Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2011-IJ-001

Path-oriented bounded reachability analysis of composed linear hybrid systems

Lei Bu, Xuandong Li

Postprint Version. Originally Published in: International Journal on Software Tools for Technology Transfer, Volume 13, Number 4, pp.307-317, Springer

Int J Softw Tools Technol Transfer (2011) 13:307–317
DOI 10.1007/s10009-010-0163-9

REGULAR PAPER

Path-oriented bounded reachability analysis of composed linear hybrid systems

Lei Bu · Xuandong Li

Published online: 18 June 2010

Abstract The existing techniques for reachability analysis of linear hybrid systems do not scale well to the problem size of practical interest. The performance of existing techniques is even worse for reachability analysis of a composition of several linear hybrid automata. In this paper, we present an efficient path-oriented approach to bounded reachability analysis of composed systems modeled by linear hybrid automata with synchronization events. It is suitable for analyzing systems with many components by selecting critical paths, while this task was quite insurmountable before because of the state explosion problem. This group of paths will be transformed to a group of linear constraints, which can be solved by a linear programming solver efficiently. This approach of symbolic execution of paths allows design engineers to check important paths, and accordingly increase the faith in the correctness of the system. This approach is implemented into a prototype tool Bounded reAchability CHecker (BACH). The experimental data show that both the path length and the number of participant automata in a system checked using BACH can scale up greatly to satisfy practical requirements.

Keywords Hybrid systems · Bounded reachability analysis · Linear hybrid automata · Linear programming

L. Bu · X. Li (✉)
State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, Jiangsu 210093, People's Republic of China
e-mail: lxd@nju.edu.cn

L. Bu · X. Li
Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu 210093, People's Republic of China
e-mail: bl@seg.nju.edu.cn

1 Introduction

The model checking problem for hybrid systems is very difficult. Even for a relatively simple class of hybrid systems, linear hybrid automata (denoted as LHA), the reachability analysis problem is undecidable [1–4]. Several model checking tools have been developed for reachability analysis of LHA, but they do not scale well to the size of practical problems. The state-of-the-art tool HYTECH [5] and its improvement PHAVer [6] need expensive polyhedra computation, which greatly restricts the solvable problem size.

In recent years, *Bounded Model Checking* (denoted as BMC) [7] has been presented as a technique alternative to BDD-based symbolic model checking, whose basic idea is to encode the next-state relation of a system as a propositional formula, unroll this formula to some integer k, and search for a counterexample in the model executions whose length is bounded by k. The BMC problems can be solved by Boolean Satisfiability (denoted as SAT) methods, which have achieved tremendous progress in recent years, as summarized in [8].

As extensions to BMC, there are several related works [9–11] to check linear hybrid systems. In these techniques, the model checking problems are reduced to the satisfiability problem of a boolean combination of propositional variables and linear mathematical constraints. Based on these techniques, several tools were developed, such as MathSAT [11] and HySAT [9], and all of them are based on a SAT-solver that calls on demand solver for conjunctions of the domain-specific constraints [12]. But the experiment results show that it is difficult to apply those tools to analysis problems of practical size. The performance of existing techniques is even worse for reachability analysis of a composition of several linear hybrid automata.

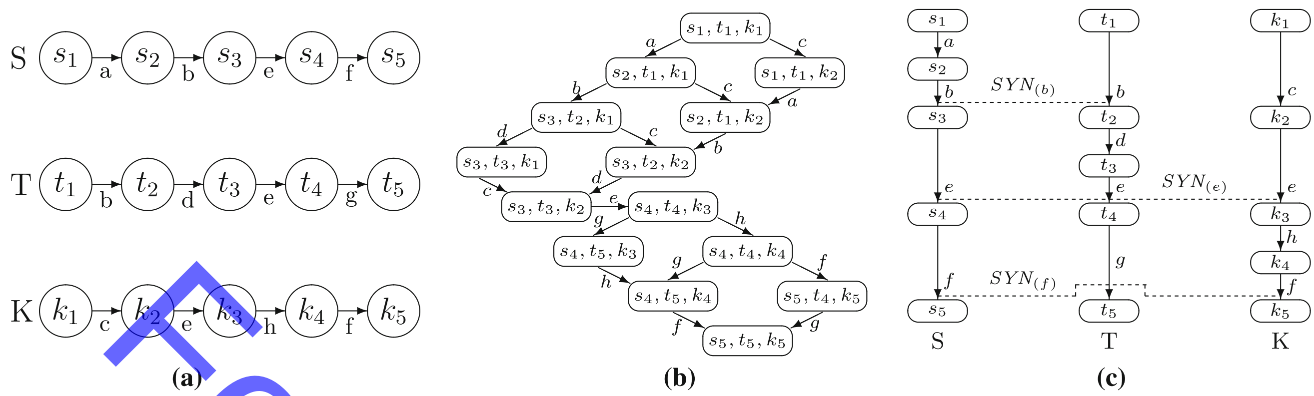

Fig. 1 Sample automata and their compositional state space representations

As the existing techniques do not perform well concerning analysis problems of practical size, in this paper, we propose a complementary approach to develop an efficient path-oriented technique for bounded reachability analysis of LHA compositions. This technique checks a group of paths at a time, one path for each LHA, where both the path length and the number of participant automata checked can scale up greatly to satisfy practical requirements. This approach of symbolic execution of paths can be used by design engineers to check critical paths, and thereby increase the faith in the system correctness.

For a linear hybrid system consisting of several components (LHA), with our approach users can assign a specific path to each LHA, respectively, and all of the paths are transformed into a group of linear constraints automatically. Then, a few of constraints about system integration according to the synchronization events in each path will be added to ensure that the components cooperate correctly. It follows that the reachability problem along those specific paths can be reduced to a linear program. We shall use a simple example to illustrate this idea below.

Most traditional verification methods of hybrid systems consisting of several components require the composition of the set of automata to a unique global automaton, which leads to the critical problem of state explosion. For example, Fig. 1a gives a simple system consisting of three subsystems: S, T, and K which synchronize with each other by events b, e, and f. Even if these three subsystems are all very simple, the state space of the resulting automata is still quite large as we can see from Fig. 1b. While using our path-oriented approach, as each of those three subsystems only has one path, we simply select all of them for analysis, as shown in Fig. 1c. First, for each of these three paths we generate a group of linear constraints that represents all the timed runs corresponding to the path. For example, for the path $\langle t_1\rangle \rightarrow \langle t_2\rangle \rightarrow \langle t_3\rangle \rightarrow \langle t_4\rangle \rightarrow \langle t_5\rangle$ of the system T, we use $\begin{pmatrix} t_i \\ \delta_i \end{pmatrix}$ to indicate that the system has stayed in location t_i for time delay δ_i (nonnegative variable). Any timed run corresponding to this path can be represented by

$$\begin{pmatrix} t_1 \\ \delta_1 \end{pmatrix} \rightarrow \begin{pmatrix} t_2 \\ \delta_2 \end{pmatrix} \rightarrow \begin{pmatrix} t_3 \\ \delta_3 \end{pmatrix} \rightarrow \begin{pmatrix} t_4 \\ \delta_4 \end{pmatrix} \rightarrow \begin{pmatrix} t_5 \\ \delta_5 \end{pmatrix}$$

where $\delta_1, \delta_2, \delta_3, \delta_4, \delta_5$ must satisfy all the time constraints enforced by the system, which forms a group of linear constraints. Second, several constraints will be added to ensure that these three components cooperate accurately according to the synchronization events, which are illustrated by the dashed lines and $SYN_{(event)}$ in Fig. 1c. Because such a state space representation by linear constraints is equivalent to the Cartesian product representation shown in Fig. 1b in terms of reachability analysis, the reachability analysis problem along these three paths can be transformed into a linear programming problem, which can be solved efficiently.

Therefore, the path-oriented approach presented in this paper is to check if an LHA composition satisfies a reachability specification along a given group of its component paths. This approach has been implemented into a prototype tool Bounded reAchability CHecker (BACH). The experimental data show that both the path length and the number of participant components in a system checked using BACH can scale up greatly to satisfy practical requirements.

The rest of the paper is organized as follows. In the next section, we define the class of linear hybrid automata and the compositions of linear hybrid automata considered in this paper. Section 3 presents the linear programming based solution for the path-oriented reachability analysis of LHA compositions. Section 4 describes several case studies to show the ability of BACH, and also gives a comparison with other tools. Finally the conclusion is made in Sect. 5.

2 Linear hybrid automata and their composition

2.1 Linear hybrid automata

The linear hybrid automata considered in this paper are a variation of the definition given in [1]. The flow conditions of variables in a linear hybrid automaton considered here may be given as a range of possible values for their derivatives.

Definition 1 A linear hybrid automaton (LHA) H is a tuple $H = (X, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$, where

- X is a finite set of real-valued variables; Σ is a finite set of event labels; V is a finite set of *locations*; $V^0 \subseteq V$ is a set of *initial locations*.
- E is a *transition relation* whose elements are of the form $(v, \sigma, \phi, \psi, v')$, where v, v' are in V, $\sigma \in \Sigma$ is a label, ϕ is a set of *transition guards* of the form $a \le \sum_{i=0}^{l} c_i x_i \le b$, and ψ is a set of *reset actions* of the form $x := c$ where $x_i \in X$, $x \in X$, a, b, c and c_i are real numbers (a, b may be ∞).
- α is a labeling function which maps each location in V to a *location invariant* which is a set of *variable constraints* of the form $a \le \sum_{i=0}^{l} c_i x_i \le b$ where $x_i \in X$, a, b and c_i are real numbers (a, b may be ∞).
- β is a labeling function which maps each location in V to a set of *flow conditions* which are of the form $\dot{x} = [a, b]$, where $x \in X$, and a, b are real numbers ($a \le b$). For any $v \in V$, for any $x \in X$, there is one and only one flow condition $\dot{x} = [a, b] \in \beta(v)$.
- γ is a labeling function which maps each location in V^0 to a set of *initial conditions* which are of the form $x = a$ where $x \in X$ and a is a real number. For any $v \in V^0$, for any $x \in X$, there is at most one initial condition definition $x = a \in \gamma(v)$.

We use the sequences of locations to represent the evolution of an LHA from location to location. For an LHA $H = (X, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$, a *path segment* is a sequence of locations of the form $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$, which satisfies $(v_i, \sigma_i, \phi_i, \psi_i, v_{i+1}) \in E$ for each i $(0 \le i < n)$. A *path* in H is a path segment starting at an initial location in V^0. For a path in H of the form $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$, by assigning each location v_i with a time delay stamp δ_i we get a *timed sequence* of the form $\begin{pmatrix} v_0 \\ \delta_0 \end{pmatrix} \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \begin{pmatrix} v_1 \\ \delta_1 \end{pmatrix} \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \begin{pmatrix} v_n \\ \delta_n \end{pmatrix}$ where δ_i $(0 \le i \le n)$ is a nonnegative real number, which represents a behavior of H such that the system starts at v_0, stays there for δ_0 time units, then jumps to v_1 and stays at v_1 for δ_1 time units, and so on.

The behavior of an LHA can be described informally as follows. The automaton starts at one of the initial locations with some variables initialized to their initial values. As time progresses, the values of all variables change continuously according to the flow condition associated with the current location. At any time, the system can change its current location from v to v' provided that there is a transition $(v, \sigma, \phi, \psi, v')$ from v to v' whose all transition guards in ϕ are satisfied by the current value of the variables. With a location change by a transition $(v, \sigma, \phi, \psi, v')$, some variables are reset to the new value accordingly to the reset actions in ψ. Transitions are assumed to be instantaneous.

Let $H = (X, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$ be an LHA. Given a timed sequence ω of the form $\begin{pmatrix} v_0 \\ \delta_0 \end{pmatrix} \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \begin{pmatrix} v_1 \\ \delta_1 \end{pmatrix} \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \begin{pmatrix} v_n \\ \delta_n \end{pmatrix}$, let $\zeta_i(x)$ represent the value of x $(x \in X)$ when the automaton has stayed at v_i for delay δ_i along with ω $(0 \le i \le n)$, and $\lambda_i(x)$ represent the value of x $(x \in X)$ at the time the automaton reaches v_i along with $\omega (0 \le i \le n)$. It follows that $\lambda_0(x) = a$ if $x = a \in \gamma(v_0)$, and $\lambda_{i+1}(x) = \begin{cases} d & \text{if } x := d \in \psi_i \\ \zeta_i(x) & \text{otherwise} \end{cases}$ $(0 \le i < n)$.

Definition 2 For an LHA $H = (X, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$, a timed sequence of the form $\begin{pmatrix} v_0 \\ \delta_0 \end{pmatrix} \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \begin{pmatrix} v_1 \\ \delta_1 \end{pmatrix} \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \begin{pmatrix} v_n \\ \delta_n \end{pmatrix}$ represents a behavior of H if and only if the following condition is satisfied:

- $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$ is a path;
- $\delta_1, \delta_2, \ldots, \delta_n$ ensure that each variable $x \in X$ evolves according to its flow condition in each location v_i $(0 \le i \le n)$, i.e., $u_i \delta_i \le \zeta_i(x) - \lambda_i(x) \le u_i' \delta_i$ where $\dot{x} = [u_i, u_i'] \in \beta(v_i)$;
- all the transition guards in $\phi_i (1 \le i \le n-1)$ are satisfied, i.e., for each transition guard

  $$a \le c_0 x_0 + c_1 x_1 + \cdots + c_l x_l \le b \quad in \quad \phi_i,$$

  $$a \le c_0 \zeta_i(x_0) + c_1 \zeta_i(x_1) + \cdots + c_l \zeta_i(x_l) \le b;$$

- the location invariant of each location v_i $(1 \le i \le n)$ is satisfied, i.e.,
  - at the time the automaton leaves v_i, each variable constraint $a \le c_0 x_0 + c_1 x_1 + \cdots + c_l x_l \le b$ in $\alpha(v_i)$ $(0 \le i \le n)$ is satisfied, i.e., $a \le c_0 \zeta_i(x_0) + c_1 \zeta_i(x_1) + \cdots + c_l \zeta_i(x_l) \le b$, and
  - at the time the automaton reaches v_i, each variable constraint $a \le c_0 x_0 + c_1 x_1 + \cdots + c_l x_l \le b$ in

$\alpha(v_i)$ $(0 \le i \le n)$ is satisfied, i.e.,

$$a \le c_0\lambda_i(x_0) + c_1\lambda_i(x_1) + \cdots + c_l\lambda_i(x_l) \le b.$$

For $\rho = \langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$ which is a path of an LHA H, let $\mathcal{L}(\rho)$ represent the set of the behaviors of H of the form

$$\left\langle \begin{matrix} v_0 \\ \delta_0 \end{matrix} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{matrix} v_1 \\ \delta_1 \end{matrix} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{matrix} v_n \\ \delta_n \end{matrix} \right\rangle.$$

2.2 Composition of linear hybrid automata

For a group of linear hybrid automata, their composition is defined as a product linear hybrid automaton generated by synchronizing all the components with respect to the same event labels.

Definition 3 Let $H_1 = (X_1, \Sigma_1, V_1, V_1^0, E_1, \alpha_1, \beta_1, \gamma_1)$ and $H_2 = (X_2, \Sigma_2, V_2, V_2^0, E_2, \alpha_2, \beta_2, \gamma_2)$ be two LHA, where $X_1 \cap X_2 = \emptyset$. The *composition* of H_1 and H_2, denoted as $H_1||H_2$, is an LHA $N = (X, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$ where

- $X = X_1 \cup X_2$; $\Sigma = \Sigma_1 \cup \Sigma_2$; $V = V_1 \times V_2$; $V^0 = V_1^0 \times V_2^0$; $\alpha((v_1, v_2)) = \alpha(v_1) \cup \alpha(v_2)$; $\beta((v_1, v_2)) = \beta(v_1) \cup \beta(v_2)$; $\gamma((v_1, v_2)) = \gamma(v_1) \cup \gamma(v_2)$;
- E is defined as follows:
  - for $a \in \Sigma_1 \cap \Sigma_2$, for every $(v_1, a, \phi_1, \psi_1, v_1')$ in E_1 and $(v_2, a, \phi_2, \psi_2, v_2')$ in E_2, E contains $((v_1, v_2), a, \phi_1 \cup \phi_2, \psi_1 \cup \psi_2, (v_1', v_2'))$;
  - for $a \in \Sigma_1 \setminus \Sigma_2$, for every (v, a, ϕ, ψ, v') in E_1 and every t in V_2, E contains $((v, t), a, \phi, \psi, (v', t))$;
  - for $a \in \Sigma_2 \setminus \Sigma_1$, for every (v, a, ϕ, ψ, v') in E_2 and every t in V_1, E contains $((t, v), a, \phi, \psi, (t, v'))$.

For all $m > 2$, the *composition* of LHA $H_1, H_2, \cdots, H_m$, denoted as $H_1||H_2||\ldots||H_m$, is an LHA which is defined recursively as $H_1||H_2||\ldots||H_m = H_1||H'$ where $H' = H_2||H_3||\ldots||H_m$.

We call a composition of linear hybrid automata CLHA for short. Let $N = H_1||H_2||\ldots||H_m$ be a CLHA where $H_i = (X_i, \Sigma_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ $(1 \le i \le m)$ and ρ be a path in N of the form

$$\rho = \langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle.$$

It follows that $v_i = (v_{i1}, v_{i2}, \cdots, v_{im})$ $(0 \le i \le n)$ where $v_{ik} \in V_k$ $(1 \le k \le m)$. For any k $(1 \le k \le m)$, we construct the sequence ρ_k from ρ as follows: replace any v_i with $v_{ik} (0 \le i \le n)$, and for any $\xrightarrow[\sigma_{i-1}]{(\phi_{i-1},\psi_{i-1})} \langle v_{ik} \rangle$ $(1 \le i \le n)$, if $(v_{i-1k}, \sigma_{i-1}, \phi, \psi, v_{ik}) \in E_k$, then replace it with $\xrightarrow[\sigma_{i-1}]{(\phi,\psi)} \langle v_{ik} \rangle$, otherwise remove it. It follows that ρ_k is a path in H_k. We say that ρ_k is the *projection* of ρ on H_k. Intuitively, ρ_k is the execution trace of N on H_k when N runs along ρ.

3 Path-oriented bounded reachability analysis using linear programming

In this section, we present a solution for path-oriented bounded reachability analysis of compositions of linear hybrid automata based on linear programming.

For an LHA $H = (X, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$, a *reachability specification*, denoted as $\mathcal{R}(v, \varphi)$, consists of a location v in H and a set φ of variable constraints of the form $a \le c_0x_0 + c_1x_1 + \cdots + c_lx_l \le b$ where $x_i \in X$ for any i $(0 \le i \le l)$, a, b and c_i $(0 \le i \le l)$ are real numbers.

Definition 4 Let $H = (X, \Sigma, V, V^0, E, \alpha, \beta, \gamma)$ be an LHA, and $\mathcal{R}(v, \varphi)$ be a reachability specification. A behavior of H of the form

$$\left\langle \begin{matrix} v_0 \\ \delta_0 \end{matrix} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{matrix} v_1 \\ \delta_1 \end{matrix} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{matrix} v_n \\ \delta_n \end{matrix} \right\rangle$$

satisfies $\mathcal{R}(v, \varphi)$ if and only if $v_n = v$ and each variable constraint in φ is satisfied when the automaton has stayed in v_n for delay δ_n, i.e., for each variable constraint $a \le c_0x_0 + c_1x_1 + \cdots + c_lx_l \le b$ in φ,

$$a \le c_0\zeta_n(x_0) + c_1\zeta_n(x_1) + \cdots + c_m\zeta_n(x_l) \le b$$

where $\zeta_n(x_k)$ $(0 \le k \le l)$ represents the value of x_k when the automaton has stayed at v_n for the delay δ_n.H *satisfies* $\mathcal{R}(v, \varphi)$ if and only if there is a behavior of H which satisfies $\mathcal{R}(v, \varphi)$.

In this paper, the problem we concern is to check if a CLHA $N = H_1||H_2||\ldots||H_m$ satisfies a reachability specification by having a behavior along a set of finite paths in $H_1, H_2, \ldots, H_m$, which is defined formally as follows:

Definition 5 Let $N = H_1||H_2||\ldots||H_m$ be a CLHA, $P = \{\rho_1, \rho_2, \ldots, \rho_m\}$ be a path set, where ρ_i is a finite path in H_i $(1 \le i \le m)$, and $\mathcal{R}(v, \varphi)$ be a reachability specification. P *satisfies* $\mathcal{R}(v, \varphi)$ if and only if there is a path ρ of N satisfies the following condition:

- the projection of ρ on H_i is ρ_i $(1 \le i \le m)$, and
- there is a behavior of N in $\mathcal{L}(\rho)$ which satisfies $\mathcal{R}(v, \varphi)$.

Let $N = H_1||H_2||\ldots||H_m$ be a CLHA, and $P = \{\rho_1, \rho_2, \ldots, \rho_m\}$ be a path set where ρ_i is a finite path in H_i $(1 \le i \le m)$. In general, according to Definitions 4 and 5, the problem of checking P for a reachability specification $\mathcal{R}(v, \varphi)$ could be solved by traversing the related behavior of N and checking if $\mathcal{R}(v, \varphi)$ is satisfied. But this approach suffers from the infinite state space and state space explosion problems. In the following, we present a linear programming based solution to the problem, which is based on the concept of *trails*. Intuitively, a *trail* of N consists of the behavior of $H_1, H_2, \ldots, H_m$ which are synchronized in terms of the same event labels.

Definition 6 Let $N = H_1||H_2||\ldots||H_m$ be a CLHA, where $H_i = (X_i, \Sigma_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ $(1 \le i \le m)$. A *trail* τ of N is of the form $(\omega_1, \omega_2, \ldots, \omega_m)$ where each ω_i $(1 \le i \le m)$ is a behavior of H_i of the form $\begin{pmatrix} v_{i0} \\ \delta_{i0} \end{pmatrix} \xrightarrow[\sigma_{i0}]{(\phi_{i0}, \psi_{i0})} \begin{pmatrix} v_{i1} \\ \delta_{i1} \end{pmatrix} \xrightarrow[\sigma_{i1}]{(\phi_{i1}, \psi_{i1})} \ldots \xrightarrow[\sigma_{in_i-1}]{(\phi_{in_i-1}, \psi_{in_i-1})} \begin{pmatrix} v_{in_i} \\ \delta_{in_i} \end{pmatrix}$, and satisfies the *synchronization constraint*, i.e. for any k, j $(1 \le k, j \le m)$,

- $\delta_{k0} + \delta_{k1} + \cdots + \delta_{kn_k} = \delta_{j0} + \delta_{j1} + \cdots + \delta_{jn_j}$,
- for any σ_{kp} $(0 \le p \le n_k)$ which is the dth occurrence in ω_k of the elements in $\Sigma_k \cap \Sigma_j$, there is σ_{jq} $(0 \le q \le n_j)$, which is the dth occurrence in ω_j of the elements in $\Sigma_k \cap \Sigma_j$, such that $\sigma_{jq} = \sigma_{kp}$ and $\delta_{k0} + \delta_{k1} + \cdots + \delta_{kp} = \delta_{j0} + \delta_{j1} + \cdots + \delta_{jq}$.

For a CLHA, in essence its trails form another representation of its behavior.

Definition 7 Let $N = H_1||H_2||\ldots||H_m$ be a CLHA where $H_i = (X_i, \Sigma_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ $(1 \le i \le m)$, $\tau = (\omega_1, \omega_2, \ldots, \omega_m)$ be a trail of N where ω_i $(1 \le i \le m)$ is of the form $\begin{pmatrix} v_{i0} \\ \delta_{i0} \end{pmatrix} \xrightarrow[\sigma_{i0}]{(\phi_{i0}, \psi_{i0})} \begin{pmatrix} v_{i1} \\ \delta_{i1} \end{pmatrix} \xrightarrow[\sigma_{i1}]{(\phi_{i1}, \psi_{i1})} \ldots \xrightarrow[\sigma_{in_i-1}]{(\phi_{in_i-1}, \psi_{in_i-1})} \begin{pmatrix} v_{in_i} \\ \delta_{in_i} \end{pmatrix}$, and $\mathcal{R}(v, \varphi)$ be a reachability specification. τ *satisfies* $\mathcal{R}(v, \varphi)$ if and only if $v = (v_{1n_1}, v_{2n_2}, \ldots, v_{mn_m})$ and each variable constraint in φ is satisfied when H_i has stayed in v_{in_i} for the delay δ_{in_i} $(1 \le i \le m)$, i.e., for each variable constraint $a \le c_0x_0 + c_1x_1 + \cdots + c_lx_l \le b$ in φ,

$$a \le c_0\zeta_n(x_0) + c_1\zeta_n(x_1) + \cdots + c_m\zeta_n(x_l) \le b$$

where for any k $(0 \le k \le l)$, if $x_k \in X_i$ $(1 \le i \le m)$ then $\zeta_n(x_k)$ $(0 \le k \le l)$ represents the value of x_k when H_i has stayed at v_{in_i} for the delay δ_{in_i}.

Theorem 1 *Let $N = H_1||H_2||\ldots||H_m$ be a CLHA, and $\mathcal{R}(v, \varphi)$ be a reachability specification. N satisfies $\mathcal{R}(v, \varphi)$ if and only if there is a trail of N which satisfies $\mathcal{R}(v, \varphi)$.*

The proof of this theorem is presented in "Appendix". For a CLHA $N = H_1||H_2||\ldots||H_m$, for a path set $P = \{\rho_1, \rho_2, \ldots, \rho_m\}$ where ρ_i is a finite path in H_i $(1 \le i \le m)$, let $\mathcal{L}(P)$ represent the set of the trails of N which are of the form $(\omega_1, \omega_2, \ldots, \omega_m)$ where $\omega_i \in \mathcal{L}(\rho_i)(1 \le i \le m)$. From Definition 5 and Theorem 1, we have the following corollary.

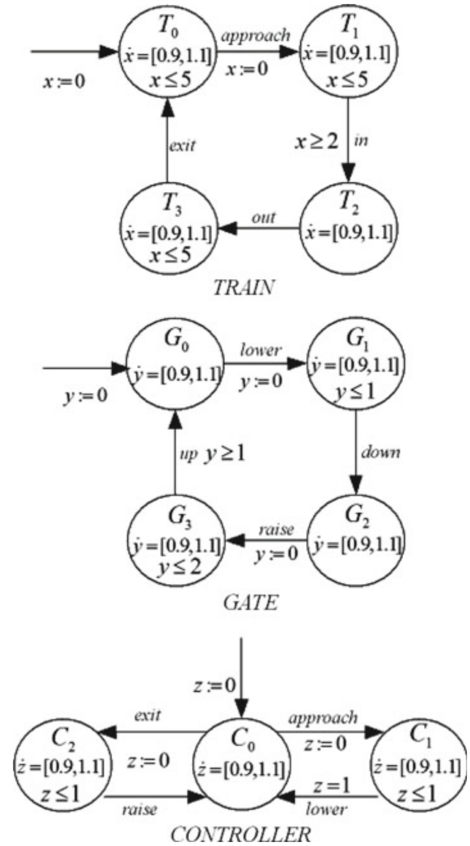

Fig. 2 Train Gate Controller

Corollary 1 *Let $N = H_1||H_2||\ldots||H_m$ be a CLHA, $P = \{\rho_1, \rho_2, \ldots, \rho_m\}$ be a path set where ρ_i is a finite path in H_i $(1 \le i \le m)$, and $\mathcal{R}(v, \varphi)$ be a reachability specification. P satisfies $\mathcal{R}(v, \varphi)$ if and only if there is $\tau \in \mathcal{L}(P)$ which satisfies $\mathcal{R}(v, \varphi)$.*

Let $\mathcal{R}(v, \varphi)$ be a reachability specification. For a CLHA $N = H_1||H_2||\ldots||H_m$, for a path set $P = \{\rho_1, \rho_2, \ldots, \rho_m\}$ where ρ_i is a finite path in H_i $(1 \le i \le m)$, based on Corollary 1 we can reduce the satisfaction problem of P for $\mathcal{R}(v, \varphi)$ to a linear program as follows. Suppose that any $\tau \in \mathcal{L}(P)$ is of the form $(\omega_1, \omega_2, \ldots, \omega_m)$ where each $\omega_i \in \mathcal{L}(\rho_i)(1 \le i \le m)$ is of the form $\begin{pmatrix} v_{i0} \\ \delta_{i0} \end{pmatrix} \xrightarrow[\sigma_{i0}]{(\phi_{i0}, \psi_{i0})} \begin{pmatrix} v_{i1} \\ \delta_{i1} \end{pmatrix} \xrightarrow[\sigma_{i1}]{(\phi_{i1}, \psi_{i1})} \ldots \xrightarrow[\sigma_{in_i-1}]{(\phi_{in_i-1}, \psi_{in_i-1})} \begin{pmatrix} v_{in_i} \\ \delta_{in_i} \end{pmatrix}$, and $H_i = (X_i, \Sigma_i, V_i, V_i^0, E_i, \alpha_i, \beta_i, \gamma_i)$ $(1 \le i \le m)$. Since that τ satisfies $\mathcal{R}(v, \varphi)$ means that the following condition holds: for any i $(1 \le i \le m)$,

- $\delta_{i0}, \delta_{i1}, \ldots, \delta_{in_i}$ ensure that each variable $x \in X_i$ evolves according to its flow condition in each location v_{ij} $(0 \le j \le n_i)$, all the transition guards in ϕ_{ij} $(0 \le j < n_i)$ are satisfied, and that all the variable constraints in $\alpha_i(v_{ij})$ $(0 \le j \le n_i)$ are satisfied (Definition 2),

Table 1 Data on the Train Gate Controller system

Path	Train	$(\langle T_0\rangle \xrightarrow[approach]{} \langle T_1\rangle \xrightarrow[in]{} \cdots \langle T_3\rangle)^k \xrightarrow[exit]{} \langle T_0\rangle$		
	Gate	$(\langle G_0\rangle \xrightarrow[lower]{} \langle G_1\rangle \xrightarrow[down]{} \cdots \langle G_3\rangle)^k \xrightarrow[up]{} \langle G_0\rangle$		
	Controller	$(\langle C_0\rangle \xrightarrow[approach]{} \langle C_1\rangle \xrightarrow[lower]{} \cdots \langle C_2\rangle)^k \xrightarrow[raise]{} \langle C_0\rangle$		
k	Constraint	Variable	Memory (MB)	Time (s)
90	7,033	2,166	256	178.456
130	10,153	3,126	512	534.058
180	14,053	4,326	1,024	1,189.788
250	19,513	6,006	2,048	3,147.031

– $\delta_{i0}, \delta_{i1}, \ldots, \delta_{in_i}$ satisfy the synchronization constraint (Definition 6), and
– $\delta_{i0}, \delta_{i1}, \ldots, \delta_{in_i}$ ensure that all the variable constraints in φ are satisfied (Definition 7),

which forms a group of linear inequalities on $\delta_{i0}, \delta_{i1}, \ldots, \delta_{in_i}$ $(1 \le i \le m)$, denoted as $\Theta(P, \mathcal{R}(v, \varphi))$, we can check if P satisfies $\mathcal{R}(v, \varphi)$ by checking if the group $\Theta(P, \mathcal{R}(v, \varphi))$ of linear inequalities has a solution, which can be solved by linear programming.

There are a number of efficient software packages available for linear programming. Utilizing these software packages we can develop an efficient tool for path-oriented bounded reachability analysis of CLHAs where the length of the path, the size of each LHA, and the component number are all closer to the practical problem scales.

4 Implementation and evaluation

The solution presented in the above section has been implemented into a prototype tool BACH (Bounded reAchability CHecker for linear hybrid automata). This section examines its performance with several case studies, and compares it with the existing tools.

4.1 Tool description

BACH is implemented in Java, and can be downloaded from http://seg.nju.edu.cn/BACH/. It provides a convenient graphical LHA editor and two reachability checkers: path-oriented reachability checker and bounded reachability checker. The path-oriented reachability checker does bounded reachability analysis of a specific path given by user, while the bounded reachability checker investigates all the paths in the bound limit one by one by using the path-oriented checker to perform bounded reachability analysis. The early version of BACH is a reachability analyzer taking as input a unique LHA [13,14]. Through the implementation of the solution

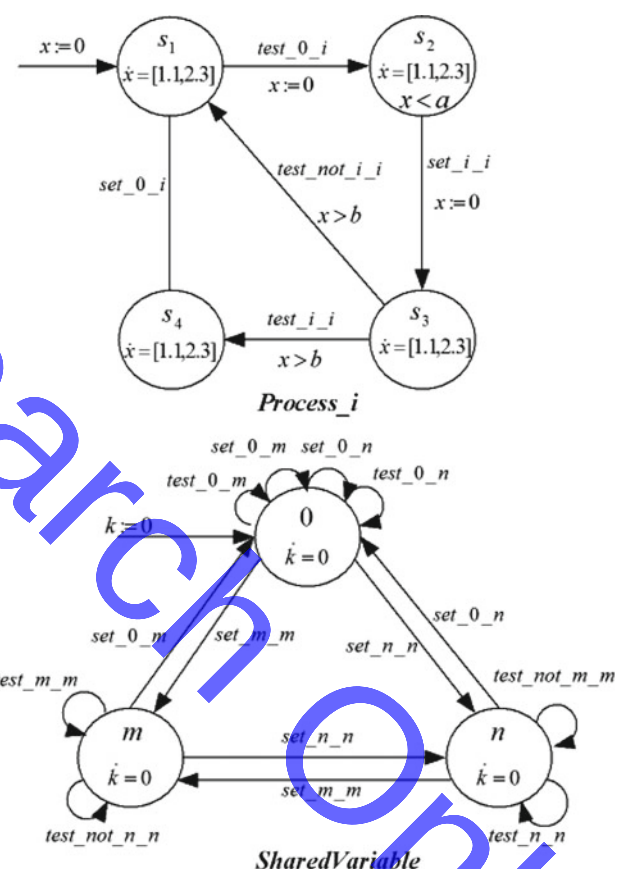

Fig. 3 Fischer Mutual Exclusion Protocol

presented in the above section, the current version of BACH can support path-oriented reachability analysis of LHA compositions, and it has also integrated a bounded reachability checker for LHA compositions [15]. The main functionality of BACH is provided by the following set of services:

– *Graphical LHA Editor* This component allows users to construct, edit, and perform syntax analysis of LHA interactively. This Editor can also transform the graphical representation of LHA to a readable text file which is used as the input file for reachability checking.

Table 2 Data on the Fischer Mutual Exclusion Protocol with 10 processes

Path	Pro_1	$(\langle s_1\rangle \xrightarrow[test\_0\_1]{} \langle s_2\rangle \xrightarrow[set\_1\_1]{} \cdots \langle s_4\rangle)^k \xrightarrow[set\_0\_1]{} \langle s_1\rangle$		
	Pro_2	$(\langle s_1\rangle \xrightarrow[test\_0\_2]{} \langle s_2\rangle \xrightarrow[set\_2\_2]{} \ldots \langle s_4\rangle)^k \xrightarrow[set\_0\_2]{} \langle s_1\rangle$		
	...	...		
	Pro_10	$(\langle s_1\rangle \xrightarrow[test\_0\_10]{} \langle s_2\rangle \xrightarrow[set\_10\_10]{} \ldots \langle s_4\rangle)^k \xrightarrow[set\_0\_10]{} \langle s_1\rangle$		
	SV	$(\langle 0\rangle \xrightarrow[test\_0\_1]{} \ldots \langle 0\rangle)^k \xrightarrow[set\_0\_1]{} \langle 0\rangle$		
k	Constraint	Variable	Memory (MB)	Time (s)
15	5,753	2,422	256	213.32
20	7,653	3,222	512	498.6
30	11,453	4,822	1,024	1,638.844
40	15,253	6,422	2,048	3,845.857

Table 3 Data on the Fischer Mutual Exclusion Protocol with 40 processes

Path	Pro_1	$(\langle s_1\rangle \xrightarrow[test\_0\_1]{} \langle s_2\rangle \xrightarrow[set\_1\_1]{} \ldots \langle s_4\rangle)^k \xrightarrow[set\_0\_1]{} \langle s_1\rangle$		
	Pro_2	$(\langle s_1\rangle \xrightarrow[test\_0\_2]{} \langle s_2\rangle \xrightarrow[set\_2\_2]{} \ldots \langle s_4\rangle)^k \xrightarrow[set\_0\_2]{} \langle s_1\rangle$		
	...	...		
	Pro_40	$(\langle s_1\rangle \xrightarrow[test\_0\_40]{} \langle s_2\rangle \xrightarrow[set\_40\_40]{} \ldots \langle s_4\rangle)^k \xrightarrow[set\_0\_40]{} \langle s_1\rangle$		
	SV	$(\langle 0\rangle \xrightarrow[test\_0\_1]{} \ldots \langle 0\rangle)^k \xrightarrow[set\_0\_1]{} \langle 0\rangle$		
k	Constraint	Variable	Memory (MB)	Time (s)
3	4,763	2,002	256	147.15
5	7,083	3,282	512	563.867
7	10,483	4,562	1,024	1,463.619
10	15,403	6,482	2,048	4,279.917

- *Path-Oriented Reachability Checker* The checker requires users to select a specific path set which includes one path for each component, respectively, and uses the method presented in this paper to check whether the reachability specification is satisfied along with the given path set.
- *Bounded Reachability Checker* This checker uses the path-oriented checker as underlying solver. It traverses all the path sets below the threshold by a tailored Depth-First Search algorithm, and checks the related path set for reachability using linear programming to perform bounded reachability checking.

4.2 Case studies

On a DELL workstation (Intel Core2 Quad CPU 2.4 GHz, 4 GB RAM, actually only 2 GB is used by the limitation of Java memory allocation), we evaluate the potential of the path-oriented reachability checker in BACH by three groups of case studies which are explained below. These three examples are all coming from previous studies of real-time and hybrid systems [3,16,17]. Although they are still academic examples, the sizes of the problems we solve here are quite big and close to the practical interest, e.g., for Fischer Mutual Exclusion Protocol, the largest system we solve consisting of 320 processes.[1]

Train Gate Controller. The first case study is the well-studied Train Gate Controller model [16]. This system is

[1] The paths we selected in Sects. 4.2 and 4.3 are all reachable. We do not include unreachable paths in the experiments because the performance of BACH's path-oriented reachability checker is only related to the size of the problem, e.g., the number of locations and constraints in the path-set.

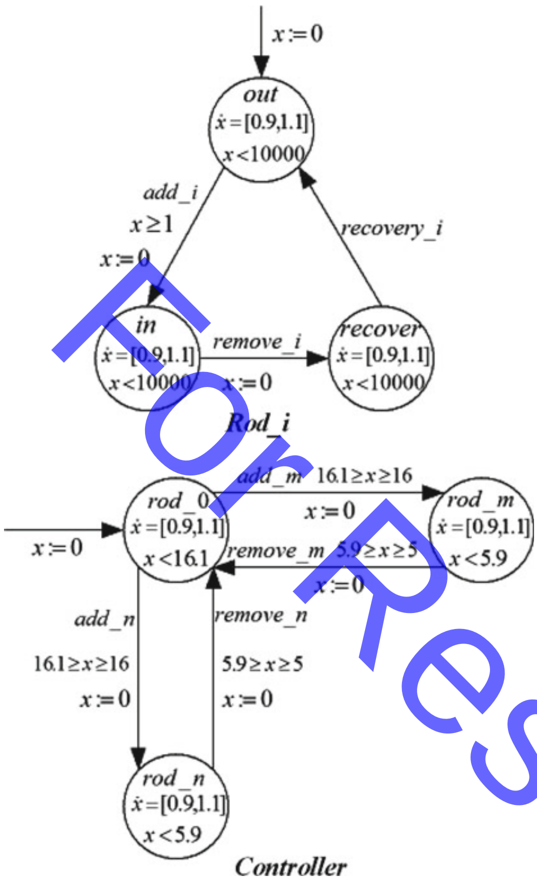

Fig. 4 Nuclear Reactor System

composed of three components: TRAIN, GATE and CONTROLLER as shown in Fig. 2. In this system, we conduct an experiment to check whether the time consistency of these three components can still be satisfied after the integrated systems have run together for many loops. The paths we choose and the experimental data are shown in Table 1. From this table, we can find that in the biggest case we solved, all the checked paths of these three components can traverse the unique main loop at least 250 times before the system can get blocked.

Fischer Mutual Exclusion Protocol. The second case study is the Fischer Mutual Exclusion Protocol [3]. This system consists of several competing processes which all attempt to enter the critical section. The automaton we use to model process_i is shown in Fig. 3. As these processes communicate with each other by a shared variable, in order to handle them in BACH's context (synchronization by share labels), we build an LHA: Shared Variable (denoted as SV) to represent all the evaluation and reset actions on the shared variable. For example, Fig. 3 shows an automaton which models all the possible actions that process_m and process_n can manipulate the shared variable. The LHA we used in case study for SV is based on this but with more processes. We conduct a group of experiments based on this protocol by scaling up the number of processes, i.e., 10 and 40 processes. The experiment and overhead data shown in Tables 2 and 3 can greatly support our belief in our tool's processing ability.

Nuclear Reactor System. The third case study is the Nuclear Reactor System from [17]. This system controls a nuclear reactor with n rods whose model is shown in Fig. 4. The system uses these rods to absorb neutrons one by one and hence lowers the temperature of the system. Each rod that has just been moved out of the heavy water must stay out of the heavy water and cool for several time units. The reachability specification we want to check is whether the system can keep running safely with these rods for a long time. Similarly to the experiments of the Fischer Mutual Protocol, we conduct two groups of experiments using 10 and 40 rods accordingly. The performance data are shown in Tables 4 and 5. We can see that the largest problem we can solve is a system with 40 rods and loops for 12 times. As the size of the linear program generated is linear in the size of system (number of paths and locations in each path), it is possible to check a system containing much more component automata with a shorter path for each component.

4.3 Comparison

We also conduct several experiments to compare the path-oriented reachability checker in BACH with other competitive tools. As we are concerning path-oriented reachability in this paper, we select the models which have only one path for each process for comparison. The automaton we chose are the acyclic version of Fischer Mutual Exclusion Protocol (Fig. 5) and Nuclear Reactor System (Fig. 6) which have only one path. The reachability specification we want to check is whether there exists such an execution that all the processes can enter the $s_4/recover$ location eventually.

We conduct the experiments using two state-of-the-art tools with respect to linear hybrid automata. They are the traditional LHA checker: PHAVer [6], and BMC-style HA checker: HySAT [9]. We also encode the path-oriented reachability problem of these acyclic models to SMT problems using classical interleaving encoding method, and solve them by MathSAT [10].

If the checker fails to generate results within 1 h, we treat it as a time out. The experiment data are shown graphically in Figs. 7 and 8 with respect to Fischer Mutual Exclusion Protocol and Nuclear Reactor System. From these figures, we can see that for the path-oriented reachability analysis, our tool is much more efficient than the other tools. BACH can handle the system consisting of 320 processes/rods in one hour, while PHAVer can only handle 11 processes/rods, HySAT exhausted all the memory when tackling problem of

Table 4 Data on the Nuclear Reactor System with 10 rods

Path	Rod_1	$(\langle out\rangle \xrightarrow[add\_1]{} \ldots \langle recover\rangle)^k \xrightarrow[recovery\_1]{} \langle out\rangle$		
	Rod_2	$(\langle out\rangle \xrightarrow[add\_2]{} \ldots \langle recover\rangle)^k \xrightarrow[recovery\_2]{} \langle out\rangle$		
	…	…		
	Rod_10	$(\langle out\rangle \xrightarrow[add\_10]{} \ldots \langle recover\rangle)^k \xrightarrow[recovery\_10]{} \langle out\rangle$		
	Con.	$(\langle rod\_0\rangle \xrightarrow[add\_1]{} \ldots \langle rod\_10\rangle)^k \xrightarrow[remove\_10]{} \langle rod\_0\rangle$		
k	Constraint	Variable	Memory (MB)	Time (s)
15	6,825	1,522	256	166.359
25	11,325	2,522	512	686.799
35	15,825	3,522	1,024	1,840.322
50	22,575	5,022	2,048	5,400.634

Table 5 Data on the Nuclear Reactor System with 40 rods

Path	Rod_1	$(\langle out\rangle \xrightarrow[add\_1]{} \ldots \langle recover\rangle)^k \xrightarrow[recovery\_1]{} \langle out\rangle$		
	Rod_2	$(\langle out\rangle \xrightarrow[add\_2]{} \ldots \langle recover\rangle)^k \xrightarrow[recovery\_2]{} \langle out\rangle$		
	…	…		
	Rod_40	$(\langle out\rangle \xrightarrow[add\_40]{} \ldots \langle recover\rangle)^k \xrightarrow[recovery\_40]{} \langle out\rangle$		
	Con.	$(\langle rod\_0\rangle \xrightarrow[add\_1]{} \ldots \langle rod\_40\rangle)^k \xrightarrow[remove\_40]{} \langle rod\_0\rangle$		
k	Constraint	Variable	Memory (MB)	Time (s)
4	7,485	1,682	256	220.724
6	11,085	2,482	512	738.245
9	16,485	3,682	1,024	2,121.607
12	21,885	4,882	2,048	5,144.191

18 processes/17 rods, and the largest problem which MathSAT can solve consists of 45 processes/63 rods, respectively. These data support our belief in BACH greatly. They also support our argument that for path-oriented reachability analysis, our technique outperforms the Cartesian Product technique used in general model checking methods and the interleaving encoding technique introduced in SAT-based bounded model checking methods.

The above experiments are preliminary, but they can basically indicate a clear potential of our approach. We believe if the linear programming package[2] in BACH is replaced by an advanced commercial package, the performance will be even better.

For a composed linear hybrid system considered in this paper, its components communicate with each other by shared labels. Recently, the shared variables whose values can also change with time has been used for communication in composed timed systems. For this kind of communication mode, a partial-order reduction based technique has been presented for bounded model checking of timed automata compositions [19]. Extending our approach for supporting this kind of mode will be a next work.

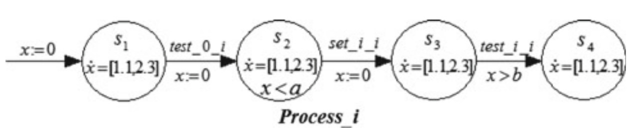

Fig. 5 Acyclic version of Fischer Mutual Exclusion Protocol

[2] The linear programming software package integrated in BACH is from OR-Objects of DRA Systems [18] which is a free collection of Java classes for developing operations research, scientific and engineering applications.

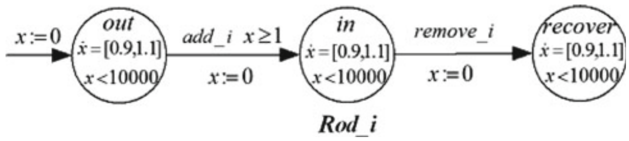

Fig. 6 Acyclic version of Nuclear Reactor System

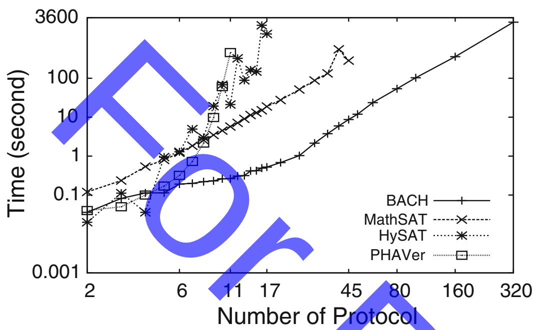

Fig. 7 Data on the acyclic Fischer Mutual Exclusion Protocol

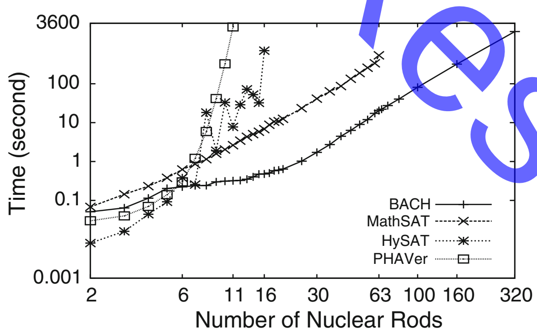

Fig. 8 Data on the acyclic Nuclear Reactor System

5 Conclusion

In this paper, we present an efficient path-oriented approach for bounded reachability analysis of composed linear hybrid automata. It is suitable for analyzing the systems with many components by selecting critical paths, respectively, while this task is quite insurmountable before due to the state explosion problem. This approach has been implemented into a prototype tool BACH, and the experiment data show that BACH has good performance and scalability.

Since the existing reachability analysis tools for linear hybrid systems do not scale well to the size of practical problems, it is necessary to do deeper analysis for the critical paths in the system as complement so as to increase the faith in the correctness of the system. We believe that BACH could become a powerful assistant to design engineers for reachability analysis of linear hybrid systems.

Acknowledgments Thanks to the anonymous reviewers and editors for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (No. 90818022, No. 60721002), the National Grand Fundamental Research 973 Program of China (No. 2009CB320702), and by the National 863 High-Tech Programme of China (No. 2009AA01Z148, No. 2007AA010302).

Appendix: Proofs of Theorems

Theorem 1 *Let $N = H_1||H_2||\ldots||H_m$ be a CLHA, and $\mathcal{R}(v, \varphi)$ be a reachability specification. N satisfies $\mathcal{R}(v, \varphi)$ if and only if there is a trail of N which satisfies $\mathcal{R}(v, \varphi)$.*

Proof The half of the claim, if N satisfies $\mathcal{R}(v, \varphi)$ then there is a trail of N which satisfies $\mathcal{R}(v, \varphi)$, can be proved as follows. According to Definitions 4 and 5, N satisfies $\mathcal{R}(v, \varphi)$ if and only if there is a behavior ω of N of the form

$$\omega = \begin{pmatrix} v_0 \\ \delta_0 \end{pmatrix} \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \begin{pmatrix} v_1 \\ \delta_1 \end{pmatrix} \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \begin{pmatrix} v_n \\ \delta_n \end{pmatrix}$$

which satisfies $\mathcal{R}(v, \varphi)$, where $v_i = (v_{i1}, v_{i2}, \ldots, v_{im})$ for any i $(0 \le i \le n)$. Similarly to the projection construction of a path in Sect. 2.2, we can construct a trail $\tau = (\omega_1, \omega_2, \ldots, \omega_m)$ of N as follows. For any k $(1 \le k \le m)$, we construct a behavior ω_k of H_k from ω as:

1. replacing any v_i with v_{ik} $(0 \le i \le n)$, and
2. for any $\xrightarrow[\sigma_{i-1}]{(\phi_{i-1},\psi_{i-1})} \begin{pmatrix} v_{ik} \\ \delta_{ik} \end{pmatrix}$ $(1 \le i \le n)$, if

$$(v_{i-1k}, \sigma_{i-1}, \phi, \psi, v_{ik}) \in E_k$$

then replacing it with $\xrightarrow[\sigma_{i-1}]{(\phi,\psi)} \begin{pmatrix} v_{ik} \\ \delta_{ik} \end{pmatrix}$ otherwise removing $\xrightarrow[\sigma_{i-1}]{(\phi_{i-1},\psi_{i-1})} \begin{pmatrix} v_{ik} \\ \delta_{ik} \end{pmatrix}$ and replacing $\begin{pmatrix} v_{i-1k} \\ \delta_{i-1k} \end{pmatrix}$ with $\begin{pmatrix} v_{i-1k} \\ \delta_{i-1k}+\delta_{ik} \end{pmatrix}$.

Let $\tau = (\omega_1, \omega_2, \ldots, \omega_m)$. Since ω is a behavior of N which satisfies $\mathcal{R}(v, \varphi)$, τ is a trail of N and satisfies $\mathcal{R}(v, \varphi)$.

The other half claim follows the claim that if there is a trail of N which satisfies $\mathcal{R}(v, \varphi)$ then there is a behavior of N satisfies $\mathcal{R}(v, \varphi)$, which can be proved as follows. Given a trail $\tau = (\omega_1, \omega_2, \ldots, \omega_m)$ of N which satisfies $\mathcal{R}(v, \varphi)$, where ω_i is a behavior of H_i $(1 \le i \le m)$ of the form

$$\begin{pmatrix} v_{i0} \\ \delta_{i0} \end{pmatrix} \xrightarrow[\sigma_{i0}]{(\phi_{i0},\psi_{i0})} \begin{pmatrix} v_{i1} \\ \delta_{i1} \end{pmatrix} \xrightarrow[\sigma_{i1}]{(\phi_{i1},\psi_{i1})} \ldots \xrightarrow[\sigma_{in_i-1}]{(\phi_{in_i-1},\psi_{in_i-1})} \begin{pmatrix} v_{in_i} \\ \delta_{in_i} \end{pmatrix},$$

we can get a total order of all the transitions in τ according to the time spot the transition is fired, and do the reverse work of the construction process in the last part simply, which will results in a single behavior ω of N of the form

$$\omega = \begin{pmatrix} v_0 \\ \delta_0 \end{pmatrix} \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \begin{pmatrix} v_1 \\ \delta_1 \end{pmatrix} \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \begin{pmatrix} v_n \\ \delta_n \end{pmatrix}$$

where $v_i = (v_{i1}, v_{i2}, \ldots, v_{im})$ $(0 \le i \le n)$. Since τ is a trail of N which satisfies $\mathcal{R}(v, \varphi)$, which means all the conditions in Definitions 2, 6, 7 are satisfied, ω is a behavior of N which satisfies $\mathcal{R}(v, \varphi)$ also. Above all, the claim holds. □

References

1. Henzinger, T.: The theory of hybrid automata. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, pp. 278–292 (1996)
2. Kesten, Y., Pnueli, A., Sifakis, J., Yovine, S.: Integration graphs: a class of decidable hybrid systems. In: Hybrid System. LNCS, vol. 736, pp. 179–208
3. 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. Theor. Comput. Sci. **138**, 3–34 (1995)
4. Henzinger, T., Kopke, P., Puri, A., Varaiya, P.: What's decidable about hybrid automata? J. Comput. Syst. Sci. **57**, 94–124 (1998)
5. Henzinger, T., Ho, P.-H., Wong-Toi, H.: HYTECH: a model checker for hybrid systems. In: Software Tools for Technology Transfer, vol. 1, pp. 110–122 (1997)
6. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. In: Proceeding of Hybrid Systems: Computation and Control'05. LNCS, vol. 2289, pp. 258–273 (2005)
7. Biere, A., Cimatti, A., Clarke, E., Strichman, O., Zhu, Y.: Bounded model checking. In: Advance in Computers, vol. 58. Academic Press, London (2003)
8. Zhang, L., Malik, S.: The quest for efficient boolean satifiability solvers. In: Proceedings of CAV 2002. LNCS, vol. 2404, pp. 17–36. Springer, Berin (2002)
9. Fränzle, M., Herde, C., Ratschan, S., Schubert, T., Teige, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. J. Satisf. Boolean Model. Comput. **1**, 209–236 (2007)
10. Audemard, G., Bozzano, M., Cimatti, A., Sebastiani, R.: Verifying industrial hybrid systems with MathSAT. Electron. Notes Theor. Comput. Sci. **119**(2), 17–32 (2005)
11. Audemard, G., Cimatti, A., Kornilowicz, A., Sebastiani, R.: Bounded model checking for timed systems. In: Conference on Formal Techniques for Networked and Distributed Systems. In: LNCS, vol. 2529, pp. 243–259 (2002)
12. Ábrahám, E., Becker, B., Klaedtke, F., Steffen, M.: Optimizing bounded model checking for linear hybrid systems. In: Proceedings of VMCAI 2005. LNCS, vol. 3385, pp. 396–412
13. Li, X., Jha, S., Bu, L.: Towards an efficient path-oriented tool for bounded reachability analysis of linear hybrid systems using linear programming. In: ENTCS, vol. 174, issue 3, pp. 57–70 (2007)
14. Bu, L., Li, Y., Wang, L., Li, X.: BACH: Bounded ReachAbility CHecker for linear hybrid automata. In: Proceedings of the 8th International Conference on Formal Methods in Computer Aided Design, pp. 65–68. IEEE Computer Society Press, Portland, OR, USA (2008)
15. Bu, L., Li, Y., Wang, L., Chen, X., Li, X.: BACH 2: Bounded ReachAbility CHecker for compositional linear hybrid systems. In: Proceedings of the 13th Design Automation and Test in Europe Conference, Dresden, Germany, pp. 1512–1517 (2010)
16. Alur, R.: Timed automata. In: Proceedings of the 11th International Conference on Computer-Aided Verification. In: LNCS, vol. 1633, pp. 8–22. Springer, Berlin (1999)
17. Wang, F.: Symbolic parametric safety analysis of linear hybrid systems with bdd-like data structures. IEEE Trans. Softw. Eng. **31**(1), 38–51 (2005)
18. OR-Objects of DRA Systems. http://OpsResearch.com/OR-Objects/index.html
19. Malinowski, J., Niebert, P.: SAT based bounded model checking with partial order semantics for timed automata. In: Proceedings of TACAS 2010, Paphos, Cyprus. LNCS, vol. 6015, pp. 405–419 (2010)