Technical Report No. NJU-SEG-2008-IC-003

BACH : Bounded ReachAbility CHecker for Linear Hybrid Automata

Lei Bu, You Li, Linzhang Wang, Xuandong Li


Most of the papers available from this document appear in print, and the corresponding copyright is held by the publisher. While the papers can be used for personal use, redistribution or reprinting for commercial purposes is prohibited.
**BACH : Bounded ReachAbility CHecker for Linear Hybrid Automata**

Lei Bu, You Li, Linzhang Wang, Xuandong Li

State Key Laboratory for Novel Software Technology, Nanjing, Jiangsu, P.R.China, 210093

Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu, P.R.China, 210093

Email: bl@seg.njtu.edu.cn, lxd@nju.edu.cn

**Abstract**—Hybrid automata are well studied formal models for hybrid systems with both discrete and continuous state changes. However, the analysis of hybrid automata is quite difficult. Even for the simple class of linear hybrid automata, the reachability problem is undecidable. In the author’s previous work, for linear hybrid automata we proposed a linear programming based approach to check one path at a time while the length of the path and the size of the automaton being checked can be large enough to handle problems of practical interest. Based on this approach, in this paper we present a prototype tool BACH to perform bounded reachability checking of linear hybrid automata. The experiment data shows that BACH has good performance and scalability, and supports our belief that BACH could become a powerful assistant to design engineers for the reachability analysis of linear hybrid automata.

**I. Introduction**

Hybrid automata [1] are well studied formal models for hybrid systems with both discrete and continuous state changes. However, the analysis of hybrid automata is very difficult. Even for the simple class of linear hybrid automata (LHA), the reachability problem is undecidable [1] [2] [3]. Several model checking tools have been developed for LHA, but they do not scale well to the size of practical problems. The state-of-the-art tool HYTECH [9] and its improvement PHA Ver [10] need to perform expensive polyhedra computation, and their algorithm complexity is exponential in number of variables in the automata.

In recent years, bounded model checking (denoted as BMC) [4] has been presented as an alternative technique for BDD-based symbolic model checking, whose basic idea is to encode the next-state relation of a system as a propositional formula, and unroll this formula to some integer $k$, using SAT idea to search for a counterexample in the model executions whose length is bounded by $k$. As extensions to BMC, there are several related works [5] [6] using linear programming technique with SAT to check linear hybrid systems. In these works, the bounded model checking problems are reduced to the satisfiability problem of a boolean combination of propositional variables and linear mathematical constraints. Based on these works, several tools were proposed accordingly, such as HySAT [5], MathSAT [6]. All of these solvers are built based on a SAT-solver that calls a linear program solver for conjunctions of the linear continuous-part constraints. As this technique requires to encode the whole problem space firstly, when the system size or the given step threshold is large, the object problem will be very huge, which greatly restricts the size of the problem that can be solved.

In our previous study [7], for LHA we proposed an linear programming based approach to develop an efficient path-oriented tool to check one path at a time while the length of the path and the size of the automaton being checked can be large enough to handle problems of practical interest. Based on this approach, in this paper we present a prototype tool BACH to do bounded reachability checking of linear hybrid automata. The main function of BACH includes

- **Path-Oriented Reachability Analysis:** to check a specific path in a LHA using linear programming;
- **“Full” Bounded Reachability Analysis:** to check all the paths in a LHA in a given step threshold of the system by traversing in depth first search (DFS).

We have conducted some case studies using BACH, and the experiment data shows that:

- for the path-oriented reachability analysis, the length of the path being checked can be made especially large and the size of the automaton can be made large enough to handle problems of practical interest; and
- for the “full” bounded reachability analysis, BACH has much better performance and scalability than the SAT-style solver HySAT.

The rest of the paper is organized as follows: Section II gives a simple description of the underlying techniques of BACH. Section III depicts the main functionality of BACH. Section IV describes several case studies to show the processing ability of BACH. Finally the conclusion is stated in Section V.

**II. The Underlying Techniques**

Here we use one simple case scenario to illustrate the main underlying techniques of BACH. The automaton in Fig. 1 describes a model of water level monitor cited from [3]. The water level is controlled through a monitor, which continuously senses the water level and turns a pump on and off. When the pump is off the water level falls by 2 per second, and when the pump is on the water level rises by 1 per second. And there is a delay of 2 seconds between the time monitor signals to change the status of the pump and the time that the change becomes effective. When the system is in location $v_1$, if the monitor finds the water level is 0, the system will jump to location $v_3$ and stop.
According to the above steps, the problem of checking if the linear inequality set is defined as follows:

Two optimization techniques in [7], path decomposition programming problem with 43 constraints and 12 variables. In order to illustrate our method succinctly without oriented rechability analysis approach given in our previous study [7]. In order to illustrate our method succinctly without oriented rechability analysis approach given in our previous study [7]. The primary decision procedure under BACH is the path-oriented reachability analysis approach given in our previous study [7]. In order to illustrate our method succinctly without oriented rechability analysis approach given in our previous study [7].

A. Path-Oriented Reachability Analysis

The main decision procedure under BACH is the path-oriented reachability analysis approach given in our previous study [7]. In order to illustrate our method succinctly without loss of generality, the path we choose is a traversal of each transition for exactly once, that is \( v_0 v_1 v_2 v_3 v_4 v_5 \). The problem we concern is to check the reachability of location \( v_5 \) along this path, which can be reduced to a linear program whose linear inequality set is defined as follows:

1) We represent the timed behavior of the path by the form \( (v_0, t_0)(v_1, t_1)(v_2, t_2)(v_3, t_3)(v_4, t_4)(v_5, t_5) \) where \( t_i \) stands for the time spent in the \( i \)-th location in the path. Thus, we will generate 6 variables \( t_i \) (\( 1 \leq i \leq 6 \)), and 6 corresponding constraints in the linear program \( t_i \geq 0 \). As initial location fires its transition immediately, the time spent in it is set to 0, which means \( t_0 = 0 \), so we don’t need to generate variable \( t_0 \).

2) For each location invariant and transition guard in the path, we will generate the corresponding linear constraints, e.g., for \( y \leq 10 \) in location \( v_1 \) and \( y = 10 \) in transition \( e_1 \), as the change rate of \( y \) in \( v_1 \) is 1, we can generate four constraints: \( y_{v_{1_{in}}} \leq 10 \), \( y_{v_{1_{end}}} \leq 10 \), \( y_{v_{1_{in}}} + t_1 = y_{v_{1_{end}}} \), and \( y_{v_{1_{end}}} = 10 \). For \( y_{v_{1_{in}}} \) stands for the value of \( y \) when the automaton comes into location \( v_1 \), and \( y_{v_{1_{end}}} \) stands for the value of \( y \) when the automaton is going to jump to \( v_2 \) after having stayed at location \( v_1 \) for time \( t_1 \).

3) For each reset action in the path, we will also generate the corresponding constraints, e.g., for \( y := 1 \) in the transition \( e_0 \), we get \( y_{v_{1_{in}}} = 1 \).

According to the above steps, the problem of checking if \( v_5 \) is reachable along the path will be reduced into an linear programming problem with 43 constraints and 12 variables.

Besides of the above original technique, we also develop two optimization techniques in [7], path decomposition and path shortening, to reduce the size of the resulting linear program so that the tool can be used to solve problems of size as large as possible. The path decomposition technique is to decompose the linear program corresponding to the checked path into several separate smaller linear programs so that the tool can check longer paths. Since the size of the linear program corresponding to the checked path is proportional to the length of the path, the path shortening technique is to find a shorter path in lieu of the checked one such that both of them are equivalent with respect to the given reachability specification - if one of them satisfies the reachability specification, so does the other.

B. “Full” Bounded Reachability Analysis

The bounded reachability analysis is to look for a system trajectory in a given length threshold which can reach a given target. As we can already check the reachability of one single path, we can traverse the system structure directly in DFS and check all the potential paths one by one until a feasible path to the reachability target is found or the length threshold is reached. Instead of encoding the whole problem space to a group of formulas like SAT-style solver, which will suffer a lot when dealing with big problems, this plain DFS style approach only needs to keep the discrete structure and current visiting path in memory, and check each potential path one by one, which makes it possible to solve big problems as long as enough time is given.

We have also devised some optimizations for decreasing the number of paths that need to be checked using linear programming. In general, BACH checks each path \( p \) in a given length threshold for the reachability by solving the corresponding linear program. However, sometimes there is no target location in \( p \) at all so that we can simply falsify \( p \) for the reachability. For example, when we check whether location \( v_5 \) is reachable in Fig.1, the path \( v_1 v_2 v_3 v_4 v_5 \) can simply be falsified for the reachability. With this simple optimization technique, BACH only needs to check the reachability for the paths reaching the target location. Furthermore, as BACH does the bounded reachability analysis by a DFS based traversing of paths, once we find that the current checked path is infeasible (no timed behavior corresponding to the path) we should backtrack. We can take advantage of the irreducible infeasible subset (IIS) technique [11] to find the constraints which result in no solution of the linear program, which can help us to return to the most reasonable branch location for starting new search.

Thanks to the advancements in computing during the past decade, linear programs with several hundreds variables and constraints are considered “small” nowadays, and the ones with tens or hundreds of thousand of variables can be solved very efficiently, which makes it possible for BACH to check considerable number of long paths within tolerable time.

III. Functionality of BACH

The Bounded reachability Checker of LHA (BACH) is implemented in Java, and the latest version of BACH can be downloaded from (http://seg.nju.edu.cn/BACH/BACH.html). Several parts of GUI are shown in Fig.2 with the system structure shown in Fig.3 as well. The linear programming software package integrated in BACH is from OR-Objects of DRA Systems [8] which is a free collection of Java classes for developing operations research, scientific and engineering applications.
BACH is composed of two main parts as follows:

- **Graphical LHA Editor**: It allows users to construct, edit, and perform syntax analysis of LHA interactively. This Editor can also save the graphical representation of LHA to a human readable text file which is used as the input file for reachability checker.

- **Bounded Reachability Checker**: It accepts a LHA file generated by the Editor, and perform the bounded reachability checking. BACH supports two kinds of reachability checking:
  - **Path-Oriented Reachability Checking**: The checker requires users to select a specific path in a LHA, and use linear programming to check the path for reachability.
  - **“Full” Bounded Reachability Checking**: The checker requires users to input the bounded threshold for a LHA, traverses all the paths in the bounded threshold by DFS, and checks the related paths for reachability using linear programming. Once the reachability specification is satisfied, a witness will be given also by the checker.

For the water level monitor whose model is depicted in Fig.1, we want to check whether the tank will be empty in the future (the location $v_5$ is reachable). For the temperature control system whose model is depicted in Fig.4, which describes a cooling system with two control rods, we plan to check whether a complete shutdown is required after the system has run for a while (the location $v_4$ is reachable). We set up the parameters such that both of the locations are not reachable in the models.

### Table I
**Path-oriented checking results on the water-level monitor**

<table>
<thead>
<tr>
<th>$k$</th>
<th>Original technique</th>
<th>Decomposing</th>
<th>Shortening</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>constraints</td>
<td>variables</td>
<td>time (s)</td>
</tr>
<tr>
<td>200</td>
<td>2007</td>
<td>2002</td>
<td>503.140</td>
</tr>
<tr>
<td>400</td>
<td>14407</td>
<td>4002</td>
<td>4202.421s</td>
</tr>
<tr>
<td>500</td>
<td>Java.lang.out of memory error</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td>Java.lang.out of memory error</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table II
**Path-oriented checking results on the temperature control system**

<table>
<thead>
<tr>
<th>$k$</th>
<th>Original technique</th>
<th>Decomposing</th>
<th>Shortening</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>constraints</td>
<td>variables</td>
<td>time (s)</td>
</tr>
<tr>
<td>200</td>
<td>8815</td>
<td>2004</td>
<td>686.938s</td>
</tr>
<tr>
<td>400</td>
<td>17591</td>
<td>3998</td>
<td>5574.312s</td>
</tr>
<tr>
<td>450</td>
<td>Java.lang.out of memory error</td>
<td>Java.lang.out of memory error</td>
<td></td>
</tr>
</tbody>
</table>

The experimental data in Table I and Table II illustrate the processing ability of the Path-Oriented Checking in BACH. It is shown that the length of the single path analyzed by BACH without optimization can be as long as 4000 (1000 * 4) steps. Furthermore, if the optimization techniques are applicable which split a big linear program into several independent small ones, the problem size solved by BACH can be extremely large. For example with the path decomposition technique, BACH can analyse paths of length up to 40000 (10000 * 4) in only 0.031 seconds, as shown in Table I. Actually the inherent reason for such good performance and scalability is that the size of linear program solved in the Path-Oriented Checking is linear in the path length and the variable number in the model.

We have also evaluated the the processing ability of the “Full” Bounded Reachability Checking (BRC) in BACH using the same examples, and conduct a comparison with the SAT-style solver HySAT [5]. The experiment data are shown in
The above opinions contain some intuition, and the more deep reason that BACH works better than HySAT merits further investigation.

We also attempt to compare BACH with another SAT-style solver MathSAT [6]. But because from [6] we cannot get the details of encoding LHA with MathSAT language, we have to leave the comparison in the future work when we gain a mastery of encoding LHA in MathSAT.

V. CONCLUSION

In this paper, we present a bounded reachability checker BACH for linear hybrid automata. BACH provides a convenient GUI to construct LHA model, a powerful path-oriented reachability checker to analyze one single LHA path, and a “full” bounded reachability checker to do reachability checking in a given threshold. The experiment data shows that BACH has good performance and scalability, and supports our belief that BACH could become a powerful assistant to design engineers for the reachability analysis of linear hybrid automata.

ACKNOWLEDGMENT

We would like to thank Prof. Tom Melham and the anonymous reviewers for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (No.60425204 and No.60721002), the National 863 High-Tech Programme of China (No.2007AA010302), and by the Jiangsu Province Research Foundation (No.BK2007714).

REFERENCES