Preprints of the 2nd IFAC Conf. on Analysis and Design of Hybrid Systems (Alghero, Italy), 7-9 June 2006

AN EVALUATION OF TWO RECENT REACHABILITY ANALYSIS TOOLS FOR HYBRID SYSTEMS

Ibtissem Ben Makhlouf, Stefan Kowalewski *

* *RWTH Aachen University, Chair "Informatik 11" – Embedded Software Laboratory, Ahornstr. 55, 52074 Aachen, Germany*

Abstract: The hybrid systems community is still struggling to provide practically applicable verification tools. Recently, two new tools, PHAVer and Hsolver, were introduced which promise to be a further step in this direction. We evaluate and compare both tools with the help of several benchmark examples. The results show that both have their strengths and weaknesses, and that there still is no all-purpose reachability analysis tool for hybrid systems.

Keywords: Hybrid systems, reachability analysis, Hsolver, PHAVer.

1. INTRODUCTION

Automatic verification of hybrid systems is still a topic of research and far from being established as a standard tool in industry, like, e.g., simulation. To become so, reachability analysis, the core procedure of hybrid systems verification, must be made sufficiently efficient for models of real-world systems, including large, complex and non-linear dynamics.

The first publicly available reachability analysis tool for hybrid systems was HyTech (Henzinger *et al.*, 1997; Henzinger *et al.*, 2000), developed more than a decade ago. It was intended to be a demo implementation of theoretical results and not a prototype of an engineering tool. Consequently, numerical issues were largely neglected which lead to overflow errors even for relatively simple examples. In the following years, however, HyTech became the archetype for numerous attempts to implement reachability tools with better practical applicability, usually extending the basic HyTech algorithm by abstraction techniques.

Recently, the research on reachability analysis of hybrid systems got fresh momentum by the introduction by two different tools, Hsolver (Ratschan and She, 2005) and PHAVer (Frehse, 2005*c*). Both tools build on well tried algorithms while introducing new methodical approaches at the same time. And in both cases, the example computations which were presented in the corresponding publications promised that a new level of practical efficiency has finally been reached.

The aim of this paper is to evaluate the two tools from an independent point of view. For this purpose, we applied both tools to more than ten benchmark examples with several modifications, trying to fathom the boundaries of applicability. An excerpt of the results is presented in the sequel. We concentrate on the examples which revealed qualitative differences rather than quantitative performance differences. Some detailed data including runtime measurements are provided.

The paper is organized as follows. In the next section, we briefly introduce both tools. Section 3 is presenting the experimental results. Numerical results are depicted to make it possible to compare the analysis results to the "real" behavior. Finally, a comparison and some conclusions are given.

2. HSOLVER AND PHAVER

Hsolver is a safety verification tool for non-linear hybrid systems developed by Ratschan and She (Ratschan and She, 2005). The tool is based on interval arithmetics to delimit the trajectories in a piecewise manner to a rectangular grid. The method avoids explicit computation of continuous reachability sets reducing, therefore, rounding errors drastically. To avoid the drawbacks of this approach, namely vast over-approximations, an abstraction refinement framework is developed in which the abstract states are hyper-rectangles in the continuous part of the state space. The abstraction is refined using a splitting strategy, which is optimized by using constraint programming. The splitting method reduces information from possibly incomplete constraint propagation steps in order to avoid new splitting. Jump conditions, initial states, and unsafe states are also described by constraints. Moreover, a pruning function makes it possible to eliminate unreachable sets avoiding inefficient repetitions.

The other tool is PHAVer (Polyhedral Hybrid Automation Verifyer) developed by Frehse (Frehse, 2005*c*) (Frehse, 2005*b*) for verifying safety of linear hybrid automata [1] (LHA). PHAVer uses Hybrid I/O-Automata with affine dynamics (Lynch *et al.*, 2003). Computations use convex polyhedra as the basic data structure as it was already done in HyTech (Henzinger *et al.*, 1997; Henzinger *et al.*, 2000). The implementation is based on the Parma Polyhedra Library (PPL) (Bagnara *et al.*, 2002). The PPL supports closed and non-closed convex polyhedra and infinite precision arithmetic.

Beginning with the initial state, the reachability strategy of PHAVer computes reachable sets by using refinement of locations when affine dynamics are over-approximated by LHA-dynamics. The refinement can be adjusted by parameters. To do so, the constraints are prioritized according to the refinement parameters. The tool offers the possibility to enforce other prioritization criteria. For more details see (Frehse, 2005*a*). Over-approximation using bounding boxes and convex hull abstractions are also possible. The user can combine the different types of over-approximation and control the number of iterations for each type with special parameters given by the tools. PHAVer supports compositional and assume-guarantee reasoning (Frehse *et al.*, 2004).

2.1 Reachability analysis strategy of Hsolver

In Hsolver, a hybrid system is described by:

- a set of modes $s_1, ..., s_n$ each corresponding to a different continuous behavior of the system,
- variables $x_1, ..., x_k$ ranging over closed real intervals $I_1, ..., I_k$ and
- constraints defining the flow in each mode, the jump, the initial and safe conditions.

A hybrid system is then defined as a tuple $(Flow, Jump, Init, UnSafe)$ with:

- the constraint $Flow \subseteq \Phi \times \mathbb{R}^k$ in which Φ is the state space $\{s_1, ..., s_n\} \times I_1 \times ... \times I_k$, and $\mathbb{R}^k$ represents $\{\dot{x}_1, ..., \dot{x}_k\}$,
- the constraint $Jump \subseteq \Phi \times \Phi$, defined over the variables $x_1, ..., x_k$ of the start mode s and the variables $x'_1, ..., x'_k$ of the target mode s', and
- the constraints $Init \subseteq \Phi$ und $UnSafe \subseteq \Phi$ describing the set of the initial and unsafe states.

The original hybrid system described above will be abstracted by a discrete system defined over a new variable state space S.
A discrete system over S is defined as a tuple (Trans, Init, UnSafe) where $Trans \subseteq S \times S$, $Init \subseteq S$ and $UnSafe \subseteq S$ are new constraints deduced from original constraints via the definition of an abstraction relation between the original and the abstracted system as follows:

Definition: An abstraction relation between a hybrid system $(Flow_1, Jump_1, Init_1, UnSafe_1)$ and a discrete system $(Trans_2, Init_2, UnSafe_2)$ over S is a relation $\alpha \subseteq \Phi \times S$ such that:

- for all $q \in \Phi$, if there is a trajectory from an element of $Init_1$ to q according to $Flow_1$ and $Jump_1$, then for all q_α with $\alpha(q, q_\alpha)$ there is a trajectory from an element of $Init_2$ to q_α according to $trans_2$,
- for all $q \in Init_1$, there is a $q_\alpha \in Init_2$, with $\alpha(q, q_\alpha)$ and
- for all $q \in UnSafe_1$, if q is reachable from $Init_1$, then there is a $q_\alpha \in UnSafe_2$, with $\alpha(q, q_\alpha)$.

Hence, Hsolver computes for every hybrid system with a description D an abstraction $Abstract_D(\mathcal{B})$ over sets of abstract states $\mathcal{B}$ containing all elements of the state space reachable from the initial set such that all boxes corresponding to the same mode are non-overlapping. The resulting abstracted system is refined, during the computation, with another reduced abstracted system under the condition that an abstraction relation between the two systems exists, which preserve the reachable states during the transformation. The refinement strategy is repeated until a fixpoint is attained. Using strategies based on constraint propagation programming (Mackworth and Freud, 1985) parts of state space showing not to be reachable are excluded during the ab-

[1] In the sense that the continuous dynamics in each discrete location are linear.

straction process. For this purpose, a constraint $reachable_{B'}(s', z)$ is formulated expressing the conditions under which a point in a box B is reachable. Before that, the flow within a box B and a mode s is described with the constraint $flow_B(s, y, z)$. It expresses the fact that a point $y = (y_1, ..., y_k) \in B$ is reachable from a point $x = (x_1, ..., x_k) \in B$ via a flow in B and s.

The constraint $reachable_{B'}(s', z)$ is defined as the disjunction of three principle constraints:

- Constraints formulating the reachability from the initial set $initflow_B(s, z)$ expressed for a mode s, a box B and a point $z \in B$ reachable from the initial set via a flow in s and B.
- Constraints $jumpflow_{B,B'}(s, s', z)$ describing the reachability from a jump between two modes s and s' corresponding to two boxes B and B' and a point $z \in B'$.
- Constraints $boundaryflow_{B,B'}(s', z)$ expressing the reachability of a point $z \in B'$ from a common point of neighboring boxes B and B' via a flow in B.

After the determination of the reachability constraint $reachable_{B'}(s', z)$, the problem of reachability is transformed in a constraint satisfaction problem. It is solved by constraint propagation programming, applying several techniques for making the analysis more efficient like, e.g., pruning.

2.2 Reachability analysis strategy of PHAVer

PHAVer is a verification framework for linear hybrid systems and compositional reasoning. The automata model differentiates between state, input and output variables. The output can be declared as the inputs for an other automaton. A hybrid I/O-automaton (HIOA) in PHAVer is a tuple $H = (Loc, Var_S, Var_I, Var_O, Lab, \rightarrow, Act, Inv, Init)$ where Loc, Lab, Act, Inv, and $Init$ are defined as usual, and where:

- Var_S, Var_I and $Var_O \subseteq Var_S$ are finite and disjoint sets of state, input and output variables.
- $\rightarrow \subseteq Loc \times Lab \times 2^{V(Var)\times V(Var)} \times Loc$ is a finite set of discrete transitions, where $Var = Var_S \cup Var_I$ and $V(Var)$ denotes the set of evaluations over Var. A transition $(l, a, \mu, l') \in \rightarrow$ means $l \xrightarrow{a,\mu} l'$.

PHAVer deals with I/O-automata with affine dynamics given for each location loc by a conjunction of constraints of the form:

$$a_i^T \dot{x} + \hat{a}_i^T x \bowtie_i b_i, \qquad (1)$$
$$a_i, \hat{a}_i \in \mathbb{Z}^n, \bowtie_i \in \{<, \leq, =\}, i = 1, ..., m.$$

Using linear programming, these constraints are first overapproximated conservatively with linear constraints of the form $\alpha_i^T \dot{x} \bowtie_i \beta_i,\ \alpha_i \in \mathbb{Z}^n,\ \beta_i \in \mathbb{Z}$ which are obtained using the result of an infimum computation of (1) inside $Inv(loc)$ denoted (when it exits) with:

$$p \backslash q = \inf_{x \in Inv(loc)} \alpha_i^T x, \quad p, q \in \mathbb{Z}.$$

The linear constraint on $\dot{x}$ is then given by $\alpha_i = qa_i$ and $\beta_i = qb_i - p$.
This overapproximation introduces a loss of accuracy depending on the size of the location and the angular spread of the derivative vectors in the location. To improve the accuracy, locations are recursively split along hyperplanes which parameters are adequately chosen from the user to minimize the partitions. The splitting of a location is stopped when a minimum size is attained. Otherwise, the splitting process can include a refinement procedure of a location controlled from the user. The user provides a list of candidate normal vectors $a_{h,i}$ of hyperplanes defined by equations having this form $a_h^T x = b_h$ and the minimum and maximum slack $\triangle_{min,h}$, $\triangle_{max,h}$ that the hyperplanes will have in the refined location. The slack is here defined by:

$$\triangle(a_h) = \max_{x \in Inv(loc)} a_h^T x - \min_{x \in Inv(loc)} a_h^T x$$

The candidate hyperplanes are prioritized according to a user controlled list of criteria. For each constraint of type $a^T x \bowtie b$ in a location loc a set of evaluations included in the reachable states N of the location are associated according to a chosen criterium from the following list (Frehse, 2005*c*):

- Prioritize constraints according to their slack.
- Prioritize constraints that have reachable states only on once side.
- Prioritize constraints according to the spread of the derivatives where constraints are discarded if a minimum spread is reached and the slack is smaller than $\triangle_{max,h}$.
- Prioritize constraints according to the derivative spread after the constraint is applied.

Moreover, the number of constraints during the computation is reduced in order to save memory. To select the constraints to be eliminated, PHAVer uses for instance an angle criterion for which the negative cosine of the closest angle between the normal vector of the ith constraint and all others is measured and then compared. Only a predefined number of constraints with the greatest angles are retained. In addition, a procedure for limiting the number of bits necessary to represent a constraints is executed as soon as a given threshold z is exceeded. A new constraint $\alpha_i^T x \bowtie_i \beta_i$ including the original ith constraint $a_i^T x \bowtie_i b_i$ of a polyhedron is computed so that $|\alpha_{i,j}|, |\beta_i| \leq 2^{z+1} - 1$. The procedure makes an

estimation of the scaling error f. It recomputes β_i using linear programming. If β_i has more than z bits, f will be decreased and the procedure starts again.

3. EXPERIMENTAL RESULTS

3.1 Benchmarks

In order to assess the performance of the tools, we applied them to benchmark examples from the the literature, including examples from (Ratschan and She, 2004). The computations were performed on a PC with a Pentium IV processor at 3,06 GHz and 1 GB of memory. The obtained results summarized in table 1 allow a first comparison between the two tools. A brief description of the studied examples is given here:

Example 1: The two tanks problem. It is a nonlinear problem, formulated in (Ratschan and She, 2005) as follows:

$$\textit{Flow}:\left(s=1\rightarrow\begin{pmatrix}\dot{h}_1\\ \dot{h}_2\end{pmatrix}=\begin{pmatrix}1-\sqrt{h_1}\\ \sqrt{h_1}-\sqrt{h_2}\end{pmatrix}\right)\bigwedge\left(s=2\rightarrow\begin{pmatrix}\dot{h}_1\\ \dot{h}_2\end{pmatrix}=\begin{pmatrix}1-\sqrt{h_1-h_2+1}\\ \sqrt{h_1-h_2+1}-\sqrt{h_2}\end{pmatrix}\right)$$

$$\textit{Jump}:(s=1\wedge 0.99\leq h_2\leq 1)\rightarrow(s'=2\wedge h_1'=h_1\wedge h_2'=1)$$

$$\textit{Init}:s=1\wedge(h_1-5.5)^2+(h_2-0.25)^2\leq 0.0625$$

$$\textit{Unsafe}:(s=1\wedge(h_1-4.5)^2+(h_2-0.25)^2<0.0625)$$

$$\textit{State space}:(1,[4,6]\times[0,1])\cup(2,[4,6]\times[1,2])$$

Example 2: The navigation benchmark problem is a linear problem from (Fehnker and Ivancic, 2004). An object at a position $x=(x_1,x_2)^T$ moves within a 3×3 map with a velocity v determined by the differential equation $\dot{v}=A(v-v_d)$, where $\text{A}=\begin{pmatrix}-1.2 & 0.1\\ 0.1 & -1.2\end{pmatrix}$ and $v_d=(sin(i\cdot\pi/4),cos(i\cdot\pi/4))^T$ is a desired velocity adopting different values depending on the chosen map (i is equal to the value in each cell of the map). We take two different maps:

$$map_1=\begin{bmatrix}\mathcal{B} & 2 & 4\\ 2 & 3 & 4\\ 2 & 2 & \mathcal{A}\end{bmatrix}\text{ and }map_2=\begin{bmatrix}\mathcal{B} & 2 & 4\\ 2 & 2 & 4\\ 1 & 1 & \mathcal{A}\end{bmatrix}.$$

$\mathcal{A}$ and $\mathcal{B}$ correspond to the reachable and forbidden cells. For map_1, we choose $x_0\in[0,1]\times[0,1]$, $v_0\in[-0.1,0.5]\times[-0.05,0.25]$ as initialization. However, for map_2 the computation begins with $x_0\in[0,1]\times[0,1]$, $v_0\in[-0.1,0.5]\times[-0.05,0.25]$ as initial sets.

Example 3: The 1-flow problem used in (Preußig *et al.*, 1998) is a simple example without jumps described as follows:

$$\textit{Flow}:\ \dot{x}_1=\dot{x}_2=\dot{x}_3=1$$

$$\textit{Init}:\ 0\leq x_1\leq 2\wedge 1<x_2\leq 2\wedge 0<x_3<1$$

$$\textit{Unsafe}:\ 0\leq x_1\leq 2\wedge 1<x_2\leq 2\wedge 0<x_3<1$$

$$\textit{State space}:\ [0,2]\times[0,2]\times[0,4]$$

Example 4: The collision example from a part of the car convoi control problem proposed in (Puri and Varaiya, 1995). It has been transformed in (Ratschan and She, 2004). The problem has no jumps and can be described as follows:

$$\textit{Flow}:\ \dot{x}_1=x_3-x_2,\ \dot{x}_2=x_4,\ -2\leq\dot{x}_3\leq-0.5,\ \dot{x}_4=-3\cdot x_4-3\cdot(x_2-x_3)+(x_1-x_2-10)$$

$$\textit{Init}:\ x_1=1,\ x_2=2,\ x_3=2,\ x_4=-0.5$$

$$\textit{Unsafe}:\ x_1\leq 0$$

$$\textit{State space}:\ [0,4]\times[0,2]\times[0,2]\times[-2,-0.5]$$

Another version of this problem denoted by *convoi1* is also taken from (Ratschan and She, 2004). The main difference is the dynamics of $\dot{x}_3=-4x_1+3x_2-3x_3+x_4$.

Example 5: A heating example of 3 rooms with 2 heaters from (Fehnker and Ivancic, 2004):

$$\dot{x}=\begin{pmatrix}-0.9 & 0.5 & 0\\ 0.5 & -1.3 & 0.5\\ 0 & 0.5 & -0.9\end{pmatrix}x+\begin{pmatrix}0.4\\ 0.3\\ 0.4\end{pmatrix}u+diag(6,7,8)h$$

where x is the vector of temperatures, h is indicating whether a heater is on or off and $u=4$. We performed computations with the minimal temperature threshold equal to 14 and a space state given by $[14,22]\times[14,22]\times[14,22]$.

3.2 Hsolver

Hsolver permits the verifications of nonlinear system from the outset. Difficulties appear, however,when arithmetic operations different than $*,+,$' sin, cos, exp appear, as in the two tank example which involves a *sqrt* operation. In this case, we must transform the equations in order to avoid this operation. The computation results for the different examples are given in table 1. We also present some graphical results of the two tanks and the navigation benchmark examples, in order to understand better the function of the tools. For the two tanks example, in Fig. 1 we see the pruned region in light grey boxes, the 10 boxes corresponding to mode 1 are dark grey and the 12 boxes of mode 2 are medium grey. The trajectory is obtained from a Matlab simulation. The figure shows that the unsafe region (circle left) is in the pruned region. This means, that this region was already excluded from the reachable set at the beginning of the computation which accelerates the termination of the verification process, as we expected. For the navigation benchmark, the computation time for the two instances was very long and the tool could not conclude about the safety in both cases (resulting output "safety unknown"). The problem behind that is illustrated by Fig. 2. It represent the different box decompositions in the three modes of the systems together with one trajectory from Matlab simulations. The issue is that the whole state space needed to be examined during the analysis without ever being able to

	Hsolver						PHAVer		
Example	refinement steps	box recomputation	# calling prune	safety	time(s)	memory(KB)	safety	time	memory
2-tanks	11	13	397	safe	0.33	1548	safe	0.11	1688
map1	362	511	509306	unsafe	2647.75	48048	safe	139.02	126608
map2	342	107	889110	unsafe	4457.7	46892	safe	22.14	54168
flow	1	0	2	safe	0.12	1136	safe	0	868
convoi	367	443	68675	safe	9462	7200	safe	0.10	148
convoi1	0	1	3	safe	0.31	6204	safe	0.11	1652
heating	142	191	469416	unsafe	638.76	37744	safe	0.41	5720

Table 1. Implementation results of the benchmark examples.

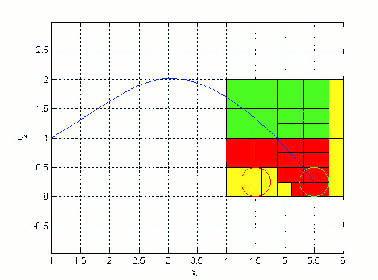

Fig. 1. Hsolver: two tanks problem, initial region: $(x_1 - 5.5)^2 + (x_2 - 0.25)^2 < 0.065$, unsafe region $(x_1 - 4.5)^2 + (x_2 - 0.25)^2 < 0.065$

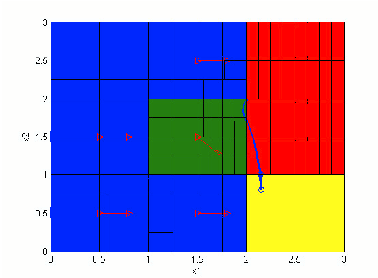

Fig. 2. Hsolver:Navigation benchmark, $map = [B24, 234, 22A]$, $x_0 \in [2,3] \times [1,2]$, $v_0 \in [-0.3, 0.3] \times [-0.3, 0]$

exclude safely the unsafe region. Such a behavior can not be predicted by a standard user.

3.3 PHAVer

Nonlinear problems must be linearized to be analyzable by PHAVer. For the two tanks example, we use the Jacobi method to linearize the system for each box using the center of the boxes as reference point. Of course, this is not an abstraction, and any reachability results for the linearized system will be inconclusive with respect to safety. However, it allows us to gather data for the comparison. We chose two different unsafe sets for the verification. From Matlab simulations we know that the system is safe for the first case and unsafe for the second case. The results of our verification with PHAVer confirm these findings. Fig.3 shows the results for the unsafe case.

The compositional feature of PHAVer is illustrated with the navigation benchmark examples. Here, the automaton is decomposed in two automata, one for the position and another for the

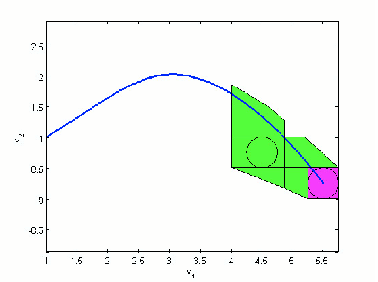

Fig. 3. PHAVer: Two tanks problem, initial region $[5.25, 5.75] \times [0, 0.5]$, unsafe region $[4.25, 4.75] \times [0.5, 1]$

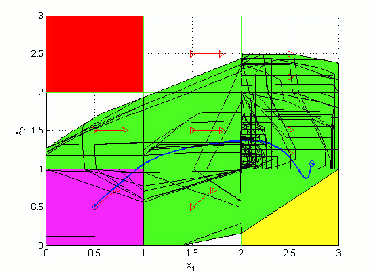

Fig. 4. PHAVer: Navigation benchmark, $map = [B24, 224, 11A]$, $x_0 \in [0,1] \times [0,1]$, $v_0 \in [-0.1, 0.5] \times [-0.05, 0.25]$

velocity. Both automata are synchronized with global labels. The refinement is controlled with the factor $d1min = d2min = 0.25$ and $d1max = d2max = \infty$. For map_1 with the default polyhedra abstraction, the computation time of the verification lies in acceptable range. Using this method for the second map, we get a very long computation time. For such cases PHAVer offers a further feature which permits the combination of different types of abstractions. In our example, we chose a convex hull abstraction during 20 iterations followed by the default abstraction with bounding box iterations until the 39th iteration. We took two different maps: $map1 = [B24, 234, 22A]$ and $map2 = [B24, 224, 11A]$. For the second map the termination is accelerated by the convex hull abstractions for the 20 first iterations, then the processing is switched to normal reachability with bounding box iterations until the 39th iteration. The results for the second map are shown in Fig. 4. The examples are proved to be safe. The figures show the polyhedra decomposition of a restricted region of the position state space. The other examples have been implemented as simple automata because the decomposition is not necessary in

this case. The corresponding results are shown in Table 1.

3.4 Comparison of the results of both tools

Comparing the time and the memory requirements in table 1, we can conclude that for all our examples PHAVer is faster than Hsolver. From the memory consumption point of view, we note that, against our expectation, the amount of memory used by Hsolver in examples 3, 2, 4 and 5 is greater than that used by PHAVer. When the safety is unknown, the verification analysis by Hsolver lasts a long time. In the case of non linear systems, we are not able to conclude about the safety of this type of systems because we have no estimation of the linearization error. In summary, Hsolver is an efficient and fast tool, as long as its results are conclusive. In these cases, it is well applicable to non-linear hybrid systems. However, the navigation and heating benchmarks show that the tool still has some limitation. On the other hand, Phaver is an efficient tool for verifying linear hybrid systems. It offers possibilities to control and choose refinement and over-approximation strategies used during computation with a number of functions, commands and parameters. Complicated hybrid systems could be decomposed or simplified and treated by Compositional Reasoning provided by the tool. This could be used to verify non-linear systems, if it would be possible to abstract the original problem by linear hybrid automata. Otherwise, we have to linearize the system and take the linearization error into account. Finally, we could not confirm the conjecture that the use of polyhedra and refinement steps will increase strongly the necessary amount of memory space.

4. CONCLUSIONS

Not quite surprisingly, the application results show that both tools have their strengths and weaknesses. Hsolver is extremely fast for examples where safety can be proven or disproved, but it shows excessive computation times when the results are inconclusive. Since it is able to treat non-linear dynamics directly, it has a clear advantage over PHAVer in this respect since the latter is restricted to affine hybrid systems. PHAVer, on the other side, provided more exact results in the sense that it was able to prove safety correctly for examples for which Hsolver gave the result 'safety unknown'.

Both tools represent a good step in the direction of making reachability analysis applicable for practioneers. However, considering that the benchmarks still were more academic than real world examples and that, nevertheless, there were examples for which computation times exceeded hours or the result was unnecessarily inconclusive, it seems fair to say that the goal has not been reached yet.

REFERENCES

Bagnara, R., E. Ricci, E. Zaffanella and P.M. Hill (2002). Possibly not closed convex polyhedra and the parma polyhedra library. In: *Static Analysis: Proc. Int. Symp., LNCS 2477*. Springer. pp. 213–229.

Fehnker, A. and F. Ivancic (2004). Benchmarks for hybrid systems verification. In: *HSCC'04, LNCS 2993*. Springer. pp. 326–341.

Frehse, G. (2005*a*). Compositional Verification of Hybrid Systems using Simulation Relations. PhD thesis. Cornell University.

Frehse, G. (2005*b*). Phaver. http://www.andrew.cmu.edu/~gfrehse. Software package.

Frehse, G. (2005*c*). PHAVer: Algorithmic verification of hybrid systems past HyTech. In: *HSCC, LNCS 3414*. Springer. pp. 258–273.

Frehse, G., Z. Han and B.H. Krogh (2004). Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In: *Proc. CDC'04*. Bahamas.

Henzinger, T.A., B. Horowitz, R. Majumdar and H. Wong-Toi (2000). Beyond HyTech: Hybrid systems analysis using interval numerical methods. In: *HSCC'00, LNCS 1790*. Springer. pp. 130–144.

Henzinger, T.A., P.S. Ho and H. Wong-Toi (1997). Hytech: A model checker for hybrid systems. *Soft. Tools Techn. Transf.* **1**(1,2), 110–122.

Lynch, N., R. Segala and F. Vaandrager (2003). Hybrid i/o automata. *Inf. Comput.* **185**(1), 105–157.

Mackworth, A. K. and E.C. Freud (1985). The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence* **25**, 65–74.

Preußig, J., S. Kowalewski, H. Wong-Toi and T.A. Henzinger (1998). An algorithm for the approximative analysis of rectangular automata. In: *Proc. FTRTFT'98, LNCS 1486*. Springer. pp. 228–240.

Puri, A. and P. Varaiya (1995). Driving safely in smart cars. In: *Proc. of the 1995 American Control Conference*. pp. 3597–3599.

Ratschan, S. and Z. She (2004). Hsolver. http://www.mpi-sb.mpg.de/ratschan/hsolver. Software package.

Ratschan, S. and Z. She (2005). Safety verification of hybrid systems by constraint propagation based abstraction refinement. In: *HSCC 2005, LNCS 3414*. Springer. pp. 573–589.