Forward and Backward : Bounded Model Checking of Linear Hybrid Automata From Two Directions

Yang Yang*, Lei Bu*†, and Xuandong Li*

***Abstract*—Instead of encoding the bounded state space of a linear hybrid automaton(LHA) in given threshold k into SMT formulas then solving them by SMT solvers, the authors proposed a different approach to handle the bounded reachability verification(BMC) of LHA in the previous work. First, the reachability specification along one abstract path in LHA can be checked by linear programming (LP). Then, all the abstract paths under the threshold can be checked one by one by depth-first-search (DFS) traversing. This approach was implemented in a prototype tool BACH, and the experiment result shows it is efficient.**

As BACH uses DFS to traverse the bounded state space, clearly, if DFS traverses more quickly, the BMC can be finished more efficiently. Nevertheless, in many cases, the path segments which make the system infeasible are hidden "deeply" in the model or have many entry points which makes the DFS difficult to find them or has to traverse them many times. This burdens the DFS-style BMC approach a lot.

To handle this problem, in this paper, the authors propose a backward-DFS BMC approach for LHA. First, reverse the graph structure of LHA. Then, conduct the DFS-style BMC on the reversed LHA. In this way, the "deep" path segments in the forward-DFS can be found very quickly to prune the DFS tree which is needed to be traversed. This backward-DFS approach is implemented into BACH. The experiment shows the performance of BACH is optimized significantly to handle large cases.

I. I

Reachability verification of Linear Hybrid Automata (LHA) [1] is a very difficult and important problem. Currently, researchers always try to handle this problem in two ways:

Classical Model Checking(CMC)[4]: Compute the complete state space by methods like polyhedra computation, like HYTECH[6] and PHAVer[7]. First of all, the classical reachability verification problem of LHA is proven to be undecidable[2], [3]. Furthermore, these methods are very complex and sensitive to the number of variables. Thus, they do not scale well to the size of practical problems.

Bounded Model Checking(BMC)[5]: Encoding the bounded reachability problem into the satisfiability problem of a boolean combination of propositional variables and linear mathematical constraints, which can be solved by SMT solvers[8], [9]. As this technique requires to encode the bounded problem space firstly, when the system size or the given threshold is large, the object problem will be very huge, which restricts the size of the problem that can be solved.

Both classical and bounded model checking are feeding the (partly) complete state space to the underlying solver at one time which suffers from the state explosion problem and restricts the problem size that can be solved. Study[13] proposed a linear programming (LP)-based approach to check one abstract path at one time to find whether there exists a behavior of LHA along this path and satisfy the given reachability specification. Study[15] extended this approach by using forward depth-first-search (F-DFS) to traverse on the graph structure of LHA to enumerate and check all the abstract paths with length no longer than the threshold one by one to answer the bounded reachability problem. Furthermore, when the DFS is finished before touching the bound, this approach can prove the given specification is not satisfied in general, not only in the given bound. A prototype tool BACH[15], [16] was implemented based on this idea. The experiments show that BACH is efficient in many cases.

Clearly, if DFS traverses more quickly, the BMC can be finished more efficiently. In BACH, once a path is found to be infeasible, the DFS will ask the underlying LP solver to locate the path segment which makes this path infeasible and backtrack to the path segment to prune the behavior tree which is needed to be traversed [17]. Nevertheless, in many cases, the path segments which make the path infeasible are hidden "deeply" or have many entry points in the graph structure which makes the DFS difficult to find these path segments or has to traverse them time and time again. This asks the DFS to traverse a lot of obviously infeasible paths, which makes the backtracking strategy inefficient in many cases and burdens the DFS-style BMC a lot.

To handle this problem, in this paper, the authors propose a backward-DFS BMC (B-DFS)[10], [11] approach to complement the classical F-DFS. First, reverse the graph structure of LHA. Mark the original target location as initial location, and mark the original initial location as target. Then, conduct the F-DFS BMC on the reversed LHA. As these path segments are "deep" in the original graph structure, clearly, if the DFS is conducted in a backward way, the path segments will be "shallow" in the reversed graph and can be found more quickly. Furthermore, for those path segments which have many entry points, the abstract paths with these path segments as suffix will be pruned easily in the reversed LHA to shrink the size of the DFS tree which is needed to be traversed.

This B-DFS BMC idea is implemented into BACH as a complement to the F-DFS BMC. Once a LHA and a reachability specification is given, BACH will start two threads, one conducts the F-DFS BMC on the original LHA and the other conducts the B-DFS BMC on the reversed model. The procedure terminates when any of these two threads finish. We conduct a series of case studies on the new BACH, and

*State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing, Jiangsu, P.R.China, 210046 Email: yangyang@seg.nju.edu.cn,{bulei, lxd}@nju.edu.cn
† Corresponding author.

978-0-9835678-2-0/12/$31.00 ©2012 FMCAD Inc.

compare it with both state-of-the-art classical model checker PHAVer and SMT solver MathSAT. The experiment results show that:

- By B-DFS, the state space needed to search and verify is pruned substantially. Therefore, BACH outperforms the other competitors significantly.
- In many cases, the DFS is finished before touching the bound. Then, in this situation, BACH can prove the given specification is not satisfiable in general, not only in the given bound, which is incapable for other BMC checkers.

II. T U T

A. Forward-DFS Approach

Now, let's recall the F-DFS BMC approach given in study [15]. The main idea is to traverse all the abstract paths with length shorten than or equal to the bound by DFS on the graph structure of the LHA. Whenever an abstract path ρ, which contains the target location, is found, a decision procedure will be called to check whether the LHA has a feasible behavior according with ρ. The decision procedure will encode all the syntax elements, i.e., invariants, guards and e.t.c., in ρ into a linear constraint set $\mathbb{R}$ as guided by [13] and [14]. In this manner, the reachability problem of ρ is transformed into the feasibility of $\mathbb{R}$ which can be answered by an LP solver very efficiently.

It is obvious that the performance of the F-DFS-based BMC depends on the performance of the DFS algorithm. If the DFS can be finished more quickly, the BMC can be finished more efficiently. Therefore, we gave an optimization technique to generate unsatisfiable core from the proven infeasible paths to tailor the state space needed to be traversed[17]. Once ρ is proved to be infeasible by the underlying LP solver, the solver will provide a infeasible irreducible set (IIS) [18] of $\mathbb{R}$, which is a minimal set of constraints in $\mathbb{R}$ that makes $\mathbb{R}$ infeasible. Then the constraint set $\mathbb{R}'$ that IIS located can be mapped back to a path segment ρ' in the path ρ, which means ρ' is the infeasible core in the path that makes ρ infeasible. Since removing any constraint in $\mathbb{R}'$ will make it feasible, so the DFS will backtrack to the location preceding the last location in ρ'.

Take the sample LHA in Fig.1 for example. We want to check whether location v_6 is reachable within bound 20. Suppose the current visiting path is $\rho = \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_5\rangle \xrightarrow[e_5]{} \langle v_6\rangle$ which is proved to be infeasible by an LP solver, and the infeasible path segment located by IIS is $\rho' = \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle$, so the location that DFS will backtrack to is v_3 instead of v_5. Then the DFS will begin to search the next branch under $\rho'' = \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle$, which is $\langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_9]{} \langle v_1\rangle \xrightarrow[e_8]{} \langle v_5\rangle \xrightarrow[e_5]{} \langle v_6\rangle$. As we can see, the subtree beneath $\rho'' \xrightarrow[e_3]{} \langle v_4\rangle$ is pruned completely to speed up the DFS.

Nevertheless, let's look at the following path $\rho_1 = \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_9]{} \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_5\rangle \xrightarrow[e_5]{} \langle v_6\rangle$. ρ_1 is a graphically correct path beneath ρ'' and contains target location v_6. Therefore, the F-DFS BMC will ask the underlying

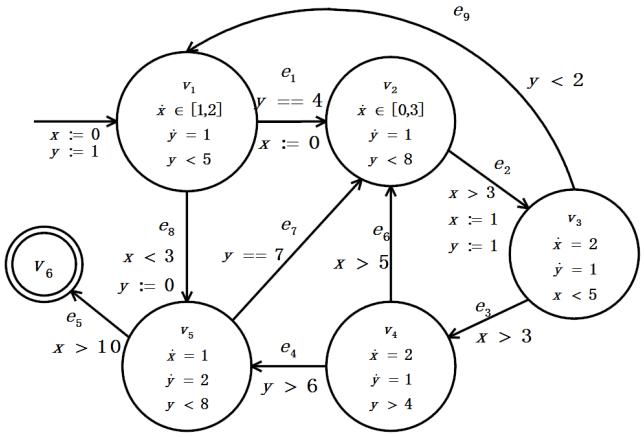

Fig. 1. Sample Automaton

decision procedure to check whether ρ_1 is feasible or not. Indeed, as ρ_1 contains ρ', it can be proved to be infeasible at once. Then DFS will backtrack to the second v_3 in ρ_1. But, as the bound 20 is not reached yet, the DFS will traverse the loop $\langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle$ for the third time and then ρ' again, i.e., traverse a new path $\langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_9]{} \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_9]{} \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_5\rangle \xrightarrow[e_5]{} \langle v_6\rangle$. In this manner, lots of time will be wasted on DFS to traverse these paths which are doomed to be infeasible. Obviously, if there is a method to prune the DFS tree and avoid the visiting of such paths, the DFS-BMC will be much more efficient.

B. Backward-DFS Approach

As discussed above, in many times, the path segments which make the path infeasible are hidden quite "deeply" or could have many entries in the graph structure which makes the F-DFS difficult to find them or has to traverse them lots of times. To solve this problem, in this paper we propose to conduct the DFS search in a backward way(B-DFS BMC): First, reverse the transition relation of the LHA model. For each transition $\langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle$ in LHA H, there will be a transition $\langle v_2\rangle \xrightarrow[e_1]{} \langle v_1\rangle$ in the reversed LHA $\neg H$. Then mark the original target location as initial location, and mark the original initial location as target. For example, Fig.2 is the reversed model of the LHA given in Fig.1.

Now, we can conduct the F-DFS BMC on the reversed LHA, which means searching for a reversed path from the original target location to the initial location. Once a reversed path is found, we will ask the LP solver to check the accordingly path in the original model and locate the infeasible path segments as well. As many infeasible path segments are "deep" in the original graph structure, if the DFS is conducted in a backward way, these path segments will be "shallow" in the reversed structure and can be found more quickly. Furthermore, for those path segments which have many entry points, all the paths containing those path segments as "suffix" will be reversed, then those infeasible path segments will become "prefix" now. As a result, these paths can be pruned easily

in the reversed LHA to shrink the size of the DFS tree which is needed to be traversed.

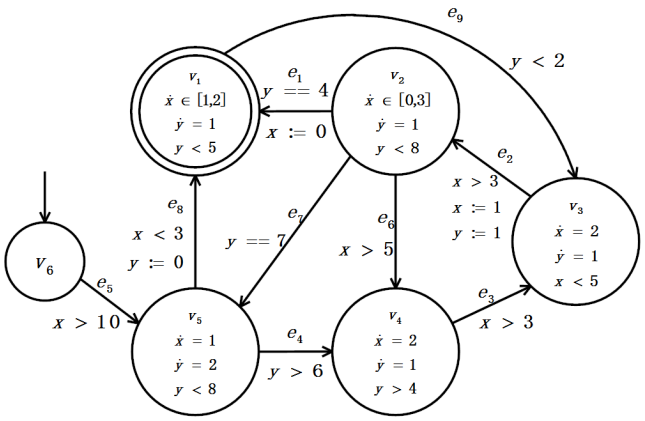

Fig. 2. Reversed Sample Automaton

Now, let's look at the previous example again (whether v_6 is reachable within bound 20 in Fig.1). On the reversed model, Fig.2, the first path that is traversed and solved is: $\neg\rho_1 = \langle v_6\rangle \xrightarrow[e_5]{} \langle v_5\rangle \xrightarrow[e_4]{} \langle v_4\rangle \xrightarrow[e_3]{} \langle v_3\rangle \xrightarrow[e_2]{} \langle v_2\rangle \xrightarrow[e_1]{} \langle v_1\rangle$. The accordingly path in the original model is $\rho_1 = \langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_5\rangle \xrightarrow[e_5]{} \langle v_6\rangle$. Therefore, ρ_1 is given to the underlying LP solver. The LP solver can tell that ρ_1 is infeasible and $\rho_1' = \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle \xrightarrow[e_3]{} \langle v_4\rangle$ is the infeasible core. The according infeasible core in $\neg\rho_1$ is $\neg\rho_1' = \langle v_4\rangle \xrightarrow[e_3]{} \langle v_3\rangle \xrightarrow[e_2]{} \langle v_2\rangle$. So, the location which is backtracked to is v_3 in $\neg\rho_1$. As v_3 has no other successor locations in $\neg H$, the DFS keeps backtracking to v_5 and traverses the next path $\neg\rho_2 = \langle v_6\rangle \xrightarrow[e_5]{} \langle v_5\rangle \xrightarrow[e_8]{} \langle v_1\rangle$. Similarly, LP solver locates the infeasible core path segment of $\neg\rho_2$ as $\neg\rho_2' = \langle v_6\rangle \xrightarrow[e_5]{} \langle v_5\rangle \xrightarrow[e_8]{} \langle v_1\rangle$. In this case, the DFS backtracks to v_5 in $\neg H$ again and finds there is no more path to travel. This means there doesn't exist a feasible path which can go from the initial location to the target. Therefore, the target location v_6 can be proved to be not reachable in general, not only within bound 20!

Clearly, in this example, the DFS tree that needed to be traversed are shrunk significantly by B-DFS BMC. Many paths like $(\langle v_1\rangle \xrightarrow[e_1]{} \langle v_2\rangle \xrightarrow[e_2]{} \langle v_3\rangle)^k \xrightarrow[e_3]{} \langle v_4\rangle \xrightarrow[e_4]{} \langle v_5\rangle \xrightarrow[e_5]{} \langle v_6\rangle$ in F-DFS are pruned. The B-DFS only needs to check two paths at all to tell location v_6 is unreachable.

C. Bidirectional-DFS Implementation

As discussed above, if the path segment which makes the whole path infeasible is "deep" or has many entry points, the B-DFS BMC method will be more efficient than the F-DFS BMC method. Nevertheless, in another word, if the path segment is "shallow" or has many exit points, then F-DFS BMC will outperform B-DFS BMC. So, none of these two techniques can take over the other one.

Therefore, we combine the F-DFS and B-DFS BMC to together, which results in a bidirectional-DFS BMC. In our bidirectional-DFS, the algorithm will start two threads, one

TABLE I
P B -DFS BMC

```
S   (HybridAutomata ha)
1.    HybridAutomata ha_r=reverse(ha);
2.    Thread t_1=new subSolver(ha);
3.    Thread t_2=new subSolver(ha_r);
4.    while t_1.isAlive() and t_2.isAlive()
5.        sleep(10);
6.    if (t_1.isAlive())
7.        return t_2.result();
8.    else
9.        return t_1.result();
10.   return 0;
```

conducts the F-DFS BMC on the original LHA and the other conducts the B-DFS on the reversed model. Be different from classical bidirectional-DFS algorithms like[11], [12], where the two threads can cooperate with each other, in our algorithm, these two threads work in a competition nature, that these two threads will not communicate with each other during searching, and the algorithm terminates when any of these two threads finish the searching. Under this setting, no matter where the infeasible path segment is, our bidirectional-DFS BMC can traverse the state space efficiently. The pseudo code for our implementation is shown in Table. I.

III. C S

We implement the bidirectional-DFS BMC, (F-DFS plus B-DFS), into our LHA bounded model checker BACH [15], [16] as guided by the last section. The latest version of BACH (V4.0) is implemented in Java, and can be downloaded from http://seg.nju.edu.cn/BACH/. As the LP solver underlying the previous versions of BACH is OR-objects[19] which does not support the functionality of IIS analysis, BACH 4 calls the IBM CPLEX[20] instead, which gives a nice support of IIS analysis. The main functionality of BACH is provided by the following set of services:

- Graphical LHA Editor: This component allows users to construct, edit, and perform syntax analysis of LHA interactively. This Editor can also transform the graphical representation of LHA to a readable text file which is used as the input file for reachability checking.
- Path-Oriented Reachability Checker: The checker requires users to select a specific path in the model. Then, it can check whether the reachability specification is satisfied along with the given path.
- Bounded Reachability Checker: This checker uses the path-oriented checker as underlying solver. It traverses the behavior tree of the model under the threshold by our adapted DFS algorithm, and checks the related path for reachability to perform bounded reachability checking.

In order to evaluate the performance of the BACH 4.0, we conduct a series of case studies on a set of well-known cases, which includes the temperature control system in Fig.3 and water-level monitor system in Fig.4, the scalable automated highway system in Fig.5, and the sample automaton given in Fig.1. The target locations are all marked by double circle in the models and they are all unreachable

We compare the BMC performance of BACH 4.0, marked as $BACH_{F+B}$, with the previous BACH, which only includes F-DFS and marked as $BACH_F$, and the-state-of-the-art SMT solver MathSAT5[9]. The experiments are conducted on a Think Center desktop machine(Intel Core2 Quad CPU 2.83GHz, 4GB RAM and Ubuntu 10.04). The time limit for experiment is set as one hour. The input models we used in experiments are all available from http://seg.nju.edu.cn/BACH/.

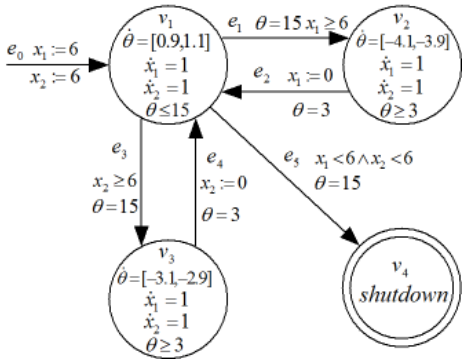

Fig. 3. Temperature Control System

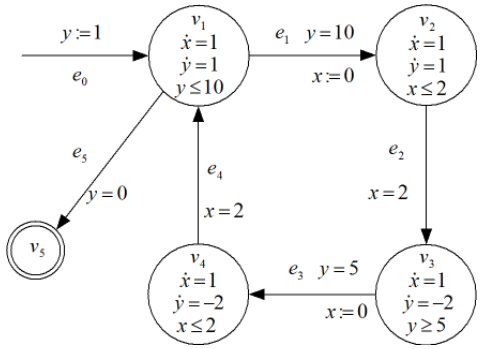

Fig. 4. Water-Level Monitor System

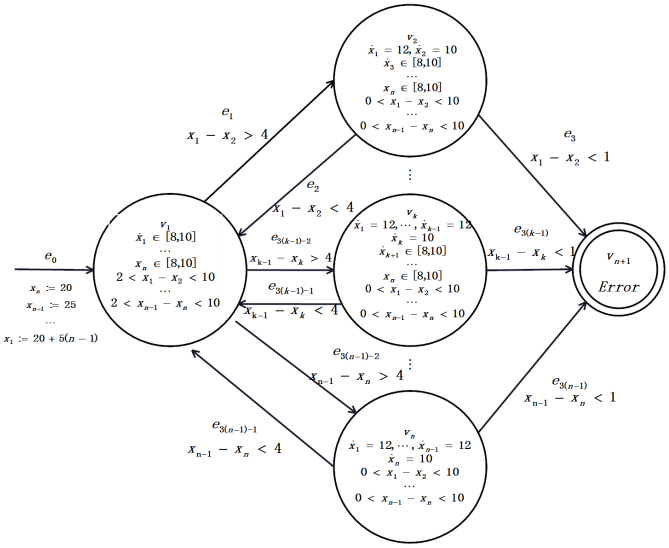

Fig. 5. Automated Highway System

The experiment data for the time (second) spent in each benchmark is shown below in Table.II-V respectively. If the checker fails to give a result in the time limit, the corresponding blank is marked as N/A. The number of bound means the largest number of discrete locations that a path can have in the state space under searching. Furthermore, as we mentioned that if the DFS finished before touching the bound, then BACH can prove the target is not reachable in general. In such cases, the time that BACH spent for solving the problem are marked with subscript G. For example, for the sample automaton, no matter how large the bound is, the new BACH only needs to check two paths to tell that the location v_6 is not reachable in general which only took 0.01 second. Therefore, in Table.II, all the blanks in column $BACH_{F+B}$ are combined to together. Besides of that, a special row ∞ is added to emphasise this problem can be proved in general not only in given bound.

As we can see from these tables that, for sample, water and highway system, new BACH can all give a general proof of the unreachability of the targets. Therefore, we make a comparison of new BACH with PHAVer[7] which is the-state-of-the-art classical model checker for LHA. We find that for sample automaton and water automaton, both BACH and PHAVer can finish in 0.01 second. For highway system which size, number of locations and variables, is scalable by increasing the number of vehicles in the system, the experimental data is plotted in Fig.6. We can see that the largest highway system that new BACH solved in one hour has 150 vehicles included, which is a big model of 150 variables and 151 locations, while in one hour, PHAVer can only solve a system with 6 vehicles. Furthermore, the only model that new BACH can not give a general proof is the temperature control system, while the computation of PHAVer can not terminate on this model and it fails to give any result about this model.

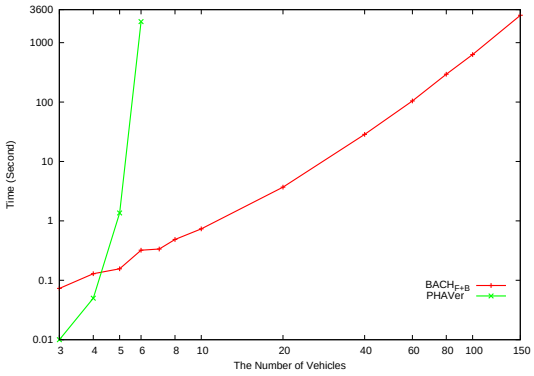

Fig. 6. BACH VS PHAVer on Automated Highway System

From these data, we can see that:

- By the help of integrating backward-DFS into BACH, the performance of BACH is optimized significantly.
- As BACH only checks one path at a time, the complexity of the verification is well controlled. As a result, the scalability of BACH is much better than the SMT-style BMC solvers, like MathSAT.
- When the DFS terminates before touching the bound, DFS-style BMC can prove the unreachability of certain

TABLE II
P O T S A

Tech. / Bound	$BACH_F$	$BACH_{F+B}$	MathSAT
40	0.798	0.01_G	4.44
60	86.303		17.977
80	N/A		38.006
200	N/A		2597.77
∞	N/A		N/A

TABLE III
P O T T C S

Tech. / Bound	$BACH_F$	$BACH_{F+B}$	MathSAT
20	0.201	0.136	0.748
40	140.618	0.665	7.896
100	N/A	6.903	613.774
600	N/A	1399.473	N/A

TABLE IV
P O T W - M A

Tech. / Bound	$BACH_F$	$BACH_{F+B}$	MathSAT
50	0.016	0.01_G	3.544
150	0.046		139.141
250	0.167		2050.204
8000	2194.268		N/A
∞	N/A		N/A

TABLE V
P O T A H S 10 V

Tech. / Bound	$BACH_F$	$BACH_{F+B}$	MathSAT
10	1.882	0.733_G	0.648
50	N/A		21.157
100	N/A		146.665
150	N/A		3100.934
∞	N/A		N/A

targets in general which is incapable for SMT-style BMC checker.

- The underlying decision procedure of BACH is LP. It is well known that LP is much cheaper than polyhedral computation which is under PHAVer. As polyhedral computation is extremely sensitive to number of variables, we can see BACH outperforms PHAVer substantially when facing system with large number of variables.

IV. C F W

The state-of-the-art tools for the bounded reachability analysis of LHA can only analyze systems with small dimension and bound. In the previous work, we present a DFS-style BMC method to traverse and check each abstract path in bound in the graphical structure of the LHA. Clearly, the DFS finishes more quickly, the bounded model checking can be conducted more efficiently.

In this paper, we present a backward searching technique for the DFS-style traversing strategy and combine it with the classical forward DFS in our tool BACH to accelerate the DFS traversing. The experiments show that the size of the problem that BACH can solve is increased substantially. By this forward plus backward DFS approach, BACH outperforms the-state-of-the-art competitors significantly.

In the current setting, the F-DFS and B-DFS are running independently. In the future, we will try to let these two threads to cooperate with each other. Then the state space has been pruned by one thread can benefits the other one, and the DFS-based BMC shall be finished more efficiently.

A

We would like to thank the anonymous reviewers for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (No.61100036, No.90818022, and No.61170066), the National Grand Fundamental Research 973 Program of China (No.2009CB320702), the National 863 High-Tech Programme of China (No.2011AA010103, No.2012AA011205) and by the Jiangsu Province Research Foundation (No.BK2011558).

R

[1] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of LICS 1996*, IEEE Computer Society Press, 1996, pp. 278-292.
[2] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's Decidable About Hybrid Automata? In *Journal of Computer and System Sciences*, 57:94-124, 1998.
[3] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H.Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. The algorithmic analysis of hybrid systems. In *Theoretical Computer Science*, 138(1995), pp.3-34.
[4] E. Clarke, O. Grumburg, and D. Peled. Model Checking. The MIT Press, 1999.
[5] A. Biere, A. Cimatti, E. Clarke, O. Strichman, Y. Zhu. Bounded Model Checking. In *Advance in Computers*, Vol.58, Academic Press, 2003, pp.118-149
[6] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: a model checker for hybrid systems. In *STTT*, 1:110-122, Springer, 1997.
[7] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. In *Proceedings of HSCC'05*, LNCS 2289, pp.258-273.
[8] M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. In *Journal on Satisfiability, Boolean Modeling and Computation*, 2007, vol.1, pp.209-236
[9] Gilles Audemard, Marco Bozzano, Alessandro Cimatti, Roberto Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In *Proceedings of BMC 04*, ENTCS 119:2, Elsevier Science, 2005, pp. 17-32.
[10] Ofer Strichman. Accelerating Bounded Model Checking of Safety Properties", In *Formal Methods for System Design*, 24, pp.5-24, 2004.
[11] Dennis de Champeaux, Lenie Sint. An improved bidirectional heuristic search algorithm, In *Journal of the ACM* 24:(2), pp.177-191, 1977.
[12] Ira Pohl. Bi-directional Search, In *Machine Intelligence*, 6, Edinburgh University Press, pp.127C140, 1971.
[13] X. Li, S. Jha, and L. Bu. Towards an Efficient Path-Oriented Tool for Bounded Reachability Analysis of Linear Hybrid Systems using Linear Programming. In *Proceedings of BMC06*, ENTCS 174:3, Elsevier Science, 2007, pp.57-70.
[14] L. Bu, and X. Li. Path-Oriented Bounded Reachability Analysis of Composed Linear Hybrid Systems, In *Software Tools Technology Transfer*, 13:4, pp.307-317, Springer, 2011.
[15] L. Bu, Y. Li, L. Wang and X. Li. BACH: Bounded Reachability Checker for Linear Hybrid Automata. In *Proceedings of FMCAD 08*, IEEE Computer Society, pp.65-68,2008.
[16] L. Bu, Y. Li, L. Wang, X. Chen and X. Li. BACH 2: Bounded ReachAbility CHecker for Compositional Linear Hybrid Systems, In *Proceedings of the 13th Design Automation & Test in Europe Conference*, Dresden, Germany, pp. 1512-1517, 2010.
[17] L. Bu, Y. Yang and X. Li. IIS-Guided DFS For Efficient Bounded Reachability Analysis of Linear Hybrid Automata, In *Proceedings of HVC 2011*.
[18] Chinneck, J., Dravnieks, E. Locating minimal infeasible constraint sets in linear programs. In *ORSA Journal on Computing*, 3 (1991), 157-168.
[19] OR-Objects. http://OpsResearch.com/ OR-Objects/index.html.
[20] CPLEX. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/