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

Technical Report No. NJU-SEG-2014-IC-002

Deriving Unbounded Proof of Linear Hybrid Automata From Bounded Verification

Xie Dingbao, Bu lei, Li Xuandong

Postprint Version. Originally Published in: IEEE Real-Time Systems Symposium 2014, pp.112-122

Deriving Unbounded Proof of Linear Hybrid Automata From Bounded Verification

Dingbao Xie*, Lei Bu*†, and Xuandong Li*
*State Key Laboratory for Novel Software Technology,
Department of Computer Science and Technology,
Nanjing University, Nanjing, Jiangsu, P.R.China, 210023
Email: xdb@seg.nju.edu.cn,{bulei, lxd}@nju.edu.cn
†Corresponding author

***Abstract*—The behavior space of real time hybrid systems is very complex and hence expensive to conduct the classical full state space model checking. Compared to the classical model checking, bounded model checking (BMC) is much cheaper to conduct and has better scalability. This work presents a technique that can derive, in some cases, a proof of unbounded reachability argument of Linear Hybrid Automata (LHA) from a BMC procedure.**

During BMC of LHA, typical procedures can discover sets of unsatisfiable constraint cores, a.k.a. UC or IIS, in the constraint set according to the bounded continuous state space of LHA. Currently, such unsatisfiable constraints are only fed back to the constraint set to accelerate the BMC solving. In this paper, we propose that such unsatisfiable constraint core can be exploited to give general unbounded verification result of the system model. As each constraint can be mapped back to certain semantical elements of the system model, the unsatisfiable constraint cores can be mapped back into path segments, which are not feasible, in the graph structure of the LHA model. Clearly, if all the potential paths to reach the target location in the graph structure have to go through such infeasible path segments, the target location is not reachable in general, not only in the given bound.

Based on this observation, we propose to encode the infeasible path segments as linear temporal logic (LTL) formulas, and present the graph structure, the discrete part, of the LHA model as a transition system. Then, we can take advantage of the mature off-the-shelf LTL model checking techniques to verify whether there exists a path to reach the target location without touching any detected IIS path segment in the graph structure of the LHA model. We implement this technique into a bounded LHA checker BACH. The experiments show that most of the benchmarks can be verified by the enhanced BACH with a clearly better performance and scalability.

I. Introduction

Hybrid Automata [1] are the classical modeling language for real time hybrid systems with both discrete and continuous state changes. The model checking problem for hybrid automata is considerably difficult. Even for a relatively simple class such as Linear Hybrid Automata (LHA), the reachability problem is undecidable [1], [2]. Classical model checking techniques attempt to compute the whole reachable state space of LHA by the expensive polyhedral computation which is sensitive to the number of continuous variables and not guaranteed to terminate. The state-of-the-art tools based on these techniques such as HyTech [3], PHAVer [4] and PHAVer's new implementation SpaceEx [5] do not scale well to problems of practical interest.

In recent years, bounded model checking (BMC) [6] has been presented as an alternative technique to the classical symbolic model checking. The basic idea of BMC is to search for a counterexample in executions whose lengths are bounded by some integer k [7], [8], 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]. During the solving, techniques like conflict driven clause learning are applied to utilize the unsatisfiable constraint core (UC) of the previous solved constraint set to accelerate the follow-up solving. However, as the whole problem has to be encoded firstly, the object SMT problem will become huge and hence difficult to solve when the system size or the given threshold is large. This greatly restricts the size of the LHA model that can be checked.

Different from the SMT-based BMC approach, a path-oriented approach was proposed to conduct the bounded reachability analysis of LHA [9]. The basic idea of this approach is to check one abstract path in the graph structure of an LHA at a time using Linear Programming (LP) to find whether there exists a feasible continuous behavior of the LHA along this discrete path. As the number of paths in the discrete graph structure of an LHA under a given bounded number of discrete transitions is finite, all the candidate paths can be enumerated and checked one by one to tackle the BMC problem of LHA [10]. Furthermore, when a path is proved to be infeasible by the underlying LP solver, irreducible infeasible set (IIS) technique [12] can be deployed on the constraint set generated according to the path under checking to extract a minimal inconsistent constraint set. Such inconsistent constraint set can be mapped back to an infeasible path segment in the original path. As any path containing an infeasible path segment is definitely infeasible, the infeasible path segments, IIS, found by the LP solver can be utilized to accelerate the BMC process [11], [15].

With the efforts devoted to the BMC verification of LHA in numerous studies [7], [8], [10], the size of the BMC problem that can be verified is increased significantly. However, the result of BMC verification only covers the bounded behavior of the LHA model. It remains a very interesting problem whether we can get an unbounded proof of the system from the BMC result. In finite-state BMC, k-induction [20] is widely used to get an unbounded result using induction-based proof. In study [21], Moura et al. proposed a generalized k-induction schema which can solve infinite-state systems including LHA. Nevertheless, as expensive quantifier-elimination is used in the procedure, the scalability and performance of the generalized k-induction is restricted.

In this paper, we propose a novel method to tackle such problem from a new direction. We propose that we can take advantage of the intermediate result of BMC, the unsatisfiable constraint core, to achieve such target. In the previous mentioned path-oriented bounded reachability analysis of LHA [15], the IIS found by the underlying LP solver is only used to accelerate the BMC solving process. Nevertheless, as each IIS can be mapped back to an infeasible path segment, any path containing this IIS path segment cannot be feasible as well. In other words, once an IIS is found, we can block a path segment in the discrete graph structure of the LHA model. During the experiment, we often 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 detected IIS path segments in the LHA model. In this situation, there does not exist a continuous behavior to reach the target location in general, not only in the bound.

Based on this observation, in this paper, we exploit the IIS path segments to prove whether the bounded unreachable argument can be extended to the unbounded state space. We propose a linear temporal logic (LTL) [17], [18] based approach to tackle such problem. In our approach, we present the discrete part of the LHA model, the graph structure as a finite-state transition system [18]. 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 implies the bounded unreachable statement also stands in general unbounded state space.

Linear temporal logic (LTL) is a logical formalism suited for specifying linear time properties and is widely used to describe system properties. The basic idea of LTL model checking is to firstly construct an equivalent Büchi automata from the LTL formulae [27] and then find a strong connected component containing the accepted state. During the past two decades, the techniques for LTL model checking have made great progress and there are many off-the-shelf LTL model checkers. NuSMV [16] is among the most well-known tools for LTL model checking and is widely applied in verifying practical problems. Recently, IC3 [22] algorithm was proposed to perform SAT-based model checking without unfolding the transition relation. It attracts a lot of attention because the algorithm shows good performance in practical applications. By using certain converting tool, it is viable to verify an LTL model checking problem by IC3 technique, which can benefit for free of all the achievements in IC3 algorithm.

The above LTL-based approach is implemented into a bounded LHA checker BACH [10]. Once an LHA model and a reachability specification are given, BACH starts to perform path-oriented bounded reachability analysis. When the BMC procedure terminates and reports the target is not reachable in the given bound, all the infeasible path segments detected during 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 graph structure of the LHA model. If yes, an unbounded result can be given, otherwise a bounded result is given. We conduct a series of case studies on the enhanced BACH, and compare it with the state-of-the-art classical model checker SpaceEx. The experiment results show that most of the benchmarks can be proved that the reachability specification is not satisfied in general by analyzing the intermediate results of BMC procedure without performing the expensive classical model checking. Furthermore, the experiment also shows a nice performance and scalability of this procedure.

The rest of the paper is organized as follows: Sect. II gives a quick review of the path-oriented bounded reachability verification approach for LHA. After that, we present the main contribution of this paper in Sect. III: deriving an unbounded result from bounded verification using LTL model checking. Sect. IV presents the implementation and performance of our approach. Comparison with the related work is presented in Sect. V. Finally the conclusion and future work are stated in Sect. VI.

II. 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 (IIS) technique to locate infeasible path segments from a path which is proved to be infeasible by the path-oriented checking to accelerate the BMC procedure [15]. Such IIS-based infeasible path segment locating builds the base of our LTL-based unbounded proof in Sect. III.

A. 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 the initial location;
    - $q_{bad} \in Q$ is the 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 $a \le \sum_{i=0}^{l} c_i x_i \le b$ where $x_i \in X$, a, b and c_i are real numbers (a, b may be (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 ($a \le b$). For any location q, for any

$x \in X$, there is one and only one flow condition $\dot{x} \in [a, b] \in \beta(q)$.

- 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 $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$, 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 $\langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$, by assigning each location v_i with a time delay stamp δ_i we get a *timed sequence* of the form $\left\langle \begin{array}{c} v_0 \\ \delta_0 \end{array} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{array}{c} v_1 \\ \delta_1 \end{array} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{array}{c} v_n \\ \delta_n \end{array} \right\rangle$ where δ_i $(0 < i \leq n)$ is a nonnegative real number and $\delta_0 = 0$ as $v_0 = q_0$ is the 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 the 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 $\left\langle \begin{array}{c} v_0 \\ \delta_0 \end{array} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{array}{c} v_1 \\ \delta_1 \end{array} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{array}{c} v_n \\ \delta_n \end{array} \right\rangle$, 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 $\left\langle \begin{array}{c} v_0 \\ \delta_0 \end{array} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{array}{c} v_1 \\ \delta_1 \end{array} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{array}{c} v_n \\ \delta_n \end{array} \right\rangle$ 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;
- 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 $\left\langle \begin{array}{c} v_0 \\ \delta_0 \end{array} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{array}{c} v_1 \\ \delta_1 \end{array} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{array}{c} v_n \\ \delta_n \end{array} \right\rangle$ is a behavior of H, we say path $\rho = \langle v_0 \rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \langle v_1 \rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \langle v_n \rangle$ 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 $\left\langle \begin{array}{c} v_0 \\ \delta_0 \end{array} \right\rangle \xrightarrow[\sigma_0]{(\phi_0,\psi_0)} \left\langle \begin{array}{c} v_1 \\ \delta_1 \end{array} \right\rangle \xrightarrow[\sigma_1]{(\phi_1,\psi_1)} \ldots \xrightarrow[\sigma_{n-1}]{(\phi_{n-1},\psi_{n-1})} \left\langle \begin{array}{c} v_n \\ \delta_n \end{array} \right\rangle$ *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 δ_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.

As we all know, the basic idea of bounded model checking (BMC) 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.

B. IIS based BMC Acceleration

Last paragraph gives a simple solution of 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 (IIS) [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.

Intuitively speaking, the 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 [29] and LINDO [30].

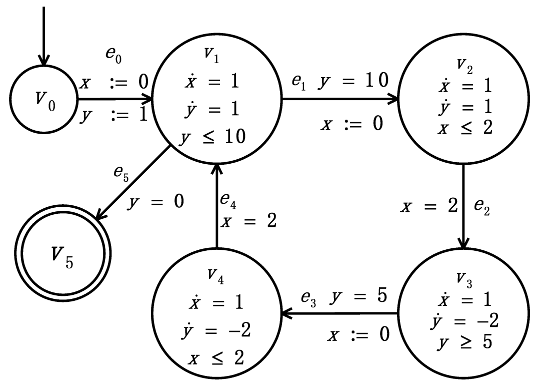

Fig. 1: Water-Level Monitor System

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 $\langle v_3 \rangle \xrightarrow[e_3]{} \langle v_4 \rangle \xrightarrow[e_4]{} \langle v_1 \rangle \xrightarrow[e_5]{} \langle v_5 \rangle$.

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 which 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 [15]. 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 $\rho_3 = \langle v_0 \rangle \xrightarrow[e_0]{} \langle v_1 \rangle \xrightarrow[e_1]{} \langle v_2 \rangle \xrightarrow[e_2]{} \langle v_3 \rangle \xrightarrow[e_3]{} \langle v_4 \rangle \xrightarrow[e_4]{} \langle v_1 \rangle \xrightarrow[e_1]{}$

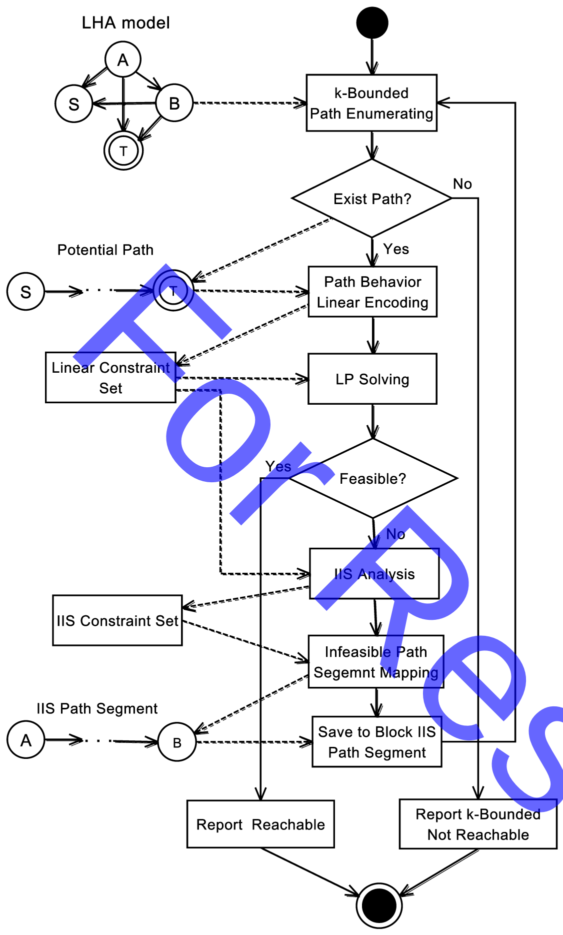

Fig. 2: Workflow of Path-oriented Bounded Analysis for LHA

$\langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_1\rangle \xrightarrow[e_5]{} \langle v_5\rangle$. 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.

III. Deriving Unbounded Result From BMC Solving Process

A. Motivation Overview

The last section gives a quick review of the path-oriented bounded reachability analysis of LHA [15]. The basic idea is to find an abstract path, sequence of locations, in the discrete graph structure of the 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 used 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 the experiment, we often see that the 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 in general, 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.

Under this observation, if we can prove that there does not exist any path to reach the target location without going through certain path segments in the graph structure of the LHA model, we can tell that the reachability specification is not satisfied in general. 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 there exists a path from the initial location to the target location in the directed graph?

B. LTL Based IIS Representation and Verification

The problem summarized 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 the reachability verification on G, we propose to extend G into a typical transition system (TS) [18] T firstly.

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 the (labeled) location graph of H.
- 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.

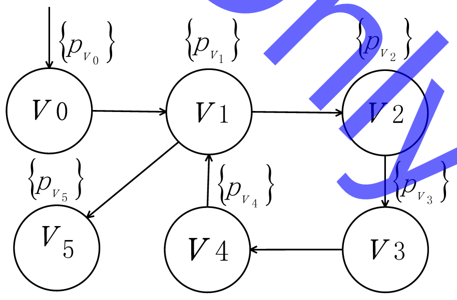

Fig. 3: The DTS Model of The Water-Level Monitor System

It is well known that linear temporal logic (LTL) [17] is a powerful temporal logic of describing system behaviors.

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

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 $\rho = \langle v_0 \rangle \to \langle v_1 \rangle \to \ldots \to \langle v_n \rangle$ in an LHA H, and ρ contains a path segment $\rho' = \langle v_i \rangle \to \langle v_{i+1} \rangle \to \ldots \to \langle v_j \rangle (i \geq 0 \land j \leq n)$. 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 Def. 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 $p_{v_i} \& X\ p_{v_{i+1}} \& \ldots \& \underbrace{X\ X\ \ldots X}_{j-i}\ p_{v_j}$. Based on this, given an IIS path segment $\rho' = \langle v_i \rangle \to \langle v_{i+1} \rangle \to \ldots \to \langle v_j \rangle$, we give the LTL formula representing this path segment as:

$$IIS_{\rho'} = p_{v_i} \& X\ p_{v_{i+1}} \& \ldots \& \underbrace{X\ X\ \ldots X}_{j-i}\ p_{v_j} \qquad (1)$$

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

Reaching the Target Location without Containing any IIS Path Segment. The property that the 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 path reaching the target location without containing any above IIS path segment is shown below:

$$(G(\bigwedge_{1 \leq i \leq n} \neg IIS_{\rho_i})) \land F\ p_{q_{bad}} \qquad (2)$$

where IIS_{ρ_i} represents the i-th 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:

$$\neg((G(\bigwedge_{1 \leq i \leq n} \neg IIS_{\rho_i})) \land F\ p_{q_{bad}}) \qquad (3)$$

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 $\neg((G(\bigwedge_{1 \leq i \leq n} \neg IIS_{\rho_i})) \land F\ p_{q_{bad}})$, then q_{bad} is not reachable in H in general.

Proof: Suppose not. [we take the negation of the given statement and suppose it to be true.] Assume T satisfies the LTL specification and q_{bad} is reachable in H along a path $\rho = \langle v_0 \rangle \to \langle v_1 \rangle \to \ldots \to \langle v_n \rangle (v_n = q_{bad})$. 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 $(G(\bigwedge_{1 \leq i \leq n} \neg IIS_{\rho_i})) \land F\ p_{q_{bad}}$ is true for ρ. As we can see, ρ is the counterexample of the given LTL specification $\neg((G(\bigwedge_{1 \leq i \leq n} \neg IIS_{\rho_i})) \land F\ p_{q_{bad}})$. 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 Sect. II, it has two IIS path segments: $\langle v_0 \rangle \to \langle v_1 \rangle \to \langle v_5 \rangle$, and $\langle v_3 \rangle \to \langle v_4 \rangle \to \langle v_1 \rangle \to \langle v_5 \rangle$ and the target location is v_5. According to the above encoding method, the associated LTL specification is:

$$\neg((G(\neg(p_{v_0} \& X\ p_{v_1} \& X\ X\ p_{v_5}) \land \neg(p_{v_3} \& X\ p_{v_4} \& X\ X\ p_{v_1} \land X\ X\ X\ p_{v_5}))) \land F\ p_{v_5}) \qquad (4)$$

The model checking of LTL property on a transition system is well-studied during the last two decades and there are various efficient tools such as Spin [19], NuSMV [16] to tackle this problem. Therefore, we can take advantage of these off-the-shelf LTL checkers to prove the reachability of certain target locations in the discrete graph structure efficiently. Furthermore, with the help of the converting tool "smvtoaiger" [25], we can formulate the LTL model checking problem with AIGER [25] standard which is widely used in hardware verification. Then, we can also solve the problem efficiently with tools based on IC3 technique, such as iimc [24].

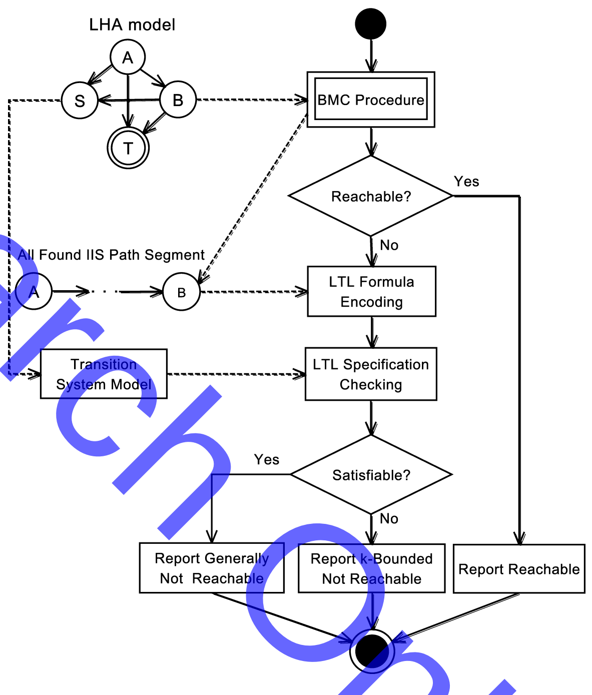

Fig. 4: Workflow of Unbounded Proof

Workflow of the LTL-based Proof. Now, by performing the LTL verification on the discrete transition graph of the 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 in general. This provides a procedure that allows us to derive, in some cases a proof of unbounded result from a BMC solving process.

Given an LHA and a bound k, the workflow of this LTL-based solution is shown in Fig. 4. The path-oriented reachability analysis is conducted firstly and all detected IIS path segments are saved during the BMC solving process, marked as "BMC Procedure" by double square. The detailed flow of “BMC Procedure” is shown in Fig. 2 previously. When the bounded analysis is completed, if a feasible path is found then it is reported as a witness which can confirm the target location is reachable. Otherwise, all IIS path segments found in the BMC procedure are encoded into an LTL specification. After that, an LTL model checker is called to conduct the unbounded proof by checking whether the specification is true on the graph structure of the LHA. If yes, then the target location is not reachable in general, otherwise a k-bounded unreachable conclusion is reported.

IV. Implementation and Case Studies

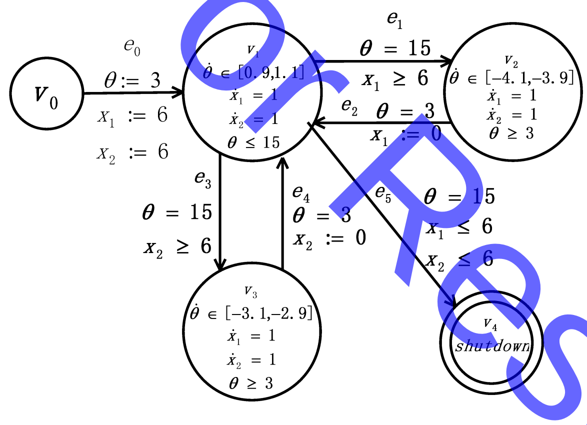

Fig. 5: Temperature Control System

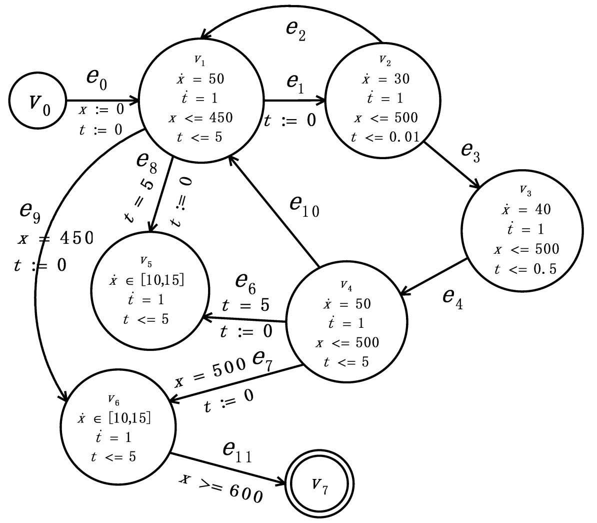

Fig. 6: Train Control System

The LTL-based verification procedure presented in this paper is implemented into a path-oriented bounded reachability checker of LHA - BACH [10]. The first underlying LTL model checker selected is a typical LTL model checker NuSMV [16]. In addition, as mentioned in the last section, we also implement the IC3 based approach into BACH and the underlying IC3 based tool we selected is iimc [24] which supports analysis of liveness properties in LTL specification. In the following paragraph, the experimental data given by the two different implementation of BACH are marked as BACH (NuSMV) and BACH (IC3), respectively.

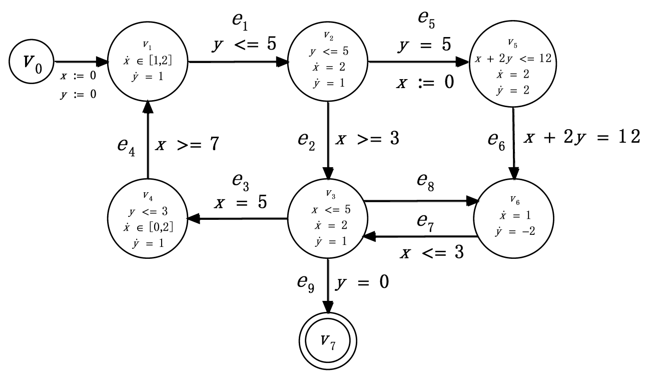

Fig. 7: Sample Automaton

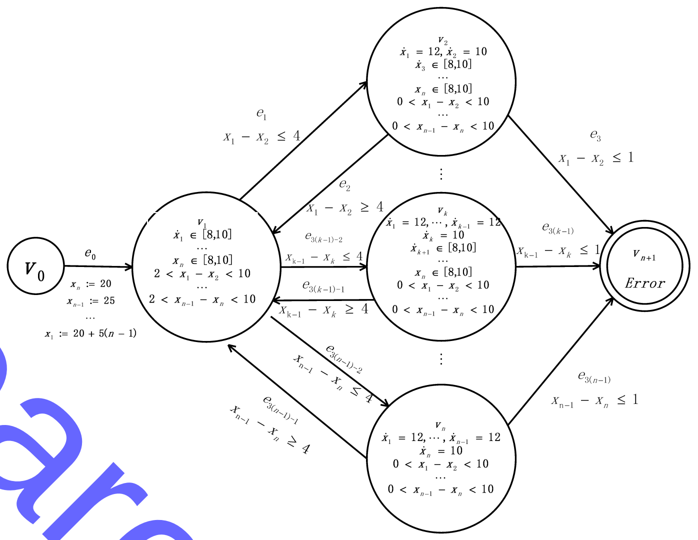

Fig. 8: Automated Highway System

In order to evaluate the performance of the technique presented, we carry out an extensive evaluation, over a set of widely-used benchmarks, which include the aforementioned water-level monitor system in Fig. 1, the temperature control system in Fig. 5, the communication based train control system in Fig. 6, and sample automata from [15] in Fig. 7.

Besides of the above small scale models, we also conducted the case studies on the scalable automated highway system from [13] in Fig. 8. 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. For example, Fig. 9 and Fig. 10 are the models for the automated highway system with three cars and four cars respectively. The target locations, q_{bad} are all marked by double circles in the models and they are all unreachable[2].

As the performance of the path-oriented bounded reachability analysis approach has already been compared with other BMC competitors in study [15] intensively, in this paper, we focus on the comparison of the performance of the enhanced BACH, with the state-of-the-art classical LHA model checker

[2]Due to the space limit, please refer to [15] for the detail of these automata.

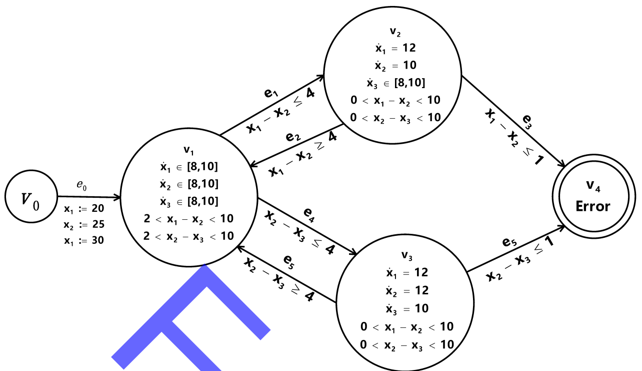

Fig. 9: Automated Highway System with three cars

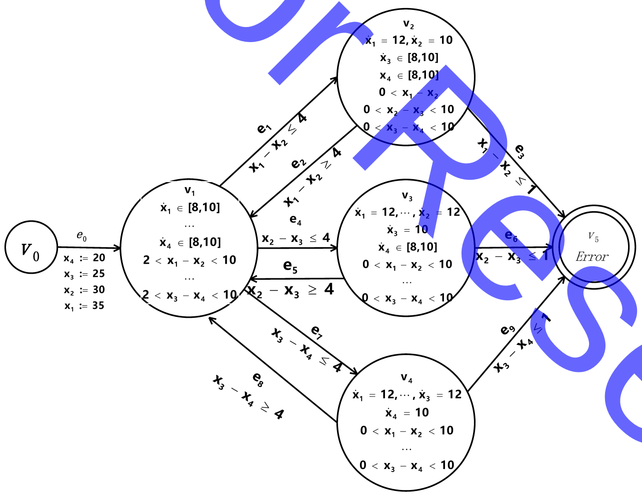

Fig. 10: Automated Highway System with four cars

SpaceEx [5]. According to the documentations, 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, while the later adopts support function [28] 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 experiments are conducted on a ThinkCenter workstation (Intel i5 Quad CPU $3.1\text{GHz}, 8\text{GB}$ RAM and Ubuntu 13 64Bit). The time and memory usage limit for the experiment are set to one hour and 4GB, respectively. The number of 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/rtss14.

The experiment data for the time and memory usage spent in each benchmark is shown in Table I. We show the time and memory usage BACH and SpaceEx spend for each problem. In addition, for BACH, the number of IIS path segments detected by the underlying LP solver is also given. Furthermore, it is worth mentioning that the time spent by BACH means the total time of the BMC and the follow-up unbounded proof procedure.

From these data, we can see that:

- BACH successfully proved the bounded unreachable results also stand in general for most of the benchmarks, 8/9, except the sample automaton, by the LTL based approach. This increases our confidence that many practical cases can be proved to be unreachable in general due to the existence of IIS path segments. In this situation, BACH can give an unbounded result while other BMC checkers are incapable of doing it. BACH only fails to give an unbounded result for the sample automaton. The reason is that in this case 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 HA with piecewise affine dynamics, for example model with flow condition like $\dot{x} = Ax + Bu + c$. For the class of LHA considered in this paper, SpaceEx prefers to use the "PHAVer" scenario, SpaceEx (PHA.), to handle. As we can see from Table I, 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 solve them quickly in less than 1 second. We mark the quickest result by bold font. We can see that SpaceEx (PHA.) wins in the water and sample automaton, while BACH outperforms on the other three cases.
  - On the other hand, as the reachability analysis of LHA is undecidable, classical model checking technique do not guarantee to terminate. The experiments show 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 cannot terminate. In contrast, BACH is guaranteed to terminate.
  - On large cases, BACH has better scalability. Take the automated highway system for example, BACH can solve such a system with 200 cars, 202 locations and 200 continuous variables, in less than 70s, while SpaceEx can only handle the system with 5 cars within the 1h time limit. As the polyhedra based computation is sensitive to the number of continuous variables in the LHA, the performance of SpaceEx degrades badly when the size of

TABLE I: Results of applying BACH and SpaceEx to different LHA cases. (#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 3600 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. 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 benchmarks are unreachable in general.)

System	#locs	#vars	BACH (NuSMV)			BACH (IC3)		SpaceEx (PHA.)		SpaceEx (Supp.)	
			#IIS	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)	Time (s)	Mem. (MB)
water	6	2	2	0.94_U	<1	0.87_U	30.4	$\mathbf{0.07_U}$	<1	0.22_U	7.9
tcs	5	3	4	$\mathbf{0.97_U}$	<1	0.98_U	16.4	T.O.	-	0.36_U	9.4
sample	8	2	9	0.96_B	26.8	0.41_B	21.2	$\mathbf{0.93_U}$	<1	EXC	-
train	8	2	2	1.02_U	<1	$\mathbf{0.3_U}$	<1	0.62_U	<1	1.24_U	24.8
motorcade_5	7	5	4	$\mathbf{0.05_U}$	<1	0.4_U	<1	4.94_U	16	T.O.	-
motorcade_10	12	10	9	$\mathbf{0.12_U}$	<1	0.6_U	16.9	T.O.	-	T.O.	-
motorcade_20	22	20	19	$\mathbf{0.53_U}$	60.8	1.1_U	25.4	T.O.	-	T.O.	-
motorcade_100	102	100	99	$\mathbf{6.66_U}$	163.9	15.7_U	389	T.O.	-	T.O.	-
motorcade_200	202	200	199	$\mathbf{61.8_U}$	652.7	115.3_U	3299	T.O.	-	T.O.	-

the model increases. Different from SpaceEx, BACH conducts lightweight 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.

In summary, on one hand, compared to BMC checkers, our approach can give an unbounded result in many cases. On the other hand, in comparison to unbounded LHA checkers, our approach is efficient and scalable and guaranteed to terminate.

V. Related Work

Reachability analysis of linear hybrid automata (LHA) is a very important problem. Classical model checking techniques [4], [5] try to compute the whole reachable state space of LHA. Their basic idea is to compute iteratively the next-step reachable state space based on the current state space. The algorithm terminates if the next-step reachable state space 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 based on this technique do not scale well to the size of practical problems.

Recently, bounded model checking (BMC) [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 get an unbounded proof from the BMC result.

In study [26], 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. However, as the interpolants are typically highly redundant, it is very difficult to apply this technique to reachability analysis of LHA.

Sheeran et al. proposed an induction based approach called k-induction [20] 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 not reachable in general. In [21], 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 simulation relations. As the implementation of [21] is not available, we cannot compare the performance between our approach and [21] numerically. However, as the unbounded analysis of [21] 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. Furthermore, quantifier-elimination in that work might not always be available, which means the approach may fail to give a result.

In study [23], 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 with 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 translation algorithm [27]. Then the refinement will be done by computing the Cartesian product of the ω-automata and the abstract model. If the production produces an empty automata, then the original model is safe. Different from our work, this work is about the CEGAR verification of HA and it only focuses on step-discrete linear hybrid models which have no continuous dynamics. Instead, our work focuses on general LHA. On the other hand, the main contribution of the work is an efficient algorithm to construct an minimal ω-automata on-the-fly, which can be used in our future work to increase the efficiency of our method.

There are also works focusing on CEGAR of LHA. Study [31] presents 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 [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. As this technique still relies on PHAVer, in another word, geometric computation, as the underlying checker for the abstracted model, when the abstracted model is large, it may still unable to handle it.

VI. Conclusion and Future Work

Compared to classical full state space reachability verification of real time hybrid systems, specifically Linear Hybrid Automata considered in this paper, bounded verification is much more convenient and efficient to conduct. However, how to derive an unbounded argument of the LHA from the bounded analysis remains a very interesting problem. In this paper, we propose an LTL-based solution to tackle such problem.

We propose to collect the unsatisfiable constraint cores discovered during BMC solving and map them back to infeasible path segments in the discrete graph structure of the LHA model. Then, we can present and encode such path segments as an LTL formula. Afterwards, we propose to conduct LTL model checking on the discrete graph structure of LHA model to prove that the model does not have any potential path to reach the target location without going through any of the infeasible path segments. If the LTL proof stands, it means the bounded unreachable result stands in the general unbounded state space.

We implement this LTL-based solution into a bounded LHA checker BACH. The experiments show that by this approach, BACH can give an unbounded result for most of the benchmarks successfully, efficiently and also with a better scalability.

For the future work, in the current setting, the unbounded proof procedure is conducted after the BMC completes. Then, all the IISes founded during the BMC procedure are considered together once for all. In the future, we will implement an incremental LTL model checking algorithm and perform the LTL checking on-the-fly parallel with the main BMC procedure once an IIS is detected.

Acknowledgment

The authors would like to thank Dr. Pascal Fontaine for insightful discussions and the anonymous reviewers for valuable advices on improving this paper. This work is supported by the National 863 High-Tech Program of China (No.2012AA011205), the National Natural Science Foundation of China (No.61100036, No.91318301, No.61321491) and by the Jiangsu Province Research Foundation (No.BK2011558).

References

[1] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of LICS 1996*, IEEE Computer Society Press, 1996, pp. 278-292.

[2] T. A. Henzinger, Peter W. Kopke, A. Puri, and P. Varaiya. What's Decidable About Hybrid Automata? In *Journal of Computer and System Sciences*, 57: 94-124, 1998.

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

[4] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. In *HSCC (2005)*, LNCS 2289, pp. 258-273.

[5] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *CAV (2011)*, pp. 379-395.

[6] A. Biere *et al.* . Bounded Model Checking. In *Advance in Computers*, Vol.58, Academic Press, 2003, pp. 118-149

[7] G. Audemard, M. Bozzano, A. Cimatti and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In *2nd International Workshop on Bounded Model Checking (BMC2004)*, ENTCS, 119:2, Elsevier Science, 2005, pp. 17-32.

[8] L. de Moura, N. Bj∅rner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS2008)*, LNCS, Vol. 4963, 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. In *Fourth International Workshop on Bounded Model Checking (BMC06)*, ENTCS, 174:3, Elsevier Science, 2007, pp. 57-70.

[10] L. Bu, Y. Li, L. Wang and X. Li. BACH: Bounded Reachability Checker for Linear Hybrid Automata. In *Proceedings of the 8th International Conference on Formal Methods in Computer Aided Design*, IEEE Computer Society, pp. 65-68,2008.

[11] L. Bu, Y. Yang and X. Li. IIS-Guided DFS for Efficient Bounded Reachability Analysis of Linear Hybrid Automata, In *HVC (2012)*. Volume 7261, 2012, pp. 35-49

[12] J. Chinneck *et al.* . Locating Minimal Infeasible Constraint sets in linear programs. In *ORSA Journal on Computing*, 3(1991), pp. 157-168.

[13] S. Jha, B. H. Krogh, J. E. Weimer and E. M. Clarke. Reachability for Linear Hybrid Automata Using Iterative Relaxation Abstraction. In *HSCC (2007)*, pp. 287-300.

[14] E. Clarke, *et al.*. Counterexample-Guided Abstraction Refinement. *In CAV 2000*. LNCS 1855, pp. 154-169. Springer, Heidelberg (2000).

[15] D. Xie, L. Bu and X. Li. SAT-LP-IIS Joint-Directed Path-Oriented Bounded Reachability Analysis of Linear Hybrid Automata. In *Formal Methods in System Design*. 45(1): 42-62, 2014.

[16] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *CAV (2002)* pp. 259-364, 2002.

[17] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science* pp. 46-57, 1977.

[18] C. Baier and J. Katoen. Principles of Model Checking. The MIT Press.

[19] G. J. Holzmann. The Model Checker SPIN. In *IEEE TRANSASCIONS ON SOFTWARE ENGINEERING* vol. 23, 1997, pp. 279-295.

[20] M. Sheeran, S. Singh, and G. Stalmarck. Checking safety Properties Using induction and a SAT solver. In *FMCAD (2000)* pp. 108-125.

[21] L. de Moura, H. Rueß and M. Sorea. Bounded Model Checking and Induction: From Refutation to Verification. In *CAV (2003)* vol. 2725, 2003, pp. 14-26.

[22] A. R. Bradley. SAT-Based Model Checking without Unrolling. In *VMCAI (2011)* pp. 70-87.

[23] M. Segelken. Abstraction and Counterexample-Guided Construction of ω-Automata for Model Checking of Step-Discrete Linear Hybrid Models. In *CAV (2007)*, pp. 433-448.

[24] iimc. http://ecee.colorado.edu/wpmu/iimc/.

[25] AIGER. http://fmv.jku.at/aiger/.

[26] K.L. McMillan. Interpolation and SAT-based model checking. IN *CAV 2003* pp. 1-13.

[27] F. Somenzi and R. Bloem. Efficient Büchi automata from LTL formulae. In *CAV (2000)* pp. 248-263.

[28] C. Le Guernic A. Girard. Reachability analysis of linear systems using support functions. In *Nonlinear Analysis: Hybrid Systems*, 2010, 4(2), pp. 250-262.

[29] CPLEX. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

[30] Systems Inc., L.: http://www.lindo.com/products/api/dllm.html

[31] P. Prabhakar *et al.* Hybrid Automata-Based CEGAR for Rectangular Hybrid Systems. In *VMCAI (2013)*. pp. 48-67, 2013.