# Recursive Timed Automata

Ashutosh Trivedi and Dominik Wojtczak

Computing Laboratory, Oxford University, UK

**Abstract.** We study *recursive timed automata* that extend timed automata with recursion. Timed automata, as introduced by Alur and Dill, are finite automata accompanied by a finite set of real-valued variables called clocks. Recursive timed automata are finite collections of timed automata extended with special states that correspond to (potentially recursive) invocations of other timed automata from their collection. During an invocation of a timed automaton, our model permits passing the values of clocks using both *pass-by-value* and *pass-by-reference* mechanisms. We study the natural reachability and termination (reachability with empty invocation stack) problems for recursive timed automata. We show that these problems are decidable (in many cases with the same complexity as the reachability problem on timed automata) for recursive timed automata satisfying the following condition: during each invocation either all clocks are passed by reference or none is passed by reference. Furthermore, we show that for recursive timed automata that violate this condition reachability/termination problems are undecidable for automata with as few as three clocks. We also establish similar results for two-player game extension of our model against reachability/termination objective.

## 1 Introduction

Recursion is one of the central ideas in mathematics and computer science. Informally, recursion is a process in which objects are defined in terms of other objects of same type. For instance, *recursive state machines* [1] are defined as collection of rather peculiar state machines whose states, in addition to being states in usual sense, are allowed to be other state machines, including themselves; or in other words, some states may correspond to potentially recursive invocation of other state machines. Similarly, *recursive Markov decision processes* [14] are collection of special Markov decision processes whose states may correspond to the invocation of other Markov decision processes. Following this line of work, we define recursive extension of timed automata [2] and study reachability and termination problems for this model.

Timed automata are finite automata—a finite set of locations and a finite set of transitions—coupled with a finite set of continuous variables, called clocks, which grow with uniform slope. Simple form of constraints on clocks are allowed to appear as guards on the transitions and as location invariants. Syntax of timed automata also permits resetting the clocks to zero. The reachability problem for timed automata with at least three clocks is known to be PSPACE-complete, while the reachability problem is NLOGSPACE-complete for timed automata with one clock.

Recursive timed automaton consists of finite number of *components* where each component is a special form of timed automaton with specially marked entry and exit locations. Moreover components can also have special form of locations, called *boxes*, that correspond to recursive invocation of other components. We allow passing the values of clocks to the invoked component in the sense that the values of these clocks are available to the invoked component, and passed clocks grow normally while the invoked component is under execution. Moreover, the passed clocks can be used in guards and location invariants inside the component, and transitions of the component may reset these clocks to zero. We allow two different mechanisms of passing the clocks: 1) *pass-by-value*, where upon returning from the invoked component clocks assume the value prior to the invocation; and 2) *pass-by-reference*, where upon returning from the invoked component their value is unaltered (it is as if a copy of these clocks is restored in the calling component). We say that a clock is *global* if it is always passed by reference, and it is *local* if it is always passed by value. Notice that, since there is no bound on the depth of recursive calls in our model, we will need to be able to analyse potentially infinitely many clocks, but at any point of the execution only a fixed number of them is not stopped.

We study reachability and termination (reachability of one of the exits with the empty calling context) problems on recursive timed automata. We show that the reachability problem of recursive timed automata is decidable, and EXPTIME-complete, if for every component, either all clocks of that component are passed by reference or none is passed by reference. Moreover, we study reachability games on recursive time automata, where the control state determines which of the two players picks the action to be performed. The objective of one player is reaching a particular subset of the control states, while the objective of the other player is complementary, i.e. avoiding them forever throughout the run of the recursive time automaton. We show that determining the winner of such games is in 2EXPTIME.

*Applications.* Much in the same way as recursive state machines can model *Boolean programs* [4] (or more general software systems using predicate abstraction [16]), it can be argued that recursive timed automata can model *hard real-time software systems* [8]. The need to use the dense semantics of time is more pressing in the case of real-time distributed software systems, i.e., computer programs that run on multiple autonomous computers communicating through computer network. Even after disallowing concurrency, verifying the correctness of real-time distributed software is a fantastic challenge as each participating computer has its own physical clock of varying frequency, while no global clock is available. Under such circumstances it is impossible to model system using discrete semantics of time without knowing the clock frequencies of participating computers. Hence it is natural to study these systems with dense semantics of time.

In [9], the authors study the problem of automatic generation of an optimal controller for an oil pump by defining this model as a 2-player game played on a time automaton. The actual controller used in practice for controlling this oil pump was a 400 lines long C program. Most C programs, apart from the simplest ones, make use of functions and recursive invocations of one function by another. Parameters to such functions are either passed by value or by reference (which in C language is done explicitly by passing a pointer). These kind of controllers operate on variables that are constantly growing in real-time, e.g. total time, oil pressure, temperature etc. The natural model to study correctness of such a system is a game played on recursive timed automaton with a safety critical objective, i.e. the aim for the controller is to avoid a certain set of bad states, while the aim of the other player, the "malicious environment", is trying to reach one of these states. If the controller has a winning strategy, i.e. no matter what how the environment behaves, none of the unsafe states will ever be reached, then the implementation of the controller is correct.

*Related work.* All the work with pushdown timed automata, see e.g. [10,11], has considered only global clocks. Bouajjani, Echahed, and Robbana [5] studied linear hybrid automata with pushdown stack and counters, and showed decidability of reachability in pushdown timed systems. Emmi and Majumdar [13] showed the decidability of language inclusion for implementation as timed pushdown automata and specification as timed automata with one clock; however they proved that it is undecidable when the specification is visibly pushdown timed automata even with one clock [13]. The work on timed automata with counters [7] studies extending time automata with multiple counters. The reachability problem for such systems is already undecidable without clocks, so the authors study several decidable subclasses of this model. Context-Free Timed Systems, studied in [6], are less expressive than our model, and [6] shows decidability of various verification problems for context-free timed systems with linear-hybrid observers (a variable that cannot be used in the constraints used on any edge, similar to prices/cost variables in timed automata).

The paper is organised as follows. In the next section we set definitions of key concepts like labelled transition systems, games, and recursive state machines. In Section 3 we introduce our model and define problems studied in this paper. In Section 4 we prove the undecidability of termination problem and games on the general model, while in Section 5 we discuss decidable subclasses and give complexity results.

## 2 Definitions

### 2.1 Preliminaries

**Notation.** We assume, the sets $\mathbb{N}$ of non-negative integers, $\mathbb{R}$ of reals and $\mathbb{R}_\oplus$ of non-negative reals. For $n \in \mathbb{N}$, let $[\![n]\!]_\mathbb{N}$ and $[\![n]\!]_\mathbb{R}$ denote the sets $\{0, 1, \ldots, n\}$, and $\{r \in \mathbb{R} \mid 0 \leq r \leq n\}$ respectively.

**Labelled Transition System.** A *labelled 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., the set of actions $a \in A$ for which $X(s, a)$ is non-empty.

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 finite run $r{=}\langle s_0, a_1, \ldots, s_n \rangle$ we write *last*$(r){=}s_n$ for the last state of the run. A *strategy* in $\mathcal{L}$ is a function $\sigma$ : *FRuns*$^\mathcal{L} \to A$ such that for all runs $r \in$ *FRuns* we have that $\sigma(r) \in A($*last*$(r))$. We write $\Sigma^\mathcal{L}$ for the set of strategies in $\mathcal{L}$. For a state

$s \in S$ and a strategy $\sigma \in \Sigma^{\mathcal{L}}$, we write $\text{Run}(s, \sigma)$ for the unique run $\langle s_0, a_1, s_1, \ldots \rangle \in$ *Runs*$^{\mathcal{L}}(s)$ such that $s_0 = s$ and for every $i \geq 0$ we have that $\sigma(r_n) = a_{n+1}$, where $r_n = \langle s_0, a_1, \ldots, s_n \rangle$ (here $r_0 = \langle s_0 \rangle$). 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 strategy $\sigma \in \Sigma^{\mathcal{L}}$ such that *Stop*$(F)(\text{Run}(s, \sigma)) < \infty$. A *reachability problem* is to decide whether in a given LTS a final state is reachable from a given initial state.

**Games on Labelled Transition Systems.** A *game arena* $G$ is a tuple $(\mathcal{L}, S_{\text{Ach}}, S_{\text{Tor}})$, where $\mathcal{L} = (S, A, X)$ is an LTS, $S_{\text{Ach}} \subseteq S$ is the set of states controlled by player Achilles, and $S_{\text{Tor}} \subseteq S$ is the set of states controlled by player Tortoise. Moreover, sets $S_{\text{Ach}}$ and $S_{\text{Tor}}$ form a partition of the set $S$.

A strategy of player Achilles is a partial function $\alpha$ : *FRuns*$^{\mathcal{L}} \to A$ such that for a run $r \in$ *FRuns*$^{\mathcal{L}}$ we have that $\alpha(r)$ is defined if *last*$(r) \in S_{\text{Ach}}$, and $\alpha(r) \in A($*last*$(r))$ for every such $r$. A strategy of player Tortoise is defined analogously. Let $\Sigma^{\mathcal{L}}_{\text{Ach}}$ and $\Sigma^{\mathcal{L}}_{\text{Tor}}$ be the set of strategies of player Achilles and Tortoise, respectively. The unique run $\text{Run}(s, \alpha, \tau)$ from a state $s$ when players use strategies $\alpha \in \Sigma^{\mathcal{L}}_{\text{Ach}}$ and $\tau \in \Sigma^{\mathcal{L}}_{\text{Tor}}$ is defined in a straightforward manner.

In a *reachability game* on $G$, rational players Achilles and Tortoise take turns to move a token along the states of $\mathcal{L}$. The decision to choose the successor state is made by the player controlling the current state. The objective of Achilles is to eventually reach certain states, while the objective of Tortoise is to avoid them forever. For an initial state $s$ and a set of final states $F$, the lower value $\underline{\text{Val}}^{\mathcal{L}}_F(s)$ of the reachability game is defined as the upper bound on the number of transitions that Tortoise can ensure before the game visits a state in $F$ irrespective of the strategy of Achilles, and is equal to $\sup_{\tau \in \Sigma^{\mathcal{L}}_{\text{Tor}}} \inf_{\alpha \in \Sigma^{\mathcal{L}}_{\text{Ach}}}$ *Stop*$(F)(\text{Run}(s, \alpha, \tau))$. The concept of upper value is $\overline{\text{Val}}^{\mathcal{L}}_F(s)$ is analogous and defined as $\inf_{\alpha \in \Sigma^{\mathcal{L}}_{\text{Ach}}} \sup_{\tau \in \Sigma^{\mathcal{L}}_{\text{Tor}}}$ *Stop*$(F)(\text{Run}(s, \alpha, \tau))$. If $\underline{\text{Val}}^{\mathcal{L}}_F(s) = \overline{\text{Val}}^{\mathcal{L}}_F(s)$ then we say that the reachability game is determined, or the value $\text{Val}^{\mathcal{L}}_F(s)$ of the reachability game exists and it is such that $\text{Val}^{\mathcal{L}}_F(s) = \underline{\text{Val}}^{\mathcal{L}}_F(s) = \overline{\text{Val}}^{\mathcal{L}}_F(s)$. We say that Achilles wins the reachability game if $\text{Val}^{\mathcal{L}}_F(s) < \infty$. A *reachability game problem* is to decide whether in a given game arena $G$, an initial state $s$ and a set of final states $F$, player Achilles has a strategy to win the reachability game.

## 2.2 Recursive State Machines

*Recursive state machines* (RSMs) generalise LTSs, and can be used to model systems exhibiting recursion and non-deterministic behaviour.

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

- *a finite set* $N_i$ *of* nodes*, including the set* $En_i$ *of* entry nodes *and the (disjoint from* $En_i$*) set* $Ex_i$ *of* exit nodes.
- *a finite set* $B_i$ *of* boxes.

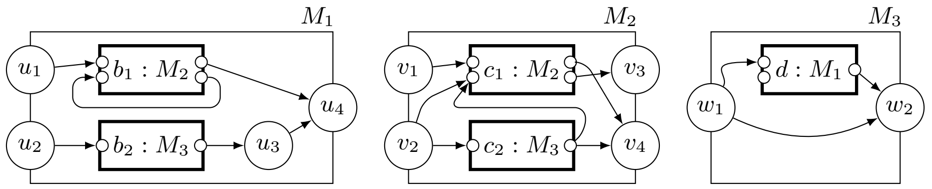

**Fig. 1.** Example recursive state machine taken from [1]

– boxes-to-components mapping $Y_i : B_i \to \{1, 2, \dots, k\}$ *that assigns every box to a component. To each box* $b \in B_i$ *we associate a set of* call ports $Call(b)$*, and a set of return ports* $Ret(b)$*:*

$$Call(b) = \left\{(b, en) \ : \ en \in En_{Y_i(b)}\right\}, \text{ and } Ret(b) = \left\{(b, ex) \ : \ ex \in Ex_{Y_i(b)}\right\}.$$

*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 write* $Q_i = N_i \cup Call_i \cup Ret_i$ *for the union of the set of nodes, call ports and return ports, and we collectively refer to them as the* vertices *of the component* $\mathcal{M}_i$*.*

– *a finite set* $A_i$ *of* actions.

– *a* transition function $X_i : Q_i \times A_i \to Q_i$ *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 $B_1, \dots, B_k$ and set of nodes $N_1, N_2, \dots, N_k$ are mutually disjoint. We use symbols $N, B, A, Q, X$, etc. to denote the union of the corresponding symbols over all components. For example, $N = \cup_{i=1}^{k} N_i$.

*Example 2.* The visual presentation of a finite recursive state machine with three components $M_1, M_2$, and $M_3$ is depicted in Figure 1. Components are shown as thinly framed rectangles with their labels written close to upper right corner, e.g. see component $M_1$. 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 $u_4$). Boxes are shown as thickly framed rectangles inside components labelled $b : M$, where $b$ is the label of the box and $M$ is the component it is mapped to. Call ports of boxes are drawn as small circles on the left of the box, while return ports are on the right. We omit labelling 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.

Intuitively, a run of an RSM starts at one of the entries of its components and proceeds via the edges from one state to another until it reaches an entry port of a box or an exit of the current component. In the former, this box is pushed onto the *stack of pending (recursive) calls* and the run starts from the corresponding entry of the component this box is mapped to. In the latter, 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.

Formally, the semantics of a recursive state machine is given by a discrete LTS, whose states are pairs consisting of a sequence of boxes, called the context, and a vertex.

The context corresponds to the sequence of unreturned component calls, and the vertex is a vertex of the current component.

**Definition 3 (RSM semantics).** *Let $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k)$ 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 countable 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 function 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. $q = ex \in Ex$ and $s' = (\langle \kappa' \rangle, (b, ex))$ where $(b, ex) \in Ret(b)$ and $\kappa = (\kappa', b)$;*
  3. *the vertex $q$ is any other kind of vertex, and $s' = (\langle \kappa \rangle, q')$ and $q' \in X(q, a)$.*

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 $\{(\langle \varepsilon \rangle, ex) \,:\, ex \in Ex\}$.

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 the set of final states $[\![F]\!]$. We also define *termination problem* as the reachability problem of one of the exits with the empty context. Hence, given a recursive state machine $\mathcal{M}$ and an initial node $v$, the termination problem on $\mathcal{M}$ is defined as the reachability problem on LTS $[\![\mathcal{M}]\!]$ with the initial state $(\langle \varepsilon \rangle, v)$ and the set of final states *Term*$_{\mathcal{M}}$. It is easy to show that the reachability problem is at least as hard as the termination problem. We can see this on the example in Figure 1: if we can decide whether state $u_3$ is reachable from $(\langle \varepsilon \rangle, u_2)$, we will also know whether it is possible to terminate from $(\langle \varepsilon \rangle, w_1)$ (simply because it is impossible to reach node $u_3$ from $(\langle \varepsilon \rangle, w_1)$). Hence, all the complexity upper bounds for the reachability problem in this paper apply also to the termination problem, and all the complexity lower bounds for the termination problem apply also to the reachability problem.

*Games on Recursive State Machines.* A partition $(Q_{\text{Ach}}, Q_{\text{Tor}})$ of vertices $Q$ of an RSM $\mathcal{M}$ (between players Achilles and Tortoise) gives rise to recursive game arena $G = (\mathcal{M}, Q_{\text{Ach}}, Q_{\text{Tor}})$. Given an initial state, $v$, and a set of final states, $F$, the reachability game on $\mathcal{M}$ is defined as the reachability game on the game arena $([\![\mathcal{M}]\!], [\![Q_{\text{Ach}}]\!]_{\mathcal{M}}, [\![Q_{\text{Tor}}]\!]_{\mathcal{M}})$ with the initial state $(\langle \varepsilon \rangle, v)$ and the set of final states $[\![F]\!]_{\mathcal{M}}$. Also, the termination game $\mathcal{M}$ is defined as the reachability game on the game arena $([\![\mathcal{M}]\!], [\![Q_{\text{Ach}}]\!]_{\mathcal{M}}, [\![Q_{\text{Tor}}]\!]_{\mathcal{M}})$ with the initial state $(\langle \varepsilon \rangle, v)$ and the set of final states *Term*$_{\mathcal{M}}$. It is a well known result (see, e.g. [22,15]) that reachability games and termination games on RSMs are determined.

*Complexity results for RSMs and their subclasses.* The two most natural subclasses of RSMs are *1-box RSMs* and *1-exit RSMs*. 1-box RSMs are these RSMs that have just a single component and a single box inside of it (this box of course has to be mapped to that single component). On the other hand, an RSM is a 1-exit RSM iff each of its components has just one exit (and hence also all of its boxes), i.e. $Ex_i$ is a singleton for

| # Players | 1-box RSMs | 1-exit RSMs | multi-exit RSMs |
|---|---|---|---|
| 1 | NLOGSPACE-complete [12] | PTIME-complete [3] | PTIME-complete [3] |
| 2 | PSPACE-complete [20,17] | PTIME-complete [22,15] | EXPTIME-complete [22] |

**Table 1.** Complexity results for reachability objective for RSMs

all possible $i$-s. The general class of RSMs is sometimes referred to as *multi-exit RSMs*. Table 1 summarises some key results for RSMs and their subclasses.

The results for 1-box RSMs are derived from the corresponding results for one-counter automata (for their definition, see, e.g. [18]), due to their exact correspondence: the counter value is equal to the number of boxes in the calling context, calling a box results in increasing the counter by 1, while reaching an exit corresponds to decreasing the counter by 1.

## 3 Recursive Timed Automata

Recursive timed automata (RTAs) extend classical timed automata [2] (TAs) with recursion feature similar to RSMs. Instead of defining TAs explicitly, we directly define RTAs whose degenerate case corresponds to TAs. Just as a TA is a finite automaton with a finite set of clocks (continuous variables), a recursive timed automaton is an RSM with a finite set of clocks which can be passed to components during invocation either by value or by reference. Before formally defining the syntax and semantics of RTA we need to introduce the concept of clock valuations, regions and zones.

### 3.1 Clocks, clock valuations, regions and zones.

Let $\mathcal{C}$ be a finite set of *clocks*. In the definition of recursive timed automata (and timed automata [2]) constraints on clocks may appear in the guards on the transitions, where a clock or the difference of two clocks can be compared against natural numbers (in general with rational numbers). Let $K$ be the largest such number. The set of *clock constraints* over $\mathcal{C}$ is the set of conjunctions of *simple constraints*, which are constraints of the form $c \bowtie i$ or $c{-}c' \bowtie i$, where $c, c' \in \mathcal{C}$, $i \in [\![K]\!]_{\mathbb{N}}$, and $\bowtie \in \{<, >, =, \leq, \geq\}$. Let SCC be the finite set of simple clock constraints.

A *clock valuation* on $\mathcal{C}$ is a function $\nu : \mathcal{C} \to \mathbb{R}_{\oplus}$ and we write $V$ for the set of clock valuations. For a clock valuation $\nu \in V$ and delay $t \in \mathbb{R}_{\oplus}$ we write $\nu{+}t$ for the clock valuation defined by $(\nu{+}t)(c) = \nu(c){+}t$, for all $c \in \mathcal{C}$. For a subset of clocks $C \subseteq \mathcal{C}$ and a clock valuation $\nu' \in V$, we write $\nu[C{:=}\nu']$ for the clock valuation where $\nu[C{:=}\nu'](c) = \nu'(c)$ if $c \in C$, and $\nu[C{:=}\nu'](c) = \nu(c)$ otherwise. Clock valuation $\mathbf{0} \in V$ is a special valuation such that $\mathbf{0}(c) = 0$ for all $c \in \mathcal{C}$. Hence, for $C \subseteq \mathcal{C}$, we write $\nu[C{:=}\mathbf{0}]$ for the clock valuation where $\nu[C{:=}\mathbf{0}](c) = 0$ if $c \in C$, and $\nu[C{:=}\mathbf{0}](c) = \nu(c)$ otherwise.

A *clock region* is a maximal set $\zeta \subseteq V$, such that SCC$(\nu)$=SCC$(\nu')$ for all $\nu, \nu' \in \zeta$. We write $\mathcal{R}$ for the finite set of clock regions. Every clock region is an equivalence class of the indistinguishability-by-clock-constraints relation, and vice versa. Note that $\nu$ and $\nu'$ are in the same clock region if and only if the integer parts of the clocks and the partial orders of the clocks, determined by their fractional parts, are the same in $\nu$ and $\nu'$. We write $[\nu]$ for the clock region of $\nu$. For a clock region $\zeta$, a subset of clocks $C \subseteq \mathcal{C}$, and a clock valuation $\nu$, we write $\zeta[C{:=}\nu]$ for the set

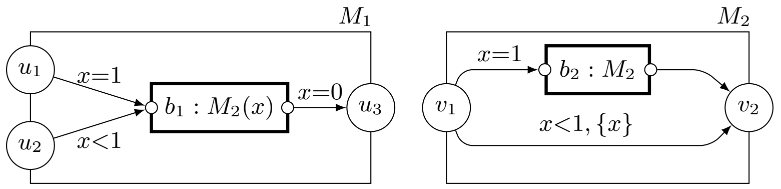

**Fig. 2.** Example recursive timed automaton

$\{[\nu'[C{:=}\nu]] \,:\, \nu' \in \zeta\}$. Observe that if $\nu = \mathbf{0}$ then the set $\zeta[C{:=}\mathbf{0}]$ is a singleton, and we sometimes abuse the notation to write $\zeta[C{:=}\mathbf{0}]$ for the unique region.

A *clock zone* is a convex set of clock valuations, which is a union of a set of clock regions. We write $\mathcal{Z}$ for the set of clock zones. A set of clock valuations is a clock zone if and only if it is definable by a clock constraint.

## 3.2 Syntax

**Definition 4 (Syntax).** *A* recursive timed automaton $\mathcal{T} = (\mathcal{C}, (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k))$ *is a pair made of a set of clocks $\mathcal{C}$ and a collection of components $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k)$. Each component $\mathcal{T}_i = (N_i, En_i, Ex_i, B_i, Y_i, A_i, X_i, P_i, Inv_i, E_i, \rho_i)$ consists of:*

- *a finite set $N_i$ of* nodes*, including the set $En_i$ of entry nodes and the (disjoint from $En_i$) set $Ex_i$ of exit nodes;*
- *a finite set $B_i$ of* boxes*;*
- *boxes-to-components mapping $Y_i : B_i \to \{1, 2, \ldots, k\}$ 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 $\mathcal{T}_i$ 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 vertices of $\mathcal{T}_i$.)*
- *a finite set $A_i$ of* actions*;*
- *the* transition function $X_i : Q_i{\times}A_i \to Q_i$ *is with the condition that call ports and exit nodes do not have any outgoing transitions;*
- *pass-by-value mapping $P_i : B_i \to 2^{\mathcal{C}}$ 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.)*
- *the* invariant condition $Inv_i : Q_i \to \mathcal{Z}$*;*
- *the* action enabledness function $E_i : Q_i{\times}A_i \to \mathcal{Z}$*; and*
- *the* clock reset function $\rho_i : A_i \to 2^{\mathcal{C}}$.

We assume that the sets of boxes, nodes, etc. are mutually disjoint and we use symbols $(N, B, Y, Q, P, X$, etc.) without a subscript, to denote the union of the corresponding objects over all components. When we consider an RTA as an input of an algorithm, its size should be understood as the sum of the sizes of encodings of $Q$, $\mathcal{C}$, $Inv$, $A$, $E$, and $X$. Analogously as for RSMs, we define special subclasses of RTAs: *1-exit RTAs*, for which each component is allowed to have just one exit, and *1-box RTAs*, that just consist of a single component with a single box inside of them.

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 $P(b) = \mathcal{C}$ or $P(b) = \emptyset$. Any general recursive timed automaton with one clock is trivially glitch-free.

*Example 5.* The visual presentation of a recursive timed automaton with two components $M_1$ and $M_2$, and one clock $x$ is shown in Figure 2. The visual representation is similar to that in RSMs. However, each transition is labelled with a guard and the clocks to be reset, (e.g. transition from node $v_1$ to $v_2$ can be taken only when clock $x{<}1$, and after taking this transition, clock $x$ is reset), and a box is labelled as $b : M(C)$ to denote that box $b$ is mapped to $M$ and all the clocks in the set $C$ are passed by value, and the rest of the clocks are passed by reference. When the set $C$ is empty, we just write $b : M$ for $b : M(\emptyset)$.

## 3.3 Semantics

A *configuration* of an RTA $\mathcal{T}$ is a tuple $(\langle\kappa\rangle, q, \nu)$, where $\kappa \in (B \times V)^*$ is (possibly empty) sequence of pairs of boxes and clock valuations, $q \in Q$ is a vertex and $\nu \in V$ is a clock valuation over $\mathcal{C}$ such that $\nu \in Inv(q)$. The sequence $\langle\kappa\rangle \in (B \times V)^*$ denotes the stack of pending recursive calls and the valuation of all the clocks 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 clocks passed by value, because other clocks 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 behaviour 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 $q' \in \delta(q, a)$ and $\nu' = (\nu + t)[\rho(a) := \mathbf{0}]$. 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 $\mathcal{T} = (\mathcal{C}, (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k))$ be an RTA where each component is of the form $\mathcal{T}_i = (N_i, En_i, Ex_i, B_i, Y_i, A_i, X_i, P_i, Inv_i, E_i, \rho_i)$. The semantics of $\mathcal{T}$ is a labelled transition system $[\![\mathcal{T}]\!] = (S_\mathcal{T}, A_\mathcal{T}, X_\mathcal{T})$ where:*

- *$S_\mathcal{T} \subseteq (B \times V)^* \times Q \times V$, the set of states, is such that $(\langle\kappa\rangle, q, \nu) \in S_\mathcal{T}$ if $\nu \in Inv(q)$.*
- *$A_\mathcal{T} = \mathbb{R}_\oplus \times A$ is the set of* timed actions*;*
- *$X_\mathcal{T} : S_\mathcal{T} \times A_\mathcal{T} \to S_\mathcal{T}$ is the transition function such that for $(\langle\kappa\rangle, q, \nu) \in S_\mathcal{T}$ and $(t, a) \in A_\mathcal{T}$, we have $(\langle\kappa'\rangle, q', \nu') = X_\mathcal{T}((\langle\kappa\rangle, q, \nu), (t, a))$ if and only if the following condition holds:*
  1. *if the vertex $q$ 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 vertex $q$ 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 vertex $q$ is any other kind of vertex, then $\nu{+}t' \in Inv(q)$ for all $t' \in [0, t]$; $\nu{+}t \in E(q, a)$; and $\langle\kappa'\rangle = \langle\kappa\rangle$, $q' \in X(q, a)$, and $\nu' = (\nu + t)[\rho(a) := \mathbf{0}]$.*

### 3.4 Reachability (Termination) Problems and Games

For a subset $Q' \subseteq Q$ of states of recursive timed automaton $\mathcal{T}$ we define the set $[\![Q']\!]_{\mathcal{T}}$ as the set $\{(\langle\kappa\rangle, q, \nu) \in S_{\mathcal{T}} \,:\, q \in Q'\}$. We also define the set of terminal configuration $Term_{\mathcal{T}}$ as the set $Term_{\mathcal{T}} = \{(\langle\varepsilon\rangle, q, \nu) \in S_{\mathcal{T}} \,:\, q \in Ex\}$.

Given a recursive timed automaton $\mathcal{T}$, an initial node $q$ and valuation $\nu \in V$, and a set of *final vertices* $F \subseteq Q$, the *reachability problem* on $\mathcal{T}$ is defined as the reachability problem on LTS $[\![\mathcal{T}]\!]$ with the initial state $(\langle\varepsilon\rangle, q, \nu)$ and the set of final states $[\![F]\!]_{\mathcal{T}}$. As with RSMs, we also define *termination problem* as reachability of one of the exits with the empty context. Hence, given an RTA $\mathcal{T}$ and an initial node $q$ and a valuation $\nu \in V$, the termination problem on $\mathcal{T}$ is defined as the reachability problem on LTS $[\![\mathcal{T}]\!]$ with the initial state $(\langle\varepsilon\rangle, q, \nu)$ and the set of final states $Term_{\mathcal{T}}$.

*Example 7.* Consider the RTA shown in Figure 2. From the vertex $u_1$ of $M_1$ there is no path that visits the exit node $u_3$ with the empty calling context, as the only transition available form $u_1$ is to wait until clock $x = 1$, and then invoking component $M_2$ which recursively calls itself forever if the value of clock $x = 1$. On the other hand, from node $u_2$ there are infinitely many paths that reach $u_3$ with the empty context. Notice that termination at $u_3$ is possible only when delay at $u_2$ is 0 time-units, as upon exiting box $b$ clock $x$ is tested against 0. Since clock $x$ was passed by value to component $M_2$, the current value of clock $x$ is the one before the invocation of $M_2$, and hence the clock reset inside $M_2$ does not help.

A partition $(Q_{\text{Ach}}, Q_{\text{Tor}})$ of vertices $Q$ of an RTA $\mathcal{T}$ gives rise to a recursive timed game arena $\Gamma = (\mathcal{T}, Q_{\text{Ach}}, Q_{\text{Tor}})$. Given an initial vertex $q$, a valuation $\nu \in V$ and a set of final states $F$, the reachability game on $\Gamma$ is defined as the reachability game on the game arena $([\![\mathcal{T}]\!], [\![Q_{\text{Ach}}]\!]_{\mathcal{T}}, [\![Q_{\text{Tor}}]\!]_{\mathcal{T}})$ with the initial state $(\langle\varepsilon\rangle, (q, \nu))$ and the set of final states $[\![F]\!]_{\mathcal{T}}$. Also, termination game on $\mathcal{T}$ is defined as the reachability game on the game arena $([\![\mathcal{T}]\!], [\![Q_{\text{Ach}}]\!]_{\mathcal{T}}, [\![Q_{\text{Tor}}]\!]_{\mathcal{T}})$ with the initial state $(\langle\varepsilon\rangle, (q, \nu))$ and the set of final states $Term_{\mathcal{T}}$.

## 4 Undecidability Results

The following is one of the key results of this paper.

**Theorem 8.** *Termination problem is undecidable for recursive timed automata with at least three clocks. Moreover, termination game problem is undecidable for recursive timed automata with at least two clocks.*

For the undecidability proofs we use reduction from the halting problem of two-counter *Minsky machines* [19]. A Minsky machine $\mathcal{A}$ is a tuple $(L, C, D)$ where: $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$ is the set of states including the distinguished terminal state $\ell_n$; $C = \{c_1, c_2\}$ is the set of two *counters*; $D = \{\delta_0, \delta_1, \ldots, \delta_{n-1}\}$ is the set of transitions of the following type:

1. (increment $c$) $\delta_i : c := c + 1$; goto $\ell_k$,
2. (test-and-decrement $c$) $\delta_i$ : if $(c > 0)$ then $(c := c - 1$; goto $\ell_k)$ else goto $\ell_m$,

where $c \in C$, $\delta_i \in D$ and $\ell_k, \ell_m \in L$.

A configuration of a Minsky machine is a tuple $(\ell, c, d)$ where $\ell \in L$ and $c, d$ are natural numbers that specify the value of counters $c_1$ and $c_2$, respectively. The initial configuration is $(\ell_0, 0, 0)$. A run of Minsky machine is a (finite or infinite) sequence of configurations $\langle s_0, s_1, \ldots\rangle$ where $s_0$ is the initial configuration, and the relation between subsequent configurations is governed by transitions at their respective states. The run is a finite sequence if and only if the last configuration is the terminal state $\ell_n$. Note that a Minsky machine has only one run starting from the initial configuration. *Termination problem* for a Minsky machine asks whether its unique run is finite. It is well known ([19]) that the termination problem for a two-counter Minsky machine is undecidable. In the rest of the section we show a reduction from the halting problem of Minsky machines to the termination games on RTA with two clocks. The reduction to the termination problem for (1-player) RTAs with three clocks is in the technical report version of this paper [21].

We fix the clocks set $\mathcal{C} = \{x, y\}$, and we describe the construction of the central component $\mathsf{HALT}^{\mathcal{A}}$ with nodes $\ell_0, \ell_1, \ldots, \ell_n$ with the entry node $\ell_0$ and the exit node $\ell_n$. A configuration $(\ell_i, c, d)$ of a Minsky machine corresponds to the configuration $(\langle\varepsilon\rangle, \ell_i, \nu)$ such that $\nu(x) = 2^{-c} \cdot 3^{-d}$ and $\nu(y) = 0$. Decrementing and incrementing counter $c$ is simulated by doubling and halving, resp., of the clock $x$, while decrementing and incrementing the counter $d$ is simulated by tripling, and *thirding*[1], resp., the value of clock $x$. Testing counter $c$ (resp. $d$) against 0 can be simulated by multiplying clock $x$ by some power of 3 (resp., 2) and then comparing it against 3 (resp. 2). The components for doubling ($\mathsf{DB}$) and halving ($\mathsf{HF}$) the value of clock $x$, and testing whether the value of clock $x$ is of the form $2^{-i}$ or $3^{-i}$ ($\mathsf{P2O}$ or $\mathsf{P3O}$, resp.) are given in Figure 3. Due to space constraints, we omit the description of components for tripling, $\mathsf{TR}$, and thirding, $\mathsf{TH}$, of clock $x$. However such components are very similar to components $\mathsf{DB}$ and $\mathsf{HF}$. All components function as intended only when upon entering them, the value of clock $y$ is 0. The vertices of these components are partitioned between Achilles and Tortoise: the only vertex controlled by Tortoise (shown as black squares) is the return port $(B_8, ex_9)$ in component $M_8$. The component $\mathsf{DB}'$, invoked from inside the component $M_8$, is similar to gadget $\mathsf{DB}$, however it doubles the value of clock $y$, while assuming that clock $x$ is set to 0. We assume that the node labelled $\ddot{\frown}$ has no outgoing transitions. The behaviour of components $\mathsf{P2O}$ and $\mathsf{P3O}$ is as follows: if clock $x$ is of the form $2^{-i}$ and $3^{-i}$, resp., a run starting at that component's entry will terminate at its bottom exit and if clock x is not of that form then such a run will terminate at that component's top exit. Since the precise construction of $\mathsf{HALT}^{\mathcal{A}}$ is straightforward, we just present in Figure 3 a schema that simulates the test-and-decrement $c$ operation: $\delta_i$ : if $(c > 0)$ then $(c := c - 1$; goto $\ell_k)$ else goto $\ell_m$. Whenever a run reaches node $\ell_i$ inside $\mathsf{HALT}^{\mathcal{A}}$, a box mapped to $\mathsf{P3O}$ is called that tests whether the value of counter $c$ is zero. After returning from $\mathsf{P3O}$ both clocks are restored and the exit port indicates whether clock $c$ is zero or not. If clock $c$ is zero then the run proceeds straight to node $\ell_m$; otherwise the value of the counter $c$ is decremented by $1$ by multiplying clock $x$ by two using component $\mathsf{DB}$ and the run proceeds to node $\ell_k$. It should be easy to see now how to encode the increment $c$ operation and how to combine them all into the $\mathsf{HALT}^{\mathcal{A}}$ component.

[1] dividing by three

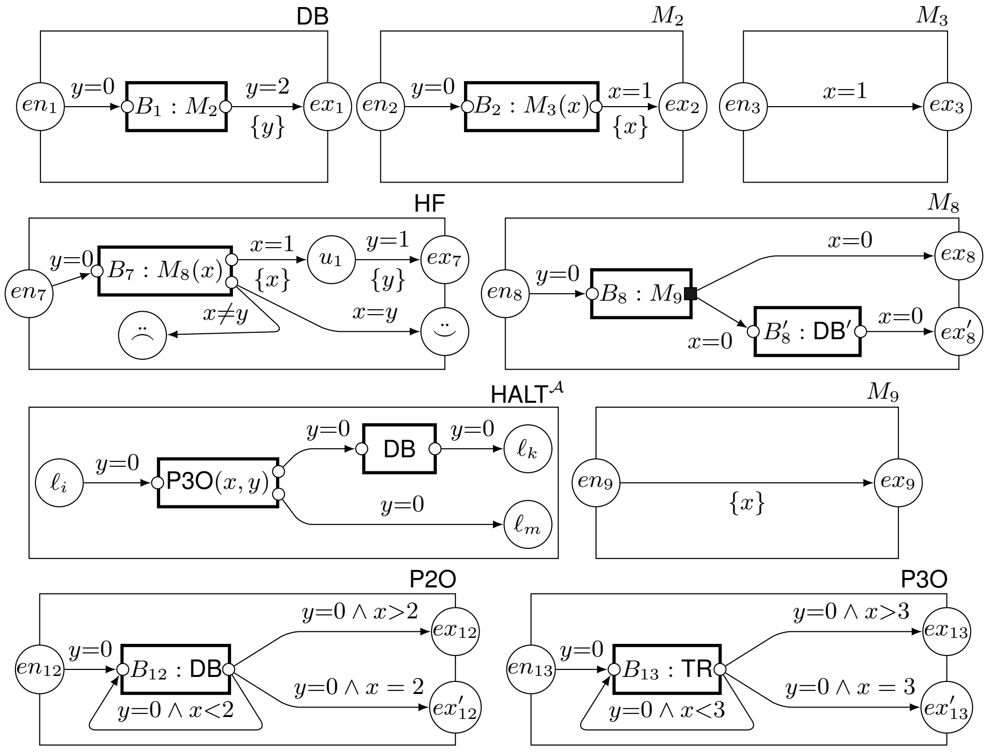

**Fig. 3.** Components for doubling DB and halving HF the value of clock $x$, and checking whether $x$ is of the form $2^{-i}$ and $3^{-i}$.

To make the proofs more comprehensible, we show a run in an RTA using three different forms of transitions $s \xrightarrow{g,C}_t s'$, $s \rightsquigarrow s'$, and $s \xrightarrow{M(C)}_* s'$ defined in the following way.

1. The transitions of form $s \xrightarrow{g,C}_t s'$, where $s = (\langle\kappa\rangle, n, \nu), s' = (\langle\kappa\rangle, n', \nu')$ are configurations of an RTA, $g$ is a clock constraint, $C$ is a set of clocks, and $t$ is a real number, holds if there is a transition in the RTA from vertex $n$ to $n'$ with guard $g$ and clock reset set $C$, moreover $\nu' = (\nu + t)[C := \mathbf{0}]$.
2. The transitions of the form $s \rightsquigarrow s'$, where $s = (\langle\kappa\rangle, n, \nu), s' = (\langle\kappa'\rangle, n', \nu')$, correspond to the following cases:
   (a) transitions from a call port to an entry node, i.e. , $n = (b, en)$ for some box $b \in B$ and $\kappa' = \langle\kappa, (b, \nu)\rangle$ and $n' = en \in En$, while $\nu' = \nu$.
   (b) transition from an exit node to a return port (which also restore the value of clocks passed by value), i.e. $\langle\kappa\rangle = \langle\kappa'', (b, \nu'')\rangle$, $n = ex \in Ex$, and $n' = (b, ex) \in Ret(b)$ and $\kappa' = \kappa''$, while $\nu' = \nu[P(b) = \nu'']$.
3. The transitions of the form $s \xrightarrow{M(C)}_* s'$, called *summary edges*, where $s = (\langle\kappa\rangle, n, \nu)), s' = (\langle\kappa'\rangle, n', \nu')$ are such that $n = (b, en)$ and $n' = (b, ex)$ are call and return ports, resp., of a box $b$ mapped to $M$ which passes by value to $M$ the clocks in $C$.

For the sake of simplicity, in this section instead of presenting the context information in the form $(b, \nu) \in B \times V$, we write $(b, (\nu(x), \nu(y)))$ if some clock is passed by value to $b$, else we just write $b$. We also write a configuration $(\langle\kappa\rangle, n, \nu)$ as $(\langle\kappa\rangle, n, (\nu(x), \nu(y)))$.

**Proposition 9.** *For any context $\kappa \in (B \times V)^*$, any box $b \in B$, and $x_0 \in [0, 1]$ we have that $(\langle\kappa\rangle, (b, en_1), (x_0, 0)) \xrightarrow{\mathsf{DB}}_* (\langle\kappa\rangle, (b, ex_1), (2 \cdot x_0, 0))$.*

*Proof.* Component $\mathsf{DB}$, shown in Figure 3, uses components $M_2$ and $M_3$. The following is the unique run starting from the configuration $(\langle\kappa\rangle, (b, en_1), (x_0, 0))$ terminating at the configuration $(\langle\kappa\rangle, (b, ex_1), (2 \cdot x_0, 0))$.

$$\begin{aligned}
& (\langle\kappa\rangle, (b, en_1), (x_0, 0)) \rightsquigarrow (\langle\kappa, b\rangle, en_1, (x_0, 0)) \\
\xrightarrow{y=0}_0 \;& (\langle\kappa, b\rangle, (B_1, en_2), (x_0, 0)) \rightsquigarrow (\langle\kappa, b, B_1\rangle, en_2, (x_0, 0)) \\
\xrightarrow{y=0}_0 \;& (\langle\kappa, b, B_1\rangle, (B_2, en_3), (x_0, 0)) \rightsquigarrow (\langle\kappa, b, B_1, B_2(x_0, 0)\rangle, en_3, (x_0, 0)) \\
\xrightarrow{x=1}_{(1-x_0)} \;& (\langle\kappa, b, B_1, B_2(x_0, 0)\rangle, ex_3, (1, 1 - x_0)) \rightsquigarrow (\langle\kappa, b, B_1\rangle, (B_2, ex_3), (x_0, 1 - x_0)) \\
\xrightarrow{x=1,\{x\}}_{(1-x_0)} \;& (\langle\kappa, b, B_1\rangle, ex_2, (0, 2 - 2 \cdot x_0)) \rightsquigarrow (\langle\kappa, b\rangle, (B_1, ex_2), (0, 2 - 2 \cdot x_0)) \\
\xrightarrow{y=2,\{y\}}_{(2 \cdot x_0)} \;& (\langle\kappa, b\rangle, ex_1, (2 \cdot x_0, 0)).
\end{aligned}$$

The intermediate steps of this sequence of transitions can be easily verified. □

**Proposition 10.** *For any context $\kappa \in (B \times V)^*$, any box $b \in B$, and $x_0 \in [0, 1]$, there exists a unique strategy of Achilles such that*

$$\begin{aligned}
\textit{either } (\langle\kappa\rangle, (b, en_7), (x_0, 0)) &\xrightarrow{\mathsf{HF}}_* (\langle\kappa\rangle, (b, ex_7), (\frac{x_0}{2}, 0)), \\
\textit{or } (\langle\kappa\rangle, (b, en_7), (x_0, 0)) &\xrightarrow{\mathsf{HF}}_* (\langle\kappa\rangle, (b, \ddot\smile), (x_0, x_0))).
\end{aligned}$$

*Moreover, for other strategies of Achilles there exists a strategy of Tortoise such that component $\mathsf{HF}$ does not terminate.*

*Proof.* The main observation here is that, in component $\mathsf{HF}$, starting from the configuration $(\langle\kappa\rangle, (b, en_7), (x_0, 0))$ Achilles has a strategy to terminate only if he chooses to delay the time by $\frac{x_0}{2}$ in component $M_9$ (called via box $B_8$). The evolution of the run from $(\langle\kappa\rangle, (b, en_7), (x_0, 0))$ to $(\langle\kappa, b, B_7(x_0, 0), B_8\rangle, en_9, (x_0, 0))$ is straightforward. Now, in component $M_9$ Achilles can wait for an arbitrary amount of time before taking a transition to $ex_9$ and resetting clock $x$. Let us assume that he waits for $t$ time units, and hence $(\langle\kappa, b, B_7(x_0, 0)\rangle, (B_8, ex_9), (0, t))$ is reached which is controlled by Tortoise. Now Tortoise has a choice between making a transition to $ex_8$ (believing that $t = \frac{x_0}{2}$) or invoking the component $B'_8$ (when suspecting that $t \neq \frac{x_0}{2}$).

If Tortoise believes that $t = \frac{x_0}{2}$ then he makes a transition to $ex_8$ and thus the system reaches the configuration $(\langle\kappa, b\rangle, (B_7, ex_8), (x_0, t))$ giving rise to the following run:

$$\begin{aligned}
(\langle\kappa, b\rangle, (B_7, ex_8), (x_0, t)) &\xrightarrow{x=1,\{x\}}_{(1-x_0)} (\langle\kappa, b\rangle, u_1, (0, 1 - x_0 + t)) \\
&\xrightarrow{y=1,\{y\}}_{(x_0-t)} (\langle\kappa, b\rangle, ex_7, (x_0 - t, 0)) \rightsquigarrow (\langle\kappa\rangle, (b, ex_7), (x_0 - t, 0)).
\end{aligned}$$

Hence if $t = \frac{x_0}{2}$ then the run terminates at configuration $(\langle\varepsilon\rangle, ex_7, (\frac{x_0}{2}, 0))$.

On the other hand if Tortoise believes that $t \neq \frac{x_0}{2}$, then he invokes the component $\mathsf{DB}'$ to double the value of clock $y$ (while keeping the value of clock $x$ equal to 0), and makes a transition, via exit $ex'_8$, to the configuration $(\langle\kappa, b, B_7(x_0, 0)\rangle, ex'_8, (0, 2 \cdot t))$. Since $x_0$ was passed by value, it is restored upon exiting from box $B_7$ and the configuration reached is $(\langle\kappa, b\rangle, (B_7, ex'_8), (x_0, 2 \cdot t))$. If Tortoise's suspicion was right and $t \neq \frac{x_0}{2}$ then the only transition available to Achilles is to move to the ☹ vertex which never terminates. Otherwise Achilles can only move to configuration $(\langle\kappa\rangle, (b, ☺), (x_0, x_0))$ and terminate. Hence, it is clear that the only winning strategy for Achilles is to choose $t = \frac{x_0}{2}$. □

**Proposition 11.** *For any context $\kappa \in (B \times V)^*$, any box $b \in B$, and $x_0 \in [0, 1]$ we have that starting from configuration $(\langle\kappa\rangle, (b, en_{12}), (x_0, 0))$ the component $\mathsf{P2O}$ terminates at $(\langle\kappa\rangle, (b, ex'_{12}), (x_0, 0))$ only when $x_0 = 2^{-i}$ for some $i \in \mathbb{N}$ and otherwise it terminates at $(\langle\kappa\rangle, (b, ex_{12}), (x_0, 0))$.*

From Propositions 9, 10, and 11 (and similar results related to other components) it follows that Achilles has a strategy to terminate at $\ell_n$ in component $\mathsf{HALT}^{\mathcal{A}}$ if and only if the Minsky machine $\mathcal{A}$ terminates.

## 5 Decidability Results

### 5.1 Region Abstraction

For every RTA $\mathcal{T}$ we define regional equivalence relation $\mathcal{E}_R \subseteq S_\mathcal{T} \times S_\mathcal{T}$ in the following way: For configurations $s = (\langle\kappa\rangle, q, \nu)$ and $s' = (\langle\kappa'\rangle, q', \nu')$ we have that $s, s' \in \mathcal{E}_R$, or equivalently we write $[s] = [s']$, if $q = q'$, $[\nu] = [\nu']$, and $\kappa = (b_1, \nu_1), (b_2, \nu_2), \ldots, (b_n, \nu_n)$ and $\kappa' = (b'_1, \nu'_1), (b'_2, \nu'_2), \ldots, (b'_n, \nu'_n)$ are such that for every $1 \leq i \leq n$ we have $[\nu_i] = [\nu'_i]$ and $b_i = b'_i$.

A relation $B \subseteq S_\mathcal{T} \times S_\mathcal{T}$ defined over the set of configurations $S_\mathcal{T}$ of a recursive timed automaton is a *time-abstract bisimulation* if for every pair of configurations $s_1, s_2 \in S_\mathcal{T}$ such that $(s_1, s_2) \in B$, for every timed action $(t, a) \in A_\mathcal{T}$ such that $X_\mathcal{T}(s_1, (t, a)) = s'_1$, there exists a timed action $(t', a) \in A_\mathcal{T}$ such that $X_\mathcal{T}(s_2, (t', a)) = s'_2$ and $(s'_1, s'_2) \in B$.

**Proposition 12.** *Regional equivalence relation for glitch-free recursive timed automata is a time-abstract bisimulation.*

*Proof.* Let us fix configurations $s = (\langle\kappa\rangle, q, \nu)$ and $s' = (\langle\kappa'\rangle, q', \nu')$ such that $[s] = [s']$, timed action $(t, a) \in X_\mathcal{T}$ such that $X_\mathcal{T}(s, (t, a)) = s_a (= (\kappa_a, (q_a, \nu_a)))$. We need to find $(t', a)$ such that $X_\mathcal{T}(s', (t', a)) = s'_a (= (\kappa'_a, (q'_a, \nu'_a)))$ and $[s_a] = [s'_a]$. There are following three cases.

1. The vertex $q$ is a call port, i.e. $q = (b, en) \in \mathit{Call}$. In this case $t = 0$, the context $\langle\kappa_a\rangle = \langle\kappa, (b, \nu)\rangle$, $q_a = en$, and $\nu_a = \nu$. Since $q' = q (= (b, en))$ is then also a call port, we have that $t' = 0$, and $\langle\kappa'_a\rangle = \langle\kappa', (b, \nu')\rangle$, $q'_a = en$, and $\nu'_a = \nu_a$. It is trivial to show that $[s_a] = [s'_a]$.
2. The vertex $q$ is an exit node, i.e. $q = ex \in \mathit{Ex}$, and let $\langle\kappa\rangle = \langle\kappa_*, (b, \nu_*)\rangle$ and $(b, ex) \in \mathit{Ret}(b)$. In this case $t = 0$; context $\langle\kappa_a\rangle = \langle\kappa_*\rangle$; $q_a = (b, ex)$; and

$\nu_a = \nu[P(b){:=}\nu_*]$. Let the context $\langle\kappa'\rangle$ be $\langle\kappa'_*, (b, \nu'_*)\rangle$. Since again $q' = q(= ex)$ is also an exit node we have that $t' = 0$, $\langle\kappa'_a\rangle = \langle\kappa'_*\rangle$ and $\nu'_a = \nu'[P(b){:=}\nu'_*]$. We need to show that $[\nu_a] = [\nu'_a]$. Notice that for glitch-free RTAs there are exactly two cases to consider:

- $P(b) = \mathcal{C}$. In this case $\nu_a = \nu_*$ and $\nu'_a = \nu'_*$, and since $[\nu_*] = [\nu'_*]$ we get that $[\nu_a] = [\nu'_a]$.
- $P(b) = \emptyset$. In this case $\nu_a = \nu$ and $\nu'_a = \nu'$, and since $[\nu] = [\nu']$ we get that $[\nu_a] = [\nu'_a]$.

3. if vertex $q$ is of any other kind, then the result follows by classical region equivalence relation.

The proof is now complete. □

The following proposition follows from the 2nd case in the proof of Proposition 12.

**Proposition 13.** *For general (non glitch-free) RTA with two clocks the successors of regionally equivalent configurations are not necessarily regionally equivalent.*

By using two boxes mapped to $\mathsf{DB}$ in a sequence, one is able to construct a new component $D_1$ that multiplies the value of clock $x$ by $2 \cdot 2 = 2^{2^0} \cdot 2^{2^0} = 2^{2^1} = 4$. (See, e.g. how component $\mathsf{DB}$ is exploited in component $\mathsf{P2O}$ in Figure 3.) In general, by using two boxes mapped to $D_i$, one is able to construct a new component $D_{i+1}$ that multiplies the value of clock $x$ by $2^{2^i} \cdot 2^{2^i} = 2^{2^{i+1}}$. So, to solve reachability problem for general RTA with two clocks, one needs to consider doubly-exponentially many (in the size of the RTA) partitions of a region.

Proposition 12 allows us to extend the concept of region abstraction in the setting of glitch-free RTA. Before we introduce the abstraction, we need to define some notations.

For $\zeta, \zeta' \in \mathcal{R}$, we say that clock region $\zeta'$ is in the future of clock region $\zeta$, or that $\zeta$ is in the past of $\zeta'$, if there are $\nu \in \zeta$, $\nu' \in \zeta'$ and delay $d \in \mathbb{R}_\oplus$ such that $\nu' = \nu{+}d$; we then write $\zeta \rightarrow_* \zeta'$. We say that $\zeta'$ is the *time successor* of $\zeta$ if $\zeta \rightarrow_* \zeta'$, $\zeta{\neq}\zeta'$, and $\zeta \rightarrow_* \zeta'' \rightarrow_* \zeta'$ implies $\zeta''{=}\zeta$ or $\zeta''{=}\zeta'$ and write $\zeta \rightarrow \zeta'$ and $\zeta' \leftarrow \zeta$. Time successor definition is extended to $n$-th time successor in a natural way: we say that $\zeta'$ is the $n$-th successor of $\zeta$, and write $\zeta \rightarrow_{+n} \zeta'$, if there is a sequence of regions $\langle\zeta_1, \zeta_2, \ldots, \zeta_n\rangle$ such that $\zeta_1{=}\zeta$, $\zeta_n{=}\zeta'$ and $\zeta_i \rightarrow \zeta_{i+1}$ for every $1{\le}i{<}n$. In this case we also write $[\zeta_1, \zeta_n]$ for the union of regions $\zeta_1, \ldots \zeta_n$.

**Definition 14 (Region Abstraction).** *Let $\mathcal{T} = (\mathcal{C}, (\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k))$ be a glitch-free RTA, where each $\mathcal{T}_i$ is the tuple $(N_i, En_i, Ex_i, B_i, Y_i, A_i, X_i, P_i, Inv_i, E_i, \rho_i)$. The region abstraction of $\mathcal{T}$ is a* finite *RSM $\mathcal{T}^{\text{RG}} = (\mathcal{T}_1^{\text{RG}}, \mathcal{T}_2^{\text{RG}}, \ldots, \mathcal{T}_k^{\text{RG}})$ where for each $1 \le i \le k$, component $\mathcal{T}_i^{\text{RG}} = (N_i^{\text{RG}}, En_i^{\text{RG}}, Ex_i^{\text{RG}}, B_i^{\text{RG}}, Y_i^{\text{RG}}, A_i^{\text{RG}}, X_i^{\text{RG}})$ consists of :*

- *a finite set $N_i^{\text{RG}} \subseteq (N_i \times \mathcal{R})$ of* nodes *such that $(n, \zeta) \in N_i^{\text{RG}}$ if $\zeta \in Inv(n)$. Moreover, $N_i^{\text{RG}}$ includes the sets of entry nodes $En_i^{\text{RG}} \subseteq En_i \times \mathcal{R}$ and exit nodes $Ex_i^{\text{RG}} \subseteq Ex_i \times \mathcal{R}$;*
- *a finite set $B_i^{\text{RG}} = B_i \times \mathcal{R}$ of* boxes*;*
- boxes-to-components mapping $Y_i^{\text{RG}} : B_i^{\text{RG}} \rightarrow \{1, 2, \ldots, k\}$ *is such that* $Y_i^{\text{RG}}(b, \zeta) = Y_i(b)$. *To each box $(b, \zeta) \in B_i^{\text{RG}}$ we associate a set of call ports $Call^{\text{RG}}(b, \zeta)$, and*

*a set of return ports* $Ret^{\text{RG}}(b,\zeta)$*:*

$$Call^{\text{RG}}(b,\zeta) = \left\{(((b,\zeta),en),\zeta') \;:\; \zeta' \in \mathcal{R} \text{ and } en \in En_{Y_i(b)}\right\}, \text{ and}$$
$$Ret^{\text{RG}}(b,\zeta) = \left\{(((b,\zeta),ex),\zeta') \;:\; \zeta' \in \mathcal{R} \text{ and } ex \in Ex_{Y_i(b)}\right\}.$$

*Let* $Call_i^{\text{RG}}$ *and* $Ret_i^{\text{RG}}$ *be the set of call and return ports of component* $\mathcal{T}_i^{\text{RG}}$*. We write* $Q_i^{\text{RG}} = N_i^{\text{RG}} \cup Call_i^{\text{RG}} \cup Ret_i^{\text{RG}}$ *for the* vertices *of the component* $\mathcal{T}_i^{\text{RG}}$*.*

- $A_i^{\text{RG}} \subseteq \mathbb{N}\times A_i$ *is the set of actions, such that if* $(h,a) \in A_i^{\text{RG}}$ *(here* $h$ *is number of region hops before taking* $a$*) then* $h \leq (2\cdot|\mathcal{C}|)^K$*, where* $K \in \mathbb{N}$ *is the largest constant that appears in one of the clock constraints in* $E$ *or* $Inv$*;*
- *a* transition function $X_i^{\text{RG}} : Q_i^{\text{RG}}\times A_i^{\text{RG}} \to Q_i^{\text{RG}}$ *with the natural condition that call ports and exit nodes do not have any outgoing transitions. Moreover, for* $q,q' \in Q_i^{\text{RG}}$*,* $(h,a) \in A_i^{\text{RG}}$ *we have that* $q' = X_i^{\text{RG}}(q,(h,a))$ *if one of the following conditions holds:*
  1. $q = (n,\zeta) \in N_i^{\text{RG}}$*, there exists a region* $\zeta_a$ *such that* $\zeta \to_{+h} \zeta_a$*,* $[\zeta,\zeta_a] \subseteq Inv_i(n)$*,* $\zeta_a \in E_i(n,a)$*, and*
     - *if* $q' = (n',\zeta')$ *then* $\zeta' = \zeta_a[\rho_i(a) := \mathbf{0}]$ *and* $X_i(n,a) = n'$*.*
     - *if* $q' = (((b,\zeta'),en),\zeta'')$ *then* $\zeta' = \zeta'' = \zeta_a[\rho_i(a) := \mathbf{0}]$ *and* $X_i(n,a) = (b,en)$*.*
  2. $q = (((b,\zeta_{\text{Saved}}),ex),\zeta_{\text{Curr}})$ *is a return port of* $\mathcal{T}_i^{\text{RG}}$*. Let* $\zeta = \zeta_{\text{Saved}}$ *if* $P_i(b) = \mathcal{C}$ *and* $\zeta = \zeta_{\text{Curr}}$ *otherwise. There exists a region* $\zeta_a$ *such that* $\zeta \to_{+h} \zeta_a$*, and* $[\zeta,\zeta_a] \subseteq Inv_i((b,ex))$*,* $\zeta_a \in E_i((b,ex),a)$*, and*
     - *if* $q' = (n',\zeta')$ *then* $\zeta' = \zeta_a[\rho_i(a) := \mathbf{0}]$ *and* $X_i(n,a) = n'$*.*
     - *if* $q' = (((b,\zeta'),en),\zeta'')$ *then* $\zeta' = \zeta'' = \zeta_a[\rho_i(a) := \mathbf{0}]$ *and* $X_i(n,a) = (b,en)$*.*

The following proposition is a direct consequence of Proposition 12 and the definition of region abstraction.

**Proposition 15.** *Reachability (termination) problems and games on glitch-free RTA* $\mathcal{T}$ *can be reduced to solving reachability (termination) problems and games, respectively, on the corresponding region abstraction* $\mathcal{T}^{\text{RG}}$*.*

## 5.2 Computational complexity

All the results stated here concern glitch-free recursive timed automata only and their formal proofs can be found in [21]. First, we summarise the complexity results for the reachability problem for glitch-free RTAs in Table 2.

| # Players | RTAs with 1 clock | RTAs with at least 2 clocks |
|---|---|---|
| 1 | PTIME-complete | EXPTIME-complete |
| 2 | EXPTIME-complete | 2EXPTIME |

**Table 2.** Complexity results for glitch-free RTAs

By examining the reduction of RTAs to the corresponding RSMs via region abstraction in the previous section, it can be observed that in the case where all the clocks are passed by reference (i.e. they are global) only the number of internal nodes and exits grows exponentially, not the number of boxes. It is simply because the clocks values are never being restored to the value they had before the box was called and hence the valuation of the clocks does not have to be stored at the boxes in the region abstraction. This observation allows us to provide better complexity upper and lower bounds for the reachability problem and games on 1-box RTAs with global clocks, summarised in Table 3, because 1-box RSMs can be analysed a lot more efficiently than multi-exit RSMs. Since the number of exits can grow arbitrarily large after region abstraction is applied to a 1-exit RTA with just a single global clock, no similar improvement can be obtained for 1-exit RSMs with only global clocks.

| # Players | 1-box RTAs with 1 global clock | 1-box RTAs with at least 2 global clocks |
|---|---|---|
| 1 | PTIME-complete | PSPACE (PSPACE-complete for 3+ clocks) |
| 2 | PSPACE-complete | EXPSPACE (and EXPTIME-hard) |

**Table 3.** Complexity results for 1-box RTAs with only global clocks

On the other hand, if all clocks are local then only the number of boxes grows exponentially, not the number of control states (in particular the number of exit ports of each box does not increase). This allows us to provide a much better complexity upper and lower bounds for 1-exit RTAs with only local clocks. We summarise the results for the reachability problem for this subclass of RTAs in Table 4. Again, even if there is only one single local clock, the number of boxes can grow arbitrarily large after the region abstraction is applied to such a system, hence no similar improvements can be achieved when restricting the model to 1-box RTAs with local clocks.

| # Players | 1-exit RTAs with 1 local clock | 1-exit RTAs with at least 2 local clocks |
|---|---|---|
| 1 | PTIME-complete | EXPTIME-complete |
| 2 | PTIME-complete | EXPTIME-complete |

**Table 4.** Complexity results for 1-exit RTAs with only local clocks

## 6 Conclusion

We defined a natural extension of boolean programs with real-time clocks. These clocks, among others, may either correspond to physical time or other continuous values read from sensors. Just like in any advanced imperative programming language, we allow to pass these clocks by value or by reference. We showed that unfortunately arbitrary mixing of these two kinds of variable passing leads to undecidability. On the other hand, if we disallow it, the model becomes decidable and for many special subclasses of this model, the computational complexity is not higher than PSPACE for 1 player setting and EXPTIME for 2 players setting, which is the same as the respective reachability analysis of ordinary finite-state timed automata.

## Acknowledgment

The authors are supported in part by EPSRC grants EP/F001096 and EP/G050112/1.

## References

1. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Transactions on Programming Languages and Systems*, 27:786–818, July 2005.
2. R. Alur and D. Dill. A theory of timed automata. In *Theor. Comput. Sci.*, volume 126, 1994.
3. Rajeev Alur and Mihalis Yannakakis. Model checking of hierarchical state machines. In *ACM SIGSOFT'98*, pages 175–188, 1998.
4. T. Ball and S. Rajamani. The slam toolkit. In *International Conference on Computer Aided Verification, CAV 2001*, pages 260–264, 2001.
5. A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 64–85, 1995.
6. Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. Verification of context-free timed systems using linear hybrid observers. In *International Conference on Computer-Aided Verification, CAV'94*, pages 118–131, 1994.
7. F. Bouchy, A. Finkel, and A. Sangnier. Reachability in timed counter systems. *Electronic Notes in Theoretical Computer Science*, 239:167 – 178, 2009. Proc. of 8th, 9th, and 10th Intl. Workshops on Verification of Infinite-State Systems (INFINITY 06, 07, 08).
8. Giorgio C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications*. Springer-Verlag, Santa Clara, CA, USA, 2004.
9. Franck Cassez, Jan J. Jessen, Kim G. Larsen, Jean-François Raskin, and Pierre-Alain Reynier. Automatic synthesis of robust and optimal controllers — an industrial case study. In *HSCC '09*, pages 90–104. Springer-Verlag, 2009.
10. Z. Dang. Binary reachability analysis of pushdown timed automata with dense clocks. In *International Conference on Computer Aided Verification, CAV 2001*, volume 2102 of *LNCS*, pages 506–517. Springer, 2001.
11. Z. Dang. Pushdown timed automata: a binary reachability characterization and safety verification. *Theor. Comput. Sci.*, 302(1-3):93–121, 2003.
12. Stephane Demri and Regis Gascon. The Effects of Bounding Syntactic Resources on Presburger LTL. *J. Logic Computation*, 19(6):1541–1575, 2009.
13. M. Emmi and R. Majumdar. Decision problems for the verification of real-time software. In *Hybrid Systems: Computation and Control*, pages 200–211, 2006.
14. K. Etessami and M. Yannakakis. Recursive markov decision processes and recursive stochastic games. In *Proc. ICALP'05*, pages 891–903, 2005.
15. Kousha Etessami. Analysis of recursive game graphs using data flow equations. In *VMCAI'04*, pages 282–296, 2004.
16. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *CAV'97*, pages 72–83, 1997.
17. Petr Jancar and Zdenek Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *Inf. Process. Lett.*, 104(5):164–167, 2007.
18. Dominik Wojtczak Kousha Etessami and Mihalis Yannakakis. Quasi-Birth-Death processes, Tree-like QBDs, Probabilistic 1-Counter Automata, and Pushdown Systems. *Performance Evaluation*, 67(9):837 – 857, 2010. Special Issue of QEST 2008.
19. Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
20. Olivier Serre. Parity games played on transition graphs of one-counter processes. In *FoSSaCS'06*, pages 337–351, 2006.
21. Ashutosh Trivedi and Dominik Wojtczak. Recursive timed automata. *Oxford University Computing Laboratory technical report, RR-10-09*, 2010.
22. Igor Walukiewicz. Pushdown processes: Games and model checking. In *International Conference on Computer Aided Verification, CAV 1996*, pages 62–74, 1996.