What's Decidable About Recursive Hybrid Automata?

Shankara Narayanan Krishna
IIT Bombay, India
krishnas@cse.iitb.ac.in

Lakshmi Manasa
IIT Bombay, India
manasa@cse.iitb.ac.in

Ashutosh Trivedi
IIT Bombay, India
trivedi@cse.iitb.ac.in

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 recursive timed automata model introduced by Trivedi and Wojtczak and dense-timed pushdown automata by Abdulla, Atig, and Stenman. We study natural reachability problem for recursive hybrid automata. Given the undecidability of this problem for hybrid automata, it is not surprising that the problem remains undecidable without further restrictions. We consider various restrictions of recursive hybrid automata and characterize the boundaries between decidable and undecidable variants.

Categories and Subject Descriptors

D.4.7 [**Organization and Design**]: Real-time systems and embedded systems; B.5.2 [**Design Aids**]: Verification

General Terms

Theory, Verification

Keywords

Recursive State Machines, Hybrid Automata, Simulink

1. INTRODUCTION

Recursive state machines (RSMs), as introduced by Alur, Etessami, and Yannakakis [2], are a variation on various visual notations to represent hierarchical state machines, notably Harel's statecharts [12] and Object Management Group supported UML diagrams [18], that permits recursion while disallowing concurrency. RSMs closely correspond [2] to pushdown systems [8], context-free grammars, and Boolean programs [7], and provide a natural specification and verification framework to reason with sequential programs with recursive procedure calls. Hybrid automata [5, 3] extend finite state machines with continuous variables that permit a natural modeling of hybrid systems. In a hybrid automaton the variables continuously flow according to a given set of ordinary differential equations within each discrete states, while they are allowed to have discontinuous jumps during transitions between states that are guarded by constraints over variables. In this paper we study *recursive hybrid automata* (RHA), that generalize both the recursive state machines as well as the hybrid automata, as a potential candidate for defining the formal semantics of popular model-based design frameworks such as Stateflow/Simulink (`http://www.mathworks.in/products/stateflow/`) and Scicos (`http://www.scicos.org`). In this paper we are concerned with the reachability problem for recursive hybrid automata.

The reachability problem for recursive state machines is known to be decidable in polynomial time [2, 11]. On the other hand, the reachability problem for hybrid automata even with extremely simple dynamics—such as variables with two constant slopes [13] and 3-dimensional piecewise constant derivative systems [6]—gets undecidable. There are two main recipes to recover decidability in the context of hybrid automata. The first one is to restrict the continuous behavior, for example timed automata [5], and initialized rectangular hybrid automata [13]; while the second option is to restrict the discrete behavior, for instance o-minimal hybrid automata [16] and hybrid automata with strong resets. Recently, Ouaknine and Worrell [17] added another recipe to this mix: they proposed a time-bounded theory of real-time verification by claiming that restriction to bounded-time recovers decidability for several key decision problem related to real-time verification. In support of this theory, the list of undecidable problems recently shown decidable under time-bounded restriction is rather impressive: language inclusion for timed automata, emptiness problem for alternating timed automata, and emptiness problem for rectangular hybrid automata [9].

The main objective of our study was to recover decidability for general recursive hybrid automata—or at least for hierarchical hybrid automata—under time-bounded restriction in order to provide an appealing verification framework for various model-based design frameworks. Unfortunately,

HSCC '15, April 14 - 16, 2015, Seattle, WA, USA

http://dx.doi.org/10.1145/2728606.2728624

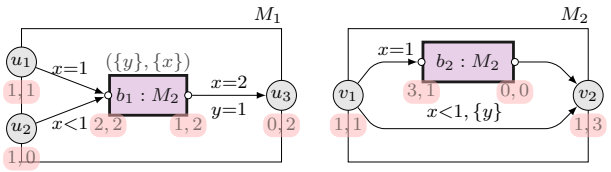


Figure 1: A recursive hybrid automaton with two components and two variables.

however, we answer this question in negative by showing that time-bounded reachability problem stays undecidable for hierarchical timed automata with 7 variables (improved to 5 variables in [14]). On a positive side, we characterize several restrictions of our model to recover decidable subclasses.

As a blueprint for the definition of the recursive hybrid automata, we studied recursive timed automata introduced by Trivedi and Wojtczak [19] and dense-timed pushdown automata introduced by Abdulla, Atig, Stenman [1]. A *recursive timed automaton* (RTA) is a finite collection of components where each component is a timed automaton that, in addition to making transitions between various states, can have transitions to "boxes" that are mapped to other components modeling a potentially recursive call to a subroutine. During such invocation a limited information can be passed through clock values from the "caller" component to the "called" component via two different mechanisms:

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

Trivedi and Wojtczak [19] showed that the reachability and termination (reachability with empty calling context) problem is undecidable for RTAs with three or more clocks, while it is decidabe 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. Unlike the RTA model, where it is compulsory to pass all the clocks at every invocation with either mechanism, Abdulla, Atig, and Stenman [1] studied a related model called timed pushdown automata (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.

In this paper, we consider a general class of recursive hybrid automata, where the variables can be passed either by value (V) or reference (R) as in [19], or not passed at all (N), as in [1]. The resulting model is the most general one combining features from both RTA and TPDA. We explore the decidability of the reachability under the time-bounded and general semantics.

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.*

We classify Recursive Hybrid Automata based on the variable passing mechanisms in the following manner. We write RHA_R and RTA_R 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.

The key contributions of the paper are summarized in Table 1. We show 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. 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 ≤ 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 (Theorem 3.3) or pass-by-value+pass-by-reference with 5 or more clocks [14].

For a survey of models related to RTA and TPDA we refer the reader to [19]and [1]. Two special kinds of RSMs with restricted recursion are hierarchical RSMs and bounded stack RSMs [10]. [10] gives efficient algorithms for the reachability analysis of hierarchical and bounded stack RSMs, and algorithms for the latter might be useful in the analysis of programs without infinite recursion. The language theory of bounded context recursion has been studied recently [15]. Hierarchical hybrid systems studied by Alur et al. [4] are a restriction of recursive hybrid automata where recursion is disallowed but concurrency is allowed. A number of case-studies with the tool CHARON [4] demonstrate the benefit of hierarchical modeling of hybrid systems.

	Recursive Timed Automata		Recursive Hybrid Automata	
	TUB	TB	TUB	TB
R	D [19]	D [19]	**U** (3sw) [13] **D** (2sw) Thm 2.3	**D** (Bd. Cnxt) Thm 2.4
N	**D** Thm 2.2	**D** Thm 2.2	**U** (3sw)[13]	**D**(Bd.Cnxt) Thm 2.4
V	D [19]	D [19]	**U**(2cl+1sw) Thm 3.2 **D**(2sw)Thm 2.3	**U**(14 sw)Thm 3.4 **D**(2 sw) Thm 2.3
Unrestricted RN	**D** Thm 2.2	**D** Thm 2.2	**U** (3sw)[13]	**D**(Bd. Cnxt) Thm 2.4
Glitch-free RV	D [19]	D [19]	**U**(2cl+1sw)Thm 3.2 **D**(2sw) Thm 2.3	**U**(14 sw)Thm 3.4 **D**(2 sw) Thm 2.3
Unrestricted RV	U ($\geq$ 3 clocks) [19]	**U**(5 cl)[14]	**U**(2sw)Thm 3.1	**U**(5 cl)[14]
Glitch-free NV	**D**(Bd. Cnxt) Thm 2.1	**D** (Bd. Cnxt) Thm 2.1	**U**(2cl+1sw) Thm 3.2	**U**(14 sw) Thm 3.4
Unrestricted NV	**U** (3cl) Thm 3.5	**U** (7cl) Thm 3.3	**U** (3cl) Thm 3.5	**U** (7cl)Thm 3.3
Glitch-free RNV	**D** (Bd. Cnxt) Thm 2.1	**D** (Bd. Cnxt) Thm 2.1	**U**(2cl+1sw)Thm 3.2	**U**(14 sw)Thm 3.4
Unrestricted RNV	U ($\geq$ 3 clocks) [19]	**U**(5 cl)[14]	**U**(2sw)Thm 3.1	**U**(5 cl)[14]

Table 1: Our contributions are emphasized in bold letters along with reference to the theorems they follow from. We write TUB for unbounded time as opposed to TB for bounded time, cl stands for clock variables, sw for stopwatch variables, U for undecidability and D for decidability. Additionally, Bd. Cnxt stands for bounded context restriction. All of our undecidability results hold under bounded context restriction.

2. PRELIMINARIES

A *labeled transition system* (LTS) is a tuple $\mathcal{L} = (S, A, X)$ where S is the set of *states*, A is the set of *actions*, and $X : S \times A \to S$ is the *transition function*. We say that an LTS $\mathcal{L}$ is *finite* (*discrete*) if both S and A are finite (countable). We 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 $\langle s_0, a_1, s_1, \ldots \rangle$ such that (s_i, a_{i+1}, s_{i+1}) is a transition of $\mathcal{L}$ for all $i \geq 0$. Given a state $s \in S$ and a set of final states $F \subseteq S$ we say that a final state is reachable from s_0 if there is a run r such that $\exists i < \infty, s_i \in F$. Given an LTS, an initial state, and a set of final states, the *reachability problem* for LTS is to decide whether a final state is reachable from the given initial state.

Definition 1. *A* recursive state machine *[2] is a tuple $(\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k)$ of components, where each component $\mathcal{M}_i = (N_i, \mathrm{EN}_i, \mathrm{EX}_i, B_i, Y_i, A_i, X_i)$ 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. We associate a set of* call ports *$Call(b)$ and return ports $Ret(b)$ to each box $b \in B_i$:*

$$Call(b) = \{(b, en) : en \in \mathrm{EN}_{Y_i(b)}\} \quad \text{and}$$
$$Ret(b) = \{(b, ex) : ex \in \mathrm{EX}_{Y_i(b)}\}.$$

  Let $Call_i = \cup_{b \in B_i} Call(b)$ and $Ret_i = \cup_{b \in B_i} Ret(b)$ be the set of call and return ports of component $\mathcal{M}_i$. We define the set of vertices Q_i of component $\mathcal{M}_i$ as the union of the set of nodes, call ports and return ports, i.e. $Q_i = N_i \cup Call_i \cup Ret_i$;
- *A_i is a finite set of* actions*; and $X_i : Q_i \times A_i \to Q_i$ is the transition function with a condition that call ports and exit nodes do not have any outgoing transitions.*

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

An example of a RSM is shown in Figure 2. An execution of a RSM begins at the entry node of some component and depending upon the sequence of input actions the state evolves naturally like a labeled transition system. However, when the execution reaches an entry port of a box, this box is stored on a stack of pending calls, and the execution continues naturally from the corresponding entry node of the component mapped to that box. When an exit node of a component is encountered, and if the stack of pending calls is empty then the run terminates; otherwise, it pops the box from the top of the stack and jumps to the exit port of the just popped box corresponding to the exit of the component.

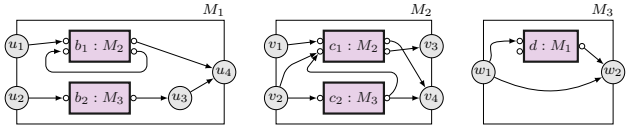


Figure 2: Example recursive state machine [2]

We formalize the semantics of a RSM using a discrete LTS, whose states are pairs of current vertex and calling context.

Definition 2. *The semantics of an RSM $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k)$ where each component $\mathcal{M}_i$ is $(N_i, En_i, Ex_i, B_i, Y_i, A_i, X_i)$, is the discrete labelled transition system $[\![\mathcal{M}]\!] = (S_{\mathcal{M}}, A_{\mathcal{M}}, X_{\mathcal{M}})$ where:*

- *$S_{\mathcal{M}} \subseteq B^* \times Q$ is the set of states;*
- *$A_{\mathcal{M}} = \cup_{i=1}^{k} A_i$ is the set of actions;*
- *$X_{\mathcal{M}} : S_{\mathcal{M}} \times A_{\mathcal{M}} \to S_{\mathcal{M}}$ is the transition 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. $q = ex \in \mathrm{EX}$ and $s' = (\langle \kappa' \rangle, (b, ex))$ where $(b, ex) \in Ret(b)$ and $\kappa = (\kappa', b)$;*
  3. *the vertex q is any other kind of vertex, and $s' = (\langle \kappa \rangle, q')$ and $q' \in X(q, a)$.*

Given $\mathcal{M}$ and a subset $Q' \subseteq Q$ of its nodes we define the set $[\![Q']\!]_{\mathcal{M}}$ as the set $\{(\langle \kappa \rangle, v') : \kappa \in B^* \text{ and } v' \in Q'\}$. Given a recursive state machine $\mathcal{M}$, an initial node v, and a set of *final vertices* $F \subseteq Q$ the *reachability problem* on $\mathcal{M}$ is defined as the reachability problem on the LTS $[\![\mathcal{M}]\!]$ with the initial state $(\langle \varepsilon \rangle, v)$ and final states $[\![F]\!]$. It is known [2] that the reachability problem for recursive state machines can be solved in polynomial time.

3. RECURSIVE HYBRID AUTOMATA

Let $\mathcal{X}$ be a finite set of real-valued 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 s.t. $\mathbf{0}(x) = 0$ 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 $\mathrm{rect}(\mathcal{X})$ for the set of rectangular constraints over $\mathcal{X}$.

Definition 3 (Recursive Hybrid Automata). *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, $F_i)$ *is such that:*

- N_i *is a finite set of* nodes *including a distinguished set* EN_i *of* entry nodes *and a set* EX_i *of* exit nodes *such that* EX_i *and* EN_i *are disjoint sets;*
- B_i *is a finite set of* boxes*;*
- $Y_i : B_i \to \{1, 2, \ldots, k\}$ *is a mapping that assigns every box to a component. (Call ports* $Call(b)$ *and return ports* $Ret(b)$ *of a box* $b \in B_i$*, and call ports* $Call_i$ *and return ports* Ret_i *of a component* $\mathcal{H}_i$ *are defined as before. We set* $Q_i = N_i \cup Call_i \cup Ret_i$ *and refer to this set as the set of nodes of* $\mathcal{H}_i$*. Let* $Q = \bigcup_{i=1}^{n} Q_i$*.)*
- A_i *is a finite set of* actions.
- $X_i \subseteq Q_i \times A_i \times rect(\mathcal{X}) \times 2^{\mathcal{X}} \times Q_i$ *is the* transition relation *with a condition that call ports and exit nodes do not have any outgoing transitions.*
- 1. $\mathsf{PV}_i : B_i \to 2^{\mathcal{X}}$ *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;*
  2. $\mathsf{PR}_i : B_i \to 2^{\mathcal{X}}$ *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;*
  3. $\mathsf{PN}_i : B_i \to 2^{\mathcal{X}}$ *is non-passing mapping that assings to every box the set of variables not passed to the box.*
  4. *For every box* $b \in B_i$*, the sets* $\mathsf{PV}(b)$*,* $\mathsf{PR}(b)$ *and* $\mathsf{PN}(b)$ *form a partition of* $\mathcal{X}$*.*
- $F_i : Q_i \to \mathbb{N}^{|\mathcal{X}|}$ *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 $(N, B, Y, Q, \mathsf{PV}, \mathsf{PR}, \mathsf{PN}$*, etc.) to for union over all components.*

A *configuration* 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 $\mathcal{X}$. The sequence $\langle\kappa\rangle \in (B \times \mathbb{R}^{|\mathcal{X}|})^*$ denotes the stack of pending recursive calls and the valuation of all the variables at the moment that call was made, and we refer to this sequence as the context of the configuration.

Technically, it suffices to store the valuation of variables passed by value 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 $1 \leq i \leq n$

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

Note that for all $1 \leq i \leq n, \forall x \in \mathsf{PV}(b_i) \cup \mathsf{PR}(b_i), \nu_i'(x) = \nu_i(x)$ i.e; the valuations of variables passed-by-reference and passed-by-value are the same in $\langle\kappa\rangle$ and $\langle\kappa'\rangle$.

Informally, the behaviour of an RHA is as follows. In configuration $(\langle\kappa\rangle, q, \nu)$ time passes before an available action is triggered, after which a discrete transition occurs. A transition is (q, a, φ, C, q'), with $\varphi \in \mathrm{rect}(\mathcal{X})$ 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.

Definition 4 (RHA semantics). *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* $\mathcal{H}_i = (N_i, \mathrm{EN}_i, \mathrm{EX}_i, B_i, Y_i, A_i, X_i, \mathsf{PV}_i, \mathsf{PR}_i, \mathsf{PN}_i, F_i)$*. 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,*
- $A_{\mathcal{H}} = \mathbb{R}_{\oplus} \times A$ *is the set of* timed actions*, where* $\mathbb{R}_{\oplus}$ *is the set of non-negative reals;*
- $X_{\mathcal{H}} : S_{\mathcal{H}} \times A_{\mathcal{H}} \to S_{\mathcal{H}}$ *is the transition function such that for* $(\langle\kappa\rangle, q, \nu) \in S_{\mathcal{H}}$ *and* $(t, a) \in A_{\mathcal{H}}$*, we have* $(\langle\kappa'\rangle, q', \nu') = X_{\mathcal{H}}((\langle\kappa\rangle, q, \nu), (t, a))$ *if and only if the following condition holds:*
  1. *if the location* q *is a call port, i.e.* $q = (b, en) \in Call$ *then* $t = 0$*, the context* $\langle\kappa'\rangle = \langle\kappa, (b, \nu)\rangle$*,* $q' = en$*, and* $\nu' = \nu[\mathsf{PN}(b) := 0]$*.*
  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* $(b, ex) \in Ret(b)$*, then* $t = 0$*;* $\langle\kappa'\rangle = \langle\kappa''\rangle$*;* $q'{=}(b, ex)$*; and* $\nu'{=}\nu[\mathsf{PV}(b), \mathsf{PN}(b){:=}\nu'']$*.*
  3. *if location* q *is any other kind of location, then* $\langle\kappa'\rangle = \langle\kappa\rangle{+}t$*,* $(q, a, \varphi, C, q') \in X$*,* $\nu{+}F(q){\cdot}t \in [\![\varphi]\!]$ *and* $\nu' = (\nu + F(q){\cdot}t)[C := \mathbf{0}]$*.*

Subclasses of RHA. We classify Recursive Hybrid Automata based on the variable passing mechanisms as follows :

1. 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.
2. 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 on stack keep growing with the rates of their called points.

3. 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 $\mathsf{PV}(b) = \mathcal{X}$ or $\mathsf{PV}(b) = \emptyset$. 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 configuration $(\langle\kappa\rangle, q, \nu)$, $|\kappa| \leq K$. 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.

Reachability Problem. 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 LTS $[\![\mathcal{H}]\!]$ starting from $(\langle\varepsilon\rangle, q, \nu)$ to some configuration in $[\![F]\!]_{\mathcal{H}}$.

Given a run $r = \langle s_0, (t_1, a_1), s_1, (t_2, a_2), \cdots, s_n\rangle$ 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 $[\![\mathcal{H}]\!]$ starting from the initial configuration $(\langle\varepsilon\rangle, q, \nu)$ to some configuration in $[\![F]\!]_{\mathcal{H}}$ such that $\mathsf{time}(r) \leq T$.

The following theorems summarize our key contributions.

Theorem 1. *Under bounded context restriction RHA_{RNV} is as expressive as RHA_{RV}.*

Theorem 2 (Decidability). *The reachability problems is decidable for the following fragments of RHA:*
(1) Bounded-context glitch-free RTA_{RNV},
(2) RTA_{RN},
(3) Glitch-free RHA_{RV} with 2 stopwatches,
(4) Bounded-context RHA_{RN} under bounded-time.
Moreover, the reachability problems for (1)-(3) are ExpTime*-complete while (4) is* NExpTime*-complete.*

Theorem 2.3 is due to region abstraction with 2 stopwatches. We shall provide the intuitive proof sketches in Section 5 and the detailed proofs are available in [14].

Theorem 3 (Undecidability). *The reachability problem is undecidable for*
(1) Unrestricted RHA_{RV} with 2 stopwatches.
(2) RHA_V with 2 clocks and 1 stopwatch.
(3) Unrestricted RTA_{NV} with 7 clocks under bounded time.
(4) RHA_V with 14 stopwatches under bounded time.
(5) RTA_{NV} with 3 clocks.
Moreover, all of these results hold even under the hierarchical, bounded context restrictions.

4. PROOF OF THEOREM 1

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.

Example. We show how this construction works using the RHA_{RNV} $\mathcal{A}$ (Figure 3,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 3,right). The box b_1 in component A calls the component B, and neither x nor y is passed. 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 componnet 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 callport of $e_2 : (C)^{\mathbf{b_2}}_{\mathbf{3}}$ will check if $x_2 < 1$. An amount of time $t_1 < \frac{1}{6}$ is 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 $2t_1$.

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 $2(t_1 + t_2)$. Also, since the new copy y_3 is not needed on return, we reset it

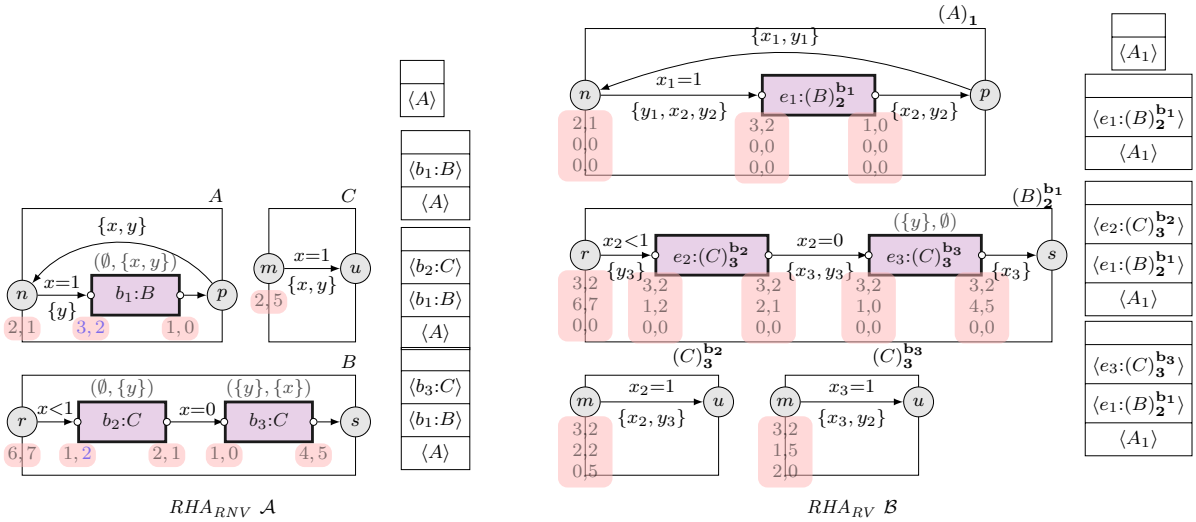


Figure 3: RHA_{RNV} $\mathcal{A}$ **to** RHA_{RV} $\mathcal{B}$ **reduction under bounded context assumption**

to zero on the transition from the return port of $e_2 : (C)_3^{b_2}$. The transition from the return port of $e_2 : (C)_3^{b_2}$ to the call port of $e_3 : (C)_3^{b_3}$ will check if the active variable x_2 of component $(B)_2^{b_1}$ is zero, which it is already. In $\mathcal{A}$ the box b_3 calls component C again, passing y by value, and not passing x. In $\mathcal{B}$, since the active variables of $(B)_2^{b_1}$ are x_2, y_2, and x is not passed, we use a fresh copy x_3 of the variable x, and reset it on the transition from the return port of $e_2 : (C)_3^{b_2}$ to the call port of $e_3 : (C)_3^{b_3}$. The value of y_2 remains unchanged when we return to the return port of $e_3 : (C)_3^{b_3}$, and x_2 will continue growing at the rate of 1 inside $(C)_3^{b_3}$.

The active variables of $(C)_3^{b_3}$ this time are thus x_3, y_2. A time $t_3 = \frac{1}{2}$ is spent at node m, and hence on return to the return port of $e_3 : (C)_3^{b_3}$, we obtain $x_3 = 0, x_2 = t_3, y_2 = 7t_1 + 2.t_2$. Also, we have at this point $x_1 = 1+3(t_1+t_2+t_3)$, $y_1 = 2(t_1 + t_2 + t_3)$. On return, since we no longer need the new copy x_3, we reset it to zero on the transition from the return port of $e_3 : (C)_3^{b_3}$ to the exit node of $(B)_2^{b_1}$. Finally, at the return port of $e_1 : (B)_2^{b_1}$, we reset the active variables x_2, y_2 of $(B)_2^{b_1}$ since it is not needed in $(A)_1$. The active variables x_1, y_1 of $(A)_1$ 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.

Details of the Reduction. We summarize the salient points in the construction below: For ease of explanation, we assume we have only 3 variables x, y, z. The components in the constructed RHA_{RV} $\mathcal{B}$ are of the form $(H_j)_m^b$, where m indicates the current stack(or context) depth, while b indicates that the box b has called the component H_j in $\mathcal{A}$. To begin with, we just have the initial component H_1 in the stack, with set of active variables, all with index 1. The first time a box b in H_1 calls a component H_j, not passing a variable, say z, we obtain the component $(H_j)_2^b$ in the stack as the topmost element, and the active variables of $(H_j)_2^b$ are z_2 and all variables other than z with index 1 (that is, x_1, y_1). If there is a box c in $(H_j)_2^b$ which calls H_g, not passing say, y, z, then we obtain the topmost stack element as $(H_g)_3^c$, and the active variables of component $(H_g)_3^c$ are z_3, y_2 and all other variables with index 1 (that is, x_1).

Assume in general, that the current component (topmost stack element) is some $(H_d)_K^e$, and the stack element below it is $(H_o)_{K-1}^f$. Further, assume the box e in $(H_o)_{K-1}^f$ called component H_d not passing variables x, y. Let the active variables of $(H_d)_K^e$ be x_K, y_l, z_j with $l, j < K$. On reaching the exit node of $(H_d)_K^e$, we will have the topmost element of the stack as $(H_o)_{K-1}^f$, with active variables x_{K-1}, y_{l-1}, z_j. Now if, from the return port of the box e in $(H_o)_{K-1}^f$, we eventually reach the call port of a box e' (without visiting any boxes in between) which calls a component H_s, not passing x, z, then we obtain as topmost element of the stack, $(H_s)_K^{e'}$ with active variables x_K, y_{l-1}, z_{j+1}. It is clear that we will reuse the variables x_i, y_i, z_i, for $1 \leq i \leq K$ at each level $\leq K$.

To handle cases when the given RHA_{RNV} has a stack depth $> K$, in the constructed RHA_{RV} $\mathcal{B}$, add a sink component $Sink$ with a single entry node and a single exit node, with no outgoing transitions from the entry node. If in $\mathcal{B}$, the topmost stack element is some $(H_d)_K^e$, and if some box in $(H_d)_K^e$ calls a further component, then we push $(Sink)_{K+1}$ as the topmost stack element in the constructed RHA_{RV} $\mathcal{B}$; in this case $\mathcal{B}$ has a depth $K + 1$, and is stuck at the entry node of $Sink$.

5. PROOF OF THEOREM 2

In this section, we discuss decidable subclasses of RHA as discussed in Theorem 2.

5.1 Bounded context and glitch-free RTA_{RNV}

Given a bounded context and glitch-free RTA_{RNV} $\mathcal{A}$, we first obtain a reachability equivalent glitch-free RTA_{RV} $\mathcal{B}$ using the technique given in section 4. The decidability of the reachability problem now follows from the decidability of glitch-free RTA_{RV} via the region abstraction [19].

5.2 RTA_{RN}

This result generalizes the results of [1]. We show that

given an RTA_{RN} $\mathcal{A}$, we can construct an untimed PDA $\mathcal{B}$ which is reachability equivalent to $\mathcal{A}$. Our construction closely follows [1] and in particular exploits the nice idea of shadow regions. We explain the technique through an example given in Figure 4, 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 PDA 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 4 represents the initial configuration of the RTA in Figure 4. 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 4). 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 4, 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 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. Next, the component C is called, not passing y. This amounts to resetting the value of y to 0, while y continues to tick in the context b_2. This call amounts to pushing a new entry onto the stack, with x and $x^\bullet$ having the value 0.5, y having value 0, and $y^\bullet$ having the previous value 0.5. The objects $(b_1, x), (b_1, y)$ corresponding to the previous context are stored in the shadow form. Thus, we have $(b_1, x)^\bullet$ with value 2, and $(b_1, y)^\bullet$ with value 0.5. The new context b_2 corresponds to the introduction of new objects (b_2, x) and (b_2, y), where (b_2, y) has age 0.5, and (b_2, x) has age 0. The entry R_5 depicts this. The entries R_2, R_4 are frozen.

This is followed by a time elapse of 1 at the node n_3 and this updates the top of the stack entry R_5 to R_6. From the exit node of C, we get to the return port of $b_2 : C$, and this amounts to popping context b_2. This amounts to deleting

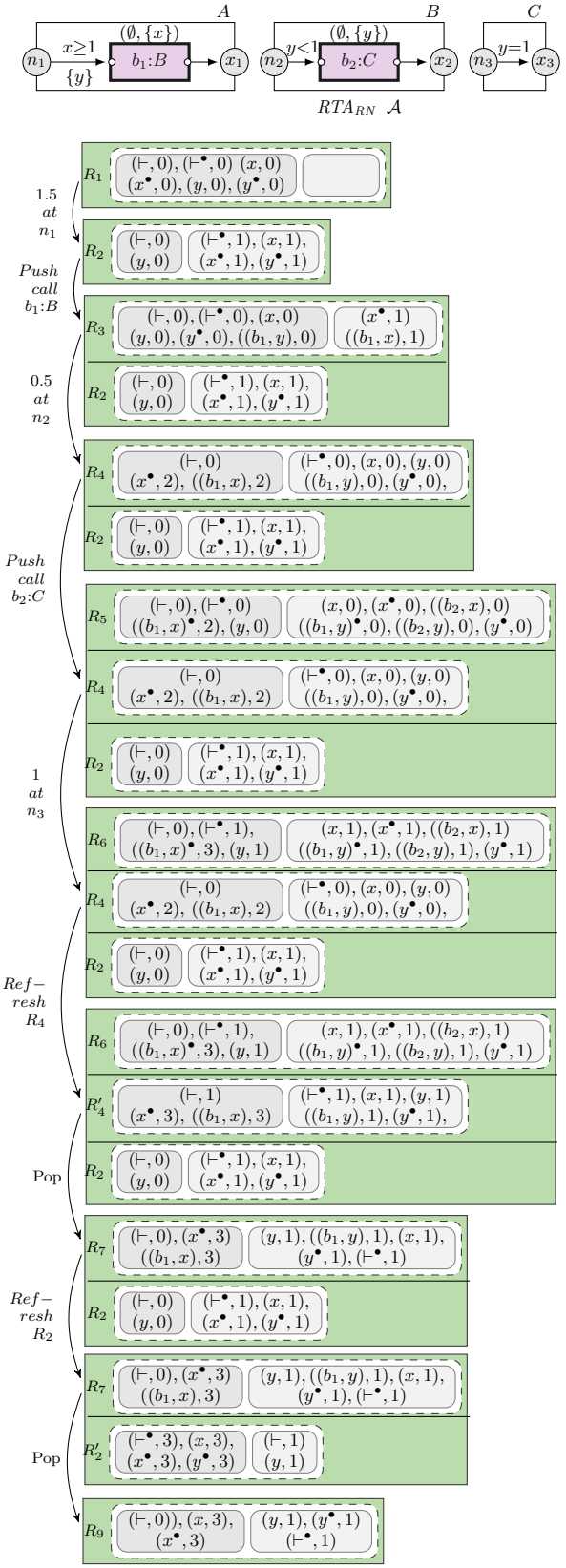


Figure 4: RTA_{RN} $\mathcal{A}$ **and a run in the PDA** $\mathcal{P}$

the objects (b_2, x) and (b_2, y) from the topmost stack entry R_6. For this, we first have to take into account, the time that has elapsed between the time R_4 was pushed on to the stack, and R_6 was pushed on to the stack. Elapsing this time in R_4, we must be able to match (i) the values of the plain objects in R_4 with those of the corresponding shadow objects in R_6, (ii) the fractional ordering between the plain objects in R_4 and the fractional ordering between the corresponding shadow objects in R_6. Elapsing time in R_4 amounts to rotating R_4 suitable number of times. Elapsing 1 time unit in R_4, we refresh R_4, and obtain R'_4. The above match indeed exists between R_6 and R'_4. Now that the shadow objects of R_6 have matched with the plain objects of R'_4, we can indeed take the values of the plain objects of R_6 and merge them with the values of the shadow objects of R'_4 as the current correct values of the objects and shadow objects and form a new top-of-the-stack region R_7. However, care has to be taken about the value of the plain object y, since y was not passed during the call to C. The correct value of y is stored in the value of the object (b_2, y), which is 1.5. Thus, simulating the return of context b_2, where y was not passed we obtain R_7 by taking the correct value of y from the value of (b_2, y), while the correct value of $x, (b_1, x), (b_1, y)$ are obtained from R_6, and the correct values of $x^\bullet, y^\bullet, \vdash^\bullet$ are taken from R'_4. The objects (b_1, x) and (b_1, y) correspond to the current context, and thus are stored in plain form, retaining their values from the shadow form in R_6. With this update, and popping the entries $(b_2, x), (b_2, y)$, we merge R'_4 and R_6 obtaining the new top of stack element R_7.

The remaining step is now to return to the return port of $b_1 : B$. For this, we must rotate R_2 by elapsing 1.5 units of time, and merge with R_7 as described above to obtain R_9. Note that in this step, since we are simulating the return of context b_1, where x was not passed, the correct value of x must be taken from that of (b_1, x).

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).

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.

5.3 Time-bounded reachability for bounded-context RHA_{RN}

Given a bounded context RHA_{RN} $\mathcal{A}$, we first apply the construction given in Section 4 to obtain a reachability equivalent bounded context RHA_R $\mathcal{B}$. We show that the time-bounded reachability problem is decidable for RHA_R $\mathcal{B}$.

The time bounded reachability was shown to be decidable for hybrid automata with no negative rates and no diagonal constraints [9]. The main idea in [9] is that if there is a run ρ between two configurations (q_1, ν_1) and (q_2, ν_2) in a hybrid automata H such that $\mathsf{duration}(\rho) \leq T$ (called $T-$time bounded run), then there exists a *contracted* run ρ' between the same configurations, such that $\mathsf{duration}(\rho') \leq T$, length of ρ' is atmost C, a constant exponential in H and linear in T, and is dependent on $rmax$ (maximal rate in H) and $cmax$ (largest constant in the constraints of H). The construction of ρ' from ρ relies on a *contraction* operator. This operator identifies positions $i < j$ in ρ, such that all locations between i and j are visited before i in ρ and locations $l_i = l_j$ and $e_{i+1} = e_{j+1}$ the outgoing edges from l_i and l_j respectively. The operator then deletes all the locations $i+1, \ldots, j$ and adds their time to the other occurrences before i. It then connects $l_i \overset{e_{j+1}}{\rightarrow} l_{j+1}$ with sum of time delays accompanying e_{i+1} and e_{j+1}. This operator is used as many times as required until a fixpoint is reached. Care should be taken to ensure that the *contracted run* is a valid run : it should satisfy the constraints. To ensure this, the run ρ is first carefully partitioned into exponentially many pieces, so that contracting the pieces and concatenating them yields a valid run.

We shall now describe briefly, how the contraction is carefully applied to partitions of ρ so as to respect the constraints and yield a valid run. Firstly, to help track whether the valuations resulting from contraction satisfy constraints, the region information is stored in the locations to form another hybrid automaton $R(H)$. Given $cmax$, the set of regions is $\{(a-1, a), [a, a] | a \in \{1, \cdots cmax\}\} \cup \{\mathbf{0}^=, \mathbf{0}^+, (cmax, +\infty)\}$. It differs from the classical region notion due to lack of fractional part ordering and special treatment of valuations which are 0. $R(H)$ checks whether a variable x never changes from 0 before the next transition, or if it becomes > 0 before the next transition. This helps bound the number of sub-runs that are constructed later, and prevents the contraction operator from merging locations where x remains 0 with those where x becomes > 0. The construction ensures that H admits a run between two states of duration T

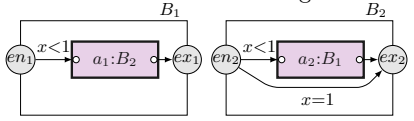
iff $R(H)$ admits a run between the same states and for the same time T.

As the rest of the automaton is untouched, the equivalent of run ρ in $R(H)$ is a run same as ρ, but having region information along with locations. Let us continue to call the run in $R(H)$ as ρ. ρ is called a *type-0 run.* ρ is chopped into fragments of duration $\leq \frac{1}{rmax}$, each of which is called a *type-1* run. There will be atmost $T.rmax + 1$ type-1 runs. Additionally, as $rmax$ is the maximal rate of growth of any variable, a variable changes its region at most 3 times such that, when starting in $(b, b+1)$ region, growing through $[b+1, b+1], (b+1, b+2)$, gets reset and stays in $[0, 1)$. Each type-1 run is further split into type-2 runs based on region changes which is at most 3 times per variable. Thus each type-1 run is split into atmost $3.|\mathcal{X}|$ type-2 runs. Respecting region changes ensures that constraints continue to be satisfied post contraction. Type-2 runs are again split into type-3 runs based on the first and last reset of a variable. This is to enable concatenation of consecutive contracted fragments by ensuring the valuations in the start configuration and end configuration of each fragment are compatible with their neighbors. Each type-2 run is split into atmost $2.|\mathcal{X}| + 1$ type-3 runs. The contraction is applied to type-3 runs, removing second occurrences of loops. Hence, each contracted type-3 run will be at most $|Loc'|^2+1$ long (Lemma 7 of [9]), where Loc' is the set of locations of $R(H)$. Note that $|Loc'| = |Loc|.(2.cmax + 1)^{|\mathcal{X}|}$ where Loc is the set of locations of H. After concatenating these contracted type-3 runs, we get contracted type-2 runs with the same start and end states. These contracted type-2 runs are then concatenated to obtain a run ρ' of that has the same start and end states as ρ, $\mathsf{duration}(\rho') = \mathsf{duration}(\rho)$ and $|\rho'| \leq C = 24.(T.rmax + 1).|\mathcal{X}|^2.|Loc|^2.(2.cmax + 1)^{2.|\mathcal{X}|}$. To solve time-bounded reachability, we nondeterministically guess a run of length at most C, and solve an LP to check if there are time delays and valuations for each step to make the run feasible.

Theorem 4. *Time-bounded reachability is decidable for bounded-context RHA_R.*

Proof. We adapt the contraction operator of [9] and make it *context-sensitive*. This operator respects the context of the RHA configurations. We split a given run ρ of the RHA into type-3 runs exactly as done for hybrid automata [9]. While applying contraction, we identify the second occurence of a loop by matching not just the location, but by the (context, location) pair in the configurations. The context matching ensures that we do not alter the sequence of recursive calls made in the contracted run, thus maintaining validity w.r.t recursion. The second occurences of a (location,context) pair are then deleted and the time delays are added to their first occurence in the loop. We match the (context, location) pair as opposed to only locations in [9]. Thus, the size of the contracted type-3 run is atmost $(\alpha.|Q|.(2.cmax + 1)^{|\mathcal{X}|})^2 + 1$, where $Q = \bigcup_{i=1}^{n} Q_i$ is the union of the set of vertices Q_i of all the n components of the RHA, $|Q|.(2.cmax + 1)^{|\mathcal{X}|}$ is the number of vertices in the region RHA, K is the context bound, and $\alpha = \sum_{i=1}^{K} n^i$. There are atmost α different contexts of size atmost K with any sequence of the n boxes including a box being called more than once. Thus our contracted run ρ' has the same duration as ρ and is of length atmost $|\rho'| \leq C = 24(T.rmax+1).|\mathcal{X}|^2.(\alpha|Q|)^2(2.cmax+1)^{2.|\mathcal{X}|}$. To prove that the contracted run ρ' is valid, we need to ensure that (i) ρ' satisfies all the constraints as ρ, and has the same start and end configuration, and (ii) recursive calls, returns and contexts are valid in the given RHA. All the variables are always passed by reference and hence, the contexts have no valuations. Thus the precautions (in carefully splitting from type-0 to type-3 runs) taken in [9] for hybrid automata suffice to ensure constraints are satisfied. The second condition is satisfied due to context-sensitive contraction where in, the context is also matched in the loop detection. Due to this, a context in the contracted run will be a valid successor of the preceding context. This establishes the decidability of RHA with bounded context using only pass-by-reference. □

We show that our adaptation of contraction does not work on RHAs with unbounded context using the following RHA.



We can not apply context-sensitive contraction as the context might grow unboundedly and matching pairs may not be found within a type-3 run. In this example consider a run ρ_1 where B_1 calls B_2 and B_2 calls B_1 until $x = 1$ in B_2, after which the context shrinks to the first instance of B_1. In another run ρ_2, B_1 calls B_2 once, x grows to be 1 and B_2 returns to B_1. The context size is very small while achieving the same effect - same start and end configurations and same duration. To be able to obtain ρ_2 from ρ_1, we would need to apply contraction to the contexts too and shorten them in a sensible manner. Our context-sensitive contraction studied earlier does not alter the contexts and hence does not readily extend to this class of RHA. However, we conjecture decidability using a double layered contraction, as seen in [9].

6. UNDECIDABILITY

In this section we provide a proof sketch of one of our undecidability results (Theorem 3.3) by reducing the halting problem for two counter machines to the time-bounded reachability problem for unrestricted RTA_{NV}. In this reduction, the simulation of the increment, decrement and zero-check instructions of the machine is achieved through paths in RTA_{NV}. We have one path per instruction which adjusts the clocks to reflect changes in counter values.

We give a brief sketch of the reduction from halting problem to time bounded reachability in RTA_{NV} with 7 clocks such that the total time elapsed is ≤ 18 time units. We maintain 3 sets of clocks : $X = \{x_1, x_2\}$, $Y = \{y_1, y_2\}$ and $Z = \{z_1, z_2\}$. Let $\mathcal{X} = X \cup Y \cup Z \cup \{b\}$ denote the set of all 7 clocks. Upon entering location S_{k+1} of the path simulating the $(k+1)$th instruction, we have the values $\nu(Z) = 1 - \frac{1}{2^k}$, $\nu(X) = 1 - \frac{1}{2^{c+k}}$, $\nu(Y) = 1 - \frac{1}{2^{d+k}}$ and $\nu(b) = 0$, where c, d are the counter values. We show by our construction that, the time taken to simulate the $(k+1)$th instruction is $< \frac{9}{2^k}$. Thus, the total time is bounded from above by 18.

Figure 5 gives the set of paths and components for the $(k+1)$th instruction incrementing counter c. At the first node S_{k+1} of the path for $Inc\ c$, we have $\nu(X) = 1 - \frac{1}{2^{c+k}} = 1 - \beta_c$, $\nu(Y) = 1 - \frac{1}{2^{d+k}} = 1 - \beta_d$ and $\nu(Z) = 1 - \frac{1}{2^k} = 1 - \beta$, and $\nu(b) = 0$. The path S_{k+1} to E_{k+1} for $Inc\ c$ has three

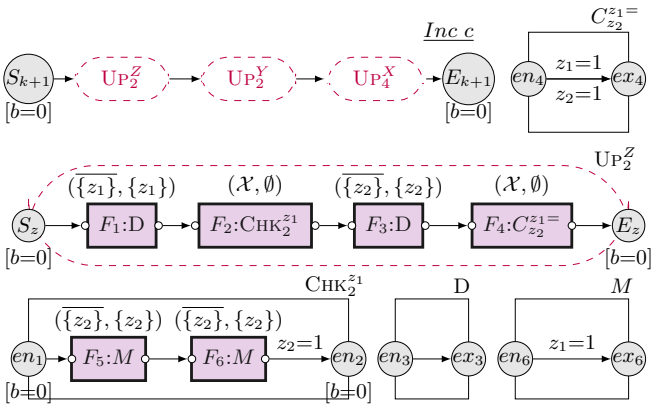


Figure 5: Increment c is simulated by path $\underline{Inc\ c}$ between S_{k+1} and E_{k+1}. The path from S_z to E_z is placed at the position specified by Up_2^Z. Similar path is drawn for Up_2^Y and Up_4^X replacing Z by Y, X.

subpaths $\text{UP}_2^Z, \text{UP}_2^Y$ and UP_4^X sequentially lined up one after the other. Each subpath changes only one of the three sets of clocks. UP_2^Z updates clocks Z to $\nu(Z) = 1 - \frac{\beta}{2}$ while UP_2^Y updates Y to $\nu(Y) = 1 - \frac{\beta_d}{2}$ and UP_2^X update X to $\nu(X) = 1 - \frac{\beta_c}{4}$. These changes of the clock values reflect the increment of c and end of $k+1$ instruction.

7. CONCLUSION

We introduced recursive hybrid automata as a generalization of hierarchical state machines with continuous variables that allows recursion while disallowing concurrency. We combined and further generalized existing models of recursive timed automata and timed pushdown automata to allow us to work on a general framework to clearly state decidability results. Our experience with this exercise can be summarized in the following take-away points.

1. The strong non-passing mechanism collapses under the bounded context restriction as it can be replaced by a few extra variables using pass by reference mechanism. This helps us improve complexity for RTA_{RN} from ExpTime-complete to PSpace-complete. From RTA_{RN}, we remove non-passing to get bounded context RTA_R which can be unrolled into a timed automaton. Similar results hold for recursive hybrid automata as well.

2. Our decidability results show that pass-by-reference mechanism is most well-behaved among all other studied mechanisms, while pass-by-value is the strongest, and allowing it yields undecidability.

In this paper we primarily focused on achieving decidability under bounded-time restriction, an analogous exercise can be performed for other decidable variants of hybrid automata. For instance, in [14] we showed that initialized [13] variant of RHA_{RNV} $\mathcal{A}$ can by reduced in polynomial time to RTA_{RNV} $\mathcal{B}$ which is polynomial in size compared to $\mathcal{A}$. As part of future work we plan to look into generalization of RHA that permit concurrency and aim to derive favorable conditions for decidability. We also plan to implement decidability results as part of a verification framework that integrate seamlessly with popular model-based design tools such as Simulink/Stateflow and Scicos.

8. REFERENCES

[1] P. Abdulla, M. Atig, and J. Stenman. Dense-timed pushdown automata. In *LICS*, pages 35–44, 2012.

[2] 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.

[3] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*, volume 736 of *LNCS*, pages 209–229, 1993.

[4] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. Pappas, and S. Oleg. Hierarchical hybrid modeling of embedded systems. In *EMSOFT*, pages 14–31, 2001.

[5] R. Alur and D. Dill. Automata for modeling real-time systems. In *ICALP*, volume 443 of *LNCS*, 1990.

[6] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, volume 999 of *LNCS*, 1995.

[7] T. Ball and S. K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN*, pages 113–130, 2000.

[8] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, pages 135–150, 1997.

[9] 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.

[10] S. Chaudhari. Subcubic algorithms for recursive state machines. In *POPL*, pages 159–169, 2008.

[11] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *CAV*, volume 1855 of *LNCS*, pages 232–247. 2000.

[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] S. Krishna, L. Manasa, and A. Trivedi. Recursive hybrid automata. In *www.cse.iitb.ac.in/∼krishnas/TR15.pdf*, 2015.

[15] S. La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. *LATIN*, pages 96–107, 2010.

[16] G. Lafferriere, G. J. Pappas, and S. Sastry. O-minimal hybrid systems. *Mathematics of control, signals and systems*, 13(1):1–21, 2000.

[17] J. Ouaknine and J. Worrell. Towards a theory of time-bounded verification. In *ICALP*, pages 22–37. 2010.

[18] J. Rumbaugh, I. Jacobson, and G. Booch, editors. *The Unified Modeling Language Reference Manual*. Addison-Wesley Longman Ltd., 1999.

[19] A. Trivedi and D. Wojtczak. Recursive timed automata. In *ATVA*, volume 6252 of *LNCS*, pages 306–324. 2010.