Reachability for Linear Hybrid Automata Using Iterative Relaxation Abstraction*

Sumit K. Jha[1], Bruce H. Krogh[2], James E. Weimer[2], and Edmund M. Clarke[1]

[1] Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213,USA
{jha|emc}@cs.cmu.edu
[2] ECE Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA15213, USA
{krogh|jweimer}@ece.cmu.edu

Abstract. This paper introduces *iterative relaxation abstraction* (IRA), a new method for reachability analysis of LHA that aims to improve scalability by combining the capabilities of current tools for analysis of low-dimensional LHA with the power of linear programming (LP) for large numbers of constraints and variables. IRA is inspired by the success of counterexample guided abstraction refinement (CEGAR) techniques in verification of discrete systems. On each iteration, a low-dimensional LHA called a *relaxation abstraction* is constructed using a subset of the continuous variables from the original LHA. Hybrid system reachability analysis then generates a regular language called the *discrete path abstraction* containing all possible counterexamples (paths to the bad locations) in the relaxation abstraction. If the discrete path abstraction is non-empty, a particular counterexample is selected and LP infeasibility analysis determines if the counterexample is spurious using the constraints along the path from the original high-dimensional LHA. If the counterexample is spurious, LP techniques identify an *irreducible infeasible subset* (IIS) of constraints from which the set of continuous variables is selected for the the construction of the next relaxation abstraction. IRA stops if the discrete path abstraction is empty or a legitimate counterexample is found. The effectiveness of the approach is illustrated with an example.

1 Introduction

Hybrid automata are a well studied formalism for representing and analyzing hybrid systems, that is, dynamic systems with both discrete and continuous state variables [1]. LHA are an important subclass of hybrid automata that can be

* This research was sponsored by the National Science Foundation under grant nos. CNS-0411152, CCF-0429120, CCR-0121547, and CCR-0098072, the US Army Research Office under grant no. DAAD19-01-1-0485, the Office of Naval Research under grant no. N00014-01-1-0796, the Defense Advanced Research Projects Agency under subcontract no. SA423679952, the General Motors Corporation, and the Semiconductor Research Corporation.

analyzed algorithmically and can asymptotically approximate hybrid automata with nonlinear continuous dynamics [2]. Tools for reachability analysis of LHA typically compute the sets of reachable states using polyhedra [3], but the sizes of the polyhedral representations can be exponential in the number of continuous variables of the LHA. Therefore, procedures for analysis of linear hybrid automata (LHA) do not scale well with the number of continuous state variables in the model. Although there has been considerable progress in the development of tools and algorithms for analyzing LHA [4,5], there is still a great need to develop new techniques that can handle high-dimensional LHA.

The approach to LHA reachability analysis proposed in this paper is inspired by the success of the *counterexample guided abstraction refinement* (CEGAR) technique for hardware and software verification [6,7,8]. In the CEGAR approach, in each iteration an abstraction of the original model (the concrete system) is constructed using only some of the state variables. The model with the smaller state space is then analyzed by a traditional model checking [9] algorithm. If this reduced model satisfies the given property, then the original system also satisfies the property and the algorithm terminates. Otherwise, the CEGAR loop picks a counterexample reported by the model checking algorithm, which determines a path in the location graph of the LHA. A decision procedure is then applied to the constraints along this path in the concrete system to determine if the counterexample is valid (the constraints can be satisfied by some run of the LHA) or spurious (the constraints cannot be satisfied) in the concrete system. The constraints used to check the feasibility of the counterexample involve all variables in the concrete system. If the constraints can be satisfied, the path corresponds to a true counterexample and the algorithm terminates. If the counterexample is spurious, a subset of variables is selected such that the constraints along the counterexample path are still infeasible by asking the decision procedure for an unsatisfiable core [10] or by using heuristics [11]. These variables are added to the set of variables used thus far and a new abstraction is constructed. On each iteration, the abstractions are therefore more refined and exclude any previously discovered spurious counterexamples.

The power of the above approach derives from its construction of abstractions with a small number variables for which model checking is feasible, while leveraging the power of decision procedures to deal with constraints involving many variables to test the feasibility of counterexamples in the original high-dimensional system. For LHA, we propose a similar approach in which full reachability analysis is performed on abstractions that have a small number of continuous variables. Linear programming (LP) is then applied as the decision procedure to check the validity of counterexamples using all of the variables in the original LHA. LP methods also find the variables to be used for constructing further abstractions. Linear programming for testing the feasibility of a path of a LHA was proposed in [12].

The following steps comprise this approach, which we call *iterative relaxation abstraction* (IRA). On each iteration, a low-dimensional LHA called a *relaxation abstraction* is first obtained by using a subset of the continuous variables from the original LHA. Hybrid system reachability analysis then generates a regular language called the *discrete path abstraction* containing all possible counterexamples (paths to the bad locations) in the relaxation abstraction. If the discrete path abstraction is non-empty, a particular counterexample is selected and linear programming (LP) determines if the counterexample is spurious using the constraints along the path from the original high-dimensional LHA. If the counterexample is spurious, infeasibility analysis of linear programs is applied to find an *irreducible infeasible subset* (IIS) of constraints [13] for the infeasible linear program corresponding to the spurious counterexample. The variables in the IIS are then used to construct the next relaxation abstraction. IRA stops if the discrete path abstraction is empty or a legitimate counterexample is found.

In the CEGAR loop for the analysis of discrete systems, the variables obtained from a spurious counterexample on each iteration are added to the set of variables used in previous iterations to construct a new abstraction. Such an approach would be counterproductive for LHA, however, as LHA reachability analysis does not scale well with increasing numbers of continuous variables. To avoid growth in the number of variables in the relaxation abstractions, only the variables in the current IIS are used on each iteration to construct the next relaxation abstraction, rather than adding these variables to the set of previously used variables. This assures that the number of variables in the LHA to which reachabililty analysis is performed is as small as possible. Counterexamples from previous iterations are excluded at each stage by intersecting the discrete path abstraction generated by the hybrid system analysis with the discrete path abstractions from previous iterations before checking for new counterexamples.

The paper is organized as follows. The next section introduces definitions and notation used throughout the paper. Section 3 defines the relaxation abstraction for LHA and a method for determining if a counterexample from a relaxation abstraction is also a counterexample for the original LHA. Section 4 presents the IRA procedure and Section 5 illustrates its application to a simple example. The performance of IRA is compared to the performance of PHAVer, a recently-developed LHA reachability analysis tool [5]. Section 6 summarizes the contributions of this paper and identifies directions for future work.

2 Preliminaries

2.1 Linear Constraints

We wish to apply a given set of constraints to different sets of variables. Therefore, we define a *linear constraint of order* m as a triple $l = (c, \sim, b)$ where $c = [c_1, \dots, c_m] \in \mathbb{R}^m$, $\sim \in \{=, \geq, \leq\}$, and $b \in R$. L^m denotes the set of all linear constraints of order m. Given an ordered set of m variables $X = \{X_1, \dots, X_m\}$ each ranging over the reals, l_X defines a (closed) *linear constraint over* X given by the expression $l_X : \sum_{i=1}^{m} c_i X_i \sim b$. For a given $x = [x_1, \dots, x_m] \in \mathbb{R}^m$, $l_X(x)$ denotes the value of the expression l_X (TRUE or FALSE) for the valuation $X_1 = x_1, X_2 = x_2, \dots, X_m = x_m$. Thus, l_X defines a predicate corresponding to a closed half-space in $\mathbb{R}^m$.

For $P \in \mathrm{FIN}(L^m)$, where $\mathrm{FIN}(A)$ denotes the set of finite subsets of a set A, $P_X \triangleq \bigwedge_{l \in P} l_X$, that is, P_X is the predicate over $\mathbb{R}^m$ defined by conjunction of the predicates determined by the linear constraints in P. The predicate P_X corresponds to a closed polyhedron in $\mathbb{R}^m$ defined by the intersection of the closed half-spaces determined by the linear constraints in P. We denote this polyhedron by $[\![P]\!]$ and write $P \Rightarrow P'$ to indicate that $[\![P]\!] \subseteq [\![P']\!]$.

Given a set of linear constraints $P \subset L^m$, the *support of* P, is defined as

$$support(P) = \{i \in \{1, \ldots, m\} | \; \exists \; l = (c, \sim, b) \in P \ni c_i \neq 0\}.$$

Given a second set of linear constraints $P' \subset L^m$, and a subset of indices $I \subset \{1, \ldots, m\}$, P' is said to be a *relaxation of* P *over* I, denoted $P' \sqsupseteq_I P$ if: (i) $P \Rightarrow P'$; and (ii) $support(P') \subseteq I$.

Example 1. If $P = \{([1\ 0\ 0], \geq, 0), ([0\ 1\ 0], \geq, 3), ([1\ 0\ 3], \geq, 8)\}$, $P' = \{([1\ 0\ 0], \geq, 0), ([0\ 1\ 0], \geq, 3)\}$, and $\mathcal{P}'' = \{([1\ 0\ 0], \geq, -1)\}$, then $P'' \sqsupseteq_{\{1\}} P' \sqsupseteq_{\{1,2\}} P$.

Relaxations of sets of linear constraints can be produced in many ways. For example, for an ordered set of m variables X, if X_I denotes the variables from X corresponding to the indices in a set $I \in \{1, \ldots, m\}$, applying the Fourier-Motzkin procedure [14] for existential quantifier elimination of the variables in $X - X_I$ from P_X produces the *projection* of P_X onto X_I. This corresponds to the tightest relaxation of P over I, which we denote by $proj_I(P)$. By "tightest" we mean that if P' is any relaxation of P over I, then $P' \sqsupseteq_I proj_I(P)$. A much looser relaxation of P is generated by simply eliminating the constraints involving variables not in X_I. We call this method of relaxation *localization* because of its similarity to the localization abstraction proposed by Kurshan for discrete systems [6]. Localization of the constraints in P to the variables with indices in I is given by

$$loc_I(P) = \{l \in P | support(l) \subseteq I\}.$$

A set of linear constraints $P \subset L^m$ is said to be *satisfiable* if there exists a valuation $x \in \mathbb{R}^m$ for a set X of m real-valued variables such that $P_X(x)$ is TRUE. We write $\mathrm{SAT}(P)$ to indicate a set of linear constraints is satisfiable. If P is not satisfiable (in which case we write $\mathrm{UNSAT}(P)$), we are interested in finding a minimal subset of constraints in P that is not satisfiable. This is known as an *irreducible infeasible subset* (IIS) of P, which is a subset $P' \subseteq P$ such that (i) $\mathrm{UNSAT}(P')$ and (ii) for any $l \in P'$, $\mathrm{SAT}(P' - \{l\})$ [13]. Although the problem of finding a minimal IIS (an IIS with the least number of constraints) is NP hard [15], several LP packages include functions implementing efficient heuristic procedures to compute IISs that are often minimal (e.g., MINOS (IIS)[16], CPLEX [17], IBM OSL [18], LINDO [19]).

2.2 Linear Hybrid Automata

Following [20], we define a *linear hybrid automaton* (LHA) [20] as a tuple $H = (G, n, \iota, \phi, \gamma, \rho)$, where

- $G = (Q, q_0, Q_{bad}, \Sigma, E)$ is the (labeled) *location graph* of H, where
  - Q is a finite set of *locations*;
  - $q_0 \in Q$ is the *initial* location;
  - $Q_{bad} \subset Q$ is the set of bad locations (the locations that should not be reachable);
  - Σ is a finite set of *labels*;
  - $E \subseteq (Q - Q_{bad}) \times \Sigma \times Q$ is finite set of (labeled) *transitions*, where no two outgoing transitions from a given location have the same label;
- n is the number of *continuous state variables*,
- $\iota : Q \longrightarrow \text{FIN}(L^n)$ identifies the *invariant* for each location.
- $\phi : Q \longrightarrow \text{FIN}(L^n)$ identifies the *flow constraints* for each location.
- $\gamma : E \longrightarrow \text{FIN}(L^n)$ identifies the *guard* for each transition.
- $\rho : E \longrightarrow \text{FIN}(L^{2n})$ identifies the *jump relation* for each transition.

Example 2. Figure 1 shows an LHA with continuous state variables $X = \{x_1, x_2\}$, discrete states $Q = \{q_0, q_1, q_2, q_3, q_4\}$, discrete state transition labels $\{a, b, c, d\}$, initial location q_0 (the unlabeled rectangle) with an invariant defining a unique initial continuous state $x(0) = \{.5, 0\}$, and $Q_{bad} = \{q_4\}$.

A *run* for $t \geq 0$ for an LHA H is a (possibly infinite) sequence of the form

$$q_0, x^0, \sigma_0, q_1, x^1, \sigma_1, q_2, x^2, \ldots,$$

where for all $k = 0, 1, \ldots$

- $x^k : [t_s^k, t_f^k] \to R^n$ denotes the continuous evolution of the continuous state variables for $t_s^k \leq t \leq t_f^k$, where $0 = t_s^0 \leq t_f^0 = t_s^1 \leq t_f^1 = t_s^2 \cdots$;
- $x^k(t) \in [\![\iota(q_k)]\!]$ (location invariants), where $t_s^k \leq t \leq t_f^k$;
- $\dot{x}^k \in [\![\phi(q_k)]\!]$ (flow constraints), where $t_s^k \leq t \leq t_f^k$;

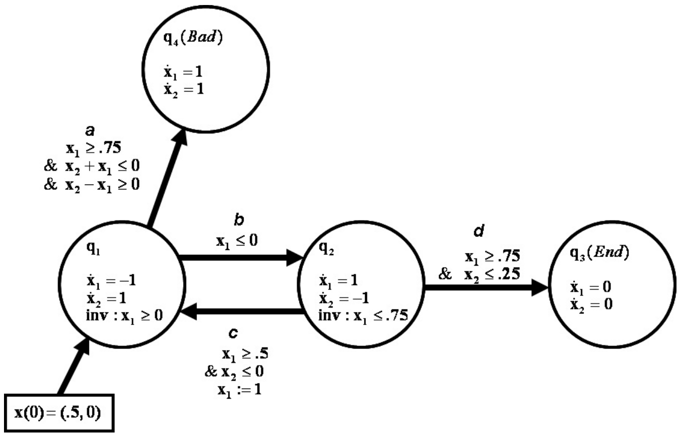

Fig. 1. An example LHA

- $(q_k, \sigma_{(k+1)}, q_{k+1}) \in E$ (transitions);
- $x^k(t_f^k) \in [\![\gamma((q_k, \sigma_{(k+1)}, q_{k+1}))]\!]$ (guards);
- $(x^k(t_f^k), x^k(t_s^{k+1})) \in [\![\rho((q_k, \sigma_k, q_{k+1}))]\!]$ (jump relation).

Projecting a run onto the transitions (i.e., eliminating the continuous state variable evolution) leads to a sequence of the form $\pi = q_0, \sigma_1, q_1, \sigma_2, q_2, \ldots$, which corresponds to a path in the location graph. Projecting a path onto the transition labels gives a sequence of labels, $\omega = \sigma_1, \sigma_2, \ldots$. Since the labels on the outgoing transitions from each location are distinct, the sequence ω corresponds to a unique path (π) in the location graph. A sequence of transition labels is said to be *feasible* if the path to which it corresponds could be generated by a run of the LHA; otherwise, the sequence is said to be *infeasible*. A path that leads to a state in Q_{bad} is called a *counterexample*.

We let $\mathcal{L}_{CE}(H)$ denote the set of all feasible sequences of transition labels generated by runs that lead to states in Q_{bad}. The definition of the transitions in the location graph precludes transitions from any state in Q_{bad}, therefore the sequences in $\mathcal{L}_{CE}(H)$ are all finite, that is, $\mathcal{L}_{CE}(H) \subseteq \Sigma^*$. $\mathcal{L}_{CE}(H)$ is not necessarily a regular language, however.

3 Relaxation Abstractions and Counterexample Analysis

In this section we first introduce a new class of abstractions for LHA based on relaxations of the linear constraint sets defining the invariants, flow constraints, guards, and jump relation for a given LHA. We then present a method using linear programming (LP) analysis for determining whether a counterexample for a relaxation abstraction is spurious for the original LHA.

Given an LHA $H = (G, n, \iota, \phi, \gamma, \rho)$ and an index set $I \subset \{1, \ldots, n\}$ with $|I| = n' < n$, a linear hybrid automaton $H' = (G', n', \iota', \phi', \gamma', \rho')$ is said to be a *relaxation of H over I*, denoted $H' \sqsupseteq_I H$, if

- $G' = G = (Q, q_0, Q_{bad}, \Sigma, E)$;
- for each $q \in Q$:
  - $\iota'(v) \sqsupseteq_I \iota(v)$ (invariants);
  - $\phi'(e) \sqsupseteq_I \phi(e)$ (flows);
- for each $e \in E$:
  - $\gamma'(e) \sqsupseteq_I \gamma(e)$ (guards);
  - $\rho'(e) \sqsupseteq_I \rho(e)$ (jump relations).

Example 3. Figure 2 shows a relaxed linear hybrid automaton derived from the LHA in Figure 1 over the index set $I = 1$. This relaxation is obtained by applying localization over I to each of the constraints in the original LHA.

Since the constraints defining a relaxation abstraction H' are constraints of the relation defining the original LHA H, it follows that any run for H is also a run for H'. Similarly, $\mathcal{L}_{CE}(H') \supseteq \mathcal{L}_{CE}(H)$.

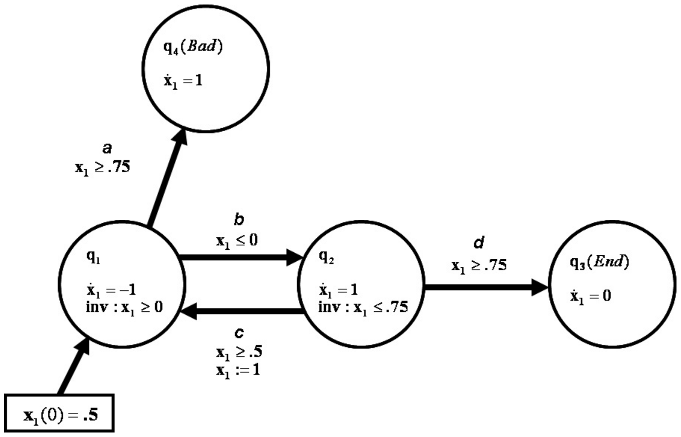

Fig. 2. A relaxation abstraction for the LHA in Fig. 1

Given an LHA H and an index set I, let H' be a relaxation of H over I. Given a counterexample for H', $ce = \sigma_1\sigma_2 \ldots \sigma_K \in \mathcal{L}_{CE}(H')$, there is a *unique* corresponding path in the location graph G of H of the form $\pi_{ce} = q_0\sigma_1 q_1\sigma_2 \ldots \sigma_K q_K$, where $q_K \in Q_{bad}$. To determine if there is a run for H corresponding to ce, we consider whether or not the constraints along this state-transition sequence are feasible as follows.

Given a feasible path $\pi = q_0\sigma_1 q_1\sigma_2 \ldots \sigma_K q_K$, we introduce the following variables:

– X_s^0, corresponding to the initial continuous state in q_0;
– $X_s^1, \ldots, X_s^K$, corresponding to the values of the continuous states when the transitions occur *into* locations $q^1, \ldots, q^K$, respectively;
– $X_f^0, \ldots, X_f^{K-1}$, corresponding to the values of the continuous states when the transitions occur *out of* locations $q^0, \ldots, q^{K-1}$, respectively;
– $\Delta_0, \Delta_1, \ldots, \Delta_{K-1}$, corresponding to the amount of time the run spends in $q^0, \ldots, q^{K-1}$, respectively.

We let V_π denote the set of variables defined above for a path π. From the constraints in H we construct the following constraints over the variables in V_π that must be satisfied by a valid run for H. We denote this collection of linear constraints by $\mathcal{C}(H,\pi)$ or $\mathcal{C}_\pi$ depending on the context:

– $\iota(q_0)_{X_s^0}$: the initial continuous states must be in the invariant of q_0;
– for $k = 1, \ldots, K$, the k^{th} transition in the path is $e_k = (q_{k-1}, \sigma_k, q_k)$ and for each transition:
  • $\iota(q_{k-1})_{X_f^{k-1}}$: the continuous state before the transition must satisfy the invariant of q_{k-1};

  - $\iota(q_k)_{X_s^k}$: the continuous state after the transition must satisfy the invariant of q_k; [1]
  - $\gamma(e_k)_{X_f^{k-1}}$: the continuous state before the transition must satisfy the guard of e_k;
  - $\rho(e_k)_{X_f^{k-1},X_s^k}$: the continuous states before and after the transition must satisfy the jump relation for e_k;
- for $k = 0, \ldots, (K-1)$:
  - $\hat{\phi}(q_k)_{(X_f^k, X_s^k, \Delta_k)}$, where $\hat{\phi}(q_k)$ is the set of linear constraints of the form $\hat{l} = ([c, -c, -b], \sim, 0)$, each corresponding to a constraint $l = (c, \sim, b) \in \phi(q_k)$.

The final constraint, which represents the flow constraint in each location, follows from the following derivation: for each $l = (c, \sim, b) \in \phi(q_k)$, $c\dot{x} \sim b$ for all $t_s \leq t \leq t_f$; this implies $c(x(t_f) - x(t_s)) = \int_{t_s}^{t_f} cx(\tau)d\tau \sim b(t_f - t_s)$; therefore, $cx(t_f) - cx(t_s) - b\Delta \sim 0$, where $\Delta = t_f - t_s$, which is the constraint $\hat{l}$.

As demonstrated in [12], a path is feasible if and only if the above linear constraints are feasible.

4 Iterative Relaxation Abstraction

We now present the IRA procedure to CEGAR-based reachability analysis of LHA. The following paragraphs describe the IRA steps shown in Fig. 3.

Step 1. Construct a relaxation abstraction over index set I_i of the LHA H, $H_i \sqsupseteq_{I_i} H$. Any relaxation method can be applied to the linear constraints in H.

Step 2. Compute the next discrete path abstraction $\mathcal{A}_{CE}^{i+1}$ (a regular language containing $\mathcal{L}_{CE}(H)$) as the intersection of the previous discrete path abstraction with $\widehat{\mathcal{L}}_{CE}(H_i)$, a regular language containing $\mathcal{L}_{CE}(H_i)$, and $(\Sigma^* - ce_i)$ to assure the previous counterexample is removed from the next discrete path abstraction.

Step 3. Choose a counterexample ce_{i+1} from $\mathcal{A}_{CE}^{i+1}$, or set $ce_{i+1} == null$ if $\mathcal{A}_{CE}^{i+1}$ is empty. This operation is performed by $Select\_CE(\mathcal{A}_{CE}^{i+1})$.

Step 4. $ce_{i+1} == null$ indicates that no bad states are reachable in the original linear hybrid automaton H since $\mathcal{A}_{CE}^{i+1}$ contains $\mathcal{L}_{CE}(H)$.

Step 5. Construct $C = \mathcal{C}(H, ce_{i+1})$, the set of linear constraints along the path in the location graph of H determined by ce_{i+1}.

Step 6. Apply LP to determine the feasibility of the constraints C. We know that SAT(C) if and only if ce_{i+1} is a valid counterexample in H [12].

[1] It is sufficient to check that the invariant holds at the beginning and end of the continuous state trajectory in each location because the invariants and flow constraints are convex [20].

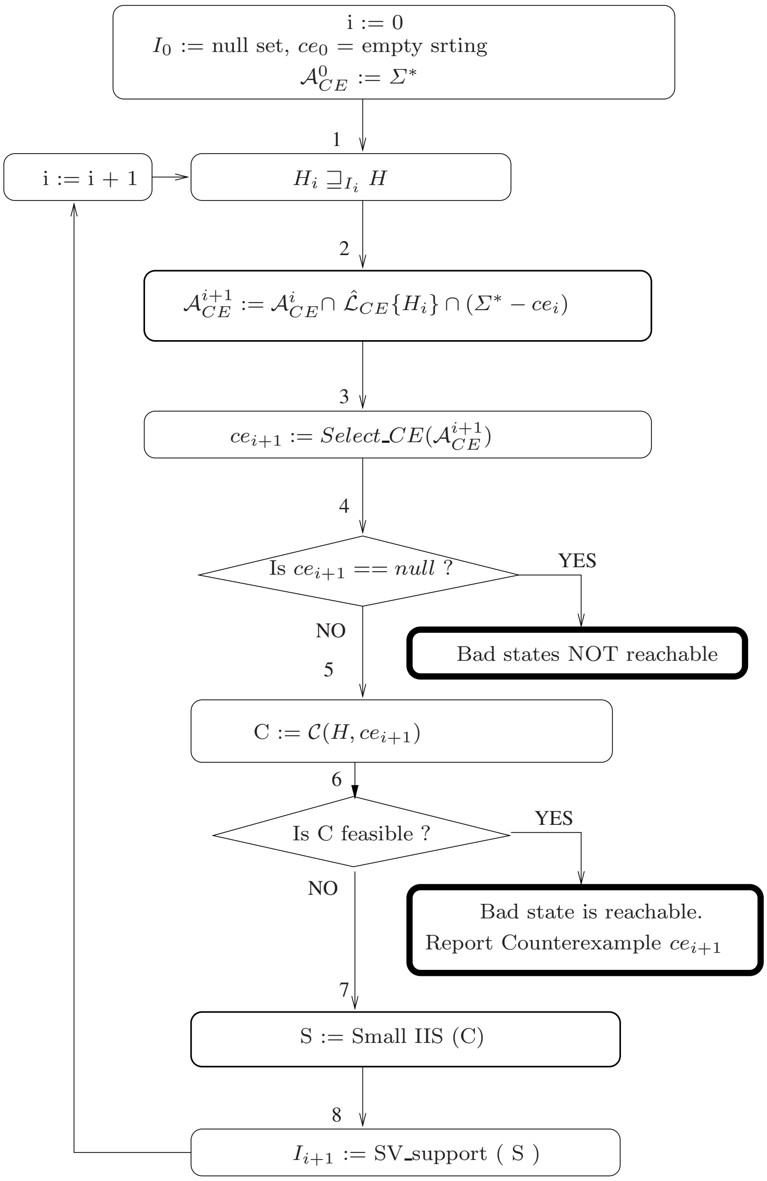
i := 0
I_0 := null set, ce_0 = empty srting
$\mathcal{A}^0_{CE} := \Sigma^*$
1
i := i + 1
$H_i \sqsupseteq_{I_i} H$
2
$\mathcal{A}^{i+1}_{CE} := \mathcal{A}^i_{CE} \cap \hat{\mathcal{L}}_{CE}\{H_i\} \cap (\Sigma^* - ce_i)$
3
$ce_{i+1} := Select\_CE(\mathcal{A}^{i+1}_{CE})$
4
Is $ce_{i+1} == null$?
YES
Bad states NOT reachable
NO
5
C := $\mathcal{C}(H, ce_{i+1})$
6
Is C feasible ?
YES
Bad state is reachable.
Report Counterexample ce_{i+1}
NO
7
S := Small IIS (C)
8
I_{i+1} := SV_support (S)

Fig. 3. The IRA procedure: Iterative relaxation abstraction reachability analysis for LHA

Step 7. If UNSAT(C), apply LP infeasibility analysis to find a minimal IIS for the constraints in C. (Heuristic procedures will actually find an IIS that may not be minimal.)

Step 8. Find the set of state variable indices corresponding to the variables with indices in *support*(C). This is the operation represented by the function $SV\_support(C)$.

Correctness of the IRA procedure (in the sense that if it terminates, it is correct) is guaranteed since the LHA H^i and languages $\mathcal{A}^i$ are overapproximations of H and $\mathcal{L}_{CE}(H)$, respectively. Although termination cannot be guaranteed (because $\hat{\mathcal{L}}_{CE}$ (H_i) maybe an overapproximation of $\mathcal{L}_{CE}(H_i)$), the sequence of discrete path abstractions generated on by the iterations is monotonically decreasing in size and any counterexample that has been analyzed is eliminated in future iterations.

5 Implementation and Example

The IRA has been implemented using PHAVer [5] for LHA reachability analysis and the CPLEX [17] library for LP analysis. PHAVer builds overapproximate discrete abstractions of the linear hybrid automata represented by finite automata. The discrete abstractions in the IRA tool are stored and manipulated using the AT&T FSM library [21]. The IRA tool provides users the ability to write their own relaxation functions. We experimented with two versions of IRA,

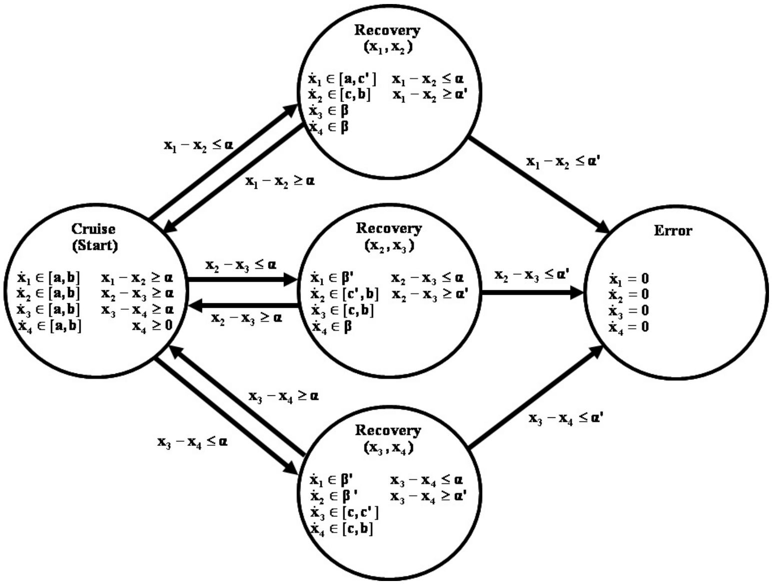

Fig. 4. A automated highway with 4 vehicles

Table 1. Comparison of analysis of Linear Hybrid Automata using PHAVer, IRA-Loc (localization relaxation) and IRA-FM (Fourier-Motzkin procedure)

	Time Taken in Seconds			Memory Used in KBs		
No of cars	**PHAVer**	**IRA-Loc**	**IRA-FM**	**PHAVer**	**IRA-Loc**	**IRA-FM**
6	0.26	1.34	61.05	5596	18344	452408
8	0.96	5.11	170.11	7532	20128	1000436
10	8.21	17.76	402.15	13904	22652	1876256
12	147.11	50.04	933.47	32800	26132	3155384
14	7007.51	123.73	1521.95	103408	30712	4194028
15	70090.06	181.74	2503.59	198520	33896	4193620
16	–	267.46	3519.51	–	36828	4194024
17	–	339.08	4741.75	–	40316	4194140
18	–	493.34	6384.94	–	44368	4194280
19	–	652.51	8485.49	–	49272	4194296

referred to as IRA-Localization and IRA-FM. IRA-Loc is the implementation that uses localization as the technique for building the relaxation. IRA-FM is another implementation which uses the Fourier Motzkin procedure for implementing quantifier elimination to build the relaxation.

As an example, we consider the model of a central arbiter for a automated highway and analyze the arbiter for safety properties, particularly for the specification that no two vehicles on the automated highway collide with each other. The electronic arbiter enforces speed limits on vehicles on the automated highway to achieve this purpose. The arbiter provides allowed ranges of velocities for each vehicle $[a, b]$. When two vehicles come within a distance α of each other, we call this a "possible" collision event. The arbiter asks the approaching car to slow down by reducing the upper bound to $[a, c']$ and asks the leading car to speed up by increasing the lower bound to $[c, b]$; it also requires that all other cars not involved in the possible collision slow down to a constant "recovery-mode" velocity β for cars behind the critical region and β' for cars in front of the critical region. When the distance between the two vehicles involved in the possible collision exceeds α, the arbiter model goes back to the dynamics of the cruise mode. The linear hybrid automata representing the case of four cars is shown in Fig. 4. The example is easily parameterized by varying the number of cars on the highway.

We ran this example on an AMD Opteron four-processor $x86\_64$ Linux machine running Fedora Core 5. The comparative results are shown in Table 1. In each case with n cars, both IRA versions verify in n-1 iterations that the bad states are not reachable. Consequently, the tighter Fourier Motzkin relaxation offer no advantage for this example. We expect, however, that tighter relaxations will be necessary to verify properties of more complex systems. Further empirical studies are currently being pursued. The plot of the log of the time taken vs. the

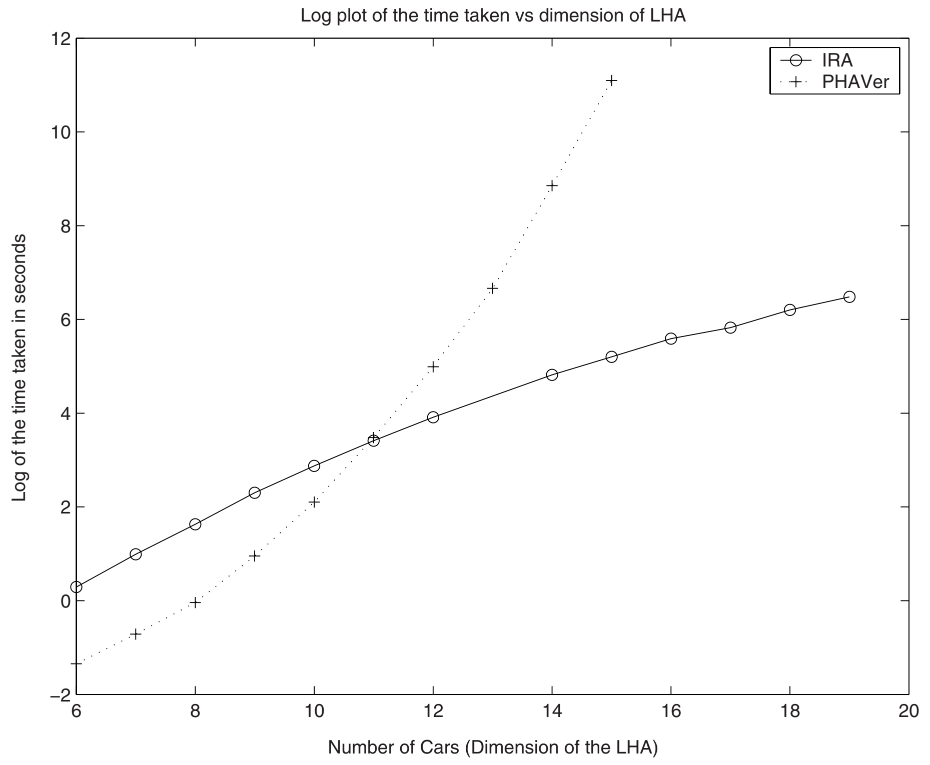

Fig. 5. Results: IRA-Loc vs. PHAVer

dimension of the LHA in Fig. 5 shows that PHAVer outperforms IRA-Loc for small dimensional systems. This happens as IRA-Loc spends time "reasoning" about the structure of the LHA and the possibility of reducing its dimension. For larger dimensions, IRA-Loc outperforms PHAVer significantly in both time and memory.

6 Discussion

IRA combines reachability analysis for low-dimensional LHA with the power of LP analysis for large numbers of variables. As proposed in [12], linear programming is used as an efficient counterexample validation algorithm. This idea of using linear programming as a counterexample validation is also used in [22]. As linear programming is in P [23], it is an efficient counterexample validation procedure for high dimensional LHA. Also, if a counterexample is found and validated, the reachability procedure can terminate immediately.

The IRA procedure uses linear constraint relaxation as a technique for generating abstractions of LHA. In contrast to previously proposed CEGAR techniques for hybrid system analysis in which abstractions are refined by splitting locations ([24],[25]), the relaxation abstraction retains the location graph as the original LHA.

We are working on extending the results in this paper to nonlinear hybrid automata. We are also currently evaluating the effectiveness of this procedure on a number of other benchmark problems.

Acknowldgments

The authors thank Xuandong Li and Prasad Sistla for several useful discussions, Goran Frehse for his help and guidance with PHAVer, and Alhad Arun Palkar for helping with the implementation.

References

1. Henzinger, T.: The Theory of Hybrid Automata. Lecture Notes in Computer Science (1996) 278
2. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138**(1) (1995) 3–34
3. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer **1**(1–2) (1997) 110–122
4. Alur, R., Henzinger, T., Wong-Toi, H.: Symbolic analysis of hybrid systems. In: Proc. 37-th IEEE Conference on Decision and Control. (1997)
5. Frehse, G.: PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. [26] 258–273
6. Kurshan, R.: Computer-aided Verification of Coordinating Processes: The Automata Theoretic Approach. Princeton University Press, 1994. (1994)
7. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification, London, UK, Springer-Verlag (2000) 154–169
8. Ball, T., Majumdar, R., Millstein, T.D., Rajamani, S.K.: Automatic Predicate Abstraction of C Programs. In: SIGPLAN Conference on Programming Language Design and Implementation. (2001) 203–213
9. E. M. Clarke, J., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA, USA (1999)
10. Zhang, L., Malik, S.: Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. In: DATE, IEEE Computer Society (2003) 10880–10885
11. Chaki, S., Clarke, E., Groce, A., Strichman, O.: Predicate abstraction with minimum predicates. In: Proceedings of 12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME). (2003)
12. Li, X., Jha, S.K., Bu, L.: Towards an Efficient Path-Oriented Tool for Bounded Reachability analysis of Linear Hybrid Systems using Linear Programming. (2006)
13. Chinneck, J., Dravnieks, E.: Locating minimal infeasible constraint sets in linear programs. ORSA Journal on Computing **3** (1991) 157–168
14. Dantzig, G.B., Eaves, B.C.: Fourier-Motzkin elimination and Its Dual. J. Comb. Theory, Ser. A **14**(3) (1973) 288–297

15. Sankaran, J.K.: A note on resolving infeasibility in linear programs by constraint relaxation. Operations Research Letters **13** (1993) 1920
16. Chinneck, J.W.: MINOS(IIS): Infeasibility analysis using MINOS. Comput. Oper. Res. **21**(1) (1994) 1–9
17. ILOG: (http://www.ilog.com/products/cplex/product/simplex.cfm)
18. Hung, M.S., Rom, W.O., Waren, A.D.: Optimization with IBM OSL and Handbook for IBM OSL (1993)
19. Systems Inc., L.: (http://www.lindo.com/products/api/dllm.html)
20. Ho, P.H.: Automatic Analysis of Hybrid Systems, Ph.D. thesis, technical report CSD-TR95-1536, Cornell University, August 1995, 188 pages (1995)
21. Mohri, M., Pereira, F., Riley, M.: The design principles of a weighted finite-state transducer library. Theoretical Computer Science **231**(1) (2000) 17–32
22. Jiang, S.: Reachability analysis of Linear Hybrid Automata by using counterexample fragment based abstraction refinement. submitted (2006)
23. Karmarkar, N.: A new polynomial-time algorithm for linear programming. Combinatorica **4**(4) (1984) 373–395
24. Fehnker, A., Clarke, E.M., Jha, S.K., Krogh, B.H.: Refining Abstractions of Hybrid Systems Using Counterexample Fragments. [26] 242–257
25. Alur, R., Dang, T., Ivancic, F.: Counterexample-guided predicate abstraction of hybrid systems. Theor. Comput. Sci. **354**(2) (2006) 250–271
26. Morari, M., Thiele, L., eds.: Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9-11, 2005, Proceedings. In Morari, M., Thiele, L., eds.: HSCC. Volume 3414 of Lecture Notes in Computer Science., Springer (2005)