. Our reduction uses an RTA with 5 clocks.

We maintain three sets of clocks. The first set $X = \{x\}$ encodes correctly the current value of counter C; the second set $Y = \{y\}$ encodes correctly the current value of counter D; while the third set $Z = \{z_1, z_2\}$ of 2 clocks helps in zero-check. An extra clock b is used to enforce urgency in some locations. The clock b is zero at the entry nodes of all the main components. Let $\mathcal{X}$ be the set of all 5 clocks.

To be precise, on entry into a main component simulating the $(k+1)$th instruction, we have the values of z_1, z_2 as $\nu(Z) = 1 - \frac{1}{2^k}$, the value of x as

$\nu(x) = 1 - \frac{1}{2^{c+k}}$, and the value of y as $\nu(y) = 1 - \frac{1}{2^{d+k}}$, where c, d are the current values of the counters after simulating the first k instructions. If the $(k+1)$th instruction ℓ_{k+1} is an increment counter C instruction, then after the simulation of ℓ_{k+1}, we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(x) = 1 - \frac{1}{2^{c+k+2}}$ and $\nu(y) = 1 - \frac{1}{2^{d+k+1}}$. Similarly, if ℓ_{k+1} is a decrement C instruction, then after the simulation of ℓ_{k+1}, we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(x) = 1 - \frac{1}{2^{c+k}}$ and $\nu(y) = 1 - \frac{1}{2^{d+k+1}}$. Likewise, if ℓ_{k+1} is a zero check instruction, then after the simulation of ℓ_{k+1}, we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(x) = 1 - \frac{1}{2^{c+k+1}}$ and $\nu(y) = 1 - \frac{1}{2^{d+k+1}}$. We show by our construction that, the time taken to simulate the $(k+1)$th instruction is $< \frac{9}{2^k}$. Thus, the time taken to simulate the first instruction is < 9, the second instruction is $< \frac{9}{2} \ldots$, so that the total time taken in simulating the two counter machine is < 18.

Increment Instruction. Let us discuss the case of simulating an increment instruction for counter C. Assume that this is the $(k+1)$th instruction. Figure 3 gives the figure for incrementing counter C. At the entry node en_1 of the component Inc c, we have $\nu(x) = 1 - \frac{1}{2^{c+k}}$, $\nu(y) = 1 - \frac{1}{2^{d+k}}$ and $\nu(Z) = 1 - \frac{1}{2^k}$, and $\nu(b) = 0$.

The component Inc c has three subcomponents sequentially lined up one after another: Let $\beta = \frac{1}{2^k}, \beta_c = \frac{1}{2^{c+k}}$, and $\beta_d = \frac{1}{2^{d+k}}$.

1. The first subcomponent is Up_2^y. If Up_2^y is entered with $\nu(y) = 1 - \beta_d$, then on exit, we have $\nu(y) = 1 - \frac{\beta_d}{2}$. The values of X, Z are unchanged. Also, the total time elapsed in Up_2^y is $\leq \frac{5\beta}{2}$.
2. The next subcomponent is Up_4^x. If Up_4^x is entered with $\nu(x) = 1 - \beta_c$, then on exit, we have $\nu(x) = 1 - \frac{\beta_c}{4}$. The values of Z, Y are unchanged. Also, the total time elapsed in Up_4^x is $\leq \frac{11\beta}{4}$.
3. The next subcomponent is Up_2^Z which updates the value of Z. If Up_2^Z is entered with $\nu(Z) = 1 - \beta$, then on exit, we have $\nu(Z) = 1 - \frac{\beta}{2}$. The values of X, Y are unchanged. Also, the total time elapsed in Up_2^Z is $\leq \frac{5\beta}{2}$.
4. Thus, at the end of the Inc c, we obtain $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(x) = 1 - \frac{1}{2^{c+k+2}}$, $\nu(y) = 1 - \frac{1}{2^{d+k+1}}$. Also, the total time elapsed in Inc c is $\leq [\frac{5}{2} + \frac{11}{4} + \frac{5}{2}]\beta < 8\beta$.

Consider subcomponent Up_2^y obtained by instantiating a by y and n by 2 in the Up_n^a in the Figure 3. Let us discuss the details of Up_2^y, Up_4^x and Up_2^Z have similar functionality.

1. On entry into the first subcomponent F_4:D, we have $\nu(Z) = 1 - \beta$, $\nu(b) = 0$, $\nu(x) = 1 - \beta_c$, $\nu(y) = 1 - \beta_d$. D is called, and clock z_2 is passed by reference and the rest by value. A non-deterministic amount of time t_1 elapses at the entry node en_3 of D. At the return port of F_4:D, we have z_2 added by t_1.
2. We are then at the entry node of the subcomponent F_5:$C_{z_2}^{y=}$ with values $\nu(z_2) = 1 - \beta + t_1$, and $\nu(z_1) = 1 - \beta$, $\nu(x) = 1 - \beta_c$, $\nu(y) = 1 - \beta_d$ and $\nu(b) = 0$. $C_{z_2}^{y=}$ is called by passing all clocks by value. The subcomponent $C_{z_2}^{y=}$ ensures that $t_1 = \beta - \beta_d$.
3. To ensure $t_1 = \beta - \beta_d$, at the entry node en_4 of $C_{z_2}^{y=}$, a time β_d elapses. This makes $y = 1$. If z_2 must be 1, then we need $1 - \beta + t_1 + \beta_d = 1$, or the time t_1 elapsed is $\beta - \beta_d$. That is, $C_{z_2}^{y=}$ ensures that z_2 has grown to be equal

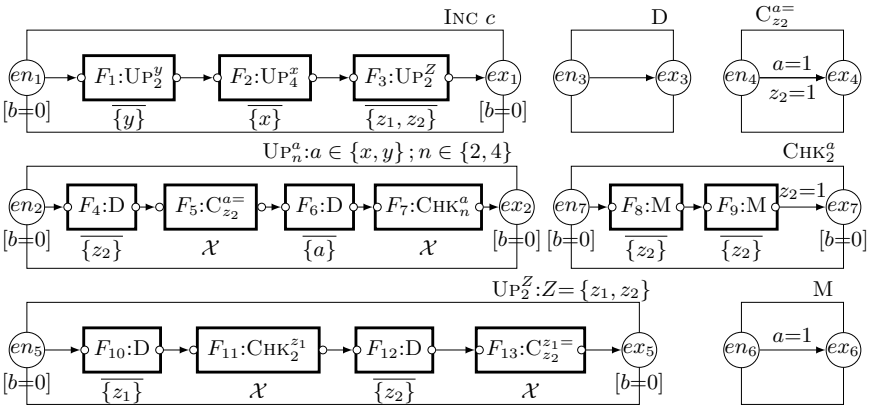

Fig. 3. *TB Term* in RTA: Increment c. Note that $a \in \{x, y\}$. $C^{z_1=}_{z_2}$ is obtained by instantiating $a = z_1$ in $C^{a=}_{z_2}$. The component Chk$^a_4$ is similar to Chk$^a_2$. It has 4 calls to M inside it each time passing only z_2 by reference.

to y by calling F_4:D. At the return port of F_5:$C^{y=}_{z_2}$, we next enter the call port of F_6:D with $\nu(z_2) = \nu(y) = 1 - \beta_d$ and $\nu(z_1) = 1 - \beta$. D is called by passing y by reference, and all others by value. A non-deterministic amount of time t_2 is elapsed in D. At the return port of F_6:D, we get $\nu(z_1) = 1 - \beta$, $\nu(z_2) = 1 - \beta_d$, and $\nu(y) = 1 - \beta_d + t_2$.

4. At the call port of F_7:Chk$^y_2$, we have the same values, since $b = 0$ has to be satisfied at the exit node ex_7 of Chk$^y_2$. That is, at the call port of F_7:Chk$^y_2$, we have $\nu(z_1) = 1-\beta$, $\nu(z_2) = 1-\beta_d$, and $\nu(y) = 1-\beta_d+t_2$. F_7 calls Chk$^y_2$, and passes all clocks by value. Chk$^y_2$ checks that $t_2 = \frac{\beta_d}{2}$.
5. At the entry port en_7 of Chk$^y_2$, no time elapses. Chk$^y_2$ sequentially calls M twice, each time passing z_2 by reference, and all others by value. In the first invocation of M, we want y to reach 1; thus a time $\beta_d - t_2$ is spent at en_6. This makes $z_2 = 1 - \beta_d + \beta_d - t_2 = 1 - t_2$. After the second invocation, we obtain $z_2 = 1 + \beta_d - 2t_2$ at the return port of F_9:M. No time can elapse at the return port of F_9:M; for z_2 to be 1, we need $t_2 = \frac{\beta_d}{2}$.
6. No time elapses in the return port of F_7:Chk$^y_2$, and we are at the exit node ex_2 of Up$^y_2$. Now, we have $\nu(z_1) = 1 - \beta$, $\nu(z_2) = 1 - \beta_d$ and $\nu(y) = 1 - \beta_d + t_2 = 1 - \frac{\beta_d}{2}$.
7. The time elapsed in Up$^y_2$ is the sum of t_1, t_2 and the times elapsed in $C^{y=}_{z_2}$ and Chk$^y_2$. That is, $(\beta - \beta_d) + \frac{\beta_d}{2} + \beta_d + 2(\beta_d - t_2) = \beta + \frac{3\beta_d}{2} \leq \frac{5\beta}{2}$ since $\beta_d \leq \beta$.

At the return port of F_1 : Up$^y_2$, we thus have $\nu(Z) = 1 - \beta$ ($\nu(z)$ restored to $1 - \beta$ as it was passed by value to Up$^y_2$), $\nu(x) = 1 - \beta_c$, and $\nu(y) = 1 - \frac{\beta_d}{2}$. No time elapses here, and we are at the call port of F_2 : Up$^x_4$. The component Chk$^x_4$ is similar to Chk$^y_2$. It has 4 calls to M inside it, each passing respectively, z_2 by reference to M and x by value. An analysis similar to the above gives that the total time elapsed in Up$^x_4$ is $\leq \frac{11\beta}{4}$, and at the return port of F_2 : Up$^x_4$,

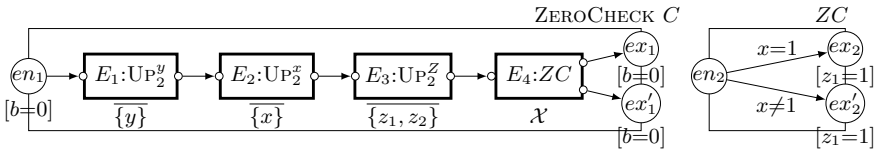

Fig. 4. Unrestricted RTA with 5 clocks: Zero Check if $(C = 0)$ then reaches exit ex_1 else reaches ex'_1. Note that $\overline{S} = \mathcal{X} - S$. Up_2^y, Up_2^x and Up_2^Z are in Figure 3.

we get $\nu(x) = 1 - \frac{\beta_c}{4}$, $\nu(y) = 1 - \frac{\beta_d}{2}$ and $\nu(Z) = 1 - \beta$. This is followed by entering $F_3 : \text{Up}_2^Z$, with these values. At the return port of $F_3 : \text{Up}_2^Z$, we obtain $\nu(x) = 1 - \frac{\beta_c}{4}$, $\nu(y) = 1 - \frac{\beta_d}{2}$ and $\nu(Z) = 1 - \frac{\beta}{2}$, with the total time elapsed in Up_2^Z being $\leq \frac{5\beta}{2}$.

From the explanations above, the following propositions can be proved. The same arguments given above will apply to prove this.

Proposition 8. *For any box B and context $\langle\kappa\rangle$, and $\nu(Z) = 1-\beta$, we have that* $(\langle\kappa\rangle, (B, en), (\nu(x), \nu(y), 1 - \beta, \nu(b))) \xrightarrow[\text{Up}_2^Z]{*} (\langle\kappa\rangle, (B, ex), (\nu(x), \nu(y), 1 - \frac{\beta}{2}, \nu(b)))$.

Proposition 9. *For any box B and context $\langle\kappa\rangle$, and $\nu(x) = 1-\beta_c$, we have that* $(\langle\kappa\rangle,(B,en),(1 - \beta_c,\nu(y),\nu(Z),\nu(b))) \xrightarrow[\text{Up}_4^x]{*} (\langle\kappa\rangle,(B,ex),(1 - \frac{\beta_c}{4},\nu(y),\nu(Z),\nu(b)))$.

Proposition 10. *For any box B and context $\langle\kappa\rangle$, and $\nu(y) = 1-\beta_d$, we have* $(\langle\kappa\rangle,(B,en),(\nu(x),1-\beta_d,\nu(Z),\nu(b))) \xrightarrow[\text{Up}_2^y]{*} (\langle\kappa\rangle,(B,ex),(\nu(x),1-\frac{\beta_d}{2},\nu(Z),\nu(b)))$.

Decrement Instruction. Assume that the $(k + 1)$st instruction is decrementing counter C. Then we construct the main component Dec c similar to the component Inc c above. The main component Dec c will have the subcomponents Up_2^y and Up_2^Z lined up sequentially. There is no need for any Up_n^x subcomponent here, since the value of x stays unchanged on decrementing c. The total time spent in Dec c is also, less than 8β.

Zero Check Instruction. The main component for checking if counter $C = 0$ is given in Figure 4. It follows the same pattern as Inc c. Components Up_2^y, Up_2^x and Up_2^Z update the clock values to account for end of the $k + 1$ instruction : changing the value of x from $1 - \beta_c$ to $1 - \frac{\beta_c}{2}$, of y from $1 - \beta_d$ to $1 - \frac{\beta_d}{2}$ and that of Z from $1 - \beta$ to $1 - \frac{\beta}{2}$. Additionally, the final component ZC, to which all clocks are passed by value, checks if $x = z$ by checking $z_1 = 1 \wedge x = 1$ which means $c = 0$ else $z_1 = 1 \wedge x_1 \neq 1$ meaning $c \neq 0$. Time elapsed in ZC component is $\frac{\beta}{2}$ (to make $z_1 = 1$). Thus, total time spent in ZeroCheck c is $< 9\beta$.

Note that the components for incrementing, decrementing and zero check of counter D can be obtained in a manner similar to the above. The proof that we reach the vertex *Halt* of the RTA iff the two counter machine halts follows: Clearly, the exit node of each main component is reached iff the corresponding

instruction is simulated correctly. Thus, if the counter machine halts, we will indeed reach the exit node of the main component corresponding to the last instruction. However, if the machine does not halt, then we keep going between the various main components simulating each instruction, and never reach $Halt$.

Acknowledgments. This work is partially supported by CEFIPRA funded project AVeRTS (Algorithmic Verification of Real-Time Systems). The second author is supported by Microsoft Research Fellowship RSMSRI0033.

References

1. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-timed pushdown automata. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science. LICS, pp. 35–44 (2012)
2. Alur, R., Benedikt, M., Etessami, K., Godefroid, P., Reps, T., Yannakakis, M.: Analysis of recursive state machines. ACM Trans. Program. Lang. Syst. **27**(4), 786–818 (2005)
3. Alur, R., Dill, D.: Automata for modeling real-time systems. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 322–335. Springer, Heidelberg (1990)
4. Ball, T., Rajamani, S.K.: Bebop: A symbolic model checker for boolean programs. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN 2000. LNCS, vol. 1885, pp. 113–130. Springer, Heidelberg (2000)
5. Benerecetti, M., Minopoli, S., Peron, A.: Analysis of timed recursive state machines. In: Proceedings of 17th International Symposium on Temporal Representation and Reasoning TIME, pp. 61–68 (2010)
6. Mazurkiewicz, A., Winkowski, J.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
7. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for modelchecking pushdown systems. In: Emerson, E., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 232–247. Springer, Heidelberg (2000)
8. Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987)
9. Krishna, S.N., Manasa, L., Trivedi, A.: Improved undecidability results for reachability games on recursive timed automata. In: Proceedings of Fifth International Symposium on Games, Automata, Logics and Formal Verification GandALF, pp. 245–259 (2014)
10. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Inc. (1967)
11. Rumbaugh, J., Jacobson, I., Booch, G.: The unified modeling language reference manual. Addison-Wesley-Longman (1999)
12. Trivedi, A., Wojtczak, D.: Recursive timed automata. In: Bouajjani, A., Chin, W.-N. (eds.) ATVA 2010. LNCS, vol. 6252, pp. 306–324. Springer, Heidelberg (2010)