Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Reachability computation for hybrid systems with Ariadne

Luca Benvenuti [*] **Davide Bresolin** [**] **Alberto Casagrande** [***]
Pieter Collins [****] **Alberto Ferrari** [†] **Emanuele Mazzi** [†]
Alberto Sangiovanni-Vincentelli [†,‡] **Tiziano Villa** [**]

[*] *Università di Roma "La Sapienza", Roma, Italy (luca.benvenuti@uniroma1.it)*
[**] *Università di Verona, Verona, Italy (bresolin@sci.univr.it, tiziano.villa@univr.it)*
[***] *Università di Udine, Udine, Italy (casagrande@dimi.uniud.it)*
[****] *CWI, Amsterdam, The Netherlands (pieter.collins@cwi.nl)*
[†] *PARADES, Roma, Italy (aferrari@parades.rm.cnr.it, emazzi@parades.rm.cnr.it)*
[‡] *Dept. of EECS, University of California, Berkeley, California(alberto@eecs.berkeley.edu)*

Abstract: ARIADNE is an in-progress open environment to design algorithms for computing with hybrid automata, that relies on a rigorous computable analysis theory to represent geometric objects, in order to achieve provable approximation bounds along the computations. In this paper we discuss the problem of reachability analysis of hybrid automata to decide safety properties. We describe in details the algorithm used in ARIADNE to compute over-approximations of reachable sets. Then we show how it works on a simple example. Finally, we discuss the lower-approximation approach to the reachability problem and how to extend ARIADNE to support it.

1. INTRODUCTION

In many applicative fields there is the need to model systems having a mixed discrete and continuous behaviour that cannot be characterized faithfully using either only discrete or continuous models. This is the case, for example, of automotive powertrain systems, where a four stroke engine is modelled by a switching continuous system and is controlled by a digital controller. Such systems consist of a discrete control part that operates in a continuous environment and are named *hybrid systems* because of their mixed nature.

In order to model and specify hybrid systems in a formal way, Alur et al. (1992) and Maler et al. (1991) introduced the notion of *hybrid automata*. Intuitively, a hybrid automaton is a "finite-state automaton" with continuous variables that evolve according to dynamics characterizing each discrete node. Of particular importance in the study of a hybrid automaton is the *reachable set*, which consists of all states that can be reached under the dynamical evolution starting from a given initial state set. Hybrid automaton states consist of a discrete location paired with a vector of continuous variables, and therefore they have the cardinality of continuum. Because of this, the reachable set is, in general, not decidable, as it has been proved in Henzinger et al. (1995). Many papers therefore propose approximation techniques to estimate the reachable set (see Halbwachs et al. (1994); Dang and Maler (1998); Asarin et al. (2000); Kurzhanski and Varaiya (2000); Botchkarev and Tripakis (2000); Silva et al. (2001)). However, even the computation of approximations to the reachable set is not straightforward; indeed, it may not even be possible to compute a sequence of over-approximations convergent to the reachable set (Collins (2005)).

Many tools have been developed to compute or approximate reachable sets for hybrid systems, using different approaches. Tools like Kronos (Daws et al. (1995); Yovine (1997)) and UPPAAL (Larsen et al. (1997)) compute the reachability relation for systems based on timed automata. Other tools, such as `d/dt` (Asarin et al. (2002)), VeriShift (Botchkarev and Tripakis (2000)), HSOLVER (Ratschan and She (2007)), HybridSal (Tiwari (2008)), and HyTech (Henzinger et al. (1997)) compute approximations to the reachable set for hybrid automata with linear continous dynamics. PHAVer (Frehse (2005)) allows to set an arbitrary level of precision; CheckMate (Clarke et al. (2003)) can compute approximations to the reachable set for hybrid automata with non-linear dynamics. Additionally, general-purpose tools for set-based analysis, such as GAIO (Dellnitz et al. (2001)), COSY Infinity (Makino and Berz (2006)) and Mitchell's Toolbox of Level Set Methods (Tomlin et al. (2003)) may be used. These tools also include many interesting features such as model checking capabilities or graphical modeling interfaces.

However, most of these tools are unable to handle nonlinear dynamics and constraints and have restrictive licences, and some are even closed source. Without access to the source code, users can neither customize or optimize them for a specific class of instances of the reachability problem, nor check that the algorithms are correctly implemented.

To overcome these limitations, in Balluchi et al. (2006) we have recently proposed a framework for hybrid system verification. This tool, called ARIADNE, is a development environment in which to construct space representation techniques and algorithms for reachability analysis. It integrates and implements existing and new algorithms and representation techniques with a high degree of flexibility and customization, to let users choose the best methods for their needs. The package is released as an open source distribution, so that different research groups may contribute with new data structures, algorithms and heuristics. In this paper we concentrate on the algorithm used in ARIADNE to compute over-approximations of reachable sets. Because of the modular structure of the package, it can work with many different types of numeric representations of real numbers, continuous functions and regions of space. It can be also easily extended to cover different classes of hybrid automata. Furthermore, it relies on a rigorous computable analysis theory to represent geometric objects, in order to achieve provable approximation bounds along the computations.

The paper is organized as follows. In Section 2 we briefly introduce both syntax and semantics of hybrid automata. In Section 3 we describe the reachability problem and give possible approaches for its solution. In Section 4 we describe in details the algorithm used in ARIADNE to compute over-approximations of reachable sets, while in Section 5 we give a simple example of how the algorithm works. In Section 6 we discuss the lower-approximation approach to the reachability problem. Finally, Section 7 ends the paper discussing the current implementation status and future work.

2. HYBRID SYSTEMS AND AUTOMATA

We first give a formal definition of a hybrid automaton, based on an underlying discrete automaton.

Definition 1. (Hybrid Automaton). A *hybrid automaton* is a tuple $H = \langle \mathcal{Q}, \mathcal{E}, \mathcal{X}, Inv, Dyn, Act, Reset \rangle$ such that:

(1) $\langle \mathcal{Q}, \mathcal{E} \rangle$ is a finite directed graph; the vertexes, $\mathcal{Q}$, are called *locations* or *control modes*, and the directed edges, $\mathcal{E}$, are called *control switches*;
(2) Each location $q \in \mathcal{Q}$ is labeled by the predicate $Inv(q)$ on the set $\mathcal{X}$ and the transitive relation $Dyn(q)$ on $\mathcal{X} \times \mathcal{X} \times \mathbb{R}^{\geq 0}$ such that if $Inv(q)[p]$ is true then $Dyn(q)[p, p, 0]$ is true;
(3) Each edge $e \in \mathcal{E}$ is labeled by the predicate $Act(e)$ on $\mathcal{X}$ and the relation $Reset(e)$ on $\mathcal{X} \times \mathcal{X}$.

The predicate $Inv(q)$ is the *invariant condition* of q, and $Dyn(q)$ is the *dynamic law* of q, while $Act(e)$ is the *activation condition* of e and $Reset(e)$ is the *reset relation* of e. $Dyn(q)$ is a transitive relation, i.e., $\forall x, y, z \in \mathcal{X}$, $Dyn(q)[x, z, t_1 + t_2]$ is true if and only if both $Dyn(q)[x, y, t_1]$ and $Dyn(q)[y, z, t_2]$ are true.

We specify *Inv*, *Act* and *Reset* by a formula in some language over the reals, while we use differential equations to define *Dyn*. In this setting the dynamic law $Dyn(q)$ of a location q can be obtained by integrating the corresponding differential equations, with the intuitive meaning that if $Dyn(q)[z, z', t]$ holds, then the continuous flow can reach z' from z after time t. This definition of hybrid automata is quite general. As an example, *O-minimal* hybrid automata (see Lafferriere et al. (2000); Brihaye et al. (2004)) are a subclass of our hybrid automata, since we do not impose restrictions on the formulæ and on the resets, and *rectangular* hybrid automata (see Henzinger and Kopke (1996)) can be easily mapped into our definition.

Example 1. Consider a tank that is controlled through a monitor, which continuously senses the water level and turns the pump on and off. Water enters the tank from the top and leaves through an orifice in its base. The rate at which water leaves is proportional to the water level, denoted by the variable y, and it is such that $\dot{y} = -0.1y$. When the pump is on, the rate at which water enters is constant and equal to 2.0. Furthermore, from the time that the monitor signals to change the status of the pump to the time that the change becomes effective, there is a 2 seconds delay.

An hybrid automaton $H = \langle \mathcal{Q}, \mathcal{E}, \mathcal{X}, Inv, Dyn, Act, Reset \rangle$, that models this example, is depicted in Figure 1. The automaton has four locations: in locations q_0 and q_1 the pump is turned on, while in locations q_2 and q_3 the pump is turned off. The continuous state variable x is used to model the delays, while the continuous state variable y is the water level. The monitor signals to stop the pump when the water level raises to 10 and signals to start the pump when the level decreases to 4.5.

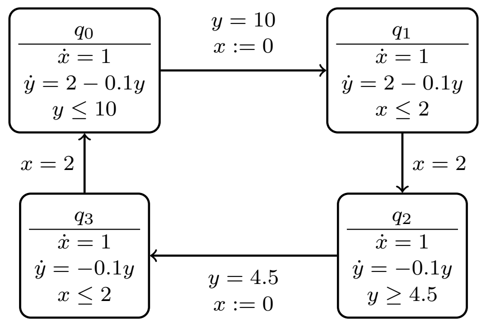

Fig. 1. Water level monitor.

In Section 5 we will show how this hybrid automaton can be analyzed using ARIADNE.

To formalize the semantics of hybrid automata, we first need to define the concept of hybrid automaton's state.

Definition 2. (Hybrid Automaton - States). Let H be a hybrid automaton. A *state* ℓ of H is a pair $\langle v, r \rangle$, where $v \in \mathcal{Q}$ is a location and $r \in \mathbb{R}^k$ is an assignment of values for the continuous variables. A state $\langle v, r \rangle$ is said to be *admissible* if $Inv(v)[r]$ holds.

Intuitively, an execution of a hybrid automaton corresponds to a sequence of transitions from a state to another. Hybrid automata have two kinds of transitions: *continuous reachability transitions*, capturing the continuous evolution of a state, and *discrete reachability transitions*, capturing the changes of location. More formally, we can define hybrid automaton semantics as follows.

Definition 3. (Hybrid Automaton - Semantics). Let H be a hybrid automaton. The *continuous transition relation* $\xrightarrow{t}_C$ between admissible states, where $t \geq 0$ is the elapsed time of the transition, is defined as follows:

$$\begin{aligned}&\langle v, r\rangle \xrightarrow{t}_C \langle v, s\rangle \iff \\ &\exists \text{ continuous } f : \mathbb{R}^{\geq 0} \to \mathbb{R}^k \text{ s.t. } r = f(0),\ s = f(t), \text{ and} \\ &\qquad \forall t' \in [0,t], \mathit{Inv}(v)[f(t')] \wedge \mathit{Dyn}(v)[r, f(t'), t'] \quad (1)\end{aligned}$$

Such a function f is called a *flow function.*

The *discrete transition relation* $\xrightarrow{e}_D$ between admissible states, where $e \in \mathcal{E}$, is defined as follows:

$$\begin{aligned}&\langle v, r\rangle \xrightarrow{\langle v,u\rangle}_D \langle u, s\rangle \iff \\ &\quad \langle v, u\rangle \in \mathcal{E} \wedge \mathit{Inv}(v)[r] \wedge \mathit{Act}(\langle v, u\rangle)[r] \\ &\qquad \wedge \mathit{Reset}(\langle v, u\rangle)[r, s] \wedge \mathit{Inv}(u)[s]. \quad (2)\end{aligned}$$

This semantics allows us to define the reachability relation.

The current user interface of ARIADNE supports directly automata with affine dynamics and, possibly, non-linear reset functions. Moreover, the user can define non-linear dynamics coding them directly with the internal data structures (in the future they will be too supported by the user interface). Every discrete location of the automaton can be defined independently from the other: it is possible to have locations with a linear dynamics and locations with a user-defined, non-linear, dynamics. The number of continuous variables too may differ for each location (for instance, in the automaton of Example 1 the variable x can be removed from locations q_0 and q_2, where it is useless). This could improve the performance of systems with a large number of continuous variables.

3. THE REACHABILITY PROBLEM

Definition 4. (Hybrid Automaton - Reachability). Let H be a hybrid automaton. A state $\langle q_r, r\rangle$ *reaches* a state $\langle q_s, s\rangle$ if there exist a sequence $(\ell_i)_{i\in I}$ of states such that $\ell_0 = \langle q_r, r\rangle$, $\ell_n = \langle q_s, s\rangle$ and either $\ell_{i-1} \xrightarrow{t}_C \ell_i$ or $\ell_{i-1} \xrightarrow{e}_D \ell_i$, for some $t \in \mathbb{R}^{\geq 0}$, and $e \in \mathcal{E}$.

We use $\mathit{ReachSet}_H(\langle q_r, r\rangle)$ to denote the set of states reachable from $\langle q_r, r\rangle$. Moreover, given a set of states $R \subseteq \mathcal{Q} \times \mathcal{X}$ we use $\mathit{ReachSet}_H(R)$ to denote the set $\cup_{\langle q_r,r\rangle\in R}\mathit{ReachSet}_H(\langle q_r, r\rangle)$.

Checking safety properties on hybrid automata reduces to the reachability problem. Suppose we wish to verify that a safety property φ holds for a hybrid automaton H; in other words, that φ remains true for all possible executions starting from a set R of initial states. Then we only need to prove that $\mathit{ReachSet}_H(R) \subseteq \mathrm{Sat}(\varphi)$, where $\mathrm{Sat}(\varphi)$ is the set of states where φ is true.

Unfortunately, the reachability problem is not decidable in general; there exists at least a hybrid automaton H and two sets of its states, R and T, such that there is no algorithm to decide whether $T \cap \mathit{ReachSet}_H(R)$ is empty or not (see Henzinger et al. (1995)). Some tools avoid these difficulties by restricting to classes of hybrid systems for which the reachable set is computable by algebraic means, and so the reachability relation is decidable.

Nevertheless, formal verification methods can be applied to hybrid automata. Suppose we can compute an over-approximation S to $\mathit{ReachSet}_H(R)$. Then if S is a subset of $\mathrm{Sat}(\varphi)$, then so is the reachable set. For these reasons, tools for reachability analysis of general hybrid automata are based on approximation techniques. The challenge is to find the "best" approximations of continuous regions. Some tools simply compute approximations to the reachable set, without any control of the errors; these may solve many practical problems, but the user cannot have absolute confidence in the answer. More sophisticated tools compute over-approximations to the reachable set, but work with fixed precision, and so may not be able to verify systems which are indeed safe. Even so, it is not possible, in general, to compute over-approximations to the reachable set to arbitrary accuracy in an approximative framework Collins (2005).

ARIADNE can compute convergent over-approximations to the *chain-reachable set*, which contains all points that can be reached by introducing an arbitrarily small amount of noise and can be proved to be the best possible over-approximation to the reachable set (see Collins (2007)). This means that, given an hybrid automaton H and an inital set $R \subseteq \mathcal{Q} \times \mathcal{X}$, it can compute a set $S \subseteq \mathcal{Q} \times \mathcal{X}$ such that $\mathit{ReachSet}_H(R) \subseteq S$. Moreover, by increasing the precision of the approximation, the set S can be made arbitrarily close to the chain-reachable set.

4. THE REACHABILITY ALGORITHM

Before discussing the reachability algorithm in details, we briefly describe how we represent regions of space in ARIADNE. For a more accurate description, see Balluchi et al. (2006).

Compact subsets of the Euclidean space $\mathbb{R}^n$ are approximated in ARIADNE by finite unions of simple sets, such as intervals, simplices, cuboids, parallelotopes, zonotopes, polytopes, spheres and ellipsoids, defined using rational coefficents. Since they form a base for the topology, they are called *basic sets*. The sets that can be represented exactly as a finite union of basic sets of a given type are called *denotable sets*.

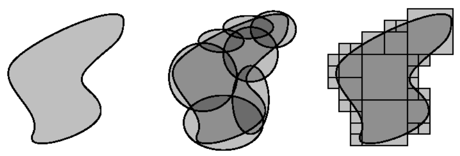

Fig. 2. Examples of approximations of a set.

Regions of space can also be represented in ARIADNE as unions of cells, described either by a fixed-size *grid*, or by a variable-size *partition tree* structure, which is typically more efficient to compute and store. Figure 2 shows how a non-denotable set can be over-approximated by using ellipsoids or partition trees.

In the case of hybrid automata we need to represent *hybrid sets*, which are regions in the space $\mathcal{Q} \times \mathbb{R}^n$. Hybrid sets are represented in ARIADNE starting from *hybrid basic sets* that pair a location of the automaton with a single basic set. Finite unions of hybrid basic sets are called *hybrid denotable sets*. ARIADNE can also use *hybrid grids* or *hybrid partition trees*, that consist of a grid/partion tree structure for every location of the automaton. Hybrid sets are then represented by marking the cells of the different grids/partition trees.

The chain-reachability algorithm of ARIADNE takes as inputs an hybrid automaton, a finite (hybrid) grid, an initial set and a bounding set (represented as sets of cells of the hybrid grid). It computes an over approximation of the infinite-time chain-reachable set of the automaton starting from the initial set and staying withing the bounding set. Since the infinite-time chain-reachable set of an hybrid automaton is (in general) not computable, we need to constrain the search space of the algorithm to assure termination. In our case, by giving a bounding set we restrict the algorithm to a bounded region of space that can be represented by a finite number of cells of the grid. The grid is used also to control the precision of algorithm (the finer the grid, the more accurate the computed region) and to store inputs, outputs and intermediate results by using a constant amount of memory (a region in a grid is represented by marking the corresponding cells).

The chain-reachability algorithm alternates continuous evolution and discrete evolution of the automaton until a fixpoint is reached, and it proceeds as follows.

(1) Start from the initial set and compute the continuous evolution of the automaton as long as possible. Mark the cells of the grid that are touched during the continuous evolution.
(2) When no more cells can be marked, compute a single discrete evolution step. Mark the new cells of the grid that are reached by the discrete step.
(3) If new cells are reached, go to (1). Otherwise, stop.

The algorithm terminates since the bounding set is divided into a finite number of cells by the underlying grid. Hence, after a finite number of steps either no more cells are marked or the whole search space is marked. In the following we describe in details how the continuous and the discrete evolution of an hybrid automaton is computed in ARIADNE.

4.1 Computing the continuous evolution

The continuous evolution of the system is computed by means of an integrator. ARIADNE's modularity allows the user to choose between different types of integrators included in the tool (we currently have an integrator for affine systems, the Euler integrator, and the Lohner integrator), or to add new custom integration methods.

Given a location $q \in \mathcal{Q}$ of the hybrid automaton, the computation of the continuous evolution in q starts from a set of cells of the hybrid grid and proceeds as follows. First, the initial set of cells is approximated as a union of basic sets. Then each basic set is integrated for a number of integration steps using the dynamic law $Dyn(q)$. The length of the integration step is adaptively determined, between user-specified minimum and maximum values. If a basic set becomes too large, the error in the evolution step becomes large, and the basic set may be subdivided to avoid catastrophic loss of accuracy. Finally, after a specified number of integration steps, the evolved sets are over-approximated or "locked" back to the grid. Figure 3 depicts this procedure.

Locking to the grid causes a loss of accuracy due to the over-approximation, but it is necessary for termination. After locking to the grid, the algorithm checks if new cells have been reached or not. In the former case, it continues with the integration phase starting from the newly reached cells. In the latter case, a fixpoint in the continuous evolution of the system has been reached and a number of "snapshots" of the system evolution have been computed, one for each locking-to-grid phase. The algorithm ends with a reachability step that computes an over approximation of the flow of the system starting from such snapshots. This last step is explained in Figure 4.

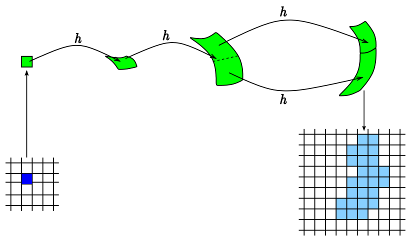

Fig. 3. Integration of a basic set with integration step h.

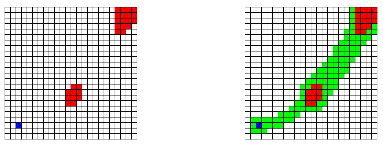
Fig. 4. Computing the flow of the system.

4.2 Computing the discrete evolution

Computing the discrete evolution of an hybrid automaton is simpler than computing its continuous evolution. Given an hybrid set of reached cells, we proceed as follows.

(1) For every control switch $e \in \mathcal{E}$ we determine the set of cells that intersects with $Act(e)$;
(2) if such set is not empty, we apply the reset function $Reset(e)$ to obtain the set of cells reached by the transition.

The discrete evolution of the system is computed using *upper semantics*: when there are multiply enabled transitions, or when the system exhibits grazing (tangential contact between a reached region and an activation set), the system evolves nondeterministically with all possible courses of action being taken. This guarantees that every point that can be reached by the automaton is included in the result, and thus that the algorithm computes an over-approximation of the reachable set.

5. AN EXAMPLE

In this section, we present a test case of reachability analysis for the hybrid automaton H defined in Example 1. The automaton models a water level monitor system: suppose that we start from an empty tank and that we want to guarantee that the water level is always between 3 and 13, except for the initial phase.

We start the reachability algorithm of ARIADNE from the initial location q_0 with initial values for the continuous variables $x = 0$ and $y = 0$. The first step of the algorithm is the computation of the continuous evolution of the system in the location q_0, that is depicted in Figure 5.a. During this phase, the water level increases up to 10. Then, the algorithm checks if there are active transitions. In this case, we have that the transition $\langle q_0, q_1\rangle$ is active: the controller signals to switch off the pump and goes to location q_1. The reset function of $\langle q_0, q_1\rangle$ is applied: x is reset to 0, y keeps its value (10) and the computation of the continuous evolution starts again from location q_1. Figure 5.b portraits the reach set after the continuous evolution in q_1.

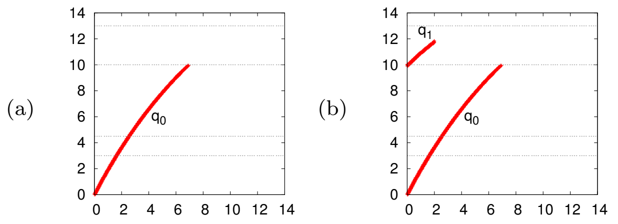

Fig. 5. Computing the continuous evolution in q_0 and q_1.

The water level continue to increase during the 2 seconds delay, then the transition $\langle q_1, q_2\rangle$ is activated: the pump is switched off and the water level decreases following the dynamics of location q_2, until it reaches 4.5 (see Figure 6.a). Then the controller signals to switch on the pump: transition $\langle q_2, q_3\rangle$ is activated and the continuous evolution of the system proceeds in location q_3. When the computation of this last continuous evolution is terminated the algorithm activates the transition $\langle q_3, q_0\rangle$. By doing this the system reaches a region that has been reached already during the computation of the continuous evolution in q_0. Hence, a fixpoint has been found: the algorithm stops and returns the reached set depicted in Figure 6.b.

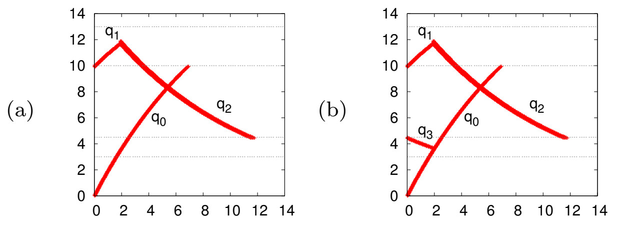

Fig. 6. Computing the continuous evolution in q_2 and q_3.

It is easy to see that, except for the initial phase, the water level is always between 3 and 13, and thus that the safety property we want to check is verified.

An over-approximation of the system evolution for a certain time can be computed by adding a new time variable t to the hybrid automaton (its dynamics is $\dot{t} = 1$ in every discrete location and its value is never reset by

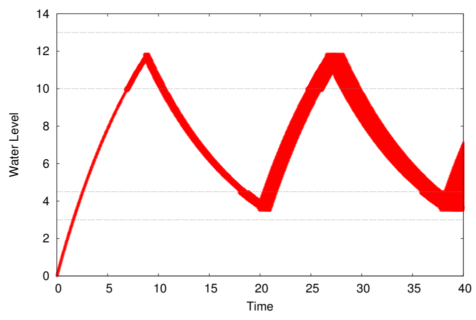

Fig. 7. Evolution of the water level with respect to time.

discrete transitions). Figure 7 represents the evolution of the water level for 40 time units. The system shows an oscillating behaviour that keeps the water level into the desired boundaries. Since we added a new variable to the automaton, Figure 7 is not directly comparable with the previous pictures. In particular, the solution's loss of precision over time is caused by over-approximation errors on the new variable t, and cannot be completely eliminated even by increasing the precision of the algorithm.

6. LOWER APPROXIMATION

In the previous sections we have described how ARIADNE computes over-approximations of reachable sets. However, there are some cases where over-approximations are not sufficient to determine if a system is safe or not. Let φ be a safety property, $R \subseteq \mathcal{Q} \times \mathcal{X}$ an initial set, and suppose that we have computed an over-approximation S of $\mathit{ReachSet}_H(R)$. If $S \subseteq \mathrm{Sat}(\varphi)$, we can conclude that the system is safe. However, if $S \nsubseteq \mathrm{Sat}(\varphi)$ we cannot say anything about the safety of the system: it could be the case that the system is safe, but the points in $S \setminus \mathrm{Sat}(\varphi)$ have been included in S because of over-approximation errors.

A possible way to determine if a system is unsafe is to compute an under-approximation of the reachable set, that is, a set $S \subseteq \mathit{ReachSet}_H(R)$. In this case, if $S \nsubseteq \mathrm{Sat}(\varphi)$ we can conclude that the system does not satisfy the safety property φ. Unfortunately, in most cases, it is not possible to compute an under-approximation of the reachable set, or we have that the only possible under-approximation is the empty set (this is the case, for instance, of single points and single trajectories). To overcome the problems with under-approximations, and to establish whether a system respect a safety property in a sufficiently large class of practical applications, we are currently extending ARIADNE to support *lower-approximations* of reachable sets. Given a set $T \subseteq \mathcal{Q} \times \mathcal{X}$, a lower-approximation of T is any finite union of basic sets $S = \bigcup_{i=1}^{n} S_i$ such that for every $i = 1, \ldots, n$, $S_i \cap T \neq \emptyset$ (there is at least one point of T in each S_i). Now, let φ be a safety property and $R \subseteq \mathcal{Q} \times \mathcal{X}$ be an initial set. If $S = \bigcup_{i=1}^{n} S_i$ is a lower-approximation of $\mathit{ReachSet}_H(R)$ and there exists an S_i that is completely disjoint from $\mathrm{Sat}(\varphi)$, we can conclude that there is at least one point of $\mathit{ReachSet}_H(R)$ that sits outside $\mathrm{Sat}(\varphi)$, and thus that the system is unsafe.

Computing lower-approximations of reachable sets is harder than computing over-approximations. Additional care should be taken in order to guarantee that the computed set is indeed a lower-approximation of the reachable set. The main problems that arise are the following.

- During the computation of the continuous evolution basic sets cannot be split. If B is a basic set of a lower-approximation of $\mathit{ReachSet}_H(R)$ (that is, B is such that $B \cap \mathit{ReachSet}_H(R) \neq \emptyset$) and we split it into B_1 and B_1, it is not guaranteed that both B_1 and B_2 satisfy the lower-approximation property.
- During the computation of the discrete evolution of the system, if there are multiply enabled transitions, or if the system exhibits grazing, it may be impossible to determine whether a transition is really activated or not.

All these problems should be considered and correctly solved in order to obtain an algorithm that computes lower-approximations of reachable sets.

7. CONCLUSION

In this paper we described the algorithm used in Ariadne to compute over-approximations of reachable sets of hybrid automata. We discussed also how to extend it to support lower-approximations as well.

Currently Ariadne supports both linear and non-linear discrete-time systems, continuous-time systems and hybrid systems in its C++ kernel. For all these systems it can compute over-approximations of the evolution and of the reachable set. A Python scripting interface is available and can be used for a fast and easy modeling and testing of real-world applications. We are working on extending the evaluation engine in order to fully support lower-approximations and timed reachability for hybrid system, and higher-order algorithms for discrete transitions.

REFERENCES

R. Alur, C. Courcoubetis, T. A. Henzinger, and P. H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, LNCS, pages 209–229. Springer, 1992.

E. Asarin, T. Dang, O. Maler, and O. Bournez. Approximate Reachability Analysis of Piecewise-Linear Dynamical Systems. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *LNCS*, pages 20–31. Springer, 2000.

E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pages 365–370. Springer-Verlag, 2002.

A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proc. of the 17th Int. Symp. on Mathematical Theory of Networks and Systems (MTNS 2006)*, 2006.

O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *LNCS*, pages 73–88. Springer, 2000.

T. Brihaye, C. Michaux, C. Rivière, and C. Troestler. On O-Minimal Hybrid Systems. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'04)*, volume 2993 of *LNCS*, pages 219–233. Springer, 2004.

E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Internat. J. Found. Comput. Sci.*, 14 (4):583–604, 2003.

P. Collins. Continuity and computability of reachable sets. *Theoretical Computer Science*, 341:162–195, 2005.

P. Collins. Optimal semicomputable approximations to reachable and invariant sets. *Theory Comput. Syst.*, 41 (1):33–48, 2007.

T. Dang and O. Maler. Reachability analysis via face lifting. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'98)*, volume 1386 of *LNCS*, pages 96–109, 1998.

C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'95)*, volume 1066 of *LNCS*, pages 208–219. Springer, 1995.

Michael Dellnitz, Gary Froyland, and Oliver Junge. The algorithms behind GAIO-set oriented numerical methods for dynamical systems. In *Ergodic theory, analysis, and efficient simulation of dynamical systems*, pages 145–174, 805–807. Springer, 2001.

G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005*, volume 3414 of *LNCS*, pages 258–273. Springer, 2005.

N. Halbwachs, Y.-E. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In *Static Analysis Symposium*, pages 223–237. Springer-Verlag, 1994.

T. A. Henzinger and P. W. Kopke. State Equivalences for Rectangular Hybrid Automata. In *Proceedings of CONCUR'96*, volume 1119 of *LNCS*, pages 530–545. Springer, 1996.

T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proc. of the 27th ACM Symp. on the Theory of Computing (STOCS '95)*, pages 373–382. ACM Press, 1995.

T. A. Henzinger, P. H. Ho, and H. Wong-Toi. HYTECH: a model checker for hybrid systems. *Int. J. on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Proceedings of Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *LNCS*, pages 202–214, 2000.

G. Lafferriere, G. J. Pappas, and S. Sastry. O-Minimal Hybrid Systems. *Mathematics of Control, Signals, and Systems*, 13:1–21, 2000.

K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. J. on Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

K. Makino and M. Berz. Cosy infinity version 9. *Nuclear Instruments and Methods*, A558:346–350, 2006.

O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600, pages 447–484. Springer-Verlag, 1991.

S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Transactions in Embedded Computing Systems*, 6(1), 2007.

B. I. Silva, O. Stursberg, B. H. Krogh, and S. Engell. An assessment of the current status of algorithmic approaches to the verification of hybrid systems. In *Proceedings of the Fortieth IEEE Conference on Decision and Control (CDC '01)*, pages 2867–2874, 2001.

A. Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008.

C.J. Tomlin, I. Mitchell, A.M. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.

S. Yovine. Kronos: a verification tool for real-time systems. *Int. J. on Software Tools for Technology Transfer*, 1(1–2):123–133, 1997.