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

Technical Report No. NJU-SEG-2017-IJ-001

2017-IJ-001

Deriving Unbounded Reachability Proof of Linear Hybrid Automata During Bounded Checking Procedure

Dingbao Xie, Wen Xiong, Lei Bu, Xuandong Li

IEEE Transactions on Computers 2017

Deriving Unbounded Reachability Proof of Linear Hybrid Automata during Bounded Checking Procedure

Dingbao Xie, Wen Xiong, Lei Bu, and Xuandong Li

Abstract—Reachability analysis of linear hybrid automata (LHA) is an important problem. Classical model checking (CMC) technique is not scalable and not guaranteed to terminate. On the other hand, bounded model checking (BMC) is more cost-effective to conduct but can not guarantee the safety beyond the bound. In this paper, we seek to bridge the gap between BMC and CMC for reachability analysis of LHA. During BMC of LHA, typical procedures can discover sets of unsatisfiable constraint cores, which can be mapped back to path segments in the graph structure of LHA. If every path connecting the initial and target location has to go through such infeasible path segment, the target location is entirely not reachable. Based on this characteristic, we propose a LTL model checking based approach to check whether the target location is blocked. To further optimize the performance, we propose an automata based solution to check the LTL specification incrementally and adopt an on-the-fly algorithm to check the accepting condition to avoid an explicit construction of product automata.

Index Terms—Reachability checking, linear hybrid automata, irreducible infeasible set, linear temporal logic, Büchi automata

1 Introduction

Hybrid Automata [1] are popular frameworks used to model hybrid systems that exhibit both continuous and discrete dynamic behavior. Reachability analysis of hybrid automata is considerably difficult. Even for a relatively simple class such as linear hybrid automata (LHA), the reachability problem is undecidable [2]. Classical model checking (CMC) techniques attempt to compute exact reachable state space of LHA by the expensive polyhedral based computation which is sensitive to the number of continuous variables and not guaranteed to terminate. Despite decades of research on it, state-of-the-art tools based on CMC technique such as HyTech [3], PHAVer [4] and PHAVer's new implementation SpaceEx [5] do not scale well to problems of practical interest.

As an alternative technique to the classical symbolic model checking, bounded model checking (BMC) [6] was proposed to find bugs within a given threshold. The basic idea of BMC is to search for a counterexample with length no longer than an integer k in model executions, when the complete behavior state space is too complex to check by classical model checking. The typical approach of BMC reachability analysis of LHA is to encode the state space of LHA within the bound into a constraint set which can then be solved with Satisfiability Modulo Theories (SMT) solvers [7], [8]. However, as the technique needs to encode the whole state space within the bound first, the high complexity restricts the size of the system that can be handled.

The past decade has witnessed a significant amount of progress in constraint solving technologies, thanks to the emergence of highly efficient SMT solvers [7], [8], the size of the BMC problem of LHA that can be verified has increased significantly. However, the result of BMC verification only covers bounded behavior of the model. It remains a very interesting problem that whether we can get an unbounded proof of the system from a BMC procedure. In finite-state BMC, techniques such as k-induction [21], interpolation [28], IC3 [23] are widely used to obtain an unbounded model checking result. Nevertheless, they are difficult to conduct on infinite state systems like hybrid system.

Another way to solve the BMC problem of LHA is a path-oriented style approach [9]. The basic idea is to check one abstract path in the graph structure of an LHA at one time using Linear Programming (LP) to find whether there exists a feasible continuous behavior of the LHA along with this discrete path. As the number of paths within a given threshold is finite, we can answer the BMC problem of LHA by enumerating and checking all the candidate paths one by one [10]. Furthermore, when a path is proved to be infeasible, irreducible infeasible set (IIS) technique [12] can be deployed on the constraint set generated according to the path under checking to retrieve a minimal inconsistent constraint set. Such inconsistent constraint set can be mapped back to a path segment in the graph structure of LHA, which is infeasible for sure. As any path containing an infeasible path segment is definitely infeasible, the infeasible path segments, IIS, located by the underlying LP solver can be utilized to accelerate the BMC solving process [11], [16].

- *The authors are with the State Key Laboratory of Novel Software Techniques, Department of Computer Science and Technology, Nanjing University, Jiangsu 210093, China.*
  E-mail: {xdb, xiongwen}@seg.nju.edu.cn, {bulei, lxd}@nju.edu.cn.

Manuscript received 11 Jan. 2016; revised 20 Aug. 2016; accepted 24 Aug. 2016. Date of publication 29 Aug. 2016; date of current version 16 Feb. 2017. Recommended for acceptance by K. Schneider.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2016.2604308

In the above path-oriented BMC approach, as any candidate path has to avoid known infeasible path segments, such path segments can also be used to block the graph structure of LHA. During the experiment, it is not rare to see that after several path segments are blocked, the path-enumerating procedure cannot find any discrete path to reach the target location without touching any of the previously detected IIS path segments in the LHA model. In this situation, there does not exist a continuous behavior to reach the target location entirely, not only within the bound.

Based on this observation, we propose to exploit IIS path segments to prove whether a bounded unreachable argument can be extended to the unbounded state space. We propose a linear temporal logic (LTL) [18], [19] based sound but not complete approach to tackle such problem. In our approach, the discrete part of the LHA model, the graph structure is presented as a finite state transition system [19]. Then, we encode all the IIS path segments detected during BMC solving into an LTL formula. By checking whether the transition system satisfies the LTL formula, we can prove whether there exists a discrete path to reach the target location without containing any known IIS path segment in the graph structure of the LHA. If the LTL specification is satisfied, it means there will not exist such a path no matter how large the bound is given, which indicates that the bounded unreachable statement also stands in the complete unbounded state space.

A basic solution of integrating the BMC and proof procedure is to conduct the LTL based proof after the completion of the BMC analysis. It works as follows: Once an LHA model and a reachability specification are given, we start to perform the path-oriented bounded reachability analysis. When the BMC procedure finishes and reports the target is not reachable in the given bound, all the IIS path segments detected during the BMC solving process are encoded into an LTL formula. Then, a state-of-the-art LTL model checker is deployed to check whether the LTL specification is satisfied by the transition system constructed from the graph structure of LHA model. If yes, an unbounded result is proved, otherwise a bounded result is given.

In the above approach, as the LTL checking is conducted after the BMC procedure finishes, they do not have any interaction. When the LTL specification can be proved at an earlier time, it is a waste of time to continue the BMC analysis. In order to integrate the BMC and the proof procedure tightly to achieve better performance, we propose to perform the LTL checking whenever a new IIS path segment is located during the BMC procedure. In this way, the proof procedure will stop the BMC procedure as soon as the LTL specification becomes satisfied.

Although the tight integration can avoid meaningless BMC checking in some cases, it has to conduct repeated LTL model checking, in other words, call a third-party checker frequently, which incurs a high overhead. In order to solve this issue, we propose to check the LTL specification incrementally by constructing an equivalent büchi automata. With the help of a dedicated LTL-to-Büchi translation algorithm introduced in [24], we can generate a small büchi automata efficiently. Furthermore, we propose an on-the-fly algorithm to check the emptiness of the product transition system in the final step of LTL model checking. In this way, we can frequently construct only a small portion of the state space before a counterexample of the property being checked is found.

The above mentioned LTL and automata based approaches have been implemented into a bounded LHA checker BACH [10]. To evaluate the performance of our approach, we conduct a series of case studies on the enhanced BACH, and compare it with the state-of-the-art classical model checker, SpaceEx [5]. The set of experiment results shows that most of the benchmarks can be proved to be unreachable in the unbounded state space by analyzing the intermediate results of the BMC procedure without performing the expensive classical model checking. Furthermore, the experiments also show the performance of the approach presented in this paper outperforms the state-of-the-art CMC checker, SpaceEx, significantly.

2 Path-Oriented Reachability Analysis

In this section, we give the formal definition of linear hybrid automata and recap the underlying technique of path-oriented bounded reachability analysis that was proposed in [9], [10]. Furthermore, we present our method of using irreducible infeasible set technique to locate infeasible path segment from a path which is proved to be infeasible by the path-oriented checking to accelerate the BMC procedure [16].

2.1 Basic Path-Oriented Reachability Analysis

Definition 1. *The LHA considered in this paper is defined as a tuple $H = (G, X, \alpha, \beta, \phi, \psi)$, 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 initial location;*
  - *$q_{bad} \in Q$ is bad location (the location that should not be reachable);*
  - *Σ is a finite set of labels;*
  - *$E \subseteq (Q - \{q_{bad}\}) \times \Sigma \times Q$ is a finite set of (labeled)* transitions, *where no two outgoing transitions from a given location have the same label;*
- *X is a finite set of* state continuous variables.
- *α is a labeling function which maps each location in $Q - \{q_0\}$ to a set of* location invariants *which are of the form $a \leq \sum_{i=0}^{l} c_i x_i \leq b$ where $x_i \in X$, a, b and c_i are real numbers (a, b may be (minus) ∞).*
- *β is a labeling function which maps each location in $Q - \{q_0\}$ to a set of* flow conditions *which are of the form $\dot{x} \in [a, b]$ where $x \in X$, and a, b are real numbers ($a \leq b$). For any location q, for any $x \in X$, there is one and only one flow condition $\dot{x} \in [a, b]$.*
- *ϕ is a set of labeling functions which map each transition in E to a set of* transition guards *which are of the form $a \leq \sum_{i=0}^{l} c_i x_i \leq b$, where $x_i \in X$, a, b and c_i are real numbers (a, b may be (minus) ∞).*
- *ψ is a set of labeling functions which map each transition in E to a set of* reset actions *which are of the form $x := c$, where $x \in X$, c is a real number.*

We use sequences of locations to represent the evolution of an LHA from location to location. For an LHA $H = (G, X, \alpha, \beta, \phi, \psi)$, a *path segment* is a sequence of locations of the form $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0, \psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1, \psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1}, \psi_{n-1})} \langle v_n \rangle$, which satisfies $(v_i, \sigma_i, v_{i+1}) \in E$ for each i $(0 \leq i < n)$,

where $\sigma_i \in \Sigma$, $\phi_i \in \phi$, $\psi_i \in \psi$. A *path* in H is a path segment starting from the initial location q_0.

For a path in H of the form $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$, by assigning each location v_i with a time delay stamp δ_i we get a *timed sequence* of the form $\left\langle \begin{matrix} v_0 \\ \delta_0 \end{matrix} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{matrix} v_1 \\ \delta_1 \end{matrix} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{matrix} v_n \\ \delta_n \end{matrix} \right\rangle$ where δ_i $(0 < i \leq n)$ is a nonnegative real number and $\delta_0 = 0$ as $v_0 = q_0$ is initial location. This time sequence represents a behavior of H such that the system starts from the initial location v_0, stays there for δ_0 time units, which is 0, then jumps to v_1 and stays for δ_1 time units, and so on.

The behavior of an LHA can be described informally as follows. The automaton jumps from initial location v_0 to v_1 to initialize all the variables. Then, as time progresses, the values of all variables change continuously according to the flow conditions 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, σ, v') from v to v', whose all transition guards are satisfied by the current values of the variables. With a location changed by a transition (v, σ, v'), some variables are reset to the new value accordingly to the reset actions in ψ. Transitions are assumed to be instantaneous.

Let $H = (G, X, \alpha, \beta, \phi, \psi)$ be an LHA. Given a timed sequence ω of the form $\left\langle \begin{matrix} v_0 \\ \delta_0 \end{matrix} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{matrix} v_1 \\ \delta_1 \end{matrix} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{matrix} v_n \\ \delta_n \end{matrix} \right\rangle$, let $\zeta_i(x)$ represents the value of x $(x \in X)$ when the automaton has stayed at v_i for delay δ_i and $\lambda_i(x)$ represents the value of x at the time the automaton reaches v_i along with ω $(0 \leq i \leq n)$. It follows that $\lambda_{i+1}(x) = \begin{cases} d & \text{if } x := d \in \psi_i \\ \zeta_i(x) & \text{otherwise} \end{cases}$ $(0 \leq i < n)$.

Definition 2. *For an LHA $H = (G, X, \alpha, \beta, \phi, \psi)$, a timed sequence of the form $\left\langle \begin{matrix} v_0 \\ \delta_0 \end{matrix} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{matrix} v_1 \\ \delta_1 \end{matrix} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{matrix} v_n \\ \delta_n \end{matrix} \right\rangle$ represents a* behavior *of H if and only if the following condition is satisfiable:*

- $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$ *is a path;*
- *each variable $x \in X$ evolves according to its flow condition in each location v_i $(0 < i \leq n)$, i.e., $u_i \delta_i \leq \zeta_i(x) - \lambda_i(x) \leq u_i' \delta_i$ where $\dot{x} \in [u_i, u_i'] \in \beta(v_i)$;*
- *all the transition guards in ϕ_i $(1 \leq i \leq n-1)$ are satisfied, i.e., for each transition guard $a \leq \sum_{k=0}^{l} c_k x_k \leq b$ in ϕ_i, $a \leq \sum_{k=0}^{l} c_k \zeta_i(x_k) \leq b$;*
- *the location invariant of each location v_i $(1 \leq i \leq n)$ is satisfied, i.e., at the time the automaton reaches and leaves v_i, each constraint $a \leq \sum_{k=0}^{l} c_k x_k \leq b$ in $\alpha(v_i)$ $(1 \leq i \leq n)$ is satisfied, i.e., $a \leq \sum_{k=0}^{l} c_k \lambda_i(x_k) \leq b$ and $a \leq \sum_{k=0}^{l} c_k \zeta_i(x_k) \leq b$.*

Definition 3. *For an LHA $H = (G, X, \alpha, \beta, \phi, \psi)$, if a timed sequence of the form $\left\langle \begin{matrix} v_0 \\ \delta_0 \end{matrix} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{matrix} v_1 \\ \delta_1 \end{matrix} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{matrix} v_n \\ \delta_n \end{matrix} \right\rangle$ is a behavior of H, we say path $\rho = \langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \cdots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$ is* feasible, *and location v_n is* reachable *along ρ.*

For an LHA $H = (G, X, \alpha, \beta, \phi, \psi)$, 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 $a \leq \sum_{i=0}^{l} c_i x_i \leq b$ where $x_i \in X$ for any i $(0 \leq i \leq l)$, a, b and c_i $(0 \leq i \leq l)$ are real numbers, (a, b may be (minus) ∞).

Definition 4. *Let $H = (G, X, \alpha, \beta, \phi, \psi)$ 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 constraint in φ is satisfied when the automaton has stayed in v_n for delay δ_n, i.e., for each variable constraint $a \leq \sum_{k=0}^{l} c_k x_k \leq b$ in φ, $a \leq \sum_{k=0}^{l} c_k \zeta_n(x_k) \leq b$ where $\zeta_n(x_k)$ $(0 \leq k \leq l)$ 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 $\mathcal{R}(v, \varphi)$.*

According to Definitions 2 and 4, the reachability verification problem can be translated into the feasibility problem of a set of constraints on variables $\delta_i, \lambda_i(x)$, and $\zeta_i(x)$ where δ_i represents the time delay that the automaton stays in location v_i, $\lambda_i(x)$ and $\zeta_i(x)$ represent the value of x $(x \in X)$ when the automaton reaches and leaves the location v_i, respectively $(0 \leq i \leq n)$. If we use notation $\Theta(\rho, \mathcal{R}(v, \varphi))$ to represent this set of linear constraints, 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 answered by linear programming efficiently.

The basic idea of bounded model checking is to look for a counterexample with length no longer than some integer k in model executions. Given an LHA and a bound k, the number of candidate paths with length no longer than k is finite. Therefore, if we enumerate and check all the paths in the bound one by one, the bounded reachability problem can be tackled in the path-oriented BMC way.

2.2 IIS Based BMC Acceleration

Last paragraph gives a simple solution of the path-oriented bounded reachability analysis of LHA, which requires to enumerate and check all the candidate paths one by one in the graph structure of LHA. Nevertheless, when the given threshold is large and the graph structure of LHA is complex, there would be numerous paths to traverse, which could consume quite a lot of time.

Fortunately, as we use LP to judge the feasibility of a path ρ, irreducible infeasible set [12] technique can be deployed to locate a minimal infeasible set of the linear constraint set w.r.t. ρ.

Generally speaking, a set of linear constraints $\mathbb{C}$ is said to be satisfiable, if there exists a valuation of all the variables, which makes all the constraints in $\mathbb{C}$ to be true. Otherwise, $\mathbb{C}$ is unsatisfiable. If $\mathbb{C}$ is unsatisfiable, then IIS of $\mathbb{C}$ is a subset $\mathbb{C}' \subseteq \mathbb{C}$ that $\mathbb{C}'$ is unsatisfiable and for any $\mathbb{C}'' \subset \mathbb{C}'$, $\mathbb{C}''$ is satisfiable.

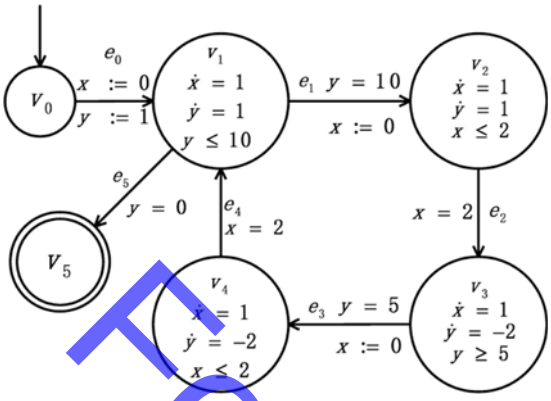


Fig. 1. Water-level monitor system.

Intuitively speaking, an IIS of a linear constraint set is an unsatisfiable set of constraints that becomes satisfiable if any constraint is removed. Therefore, given an infeasible path ρ, we can analyze the constraint set $\mathbb{C}$ generated according to this path to locate an IIS $\mathbb{C}'$. As each constraint in $\mathbb{C}$ is generated from a semantical element, e.g., location invariants or transition guards, in the locations and transitions from the model. Therefore, for each constraint in $\mathbb{C}$, we can locate the source location or transition in ρ straightforwardly. As a result, the constraint set located by the IIS technique can be mapped back to a path segment ρ' in ρ. In other words, once a path is proved to be infeasible, an infeasible path segment can be located from it by IIS analysis.

Fortunately, quoted from [12], the algorithm to locate the IIS from a unsatisfiable set is "simple, relatively efficient and easily incorporation into standard LP solvers". Many software packages are available, which supports the efficient analysis of a linear constraint set and locating of the minimal IIS, such as IBM CPLEX [30] and LINDO [31].

Now, let's see a path $\rho = \langle v_0\rangle \xrightarrow[e_0]{} \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$ in the Water-Level Monitor System (Fig. 1), which is proven to be infeasible. The IIS of the constraint set according to ρ given by the underlying LP solver is $\mathbb{C}'_\rho = \{\delta_{v_1^2} \ge 0, \zeta_{v_4}(x) - \lambda_{v_4}(x) = \delta_{v_4}, \zeta_{v_4}(y) - \lambda_{v_4}(y) = -2\delta_{v_4}, \lambda_{v_1^2}(y) = \zeta_{v_4}(y), \zeta_{v_1^2}(y) - \lambda_{v_1^2}(y) = \delta_{v_1^2}, \lambda_{v_4}(x) = 0, \lambda_{v_4}(y) = 5, \zeta_{v_4}(x) = 2, \zeta_{v_1^2}(y) = 0\}$, where v_1^2 represents the second occurrence of the location v_1 in ρ (the 6th location).

Now we show how can this IIS be mapped back to a path segment in ρ.

- $\delta_{v_1^2} \ge 0$ stands for the time elapsed in location v_1^2 is nonnegative.
- $\zeta_{v_4}(x) - \lambda_{v_4}(x) = \delta_{v_4}$, $\zeta_{v_4}(y) - \lambda_{v_4}(y) = -2\delta_{v_4}$, $\zeta_{v_1^2}(y) - \lambda_{v_1^2}(y) = \delta_{v_5}$, come from the flow conditions of x and y in location v_4: $\dot{x} = 1, \dot{y} = -2$, and location v_1^2: $\dot{y} = 1$.
- $\lambda_{v_4}(x) = 0$ and $\lambda_{v_4}(y) = 5$ come from the transition guard and reset action on transition e_3: $y = 5, x := 0$
- $\zeta_{v_4}(x) = 2$ and $\zeta_{v_1^2}(y) = 0$ come from the transition guards on transition e_4: $x = 2$ and e_5: $y = 0$.
- $\lambda_{v_1^2}(y) = \zeta_{v_4}(y)$ comes from the transition guard on transition e_4 as y is not reset on e_4.

Therefore, the related locations and transitions of $\mathbb{C}'_\rho$ include $v_4, v_1^2, e_3, e_4, e_5$. As the corresponding infeasible path segment ρ' of $\mathbb{C}'_\rho$ should be the shortest path segment which contains all these elements in ρ, it is $\langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$.

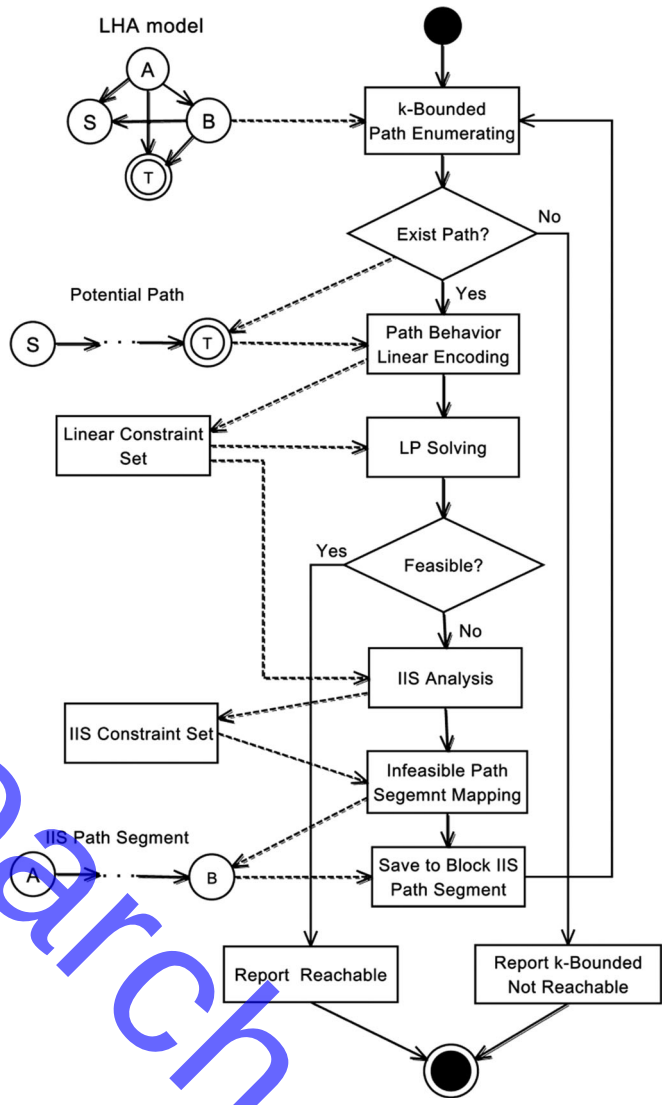


Fig. 2. Workflow of path-oriented BMC of LHA.

Clearly, a path ρ can be falsified for verification without calling the underlying decision procedure if it contains an infeasible path segment ρ', since the occurrence of ρ' in ρ will just be translated into the same set of unsatisfiable constraints with only name changed. Therefore, the path-enumerating procedure should rule out paths containing any known infeasible path segment.

For each bound k, the workflow of the IIS based path-oriented BMC solution for LHA is shown in Fig. 2 [16]. First, a potential path ρ with length no longer than k is enumerated and then analyzed by an LP solver. If ρ is feasible, the algorithm terminates and reports ρ as a witness, otherwise IIS technique is deployed to locate an infeasible path segment from ρ, which will be fed back to the path-traversing procedure to accelerate the BMC process. The algorithm terminates when no more candidate path can be found or a counterexample is confirmed.

Consider the automaton in Fig. 1, suppose we want to check whether location v_5 is reachable along a path with

bound 20 and the first enumerated path is $\rho_1 = \langle v_0\rangle \xrightarrow[e_0]{} \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$. The underlying LP solver proves ρ_1 is infeasible and locates an IIS path segment $\rho_1' = \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$, which will be fed back to the path-enumerating procedure to block such a path segment. Then the next found potential path to reach v_5 without containing ρ_1' in the graph is $\rho_2 = \langle v_0\rangle \xrightarrow[e_0]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$. Also, it is infeasible and the IIS path segment is $\rho_2' = \langle v_0\rangle \xrightarrow[e_0]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$ which is the path itself. The third candidate path to reach v_5 is $\rho_3 = \langle v_0\rangle \xrightarrow[e_0]{} \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_1\ \rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$. Nevertheless, we do not need to check this path by LP since it contains an exact IIS path segment ρ_1'. After the two IIS path segments ρ_1' and ρ_2' are blocked, no candidate path within the given bound can be found, which implies the target location is not reachable within the given step threshold.

3 Deriving Unbounded Result during BMC Solving Process

3.1 Motivation Overview

The last section gives a quick review of the path-oriented bounded reachability analysis of LHA [16]. The basic idea is to find an abstract path, sequence of locations, in the discrete graph structure of an LHA under verification first. Then, we can check whether there exists a feasible continuous behavior corresponding to certain discrete path. During the procedure, IIS technique is deployed to locate infeasible path segments in the graph model. Such path segments can be utilized to tailor the bounded graph structure of the LHA model under verification to accelerate the BMC solving process.

Clearly, the path segment w.r.t. each IIS is infeasible for sure. In other words, once a new IIS is found, a path segment in the graph structure of the LHA model under verification can be blocked. During experiments, we can see this scenario very often: an LHA model under verification does not have any path left after several IIS path segments are blocked. In this situation, the reachability specification can not be satisfied at all, not only in the bound. Consider the previous example in Fig. 1, the detected IIS path segments are $\rho_1' = \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$ and $\rho_2' = \langle v_0\rangle \xrightarrow[e_0]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$, and the target location is v_5. As we can see from the graph, if we block these two path segments in the graph structure of the model, there will not exist any path to reach v_5 anymore.

Based on the observation, if we can prove that there does not exist any path to reach the target location without going through certain infeasible path segments in the graph structure of an LHA model, we can tell that the reachability specification is not satisfied in the complete unbounded state space. In other words, the problem becomes as follows: We have a directed graph with an initial location and a target location, A set of path segments in the graph are blocked. Then, whether any path connecting the initial and target location exists in the directed graph?

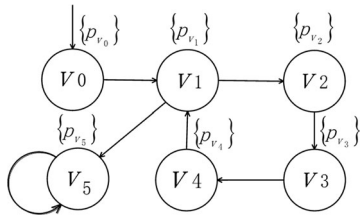


Fig. 3. DTS model of water-level monitor system.

3.2 LTL Based IIS Representation and Verification

The question raised in the end of the last paragraph concerns the reachability problem only on the discrete level, the graph structure G, of the LHA model H. In order to conduct reachability verification on G, we propose to extend G into a typical transition system (TS) [19] T first.

Definition 5. *Given an LHA $H = (G, X, \alpha, \beta, \phi, \psi)$, the related* discrete transition system *(DTS) of its graph structure $T = \{G', AP, L\}$*[1]*:*

- *$G' = (Q, q_0, q_{bad}, \Sigma, E')$ is an extension to the (labeled) location graph G, where*
  - *$Q, q_0, q_{bad}, \Sigma \in G$;*
  - *$E' \subseteq Q \times \Sigma \times Q$ is a finite set of (labeled)* transitions. *$E' = E \cup e$, where $E \in G$, e is a self-loop starting and ending in location q_{bad}.*
- *AP is the atomic proposition set in T. For each $q_i \in Q$, there exists an atomic proposition $p_{q_i} \in AP$.*
- *$L : Q \to 2^{AP}$ is a labeling function. For each $q_i \in Q$, $L(q_i) = \{p_{q_i}\}$.*

Let us review the LHA presented in Fig. 1, the DTS T modeling the graph structure of this LHA model is shown in Fig. 3.

It is well known that linear temporal logic [18] is a powerful temporal logic for describing system behaviors. Hence, we propose to use LTL to describe the property that there exists a path in the TS to reach the target location q_{bad} without containing any previously detected IIS path segment.

Avoiding IIS Path Segment. The first thing we need to do is to represent the IIS path segment in LTL. Suppose there is a path $\rho = \langle v_0\rangle \to \langle v_1\rangle \to \cdots \to \langle v_n\rangle$ in an LHA H, and ρ contains a path segment $\rho' = \langle v_i\rangle \to \langle v_{i+1}\rangle \to \cdots \to \langle v_j\rangle (i \geq 0 \wedge j \leq n)$. As T shares the same graph structure G with H, ρ and ρ' are also paths in the DTS T w.r.t. H. According to Definition 5, we have $L(v_k) = p_{v_k}, (0 \leq k \leq n)$. Therefore, $L(v_0) = p_{v_0}, \ldots, L(v_i) = p_{v_i}, \ldots, L(v_n) = p_{v_n}$ accordingly.

As ρ' starts from location v_i, v_i satisfies LTL formula $p_{v_i} \& X\ p_{v_{i+1}} \& \cdots \& \underbrace{X\ X\ \cdots X}_{j-i}\ p_{v_j}$. Based on this, given an IIS path segment $\rho' = \langle v_i\rangle \to \langle v_{i+1}\rangle \to \cdots \to \langle v_j\rangle$, we give the LTL formula representing this path segment as

$$IIS_{\rho'} = p_{v_i} \& X\ p_{v_{i+1}} \& \cdots \& \underbrace{X\ X\ \ldots X}_{j-i}\ p_{v_j}\,.$$

Furthermore, any path which does not contain path segment ρ' satisfies LTL formula $G(\neg IIS_{\rho'})$.

1. Note that this discrete transition system only focuses on the discrete graph structure of LHA. It is different from the one used to specify the semantics of LHA.

Reaching Target Location without Containing IIS Path Segment. The property that target location q_{bad} can be reached by a path in the graph structure of an LHA can be simply represented by an LTL formula $F\ p_{q_{bad}}$. Then, given a set of IIS path segments $\{\rho_1, \rho_2, \ldots, \rho_n\}$, the LTL formula which is true for a path reaching the target location without containing any above IIS path segment is shown below

$$(G(\bigwedge_{1\le i\le n} \neg IIS_{\rho_i})) \wedge F\ p_{q_{bad}},$$

where IIS_{ρ_i} represents the ith IIS path segment. As our target is to prove the nonexistence of such a path, the final LTL specification is the negation of the above formula

$$\neg((G(\bigwedge_{1\le i\le n} \neg IIS_{\rho_i})) \wedge F\ p_{q_{bad}}).$$

Theorem 1. *Given an LHA H, a target location q_{bad} and a set of IIS path segments $\{\rho_1, \rho_2, \ldots, \rho_n\}$, if the DTS model T w.r.t. H satisfies the LTL specification $\neg((G(\bigwedge_{1\le i\le n} \neg IIS_{\rho_i})) \wedge F\ p_{q_{bad}})$, then q_{bad} is not reachable in the complete state space of H.*

Proof [Proof by Contradiction]. Assume T satisfies the LTL specification and q_{bad} is reachable in H along a path $\rho = \langle v_0\rangle \rightarrow \langle v_1\rangle \rightarrow \cdots \rightarrow \langle v_n\rangle (v_n = q_{bad})$. Clearly, as T and H shares the same graph structure G, ρ is also a path in T.

As q_{bad} is reachable along ρ, this means there exists a feasible continuous behavior of H along ρ, therefore ρ doesn't contain any IIS path segments related with $\{\rho_1, \rho_2, \ldots, \rho_n\}$ for sure. As a result, the LTL formula $(G(\bigwedge_{1\le i\le n} \neg IIS_{\rho_i})) \wedge F\ p_{q_{bad}}$ is true for ρ. As we can see, ρ is the counterexample of the given LTL specification $\neg((G(\bigwedge_{1\le i\le n} \neg IIS_{\rho_i})) \wedge F\ p_{q_{bad}})$. This contradicts the assumption that H satisfies the LTL specification. [Hence, the supposition is false and the proposition is true.]

Now, let us go back to the LHA in Fig. 1. As mentioned in Section 2, it has two IIS path segments: $\langle v_0\rangle \rightarrow \langle v_1\rangle \rightarrow \langle v_5\rangle$, and $\langle v_3\rangle \rightarrow \langle v_4\rangle \rightarrow \langle v_1\rangle \rightarrow \langle v_5\rangle$ and the target location is v_5. According to the above encoding method, the associated LTL specification is

$$\neg((G(\neg(p_{v_0}\&X\ p_{v_1}\&X\ X\ p_{v_5}) \wedge \neg(p_{v_3}\&X\ p_{v_4}\& X\ X\ p_{v_1} \wedge X\ X\ X\ p_{v_5}))) \wedge F\ p_{v_5}).$$

It is worth noting that in Definition 1, we only allow one transition between any two locations. In this case, there is no need to encode the edge in the path segment into our encoding, and thus we can give a simplified encoding solution. In the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2016.2604308, we show that in the general case, when multiple transitions are allowed between two locations, our approach can also work by encoding edges into the corresponding LTL formula. □

3.3 Lazy Proof

Model checking of LTL property on a transition system is a well-studied problem and there are various efficient tools such as Spin [20], NuSMV [17] to tackle the problem. Therefore, we can take advantage of these off-the-shelf LTL checkers to prove the reachability of certain target location in the discrete graph structure efficiently.

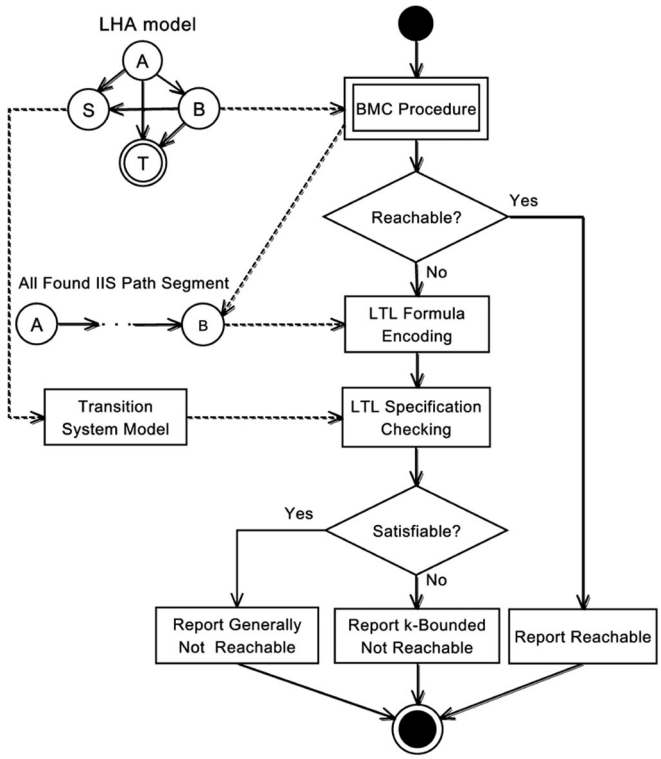


Fig. 4. Workflow of lazy proof.

Recently, IC3 [23] algorithm was proposed to perform SAT-based model checking without unfolding the transition relation. It proved to outperform the existing verification techniques for finite-state systems in the last years. With the help of the converting tool smvtoaiger [27], we can formulate a LTL model checking problem with AIGER [27] standard which is widely used in hardware verification. Afterwards, we can also solve the problem efficiently with tools based on IC3 technique, such as iimc [26].

Now, by using the mature checkers mentioned above to conduct LTL verification on the discrete transition graph of LHA model, we can prove that after certain path segments in the graph structure of the LHA model are blocked, whether there still has any potential path left to reach the target location. Clearly, if there does not exist a candidate path at all, then the target location is not reachable thoroughly. This provides a procedure that allows us to derive, in some cases a proof of unbounded result from a BMC solving process. As the LTL specification checking is conducted after the completion of BMC checking procedure, we call it *Lazy Proof*.

Workflow of Lazy Proof. Given an LHA model and a bound k, the workflow of lazy proof is shown in Fig. 4. The path-oriented BMC is conducted first and all the IIS path segments detected during the procedure are saved for the follow-up proof procedure. The BMC solving process is marked as "BMC Procedure" by double square and its detailed flow is shown in Fig. 2 previously. After the BMC analysis completes, if a feasible path is found then it is reported as a witness which can confirm the target location is reachable. Otherwise, all the IIS path segments identified in the BMC procedure are encoded into an LTL specification. After that, an state-of-the-art LTL model checker is

called to perform the unbounded proof by checking whether the specification is true on the DTS of the LHA model. If yes, then the target location is not reachable completely, otherwise a k-bounded unreachable conclusion will be given. In another word, this approach is sound but not complete.

3.4 Eager Proof

In the above paragraph, we show a basic solution to integrate the path-oriented BMC and LTL-based proof procedure. It attempts to derive an unbounded reachability result by collecting and analyzing the intermediate results, IIS path segments, of the BMC solving process. As the proof procedure has to wait for the completion of the BMC procedure and it can not give feedback to the BMC procedure, the lazy proof based solution is not as efficient as it could be.

Let us consider the following scenario: ten IIS path segments were located after the BMC procedure finishes. However, the first two IIS path segments are able to block the target location already. In this case, it will be a waste of time to continue the BMC analysis after the first two IIS path segments are located. In fact, if we can prove the target location is not reachable already, we can stop the BMC procedure in advance.

In order to solve the problem, we attempt to combine the LTL checking and BMC procedure tightly, to check the LTL specification during the BMC solving process. Different from the lazy proof, which starts the LTL proof after the completion of BMC procedure, we call the new approach *Eager Proof*.

Workflow of Eager Proof. We can adapt the lazy proof to eager proof by modifying its workflow slightly. As we want to prove the LTL specification whenever a new IIS is detected, the BMC solving procedure is not treated as a black box again. The updated workflow is shown in Fig. 5. The main change is the time spot of checking the satisfiability of the LTL specification. In the eager LTL-based approach, we call a mature LTL checker to check the LTL specification whenever a new IIS is detected during the BMC procedure. Besides, if the LTL checker finds the specification becomes satisfied, the BMC procedure will be stopped immediately and an unbounded result will be given.

3.5 Eager Automata-Based Proof

3.5.1 Incremental Büchi Construction

In the above paragraphs, we introduce two LTL based proof strategies to derive an unbounded result from a BMC procedure. The lazy LTL-based solution may waste time to do meaningless checking when an unbounded proof can be derived in an earlier time. In order to solve this problem, an eager LTL-based solution is proposed to check the LTL specification once an IIS is detected. However, the eager solution needs to call the third party LTL checker to check the LTL specification repeatedly while the lazy solution only checks the specification once. When the LTL specification is hard to prove or can not be proved at all, the overhead caused by repeated LTL model checking would slow down the whole analysis greatly.

The basic idea of LTL model checking is to construct an equivalent büchi automata from the LTL formula first and then find a strong connected component (SCC) containing the accepting state in it. In our problem, we will detect more and more IIS path segments as BMC procedure continues, which means the LTL specification is refined gradually. Inspired by the two facts, we decide to check the LTL specification by constructing a büchi automata incrementally during the BMC solving process instead of calling a third party checker.

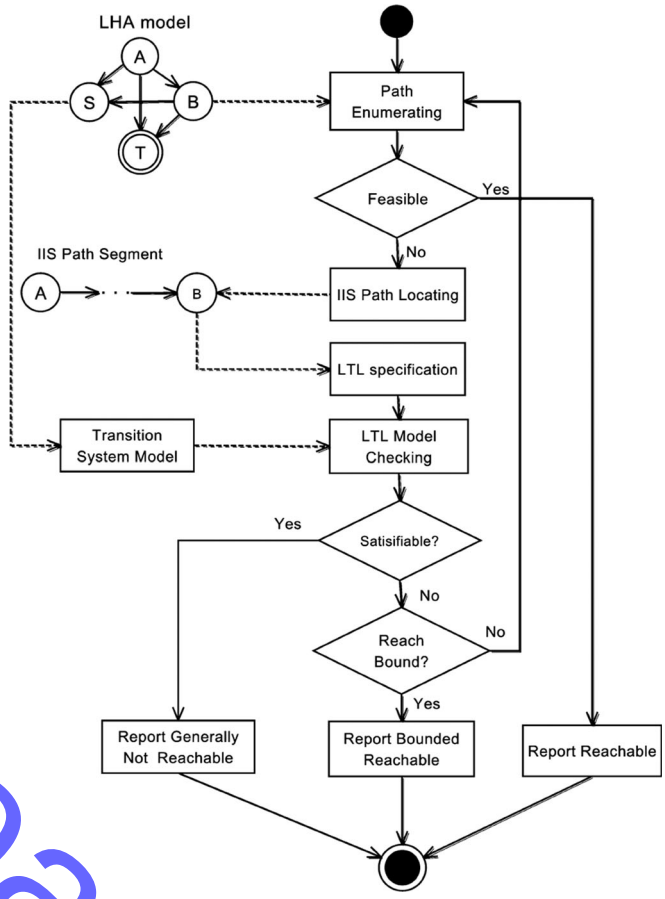


Fig. 5. Workflow of eager proof.

The LTL specification described in Section 3.2 has the following form: $\neg((G(\bigwedge_{1\leq i\leq n}\neg IIS_{\rho_i}))\wedge F\ p_{q_{bad}})$. In the corresponding büchi, we need to search for a counterexample path that satisfies LTL formula $(G(\bigwedge_{1\leq i\leq n}\neg IIS_{\rho_i}))\wedge F\ p_{q_{bad}}$. We decide to divide it into two parts: $G(\bigwedge_{1\leq i\leq n}\neg IIS_{\rho_i})$ and $F\ p_{q_{bad}}$ and search for the counterexample path in two stages. The reason is that the first part changes as more and more IIS path segments are detected while the second part remains the same all the time. It makes it easier to construct a büchi automata from LTL formula $G(\bigwedge_{1\leq i\leq n}\neg IIS_{\rho_i})$ incrementally.

We could apply an existing LTL-to-Büchi translation algorithm such as [25] to construct a büchi automata. However, since our LTL formula has a special structure, we can apply a dedicated incremental algorithm introduced in [24] to generate a very small büchi automata efficiently.[2]

The workflow of automata based approach is shown in Fig. 6. Given an LHA and a bound k, we first construct a transition system TS from the graph structure of LHA using commonly used method. Then we start the BMC procedure to enumerate candidate paths in the graph structure and check

2. Due to the space limit, please refer to [24] for detail of the algorithm to construct a büchi automata.

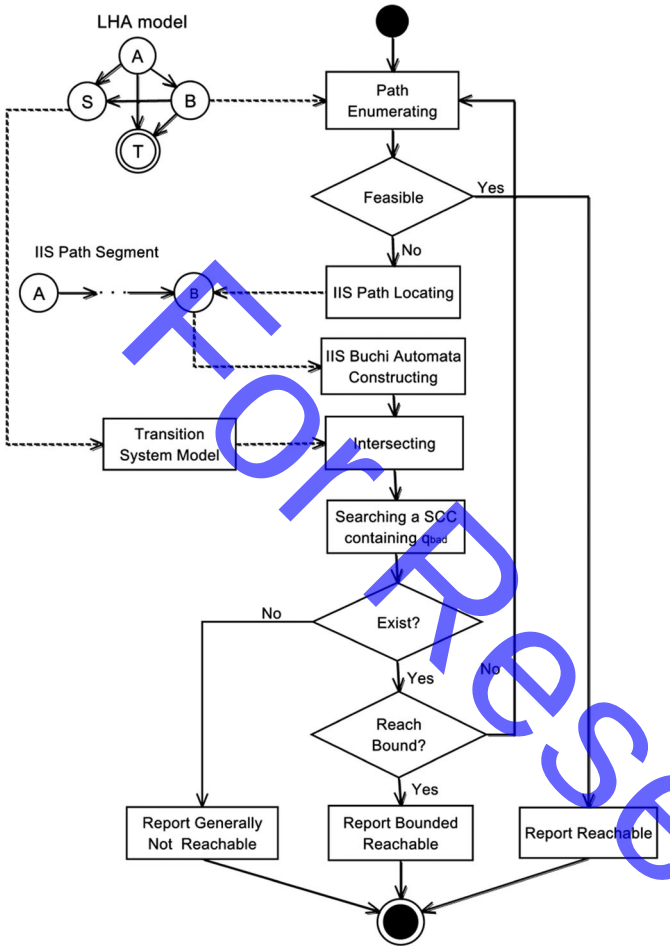


Fig. 6. Workflow of automata based eager proof.

whether it is feasible according to the semantic information along the path. If feasible, then we find a counterexample, report reachable; otherwise an IIS path segment is located and passed to the proof procedure to construct an IIS büchi automata. The IIS büchi automata is constructed incrementally according to the LTL formula $G(\bigwedge_{1\leq i\leq n} \neg IIS_{\rho_i})$ to rule out all the located IIS path segment.

After that we will intersect TS with the IIS büchi automata and then look for a path satisfying Fq_{bad} in the product transition system. If such path does not exist, the original LTL specification is satisfied. Then the checking process will terminate immediately and give an unbounded result. Otherwise, the BMC procedure continues. New IIS path segment will be located and the IIS büchi automata will be adapted incrementally using the algorithm introduced in study [24] to rule out all the located IIS path segments. The whole analysis will terminate when an unbounded result is proved or a given bound is reached.

3.5.2 On-the-Fly Emptiness Checking

As the LTL property we want to verify in the second stage is quite simple, we propose to check it on the fly, which can guide the construction of the product transition system while computing the intersection of the graph TS and IIS büchi. In this way, we may frequently construct only a small portion of the state space before we find a counterexample to the property being checked. The algorithm is shown in Algorithm 1: we start from the initial location of the product transition system and search for a cycle containing q_{bad} using the double depth first search (DFS) algorithm introduced in [35].

Algorithm 1. On-the-Fly Emptiness Checking

Input: graph transition system TS, IIS büchi automata B
Output: False if $TS \otimes B \models Fq_{bad}$, True otherwise.

```
procedure check(TS, B)
  p = TS.init
  q = B.init
  return dfs1(p, q)
end procedure
procedure dfs1(p, q)
  hash(p, q)
  for all successors p1 of p do
    for all successors q1 of q with label L(p1) do
      if not accept(q1) then
        continue
      end if
      if p1 == q_bad then
        return not dfs2(p1, q1)
      end if
      if (p1, q1) not in the hash table then
        if not dfs1(p1, q1) then
          return False
        end if
      end if
    end for
  end for
  return True
end procedure
procedure dfs2(p, q)
  flag(p, q)
  for all successors q1 of q with label L(p) do
    if not accept(q1) then
      continue
    end if
    if flagged (p, q1) then
      return True
    end if
    if dfs2(p, q1) then
      return True
    end if
  end for
  return False
end procedure
```

The typical double DFS algorithm is aimed at finding a path to a cycle containing the accepting state of a büchi automata and can be used to do on-the-fly checking. We adapt it to find an accepting run satisfying LTL formula Fq_{bad} in the intersection of a transition system and büchi automata. Procedure "check" returns true if the LTL specification is satisfied. Procedure "dfs1" is used to find an accepting run satisfying LTL formula Fq_{bad} and procedure "dfs2" is used to find a cycle containing q_{bad}. As every non accepting states in the IIS büchi automata has no outgoing transition, we can ignore them when traversing such states in the product automata since they can not lead to any new state.

In summary, the eager automata-based solution has following advantages over the LTL based approach:

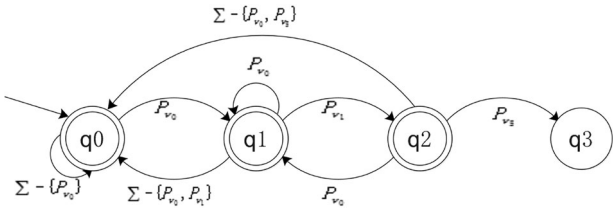


Fig. 7. Büchi automata for LTL formula $G(\neg(p_{v_0}\&X\ p_{v_1}\&X\ X\ p_{v_5})$.

- Compared with the lazy proof, it can stop the whole checking process as soon as the LTL specification becomes satisfied.
- In comparison to eager proof, as the büchi automata is constructed incrementally, it is much more efficient than constructing everything from scratch. Besides, the algorithm proposed in [24] consists of two stages: constructing and minimizing. The minimizing stage can also help to boost the performance. The size of the minimized büchi automata is not monotonically increasing since adding IIS might enable new minimization possibilities leading even to reduction.
- Last but not least, as the checking of the LTL specification is implemented in the eager automata-based approach, it can reduce the overhead brought by repeated calling the third-party LTL checkers greatly.

3.5.3 An Illustrative Example

Take the automaton in Fig. 1 for example again. Suppose the given bound is 10. We first construct an initial transition system TS from the graph structure of the LHA, Fig. 3, then the BMC procedure searches for candidate paths incrementally and the first detected IIS path segment is $\langle v_0\rangle \underset{e_0}{\longrightarrow} \langle v_1\rangle \underset{e_5}{\longrightarrow} \langle v_5\rangle$. We construct büchi automata ω_1 (shown in Fig. 7) from it and intersect ω_1 with TS. Then we search for a SCC containing q_{v_5} and succeed. So we continue the BMC procedure and get another IIS path segment $\langle v_3\rangle \underset{e_3}{\longrightarrow} \langle v_4\rangle \underset{e_4}{\longrightarrow} \langle v_1\rangle \underset{e_5}{\longrightarrow} \langle v_5\rangle$, then we construct büchi autotmata ω_2 (shown in Fig. 8) based on ω_1, which can rule out the two detected IIS path segments. At last, we intersect ω_2 with TS and find that the product transition system does not have state q_{v_5}, which shows the nonexistence of a counterexample. Then the proof procedure stops the BMC procedure immediately and gives an unbounded result. In this way, we are able to know a general model checking result by searching for a very small state space, which saves a lot of computation time.

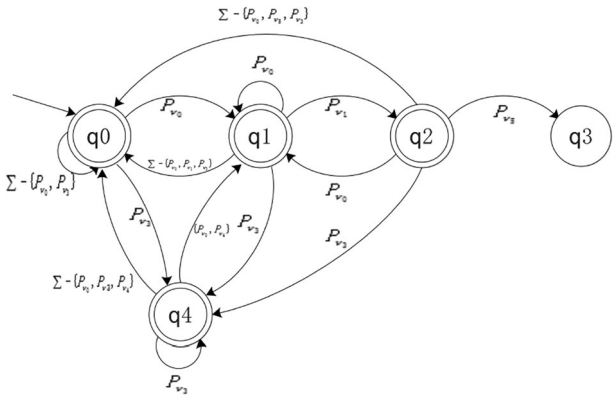


Fig. 8. Büchi automata for LTL formula $G(\neg(p_{v_0}\&X\ p_{v_1}\&X\ X\ p_{v_5})\ \wedge\neg(p_{v_3}\&X\ p_{v_4}\&X\ X\ p_{v_1}\wedge X\ X\ X\ p_{v_5}))$.

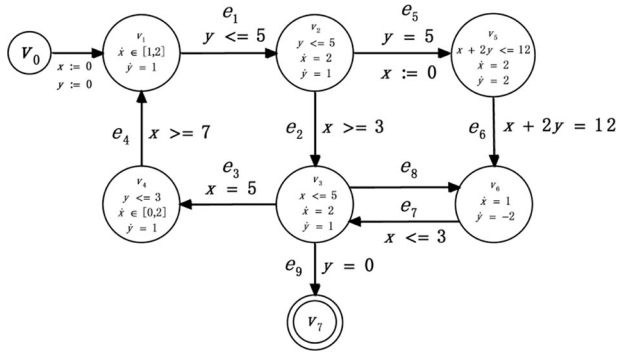


Fig. 9. Sample automaton.

4 Implementation and Case Studies

The LTL-based verification techniques presented in this paper have been implemented into a path-oriented bounded reachability checker of LHA-BACH [10]. For the lazy and eager LTL based solution, we use a typical LTL model checker NuSMV [17] to check the satisfiability of LTL specification. These two different implementations of BACH are marked as BACH(Lazy_NuSMV) and BACH (Eager_NuSMV) respectively. The implementation of the eager automata-based solution is marked as BACH (Automata). In addition, as mentioned in Section 3.3, we also implemented the IC3 based approach into BACH and the underlying IC3 based tool we selected is iimc [26] which supports analysis of LTL specification. We mark the implementation as BACH (IC3).

In order to evaluate the performance of the presented techniques, we carry out extensive evaluations over a set of widely-used benchmarks, which include the aforementioned water-level monitor system in Fig. 1, the temperature control system from [44], the communication based train control system from [16], sample automata in Fig. 9 and an extended version of the navigation benchmark [36]. The navigation example models the motion of a point robot in a n-dimensional cube. The cube is partitioned into m^n rectangular regions and each such region is associated with a vector field described by the flow equations. In order to increase both discrete and continuous complexity of the model, we use a generalization method introduced in [32] to generate a n-dimensional navigation model. As our work focuses on linear hybrid automata, the flow conditions of the velocity variables in the model are unified as in the range of $[-2, 2]$.

Aside from the above models, we also conduct case studies on a scalable automated highway system from [13] in Fig. 10. It is worth noting that the size of the highway system model can be easily expanded by introducing more cars into the system, which will increase new locations and variables in the model. The target locations, q_{bad}, are all marked by double circles in the models and they are all unreachable.[3]

3. Due to the space limit, please refer to the tool's website, http://seg.nju.edu.cn/BACH/tc16, for the detailed models.

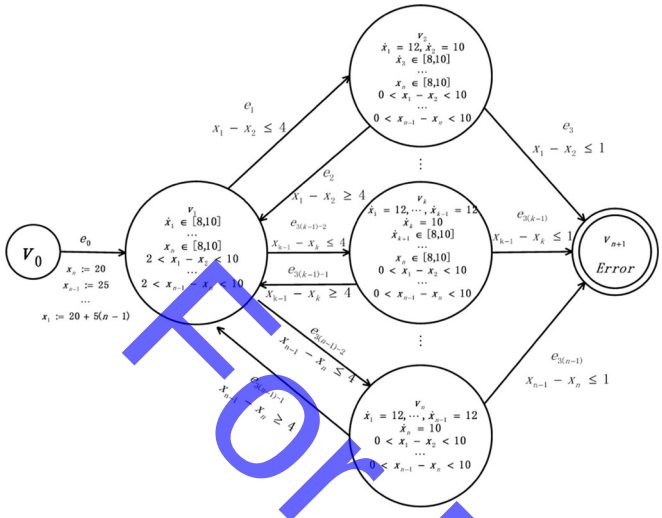


Fig. 10. Automated highway system.

4.1 Comparison between Different Proof Approaches

In the last section, we introduce several proof approaches to derive an unbounded proof after or during a BMC checking procedure. In order to evaluate the performance of these approaches, we conduct a series of experiments on the above mentioned benchmarks.

The experiments are conducted on a ThinkCenter workstation (Intel i5 Quad CPU 3.1 GHz, 8 GB RAM running Ubuntu 13 64 Bit). The time and memory usage limit for the experiment are set to one hour and 4 GB respectively. The value of the bound means the largest number of discrete locations that a path can have in the state space under searching. The executable BACH and the input models we use in experiments for both BACH and SpaceEx are all available from http://seg.nju.edu.cn/BACH/tc16.

The experiment data for the time and memory usage spent in each benchmark is shown in Table 1. The quickest result is marked in bold. In addition, the number of IIS path segments detected by the underlying LP solver is also given.

It is also worth mentioning that the time spent by BACH means the total time of the BMC and the unbounded proof procedure.

From the data, we can see that:

- All the implementations have consistent results on the benchmarks they can handle. Because the underlying technique is the same, we only change the proof strategy in attempt to improve the performance.
- In general, the performance of BACH (Eager_NuSMV) is the worst among the four implementations. Compared to BACH (Lazy_NuSMV), it needs to perform LTL model checking once a new IIS is located, in other words, call the third-party LTL checker repeatedly, which brings heavy overhead especially when the number of IIS is large. On the other hand, in comparison to BACH (Automata), it has to conduct repeated LTL model checking from scratch, which wastes a lot of time.
- BACH (Automata) outperforms BACH (Lazy_NuSMV) significantly when handling large models. The reason is twofold:
  1) We adopt a dedicated algorithm introduced in [24] to construct a small büchi automata efficiently. As the last step of LTL model checking is to check the emptiness of the product büchi, it is easier to do that in a smaller automata. For example, when handling the NAV_5_3 model, the automata based approach can prove that the LTL specification is satisfied in 3 seconds while it takes NuSMV over 40 seconds.
  2) In the automata based solution, the proof procedure can tell the BMC solving process to terminate as long as it finds the LTL specification is satisfied. In contrast, BACH(Lazy_NuSMV) has to wait for the completion of the BMC procedure. Take the LHA motorcade_500 for example, BACH(Automata) is able to prove the target

TABLE 1
Results of Applying Enhanced BACH to Different LHA Cases

				BACH (Automata)		BACH (Eager_NuSMV)		BACH (Lazy_NuSMV)		BACH(IC3)	
System	#locs	#vars	#IIS	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)
water	6	2	2	$\mathbf{0.5_U}$	<1	0.94_U	<1	0.94_U	<1	0.87_U	30
tcs	5	3	4	0.37_U	<1	$\mathbf{0.20_U}$	<1	0.97_U	<1	0.98_U	16
sample	8	2	9	0.63_B	24	0.60_B	<1	0.96_B	27	$\mathbf{0.41_B}$	21
train	8	2	2	0.9_U	<1	0.82_U	<1	1.02_U	<1	$\mathbf{0.3_U}$	<1
NAV_5_3	244	5	81	$\mathbf{3.19_U}$	784	-	M.O.	45.4_U	4,012	-	M.O
motorcade_5	7	5	4	0.1_U	<1	1.07_U	<1	$\mathbf{0.05_U}$	<1	0.4_U	<1
motorcade_10	12	10	9	0.49_U	<1	0.74_U	19	$\mathbf{0.12_U}$	<1	0.6_U	17
motorcade_20	22	20	19	0.58_U	<1	4.73_U	65	$\mathbf{0.53_U}$	61	1.1_U	25
motorcade_100	102	100	99	$\mathbf{1.71_U}$	25	-	T.O	6.66_U	164	15.7_U	389
motorcade_200	202	200	199	$\mathbf{4.01_U}$	49	T.O	-	61.8_U	653	115.3_U	3,299
motorcade_500	502	500	499	$\mathbf{29.5_U}$	212	T.O	-	T.O.	-	T.O.	-

#locs and #vars denote the number of locations and continuous variables in the LHA, respectively. #IIS denotes the number of detected IIS path segments. T.O. is a time out of 3,600 seconds. EXC means the checker threw an exception. When the result is T.O. or EXC, the corresponding blank of memory usage is marked as '-'. motorcade_i means automated highway system with i cars. NAV_5_3 denotes the navigation example with three partitions and five dimensions. The bound we set for BMC in BACH is 10 for all models and if BACH gives an unbounded result, corresponding time is marked with subscript U, otherwise, the result is marked with subscript B. Last but not least, all the target locations in the benchmarks are unreachable.

TABLE 2
Results of Applying BACH(Automata), BACH(Lazy_NuSMV) and Basic BACH to Different LHA Cases

System	bound	BACH(Automata) Time (s)	BACH(Automata) Mem. (MB)	BACH(Lazy_NuSMV) Time (s)	BACH(Lazy_NuSMV) Mem. (MB)	BACH(BMC Procedure) Time (s)	BACH(BMC Procedure) Mem. (MB)
water	1,000	0.22_U	<1	$\mathbf{0.19_U}$	<1	0.12_B	<1
	50,000	$\mathbf{0.57_U}$	<1	2.47_U	82	2.44_B	80
	200,000	$\mathbf{0.74_U}$	99	27.82_U	286	27.46_B	286
tcs	1,000	0.99_U	<1	$\mathbf{0.33_U}$	<1	0.25_B	<1
	100,000	$\mathbf{1.02_U}$	49	7.2_U	133	7.48_B	136
	200,000	$\mathbf{1.04_U}$	88	28.16_U	257	28.12_B	261
motorcade_20	1,000	$\mathbf{0.54_U}$	17	0.94_U	79	0.17_B	24
	50,000	$\mathbf{1.9_U}$	94	6.28_U	716	5.54_B	671
	100,000	$\mathbf{2.68_U}$	74	15.47_U	1,379	15.11_B	1,329
motorcade_100	1,000	$\mathbf{1.87_U}$	33	7.77_U	360	2.37_B	217
	10,000	$\mathbf{6.12_U}$	98	17.79_U	2,073	13.9_B	1,919
	20,000	$\mathbf{10.93_U}$	168	29.51_U	3,970	26.7_B	3,820

location is not reachable in the unbounded state space when unrolling the model to depth 3 and stop the BMC procedure immediately in 7.43 seconds. Unlike that, BACH(Lazy_NuSMV) has to finish the BMC search at first, which can not be even finished in the one hour time limit due to the huge search space.

As BACH(Lazy_NuSMV) outperforms BACH(IC3) in most of the cases in Table 1, we use BACH(Lazy_NuSMV) to stand for the lazy proof in the experiments. In order to further compare the performance of BACH(Automata) and BACH(Lazy_NuSMV), we also present the experiment data of applying them to analyze models in the benchmark with different bound settings. The result is shown in Table 2, we also list the time needed for the BMC analysis of the same problem as well. From the data we can see that:

- BACH (Automata) outperforms BACH (Lazy_NuSMV) significantly on almost all cases, especially on problems with large bound. In detail, the value of the bound has little influence on the performance of BACH(Automata) while the performance of BACH (Lazy_NuSMV) is greatly affected by it. BACH (Automata) adopts an eager way to check the satisfiability of the LTL specification and it can know the BMC result also stands beyond the bound whenever the collected IIS are strong enough to block the graph. In this case, the proof procedure can stop the BMC solving process to avoid the further meaningless search of the state space. In contrast, BACH (Lazy_NuSMV) has to wait for the completion of the BMC procedure whose performance depends on the value of the threshold heavily.
- Compare with the BMC procedure of BACH, BACH (Lazy_NuSMV) spends a bit more time to finish the whole analysis as it needs to conduct an extra proof besides the BMC checking. On the other hand, BACH(Eager_Automata) outperforms the BMC procedure of BACH in almost all cases, especially when the bound is large. The reason is that BACH(Automata) is able to know the unbounded statement when collecting enough IISes and will terminate to avoid the further meaningless analysis. In contrast, the basic BACH can not know such information in advance and has to search the state space within the bound exhaustively.

4.2 Comparison with State-of-the-Art Competitors

We have evaluated the performances of different proof approaches numerically in the last paragraph. In general, BACH(Automata) behaves the best in almost all cases, so we use it to represent BACH. Now, we focus on the comparison of the performance of BACH with the state-of-the-art classical LHA model checker, SpaceEx [5]. According to the documentation, SpaceEx supports two scenarios: PHAVer and Support Function. The former scenario uses the polyhedra based method to compute the exact reachable state space as the tool PHAVer [4], while the latter adopts support functions [29] to numerically compute the reachable state space. In the experiment, we compare BACH with SpaceEx in both two scenarios, which are marked as SpaceEx (PHA.) and SpaceEx (Supp.), respectively.

The experiment data for the time and memory usage spent in each benchmark is shown in Table 3. From these data, we can see that:

- BACH successfully proved the bounded unreachable results also stand in the unbounded state space for most of the benchmarks, 9/10, except the sample automaton. This confirms that the LTL-based approach in BACH is not complete, however it increases our confidence that many practical cases can be proved to be completely unreachable due to the existence of IIS path segments. In this situation, BACH can produce an unbounded result. BACH only fails to give an unbounded result for the sample automaton. The reason is that in this model the IISes found during the BMC procedure are not able to block the graph.
- Comparing with SpaceEx:
  - As pointed out in the SpaceEx's documentation and website, the support function scenario, SpaceEx (Supp.), is more suitable to handle hybrid automata with piecewise affine dynamics, for example model with flow condition like $\dot{x} = Ax + Bu + c$. For the class of LHA

TABLE 3
Results of Applying BACH(Automata) and SpaceEx to Different LHA Cases

System	#locs	#vars	BACH(Automata)			SpaceEx(PHA.)		SpaceEx(Supp.)	
			#IIS	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)
water	6	2	2	0.5_U	<1	$\mathbf{0.07_U}$	<1	0.22_U	8
tcs	5	3	4	0.37_U	<1	T.O.	-	$\mathbf{0.36_U}$	9
sample	8	2	9	0.63_B	24	$\mathbf{0.93_U}$	<1	EXC	-
train	8	2	2	0.9_U	<1	$\mathbf{0.62_U}$	<1	1.24_U	25
NAV_5_3	244	5	81	$\mathbf{3.19_U}$	784	16.15_U	27	-	M.O.
motorcade_5	7	5	4	$\mathbf{0.1_U}$	<1	4.94_U	16	T.O.	-
motorcade_10	12	10	9	$\mathbf{0.49_U}$	<1	T.O.	-	T.O.	-
motorcade_20	22	20	19	$\mathbf{0.58_U}$	<1	T.O.	-	T.O.	-
motorcade_100	102	100	99	$\mathbf{1.71_U}$	25	T.O.	-	T.O.	-
motorcade_200	202	200	199	$\mathbf{4.01_U}$	49	T.O.	-	T.O.	-

considered in this paper, SpaceEx prefers to use the "PHAVer" scenario, SpaceEx (PHA.), to handle. As we can see from Table 3, the performance of SpaceEx (PHA.) is better than SpaceEx (Supp.) in general when handling the class of LHA considered in this paper. Therefore, we focus on the comparison between SpaceEx (PHA.) and BACH in the following paragraph.

- On small cases, both BACH and SpaceEx (PHA.) finish 4 of the 5 cases efficiently. For example, for most of the models with number of locations and variables smaller than 10, both BACH and SpaceEx can solve them quickly in less than 1 second.
- On the other hand, as the reachability analysis of LHA is undecidable, classical model checking technique do not guarantee to terminate. The experiment shows that SpaceEx (PHA.) times out when dealing with the Temperature Control System model, which has only three continuous variables and five discrete locations. The reason is that the value range of the variables in this model is not closed. Therefore, the classical fix point based computation can not terminate. In contrast, BACH is guaranteed to terminate, as it is BMC-based, and can give an unbounded result in many cases according to the experiment.
- On large cases, BACH has better scalability. Take the automated highway system for example, BACH can solve such system with 200 cars, 202 locations and 200 continuous variables, in less than 70 s, while SpaceEx can only handle the system with five cars within the 1 h time limit. As the polyhedra based computation is sensitive to the number of continuous variables in the LHA, the performance of SpaceEx degrades greatly when the size of the model increases. Different from SpaceEx, BACH conducts light-weight BMC checking first. Then, BACH utilizes the IIS found during BMC process to derive an unbounded result using LTL model checking which is a very mature and efficient technique.

5 Related Work

Reachability analysis of linear hybrid automata is a very important problem. Classical model checking techniques [3], [4], [5] try to compute the complete reachable state space of LHA. The basic idea is to compute iteratively the reachable state space of the next-step based on the current state space. The algorithm terminates if the reachable state space of the next-step is contained by the current state space. However, such termination is not guaranteed. Furthermore, as the computation is very sensitive to the number of continuous variables, the state-of-the-art tools developed based on this technique do not scale well to the size of practical problems.

On the other hand, inductive invariant generation has been used in the safety/reachability analysis of hybrid system. In [38], the authors propose to use a function of state called barrier certificate to separate the unsafe region from all possible trajectories starting from a given set of initial conditions. However, this method is difficult to handle system with mixed equations and inequalities [39]. Inductive invariant generation of hybrid system has been studied in the area of theorem proving as well. In study [39], the differential invariants are computed as fixed points by differential logic for hybrid systems in an interactive way by KeYmaera.

In study [28], McMillan proposed an interpolation based approach for unbounded model checking of finite-state transition system. They proposed to compute an over-approximation of the forward reachable state space using SAT-based interpolation. Interpolation has been widely applied in different areas like software checking [42], [43]. It has been extended to nonlinear hybrid system in [40] as well. Craig interpolation (CI) plays the key role in these studies. It remains an important problem to study the scalable approach for CI synthesis for large and complex system [41].

Recently, bounded model checking [6] has attracted a lot of attention as an alternative technique to the classical model checking. The basic idea of this approach is to search for a counterexample whose length is bounded by some integer k in model executions. However, the result of BMC verification only covers the bounded behavior of the LHA model. It remains a very interesting problem that whether we can acquire an unbounded proof from a BMC procedure.

Sheeran et al. proposed an induction based approach called k-induction [21] to check the safety property of finite-state transition system. The idea of this technique is to prove that if a set of states is not reachable in k step, then it is completely not reachable. As hybrid automata contains both discrete and continuous behavior, such approach can not be directly applied. In [22], Moura et al. proposed to use k-induction to verify timed and hybrid automata and they

generalized the simple path condition used in original k-induction to simulate relations. As the implementations of the work are not available now, we cannot compare the performance between our approach and [22] numerically. However, as the unbounded analysis of [22] is conducted by performing induction on continuous dynamics of LHA and the procedure of generating strengthened invariants requires quantifier-elimination, it demands a high computational complexity, which greatly restricts the size of the problem that can be solved.

There are also works focusing on CEGAR of LHA. Study [32] presented a hybrid abstraction based CEGAR loop for the class of rectangular hybrid automata, which is a special subclass of the LHA considered in this paper. Study [37] utilized the counterexample path to synthesize good parameters during the verification in a similar path-oriented way, while the counterexample in our work is used to refine the graph of the LHA.

Study [13] proposed a CEGAR method, called "Iterative Relaxation Abstraction", for LHA by dropping variables from original LHA in each iteration, and asked PHAVer to solve the simplified model. Since this technique still relies on PHAVer, in other words, geometric computation, as the underlying checker for the abstract model, the computation burden is still heavy. Although the implementations of this work are not available as well, this work [13] also reported its performance on the motorcade series problem. According to [13], the largest motorcade model it solved has 19 variables included. It took [13] 652.51 seconds to solve such a problem, while for a larger problem, $motorcade\_20$, our tool BACH solved it in only 0.71 seconds. Furthermore, BACH solved a much larger system $motorcade\_500$ in only 7.43 seconds successfully.

In [34], Fehnker et al. proposed to locate "cut set" path segments in the graph structure of the LHA model and then validate each cut set one by one in the original model. If a cut set was confirmed to be invalid in the original model, there does not exist a behavior in the LHA that connects the initial location to the target. Hence, the target location is not reachable. Basically, the cut set path segments used in [34] are certain key path segments which are shared by many paths on the discrete graph level. Therefore, they have to validate such cut set path segments in the original LHA model. While in this paper, we use IIS technique to analyze the continuous elements of the path to locate the infeasible path segment core after one path is proved to be infeasible on the continuous level by LP solver. Then, such IIS path segments, which are definitely infeasible, can be removed from the graph structure to benefit our path-enumerating based BMC procedure. Clearly, the IIS path segment used in this paper could be any path segment in the graph structure and hence is different from the cut set path segment in [34].

There are also related works which manipulate infeasible path segments to block the target node in different area. In study [24], Segelken proposed an ω-automata based approach for CEGAR [14] verification of step-discrete linear hybrid models. In each CEGAR iteration, they present the spurious counterexample path found in the last iteration as an LTL formula which is similar to our work. After that, in order to exclude such spurious counterexample path in the refinement of the abstract model, the author proposed an incremental algorithm to translate the corresponding LTL formula to an ω-automata using a dedicated algorithm which is more efficient than the general LTL-to-Büchi translation algorithm [25]. Then the refinement will be done by computing the Cartesian product of the ω-automata and the abstract model. If the product automata satisfies the safety property, then the original model is safe. This efficient LTL-to-Büchi translation algorithm is adopted in our automata based proof to achieve better performance.

On the other side, the difference between this paper and [24] mainly dwells in the following points. First, the counterexample in their CEGAR iteration is identified by LTL model checking on the discrete state space of the step-discrete linear hybrid model. In contrast, the counterexample in our work is given by the BMC procedure of the continuous behavior of the LHA model. Another main difference is that, in the LTL specification checking, we propose an on-the-fly algorithm to check whether the product automata satisfies the LTL formula F(q_{bad}), while they adopt a commonly-used approach which has to compute the intersection first.

A similar idea has been proposed in software verification. In [33], Heizmann proposed to compute appropriate abstractions of programs based on statements as alphabet atoms in an automata framework and block inconsistent path segments on the automata level. This idea shares similar merits with us. However, they focus on finding the right abstraction of a program for a given property to dismiss a set of infeasible paths, while our work locates the minimal infeasible path segments directly. It would be an interesting topic to take advantage of static analysis techniques and interpolation generation to refine the infeasible path segments and automata in our future work.

6 Conclusion

Reachability analysis of linear hybrid automata is very difficult. Classical full state space reachability verification is not scalable, while bounded model checking is much more efficient to conduct but the result only covers the bounded behavior of the model. During path-oriented BMC of LHA, a set of infeasible path segments can be located by IIS technique. The target location is completely not reachable when it can not be reached without going through such infeasible path segments.

In this paper, we propose an LTL-based solution to derive an unbounded proof from the BMC procedure by proving whether the target location is blocked by the infeasible path segments located during the BMC checking. We implemented the proposed techniques into a bounded LHA checker BACH. The experiments show that, although this approach is not complete, BACH can give an unbounded result for most of the benchmarks successfully, efficiently and also with a good scalability.

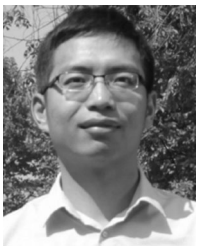
Acknowledgments

The authors would like to thank the editors and anonymous reviewers for valuable advices on improving this paper. This work is supported by the National Key Basic Research Program of China (2014CB340703), the Joint NSFC-ISF Research Program, jointly funded by the National Natural Science Foundation of China and the Israel Science

Foundation (No.61561146394), the National Natural Science Foundation of China (No.61572249, No.61632015, No.91318301, No.61321491, No.61472179), and partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization. Lei Bu is the corresponding author. This paper presents in a coherent and expanded form material that appears in the conference venue [15].

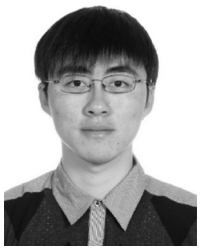
REFERENCES

[1] T. A. Henzinger, "The theory of hybrid automata," in *Proc. 11th Annu. IEEE Symp. Logic Comput. Sci.*, 1996, pp. 278–292.
[2] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *J. Comput. Syst. Sci.*, vol. 57, pp. 94–124, 1998.
[3] T. A. Henzinger, P. H. Ho, and H. Wong-Toi, "HYTECH: A model checker for hybrid systems," *Int. J. Softw. Tools Technol. Transfer*, vol. 1, pp. 110–122, 1997.
[4] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," in *Proc. 8th Int. Workshop Hybrid Syst. Comput. Control*, 2005, pp. 258–273.
[5] G. Frehse, et al. "SpaceEx: Scalable verification of hybrid systems," in *Proc. 23rd Int. Conf. Comput. Aided Verification*, 2011, pp. 379–395.
[6] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," in *Advance in Computers*, vol. 58. New York, NY, USA: Academic, 2003, pp. 118–149.
[7] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani, "Verifying industrial hybrid systems with MathSAT," *Electron. Notes Theoretical Comput. Sci.*, vol. 119, no. 2, pp. 17–32, 2005.
[8] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. Theory Practice Softw. 14th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2008, pp. 337–340.
[9] X. Li, S. K. Jha, and L. Bu, "Towards an efficient path-oriented tool for bounded reachability analysis of linear hybrid systems using linear programming," *Electron. Notes Theoretical Comput. Sci.*, vol. 174, no. 3, pp. 57–70, 2007.
[10] L. Bu, Y. Li, L. Wang, and X. Li, "BACH: Bounded ReAchability CHecker for linear hybrid automata," in *Proc. Int. Conf. Formal Methods Comput.-Aided Des.*, 2008, pp. 65–68.
[11] L. Bu, Y. Yang, and X. Li, "IIS-guided DFS for efficient bounded reachability analysis of linear hybrid automata," in *Proc. 7th Int. Haifa Verification Conf. Hardware Softw. Verification Testing*, 2012, pp. 35–49
[12] J. Chinneck and E. W. Dravnieks, "Locating minimal infeasible constraint sets in linear programs," *ORSA J. Comput.*, vol. 3, pp. 157–168, 1991.
[13] S. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke, "Reachability for linear hybrid automata using iterative relaxation abstraction," in *Proc. 10th Int. Conf. Hybrid Syst. Comput. Control*, 2007, pp. 287–300.
[14] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Proc. 12th Int. Conf. Comput. Aided Verification*, 2000, pp. 154–169
[15] D. Xie, L. Bu, and X. Li, "Deriving unbounded proof of linear hybrid automata from bounded verification," in *Proc. IEEE Real-Time Syst. Symp.*, 2014 pp. 128–137.
[16] D. Xie, L. Bu, and X. Li, "SAT-LP-IIS joint-directed path-oriented bounded reachability analysis of linear hybrid automata," *Formal Methods Syst. Des.*, vol. 45, no. 1, pp. 42–62, 2014.
[17] A. Cimatti, et al., "NuSMV 2: An opensource tool for symbolic model checking," in *Proc. 14th Int. Conf. Comput. Aided Verification*, 2002, pp. 359–364.
[18] A. Pnueli, "The temporal logic of programs," in *Proc. 18th Annu. Symp. Found. Comput. Sci.*, 1977, pp. 46–57.
[19] C. Baier and J. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press.
[20] G. Holzmann, "The model checker SPIN," *IEEE Trans. Softw. Eng.*, vol. SE-23, no. 5, pp. 279–295, May 1997.
[21] M. Sheeran, S. Singh, and G. Stalmarck, "Checking safety properties using induction and a SAT solver," in *Proc. 3rd Int. Conf. Formal Methods Comput.-Aided Des.*, 2000, pp. 108–125.
[22] L. De Moura, H. Rueß, and M. Sorea, "Bounded model checking and induction: From refutation to verification," in *Proc. 15th Int. Conf. Comput. Aided Verification*, 2003, pp. 14–26.
[23] A. R. Bradley, "SAT-based model checking without unrolling," in *Proc. 12th Int. Conf. Verification Model Checking Abstract Interpretation*, 2011, pp. 70–87.
[24] M. Segelken, "Abstraction and counterexample-guided construction of ω-automata for model checking of step-discrete linear hybrid models," in *Proc. 19th Int. Conf. Comput. Aided Verification*, 2007, pp. 433–448.
[25] F. Somenzi and R. Bloem, "Efficient Büchi automata from LTL formulae," in *Proc. 12th Int. Conf. Comput. Aided Verification*, 2000, pp. 248–263.
[26] IIMC. (2011). [Online]. Available: http://ecee.colorado.edu/wpmu/iimc/
[27] AIGER. (2006). [Online]. Available: http://fmv.jku.at/aiger/
[28] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proc. 19th Int. Conf. Comput. Aided Verification*, 2003, pp. 1–13.
[29] C. Le Guernic and A. Girard, "Reachability analysis of linear systems using support functions," *Nonlinear Anal.: Hybrid Syst.*, vol. 4, no. 2, pp. 250–262, 2010.
[30] CPLEX. (2009). [Online]. Available: http://www-01.ibm.com/software/integration/optimi-zation/cplex-optimizer/
[31] Systems Inc., L. (1995). [Online]. Available: http://www.lindo.com/products/api/dllm.html
[32] P. Prabhakar, P. S. Duggirala, S. Mitra, and M. Viswanathan, "Hybrid automata-based CEGAR for rectangular hybrid systems," in *Proc. 14th Int. Conf. Verification Model Checking Abstract Interpretation*, 2013, pp. 48–67.
[33] M. Heizmann, J. Hoenicke, and A. Podelski, "Software model checking for people who love automata," in *Proc. 25th Int. Conf. Comput. Aided Verification*, 2013, pp. 36–52.
[34] A. Fehnker, E. M. Clarke, S. K. Jha, and B. Krogh, "Refining abstractions of hybrid systems using counterexample fragments," in *Proc. 8th Int. Workshop Hybrid Syst. Comput. Control*, 2005, pp. 242–257.
[35] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.
[36] A. Fehnker and F. Ivancic, "Benchmarks for hybrid systems past HyTech," in *Proc. 7th Int. Workshop Hybrid Syst. Comput. Control*, 2004, pp. 326–341.
[37] G. Frehse, S. Kumar Jha, and B. Krogh, "A counterexample-guided approach to parameter synthesis for linear hybrid automata," in *Proc. 11th Int. Workshop Hybrid Syst. Comput. Control*, 2008, pp. 187–200.
[38] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *Proc. 7th Int. Workshop Hybrid Syst. Comput. Control*, 2004, pp. 477–492.
[39] A. Platzer and E. M. Clarke, "Computing differential invariants of hybrid systems as fixedpoints," in *Proc. 20th Int. Conf. Comput. Aided Verification*, 2008, pp. 176–189.
[40] S. Kupferschmid and B. Becker, "Craig interpolation in the presence of non-linear constraints," in *Proc. 9th Int. Conf. Formal Modeling Anal. Timed Syst.*, 2011, pp. 240–255.
[41] R. Sharma, A. V. Nori, and A. Aiken, "Interpolants as classifiers," in *Proc. 24th Int. Conf. Comput. Aided Verification*, 2012, pp. 71–87.
[42] T. A. Henzinger, R. Jhala, and G. Sutre, "Software verification with BLAST," in *Proc. 10th Int. Conf. Model Checking Softw.*, 2003, pp. 235–239.
[43] D. Kroening and G. Weissenbacher, "Interpolation-based software verification with Wolverine," in *Proc. 23rd Int. Conf. Comput. Aided Verification*, 2011, pp. 573–578.
[44] R. Alur, et al. "The algorithmic analysis of hybrid systems," *Theoretical Comput. Sci.*, vol. 138, pp. 3–34, 1995.



Dingbao Xie received the BS and PhD degrees in computer science from Nanjing University, in 2011 and 2016, respectively. His research interest mainly focus on verification of hybrid systems.

Wen Xiong received the BSc degree from Nanjing Tech University, in 2014. He is working toward the master's degree from Nanjing Univetsity. His research interest mainly focus on verification of cyber-physical system.



Lei Bu received the BS and PhD degrees in computer science from Nanjing University, in 2004 and 2010, respectively. He is an associate professor in the Department of Computer Science and Technology, Nanjing University. His main research interests include formal method, model checking, especially verification of hybrid system, and cyber-physical system.

Xuandong Li received the MS and PhD degrees from Nanjing University, China, in 1991 and 1994, respectively. He is a full professor in the Computer Science and Technology Department, Nanjing University. His research interests include formal support for design and analysis of reactive, disturbed, real-time, hybrid, and cyber-physical systems, and software testing and verification.