2015 22nd International Symposium on Temporal Representation and Reasoning

Reachability Games on Recursive Hybrid Automata

Shankara Narayanan Krishna
IIT Bombay

Lakshmi Manasa
IIT Bombay

Ashutosh Trivedi
IIT Bombay

***Abstract*—Recursive hybrid automata generalize recursive state machines in a similar way as hybrid automata generalize state machines. Recursive hybrid automata can be considered as collection of classical hybrid automata with special states that correspond to potentially recursive invocation of hybrid automata from the collection. During each such invocation, the semantics of recursive hybrid automata permits *optional* passing of the continuous variables using either pass-by-value or pass-by-reference mechanism. This model generalizes the recursive timed automata model introduced by Trivedi and Wojtczak and dense-timed pushdown automata by Abdulla, Atig, and Stenman. We study two-player turn-based reachability games on recursive hybrid automata. Given the undecidability of even the reachability problem on hybrid automata, it is not surprising that the problems remain undecidable without further restrictions. We consider various restrictions of recursive hybrid automata where we recover decidability of reachability games and characterize the boundaries between decidable and undecidable variants.**

I. Introduction

In this paper we study reachability games on recursive hybrid automata. In order to place recursive hybrid automata in the larger picture of formal verification and synthesis we take a detour and briefly discuss related formalisms with emphasis on decidability and complexity of reachability games.

Finite State Machines. Model checking is probably the most successful application [2], [3] of formal methods in development of software and hardware based control systems. In this approach, the system being developed is modeled as state machines and checked against formal specification via an exhaustive exploration of the state space. Another related approach is *control-program synthesis* [19], [17] where the action space is divided between two players—the controller and the environment—and the goal is to design a provably correct system via a winning strategy of the controller against a given objective. Finite state machines (FSMs) are the most fundamental model used for model checking and synthesis as finiteness of their state space results in decidable verification and synthesis. In particular, the fundamental verification problem of reachability (or emptiness) over FSMs is NLogSpace-Complete, while the corresponding synthesis problem of reachability game is PTIME-complete.

Concurrency, Hierarchy, and Recursion. The introduction of *concurrency* to FSMs enriches the language of finite state machines and allows the designers to model the system modularly and exponentially succinctly. Concurrency also enables separation-of-concerns by letting designers model various interacting modules individually and connect them using well-defined synchronization primitives. Hierarchical state machines [7], [8] allow another level of sophistication to concurrent FSMs by introducing states that themselves can be finite state machines or even hierarchical state machines, and give rise to even more intuitive modeling for complex systems by enabling the designer to design subsystems at various levels of hierarchy. The use of Hierarchical state machines became popular with the introduction of Harel's Statecharts [12] and now they are integral part of model-based design frameworks such as Unified Modeling language (UML) [9] and Stateflow by Mathworks [11]. Recursive state machines (RSMs) [5] generalize hierarchical state machines by permitting potentially recursive invocation of sub-modules. However, RSMs do not allow concurrency since it is well known [20] that concurrency and recursion together lead to undecidability of simple verification problems. However, RSMs can model a network of communicating state machines along with one recursive state machine, as their synchronous product remains a recursive state machine. The reachability problem for RSMs is in PTime while the reachability games for RSMs are ExpTime-complete [22].

Hybrid and Timed Automata. Hybrid automata are natural extension of finite state machines with continuous variables whose dynamics is governed by state-dependent ordinary differential equations. Hybrid automata are a suitable formalism to model composition of finite-state controllers interacting with continuous environment. An important subclass of hybrid automata is so-called singular hybrid automata where variables have state-dependent constant-rate. The reachability problem for singular hybrid automata is already undecidable for three or more variables [13]. However, for an important subclass of hybrid automata—called timed automata—where all variables grow with uniform rate (and hence are called clocks), the reachability problem remains decidable and is PSpace-Complete [6] for timed automata with 2 clocks. Also, reachability games on timed automata are ExpTime-Complete [14] for timed automata with 2 clocks. It is also known that bounded-time reachability problem for singular hybrid automata under certain restriction remains decidable [10], however no such result is known for games.

Recursive Timed Automata. Recursive Timed automata [21] and dense-timed pushdown automata [4] are two different models that introduce recursion to timed automata. A *recursive timed automaton* (RTA) can be considered as a recursive state machine with continuous variables that grow with uniform rates. 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

1530-1311/15 $31.00 © 2015 IEEE
DOI 10.1109/TIME.2015.27

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 mechanisms:

- *pass-by-value*, where upon returning from the called component a clock assumes the value prior to the invocation,
- and *pass-by-reference*, where upon return clocks reflect any changes to the value inside the invoked procedure.

Trivedi and Wojtczak [21] showed that the reachability problem is undecidable for RTAs with three or more clocks, while it is decidable (EXPTIME-COMPLETE) for 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. They also showed that the reachability games are undecidable for RTAs with two clocks, while they are decidable (2EXPTIME) for glitch-free RTAs. Unlike the RTA model, where it is compulsory to pass all the clocks at every invocation with either mechanism, Abdulla, Atig, Stenman [4] studied a related model called timed pushdown automata (TPDAs) where 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). On the other hand the model TPDAs disallowed passing clocks by value. Abdulla et al. showed that the reachability problem for this class remains decidable (EXPTIME-COMPLETE).

Recursive Hybrid Automata. We have recently introduced recursive hybrid automata (RHA) [16] that combines the models of RTA [21] and TPDA [4] and further generalize it to support singular hybrid variables (variables with state-dependent constant rates). In this model the variables can be passed either by value (V) or reference (R) as in [21], or not passed at all (N), as in [4]. We explored the decidability of the reachability problem.

Example 1. *The visual presentation of a recursive hybrid automaton with two components M_1 and M_2, and two variables x and y is shown in Figure 1, 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). The rates of the variables are written below each node. We assume a fixed ordering on the variables and show their rates in that order. Here we assume $x < y$. For example, the rate $(1, 2)$ at the return port of b_1:M_2 implies $\dot{x} = 1$ and $\dot{y} = 2$. 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. Above b:M, we write a tuple (C_1, C_2) where C_1, C_2 respectively, are the set of variables passed to M by value and not passed at all; the rest of the variables are passed by reference. When $C_1 = C_2 = \emptyset$, then we write nothing on top of b:M. Each transition is labelled 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 taking this transition, variable y is reset.*

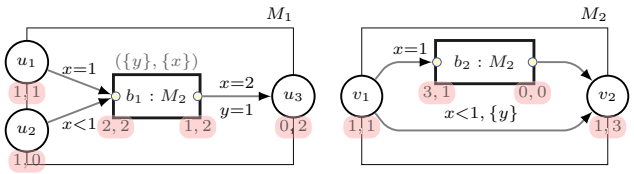

Fig. 1. A recursive hybrid automaton with two components and two variables.

We classify RHA based on the variable passing mechanisms in the following manner. We write RHA_R and RHA_V for RHAs where only pass-by-reference and pass-by-value, resp., mechanisms are used to pass variables during a recursive call. We write RHA_N for RHAs where none of the variables are passed during a recursive call; Similarly, we write RHA_{RV}, RHA_{RN}, RHA_{NV} and RHA_{RNV} for various combinations of permissible clock passing mechanisms. The RHA_{RNV} is the most permissible subclass. We say that a RHA is *glitch-free* if for every box either all variables are passed by value or none is passed by value, otherwise for emphasis we call the RHA unrestricted. We say that a RHA is *hierarchical* if there exists an ordering over components such that a component never invokes another component of higher order or same order. Finally an RHA is *bounded context* if there is a bound K such that during every run the height of the call stack is bounded by K. Note that hierarchical RHAs are always bounded context.

In [1] we showed that the time-bounded reachability problem is decidable for bounded-context RHA using the pass-by-reference as well as no-passing mechanisms, while for the time-unbounded case, it is undecidable with 3 stopwatches (variables with rates restricted to $\{0, 1\}$. When the pass-by-value mechanism is allowed, the time-bounded reachability problem becomes undecidable even for glitch-free RHA. This class has a decidable reachability with $\leq$ 2 stopwatches. For the subclass of recursive timed automata (RTA), we show that the reachability problem is decidable for bounded context, glitch-free-RTA using all three mechanisms (pass-by-value, pass-by-reference and not passing), while even the time-bounded reachability turns out to be undecidable for unrestricted RTA, using any of the 2 mechanisms pass-by-value+no-passing with 7 or more clocks or pass-by-value+pass-by-reference with 5 or more clocks [1].

Reachability Games on RHAs. In this paper we extend the results from [1] for reachability problem to two-player reachability games on RHA. We assume that the set of node are partitioned between two players—Player 1 and Player 2—where the goal of the player Player 1 is to reach a designated set of nodes, while the goal of player Player 2 is to avoid it. Table I summarizes our results on reachability games on RHAs.

TABLE I

THIS TABLE SUMMARIZES RESULTS FOR REACHABILITY GAMES ON RHAS. THE CONTRIBUTIONS OF THIS PAPER ARE EMPHASIZED IN BOLD LETTERS ALONG WITH REFERENCE TO THE RELATED THEOREMS. HERE c STANDS FOR CLOCK VARIABLES, s FOR STOPWATCH VARIABLES, AND BC STANDS FOR BOUNDED CONTEXT RESTRICTION. ALL OF OUR UNDECIDABILITY RESULTS HOLD UNDER HIERARCHIAL AND BOUNDED CONTEXT RESTRICTIONS.

	Recursive Timed Games		**Recursive Hybrid Games**	
	General	Time-bounded	General	Time-bounded
RHA_R	Decidable [21]	Decidable [21]	Undecidable(3s)[13] **Decidable** (2s)Thm1(2)	**Undecidable** (4s)Thm2(1) **Decidable** (2s)Thm1(2)
RHA_N	**Decidable**Thm1(1)	**Decidable**Thm1(1)	Undecidable (3s) [13]	**Undecidable** (4s)Thm2(1)
RHA_V	Decidable [21]	Decidable [21]	Undecidable(2c+1s) [16] **Decidable**(2s)Thm1(2)	**Undecidable**(4s)Thm2(1) **Decidable**(2s)Thm1(2)
RHA_{RN}	**Decidable**Thm1(1)	**Decidable**Thm1(1)	Undecidable(3s)[13]	**Undecidable**(4s)Thm2(1)
RHA_{RV} (glitch-free)	Decidable [21]	Decidable [21]	Undecidable(2c+1s) [16] **Decidable**(2s)Thm1(2)	**Undecidable**(4s)Thm2(1) **Decidable**(2s)Thm1(2)
RHA_{RV}	Undecidable(2c) [21]	Undecidable(3c) [15]	Undecidable(2c) [21]	Undecidable(3c) [15]
RHA_{NV} (glitch-free)	**Decidable**(BC)Thm1(3)	**Decidable**(BC)Thm1(3)	Undecidable(2c+1s) [16]	**Undecidable**(4s)Thm2(1)
RHA_{NV}	Undecidable(3c) [15]	**Undecidable**(4c)Thm2(3)	Undecidable(2c+1s) [16]	**Undecidable**(4c)Thm2(3)
RHA_{RNV} (glitch-free)	**Decidable**(BC)Thm1(3)	**Decidable**(BC)Thm1(3)	Undecidable(2c+1s) [16]	Undecidable(3c) [15]
RHA_{RNV}	Undecidable(2c) [21]	**Undecidable**(3c)Thm2(2)	Undecidable(2c) [21]	Undecidable(3c) [15]

II. PROBLEM DEFINITION

We begin our presentation by introducing reachability games on labeled transition systems and RHAs.

A. Reachability Games on Labelled Transition Systems

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 $\mathit{Runs}^{\mathcal{L}}$ ($\mathit{FRuns}^{\mathcal{L}}$) for the sets of infinite (finite) runs and $\mathit{Runs}^{\mathcal{L}}(s)$ ($\mathit{FRuns}^{\mathcal{L}}(s)$) 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 $\mathit{last}(r) = s_n$ for the last state of the run. A *strategy* in $\mathcal{L}$ is a function $\sigma : \mathit{FRuns}^{\mathcal{L}} \to A$ such that for all runs $r \in \mathit{FRuns}$ we have that $\sigma(r) \in A(\mathit{last}(r))$. 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 $\mathrm{Run}(s, \sigma)$ for the unique run $\langle s_0, a_1, s_1, \ldots \rangle \in \mathit{Runs}^{\mathcal{L}}(s)$ 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 $\mathit{Stop}(F)(r) = \inf \{i \in \mathbb{N} : s_i \in F\}$.

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 $\mathit{Stop}(F)(\mathrm{Run}(s, \sigma)) < \infty$. A *reachability problem* is to decide whether in a given LTS a final state is reachable from a given initial state.

A *game arena* G is a tuple $(\mathcal{L}, S_1, S_2)$, where $\mathcal{L} = (S, A, X)$ is an LTS, $S_1 \subseteq S$ is the set of states controlled by player Player 1, and $S_2 \subseteq S$ is the set of states controlled by player Player 2. Moreover, sets S_1 and S_2 form a partition of S.

A strategy of player Player 1 is a partial function $\mu : \mathit{FRuns}^{\mathcal{L}} \to A$ such that for a run $r \in \mathit{FRuns}^{\mathcal{L}}$ we have that $\mu(r)$ is defined if $\mathit{last}(r) \in S_1$, and $\mu(r) \in A(\mathit{last}(r))$ for every such r. A strategy of player Player 2 is defined analogously. Let $\Sigma_1^{\mathcal{L}}$ and $\Sigma_2^{\mathcal{L}}$ be the set of strategies of player Player 1 and Player 2, respectively. The unique run $\mathrm{Run}(s, \mu, \chi)$ from a state s when players use strategies $\mu \in \Sigma_1^{\mathcal{L}}$ and $\chi \in \Sigma_2^{\mathcal{L}}$ is defined in a straightforward manner.

In a *reachability game* on G, rational players Player 1 and Player 2 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 Player 1 is to eventually reach certain states, while the objective of Player 2 is to avoid them forever. For an initial state s and a set of final states F, the lower value $\underline{\mathrm{Val}}_F^{\mathcal{L}}(s)$ of the reachability game is defined as the upper bound on the number of transitions that Player 2 can ensure before the game visits a state in F irrespective of the strategy of Player 1, and is equal to $\sup_{\chi \in \Sigma_2^{\mathcal{L}}} \inf_{\mu \in \Sigma_1^{\mathcal{L}}} \mathit{Stop}(F)(\mathrm{Run}(s, \mu, \chi))$. The concept of upper value is $\overline{\mathrm{Val}}_F^{\mathcal{L}}(s)$ is analogous and defined as $\inf_{\mu \in \Sigma_1^{\mathcal{L}}} \sup_{\chi \in \Sigma_2^{\mathcal{L}}} \mathit{Stop}(F)(\mathrm{Run}(s, \mu, \chi))$. If $\underline{\mathrm{Val}}_F^{\mathcal{L}}(s) = \overline{\mathrm{Val}}_F^{\mathcal{L}}(s)$ then we say that the reachability game is determined, or the value $\mathrm{Val}_F^{\mathcal{L}}(s)$ of the reachability game exists and it is such that $\mathrm{Val}_F^{\mathcal{L}}(s) = \underline{\mathrm{Val}}_F^{\mathcal{L}}(s) = \overline{\mathrm{Val}}_F^{\mathcal{L}}(s)$. We say that Player 1 wins the reachability game if $\mathrm{Val}_F^{\mathcal{L}}(s) < \infty$. 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 Player 1 has a strategy to win the reachability game.

B. Reachability Games on Recursive Hybrid Automata

Let $\mathcal{X}$ be a finite set of real-valued continuous variables. A *valuation* on $\mathcal{X}$ is a function $\nu : \mathcal{X} \to \mathbb{R}$. We assume an arbitrary but fixed ordering on the variables and write x_i for the variable with order i. This allows us to treat a valuation ν as a point $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n)) \in \mathbb{R}^{|\mathcal{X}|}$. Abusing notations slightly, we use a valuation on $\mathcal{X}$ and a point in $\mathbb{R}^{|\mathcal{X}|}$ interchangeably. For a subset of variables $X \subseteq \mathcal{X}$ 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 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 $\text{rect}(\mathcal{X})$ for the set of rectangular constraints over $\mathcal{X}$.

A *Recursive Hybrid Automaton* $\mathcal{H} = (\mathcal{X}, (\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k))$ is a pair made of a set of variables $\mathcal{X}$ and a collection of components $(\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k)$ where every component $\mathcal{H}_i$ = (N_i, EN_i, EX_i, B_i, Y_i, A_i, X_i, PV_i, PR_i, PN_i, Inv_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 $\text{Call}(b)$ and return ports $\text{Ret}(b)$ 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 $Q_i = N_i \cup \text{Call}_i \cup \text{Ret}_i$ and refer to this set as the set of nodes of $\mathcal{H}_i$. Let $Q = \bigcup_{i=1}^{n} Q_i$.)
- A_i is a finite set of *actions*.
- $X_i \subseteq Q_i \times A_i \times \text{rect}(\mathcal{X}) \times 2^{\mathcal{X}} \times Q_i$ is the *transition relation* with a condition that call ports and exit nodes do not have any outgoing transitions.
- $PV_i : B_i \to 2^{\mathcal{X}}$ is pass-by-value mapping that assigns every box, the set of variables that are passed by value to the component mapped to the box;
- $PR_i : B_i \to 2^{\mathcal{X}}$ is pass-by-reference mapping that assigns every box, the set of variables that are passed by reference to the component mapped to the box;
- $PN_i : B_i \to 2^{\mathcal{X}}$ is non-passing mapping that assigns to every box, the set of variables not passed to the box.
- For every box $b \in B_i$, the sets $PV_i(b)$, $PR_i(b)$ and $PN_i(b)$ form a partition of $\mathcal{X}$.
- $Inv_i : Q_i \to \text{rect}(\mathcal{X})$ is the *invariant condition*;
- $F_i : Q_i \to \mathbb{Z}^{|\mathcal{X}|}$ is the *flow function* characterizing the rate of each variable in each location.

We assume that the sets of boxes, nodes, locations, etc. are mutually disjoint across components and we write (N, B, Y, Q, PV, PR, PN, etc.) for union over all components. We also assume that the transitions are deterministic with respect to the set of actions. This can be done w.l.o.g. in the case of reachability games as all non-determinism can be moved as controlled by one player or the other.

A *state* of an RHA $\mathcal{H}$ 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 node 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 state. Technically, it suffices to store the valuation of variables passed by value and those not passed, because variables by reference retain their value after returning from a call, 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 $\langle\kappa\rangle{+}t$ equal the context $\langle\kappa'\rangle$ where if $\kappa = (b_1, \nu_1)(b_2, \nu_2) \cdots (b_n, \nu_n)$ then $\kappa' = (b_1, \nu_1')(b_2, \nu_2') \cdots (b_n, \nu_n')$ where for all $1 \leq i \leq n$

$$\nu_i' = \nu_i[PN(b_i) := \nu_i{+}F(\text{Call}(b_i)) \cdot t].$$

Informally, the behavior of an RHA is as follows. In state $(\langle\kappa\rangle, q, \nu)$ time passes before an available action is triggered, after which a discrete transition occurs. A transition (q, a, φ, C, q'), with $\varphi \in \text{rect}(\mathcal{X})$ and $C \subseteq \mathcal{X}$ is enabled on an action a after a time elapse t only if $\nu{+}F(q) \cdot t \in [\![\varphi]\!]$. The successor state is given by $(\langle\kappa\rangle + t, q', \nu')$ and $\nu' = (\nu + t)[C := \mathbf{0}]$. The semantics of an RHA is given by an LTS with uncountably many states and transitions.

Let $\mathcal{H} = (\mathcal{X}, (\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k))$ be an RHA where each component is of the form $\mathcal{H}_i$ = (N_i, EN_i, EX_i, B_i, Y_i, A_i, X_i, PV_i, PR_i, PN_i, Inv_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. $q = (b, en) \in \text{Call}$ then $t = 0$, the context $\langle\kappa'\rangle = \langle\kappa, (b, \nu)\rangle$, $q' = en$, and $\nu' = \nu[PN(b) := 0]$.
  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 $(b, ex) \in \text{Ret}(b)$, then $t = 0$; $\langle\kappa'\rangle = \langle\kappa''\rangle$; $q'{=}(b, ex)$; and $\nu'{=}\nu[PV(b), PN(b){:=}\nu'']$.
  3) if location q is any other kind of location, then $\langle\kappa'\rangle = \langle\kappa\rangle{+}t$ such that for some $(q, a, \varphi, C, q') \in X$ we have:
     a) $\nu{+}F(q) \cdot t' \in Inv(q)$ for all $t' \in [0, t]$;
     b) $\nu{+}F(q) \cdot t \in [\![\varphi]\!]$
     c) $\nu' = (\nu + F(q){\cdot}t)[C := \mathbf{0}]$.

For a subset $Q' \subseteq Q$ of nodes of an RHA $\mathcal{H}$ we define $[\![Q']\!]_{\mathcal{H}} = \{(\langle\kappa\rangle, q, \nu) \in S_{\mathcal{H}} : q \in Q'\}$. Given an RHA $\mathcal{H}$, an initial node q and valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, a set of *final nodes* $F \subseteq Q$, the *reachability problem* is to decide the existence of a run in the $[\![\mathcal{H}]\!]$ starting from $(\langle\varepsilon\rangle, q, \nu)$ to a state in $[\![F]\!]_{\mathcal{H}}$.

Given a run $r = \langle s_0, (t_1, a_1), s_1, (t_2, a_2), \ldots, s_n \rangle$ of an RHA, its time duration $\mathsf{time}(r)$ is defined as $\sum_{i=1}^{n} t_i$. Given an RHA $\mathcal{H}$, an initial node q, a bound $T \in \mathbb{N}$, a valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, and a set of *final nodes* $F \subseteq Q$, 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 $\mathsf{time}(r) \leq T$.

A partition (Q_1, Q_2) of locations Q of an RHA $\mathcal{H}$ gives rise to a recursive hybrid game arena $\Gamma = (\mathcal{H}, Q_1, Q_2)$. Given an initial location q, a valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$ and a set of final states F, the reachability game on Γ is defined as the reachability game on the game arena $([\![\mathcal{H}]\!], [\![Q_1]\!]_{\mathcal{H}}, [\![Q_2]\!]_{\mathcal{H}})$ with the initial state $(\langle\varepsilon\rangle, (q, \nu))$ and the set of final states $[\![F]\!]_{\mathcal{H}}$. Similarly, time-bounded reachability game is defined.

C. Subclasses of Recursive Hybrid Automata

We consider the following subclasses of RHA.

- RHA_R and RHA_V : RHAs where only pass-by-reference and pass-by-value, respectively, mechanism is used to pass variables during a recursive call.
- RHA_N : RHA where none of the variables are passed during a recursive call i.e; variables in the invoked component start at valuation $\mathbf{0}$ and those in the context keep growing with the rates of their called points.
- RHA_{RV} : RHA using both pass-by-reference and pass-by-value but does not use "not passing" of variables. We similarly define RHA_{RN}, RHA_{NV} and RHA_{RNV}.

We say that a recursive hybrid automaton is *glitch-free* if for every box either all variables are passed by value or none are passed by value, i.e. for each $b \in B$ we have that either $PV(b) = \mathcal{X}$ or $PV(b) = \emptyset$. An RHA without this restriction is called *unrestricted*. Any general recursive hybrid automaton with one variable is trivially glitch-free. We say that a RHA is *hierarchical* if there exists an ordering over components such that a component never invokes another component of higher order or same order. An RHA is *bounded context* if there is a bound K such that in any state $(\langle\kappa\rangle, q, \nu)$, $|\kappa| \leq K$. Note that hierarchical RHAs are always bounded context RHAs.

We say that a variable $x \in \mathcal{X}$ is a *clock* (resp., a stopwatch) if for every node $q \in Q$ we have that $F(q)(x) = 1$ (resp., $F(q)(x) \in \{0, 1\}$). A recursive timed automaton (RTA) is simply a recursive hybrid automata where all variables $x \in \mathcal{X}$ are clocks.

D. Our Key Results

The following theorems list our main contributions on recursive hybrid games. We have characterized the decidability frontier for each subclass of games. For ease of reading, the decidability resuts are explained first for the one player case, and then extended for the two player case.

Theorem 1 (Decidability)**.** *The reachability game problems are decidable for the following fragments of RHA:*
(1) RTA_{RN},
(2) Glitch-free RHA_{RV} with 2 stopwatches,
(3) Bounded context Glitch-free RTA_{RNV}.

The decidability of glitch-free RHA_{RV} with 2 stopwatches (Theorem 1(2)) follows from the existence of region abstraction in the case of 2 stopwatches. Section IV is devoted to other two parts of Theorem 1.

Theorem 2 (Undecidability)**.** *The reachability game problem is undecidable for*
(1) Hybrid Automata with 4 stopwatches under bounded time.
(2) Unrestricted RTA_{RV} with 3 clocks under bounded time.
(3) Unrestricted RTA_{NV} with 4 clocks under bounded time.
Moreover, all of these results hold even under the hierarchical, bounded context restrictions.

The next section is devoted the proof of this theorem.

III. Undecidability Results

We give the details for Theorem 2, part 1 here.

Lemma 1. *The time bounded reachability game problem is undecidable for hybrid automata with at least 4 stopwatches.*

Proof. We prove that the reachability problem is undecidable for hybrid automata with 4 stopwatches. In order to obtain the undecidability result, we use a reduction from the halting problem for two counter machines. A *two-counter machine* M is a tuple (L, C) 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 $C = \{c, d\}$ of two *counters*. The instructions L are of the type: (1) (increment c) $\ell_i : c := c+1$; goto ℓ_k, (2) (decrement c) $\ell_i : c := c - 1$; goto ℓ_k, (3) (zero-check c) ℓ_i : if $(c > 0)$ then goto ℓ_k else goto ℓ_m, where $c \in C$, $\ell_i, \ell_k, \ell_m \in L$. 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, respectively. 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 ([18]) that the halting problem for two-counter machines is undecidable.

Our reduction uses a hybrid automata (HA) with four stopwatches x, y, z, b. We specify a path for each instruction of the two counter machine. On entry into a path for increment/decrement/zero check, we have $x = \frac{1}{2^{k+c}3^{k+d}}$, $y = \frac{1}{2^k}$ and $z = b = 0$, where c, d are the current values of the counters c, d and k is the current instruction. Note that z is used only to enforce urgency in several locations while b is used for rough work. Given a two counter machine, we build a 4 stopwatch hybrid automata whose building blocks are the paths for the instructions. The purpose of these paths is to simulate faithfully the counter machine by choosing appropriate delays to adjust the variables to reflect changes in counter values. On entering the first location en of a main path corresponding to an instruction l_i, we have the configuration $(\langle\epsilon\rangle, en, (\frac{1}{2^{k+c}3^{k+d}}, \frac{1}{2^k}, 0, 0))$ of the 4 stopwatch HA.

Figure 2 describes the module that increments counter c. The variables which have rate 1 at a location are written below

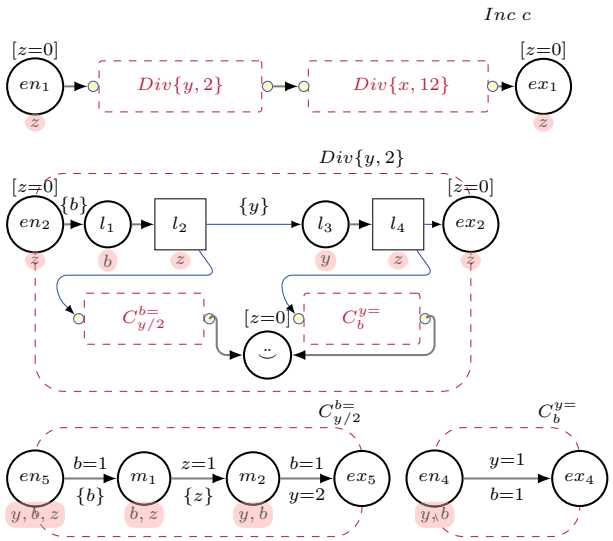

Fig. 2. Games on Hybrid Automata with 4 Stopwatches : Increment c.

each location in a pink envelope. Assume that this increment instruction is the $k+1$th instruction. At the end of the module, we require $x = \frac{1}{2^{k+c+2}3^{k+d+1}}, y = \frac{1}{2^{k+1}}$, to account for the increment of c, and to account for the $k+1$th instruction. For this, from the port en_1, the module $Div\{y,2\}$ is invoked, which divides the value of y by 2, making $y = \frac{1}{2^{k+1}}$. This is followed by the module $Div\{x,12\}$, which divides the value of x by 12, obtaining $x = \frac{1}{2^{k+c+2}3^{k+d+1}}$. Let us look at the module $Div\{y,2\}$. No time is spent at the initial location en_2 of this module. The clock b is reset to 0 on going to l_1. The rate of b is 1 at location l_1. An amount of time t is spent at location l_1. We want $t = \frac{y}{2}$. The control passes to a player 2 location l_2, where the rate of b is zero. The only variable with rate 1 at l_2 is z. Player 2 can either continue the simulation or check whether $t = \frac{y}{2}$ by invoking the module $C^{b=}_{\frac{y}{2}}$. Note that no time is spent at l_2, since otherwise, the invariant $z = 0$ will not hold at the node ex_2. Let us examine the module $C^{b=}_{\frac{y}{2}}$. At the initial location en_5, a time elapse of $1-t$ happens, and b is reset. At the same time,we obtain $y = \frac{1}{2^k} + 1 - t$ and $z = 1 - t$ (y, b, z tick at en_5). At m_1, a time t is spent, obtaining $y = \frac{1}{2^k} + 1 - t$, $z = 0$ and $b = t$ (only b, z tick at m_1). Finally, we reach ex_5 iff we reach $b = 1$ and $y = 2$ simultaneosly, that is, if $1 - t = 1 + t - \frac{1}{2^k}$. This is true iff $t = \frac{1}{2^{k+1}}$. Note that the value of z is zero at ex_5, since z was not ticking at m_2. We are thus at the exit node of $C^{b=}_{\frac{y}{2}}$. On return to l_2, player 1 can thus reach the happy state iff $t = \frac{1}{2^{k+1}}$.

Assuming that $t = \frac{1}{2^{k+1}}$, if player 2 goes ahead to location l_3 in $Div\{y,2\}$, then we obtain $y = z = 0$ and $b = \frac{1}{2^{k+1}}$ at l_3. At l_3, y is ticking at rate 1; here player 1 should elapse a time of $\frac{1}{2^{k+1}}$ to ensure $y = b$. The player 2 location l_4 can check this; we do not go into the details of this. This check is done using the module $C^{y=}_{b}$ similar to $C^{b=}_{\frac{y}{2}}$. Again, no time is spent at l_4. From here, we reach the exit node ex_2 of the $Div\{y,2\}$ module. At the return port of $Div\{y,2\}$, no time is spent, and the module $Div\{x,12\}$ is invoked. This is similar to $Div\{y,2\}$: the change is done in the $C^{b=}_{\frac{y}{2}}$ module. We will have more instances of the locations m_1, m_2 connected back to back to ensure that a time $t = \frac{1}{2^{c+k+2}3^{d+k+1}}$ is spent at the location corresponding to l_1 in $Div\{x,12\}$.

Figure 3 gives the module for zero check of counter d. We briefly describe this here. At en_1, no time elapse happens. We divide y by 2 and x by 6, to reflect the change of instruction number from k to $k+1$ in both x and y. After this, player 1 guesses wheteher d is 0 or not, by choosing one of the locations m_1 or m_2 from m. Both m_1, m_2 are player 2 locations. Player 2 can continue the simulation by going to the locations marked $d = 0$ and $d > 0$ respectively, or gone ahead with checking player 1's guess. The module $ZC^d_{=0}$ does this. The module $ZC^d_{=0}$ repeatedly multiplies y by 2 and x by 6, till y reaches 1. The components $Mul\{y,2\}$ and $Mul\{x,6\}$ do this. When y reaches 1, we obtain $x = \frac{1}{2^c 3^d}$. Now we keep on multiplying x by 2. If $d = 0$, then eventually, we will obtain $x = 2$, otherwise, $x > 2$. If player 2 had invoked $ZC^d_{=0}$ from m_1, then a happy state is reached on obtaining $x = 2$, thereby validating player 1's guess. Similarly, if player 2 had invoked $ZC^d_{=0}$ from m_2, then a happy state is reached on obtaining $x > 2$, again validating player 1's guess. The $Mul\{y,2\}$ module is similar to the $Div\{y,2\}$ module.

The decrement module is similar to the increment module: starting with $x = \frac{1}{2^{k+c}3^{k+d}}$, $y = \frac{1}{2^k}$ and $z = b = 0$, at the end of the decrement instruction of counter c (this is the $k+1$th instruction), we want $x = \frac{1}{2^{k+c}3^{k+d+1}}$, $y = \frac{1}{2^{k+1}}$ and $z = b = 0$. This is implemented in a way similar to the increment module by having the modules $Div\{y,2\}$ and $Div\{x,3\}$.

We now discuss the total time taken to reach a $\ddot\smile$ state, or the time taken to reach the last node in a path corresponding to the current instruction. We shall first discuss the time taken during the sub path $Div\{y,2\}$. The invariant $z{=}0$ prevents time delay in locations where z is ticking in the sub path. Thus the only time delay is in location l_1, say t_1 and in l_3 say t_2. Note that b is reset before entering l_1. From the above description of $C^{b=}_{\frac{y}{2}}$, we know that value in b upon entering en_5 is $\frac{y}{2} = \frac{1}{2^{k+1}}$. Thus the delay $t_1 = \frac{1}{2^{k+1}}$. Similarly, y is reset before entering l_3 and as $y = b$ upon entering en_4, we know that the delay in l_3 must have been $\frac{1}{2^{k+1}}$. Thus the total delay from en_2 to ex_2 is $2 * \frac{1}{2^{k+1}}$. Now, let us consider the path from en_2 to $\ddot\smile$ via sub path $C^{b=}_{\frac{y}{2}}$. This is the time delay at l_1 along with the time taken from en_5 to ex_5. Once again note that as the invariant on $\ddot\smile$ is $z = 0$, along this path too, no time can elapse in the location l_2. Let $b = \alpha$ upon entering en_5. Then the total time elapsed before reaching $\ddot\smile$ is $1 - \alpha + \alpha + 1 - \alpha$. This together with the delay in l_1 is 2 t.u. Similarly, we see that the time elapsed from en_2 to $\ddot\smile$ via sub path $C^{y=}_{b}$ is $1 + \alpha < 2 t.u$. Thus the total time elapsed to reach $\ddot\smile$ is at most 2 t.u and the time elapsed to reach end of sub path $Div\{y,2\}$ is $2 * \frac{1}{2^{k+1}}$.

Now let us analyse the time spent in the sub path

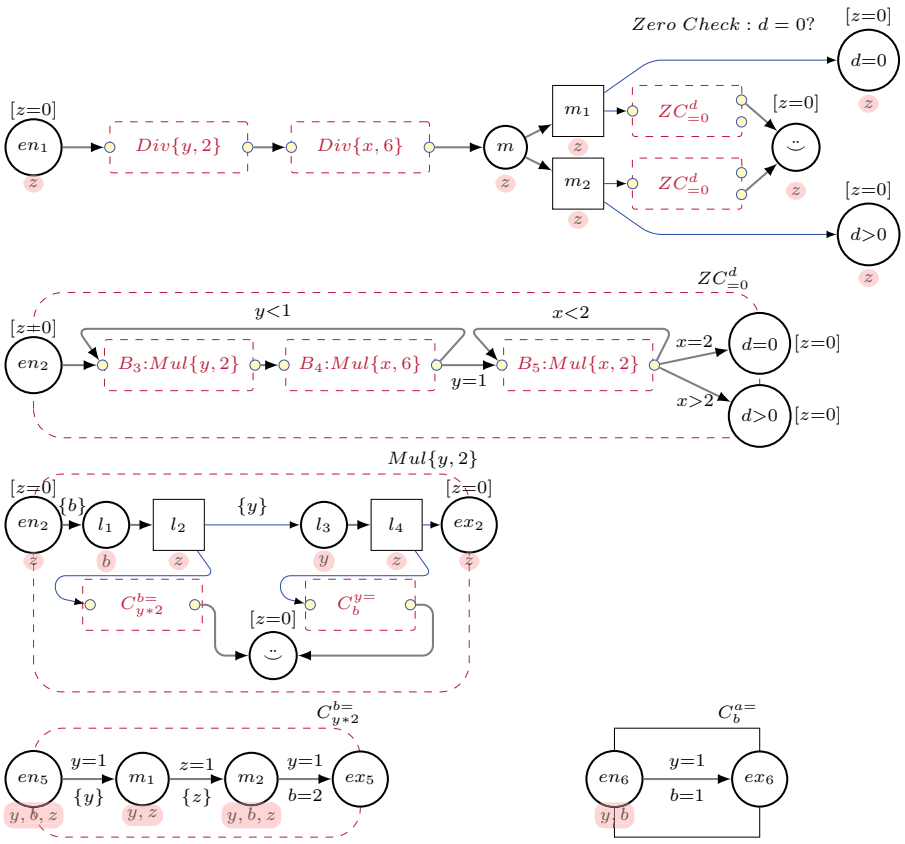

Fig. 3. Games on Hybrid Automata with 4 Stopwatches : Zero check $c = 0$?.

$Div\{x,12\}$. As this path is very similar to $Div\{y,2\}$, the time spent with in this path before reaching ex_1 is $2 * \frac{1}{2^{k+c+2}3^{k+d+1}}$. The sub path $C^{b=}_{\frac{x}{12}}$ corresponds to $C^{b=}_{\frac{y}{2}}$ but has 12 instances of the guard $b = 1$ as opposed to 2 in $C^{b=}_{\frac{y}{2}}$. This is to accumulate $12 * (1 - \alpha)$ in x where $b = \alpha$ upon entering en_5 and we check if $x = 12$ on the transition to ex_5. Thus the time delay is 12 t.u while the delay to reach $\ddot\smile$ via $C^{x=}b$ is again < 2 t.u. Thus the time spent within $Div\{x,12\}$ before reaching ex_1 is $2 * \frac{1}{2^{k+c+2}3^{k+d+1}}$. Hence, the total time elapsed between en_1 and ex_1 is $2 * \frac{1}{2^{k+1}} + 2 * \frac{1}{2^{k+c+2}3^{k+d+1}} < 4 * \frac{1}{2^{k+1}}$. It is easy to see that the maximum time spent to reach $\ddot\smile$ is 18 t.u during the sub path $Div\{x,18\}$ in the path to increment d.

In case the two counter machine does not halt, and player 2 does not check any of player 1's guesses, then the time taken to simulate the $k+1$th instruction is $\leq 4 * \frac{1}{2^{k+1}}$ (for instance, $\frac{1}{2^{k+1}}$ twice in the modules $Div\{y,2\}$ and $\frac{x}{2^{k+c+2}3^{k+d+1}}$ twice in $Div\{x,12\}$). Thus, in case the machine does not halt, the time spent is $\leq \frac{4}{2} + \frac{4}{4} + \frac{4}{8} + \frac{4}{16} + \ldots$, and when player 2 takes player 1 for a check, the total time elapse is ≤ 18. □

IV. Decidability Results

In this section, we briefly describe our decidability results.

A. Bounded Context RHA_{RNV} to RHA_{RV}

In this section, we discuss the non-passing mechanism (N) under the bounded context restriction. We show that given a RHA_{RNV} $\mathcal{A}$, we can remove the non-passing mechanism to obtain RHA_{RV} $\mathcal{B}$ such that $\mathcal{B}$ is equivalent to $\mathcal{A}$ w.r.t reachability problem for single and two players. This proves the decidability of the reachability game problem for bounded context Glitch-free RTA_{RNV} (Theorem 1(3)).

Theorem 3. *Under bounded context restriction RHA_{RNV} is as expressive as RHA_{RV}.*

The non-passing mechanism in RHA_{RNV} gives rise to unboundedly many variables as each time a variable is not-passed, a copy of it is ticking in the context while another copy is instantiated inside the called component with its starting value as 0. However, under bounded context of size at most K we notice that there could be at most K copies of variables. Using this insight, given a bounded context RHA_{RNV} $\mathcal{A}$ with bound K, we show how to construct an RHA_{RV} $\mathcal{B}$ that simulates $\mathcal{A}$. The idea is to use K copies of all the variables and use them in the place of original variables in proper context so as to remove the need of not passing the variable.

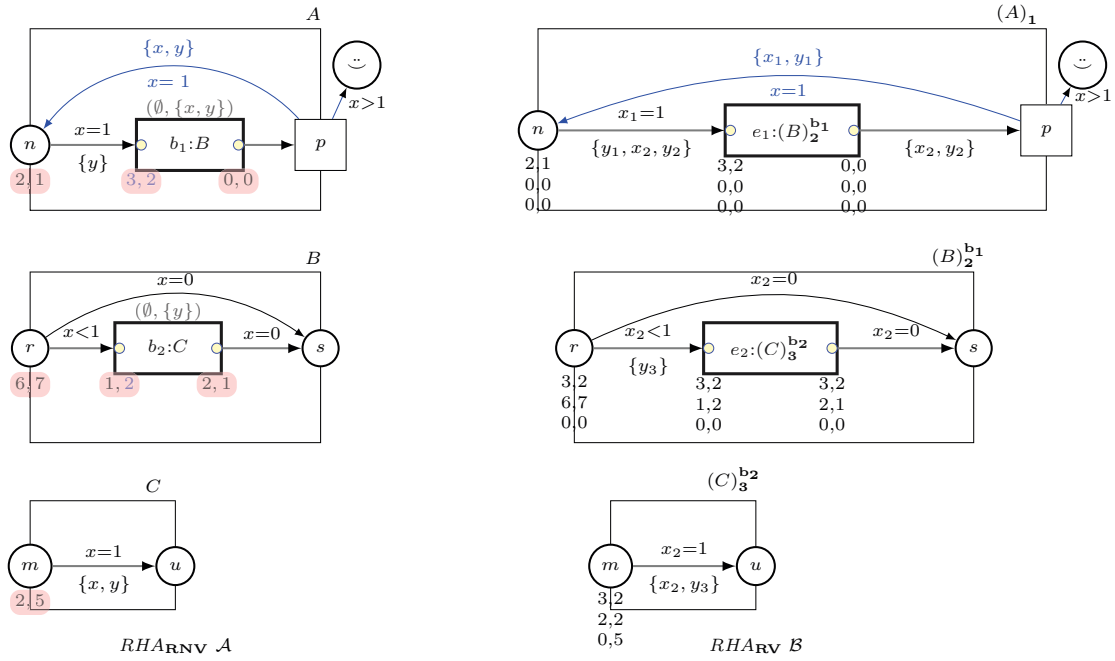

Fig. 4. Removing the Non-passing

Example: We show how this construction works using the RHA_{RNV} $\mathcal{A}$ (Figure 4, left), that has two variables x, y and three components A, B, C. Observe that the context is bounded with $K = 3$. To eliminate the non-passing mechanism, we consider three copies of each variable x, y, namely, x_1, x_2, x_3 and y_1, y_2, y_3 in $\mathcal{B}$ (Figure 4, right). The box b_1 in component A calls the component B, and neither x nor y is passed. In $\mathcal{B}$, the block of variable rates below the locations are such that the first row gives the rates of the first copy of variables, second row gives rates of second copy and so on. For example, at entry node r of $(B)^{\mathbf{b_1}}_{\mathbf{2}}$, the of $x_1, y_1, x_2, y_2, x_3, y_3$ is $3, 2, 6, 7, 0, 0$ respectively. We consider the first copies x_1, y_1 of the variables to be active in $(A)_{\mathbf{1}}$, while x_2, y_2, x_3, y_3 are inactive in $(A)_{\mathbf{1}}$. The component $(A)_{\mathbf{1}}$ in $\mathcal{B}$ represents the component A of $\mathcal{A}$, along with the information that it was the first component, not called by any box. Checking $x = 1$ and resetting y thus amounts to checking $x_1 = 1$ and resetting y_1, the active variables of $(A)_{\mathbf{1}}$. By the non-passing semantics, the values of both x, y will continue to grow at the rates 3,2 respectively in the context till b_1 returns. Component B will use the variables x, y after resetting them to 0. To simulate this in $\mathcal{B}$, we consider the next available copies of x, y in $(A)_{\mathbf{1}}$, namely x_2, y_2 and reset them on the transition from node n to the call port of $e_1 : (B)^{\mathbf{b_1}}_{\mathbf{2}}$. The component $(B)^{\mathbf{b_1}}_{\mathbf{2}}$ in $\mathcal{B}$ represents the component B of $\mathcal{A}$, and carries the information that B was called by box b_1, and the current context level is 2. Then, x_2, y_2 are the active clocks for component $(B)^{\mathbf{b_1}}_{\mathbf{2}}$ taking their rates from corresponding locations in B. Additionally, the rates of x_1, y_1 are set to $3, 2$ throughout $(B)^{\mathbf{b_1}}_{\mathbf{2}}$ and also in components $((C)^{\mathbf{b_2}}_{\mathbf{3}}$ and $(C)^{\mathbf{b_3}}_{\mathbf{3}})$ being called by $(B)^{\mathbf{b_1}}_{\mathbf{2}}$. The notation for $(C)^{\mathbf{b_2}}_{\mathbf{3}}$ and $(C)^{\mathbf{b_3}}_{\mathbf{3}}$ remind the fact that b_2, b_3 respectively call component C, and the context level is 3. Thus, if v_1, v_2 were the values of x_1, y_1 respectively at the call port of box e_1 and a time t elapses during this call (including time elapses in $(B)^{\mathbf{b_1}}_{\mathbf{2}}$, $(C)^{\mathbf{b_2}}_{\mathbf{3}}$ and $(C)^{\mathbf{b_3}}_{\mathbf{3}}$) then at the return port of e_1 the values of x_1, y_1 will be $v_1 + 3t, v_2 + 2t$ respectively. The values of x_1, y_1 indeed agree with the values of x, y at the return port of $b_1 : B$.

Since x_2, y_2 are active in $(B)^{\mathbf{b_1}}_{\mathbf{2}}$, the transition from node r to the call port of $e_2 : (C)^{\mathbf{b_2}}_{\mathbf{3}}$ will check if $x_2 < 1$. If the direct transition is taken from r to s without elapsing time at r, then we check $x_2 = 0$ in $(B)^{\mathbf{b_1}}_{\mathbf{2}}$. An amount of time $t_1 < \frac{1}{6}$ can be spent at r. While this happens, the value of x_1 gets updated to $1 + 3t_1$, while the value of y_1 gets updated to $2t_1$. If r directly goes to s without going through $b_2 : C$, then at s, we have the values $x = 0$ and $y = 0$. At the return port of $b_1 : B$, we then obtain $x = 1$ and $y = 0$. The rates of both x, y are zero at the return port of $b_1 : B$. Thus, in this case, the player 2 node p will choose to go back to node n.

Consider now the case when a time $t_1 < \frac{1}{6}$ is spent at r, and the box $b_2 : C$ was chosen. y is not passed. Note that in $\mathcal{A}$ since $b_2 : C$ does not pass the variable y, the stack entry for y will continue growing at the rate at the call port of $b_2 : C$, that is 2, while the actual variable y is reset and used in C. To simulate this in $\mathcal{B}$, in component $(B)^{\mathbf{b_1}}_{\mathbf{2}}$ where y_2 is active, we reset a new copy y_3 of the variable y on the transition from node r to the call port of $e_2 : (C)^{\mathbf{b_2}}_{\mathbf{3}}$. The active clocks for component $(C)^{\mathbf{b_2}}_{\mathbf{3}}$ at this point is x_2, y_3. The rate for y_2 is set to 2 throughout $(C)^{\mathbf{b_2}}_{\mathbf{3}}$. The transition in $(C)^{\mathbf{b_2}}_{\mathbf{3}}$ from m to u thus will check if $x_2 = 1$, and will reset x_2, y_3. A

time $t_2 = \frac{1-6t_1}{2}$ is spent at m. At the return port of $b_2 : C$, we obtain the new value of x_2 as 0, while y_2 is $7t_1 + 2.t_2$. Also, the value of x_1 is $1 + 3(t_1 + t_2)$, while the value of y_1 is $2(t_1 + t_2)$. Finally, at the return port of $e_1 : (B)^{\mathbf{b_1}}_{\mathbf{2}}$, we reset the active variables x_2, y_2 of $(B)^{\mathbf{b_1}}_{\mathbf{2}}$ since it is not needed in $(A)_{\mathbf{1}}$. At the return port of $b_1 : B$, we have $x > 1$, and hence the player 2 node goes to the happy state.

Thus, as long as the component C is not visited, player 2 will not allow player 1 to reach the happy state. In such situation, the active variables x_1, y_1 of $(A)_{\mathbf{1}}$ are reset on the transition from p to n and the computation continues. $\mathcal{B}$ passes its variables by value and reference only. The new copies of the components and variables in $\mathcal{B}$ eliminate the need for non-passing mechanism.

B. Decidability of RTA_{RN}

We shall now discuss the decidability of RTA_{RN} referring to Theorem 1(1). This result generalizes the results of [4]. We show that given a two player RTA_{RN} $\mathcal{A}$, we can construct an untimed push down game $\mathcal{B}$ which is reachability equivalent to $\mathcal{A}$. Our construction closely follows [4] and in particular exploits the nice idea of shadow regions. We explain the technique through an example given in Figure 5, having 2 clocks and 3 components. Here component A passes clock y by reference and does not pass x, while component B passes x by reference and does not pass y.

We build a push down game whose stack alphabet is a set of regions. A region is represented as a word of sets, where each set consists of three kinds of objects: (i) the plain objects are the clocks x, y as well as the current context ϵ or $(b, x)(b, y)$ for some box b of the RTA_{RN} $\mathcal{A}$, (ii) A special left endmarker $\vdash$ whose value is always 0, except in a pop operation, and (iii) shadow objects $x^\bullet, y^\bullet, \vdash^\bullet$ and $(b, x)^\bullet$ where b is a box. Each of these objects are paired with a natural number, denoting the integral part of its value or age. The sets are arranged in the region in the increasing order of fractional parts. The region R_1 in the Figure 5 represents the initial configuration of the RTA in Figure 5. All the clock values, as well as the shadows have value zero, and we assign value 0 to the symbols $\vdash, \vdash^\bullet$. The value of a shadow clock $x^\bullet$ in a region R is the value of clock x at the time when R was pushed onto the stack. $\vdash^\bullet$ represents the time that has elapsed since the region R was pushed on to the stack, while $(b, x)^\bullet$ represents the current value of the next topmost context (b, x).

The region R_1, given by the green rectangle, is the first region on the stack (Figure 5). The two subrectagles inside the green rectangle denote the sets in the region. The first subrectangle contains all the objects whose fractional parts are 0, while the second subrectangle is empty, since there are no entries with a non-zero fractional part in R_1. The leftmost subrectangle always contains the objects whose fractional part is zero.

From the entry node n_1 of component A in Figure 5, a time of 1.5 is elapsed, resetting the clock y. The stack element R_1 is then updated to the stack element R_2, reflecting the time elapse. The component B is then called, without passing the

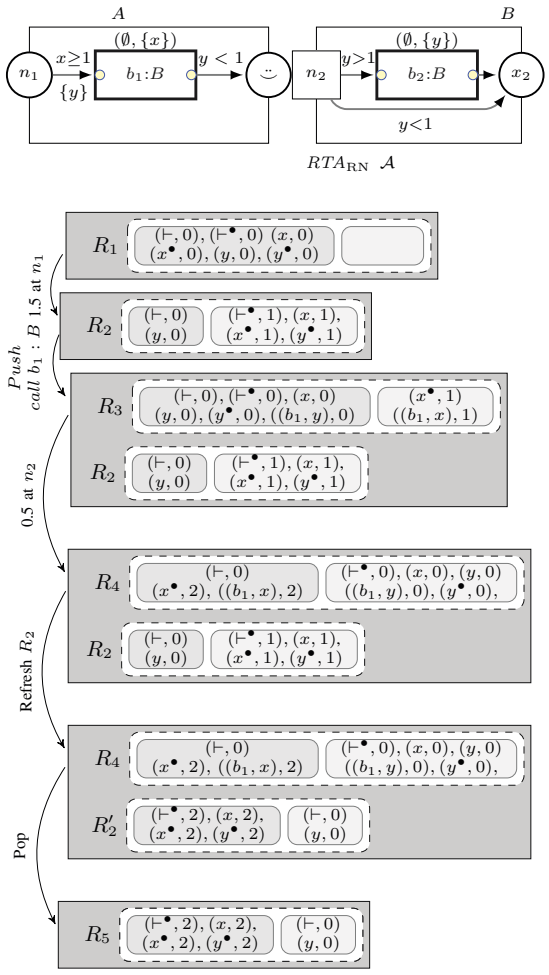

Fig. 5. RTA_{RN} $\mathcal{A}$ and a run in PDA $\mathcal{P}$

clock x. This amounts to resetting the value of x to 0, while the previous value of x, 1.5, will be stored in $x^\bullet$. The new context b_1 gives rise to symbols (b_1, x) and (b_1, y). Since the clock x will continue ticking at the context b_1, we have the value of the object (b_1, x) as 1.5. The value of $y, y^\bullet$ are both 0, since y is passed by reference to B, we also have the object (b_1, y) whose value is 0. The new top-of-the-stack entry region R_3 depicts this. The lower region R_2 is frozen from now on.

This is followed by a time elapse of 0.5 at the node n_2, and this updates the top of stack entry R_3 to R_4. The transition from the entry node of B to the exit node of B is taken. From the exit node of B, we get to the return port of $b_1 : B$ and this amounts to popping context b_1. This amounts to deleting the objects (b_1, x) and (b_1, y) from the topmost stack entry R_4. For this, we first have to take into account, the time that has elapsed between the time R_2 was pushed on to the stack, and R_4 was pushed on to the stack. Elapsing this time in R_2, we must be able to match (i) the values of the plain objects in R_2 with those of the corresponding shadow objects in R_4, (ii) the fractional ordering between the plain objects in R_2 and the

fractional ordering between the corresponding shadow objects in R_4. Elapsing time in R_2 amounts to rotating R_2 suitable number of times. Elapsing < 1 time unit in R_2, we refresh R_2, and obtain R_2'. The above match indeed exists between R_4 and R_2'. Now that the shadow objects of R_3 have matched with the plain objects of R_3', we can indeed take the values of the plain objects of R_4 and merge them with the values of the shadow objects of R_2' as the current correct values of the objects and shadow objects and form a new top-of-the-stack region R_5. However, care has to be taken about the value of the plain object x, since x was not passed during the call to B. The correct value of x is stored in the value of the object (b_1, x), which is 2.

Thus, simulating the return of context b_1, where x was not passed we obtain R_5 by taking the correct value of x from the value of (b_1, x), while the correct value of y is obtained from R_4, and the correct values of $x^\bullet, y^\bullet, \vdash^\bullet$ are taken from R_2'.

To summarize, given a RTA_{RN} $\mathcal{A}$, with clocks $x_1, \ldots, x_m$ and boxes $b_1, \ldots, b_n$, the following are the key steps:

1) We build a stack whose alphabet consists of the regions (encoded as a word of sets) as described above; each set contains the integral parts of the values of the plain objects

$$\{\vdash, x_i, (b_j, x_k) \mid 1 \leq i, k \leq m, 1 \leq j \leq n\}$$

as well as the shadow objects

$$\{\vdash^\bullet, x_i^\bullet, (b_j, x_k)^\bullet \mid 1 \leq i, k \leq m, 1 \leq j \leq n\}.$$

2) A call $b : B$ where some clock x_j is not passed, is simulated by pushing a region R to the stack, such that in R, x_j has value 0, $x_j^\bullet$ has the value val of x_j before the call, and (b, x_j) has the value val. Since x_j continues to tick in the context b, the value of (b, x_j) is initialized as val, subsequent time elapses are reflected in the value of (b, x_j) until the time comes for the context b to be popped, we are sure that at the time of pop, the value of (b, x_j) is the correct value of x_j.
3) A return to the return port of $b : B$ amounts to popping the context b in the RTA. In the PDA, this amounts to popping the topmost region of the stack which contains the objects (b, x). First, the next-top-most region of the stack is refreshed by elapsing time, until the values of the plain objects in this region match with the values of the corresponding shadow objects of the topmost region, including preserving the order of fractional parts. Once this is done, the values of the plain objects of the topmost region and values of the shadow objects of the next-top-most region are merged, taking care that the values of the clock objects x not passed during the call to b, are taken from the values of the object (b, x).

A detailed construction and proof of correctness can be seen later.It is not clear how this technique can be adapted to work for RTA_{RNV} : Consider the case when A calls B passing none of the clocks by value (let x be not passed, and the remaining by reference), and let B call C passing all clocks by value. The time elapsed in C has to be accounted for x, while for the other clocks, it is not needed. Adding time selectively amounts to partially refreshing a region; we consider this as part of our future work.

REFERENCES

[1] http://www.cse.iitb.ac.in/~krishnas/TR15.pdf, 2014.
[2] ACM Turing award citation for E. M. Clarke, E. A. Emerson, and J. Sifakis. http://awards.acm.org/citation.cfm?id=1167964&srt=alpha&alpha=C&aw=140&ao=AMTURING&yr=2007, 2007. For their role in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries.
[3] ACM Kanellakis theory and practice award citation for R. E. Bryant, E. M. Clarke, E. A. Emerson, K. L. Mcmillan. http://awards.acm.org/citation.cfm?id=1167964&srt=all&aw=147&ao=KANELLAK&yr=1998, 1998. For their invention of "symbolic model checking".
[4] P. Abdulla, M. Atig, and J. Stenman. Dense-timed pushdown automata. In *LICS*, pages 35–44, 2012.
[5] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. volume 27, pages 786–818, July 2005.
[6] R. Alur and D. Dill. A theory of timed automata. In *Theor. Comput. Sci.*, volume 126, 1994.
[7] Rajeev Alur. Formal analysis of hierarchical state machines. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 42–66. Springer, 2003.
[8] Rajeev Alur and Mihalis Yannakakis. Model checking of hierarchical state machines. In *ACM SIGSOFT'98*, pages 175–188, 1998.
[9] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.
[10] T. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J-F Raskin, and J. Worrell. Time-bounded reachability for monotonic hybrid automata: Complexity and fixed points. In *ATVA*, pages 55–70, 2013.
[11] Richard Colgren. *Basic MATLAB, Simulink and Stateflow*. AIAA (American Institute of Aeronautics & Ast, 2006.
[12] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
[13] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *JCSS*, 57(1):94 – 124, 1998.
[14] M. Jurdziński and A. Trivedi. Reachability-time games on timed automata. In *Proc. ICALP'07*, volume 4596 of *LNCS*. Springer, 2007.
[15] S. N. Krishna, L. Manasa, and A. Trivedi. Improved undecidability results for reachability games on recursive timed automata. In *GANDALF*, pages 245–259, 2014.
[16] S.N. Krishna, L. Manasa, and A. Trivedi. What is decidable about recursive hybrid automata? In *HSCC*, 2015.
[17] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *In 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 900 of *LNCS*. Springer, 1995.
[18] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
[19] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. In *In Proceedings of the IEEE*, volume 77(1), 1989.
[20] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.
[21] A. Trivedi and D. Wojtczak. Recursive timed automata. In *ATVA*, volume 6252 of *LNCS*, pages 306–324. 2010.
[22] Igor Walukiewicz. Pushdown processes: Games and model checking. In *International Conference on Computer Aided Verification, CAV 1996*, pages 62–74, 1996.