Hybrid Automata-based CEGAR for Hybrid Systems

Pavithra Prabhakar, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan

University of Illinois at Urbana-Champaign.
{pprabha2, duggira3, mitras, vmahesh}@illinois.edu

Abstract. In this paper we present a framework for carrying out counterexample guided abstraction-refinement (CEGAR) for systems modelled as hybrid automata. The main difference, between our approach and previous proposals for CEGAR for hybrid automata, is that we consider the abstractions to be hybrid automata as well. We present a counterexample guided abstraction refinement method for systems modelled as initialized rectangular hybrid automata along the lines of the general framework and show the completeness of the method. The advantages of our approach are illustrated by examples where it performs better than existing methods. Finally, we demonstrate the feasibility of our approach by performing several experiments using a prototype tool that implements our CEGAR algorithm.

1 Introduction

Counterexample guided abstraction refinement (CEGAR) [6] is an important technique that combats the state space explosion problem in model checking by *automatically* constructing abstractions of the system to be verified. It begins with a conservative initial abstract model that is progressively refined based on model checking the abstraction and analyzing the counterexamples generated by the model checker, until either a valid counterexample is found or an abstraction satisfying the safety property is obtained. CEGAR has been applied in a number of contexts with success [3,17,7,13].

Recently, there have been some proposals for carrying out CEGAR in the context of timed and hybrid systems [2,5,4,20,9,19,8,18]. The first set of proposals considers the abstractions to be finite, discrete transition systems [2,5,4,20,19]. In other words, the (infinite) configuration space of the hybrid automaton is partitioned into finitely many equivalence classes, and the abstraction has as states these equivalence classes; the abstraction has no continuous dynamics. The partitioning of the configurations maybe based on predicates [2,20,19] as in predicate abstraction [12]. A key challenges in applying this approach in practice lies in constructing the abstraction since constructing the abstract transitions requires computing the time successors of the equivalence classes in the concrete system.

The second proposal [8,18] is a CEGAR framework [1] for hybrid automata where the abstractions are constructed by ignoring certain variables. Counterexamples are used to identify new variables to be added to the abstraction. This approach has been carried out for timed automata [8] and linear hybrid automata [18].

In this paper, we extend the approach in [8,18] to present a CEGAR framework for (more general) hybrid systems. Thus, for us, abstractions of hybrid automata [14] are other hybrid automata, and not finite transition systems. We generalize the results in [8] in several directions. First, in our abstract hybrid automata, we allow control states and transitions to be collapsed; thus, the control structure of the abstraction is not the same as that of the original automaton. In addition, the continuous variables in the abstract hybrid automaton may correspond to combinations of original variables, and may have continuous dynamics distinct from those of the original system. Thus, refinement on analyzing a spurious counterexample is more involved; it may involve splitting control states/transitions, and/or adding variables that may have new dynamics.

Our main result in this paper is a CEGAR framework that is complete for the analysis of *intialized rectangular hybrid automata* [16]. Specifically, in this framework we consider abstractions to be initialized rectangular hybrid automata. We present algorithms for counterexample analysis and refinement in this setting. We prove that our framework is guaranteed to terminate, and will either prove that the original system satisfies the given safety property or demonstrate a bug by constructing a valid counterexample.

We highlight a few points that contrast our approach with that of [2,5] which considers finite abstractions: first, the abstractions used and the nature of refinements are different. Thus, the two methods are incomparable in terms of efficiency as measured by the number of abstraction-refinement cycles and the size of abstractions analyzed. Second, our approach of constructing abstractions only involves making local checks about the flow equations in the different control states. This is an easier problem than computing time successors, which is necessary when constructing finite abstractions. Thus, not only is constructing abstractions easier in our framework, it also has the pleasing consequence of clearly separating the model checking concerns (namely constructing time successors) from the concerns of constructing abstractions.

We have implemented our CEGAR algorithm in a tool that we call **H**ybrid **A**bstraction **RE**finement (HARE). HARE make calls to HyTech [15] to analyze abstract models and generate counterexamples; we considered PHAVer [11], but at the time of writing it does not produce counterexamples. We analyzed the tool on a few benchmark examples. The experiments reveal that HARE produces small abstractions to be analyzed, and as a consequence the model checking time for the final abstraction is smaller than the time taken by Hytech. Yet, the total verification time is better for HyTech than HARE primarily because HARE makes

[1] Strictly speaking, the paper [18] does not define a CEGAR framework, since the abstractions are not refined; in each cycle different abstractions are used.

numerous OS-level system calls to HyTech which incur large delays. Nonetheless, the total running time of HARE is comparable to HyTech in all the cases.

2 Preliminaries

Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ denote the set of naturals, integers, rationals, reals and non-negative reals, respectively. Let $\mathcal{I}$ denote the set of all intervals of the form $[a, b]$, $[a, \infty)$, $(-\infty, b]$ and $(-\infty, +\infty)$ where $a, b \in \mathbb{Z}$. Given $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \cdots, n\}$. Given a set X, $|X|$ will denote the size of X. Given $x = (x_1, \cdots, x_n) \in \mathbb{R}^n$, we use $(x)_i$ to denote x_i. Given $X \subseteq \mathbb{R}^n$, $(X)_i = \{(x)_i \,|\, x \in X\}$.

A *rectangular region* $B \subseteq \mathbb{R}^n$ is a cartesian product of intervals in $\mathcal{I}$, that is, $B = \prod_{i=1}^{n} (B)_i$. B is bounded if each of the $(B)_i$ is bounded. The set of all rectangular regions in $\mathbb{R}^n$ is denoted by *RectReg*(n), and the set of all bounded rectangular regions is denoted by *BRectReg*(n). Given $S \subseteq \mathbb{R}^n$, the *rectangular hull* of S, denoted *RectHull*(S), is the smallest rectangular region in *RectReg*(n) containing S.

A *transition system* $\mathcal{T}$ over an alphabet Σ is a tuple $(S, S^0, \rightarrow)$ where S is a set of states, $S^0 \subseteq S$ is a set of initial states and $\rightarrow \subseteq S \times \Sigma \times S$. We will write $s \xrightarrow{a}_{\mathcal{T}} s'$ instead of $(s, a, s') \in \rightarrow$. When the transition system $\mathcal{T}$ is understood, we will drop it from the subscript. Given two transition systems $\mathcal{T} = (S, S^0, \rightarrow)$ and $\mathcal{T}' = (S', S'^0, \rightarrow')$ over Σ, a function $f : S \rightarrow S'$ is called a *simulation function* if for all $s^0 \in S^0$, $f(s^0) \in S'^0$, and for all $a \in \Sigma$, $s_1 \xrightarrow{a}_{\mathcal{T}} s_2$ implies $f(s_1) \xrightarrow{a}_{\mathcal{T}'} f(s_2)$.

Given a matrix A, $A(i, j)$ will denote the element in the i-th row and j-th column. We call a matrix A a *scaling matrix* if all the entries of the A are non-negative rationals, every row contains exactly one non-zero element and every column contains at most one non-zero element. An example of a scaling matrix is A given by $[\frac{1}{3}\ 0\ 0; 0\ 0\ 5]$.

Let *Func*(m, n) denote the set of all continuous functions from $\mathbb{R}^m$ to $\mathbb{R}^n$ and *LinFunc*(m, n) denote the subset of all linear functions. A function $f \in$ *Func*(m, n) is called *linear* if there exists an $n \times m$ matrix A of rational numbers such that for every $x \in \mathbb{R}^m$, $f(x) = Ax$. We know that there is a unique matrix, denoted by *Matrix*(f), associated with every linear function f.

2.1 Hybrid Automata

We define hybrid automata which are used to model systems with mixed-discrete continuous behavior.

Definition A *hybrid automaton* (*HA*) $\mathcal{H}$ is a tuple (*Loc*, *Edge*, *Source*, *Target*, *Var*, *Loc*^{0}, *Cont*^{0}, *Inv*, *Flow*, *Guard*, *Reset*) where

- *Loc* is a finite set of *(discrete) control states* or *locations*.
- *Edge* is a finite set of *edges*.
- *Source*, *Target* : *Edge* $\rightarrow$ *Loc* are functions which associate a source and a target location to every edge.

- $Var = \{v_1, \cdots, v_n\}$ is a finite set of continuous *variables*. The set of continuous states is $Cont = \mathbb{R}^n$.
- $Loc^0 \in Loc$ and $Cont^0 \subseteq Cont$ are the *initial control and continuous states*.
- $Inv : Loc \to 2^{Cont}$ associates with every control state an *invariant*.
- $Flow : Loc \times Cont \to 2^{FlowsSet}$ where $FlowsSet = \{f \mid f : \mathbb{R}_{\geq 0} \to Cont$ is continuous$\}$ such that for all $q \in Loc$, $x \in Cont$ and $f \in Flow(q, x)$, $f(0) = x$.
- $Guard : Edge \to 2^{Cont}$ assigns to each edge a *guard*—a condition on the continuous variables that enables the discrete transition.
- $Reset : Edge \to 2^{Cont \times Cont}$ associates with each edge a *reset*—a binary relation that describes how the continuous state changes when a discrete transition occurs.

Notation In order to make the text more readable, we will often write the argument of a function as a subscript. In particular, Inv_q will be used to denote the invariant associated with control state q instead of $Inv(q)$, and similarly $Guard_{(p,q)}$ and $Reset_{(p,q)}$ to denote the guard and reset conditions associated with an edge (p, q). We will use $\mathcal{H}_1$, $\mathcal{H}_2$, $\mathcal{H}_A$, $\mathcal{H}_C$ and so on to represent hybrid automata and we will denote the components of automaton $\mathcal{H}_1$ by $Loc_1, Edge_1$, etc. and those of $\mathcal{H}_A$ by $Loc_A, Edge_A$, etc.

The semantics of a hybrid automaton $\mathcal{H}$ is defined in terms of the transition system $[\![\mathcal{H}]\!] = (Q, Q^0, \to)$ over $\Sigma = \mathbb{R}_{\geq 0} \cup Edge$, where $Q = Loc \times Cont$, $Q^0 = Loc^0 \times Cont^0$, and the transition relation $\to$ is given by:

Continuous transitions $(q_1, x_1) \xrightarrow{t} (q_2, x_2)$ iff $t \in \mathbb{R}_{\geq 0}$, $q_1 = q_2 = q$ and there exists $f \in Flow(q, x_1)$ such that $x_2 = f(t)$ and for all $t' \in [0, t]$, $f(t') \in Inv_q$.

Discrete transitions $(q_1, x_1) \xrightarrow{e} (q_2, x_2)$ iff $e \in Edge$ such that $q_1 = Source(e)$ and $q_2 = Target(e)$, $x_1 \in Guard_{(q_1,q_2)}$, and $(x_1, x_2) \in Reset_{(q_1,q_2)}$.

In a time transition, the discrete part q of the state does not change, but the continuous part changes according to the flow function *Flow* while remaining within the invariant Inv_q. On the other hand, in a discrete transition, control state changes according to an edge in the automaton; the continuous part of the state before the transition is required to satisfy the guard associated with the edge, and the result of taking the transition changes the continuous state according to the reset conditions associated with the edge.

An *execution* of $\mathcal{H}$ is a path $\sigma = (q_0, x_0) \xrightarrow{a_0} (q_1, x_1) \xrightarrow{a_1} (q_2, x_2) \cdots (q_{n-1}, x_{n-1}) \xrightarrow{a_{n-1}} (q_n, x_n)$ such that $q_0 = Loc^0$, $x_0 \in Cont^0$ and $a_i \in \mathbb{R}_{\geq 0}$ if i is even and $a_i \in Edge$ otherwise. We say that the execution σ *reaches* a location $q \in Loc$, if n is even and $q_n = q$. *Control state reachability problem* for hybrid systems is the problem of checking whether a given *HA* has an execution that reaches a given state.

Let us define the functions *Pre* and *Post* on the states of $\mathcal{H}$. Given a set of states $S \subseteq Loc \times Cont$ and a symbol $a \in (\mathbb{R}_{\geq 0} \cup Edge)$, $Pre_{\mathcal{H}}(S, a) = \{(q, x) \mid \exists (p, y) \in S : (q, x) \xrightarrow{a}_{\mathcal{H}} (p, y)\}$. Similarly $Post_{\mathcal{H}}(S, a) = \{(q, x) \mid \exists (p, y) \in S : (p, y) \xrightarrow{a}_{\mathcal{H}} (q, x)\}$.

Next we define a subclass of hybrid systems in which invariants, guards and resets are restricted to be rectangular and the flows have rectangular bounds on their derivative. Since it is easier to define this class in terms of the derivative of the flows instead of the flows themselves, we use a slightly different notation. We will point out how it can be written in terms of the notation used for defining the general class of hybrid systems.

A *rectangular hybrid automaton* $\mathcal{H}$ is a tuple (*Loc*, *Edge*, *Source*, *Target*, *Var*, Loc^0, $Cont^0$, *Inv*, *Activity*, *Guard*, *Reset*) where all the elements are defined as for the hybrid automaton except for the following differences:

- $Cont^0 \in BRectReg(|\,Var|)$,
- $Inv : Loc \rightarrow BRectReg(|\,Var|)$,
- $Activity : Loc \rightarrow RectReg(|\,Var|)$,
- $Guard : Edge \rightarrow RectReg(|\,Var|)$, and
- $Reset : Edge \times [|\,Var|] \rightarrow \mathcal{I} \cup \{Id\}$.

The $Cont^0$ and the invariants are constrained to be rectangular regions. The *Activity* function says that any smooth function whose derivative is always bounded by the rectangular region specified by *Activity* is allowed. The reset function assigns to every variable an interval which specifies the set of new values for the variable after the discrete transition or assigns *Id* in which case the value of the variable does not change after the edge is taken.

The above rectangular hybrid automaton $\mathcal{H}$ can be represented in the standard definition of the hybrid automaton as $\mathcal{H}' = (Loc, Edge, Source, Target, Var, Loc^0, Cont^0, Inv, Flow, Guard, Reset')$, where:

- $Flow_{q,x} = \{f : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^{|\,Var|} \mid f$ is a smooth function, $f(0) = x$, $\dot{f}(t) \in Activity(q)$ for all $t \in \mathbb{R}_{\geq 0}\}$, and
- $(x, x') \in Reset'_e$ iff for each $i \in [|\,Var|]$, $Reset(e, i) = Id$ implies $(x')_i = (x)_i$ and $Reset(e, i) \in \mathcal{I}$ implies $(x')_i \in Reset(e, i)$.

By $[\![\mathcal{H}]\!]$, we mean $[\![\mathcal{H}']\!]$.

We say that a rectangular hybrid automaton $\mathcal{H}$ is *initialized* (*IRHA*) if for every $e \in Edge$ and $i \in [|\,Var|]$, $(Activity(Source(e)))_i \neq (Activity(Target(e)))_i$ implies $Reset(e, i) \in \mathcal{I}$, that is, if the continuous evolution for a variable is different in the source and the target locations of a transition, then the variable is reset to some value in an interval.

3 CEGAR for general hybrid systems

Our CEGAR approach for hybrid automata uses hybrid automata themselves as abstractions. In order to describe this framework, we need to outline how hybrid automata will be abstracted, how abstract counterexamples would be checked, and how the abstraction will be refined if the abstract counterexample generated by the model checker is found to be spurious. These steps will be described in this section.

We begin by defining when one hybrid automata will be an abstraction of another. In the abstract models, we allow control states and transitions of the concrete automaton to be merged, and allow new dynamics which arise due to general kinds of transformations on the concrete dynamics. However, the abstract model should satisfy certain consistency conditions for it to be a simulation of the concrete model. These conditions are formally explained in the next definition.

Definition 1. *Let $\mathcal{H}_A$ and $\mathcal{H}_C$ be two hybrid automata. A* hybrid abstraction *α from $\mathcal{H}_C$ to $\mathcal{H}_A$, denoted $\alpha(\mathcal{H}_C, \mathcal{H}_A)$, is a tuple $(\alpha_{Loc}, \alpha_{Edge}, \alpha_{Var})$, where*

- *$\alpha_{Loc} : Loc_C \to Loc_A$ is the location abstraction function,*
- *$\alpha_{Edge} : Edge_C \to Edge_A$ is the edge abstraction function, and*
- *$\alpha_{Var} \in Func(|Var_C|, |Var_A|)$ is the variable abstraction function,*

and satisfies the following conditions [2]*:*

- *$\alpha_{Loc}(Loc^0_C) = Loc^0_A$ and $\alpha_{Var}(Cont^0_C) \subseteq Cont^0_A$;*
- *for every $q_C \in Loc_C$,*
  - *$\alpha_{Var}(Inv_C(q_C)) \subseteq Inv_A(\alpha_{Loc}(q_C))$, and*
  - *for every $x_C \in Cont_C$ and $f_C \in Flow_C(q_C, x_C)$, there exists an $f_A \in Flow_A(\alpha_{Loc}(q_C), \alpha_{Var}(x_C))$ such that for all $t \in \mathbb{R}_{\geq 0}$, $\alpha_{Var}(f_C(t)) = f_A(t)$;*
- *for every $e_C \in Edge_C$,*
  - *$Source_A(\alpha_{Edge}(e_C)) = \alpha_{Loc}(Source_C\ (e_C))$,*
  - *$Target_A(\alpha_{Edge}(e_C)) = \alpha_{Loc}(Target_C(e_C))$,*
  - *$\alpha_{Var}(Guard_C(e_C)) \subseteq Guard_A\ (\alpha_{Edge}(e_C))$, and*
  - *$(x_1, x_2) \in Reset_C(e_C)$ implies $(\alpha_{Var}(x_1), \alpha_{Var}(x_2)) \in Reset_A(\alpha_{Edge}(e_C))$.*

Notation Given any abstraction function named β, the elements of β representing the location, edge and variable abstraction functions are denoted by adding the appropriate subscripts. We will abuse notation and use $\beta(S)$ to denote $\{(\beta_{Loc}(q), \beta_{Var}(x)) \,|\, (q, x) \in S\}$, and similarly use $\beta^{-1}(S')$ to denote $\{(q, x) \,|\, (\beta_{Loc}(q), \beta_{Var}(x)) \in S'\}$.

The abstract hybrid automaton $\mathcal{H}_A$ simulates the concrete automaton $\mathcal{H}_C$. This is summarized in the following proposition.

Proposition 1. *Let $[\![\mathcal{H}_C]\!] = (S_C, S^0_C, \to_C)$ and $[\![\mathcal{H}_A]\!] = (S_A, S^0_A, \to_A)$. Then the function $f : S_C \to S_A$ given by $f(q_C, x_C) = (\alpha_{Loc}(q_C), \alpha_{Var}(x_C))$ is a simulation function.*

Remark 1. To construct an abstraction that is a hybrid automaton, one only needs to ensure that certain local consistency conditions are met. This is potentially less expensive than computing time successors of concrete states which is needed when constructing discrete abstractions, as time successors involve computing the solution of differential equations.

[2] We lift the definition of α_{Var} from points to sets in the natural way, that is, $\alpha_{Var}(X) = \{\alpha_{Var}(x) \,|\, x \in X\}$ where X is a set.

Having described our class of abstract models, we now describe how abstract counterexamples will be validated. In this paper, we will focus on verifying control state reachability properties. Let us fix a concrete *HA* $\mathcal{H}_C$ and a location q_C of $\mathcal{H}_C$ different from the initial location. We want to verify if $\mathcal{H}_C$ reaches q_C. Let us also fix an abstract hybrid automaton $\mathcal{H}_A$ which is an abstraction of $\mathcal{H}_C$ with the abstraction function α. Let $q_A = \alpha_{Loc}(q_C)$. We model-check $\mathcal{H}_A$ to verify if it reaches q_A. If $\mathcal{H}_A$ does not reach q_A, we can conclude that $\mathcal{H}_C$ does not reach q_C. It follows from the fact that $[\![\mathcal{H}_A]\!]$ simulates $[\![\mathcal{H}_C]\!]$. If $\mathcal{H}_A$ reaches q_A, then the model-checking algorithm is required to return an execution σ' which reaches q_A in $\mathcal{H}_A$. σ' is called the *abstract counterexample*. Let us fix an abstract counter example σ' in $\mathcal{H}_A$ to be $(q'_0, x'_0) \xrightarrow{a'_0} (q'_1, x'_1) \xrightarrow{a'_1} (q'_2, x'_2) \cdots (q'_{l-1}, x'_{l-1}) \xrightarrow{a'_{l-1}} (q'_l, x'_l)$.

To check the validity of the counterexample σ', we need to see if there is an execution of $\mathcal{H}_c$ *corresponding* to σ'. Formally, a *concrete counterexample* corresponding to σ' is an execution σ of $\mathcal{H}_C$ given by $(q_0, x_0) \xrightarrow{a_0} (q_1, x_1) \xrightarrow{a_1} (q_2, x_2) \cdots (q_{l-1}, x_{l-1}) \xrightarrow{a_{l-1}} (q_l, x_l)$ in $\mathcal{H}_C$ such that $q_0 = Loc^0_C$; $q_l = q_C$; for all i, $\alpha_{Loc}(q_i) = q'_i$ and $\alpha_{Var}(x_i) = x'_i$; and for every even i, $a_i = a'_i$ and for every odd i, $\alpha_{Edge}(a_i) = a'_i$. Let $Concrete(\sigma')$ denote the set of all concrete counterexamples corresponding to σ'. *Validation* is the process of checking if $Concrete(\sigma')$ is non-empty.

In order to check the above we compute the set of all initial states which have concrete counterexample starting from them corresponding to σ'. We call this $Reach_0$, which is computed iteratively by computing for every suffix of σ' the set of states from which there is a concrete execution corresponding to this suffix. Let $S_k = \alpha^{-1}((q'_k, x'_k))$ for $k > 0$ and $S_0 = \alpha^{-1}((q'_0, x'_0)) \cap (Loc^0_C \times Cont^0_C)$. Now, $Reach_k$, for $0 \leq k \leq l$, is given by:

- $Reach_l = S_l \cap (\{q_C\} \times \mathbb{R}^{|Var_C|})$.
- For $0 \leq k < l$, $Reach_k = S_k \cap \bigcup_{a \in \alpha^{-1}_{Edge}(a'_k)} Pre_{\mathcal{H}_C}(Reach_{k+1}, a)$, where $\alpha^{-1}_{Edge}(a'_k) = a'_k$ when $a'_k \in \mathbb{R}_{\geq 0}$ is a time transition.

The following proposition tells us how the *Reach* sets can be used to validate the abstract counterexample.

Proposition 1 *$Concrete(\sigma') = \emptyset$ iff $Reach_k = \emptyset$ for some $0 \leq k \leq l$.*

In order to perform the validation step, we need to be able to compute the *Reach* sets and check their emptiness. If we find that $Concrete(\sigma')$ is not empty, then we know that there is a concrete counterexample σ corresponding to σ', and hence that the system is faulty. Otherwise σ' is a *spurious counterexample*.

If the abstract counterexample σ' is found to be spurious in the validation step, then we use the results of the analysis during the validation to refine the abstract system. Let us first define what we mean by a refined model.

Definition 2. *A HA $\mathcal{H}_R$ is refinement of $\mathcal{H}_A$ with respect to $\mathcal{H}_C$ if there exist abstractions $\beta(\mathcal{H}_C, \mathcal{H}_R)$ and $\gamma(\mathcal{H}_R, \mathcal{H}_A)$ such that $\alpha_{Loc} = \gamma_{Loc} \circ \beta_{Loc}$, $\alpha_{Edge} = \gamma_{Edge} \circ \beta_{Edge}$ and $\alpha_{Var} = \gamma_{Var} \circ \beta_{Var}$.*

We want to find a refinement $\mathcal{H}_R$ of $\mathcal{H}_A$ with respect to $\mathcal{H}_C$ which "eliminates" the spurious counterexample σ'. Let us define a *potential counterexample* of σ' to be a sequence $\rho = (q_0, x_0)a_0(q_1, x_1)a_1(q_2, x_2)\cdots(q_{l-1}, x_{l-1})a_{l-1}(q_l, x_l)$ such that $\alpha_{Loc}(q_i) = q'_i$ and $\alpha_{Var}(x_i) = x'_i$ for all i, $q_l = q_C$, and $a_i = a'_i$ for even i and $\alpha_{Edge}(a_i) = a'_i$ for odd i. If $(q_i, x_i) \xrightarrow{a_i}_{\mathcal{H}_C} (q_{i+1}, x_{i+1})$ for every i, then ρ corresponds to an actual counterexample of $\mathcal{H}_C$. Let $Potential_{\mathcal{H}_C}(\sigma')$ be the set of potential counterexamples of σ' in $\mathcal{H}_C$. The refinement $\mathcal{H}_R$ should eliminate some potential counterexample of σ' of $\mathcal{H}_A$, more precisely,

C1 there exists a concrete counterexample $\sigma \in Potential_{\mathcal{H}_C}(\sigma')$ such that $\sigma \notin Potential_{\mathcal{H}_C}(\sigma'')$ for any counterexample $\sigma'' \in \mathcal{H}_R$.

Observe that $Reach_k = \emptyset$ iff $(\cup_{a \in \alpha^{-1}_{Edge}(a'_k)} Post_{\mathcal{H}_C}(S_k, a)) \cap Reach_{k+1} = \emptyset$. Thus, in order to refine an abstraction, we will try to ensure that the following stronger condition holds.

C2 $(\cup_{a \in \beta(\alpha^{-1}(a_{\hat{k}}))} Post_{\mathcal{H}_R}(\beta(S_{\hat{k}}), a)) \cap \beta(Reach_{\hat{k}+1}) = \emptyset$.

Condition **C2** will ensure that the condition **C1** holds.

4 CEGAR for Initialized Rectangular Hybrid Automata

In this section, we will present a counterexample guided abstraction refinement method for the class of initialized rectangular hybrid automata along the lines of the general framework discussed in the previous section. We prove the completeness of our CEGAR method. Due to lack of space, we omit some of the details and proofs, which are defered to Section B.

In order to define the framework, we begin by outlining the kinds of abstractions we will consider. Recall from Section 3, an abstraction is specified by an abstraction function α that describes how control states, discrete jumps, and variables are abstracted. In the case of initiallized rectangular hybrid automata, we restrict out attention to abstractions α, where α_{Var} is a given by a scaling matrix. Thus, every abstract variable $a \in Var_A$ is given by $a = c/k$, where $c \in Var_C$ and k is an integer constant. We do this because of the following observation. Recall that an *IRHA* is verified by translating it into Timed automata, and doing the region construction for Timed automata. Now, focussing on scaling matrices ensures that the regions of the abstract hybrid automaton $\mathcal{H}_A$ are unions of regions of $\mathcal{H}_C$; this allows us to argue completeness semantically.

Let us now describe how the abstract automaton $\mathcal{H}_A$ is constructed from $\mathcal{H}_C$ and α. $\mathcal{H}_A$ will have as control locations Loc_A (the range of α_{Loc}), and transitions $Edge_A$ (the range of α_{Edge}); all we need to do is describe the flows, invariants, guards, and resets. Recall since $\mathcal{H}_C$ and $\mathcal{H}_A$ are *IRHA*s, we only need to consider constraints of the form $x \in [a, b]$. Further, we know that α_{Var} is given by a scaling matrix. Suppose $x \in Var_A$ such that $x = y/k$, where $y \in Var_C$. The activity for x in a location q will be computed by taking the rectanular hull of all the activity constraints for y in locations p that get mapped to q. A constraint of the form

$\varphi : y \in [a, b]$ is translated into $\alpha(\varphi) : x \in [\mathit{floor}(a/k), \mathit{ceil}(b/k)]$, a constraint for x. The invariant for a variable x in abstract location q will be given as follows: translate the invariants for y in each of the concrete location p that get mapped to q, and then take the rectangular hull of their union. Guards on edges are computed in a similar fashion as the invariants. Finally, whenever resets on an abstract transition are Id only if the activity constraints on the source and target of the transition are the same. Otherwise, some of the concrete transitions that have reset as Id will be translated into an abstract transition where reset is the intersection of the guard and invariant of the source location.

In the CEGAR loop, we can start with any abstraction. To keep the presentation simple we assume that the initial concrete location Loc_C^0 and the bad location q_C are not merged with each other or any other location in the abstraction. We model check the abstraction $\mathcal{H}_A$. The control state reachability problem is decidable for the class of initialized rectangular hybrid automata [16]. If the model-checking algorithm returns that the state $\alpha_{Loc}(q_C)$ is not reachable in $\mathcal{H}_A$, then we are done. Otherwise, it returns a counterexample σ', which we need to analyse. In the case of *IRHA*, we can compute the sets $Reach_k$ for all k. Hence we can effectively validate the counterexample. Suppose that $\hat{k}$ is the largest k for which $Reach_k$ is empty. We need to find a refinement which satisfies Condition $C2$. Next, we discuss how to obtain the required β. It is obtained in two steps.

We will first discuss the case where $a_{\hat{k}} \in Edge_A$. In the first step we obtain an abstraction β' which satisfies the following conditions.

- β'_{Loc} is same as α_{Loc} except possibly that it splits $q'_{\hat{k}+1}$
- β'_{Edge} is the same as α_{Edge} except possibly it splits the transition $a_{\hat{k}}$
- β'_{Var} is a scaling matrix in which all the non-zero entries are 1.
- Let $\mathcal{H}_{R'}$ be the abstract *IRHA* corresponding to β'. Then $(\cup_{a \in \beta'(\alpha^{-1}(a_{\hat{k}}))} Post_{\mathcal{H}_{R'}}(\beta'(S_{\hat{k}}), a)) \cap \beta'(Reach_{\hat{k}+1}) = \emptyset$.

Remark 2. Note that such a β' exists. If we split all the locations in $q'_{\hat{k}+1}$, all the transitions in $a_{\hat{k}}$, and take the scaling matrix to be diagonal matrix with all the diagonal values 1, then the last condition above translates into $Post_{\mathcal{H}_C}(S_{\hat{k}}) \cap Reach_{\hat{k}+1} = \emptyset$, which is true by the choice of $\hat{k}$. Note that every scaling matrix with non-zero entries 1 corresponds to just choosing a subset of the variables and associate scaling 1 to all of them. Hence we can enumerate all the splittings of the $q'_{\hat{k}+1}$ and $a_{\hat{k}}$, and all the subsets of Var_C, and verify that the resulting β' satisfies the above conditions.

Remark 3. Note that unlike the refinement for the discrete abstractions, here we need to split both $q'_{\hat{k}+1}$ and $a_{\hat{k}}$. In order to see this, suppose we have that the abstract location $q'_{\hat{k}}$ consists of concrete locations A_1 and A_2 and the abstract location $q'_{\hat{k}+1}$ consists of the concrete location B. Let us assume that both the concrete and abstract automata have a single variable v and the corresponding scaling is 1. There is a transition t_1 from A_1 to B with guard $x \in [1, 2]$ and a transition t_2 from A_2 to B with guard $x \in [4, 5]$. Both t_1 and t_2 are collapsed into a single transition in the abstraction. Suppose that $Reach_{\hat{k}+1}$ is B with $x = 3$.

Note that all the variables have been added and $q'_{\hat{k}+1}$ can no more be split. In order to satisfy condition $C2$, we need to be able to separate t_1 and t_2.

In the next step, we update the scalings of various variables. We define a new abstraction $\beta = (\beta_{Loc}, \beta_{Edge}, \beta_{Var})$, where β_{Loc} is same as β'_{Loc} and β_{Edge} is same as β'_{Edge}. β_{Var} is defined as follows. For every variable $v \in Var_{R'}$, set s_v to be the g.c.d of all the non-zero integer constants appearing in the intervals, corresponding to v, of the guards and resets of edges $e_{R'}$ of $\mathcal{H}_{R'}$ such that $e_{R'}$ has its source in $\beta'(\alpha^{-1}(q'_{\hat{k}}))$ and target in $Disc(\beta'(Reach_{\hat{k}+1}))$, where $Disc(S) = \{q \in Loc_C \mid \exists (q, x) \in S\}$. The variables in the abstraction being defined will be the union of the Var_A and $Var_{R'}$, and the scaling of a variable in β_{Var} will be the g.c.d of the scalings of the variable in α and β'_{Var} if it occurs in both Var_A and $Var_{R'}$, otherwise its value will the value of its scaling in α_{Var} or β'_{Var} depending on whether it occurs in Var_A or $Var_{R'}$ respectively. Let $\mathcal{H}_R$ be the abstraction defined by β.

Next, we briefly sketch the case when $a_{\hat{k}}$ is a time transition. It is similar to the case where a_k is an edge transition, with the following differences. In this case, β'_{Edge} is the same as α_{Edge} (since $a_{\hat{k}}$ not a discrete jump). Next, in the second part, for each variable, s_v is chosen to be the g.c.d of the non-zero integer constants occurring in the intervals corresponding to v in the invariants and activities of the locations of $\mathcal{H}_{R'}$ which are in $Disc(\beta'(Reach_{\hat{k}+1}))$.

Termination of the algorithm. In each refinement step we either split some location of $\mathcal{H}_A$ or increase rows of the scaling matrix or change an entry in the scaling matrix. Otherwise β is same as α and Condition $C2$ fails. Since the number of locations and variables in the concrete *IRHA* are finite and the scaling can change only finitely many times before it reaches 1, the algorithm will terminate in a finite number of steps.

5 An Illustrative Example

In this section, we present an example to illustrate how our method works. The CEGAR loop in our framework works differently that the predicate abstraction based CEGAR algorithm [2,5]; one difference, for example, is that the predicate abstraction method splits one abstract state in a refinement step, whereas our method might split many abstract states in one step because it might choose to add a new variable. This is also illustrated in the example outlined here.

Consider a hybrid automata $\mathcal{H}_i$ with 3 control locations — $(i, 1), (i, 2)$, and $(i, 3)$ — and two continuous variables x and y. Both x and y have activity 1 in every location. The invariant in every location is $(0 \leq x \leq 1) \ \wedge \ (i \leq y \leq i + 1)$. Finally, there is a discrete transition from $(i, 1)$ to $(i, 2)$ and from $(i, 2)$ to $(i, 3)$; both these transitions have a guard that requires $x = 1$, and they reset x to 0. It can be seen that the reachable part of $(i, 2)$ starting in $(i, 1)$ at $x := 0$ and $y := i$ lies on or above the line $x = y - i$, and the part of $(i, 2)$ which reaches $(i, 3)$ is strictly below this line. So if we start with the abstraction containing 3 states,

namely, $(i,1)$, $(i,2)$ and $(i,3)$, CEGAR based on predicate abstraction [2,5] will add a predicate $x \leq y - i$. Note that for different values of i a different predicate is added.

Now consider an automaton $\mathcal{H}$ which is the disjoint union of the automata $\mathcal{H}_i$, for $i \in \{0, \ldots n-1\}$ as follows. $\mathcal{H}$ has a new initial location s and a transition from s to $(i,1)$ for each $i \in \{0, \ldots n-1\}$, which resets x to 0 and y to i. Suppose we want to check if we can reach any of the $i,3$ locations. CEGAR based on predicate abstraction will take n iterations, one for every i to eliminate the counterexample corresponding to the i-th copy of the subautomaton. So the sum of the number of states explored during the CEGAR is $O(n^2)$. On the other hand, our algorithm, starting with the same initial abstraction, will take some constant number of steps to eliminate the counterexample corresponding to $i = 0$ in the process adding both the variables x and y. After this the algorithm terminates. So the states explored by our CEGAR algorithm is $O(n)$.

6 Implementation and Experimental Results

In this section we describe the implementation of our algorithm and the experimental results. The tool, which we call **H**ybrid **A**bstraction **RE**finement (HARE), is developed in C and works on Linux and MacOSX. HARE makes calls to a model-checker for generating counterexamples, and to this end it currently[3] uses HyTech [15]. Indeed, this restricts the class of models that HARE can analyze, as HyTech only works with rectangular automata.

6.1 Implementation

HARE takes as input the specification of the concrete hybrid automaton and an initial abstraction function, and creates an initial abstract automaton. The abstract automaton is converted to the input language for HyTech and then model-checked using HyTech. If HyTech does not produce a counterexample, then HARE returns the current abstraction and the hybrid system is inferred to be safe, otherwise, the counterexample is parsed and validated.

Consider an abstract counterexample trace $\sigma = s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_n$, where each s_i is a set of states. Validation proceeds backwards and involves checking that each $s_i \rightarrow s_{i+1}$ corresponds to a valid transition of the concrete automaton. In validating σ, a restricted version of the concrete automaton that only allows a sequence of transitions that corresponds to σ is created. If $s_i \rightarrow s_{i+1}$ is a timed transition, HARE constructs a new hybrid automaton with the locations in $\alpha^{-1}(s_i)$. Transitions from one location to another are not added in this automaton as s_i and s_{i+1} correspond to the same set of concrete locations. If $s_i \rightarrow s_{i+1}$ is an edge transition, HARE constructs a new hybrid automaton with the locations $\alpha^{-1}(s_i)$ and $\alpha^{-1}(s_{i+1})$. However, in this case, transitions only from the locations

[3] We considered PHAVer [11], but at the time of writing it does not produce counterexamples

in $\alpha^{-1}(s_i)$ to the locations in $\alpha^{-1}(s_{i+1})$ are added in order to enforce the edge transition from abstract state s_i to s_{i+1}. After creating automaton for time or edge transition, HARE calculates $Reach_i$ as $pre(Reach_{i+1}) \cap \alpha^{-1}(s_i)$ using the pre function provided by HyTech.

If a particular transition $s_i \rightarrow s_{i+1}$ in the abstract trace is spurious then $Reach_i = \emptyset$ and HARE then performs refinement by splitting the locations in s_i and s_{i+1}. s_{i+1} is split into: one location for all the locations which do not occur in $Reach_{i+1}$ and one location for each location which occurs in $Reach_{i+1}$. Further, s_i is split into: one location for all the locations which do not have a transition to $Reach_{i+1}$ and one for each location which have transitions to $Reach_{i+1}$. As for the variables, HARE enumerates all subsets of variables that are not added to the abstraction and adds one of the smallest subsets, say v_s, such that the projection of $pre(Reach_{i+1})$ over v_s is disjoint with the projection of s_i over v_s.

6.2 Experimental Results

Our experimental evaluation of HARE is based on two classes of examples: (a) generalization of the well-known billiards example [1], and (b) modified version of the Navigation benchmark [10].

The billiards example models a point in an n-dimensional bounded rectangle. The ball has a fixed initial velocity and every time it touches a boundary the appropriate velocity component changes sign. The problem is to check if the ball reaches a certain final point in the bounded rectangle. We refer to n-dimemsional instances of this problem as Bill-n-A or Bill-n-B depending on the starting and the final point configurations (see Appendix C for details of these configurations).

The navigation example models the motion of a point robot in an n-dimensional cube. The cube is partitioned into m^n rectangular regions and each such region is associated with a vector field described by rectangular flow equations. When the robot is in region A, its motion is described by the flow equations of region A. The problem is to check if robot reaches a certain unsafe region starting from an initial region. We refer to an n-dimemsional instance of this problem with m^n partitions as Nav-n-m-A, Nav-n-m-B or Nav-n-m-C, depending on the configurations of the vector fields, the locations of starting and the unsafe regions (see Appendix C for details of these configurations).

Discussion. It is clear that in these experiments HARE produces relatively small abstractions: in some cases with two orders of magnitude reduction in the number of locations, and often reducing the continuous state space by one or two dimensions. In the extreme case of Nav-2-*-A, an abstraction with 10 discrete states is found in 6 iterations, independent of the size of the grid. This is perhaps not too surprising in the light of the fact that here only a constant number of control locations can reach the unsafe region, and the final abstraction successfully lumps all the locations that cannot reach it. Consequently, in all the experiments, the time for model-checking the final abstraction is much smaller than the time for model-checking the concrete automaton. Yet, the total verification time is better for HyTech than HAREprimarily because HARE makes

Problem	Conc. size (locs, vars)	Abst. size (locs, vars)	CEGAR steps	Last verif. time (sec)	Total verif. time (sec)	Abst. verif. time (sec)	HyTech time (sec)
Bill-2-A	6, 2	3, 0	0	0.01	0.06	0.01	0.03
Bill-2-B	6, 2	6, 2	3	0.03	0.28	0.09	0.03
Bill-3-A	10, 3	3, 0	0	0.01	0.08	0.01	0.04
Bill-3-B	10, 3	10, 3	4	0.04	0.39	0.16	0.04
Nav-2-5-A	25, 2	10, 2	6	0.02	0.48	0.16	0.06
Nav-2-10-A	100, 2	10, 2	6	0.01	0.89	0.16	0.16
Nav-2-20-A	400, 2	10, 2	6	0.01	3.01	0.15	0.41
Nav-2-30-A	900, 2	10, 2	6	0.01	6.93	0.15	0.76
Nav-2-40-A	1600, 2	10, 2	6	0.01	13.05	0.16	1.24
Nav-2-5-B	25, 2	8, 1	3	0.03	0.35	0.09	0.06
Nav-2-10-B	100, 2	24, 2	16	0.11	4.44	1.07	0.24
Nav-2-20-B	400, 2	55, 2	36	0.22	30.13	4.59	0.88
Nav-2-30-B	900, 2	85, 2	56	0.35	94.15	10.53	2.00
Nav-2-40-B	1600, 2	115, 2	76	0.45	235.98	18.62	3.52
Nav-3-5-C	125, 3	14, 1	3	0.02	1.45	0.10	0.21
Nav-3-6-C	216, 3	14, 1	3	0.02	2.38	0.10	0.30
Nav-3-7-C	343, 3	14, 3	5	0.04	3.98	0.27	0.42
Nav-3-8-C	512, 3	14, 3	5	0.04	5.92	0.25	0.57
Nav-3-9-C	729, 3	14, 3	5	0.03	8.40	0.26	0.75
Nav-3-10-C	1000, 3	14, 3	5	0.03	11.66	0.26	0.99
Nav-4-5-C	625, 4	15, 1	3	0.04	15.14	0.12	0.90
Nav-4-6-C	1296, 4	18, 1	3	0.03	32.55	0.13	1.60

Fig. 1. The columns (from left) show the problem name, sizes of the concrete and final abstract hybrid automaton, number of CEGAR iterations, time required for verifying the final abstraction, total time, time required to verify all the abstractions, and finally, the time required for direct verification with HyTech.

numerous OS-level system calls to HyTech which incur large delays. Notice that only a small fraction of the total verification time in HARE is utilized in actually verifying the abstractions.

As mentioned before, HyTech restricts us to only rectangular hybrid systems. Furthermore it also limits us to hybrid automata with about 2000 locations, beyond which HyTech's parser fails. The advantage of the variable hiding is more apparent in the models with higher dimensions, for example in Nav-3-5-C and Nav-4-5-C. The addition of continuous variables in the refinement step depends on the counterexample that is generated by HyTech. As HyTech produces one of the shortest possible counterexamples, we have observed that often a variable is added prematurely to the refinement. We believe that, carefully controlling the generation of counterexamples can yield better abstractions. All this suggests, a direction of research, one we plan on pursuing, where the model-checker is more closely integrated with an abstraction refinement tool such as HARE.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *TCS*, 138(1):3–34, 1995.
2. R. Alur, T. Dang, and F. Ivancic. Counter-Example Guided Predicate Abstraction of Hybrid Systems. In *TACAS 2003*, pages 208–223, 2003.
3. T. Ball and S. Rajamani. Bebop: A symbolic model checker for Boolean programs. In *Proc. of the SPIN Workshop on Model Checking Software*, pages 113–130, 2000.
4. E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems. *International Journal on Foundations of Computer Science*, 14(4):583–604, 2003.
5. E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Verification of Hybrid Systems Based on Counterexmple-Guided Abstraction Refinement. In *TACAS 2003*, pages 192–207, 2003.
6. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *CAV 2000*, pages 154–169, 2000.
7. J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasareanu, Robby, and H. Zheng. Bandera: Extracting finite-state models from Java source code. In *Proc. of the Intl. Conf. on Software Engineering*, pages 439–448, 2000.
8. H. Dierks, S. Kupferschmid, and K. Larsen. Automatic Abstraction Refinement for Timed Automata. In *FORMATS 2007*, pages 114–129, 2007.
9. A. Fehnker, E. Clarke, S. Jha, and B. Krogh. Refining Abstractions of Hybrid Systems using Counterexample Fragments. In *HSCC 2005*, pages 242–257, 2005.
10. A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *HSCC*, pages 326–341, 2004.
11. G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *HSCC*, pages 258–273, 2005.
12. S. Graf and H. Saidi. Construction of abstact state graphs with PVS. In *CAV 1997*, pages 72–83, 1997.
13. T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In *POPL 2002*, pages 58–70, 2002.
14. T. A. Henzinger. The theory of hybrid automata. In *LICS*, pages 278–292, 1996.
15. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *CAV '97*, volume 1254 of *LNCS*, pages 460–483, 1997.
16. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *STOC '95*, pages 373–382, New York, NY, USA, 1995. ACM.
17. G. Holzmann and M. Smith. Automating software feature verification. *Bell Labs Technical Journal*, 5(2):72–87, 2000.
18. S. K. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *HSCC 2007*, pages 287–300, 2007.
19. M. Segelken. Abstraction and Counterexample-guided Construction of Omega-Automata for Model Checking of Step-discrete linear Hybrid Models. In *CAV 2007*, pages 433–448, 2007.
20. M. Sorea. Lazy approximation for dense real-time systems. In *Proc. of the Joint Conf. on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 363–378, 2004.

A Preliminaries

Let us call a region $R \subseteq \mathbb{R}^n$ a *real rectangular region* if $R = \prod_{i=1}^{n} (R)_i$ and $(R)_i$ is an interval of the form $[a, b]$, $[a, \infty)$, $(-\infty, b]$ or $(-\infty, +\infty)$ where $a, b \in \mathbb{R}$; a real rectangular region differs from a rectangular region in that the finite-endpoints of intervals need not be integers, but need to be reals. A *real multirectangular region* $R \subseteq \mathbb{R}^n$ is a finite union of real rectangular region. We denote by $RRR(n)$ the set of all real rectangular regions which are subsets of $\mathbb{R}^n$, and by RMR the set of all real multirectangular regions which are subsets of $\mathbb{R}^n$.

B CEGAR for Initialized Rectangular Hybrid Automata

In this section, we present a counterexample guided abstraction refinement method for the class of initialized rectangular hybrid automata along the lines of the general framework discussed in the previous section. We prove completeness of our CEGAR method.

B.1 Abstraction

In our CEGAR method, we will use abstractions which fall into the class of abstractions with linear variable abstraction functions. Hence we will first define a way to construct linear abstractions given linear functions of variables.

Definition Let $\mathcal{H}_C = (Loc_C,\ Edge_C,\ Source_C,\ Target_C,\ Var_C,\ Loc_C^0,\ Cont_C^0,\ Inv_C,\ Activity_C,\ Guard_C, Reset_C)$ be an *IRHA*. Let $\alpha_{Loc} : Loc_C \to Loc_A$ and $\alpha_{Edge} : Edge_C \to Edge_A$ onto functions, and $\alpha_{Var} \in LinFunc(|Var_C|, m)$, such that

- If $e_1, e_2 \in Edge_C$ are such that $\alpha_{Edge}(e_1) = \alpha_{Edge}(e_2)$ then $Source_C(e_1) = Source_C(e_2)$ and $Target_C(e_1) = Target_C(e_2)$.

We construct the abstract *IRHA* $\mathcal{H}_A = (Loc_A,\ Edge_A,\ Source_A,\ Target_A,\ Var_A,\ Loc_A^0,\ Cont_A^0,\ Inv_A,\ Activity_A,\ Guard_A, Reset_A)$ as follows.

- $Source_A(\alpha_{Edge}(e)) = \alpha_{Loc}(Source_C(e))$ and $Target_A(\alpha_{Edge}(e)) = \alpha_{Loc}(Target_C(e))$. Note this is well defined because of the conditions on α
- $Var_A = \{y_1, \cdots, y_m\}$.
- $Loc_A^0 = \alpha_{Loc}(Loc_C^0)$.
- $Cont_A^0 = RectHull(\alpha_{Var}(Cont_C^0))$.
- $Inv_A(q_A) = RectHull(\bigcup_{q_C \in \alpha_{Loc}^{-1}(q_A)} \alpha_{Var}(Inv_C(q_C)))$.
- $Activity_A(q_A) = RectHull(\bigcup_{q_C \in \alpha_{Loc}^{-1}(q_A)} \alpha_{Var}(Activity_C(q_C)))$.
- $Guard_A(e_A) = RectHull(\bigcup_{e_C \in \alpha_{Edge}^{-1}(e_A)} \alpha_{Var}(Guard_C(e_C)))$.
- $Reset_A(e_A, i) = Id$ if for all $j \in [|Var_C|]$ such that $Matrix(\alpha_{Var})(i, j) \neq 0$ and for all $e_C \in \alpha_{Edge}^{-1}(e_A)$, $Reset_C(e_C, j) = Id$.
  Otherwise, $Reset_A(e_A, i) = RectHull(\bigcup_{e_C \in \alpha_{Edge}^{-1}(e_A)}$ $\alpha_{Var}(\prod_{j \in [|Var_C|]} R(e_C, j)))$, where $R(e_c, j) = Reset_C(e_C, j)$ if $Reset_C(e_C, j) \neq id$, and $R(e_C, j) = (Guard(e_C))_j$ otherwise.

Proposition 2 *α is a hybrid abstraction from $\mathcal{H}_C$ to $\mathcal{H}_A$.*

Our CEGAR method will consider linear hybrid abstractions where $Matrix(\alpha_{Var})$ is a scaling matrix. The control state reachability problem for *IRHA* is solved by constructing a finite region graph which is analyzed for reachability [16]. We consider linear abstraction with scaling matrices since these are the only kinds of abstractions which preserve the property that the regions analyzed in the abstract model are a finite union of the regions analyzed in the concrete model.

B.2 Initial abstraction

Let us fix a concrete *IRHA* $\mathcal{H}_C$. We will use Loc_C, Var_C and so on to represent the elements of $\mathcal{H}_C$. As a general principle, we use the subscript in the automaton name to identify its elements. We can start with any linear hybrid abstraction α such that $Matrix(\alpha_{Var})$ is a scaling matrix and run the CEGAR loop. However, to keep the technical details simple, we assume that Loc_C^0 and q_C, the initial and the final concrete states, respectively, are mapped to unique states in the abstract automaton, i.e., $\alpha_{Loc}^{-1}(\alpha_{Loc}(Loc_C^0)) = Loc_C^0$ and $\alpha_{Loc}^{-1}(\alpha_{Loc}(q_C)) = q_C$. We will fix an abstract *IRHA* $\mathcal{H}_A$ and a linear hybrid abstraction α for the rest of the section.

B.3 Model-checking

The control state reachability problem is decidable for the class of initialized rectangular hybrid automata [16]. If the model-checking algorithm returns that the state $\alpha_{Loc}(q_C)$ is not reachable in $\mathcal{H}_A$, then we are done. Otherwise, it returns a counterexample σ' which we need to analyze.

B.4 Validation

Let $\sigma' = (q'_0, x'_0) \xrightarrow{a'_0} (q'_1, x'_1) \xrightarrow{a'_1} (q'_2, x'_2) \cdots (q'_{l-1}, x'_{l-1}) \xrightarrow{a'_{l-1}} (q'_l, x'_l)$ be an abstract counterexample in $\mathcal{H}_A$. The main part of validation is to be able to compute the sets $Reach_k$ for all $0 \leq k \leq l$ and check if there exists a k for which $Reach_k$ is empty. Before proving the computability of these set, we state a proposition about the shape of the $Reach_k$ sets.

Let us fix some notation. Let the set of variables in the concrete *HA* $\mathcal{H}_C$ be $Var_C = \{x_1, \cdots, x_n\}$ and the set of variables in the abstract *HA* $\mathcal{H}_A$ be $Var_A = \{y_1, \cdots, y_m\}$. Hence $|Var_C| = n$ and $|Var_A| = m$. Let $S \subseteq Loc_C \times \mathbb{R}^n$ be a set of concrete states. Then S restricted to the concrete state $q \in Loc_C$, denoted $Rest(S)_q$, is given by $\{x \,|\, (q, x) \in S\}$. We extend the definition of multirectangular region from subsets of $\mathbb{R}^n$ to subsets of $Loc_C \times \mathbb{R}^n$ in the natural way. We call S a real multirectangular region or $RMR(n)$ if $Rest(S)_q$ is $RMR(n)$ for each $q \in Loc_C$. The next lemma says that the $Reach_k$ sets are multirectangular provided the S_k sets are.

Lemma 1. *Suppose that S_k is RMR(n) for each k. Then for each k, $Reach_k$ is RMR(n).*

Proof. We will prove this by induction on k.

Case $k = l$: $Reach_l = (q_C \times \mathbb{R}^{|Var_C|}) \cap S_l$. Since $\mathbb{R}^{|Var_C|}$ and S_l are $RMR(n)$, and real multirectangular regions are closed under intersection, we have that $Reach_l$ is $RMR(n)$.

Case $0 \leq k < l$: $Reach_k = S_k \cap \bigcup_{a \in R} Pre_{\mathcal{H}_C}(Reach_{k+1}, a)$ for $0 \leq k < n$, where $R = \{a_k\}$ if $a_k \in \mathbb{R}_{\geq 0}$ and $R = \alpha^{-1}_{Edge}(a_k)$ otherwise. Note that R is always a finite set. Since real multirectangular regions are closed under finite unions and intersections, we can conclude that $Reach_k$ is $RMR(n)$ if we can show that $Pre_{\mathcal{H}_C}(Reach_{k+1}, a)$ is $RMR(n)$ for all a. By induction hypothesis, $Reach_{k+1}$ is $RMR(n)$. We will show that given any set $S \subseteq Loc_C \times \mathbb{R}^n$ which is $RMR(n)$ and $a \in \mathbb{R}_{\geq 0} \cup Edge_C$, $Pre_{\mathcal{H}_C}(S, a)$ is $RMR(n)$. Since $Pre_{\mathcal{H}_C}(S, a) = \cup_{q \in Loc_C} Pre_{\mathcal{H}_C}(Rest(S)_q, a)$, we can assume that $S = \{q\} \times R$ where $R \in RMR(n)$. Further since $Pre_{\mathcal{H}_C}(\{q\} \times R, a) = \cup_{i=1}^{t} Pre_{\mathcal{H}_C}(\{q\} \times R_i, a)$ where $R_i \in RRR(n)$ and $R = \cup_{i=1}^{t} R_i$, we can assume that $R \in RRR(n)$.

Case $a \in Edge_C$: The only interesting case is when $a = (p, q)$. Then $Pre(\{q\} \times R, a) = \{p\} \times \prod_{j=1}^{n} I_j$, where $I_j = (Guard(a))_j$ if $Reset(a, j) = Id$ and $I_j = (Guard(a))_j \cap Reset(a, j)$ otherwise. Since each of the I_js is rectangular, $Pre(\{q\} \times R, a)$ is $RRR(n)$, and hence is $RMR(n)$.

Case $a \in \mathbb{R}_{\geq 0}$. Given an interval $I = [l, u]$, an invariant $J = [l', u']$, activity $[r_l, r_u]$ and time $t \in \mathbb{R}_{\geq 0}$, let $preint(I, J, [r_l, r_u], t)$ be the set of all points x such that for some $r \in [r_l, r_u]$, $x + rt \in I$ and $x + rt' \in J$ for all $t' \in [0, t]$. $preint(I, J, [r_l, r_u], t)$ is an interval, since if x_1 and x_2 with $x_1 \leq x_2$ are two points satisfying the conditions above, then all $x \in [x_1, x_2]$ satisfy the condition with the r chosen to be the one used for x_1 or x_2. This interval can be computed. Therefore $Pre(\{q\} \times R, a) = \{q\} \times \prod_{j=1}^{n} preint((R)_j, Inv(q), Activity(q), a)$ is computable and rectangular.

Proposition 3 *If $Matrix(\alpha_{Var})$ is a scaling matrix, then S_k is RMR(n) for all k. Also $Post_{\mathcal{H}_C}(S_k, a_k)$ is RMR(n) for all $a \in \mathbb{R}_{\geq 0} \cup Edge_C$.*

Proof. $S_k = \alpha^{-1}_{Loc}(q'_k) \times \alpha^{-1}_{Var}(x'_k)$. If $A = Matrix(\alpha_{Var})$ is a scaling matrix, then for every j, if there is a row i of A such that $A(i, j) \neq 0$, then $(\alpha^{-1}_{Var}(x'_k))_j$ is a singleton (note that now other row of A has the j-th element not equal to 0). If there is not row of A for which $A(i, j) \neq 0$, then $(\alpha^{-1}(x'_K))_j = (-\infty, +\infty)$. And $\alpha^{-1}(x'_k)$ is the product of singleton sets and $(-\infty, +\infty)$ sets, hence it is multirectangular. So S_k in $RMR(n)$.

The proof of $Post_{\mathcal{H}_C}(S_k, a_k)$ is $RMR(n)$ is similar to the proof of $Pre_{\mathcal{H}_C}(S, a)$ is $RMR(n)$ where S is a $RMR(n)$.

Lemma 2. *$Reach_k$ is computable for each $0 \leq k \leq l$.*

Proof. Observe that the proof of Proposition 3 also shows that S_k is computable, and the proof of Lemma 1 shows that if S_k is $RMR(n)$ and computable, then $Reach_k$ is computable.

Since we can check for the emptiness of a multirectangular region, we can effectively check if the abstract counterexample is spurious. If $Reach_0 \neq \emptyset$, then there is a concrete counterexample reaching q_C, and we are done. Otherwise, σ' is a spurious counterexample and hence we use it to refine the abstraction $\mathcal{H}_A$.

B.5 Refinement

If $\sigma' = (q'_0, x'_0) \xrightarrow{a'_0} (q'_1, x'_1) \xrightarrow{a'_1} (q'_2, x'_2) \cdots (q'_{l-1}, x'_{l-1}) \xrightarrow{a'_{l-1}} (q'_l, x'_l)$ is a spurious counterexample, then we need to find a refinement which satisfies Condition $C2$. Next, we discuss how to obtain the required β. It is obtained in two steps.

We will first discuss the case where $a_{\hat{k}} \in Edge_A$. In the first step we obtain an abstraction β' which satisfies the following conditions.

- β'_{Loc} is same as α_{Loc} except possibly that it splits $q'_{\hat{k}+1}$
- β'_{Edge} is the same as α_{Edge} except possibly it splits the transition $a_{\hat{k}}$
- β'_{Var} is a scaling matrix in which all the non-zero entries are 1.
- Let $\mathcal{H}_{R'}$ be the abstract *IRHA* corresponding to β'. Then $(\cup_{a \in \beta'(\alpha^{-1}(a_{\hat{k}}))} Post_{\mathcal{H}_{R'}}(\beta'(S_{\hat{k}}), a)) \cap \beta'(Reach_{\hat{k}+1}) = \emptyset$.

Remark 4. Note that such a β' exists. If we split all the locations in $q'_{\hat{k}+1}$, all the transitions in $a_{\hat{k}}$, and take the scaling matrix to be diagonal matrix with all the diagonal values 1, then the last condition above translates into $Post_{\mathcal{H}_C}(S_{\hat{k}}) \cap Reach_{\hat{k}+1} = \emptyset$, which is true by the choice of $\hat{k}$. Note that every scaling matrix with non-zero entries 1 corresponds to just choosing a subset of the variables and associate scaling 1 to all of them. Hence we can enumerate all the splittings of the $q'_{\hat{k}+1}$ and $a_{\hat{k}}$, and all the subsets of Var_C, and verify that the resulting β' satisfies the above conditions.

In the next step, we update the scalings of various variables. We define a new abstraction $\beta = (\beta_{Loc}, \beta_{Edge}, \beta_{Var})$, where β_{Loc} is same as β'_{Loc} and β_{Edge} is same as β_{Edge}. β_{Var} is defined as follows. For every variable $v \in Var_{R'}$, set s_v to be the g.c.d of all the non-zero integer constants appearing in the intervals, corresponding to v, of the guards and resets of edges $e_{R'}$ of $\mathcal{H}_{R'}$ such that $e_{R'}$ has its source in $\beta'(\alpha^{-1}(q'_{\hat{k}}))$ and target in $Disc(\beta'(Reach_{\hat{k}+1}))$, where $Disc(S) = \{q \in Loc_C \mid \exists (q, x) \in S\}$. The variables in the abstraction being defined will be the union of the Var_A and $Var_{R'}$, and the scaling of a variable in β_{Var} will be the g.c.d of the scalings of the variable in α and β'_{Var} if it occurs in both Var_A and $Var_{R'}$, otherwise its value will the value of its scaling in α_{Var} or β'_{Var} depending on whether it occurs in Var_A or $Var_{R'}$ respectively. Let $\mathcal{H}_R$ be the abstraction defined by β.

We will informally explain why the *IRHA* $\mathcal{H}_R$ so defined will be a refinement satisfying condition $C2$. Firstly, it is easy to see that $\mathcal{H}_R$ is a refinement because the only operations it possibly does are the splitting of some location or edge of $\mathcal{H}_A$, addition of a new variables, or reduction of the scaling of a variable by some factor. Secondly, $\mathcal{H}_{R'}$ satisfies condition $C2$ by construction. Thirdly, our construction ensures that $\mathcal{H}_R$ also satisfies $C2$. Suppose not. Then there

exists $a \in \beta_{Edge}(\alpha^{-1}_{Edge}(a_{\hat{k}}))$, such that $Post_{\mathcal{H}_R}(\beta(S_{\hat{k}}), a)) \cap \beta(Reach_{\hat{k}+1}) \neq \emptyset$. Note β' and β have the same set of abstract edges. For those edges whose target is in $Disc(\beta'(Reach_{\hat{k}+1}))$, the post set in $\mathcal{H}_R$ and $\mathcal{H}_{R'}$ would be the same modulo some scaling and projection, due to the choice of the scaling. The same would be true for the sets $\beta(Reach_{\hat{k}+1})$ and $\beta'(Reach_{\hat{k}+1})$. This would imply that $Post_{\mathcal{H}_{R'}}(\beta'(S_{\hat{k}}), a)) \cap \beta'(Reach_{\hat{k}+1}) \neq \emptyset$, contradicting the assumption in the construction of $\mathcal{H}_{R'}$.

Next, we briefly sketch the case when $a_{\hat{k}}$ is a time transition. It is similar to the case where a_k is an edge transition, with the following differences. In this case, β'_{Edge} is the same as α_{Edge} (since $a_{\hat{k}}$ not a discrete jump). Next, in the second part, for each variable, s_v is chosen to be the g.c.d of the non-zero integer constants occurring in the intervals corresponding to v in the invariants and activities of the locations of $\mathcal{H}_{R'}$ which are in $Disc(\beta'(Reach_{\hat{k}+1}))$.

Termination of the algorithm. In each refinement step we either split some location of $\mathcal{H}_A$ or increase rows of the scaling matrix or change an entry in the scaling matrix. Otherwise β is same as α and Condition $C2$ fails. Since the number of locations and variables in the concrete *IRHA* are finite and the scaling cannot be less than 1, the algorithm will terminate in a finite number of steps.

Remark 5. An alternate proof of termination can be given by considering the regions of the timed automaton in the standard translation from Initialized Rectangular Automata to Timed Automata. For an *IRHA* $\mathcal{H}$, let us denote these regions by $Regions(\mathcal{H})$. It can be observed that $Regions(\mathcal{H}_R)$ refines $Regions(\mathcal{H}_A)$, that is, a region in $Regions(\mathcal{H}_A)$ is the union of one or more regions in $\mathcal{H}_R$. Since control state reachability is respected by the regions of the automata, the fact that we eliminate some potential counter example in each refinement step implies that the refinement of the regions is strict. Since the number of regions of a timed automaton are finite and the regions of the timed automaton corresponding to the $\mathcal{H}_C$ refine those of $\mathcal{H}_A$ and $\mathcal{H}_R$, we are guaranteed to terminate.

C Experiment Configurations

In this section, we will describe the navigation example model, over which the CEGAR algorithm was tested. We will start by describing the various configurations in the two dimensions and then explain how the three and four dimensional navigation examples are a natural extensions of the two dimensional case. Here, we partition the n-dimensional cude m^n times (i.e. lines parallel to the axis). Each of the resultant m^n regions is modeled as a control location of the hybrid automaton. A control location determines the rate of change of each of the n variables. In this paper, we have considered three specific configurations **A**, **B** and **C**. We denote the locations by tupes such as (x, y), (x, y, z), etc.

Configuration A The initial location in this configuration is the bottom left corner control state, denoted by $(1, 1)$, and the unsafe state is the top right corner, denoted by (m, m). For each control location (x, y) the navigation

direction at each of the control states is described as follows: if $x > m-2$ and $y > m-2$ then the robot will move North-East with velocity $\sqrt{2}$ else the robot will move South-West with velocity $\sqrt{2}$. An example for $m = 5$ is shown in Figure 2.

Configuration B The initial location in this configuration is the bottom left corner control state, denoted by $(1, 1)$, and the unsafe region is the bottom right corner control state, denoted by $(m, 1)$. For each control location (x, y) the navigation direction at each of the control states is described as follows: if $y = 1$ then the robot will move East with velocity 1, else the robot will move North with velocity 1. An example for $m = 5$ is shown in the figure below.

Configuration C This configuration was used in to test the CEGAR algorithm in higher dimensions (3 and 4 dimensions respectively). This configuration is essentially a generalization of the Configuration A. The initial state is the state denoted by $(1, 1, 1)$ in three dimensions. The unsafe state in this configuration is however the control location which is at the center of the n-dimensional cube, i.e. if $m = 5$, then the unsafe state is marked as the control state $(3, 3, 3)$ in the case of three dimensions and $(3, 3, 3, 3)$ in the case of four dimensions. The navigation direction in the case of three dimensions for the control state (x, y, z) is as follows: if $z = 1$ then the robot moves in the North-East direction with velocity $\sqrt{2}$ else the robot will move in the North-East direction with velocity $\sqrt{2}$ and it has downward velocity of 1. The resultant motion of the robot is the sum of both these velocities in three dimensions. Similarly in the case of 4 dimensions, The navigation direction for the control state (x, y, z, u) is as follows: if $z = 1$ then the robot moves in the North-East direction with velocity $\sqrt{2}$ else the robot will move in the North-East direction with velocity $\sqrt{2}$ and it has downward velocity of 1. The velocity in the u dimension is determined as follows : $u = 1$ then the velocity of robot in u dimension is zero else, the velocity of the robot in the negative u dimension is 1. The motion of the robot is a result of all the vector additions in the four dimensions.

We now describe the n-dimensional billiards ball model. The model represents the dynamics of a ball constrained in n-dimensions undergoing perfectly elastic collisions with the wall. The number of states in the hybrid automaton is $2^n + 2$, where one state corresponds to the initial state, one state to the unsafe state and the rest 2^n states represent the different directions in which the ball can move. For example, in the two dimensional case, the ball can move in North-East, North-West, South-East or South-West directions and each of these correspond to a control state in the hybrid automaton. Once the ball reaches a boundary, it changes its direction and this is represented as the change in the control state of the hybrid automaton. We have tested the experiment in two scenarios **A** and **B** where the initial position of the ball and the unsafe position are changed. In the configuration **A**, the initial position is $(0, 0)$ and the final position is $(10, 10)$ in the two dimensional case. And in the case of three dimensions the initial position is $(0, 0, 0)$ and the final position is $(10, 10, 10)$. In the configuration **B**, the initial

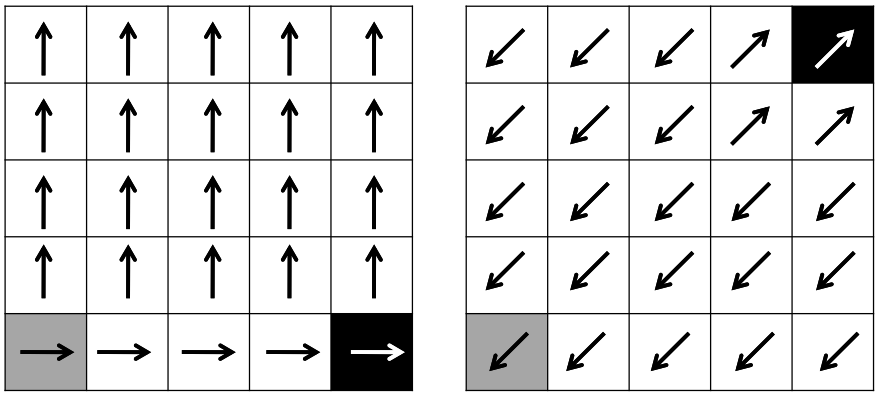

Fig. 2. 2-dimensional navigation grids. Configuration A (right) and Configuration B (left). Initial region is marked in grey and unsafe region is marked black.

position of the ball is $(0, 10)$ and the final position is $(35, 15)$ and in the case of three dimensions, the initial position is $(0, 10, 10)$ and the final position is $(35, 15, 35)$.