Chapter 39
An Introduction to the Verification of Hybrid Systems Using ARIADNE

Davide Bresolin, Luca Geretti, Tiziano Villa and Pieter Collins

39.1 Introduction

Hybrid systems are dynamical systems that exhibit both a discrete and a continuous behaviors. To model and specify hybrid systems in a formal way, the notion of *hybrid automata* has been introduced [1]. Intuitively, a hybrid automaton is a "finite-state automaton" with continuous variables that evolve according to dynamics characterizing each discrete state (called a *location* or *mode*). Of particular importance in the analysis of a hybrid automaton is the computation of the *reachable set*, i.e., the set of all states that can be reached under the dynamical evolution starting from a given initial state set. Many approximation techniques and tools to estimate the reachable set have been proposed in the literature. Most of the available software packages, like PhaVer [8] and SpaceEx [9], are limited to affine dynamics. Others, like HSOLVER [10] can handle non-linear dynamics, but can verify only safety properties.

To overcome the limitations of current tools, we recently proposed a development environment for hybrid systems verification, called ARIADNE [3–5], which differs

D. Bresolin (✉) · L. Geretti · T. Villa
Dipartimento di Informatica, Università di Verona, 37134 Verona, Italy
e-mail: davide.bresolin@univr.it

L. Geretti
e-mail: luca.geretti@univr.it

T. Villa
e-mail: tiziano.villa@univr.it

P. Collins
Department of Knowledge Engineering, Maastricht University,
6211 LH Maastricht, The Netherlands
e-mail: pieter.collins@maastrichtuniversity.nl

J.H. van Schuppen and T. Villa (eds.), *Coordination Control of Distributed Systems*,
Lecture Notes in Control and Information Sciences 456,
DOI 10.1007/978-3-319-10407-2_39

from existing tools by being based on a rigorous function calculus [7]. This calculus provides a rigorous mathematical semantics for the numerical analysis of dynamical systems, suitable for implementing formal verification algorithms.

39.2 The Reachability Problem for Hybrid Automata

We first give a formal definition of a hybrid automaton.

Definition 39.1 A *hybrid automaton* is a tuple $\mathscr{A} = \langle \mathrm{Loc}, \mathrm{Edg}, \mathbb{R}^n, Inv, Dyn, Act, Res\rangle$ such that:

1. $\langle \mathrm{Loc}, \mathrm{Edg}\rangle$ is a finite directed graph; the vertices, Loc, are called *locations* or *control modes*, and the directed edges, Edg, are called *control switches*;
2. each location $q \in \mathrm{Loc}$ is labeled by the *invariant condition* Inv[q] on $\mathbb{R}^n$ and the *dynamic law* Dyn[q] on $\mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^{\geq 0}$ such that if Inv[q](x) is true then Dyn[q](x, x, 0) is true;
3. each edge $e \in \mathrm{Edg}$ is labeled by the *activation condition* Act[e] on $\mathbb{R}^n$ and the *reset relation* Res[e] on $\mathbb{R}^n \times \mathbb{R}^n$.

A *state* ℓ of a hybrid automaton $\mathscr{A}$ is a pair , vx, where $v \in \mathrm{Loc}$ is a location and $x \in \mathbb{R}^n$ is an assignment of values for the continuous variables. A state vx is said to be *admissible* if Inv[v](x) holds.

A *trajectory* ξ of a hybrid automaton is a (finite or infinite) sequence $(\xi_i)_{i\geq 0}$ of continuous functions $\xi_i : [\tau_i, \tau_{i+1}] \to \mathrm{Loc} \times \mathbb{R}^n$ such that $\mathrm{Dyn[q]}(\xi_\mathrm{i}(\mathrm{s}), \xi_\mathrm{i}(t), t - \mathrm{s})$ holds for all $\tau_i \leq s \leq t \leq \tau_{i+1}$, and both $\mathrm{Act[e]}(\xi_\mathrm{i}(\tau_{\mathrm{i}+1}))$ and $\mathrm{Res[e]}(\xi_\mathrm{i}(\tau_{\mathrm{i}+1}), \xi_{i+i}(\tau_{i+1}))$ hold for some $e \in \mathrm{Edg}$. Here, $\xi_i(t)$ represents the state of the system after i events and at time t.

Definition 39.2 Let $\mathscr{A}$ be a hybrid automaton. A state $\langle q_r r\rangle$ *reaches* a state $q_s s$ if there exists a finite trajectory $\xi = (\xi_i)_{0\leq i\leq n}$ such that $\xi_0(0) = q_r r$ and $\xi_n(\tau_{n+1}) = q_s s$. We use $\mathit{ReachSet}_{\mathscr{A}}(q_r r)$ to denote the set of states reachable from $q_r r$. Moreover, given a set of states $X_0 \subseteq \mathrm{Loc} \times \mathbb{R}^n$, we use $\mathit{ReachSet}_{\mathscr{A}}(X_0)$ to denote the set $\cup_{q_r r \in X_0} \mathit{ReachSet}_{\mathscr{A}}(q_r r)$.

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 X_0 of initial states. Then, we only need to prove that $\mathit{ReachSet}_{\mathscr{A}}(X_0) \subseteq \mathrm{Sat}(\varphi)$, where $\mathrm{Sat}(\varphi)$ is the set of states where φ is true. Unfortunately, the reachability problem is not decidable in general [1]. Nevertheless, formal verification methods can be applied to hybrid automata: Suppose we can compute an over-approximation S to $\mathit{ReachSet}_{\mathscr{A}}(X_0)$, that is, a set $S \supseteq \mathit{ReachSet}_{\mathscr{A}}(X_0)$. Then, if S is a subset of $\mathrm{Sat}(\varphi)$, then so is the reachable set and the automaton H respects the property. Conversely, if we can compute an under-approximation S to $\mathit{ReachSet}_{\mathscr{A}}(X_0)$ (that is, a set $S \subseteq \mathit{ReachSet}_{\mathscr{A}}(X_0)$) that turns out to contain at least one point outside $\mathrm{Sat}(\varphi)$, we have proved that H does not respect the safety property φ.

39.3 The Ariadne Software Package

Ariadne's computational engine is based on a rigorous *function calculus*, where continuous functions $f : \mathbb{R}^m \mapsto \mathbb{R}^n$ are the basic building blocks used to represent and compute the evolution of hybrid automata [7]. Every component of a hybrid automaton $\mathscr{A}$ can be represented in the function calculus setting as follows:

- For every discrete location, a function $\mathrm{Dyn} : \mathbb{R}^n \mapsto \mathbb{R}^n$ is used to represent the continuous dynamics $\dot{x} = \mathrm{Dyn}(x)$.
- Invariants are represented using single-valued functions $\mathrm{Inv} : \mathbb{R}^n \mapsto \mathbb{R}$ that are *negative* exactly when the invariant is true.
- Discrete transitions are represented using a function $\mathrm{Act} : \mathbb{R}^n \mapsto \mathbb{R}$ that is *positive* when the guard of the transition is true (and negative otherwise), and a reset function $\mathrm{Res} : \mathbb{R}^n \mapsto \mathbb{R}^n$.

Additionally:

- Regions of space $R \subseteq \mathbb{R}^n$ are represented using *ImageSets*, i.e., functions mapping a box $[-1, 1]^p$ to a subset of $\mathbb{R}^n$ that approximates the desired region. [1]

In particular, given $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, and a point $x \in \mathbb{R}^m$, an *approximation* of f near x can be computed. The result is a pair $\hat{f} = (T, I)$ where T is a *polynomial expansion* of f to a given degree and I an interval such that $f(z) - T(z) \in I$ for all points z near x, giving an *error bound* on the approximation. We also call this set an *enclosure*, since it encloses the exact set of points. If we express a starting set S as an enclosure, the evolution under the continuous dynamics can be obtained by numerical integration, using the *flow tube* of the continuous evolution: A function $flow : \mathbb{R}^{n+1} \mapsto \mathbb{R}^n$ such that $flow(S, t)$ is the set of points reached from S after t time units of continuous evolution; taking also discrete evolution into account, we call *reached set* the set of points reached from S after t time units.

39.3.1 Evolution of Enclosures

Ariadne is able to compute approximations to the reachable set by "patching together" enclosures of the reached sets obtained by evolving the system for finite time intervals. Such evolution uses either an *upper semantics* or a *lower semantics*:

- upper semantics implies that, if we evolve the system for a finite time, the set of points that we obtain is a superset of the reachable set;
- lower semantics implies that, if we evolve the system for a finite time, each point that we obtain has a bounded distance to a point of the reachable set.

[1] *ImageSets* are used in the stable version of Ariadne. The development version uses a more accurate representation based on *ConstrainedImageSets* [7].

While the computation of evolution for a finite time is straightforward, the same could not be said for the case of infinite time. To do that, we need to be able to perform the intersection and subtraction of sets. Unfortunately, these two operations cannot be performed on enclosures, which instead must be *discretised* onto a set of *cells* according to a *grid*.

Given a hybrid automaton $\mathscr{A}$, and an initial set of states S_0, ARIADNE can compute two kinds of approximations to the reachable set:

- An *outer approximation* O of the reachable set using upper semantics, for both finite and infinite time evolution. Formally, a *closed set* O such that the *closure* of $\mathit{ReachSet}_{\mathscr{A}}(S_0)$ is strictly contained in the *interior* of O.
- An ε*-lower approximation* L_ε of the reachable set using lower semantics, for both finite and infinite time evolution. Formally, an *open set* L_ε where for every point $x \in L_\varepsilon$ there exists a point $y \in \mathit{ReachSet}_{\mathscr{A}}(S_0)$ such that $|x - y| < \varepsilon$.

In the case of O, the evolution can be performed either in the *forward* or *backward* direction. On the contrary, backward evolution for L_ε, while computable, has not been implemented, since lower semantics causes backward transitions to yield very coarse results that become ineffective for reachability analysis.

39.3.2 Verification

Verification in ARIADNE relies on reachability analysis. ARIADNE currently offers two classes of verification routines: *safety* and *dominance*. The safety verification routine accepts a space region in which the reachable set should be included for all times. The dominance checking routine instead compares the reachable sets of two hybrid systems on a common subspace and decides which one is included into the other. Both routines can be applied to a specific hybrid automaton or be *parametric*. Parametric safety and parametric dominance verification identify some constants of the hybrid automaton and treat them as parameters that take values within a given interval: The routines split the parametric space and verify each subspace. This approach is able to identify optima for design parameters of a system.

All verification routines based on approximations are necessarily dependent on the coarseness of the approximation: An answer to the verification problem may be unattainable only because the accuracy is insufficient to the task. ARIADNE defines the accuracy of computation by means of some *settings*, the most important being:

- the grid used for each location, to control the granularity of the state space;
- the integration step, to control the accuracy of evaluation of the flow function.

In particular, the output of the reachability routines converges to the "best possible" approximations when the accuracy settings converge to zero. Now, since discretised evolution routines are built upon the evolution of enclosures, and verification makes use of approximations obtained using discretised evolution, it is apparent that efficiency of verification (and effectiveness, if we have a limited time to obtain a definite answer) depends on a proper choice of such accuracy settings. Since we cannot decide beforehand which values are optimal for a given system and verification task, we must resort to an *iterative refinement* procedure: if we do not obtain a result with the current accuracy "level," we repeat the calculation of the approximation for finer values of the settings.

A proper manual tuning of the accuracy settings would be quite difficult, especially if iterative refinement is considered. In other words, it is desirable to automate the choice of the accuracy settings in order to improve both usability and efficiency. ARIADNE, therefore, does not require the user to tune these settings: Instead, it extracts reasonable values after a pre-analysis of the domain, at each refinement step. This implies that the choice of the domain for the state space is the only mandatory information that must be provided together with the hybrid automaton.

39.4 The Water Tank Example

In the following, we consider an application from hydraulic control, i.e., a water tank system composed of a cylindrical *tank*, equipped with an inlet pipe at the top, an outlet pipe at the bottom, and a valve that controls the inlet flow. The outlet flow is proportional to the water level, while the inlet flow is controlled by a *valve* that receives *open* and *close* position commands from a controller. In response to a command, the valve aperture $\alpha(t)$ changes linearly in time with a rate $1/T$. The inlet flow is proportional to the inlet pressure $p(t)$ and to the valve aperture $\alpha(t) \in [0, 1]$:

$$F_{in}(t) = \alpha(t) f(p(t)).$$

Location q_1 of Fig. 39.1a represents the nominal state of the tank, when the water level is under the overflow limit, while location q_2 represents overflow. When overflow occurs, the automaton stays in location q_2 until the inlet flow $u(t)$ is less than or equal to the outlet flow $\lambda\sqrt{H}$.

The current water level $x(t)$ is measured by a sensor that outputs a signal $x_s(t)$ affected by an unknown sensor error $\delta(t)$:

$$x_s(t) = x(t) + \delta(t), \tag{39.1}$$

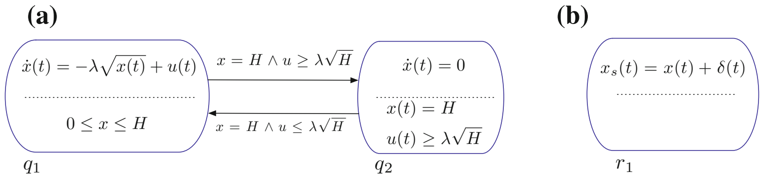
(a)
(b)
$\dot{x}(t) = -\lambda\sqrt{x(t)} + u(t)$
$0 \le x \le H$
$x = H \wedge u \ge \lambda\sqrt{H}$
$x = H \wedge u \le \lambda\sqrt{H}$
$\dot{x}(t) = 0$
$x(t) = H$
$u(t) \ge \lambda\sqrt{H}$
q_1
q_2
$x_s(t) = x(t) + \delta(t)$
r_1

Fig. 39.1 The hybrid automata for the tank and the sensor. **a** Tank. **b** Sensor

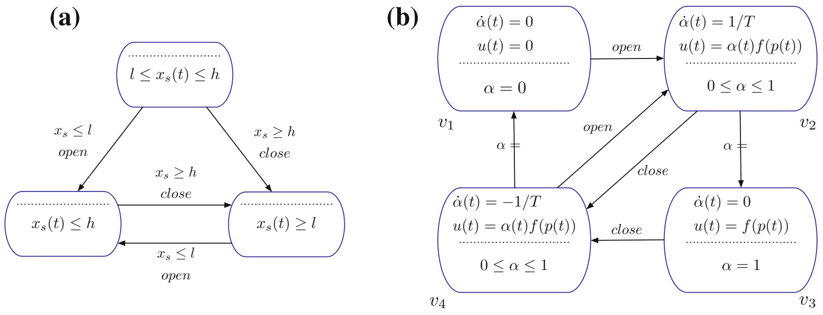

Fig. 39.2 The hybrid automata for the controller and the valve. **a** Hysteretic controller. **b** Valve

and can be modeled by the single location automaton of Fig. 39.1b, where x and δ are input variables, and x_s is the only output variable.

The automaton for the controller is depicted in Fig. 39.2a. It reads the water level $x_s(t)$ measured by the sensor and sends the position commands *open* and *close* to the valve following a simple hysteretic loop:

- when the valve is closed and the water level is decreasing, the *open* command is produced when $x_s(t) \le l$ (location c_3);
- conversely, when the valve is opened and the water level is increasing, the *close* command is produced when $x_s(t) \ge h$ (location c_2).

where l and h represent *lower* and *upper* threshold values for the water level. Location c_1 is the initial location, corresponding to the situation in which the controller does not know whether the water level is increasing or decreasing. The automaton has no output variables, two output events *open* and *close*, and one input variable x_s.

ARIADNE allows to specify these systems separately, and then automatically compose them into a monolithic automaton for evolution and verification.

In this particular example, given an initial water level $x_i \in [6.5,\ 7.0]$, we want to verify whether $x \in [5.25,\ 8.25]$, where $l \in [5.25,\ 6.25]$ and $h \in [7.25,\ 8.25]$. This is a parametric safety verification problem, with parameters l and h. The algorithm offered by the ARIADNE library splits the parameter space with a granularity chosen by the user; then for each subspace, a verification loop is performed, in which we progressively refine the accuracy settings until a definite answer is obtained (or a user-defined time budget is hit).

It must be noticed that if we reach the minimum allowed values for the accuracy settings without getting a positive or negative answer, then an *indeterminate* result is returned. Figure 39.3 shows the verification outcomes for this problem, where the squares represent the verification subspaces. A safe result is shown in green, an unsafe result in red and an indeterminate one in yellow. It can be noticed that low values of h and high values of l are required to provide a positive answer.

Finally, in Fig. 39.4, we show the result for approximations to the reachable set as computed for two different values of the two parameters. The green region represents

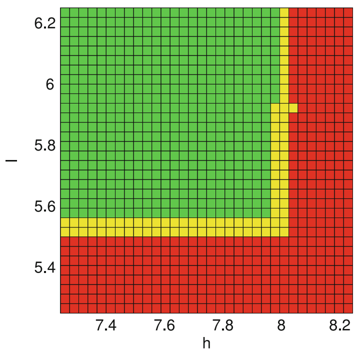

Fig. 39.3 The verification results for the contract satisfaction and dominance problems

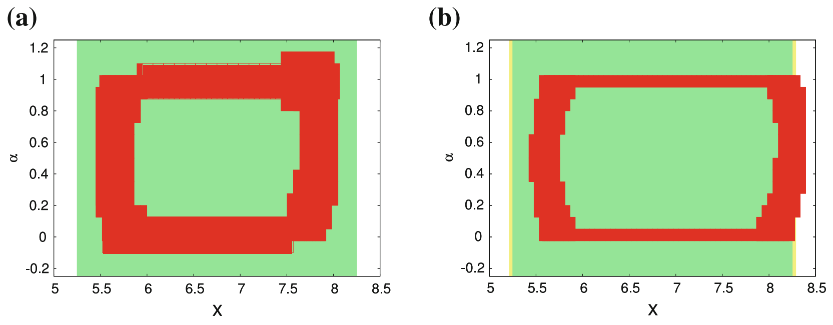

Fig. 39.4 Reachable set for two choices of the l and h thresholds, with a safe (**a**) and unsafe (**b**) result. **a** $l = [5.75,\ 5.781]$, $h = [7.75,\ 7.781]$. **b** $l = [5.75,\ 5.781]$, $h = [8.125,\ 8.156]$

the safe region, while the red region is the computed approximation. In Fig. 39.4a, the O approximation is used to verify that safety is guaranteed. In Fig. 39.4b, the L_ε approximation allows us to state that the system is unsafe. The yellow region represents the ε-tolerance on the safe region given by the approximation error. These results look rather coarse, but this is due to the fact that we need to periodically discretise the reached set onto a grid in order to decide when to stop evolving.

39.5 Conclusions and Future Work

Formal verification of hybrid systems is still in its infancy, but tools like ARIADNE show promising results also on non-trivial case studies (see Chap. 40). Further information on the framework can be found in [7] about functional calculus, [4] regarding the reachability routines, and [5] for advanced verification strategies.

ARIADNE can manage non-linear dynamics and can compute both outer and lower approximations to the reachable set. However, this high expressivity is also the main reason for some shortcomings, in particular with respect to scalability and accuracy of the approximations. The recent introduction of support functions increased substantially the size of linear hybrid systems that can be verified [9] and showed that the choice of the correct set representation is crucial. To overcome the current limitations of the tool, we are working on the following improvements: addressing scalability by means of counter-example based abstraction refinement techniques [2], handling specification properties beyond safety [6], extending the tool to synthesize switching controllers with respect to safety properties [11].

References

1. Alur R, Courcoubetis C, Henzinger TA, Ho PH (1992) Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In: Hybrid systems, LNCS. Springer, Berlin, pp 209–229
2. Alur Rajeev, Dang Thao, Ivančić Franjo (2006) Counterexample-guided predicate abstraction of hybrid systems. Theor Comput Sci 354(2):250–271
3. Ariadne: An open tool for hybrid system analysis. http://ariadne.parades.rm.cnr.it
4. Benvenuti L, Bresolin D, Collins P, Ferrari A, Geretti L, Villa T (2012) Ariadne: Dominance checking of nonlinear hybrid automata using reachability analysis. Reachability Problems., volume 7550 of LNCSSpringer, Berlin Heidelberg, pp 79–91
5. Benvenuti L, Bresolin D, Collins P, Ferrari A, Geretti L, Villa T (2014) Assume-guarantee verification of nonlinear hybrid systems with ARIADNE. Int J Robust Nonlinear Control 24(4):699–724
6. Bresolin D (2013) Improving HyLTL model checking of hybrid systems. In: Proceedings of the 4th international symposium on games, automata, logics and formal verification (GandALF2013), vol 119 of EPTCS, pp 79–92
7. Collins P, Bresolin D, Geretti L, Villa T (2012) Computing the evolution of hybrid systems using rigorous function calculus. In: Proceedings of the 4th IFAC conference on analysis and design of Hybrid Systems (ADHS12), pp 284–290
8. Frehse G (2008) PHAVer: algorithmic verification of hybrid systems past HyTech. Int J Softw Tools Technol Transf (STTT) 10:263–279
9. Frehse G, Le Guernic C, Donzé A, Cotton S, Ray R, Lebeltel O, Ripado R, Girard A, Dang T, Maler O (2011) SpaceEx: scalable verification of hybrid systems. In: Proceedings 23rd international conference on computer aided verification (CAV 2011), volume 6806 of LNCS. Springer, Berlin, pp 379–395
10. Ratschan S, She Z (2007) Safety verification of hybrid systems by constraint propagation based abstraction refinement. ACM Trans Embed Comput Syst 6(1)
11. Tomlin CJ, Lygeros J, Sastry SS (2000) A game theoretic approach to controller design for hybrid systems. Proc IEEE 88(7):949–970