Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 174 (2007) 57–70
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Towards an Efficient Path-Oriented Tool for Bounded Reachability Analysis of Linear Hybrid Systems using Linear Programming

Xuandong Li[a,b,1] , Sumit Jha Aanand[b,2] and Lei Bu[a,3]

[a] *State Key Laboratory of Novel Software Technology*
Department of Computer Science and Technology, Nanjing University
Nanjing, Jiangsu, P.R.China 210092

[b] *Computer Science Department, Carnegie Mellon University*
5000 Forbes Avenue, Pittsburgh, PA15213, USA

Abstract

The existing techniques for reachability analysis of linear hybrid automata do not scale well to problem sizes of practical interest. Instead of developing a tool to perform reachability check on all the paths of a linear hybrid automaton, a complementary approach is to develop an efficient path-oriented tool to check one path at a time where the length of the path being checked can be made very large and the size of the automaton can be made large enough to handle problems of practical interest. This approach of symbolic execution of paths can be used by design engineers to check important paths and thereby, increase the faith in the correctness of the system. Unlike simple testing, each path in our framework represents a dense set of possible trajectories of the system being analyzed. In this paper, we develop the linear programming based techniques towards an efficient path-oriented tool for the bounded reachability analysis of linear hybrid systems.

Keywords: Linear hybrid automata, bounded model checking, reachability analysis, linear programming.

1 Introduction

The model checking problem for hybrid systems is very difficult. Even for a relatively simple class of hybrid systems - the class of linear hybrid automata [1] - the most common problem of reachability is still undecidable [1]–[3].

Several model checking tools have been developed for linear hybrid automata, but they do not scale well to the size of practical problems. The state-of-the-art tool HYTECH [8] and its improvement PHAVer [9] need to perform expensive

[1] Email: lxd@nju.edu.cn
[2] Email: jha+@cs.cmu.edu
[3] Email: bl@seg.nju.edu.cn

doi:10.1016/j.entcs.2006.12.023

polyhedra computation, and their algorithm complexity is exponential in number of variables in the automata. In recent years, the bounded model checking has been presented as a complementary technique for BDD-based symbolic model checking, whose basic idea is to search for a counterexample in the model executions whose length is bounded by some integer k [5]. Several works [6,7] have been given to check linear hybrid systems using the bounded model checking technique. In these techniques, the model checking problems are reduced into the satisfiability problem of a boolean combination of propositional variables and mathematical constraints, but their experiment results show that the length of the checked model executions is still far from the practical problem size.

As the existing techniques cannot check all the paths for reachability analysis when attempting analysis of problem sizes that are of practical significance, a complementary approach is to develop an efficient path-oriented tool to check one path at a time where the length of the path being checked can be made very large and the size of the automaton can be made large enough to handle problems of practical interest. This approach of symbolic execution of paths can be used by the design engineers to check important paths and thereby, increase the faith in the correctness of the system. Unlike simple testing, each path in our framework represents a dense set of possible trajectories of the system being analyzed. In this paper, we present the linear programming based techniques towards development of an efficient path-oriented tool for the bounded reachability analysis of linear hybrid systems.

The paper is organized as follows. In next section, we define the class of linear hybrid automata considered in this paper. In section 3, we use linear programming to present our solution for the path-oriented bounded reachability analysis of linear hybrid automata. Section 4 presents some techniques to reduce the size of the linear programs corresponding to the paths that we are checking. The tool prototype and the case studies are described in section 5 . We give the conclusion in the last section.

2 Linear Hybrid Automata

The linear hybrid automata considered in this paper are a variation of the definition given in [1], in which the change rates of variables may be given a range of possible values. For simplicity, we suppose that in any linear hybrid automaton, considered in this paper, there is just one initial location with no initial conditions and no transitions to the initial location (we assume that each variable with an initial value is reset to the initial value by the transitions from the initial location).

Definition 2.1 A linear hybrid automaton is a tuple $H = (X, V, E, v_I, \alpha, \beta)$, where

- X is a finite set of real-valued variables. V is a finite set of *locations*.
- E is *transition* relation whose elements are of the form (v, ϕ, ψ, v') where v, v' are in V, ϕ is a set of *variable constraints* of the form $a \leq \sum_{i=0}^{m} c_i x_i \leq b$, and ψ is a set of *reset actions* of the form $x := c$ where $x_i \in X$ $(0 \leq i \leq m)$, $x \in X$, a, b, c

and c_i $(0 \le i \le m)$ are real numbers, and a and b may be ∞.

- v_I is an *initial* location.
- α is a labelling function which maps each location in $V - \{v_I\}$ to a *state invariant* which is a set of variable constraints of the form $a \le \sum_{i=0}^{m} c_i x_i \le b$ where $x_i \in X$ $(0 \le i \le m)$, a, b, and c_i $(0 \le i \le m)$ are real numbers, a and b may be ∞.
- β is a labelling function which maps each location in $V - \{v_I\}$ to a set of *change rates* which are of the form $\dot{x} = [a, b]$ where $x \in X$, and a, b are real numbers $(a \le b)$. For any location v, for any $x \in X$, there is one and only one change rate definition $\dot{x} = [a, b] \in \beta(v)$. □

Notice that the class of linear hybrid automata we consider here can be used to approximate any general hybrid automata to any desired level of accuracy because they are sufficiently expressive to allow asymptotically completeness of the abstraction process for a general hybrid automata [4].

We use the sequences of locations to represent the evolution of a linear hybrid automaton from location to location. For a linear hybrid automaton $H = (X, V, E, v_I, \alpha, \beta)$, a *path segment* is a sequence of locations of the form

$$v_1 \xrightarrow{(\phi_1,\psi_1)} v_2 \xrightarrow{(\phi_2,\psi_2)} \ldots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n$$

which satisfies $(v_i, \phi_i, \psi_i, v_{i+1}) \in E$ for each i $(1 \le i \le n-1)$. A *path* in H is a path segment starting at v_I.

The behavior of linear hybrid automata can be represented by *timed sequences*. Any timed sequence is of the form $(v_1, t_1)\hat{}(v_2, t_2)\hat{} \ldots \hat{}(v_n, t_n)$ where v_i $(1 \le i \le n)$ is a location and t_i $(1 \le i \le n)$ is a nonnegative real number. It represents a behavior of an automaton, that is, the system starts at the initial location and changes to the location v_1, stays there for t_1 time units, then changes to the location v_2 and stays at v_2 for t_2 time units, and so on.

Definition 2.2 For a linear hybrid automaton $H = (X, V, E, v_I, \alpha, \beta)$, a timed sequence $(v_1, t_1)\hat{}(v_2, t_2)\hat{} \ldots \hat{}(v_n, t_n)$ represents a behavior of H if and only if the following condition is satisfied:

- there is a path in H of the form $v_0 \xrightarrow{(\phi_0,\psi_0)} v_1 \xrightarrow{(\phi_1,\psi_1)} \ldots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n$;
- $t_1, t_2, \ldots, t_n$ satisfy all the variable constraints in ϕ_i $(1 \le i \le n-1)$, i.e. for each variable constraint $a \le c_0x_0 + c_1x_1 + \ldots + c_mx_m \le b$ in ϕ_i,

$$\delta_k \le \gamma_i(x_k) \le \delta'_k \text{ for any } k \ (0 \le k \le m), \text{ and}$$

$$a \le c_0\gamma_i(x_0) + c_1\gamma_i(x_1) + \ldots + c_m\gamma_i(x_m) \le b$$

where $\gamma_i(x_k)$ $(0 \le k \le m)$ represents the value of the variable x_k when the automaton stays at v_i with the delay t_i, and for any k $(0 \le k \le m)$,

$$\delta_k = d_k + u_{j_k+1}t_{j_k+1} + u_{j_k+2}t_{j_k+2} + \ldots + u_i t_i,$$

$$\delta'_k = d_k + u'_{j_k+1}t_{j_k+1} + u'_{j_k+2}t_{j_k+2} + \ldots + u'_i t_i,$$

$x_k := d_k \in \psi_{j_k}$ $(0 \le j_k < i)$, $x_k := d \notin \psi_l$ for any l $(j_k < l < i)$, and $\dot{x}_l = [u_l, u'_l] \in$

$\beta(v_l)$ for any l $(j_k < l \leq i)$; and

- $t_1, t_2, \ldots, t_m$ satisfy the state invariant for each location v_i $(1 \leq i \leq n)$, i.e.
  - for each variable constraint $a \leq c_0x_0 + c_1x_1 + \ldots + c_mx_m \leq b$ in $\alpha(v_i)$,

$$\delta_k \leq \gamma_i(x_k) \leq \delta'_k \text{ for any } k \ (0 \leq k \leq m), \text{ and}$$

$$a \leq c_0\gamma_i(x_0) + c_1\gamma_i(x_1) + \ldots + c_m\gamma_i(x_m) \leq b$$

where $\gamma_i(x_k)$ $(0 \leq k \leq m)$ represents the value of the variable x_k when the automaton stays at v_i with the delay t_i, and for any k $(0 \leq k \leq m)$,

$$\delta_k = d_k + u_{j_k+1}t_{j_k+1} + u_{j_k+2}t_{j_k+2} + \ldots + u_it_i,$$

$$\delta'_k = d_k + u'_{j_k+1}t_{j_k+1} + u'_{j_k+2}t_{j_k+2} + \ldots + u'_it_i,$$

$x_k := d_k \in \psi_{j_k}$ $(0 \leq j_k < i)$, $x_k := d \notin \psi_l$ for any l $(j_k < l < i)$, and $\dot{x}_l = [u_l, u'_l] \in \beta(v_l)$ for any l $(j_k < l \leq i)$; and

  - for each variable constraint $a \leq c_0x_0 + c_1x_1 + \ldots + c_mx_m \leq b$ in $\alpha(v_{i+1})$,

$$\delta_k \leq \gamma_i(x_k) \leq \delta'_k \text{ for any } k \ (0 \leq k \leq m), \text{ and}$$

$$a \leq c_0\lambda_i(x_0) + c_1\lambda_i(x_1) + \ldots + c_m\lambda_i(x_m) \leq b$$

where $\gamma_i(x_k)$ $(0 \leq k \leq m)$ represents the value of the variable x_k when the automaton stays at v_i with the delay t_i, if $x_k := e_{i_k} \in \psi_i$ $(0 \leq k \leq m)$ then $\lambda_i(x_k) = e_{i_k}$ else $\lambda_i(x_k) = \gamma_i(x_k)$, and for any k $(0 \leq k \leq m)$,

$$\delta_k = d_k + u_{j_k+1}t_{j_k+1} + u_{j_k+2}t_{j_k+2} + \ldots + u_it_i,$$

$$\delta'_k = d_k + u'_{j_k+1}t_{j_k+1} + u'_{j_k+2}t_{j_k+2} + \ldots + u'_it_i,$$

$x_k := d_k \in \psi_{j_k}$ $(0 \leq j_k < i)$, $x_k := d \notin \psi_l$ for any l $(j_k < l < i)$, and $\dot{x}_l = [u_l, u'_l] \in \beta(v_l)$ for any l $(j_k < l \leq i)$. □

3 Path-Oriented Bounded Reachability Analysis using Linear Programming

In this section we use linear programming to present a solution for the path-oriented bounded reachability analysis of linear hybrid automata.

3.1 Path-Oriented Bounded Reachability

For a linear hybrid automaton H, a reachability specification consists of a location v in H and a set φ of variable constraints, denoted by $\mathcal{R}(v, \varphi)$. We are concerned with the problem of checking whether a path in H satisfies a given reachability specification. The formal definition is presented below.

Definition 3.1 Let $H = (X, V, E, v_I, \alpha, \beta)$ be a linear hybrid automaton, and $\mathcal{R}(v, \varphi)$ be a reachability specification. A path ρ in H of the form

$$v_0 \xrightarrow{(\phi_0,\psi_0)} v_1 \xrightarrow{(\phi_1,\psi_1)} \ldots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n$$

satisfies $\mathcal{R}(v, \varphi)$ if and only if the following condition holds:

- $v_n = v$, and
- there is a behavior of H of the form $(v_1, t_1)\hat{}(v_2, t_2)\hat{}\ldots\hat{}(v_n, t_n)$ such that any variable constraint in φ is satisfied when the automaton stays at v_n with the delay t_n, i.e. for each variable constraint $a \leq c_0x_0 + c_1x_1 + \ldots + c_mx_m \leq b$ in φ,

$$\delta_k \leq \gamma_n(x_k) \leq \delta'_k \text{ for any } k \ (0 \leq k \leq m), \text{ and}$$
$$a \leq c_0\gamma_n(x_0) + c_1\gamma_n(x_1) + \ldots + c_m\gamma_n(x_m) \leq b$$

where $\gamma_n(x_k)$ $(0 \leq k \leq m)$ represents the value of the variable x_k when the automaton stays at v_n with the delay t_n, and for any k $(0 \leq k \leq m)$,

$$\delta_k = d_k + u_{i_k+1}t_{i_k+1} + u_{i_k+2}t_{i_k+2} + \ldots + u_nt_n\,,$$
$$\delta'_k = d_k + u'_{i_k+1}t_{i_k+1} + u'_{i_k+2}t_{i_k+2} + \ldots + u'_nt_n\,,$$

$x_k := d_k \in \psi_{i_k}$ $(0 \leq i_k < n)$, $x_k := d \notin \psi_j$ for any j $(i_k < j < n)$, and $\dot{x}_j = [u_j, u'_j] \in \beta(v_j)$ for any j $(i_k < j \leq n)$. □

3.2 Representation of a long path

Since our tool is designed to check a path which is as long as desired and can handle linear hybrid automata of practical problem size, we first need to represent such a long path.

For a linear hybrid automaton $H = (X, V, E, v_I, \alpha, \beta)$, we can represent a path segment ρ in H of the form

$$v_0 \xrightarrow{(\phi_0,\psi_0)} v_1 \xrightarrow{(\phi_1,\psi_1)} \ldots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n$$

by a simple form $v_0\hat{}v_1\hat{}\ldots\hat{}v_n$, which is called *simple regular expression.* A *simple regular expression* (SRE) R and the path segment $\mathcal{L}(R)$ it represents are defined recursively as follows:

- if $v \in V$, then v is a SRE, and $\mathcal{L}(v) = v$;
- if R_1 and R_2 are SREs and there is a transition in E from the last location in $\mathcal{L}(R_1)$ to the first location in $\mathcal{L}(R_2)$, then $R_1\hat{}R_2$ is a SRE, and

$$\mathcal{L}(R_1\hat{}R_2) = \mathcal{L}(R_1) \xrightarrow{(\phi,\psi)} \mathcal{L}(R_2)\,;$$

- if R is a SRE and there is a transition in E from the last location in $\mathcal{L}(R)$ to the first location in $\mathcal{L}(R)$, then R^k is a SRE where $k \geq 2$ is an integer, and

$$\mathcal{L}(R^k) = \underbrace{\mathcal{L}(R) \xrightarrow{(\phi,\psi)} \mathcal{L}(R) \xrightarrow{(\phi,\psi)} \ldots \xrightarrow{(\phi,\psi)} \mathcal{L}(R)}_{k}\,.$$

Using the above definition, we can represent a long path to be checked as a SRE, and the SREs can be used as a text language for the input of the tool.

3.3 Reducing the Bounded Reachability Problems into Linear Programs

Now we show how the problem of checking a path for a given reachability specification can be reduced to a linear program.

Let $H = (X, V, E, v_I, \alpha, \beta)$ be a linear hybrid automaton, $\mathcal{R}(v, \varphi)$ be a reachability specification, and ρ be a path in H of the form

$$v_0 \xrightarrow{(\phi_0,\psi_0)} v_1 \xrightarrow{(\phi_1,\psi_1)} \ldots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n$$

where $v_n = v$. For any timed sequence of the form $(v_1, t_1)\hat{}(v_2, t_2)\hat{}\ldots\hat{}(v_n, t_n)$, if it is such that ρ satisfies $\mathcal{R}(v, \varphi)$, then the following condition must hold:

- $t_1, t_2, \ldots, t_n$ satisfy all the variable constraints in $\phi_i (0 \le i \le n)$,
- $t_1, t_2, \ldots, t_n$ satisfy all the variable constraints in $\alpha(v_i)$ $(1 \le i \le n)$, and
- $t_1, t_2, \ldots, t_n$ satisfy all the variable constraints in φ,

which form a group of linear inequalities on $t_1, t_2, \ldots, t_n$ (see Definition 2.2 and 3.1), denoted by $\Theta(\rho, \mathcal{R}(v, \varphi))$. It follows that we can check if ρ satisfies $\mathcal{R}(v, \varphi)$ by checking if the group $\Theta(\rho, \mathcal{R}(v, \varphi))$ of linear inequalities has a solution, which can be solved by linear programming.

In addition to $t_1, t_2, \ldots, t_n$, each $\gamma_i(x_k)$ in Definition 2.2 and 3.1 also becomes a variable in the linear program corresponding to checking of a path. Notice that if the change rate of x_k is a constant ($\dot{x}_k = [a, a]$), then $\delta_k = \delta'_k$ in Definition 2.2 and 3.1 such that we can replace $\gamma_i(x_k)$ with δ_k. Thus, for a path checking, the numbers of the variables and the constraints in the corresponding linear program can be calculated as follows:

- we have one variable in the linear program for each location in the path,
- we have at most one variable in the linear program for each variable occurrence in a variable constraint labelled on a transition, in a location invariant, and in the reachability specification,
- for each variable occurrence in a variable constraint labelled on a transition, in a location invariant, and in the reachability specification, we have at most one constraint in the linear program,
- for each variable constraint labelled on a transition, we have one constraint in the linear program,
- for each variable constraint in a location invariant, we have two constraints in the linear program, and
- for each variable constraint in the reachability specification, we have one constraint in the linear program.

Thanks to the advances in computing during the past decade, linear programs in a few thousand variables and constraints are nowadays viewed as"small". Problems having tens or hundreds of thousands of continuous variables are regularly solved. Indeed, many software packages have been developed to efficiently find solutions for linear programs. Leveraging the research in efficient solution of linear programs, we can develop an efficient tool to check a path in a linear hybrid automaton, where the length of the path and the size of the linear hybrid automaton are both closer to the practical problem sizes.

4 Reducing Size of Linear Programs Corresponding to Path Checking

We have reduced the bounded reachability analysis for a given path into a linear programming problem. In this section, we present several techniques for reducing the size of the resulting linear programming problem so that our tool can be used to solve problems of size as large as possible.

4.1 Decomposing Linear Programs Corresponding to Path Checking

In some cases, we can decompose the linear program corresponding to the path being checked into several smaller linear programs so that the tool can check longer paths.

Let $H = (X, V, E, v_I, \alpha, \beta)$ be a linear hybrid automaton, $\mathcal{R}(v, \varphi)$ be a reachability specification, and ρ be a path in H of the form

$$v_0 \xrightarrow{(\phi_0,\psi_0)} v_1 \xrightarrow{(\phi_1,\psi_1)} \dots \xrightarrow{(\phi_{i-1},\psi_{i-1})} v_i \xrightarrow{(\phi_i,\psi_i)} v_{i+1} \xrightarrow{(\phi_{i+1},\psi_{i+1})} \dots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n$$

where $v_n = v$. If there is i $(0 < i < n)$ such that

- for any variable x occurring in a variable constraint in ϕ_j $(i < j < n)$, x is reset on a transition $(v_k, \phi_k, \psi_k, v_{k+1})$ $(i \le k < j)$, i.e. $x := a \in \psi_k$,
- for any variable x occurring in a variable constraint in $\alpha(v_j)$ $(i < j \le n)$, x is reset on a transition $(v_k, \phi_k, \psi_k, v_{k+1})$ $(i \le k < j)$, i.e. $x := a \in \psi_k$, and
- for any variable x occurring in a variable constraint in φ, x is reset on a transition $(v_k, \phi_k, \psi_k, v_{k+1})$ $(i \le k < n)$, i.e. $x := a \in \psi_k$,

then the linear program corresponding to checking ρ for $\mathcal{R}(v, \varphi)$ can be decomposed. In this case, there is a timed sequence of the form

$$(v_1, t_1)\hat{}(v_2, t_2)\hat{} \dots \hat{}(v_i, t_i)\hat{}(v_{i+1}, t_{i+1})\hat{}(v_{i+2}, t_{i+2})\hat{} \dots (v_n, t_n)$$

such that ρ satisfies $\mathcal{R}(v, \varphi)$ if and only if the following condition holds:

- $t_1, t_2, \dots, t_i$ satisfy all variable constraints in ϕ_k $(1 \le k \le i)$, and all variable constraints in $\alpha(v_k)$ $(1 \le k \le i)$, and
- $t_{i+1}, t_{i+2}, \dots, t_n$ satisfy all variable constraints in ϕ_k $(i < k \le n)$, all variable constraints in $\alpha(v_k)$ $(i < k \le n)$, and all variable constraints in φ,

which correspond to two separate linear programs according to Definition 2.2 and 3.1. Thus, in this case we can decompose the linear program corresponding to a path checking into two smaller linear programs. The resulting linear programs can be recursively decomposed by the same technique until the technique can no longer be applied.

4.2 Shortening Paths

For a path segment ρ in a linear hybrid automaton, its *length* $|\rho|$ is the number of the locations in ρ. Since the size of the linear program corresponding to the path being checked is proportional to the length of the path, shortening the path will improve the complexity of the overall method. By shortening a path, we mean to find a shorter path in lieu of the path being checked such that both of them are equivalent with respect to the given reachability specification - if one of them satisfies the reachability specification, so does the other.

For a linear hybrid automaton $H = (X, V, E, v_I, \alpha, \beta)$, a long path ρ in H, which we want to check, usually includes repetitions of path segments, which can be represented as the following form:

$$\rho = v_0 \overset{(\phi_0,\psi_0)}{\longrightarrow} \dots \overset{(\phi_{i-1},\psi_{i-1})}{\longrightarrow} v_i \overset{(\phi_i,\psi_i)}{\longrightarrow} \rho_1^k \overset{(\phi,\psi)}{\longrightarrow} v_{i+1} \overset{(\phi_{i+1},\psi_{i+1})}{\longrightarrow} \dots \overset{(\phi_{n-1},\psi_{n-1})}{\longrightarrow} v_n$$

where ρ_1 is a path segment in H, $k \geq 2$ is an integer, and ρ_1^k represents the path segment

$$\underbrace{\rho_1 \overset{(\phi',\psi')}{\longrightarrow} \rho_1 \overset{(\phi',\psi')}{\longrightarrow} \dots \overset{(\phi',\psi')}{\longrightarrow} \rho_1}_{k} .$$

In the following, we show that in some cases we can find $k' < k$ such that ρ satisfies a given reachability specification if and only if ρ' satisfies the reachability specification where ρ' is of the form

$$\rho' = v_0 \overset{(\phi_0,\psi_0)}{\longrightarrow} \dots \overset{(\phi_{i-1},\psi_{i-1})}{\longrightarrow} v_i \overset{(\phi_i,\psi_i)}{\longrightarrow} \rho_1^{k'} \overset{(\phi,\psi)}{\longrightarrow} v_{i+1} \overset{(\phi_{i+1},\psi_{i+1})}{\longrightarrow} \dots \overset{(\phi_{n-1},\psi_{n-1})}{\longrightarrow} v_n .$$

Let $H = (X, V, E, v_I, \alpha, \beta)$ be a linear hybrid automaton, $\mathcal{R}(v, \varphi)$ be a reachability specification, and ρ be a path in H of the form

$$v_0 \overset{(\phi_0,\psi_0)}{\longrightarrow} v_1 \overset{(\phi_1,\psi_1)}{\longrightarrow} \dots \overset{(\phi_{n-1},\psi_{n-1})}{\longrightarrow} v_n$$

where $v_n = v$. We say that a variable constraint is *related to* a location v_i $(0 \leq i \leq n)$ if it is in ϕ_i, $\alpha(v_i)$, or in φ when $i = n$. We define the *reference point* for a variable in a variable constraint related to a location in ρ as follows:

- for a variable x in a variable constraint related to a location v_i $(0 \leq i \leq n)$, a location v_j $(0 \leq j < i)$ is the *reference point* if x is reset on the transition $(v_j, \phi_j, \psi_j, v_{j+1})$ $(x := a \in \psi_j)$, and is not reset on any transition $(v_k, \phi_k, \psi_k, v_{k+1})$ $(j < k < i)$ $(x := b \notin \psi_k)$ (in this case, we say that a is the *reference value* of x on v_i).

Let $H = (X, V, E, v_I, \alpha, \beta)$ be a linear hybrid automaton, $\mathcal{R}(v, \varphi)$ be a reachability specification, and ρ be a path in H of the form $\rho = \rho_1 \overset{(\phi,\psi)}{\longrightarrow} \rho_2^k \overset{(\phi',\psi')}{\longrightarrow} \rho_1'$ where $k > 3$, ρ_1 is a path, and ρ_1', ρ_2 are path segments. We say that ρ_2^k is *closed* in ρ if the following condition holds:

- $\rho_2^k = \rho_{21} \overset{(\phi'',\psi'')}{\longrightarrow} \rho_3^{k-2} \overset{(\phi'',\psi'')}{\longrightarrow} \rho_{21}'$ where $\rho_2^2 = \rho_{21} \overset{(\phi'',\psi'')}{\longrightarrow} \rho_{21}'$ and $|\rho_{21}| \leq |\rho_{21}'|$, and
- $\rho_3 = v_1 \overset{(\phi_1,\psi_1)}{\longrightarrow} v_2 \overset{(\phi_2,\psi_2)}{\longrightarrow} \dots \overset{(\phi_{n-1},\psi_{n-1})}{\longrightarrow} v_n$, and for any x occurring in a variable constraints in ϕ_i or $\alpha(u_i)$ $(1 \leq i \leq n)$, $x := a \in \psi''$ or $x := a \in \psi_j$ $(1 \leq j < i)$.

Theorem 4.1 Let $H = (X, V, E, v_I, \alpha, \beta)$ be a linear hybrid automaton, $\mathcal{R}(v, \varphi)$ be a reachability specification, and $\rho = \rho_1 \xrightarrow{(\phi,\psi)} \rho_2^k \xrightarrow{(\phi',\psi')} \rho_1'$ be a path in H where ρ_2^k $(k > 3)$ is closed in ρ. If any location in ρ_1 is not the reference point for any variable in a variable constraint related to a location in ρ_1', then ρ satisfies $\mathcal{R}(v, \varphi)$ if and only if ρ' satisfies $\mathcal{R}(v, \varphi)$ where $\rho' = \rho_1 \xrightarrow{(\phi,\psi)} \rho_2^3 \xrightarrow{(\phi',\psi')} \rho_1'$.

Proof. Suppose that $\rho_2^k = \rho_{21} \xrightarrow{(\phi'',\psi'')} \rho_3^{k-2} \xrightarrow{(\phi'',\psi'')} \rho_{21}'$, and

$$\rho_3 = v_1 \xrightarrow{(\phi_1,\psi_1)} v_2 \xrightarrow{(\phi_2,\psi_2)} \dots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n \, .$$

It follows that $\rho = \rho_1'' \xrightarrow{(\phi'',\psi'')} \rho_3^{k-2} \xrightarrow{(\phi'',\psi'')} \rho_1'''$, and $\rho' = \rho_1'' \xrightarrow{(\phi'',\psi'')} \rho_3 \xrightarrow{(\phi'',\psi'')} \rho_1'''$. The half of the claim, if ρ satisfies $\mathcal{R}(v, \varphi)$ then ρ' satisfies $\mathcal{R}(v, \varphi)$, can be proved as follows. Since ρ satisfies $\mathcal{R}(v, \varphi)$, suppose that the corresponding timed sequence $\sigma = \sigma_1''\hat{\ }\sigma_r\hat{\ }\sigma_1'''$ satisfies the condition given in Definition 3.1 where σ_r corresponds to ρ_3^{k-2}. It follows that $\sigma_r = \sigma_{r1}\hat{\ }\sigma_{r2}\hat{\ }\dots\hat{\ }\sigma_{rk-2}$, and that each σ_{ri} $(1 \le i \le k-2)$ is of the form $(v_1, t_1)\hat{\ }(v_2, t_2)\hat{\ }\dots\hat{\ }(v_n, t_n)$ such that $t_1, t_2, \dots, t_n$ satisfies the condition in Definition 2.2. Since ρ_2^k is closed in ρ and any location in ρ_1 is not the reference point for any variable in a variable constraint related to a location in ρ_1', by removing $\sigma_{r2}\hat{\ }\sigma_{r3}\hat{\ }\dots\hat{\ }\sigma_{rk-2}$ from σ we get a timed sequence σ' which satisfies the condition in Definition 3.1 and corresponds ρ'. It follows that ρ' satisfies $\mathcal{R}(v, \varphi)$. The other half of the claim can be proved as follows. Since ρ' satisfies $\mathcal{R}(v, \varphi)$, suppose that the corresponding timed sequence $\sigma' = \sigma_1''\hat{\ }\sigma_3\hat{\ }\sigma_1'''$ satisfies the condition given in Definition 3.1 where σ_3 corresponds to ρ_3. Since ρ_2^k is closed in ρ and any location in ρ_1 is not the reference point for any variable in a variable constraint related to a location in ρ_1', by replacing σ_3 with $\underbrace{\sigma_3\hat{\ }\sigma_3\hat{\ }\dots\hat{\ }\sigma_3}_{k-2}$ in σ' we get a timed sequence σ which satisfies the condition in Definition 3.1 and corresponds ρ. It follows that ρ satisfies $\mathcal{R}(v, \varphi)$. $\square$

Let $H = (X, V, E, v_I, \alpha, \beta)$, and ρ be a path in H of the form

$$v_0 \xrightarrow{(\phi_0,\psi_0)} \dots \xrightarrow{(\phi_{i-1},\psi_{i-1})} v_i \xrightarrow{(\phi_i,\psi_i)} \dots \xrightarrow{(\phi_{j-1},\psi_{j-1})} v_j \xrightarrow{(\phi_j,\psi_j)} \dots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n \, .$$

A variable constraint $a \le \sum_{k=0}^{m} c_k x_k \le b$ related to v_j $(1 \le j \le n)$ is *positive* in ρ if the following condition holds:

- $c_k \ge 0$ for any k $(0 \le k \le m)$, and
- for any x_k $(0 \le x \le m)$, if the reference point is v_i, then any v_l $(i < l \le j)$ is such that if $\dot{x_k} = [a, b] \in \beta(v_l)$ then $a \ge 0$,

and we say that $b - \sum_{i=0}^{k} c_k d_k$ is the *bound* of the variable constraint where d_k $(0 \le k \le m)$ is the reference value of x_k on v_j.

Let $H = (X, V, E, v_I, \alpha, \beta)$ be a linear hybrid automaton, $\mathcal{R}(v, \varphi)$ be a reachability specification, and $\rho = \rho_1 \xrightarrow{(\phi,\psi)} \rho_2^k \xrightarrow{(\phi',\psi')} \rho_1'$ be a path in H where $\rho_2^k = \rho_{21} \xrightarrow{(\phi'',\psi'')} \rho_3^{k-2} \xrightarrow{(\phi'',\psi'')} \rho_{21}'$ $(k > 3)$ is closed in ρ, and

$$\rho_3 = v_1 \xrightarrow{(\phi_1,\psi_1)} v_2 \xrightarrow{(\phi_2,\psi_2)} \dots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n \, .$$

If there is a positive variable constraint $a \le \sum_{i=0}^{m} c_i x_i \le b$ related to a location in ρ_1' such that

- there is a variable set $\omega \subseteq \{x_0, x_1, \ldots, x_m\}$ ($\omega \ne \emptyset$) such that for any $x \in \omega$, its reference point is in ρ_1, and
- $\xi > 0$ where ξ is the infimum of the set

$$\left\{ \sum_{i=0}^{m} c_i' \delta_i \;\middle|\; \begin{array}{l} \text{if } x_i \in \omega \text{ then } c_i' = c_i \text{ else } c_i' = 0 \text{ for any } i\ (0 \le i \le m); \\ \text{for any } i\ (0 \le i \le m),\ \delta_i = u_{i1}t_1 + u_{i2}t_2 + \ldots + u_{in}t_n \\ \text{where } \dot{x_i} = [u_{ij}, u_{ij}'] \in \beta(v_j) \text{ for any } j\ (1 \le j \le n); \text{ and} \\ (v_1, t_1)\hat{}(v_2, t_2)\hat{}\ldots\hat{}(v_n, t_n) \text{ is a timed sequence such that} \\ t_1, t_2, \ldots, t_n \text{ satisfy the condition in Definition 2.} \end{array} \right\}$$

(notice that ξ can be calculated by linear programming),

then we say that p_2^k is *constrained* by $\lfloor \zeta/\xi \rfloor + 3$ where ζ is the bound of the variable constraint $a \le \sum_{i=0}^{m} c_i x_i \le b$.

Theorem 4.2 Let $H = (X, V, E, v_I, \alpha, \beta)$ be a linear hybrid automaton, $\mathcal{R}(v, \varphi)$ be a reachability specification, and $\rho = \rho_1 \xrightarrow{(\phi,\psi)} \rho_2^k \xrightarrow{(\phi',\psi')} \rho_1'$ be a path in H where ρ_2^k ($k > 3$) is closed in ρ, and constrained by k'. If $k > k'$ then ρ does not satisfy $\mathcal{R}(v, \varphi)$.

Proof. Suppose that $\rho_2^k = \rho_{21} \xrightarrow{(\phi'',\psi'')} \rho_3^{k-2} \xrightarrow{(\phi'',\psi'')} \rho_{21}'$, and

$$\rho_3 = v_1 \xrightarrow{(\phi_1,\psi_1)} v_2 \xrightarrow{(\phi_2,\psi_2)} \ldots \xrightarrow{(\phi_{n-1},\psi_{n-1})} v_n\,.$$

It follows that $\rho = \rho_1'' \xrightarrow{(\phi'',\psi'')} \rho_3^{k-2} \xrightarrow{(\phi'',\psi'')} \rho_1'''$. Suppose that ρ satisfies $\mathcal{R}(v, \varphi)$, and the corresponding timed sequence $\sigma = \sigma_1''\hat{}\sigma_r\hat{}\sigma_1'''$ satisfies the condition given in Definition 3.1 where σ_r corresponds to ρ_3^{k-2}. It follows that

$$\sigma_r = \sigma_{r1}\hat{}\sigma_{r2}\hat{}\ldots\hat{}\sigma_{rk-2}$$

where each σ_{ri} ($1 \le i \le k-2$) is of the form $(v_1, t_1)\hat{}(v_2, t_2)\hat{}\ldots\hat{}(v_n, t_n)$ such that $t_1, t_2, \ldots, t_n$ satisfies the condition in Definition 2.2. Since ρ_2^k is closed in ρ and constrained by k', if $k > k'$ then there is a positive variable constraint related to a location in ρ_1' which is not satisfied, which results in a contradiction and hence, the claim holds. □

This theorem tells us that in some cases we just need to focus a shorter path since extending the path by repeating a path segment in it will result in that the given reachability specification is not satisfied.

5 Tool Prototype and Case Studies

Based on the techniques presented in this paper, we have implemented a tool prototype for the bounded reachability analysis of linear hybrid automata. The tool is

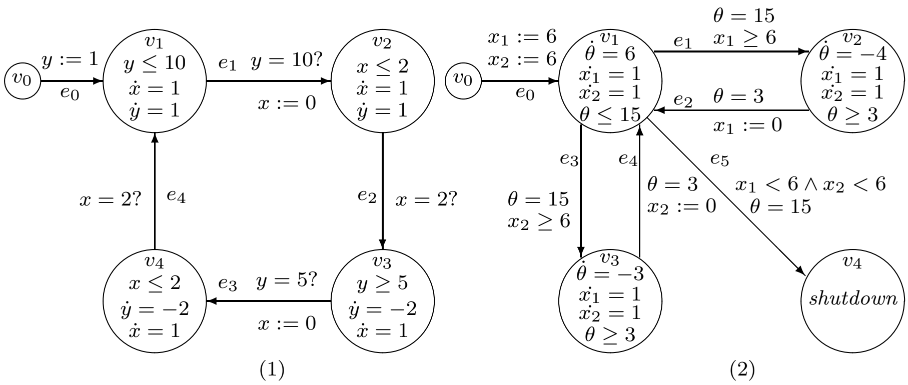

Fig. 1. The automata modelling water-level monitor and temperature control system

implemented in Java, and its graphical interface allows the users to construct, edit, and analyze linear hybrid automata interactively. The linear programming software package which is integrated in the tool is from OR-Objects of DRA Systems [11] which is a free collection of Java classes for developing operations research, scientific and engineering applications. On a HP workstation (Intel Xeon CPU 2.8GHz×2/3.78GB), we evaluated the potential of the techniques presented in this paper by several case studies.

One example depicted in Figure 1(1) is the water-level monitor in [2]. Along the path $v_0\hat{}(v_1\hat{}v_2\hat{}v_3\hat{}v_4)^k$, we check if the location v_4 is reachable, and get the positive answers from the tool with

$$k = 100, 200, 230, 400, 450, 500, 10000$$

respectively. Table 1 shows the tool performance when using the original technique (without any optimization), the optimization technique of decomposing linear programs, and the optimization techniques of shortening paths respectively. When $k \ge 500$, without one of the optimization techniques the tool cannot give a result because of the "Java.lang.out of memory" error occurring in the linear programming package integrated in the tool. Actually, with the optimization technique of shortening paths (see Theorem 4.1), for this example the tool can give a result for any k.

Another example depicted in Figure 1(2) is the temperature control system in [2]. Along the paths

$$v_0\hat{}(v_1\hat{}v_2\hat{}v_1\hat{}v_3)^k\hat{}v_1\hat{}v_4 \quad \text{and} \quad v_0\hat{}(v_1\hat{}v_2)^{k_1}\hat{}(v_1\hat{}v_3)^{k_2}\hat{}v_1\hat{}v_4\,,$$

we check if a complete shutdown is required (the location 4 is reachable), and get the negative answers with the various values of k, k_1, and k_2. The tool performance is shown in Table 2. For the path $v_0\hat{}(v_1\hat{}v_2\hat{}v_1\hat{}v_3)^k\hat{}v_1\hat{}v_4$, no optimization technique works, and the tool cannot give a result when $k \ge 450$ because of the "Java.lang.out of memory" error occurring in the linear programming package integrated in the tool. For the path $v_0\hat{}(v_1\hat{}v_2)^{k_1}\hat{}(v_1\hat{}v_3)^{k_2}\hat{}v_1\hat{}v_4$, the condition of Theorem 4.2 holds so that the optimization technique of shortening paths works.

Path: $v_0\hat{}(v_1\hat{}v_2\hat{}v_3\hat{}v_4)^k$						
k	Original technique				Decomposing LPs	Shortening paths
	constraints	variables	memory	time	time	time
100	3191	997	512M	61.172s	1.031s	0.031s
200	6391	1997	512M	466.140s	1.562s	0.031s
230	7351	2297	512M	702.766s	1.750s	0.031s
400	12791	3997	1470M	3969.421s	3.187s	0.031s
450	14391	4497	1470M	4485.328s	3.469s	0.031s
500	Java.lang.out of memory error				4.109s	0.031s
10000	Java.lang.out of memory error				38.047s	0.031s

Table 1
Experimental results on the water-level monitor

Path: $v_0\hat{}(v_1\hat{}v_2\hat{}v_1\hat{}v_3)^k\hat{}v_1\hat{}v_4$						
k	Original technique				Decomposing LPs	Shortening paths
	constraints	variables	memory	time	time	time
100	4415	1004	512M	90.218s	90.218s	90.218s
200	8815	2004	512M	686.938s	686.938s	686.938s
230	10315	2304	512M	1180.297s	1180.297s	1180.297s
400	17591	3998	1470M	5574.312s	5574.312s	5574.312s
450	Java.lang.out of memory error				Java.lang.out of memory error	

Path: $v_0\hat{}(v_1\hat{}v_2)^{k_1}\hat{}(v_1\hat{}v_3)^{k_2}\hat{}v_1\hat{}v_4$							
k_1	k_2	Original				Decomposing LPs	Shortening paths
		constraints	variables	memory	time	time	time
50	50	2215	504	512M	10.532s	10.532s	0.016s
100	100	4415	1004	512M	76.703s	76.703s	0.016s
200	200	8791	1998	1004M	496.609s	496.609s	0.016s

Table 2
Experimental results on the temperature control system

Path: $v_0\hat{}(v_1\hat{}v_2\hat{}v_1\hat{}v_3)^k\hat{}v_5\hat{}v_6$		
k	PHAVer	Our tool (original technique)
	time	time
20	$\approx$ 2400s	12.359s
30	$\approx$ 4h	36.688s
40	no result after 20 hours	82.891s
80		616.671s
100		1143.344s
150		4067.391s

Table 3
Experimental results on the experimental automaton

We also compare our technique with PHAVer [9] which is the improvement of the state-of-the-art tool HYTECH [8]. Because of performing expensive polyhedra computation, the capacity of PHAVer is restricted by the variable number in the automata. We simply construct an experimental automaton depicted in Figure 2 in which there are seven locations and variables. Along the path $v_0\hat{}(v_1\hat{}v_2\hat{}v_3\hat{}v_4)^k\hat{}v_5\hat{}v_6$, we check if the location v_6 is reachable by PHAVer and our tool respectively. Because PHAVer does not provide any timer, we manually record its execution time. The experimental result is shown in Table 3. When k is set to 20 and 30, PHAVer spends about 0.66 and 4 hours respectively for checking, which are much longer than the execution time of our tool with the original technique.

PHAVer can not give any result when $k = 40$ after running for 20 hours, but even when $k = 150$ our tool can give the result in a tolerable duration. Notice that for fairness, we use the unfolded path as the input of PHAVer for avoiding it doing the full reachability analysis. Because of performing expensive polyhedra computation, the algorithm complexity of PHAVer is exponential in the number of variables of an automaton, which gives an intuitional explanation for the experiment result.

The above experiments are preliminary and use freely available linear programming packages, but they indicate a clear potential of the techniques presented in this paper with the support of an advanced commercial linear programming package.

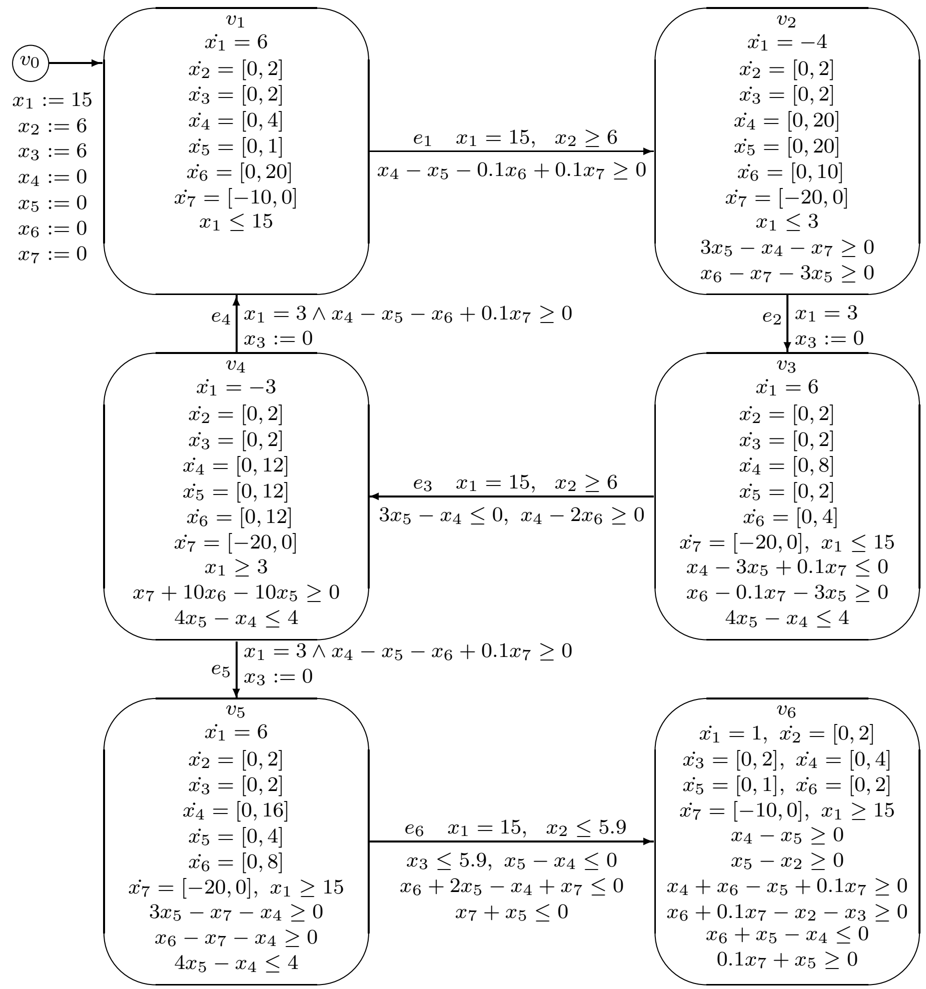

Fig. 2. An experimental automaton

6 Conclusion

In this paper, based on linear programming we develop the techniques towards an efficient path-oriented tool for the bounded reachability analysis of linear hybrid automata, which checks one path at a time where the length of the path being checked can be made very large and the size of the automaton can be made large enough to handle problems of practical interest. Since the existing techniques have not resulted in an efficient tool for checking all the paths in a linear hybrid automaton for problems with sizes of practical interest, the tool derived from the techniques presented in this paper will become a design engineer's assistant for the reachability analysis of linear hybrid automata.

Acknowledgement

We would like to thank Professor Edmund M. Clarke and the anonymous reviewers for their valuable comments and suggestions. The first and third authors are supported by the National Natural Science Foundation of China (No.60425204, No.60233020), the National Grand Fundamental Research 973 Program of China (No.2002CB312001), and by the Jiangsu Province Research Foundation (No.BK2004080). The second author acknowledges the support of a graduate fellowship from the Computer Science Department at Carnegie Mellon.

References

[1] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pp. 278-292.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H.Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. The algorithmic analysis of hybrid systems. In *Theoretical Computer Science*, 138(1995), pp.3-34.

[3] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's Decidable About Hybrid Automata? In *Journal of Computer and System Sciences*, 57:94-124, 1998.

[4] Thomas A. Henzinger, Pei-Hsin Ho, Howard Wong-Toi. Algorithmic Analysis of Nonlinear Hybrid Systems.In *IEEE Transactions on Automatic Control*, 1998, pp.540-554.

[5] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, Yunshan Zhu. Bounded Model Checking. In *Advance in Computers*, Vol.58, Academic Press, 2003.

[6] Martin Franzle, Christian Herde. Eifficient Proof Engines for Bounded Model Checking of Hybrid Systems. In *Electronic Notes in Theoretical Computer Science*, Vol.89, No.4, 2004.

[7] Gilles Audemard, Marco Bozzano, Alessandro Cimatti, Roberto Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In *Electronic Notes in Theoretical Computer Science*, Vol.89, No.4, 2004.

[8] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: a model checker for hybrid systems. In *Software Tools for Technology Transfer*, 1:110-122, 1997.

[9] Goran Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control (HSCC'05)*, Lecture Notes in Computer Science 2289, Springer-Verlag, 2005, pp.258-273.

[10] Li Xuandong, Zhao Jianhua, Pei Yu, Li Yong, Zheng Tao, and Zheng Guoliang. Positive Loop-Closed Automata: A Decidable Class of Hybrid Systems. In *Journal of Logic and Algebraic Programming*, Vol.52-53(C), Elsevier Science, 2002, pp.79-108.

[11] OR-Objects of DRA Systems. http://OpsResearch.com/OR-Objects/index.html.