INTERNATIONAL JOURNAL OF ROBUST AND NONLINEAR CONTROL
Int. J. Robust. Nonlinear Control 0000; **00**:1–31
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/rnc

Assume-Guarantee Verification of Nonlinear Hybrid Systems with ARIADNE

Luca Benvenuti[1], Davide Bresolin[2*], Pieter Collins[3], Alberto Ferrari[4], Luca Geretti[2] and Tiziano Villa[2]

[1] *Università di Roma "La Sapienza", Roma, Italy, luca.benvenuti@uniroma1.it*
[2] *Università di Verona, Verona, Italy, {davide.bresolin, luca.geretti, tiziano.villa}@univr.it*
[3] *Maastricht University, Maastricht, The Netherlands, pieter.collins@maastrichtuniversity.nl*
[4] *ALES S.r.l., Roma, Italy, alberto.ferrari@ales.eu.com*

SUMMARY

In many applicative fields there is the need to model and design complex systems having a mixed discrete and continuous behaviour that cannot be characterized faithfully using either discrete or continuous models only. Such systems consist of a discrete control part that operates in a continuous environment and are named hybrid systems because of their mixed nature. Unfortunately, most of the verification problems for hybrid systems, like reachability analysis, turn out to be undecidable. Because of this, many approximation techniques and tools to estimate the reachable set have been proposed in the literature. However, most of the tools are unable to handle nonlinear dynamics and constraints and have restrictive licenses. To overcome these limitations, we recently proposed an open-source framework for hybrid system verification, called ARIADNE, which exploits approximation techniques based on the theory of computable analysis for implementing formal verification algorithms. In this paper we will show how the approximation capabilities of ARIADNE can be used to verify complex hybrid systems, adopting an assume-guarantee reasoning approach. Copyright © 0000 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: Hybrid Systems; Verification

1. INTRODUCTION

Hybrid systems exhibit both a discrete and a continuous behaviour with a tight interaction between the two; such are, for instance, discrete controllers that operate in a continuous environment, like automotive powertrain systems, where a four stroke engine is modeled by a switching continuous system and is controlled by a digital controller. In order to model and specify hybrid systems in a formal way, the notion of *hybrid automata* has been introduced [1, 2]. Intuitively, a hybrid automaton is a "finite-state automaton" with continuous variables that evolve according to dynamics

*Correspondence to: Dipartimento di Informatica, Università di Verona, Strada Le Grazie 15, 37134, Verona, Italy. E-mail: davide.bresolin@univr.it 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. The state of a hybrid automaton consists of the pairing of a discrete location with a vector of continuous variables, therefore it has the cardinality of continuum. This makes the analysis computationally difficult. Indeed, the reachable set is, in general, not computable exactly. Moreover, 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 [3].

Many different approaches have been used in the literature to develop tools that can compute or approximate reachable sets for hybrid systems. Tools like Kronos [4] and UPPAAL [5] compute the reachability relation for systems based on timed automata, a simple class of hybrid systems where the reachable set can be computed exactly. HyTech [6] computes approximations to the reachable set for hybrid automata with piecewise-constant continuous dynamics. Other tools, such as `d/dt` [7], VeriShift [8], and HybridSal [9], can handle also affine systems. PhaVer [10] uses polytopes to represent the state space, can handle affine dynamics and guards and allows to set an arbitrary level of precision. Termination of an infinite-time reachability computation is performed by testing if no new enclosures are generated; this means that the enclosures used are necessarily coarse approximations. SpaceEx [11] is a modular open-source framework that improves upon PhaVer, with particular regard to scalability. It combines polyhedra and support function representations of the state space of systems with piecewise affine, non-deterministic dynamics. Local error bounds on the computation are guaranteed using variable time steps; however, differently from PhaVer, the results of SpaceEx are not numerically sound. CheckMate [12] and HSOLVER [13] can compute over approximations to the reachable set for hybrid automata with non-linear dynamics, but can verify only safety properties. We are aware of only two tools that can perform parametric verification of hybrid systems to discover the safe values for system parameters. One of them is KeYmaera [14], that uses automated theorem proving techniques to verify nonlinear hybrid systems symbolically, without computing approximations of the reachable set. The other one is Breach [15], that uses simulation-based techniques to estimate the safe values of the parameters.

Additionally, general-purpose tools for set-based analysis, such as GAIO [16], COSY Infinity [17] and Mitchell's Toolbox of Level Set Methods [18] may be used. These tools also include many interesting features such as model checking capabilities or graphical modeling interfaces.

However, most of the available tools can only prove safety properties, do not have any disproving capability, are unable to handle nonlinear dynamics and constraints and have restrictive licences, where sometimes their source code is closed. Without access to the code, users can neither customize or optimize them for a specific class of instances of the reachability problem, nor check that the algorithms are implemented correctly.

To overcome these limitations, we recently proposed a framework for hybrid systems verification, called ARIADNE, currently under development by a joint team from the University of Maastricht and the Centrum voor Wiskunde en Informatica (Amsterdam), the University of Verona, the University of Udine and the company PARADES/ALES (Rome). ARIADNE is a development environment in which to program data structures and algorithms for reachability analysis of hybrid systems; it differs from existing packages for the analysis of hybrid systems since it is based on the theory of computable analysis [19]. Such theory provides a rigorous mathematical semantics for the numerical analysis of continuous and hybrid systems, suitable for implementing formally sound verification algorithms. The computational kernel of ARIADNE is written using a mix of generic and polymorphic programming strategies resulting in a highly efficient, modular and extensible framework. The package is released as an open source distribution, so that different research groups may contribute new data structures and algorithms ([20], `crete-bdd` branch of the code repository).

In this paper we adopt an assume-guarantee reasoning methodology to verify hybrid systems using ARIADNE. In this approach the system is first decomposed into smaller components, equipped with assumptions about the surrounding environment and with guarantees about their behavior. The pair assumptions-guarantees is usually called the "contract" that the component must respect. In this way, the requirements of the whole system can be decomposed into a set of simpler requirements that, if satisfied, guarantee that the overall requirement is satisfied. Every component is then verified separately and, if all components satisfy their "local" requirements, then the composition is well formed and the whole system is correct. We focus in particular on two problems: the satisfaction of a contract by a system, and the dominance of a first system over a second system in respect to a contract. The verification is performed in a parametric way, meaning that some of the parameters describing a system's behavior can range over intervals, to model imperfect knowledge and the possible choices for the design parameters.

The rest of the paper is organized as follows. Section 2 introduces the hybrid automata formalism. Section 3 shows how it can be used to model a case-study coming from fluid dynamics. In Section 4 we recall some fundamental results on computable analysis for hybrid automata. Section 5 describes ARIADNE and its approximation capabilities. The assume-guarantee metodology that we use to verify hybrid systems is defined in Section 6, and then applied to verify the case-study in Section 7. Section 8 concludes the paper with some open problems and future research directions.

2. HYBRID AUTOMATA: SYNTAX AND SEMANTICS

An hybrid automaton is a finite state machine enriched with continuous dynamics labelling each discrete state (or *location*), that alternates continuous and discrete evolution. In continuous evolution, the discrete state does not change, while time passes and the evolution of the continuous state variables follows the dynamic law associated to the current location. A discrete evolution step consists of the activation of a *discrete transition* that can change both the current location and the value of the state variables, in accordance with the reset function associated to the transition. The interleaving of continuous and discrete evolution is decided by the *invariant* of the location, which must be true for the continuous evolution to keep on going, and by the *guard* predicates, which must be true for a discrete transition to be activated. Guards and invariants are not necessarily complements of each other: when both the invariant and one or more guards are true, both the continuous evolution and the activation of the discrete transitions is allowed and the behaviour of the automaton becomes non-deterministic.

Our definition of hybrid automata extends the one given in [1] to support the compositional description and analysis of systems, when they are too complex to be understood all at once and must be decomposed. To this end, we introduce a distinction between input, output and internal variables and actions as in [21]. Input and output variables and events describe the continuous and discrete interactions of a component with the environment and the other components of the system, thus defining how components can be put together to make complex systems out of simpler ones. In Section 6 we exploit this distinction to define a verification strategy, based on the assume-guarantee paradigm, that allows to break up the verification of a system into smaller tasks that involve the verification of the single components instead of the whole system.

Before formally defining the syntax and semantics of hybrid automata we need to introduce some basic terminology. Throughout the paper we fix the *time axis* to be the set of non-negative real numbers $\mathbb{R}^+$. An *interval* I is any convex subset of $\mathbb{R}^+$, usually denoted as $[t_1, t_2] = \{t \in \mathbb{R}^+ : t_1 \leq t \leq t_2\}$. For any interval I and $t \in \mathbb{R}^+$, we define $I + t$ as the interval $\{t' + t : t' \in I\}$.

We also fix a countable universal set $\mathcal{V}$ of *variables*, ranging over the reals. Given a set of variables $X \subseteq \mathcal{V}$, a *valuation* over X is a function $\mathbf{x} : X \mapsto \mathbb{R}$ that associates a value to every variable in X. The set $\mathrm{Val}(X)$ is the set of all valuations over X. Given a valuation $\mathbf{x}$ and a subset of variables $Y \subseteq X$, we denote the *restriction* of $\mathbf{x}$ to Y as $\mathbf{x}\!\downarrow\! Y$. Symmetrically, given a valuation $\mathbf{x}$ and a superset of variables $Z \supseteq X$, the *extension* of $\mathbf{x}$ to Z is denoted as $\mathbf{x}\!\uparrow\! Z$ and defined as the set $\{\mathbf{z} \in \mathrm{Val}(Z) : \mathbf{z}\!\downarrow\! X = \mathbf{x}\}$. The restriction and extension operators are extended to sets of valuations in the usual way. A valuation $\mathbf{x}$ over X and a valuation $\mathbf{y}$ over Y *agree* when they assign the same value to common variables, i.e., $\mathbf{x}\!\downarrow\! X \cap Y = \mathbf{y}\!\downarrow\! X \cap Y$. When valuations $\mathbf{x}$ over X and $\mathbf{y}$ over Y agree, we denote by $\mathbf{x} \sqcup \mathbf{y}$ the *union* of $\mathbf{x}$ and $\mathbf{y}$, defined as the valuation $\mathbf{z}$ such that $\mathbf{z}\!\downarrow\! X = \mathbf{x}$ and $\mathbf{z}\!\downarrow\! Y = \mathbf{y}$. Notice that valuations over disjoint sets of variables always agree, and thus their union is always defined.

We are now ready to formally define what is a hybrid automaton.

Definition 1

A *hybrid automaton* is a tuple $\mathcal{H} = \langle \mathrm{Loc}, (U, X, Y), (I, A, O), \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Inv}, \mathrm{Act}, \mathrm{Res} \rangle$ such that:

1. Loc is a finite set of *locations*;
2. U, X, and Y are three finite sets of *input*, *internal*, and *output* variables, disjoint from each other. Variables in $Z = X \cup Y$ are called *locally controlled*, while variables in $W = U \cup Y$ are called *external*. We define $V = U \cup X \cup Y$;
3. I, A, O are three finite sets of *input*, *internal*, and *output* actions, disjoint from each other. We assume that I contains a special action ε, which represents the *environment action*. Actions in $L = A \cup O$ are called *locally controlled*, while actions in $H = I \cup O$ are called *external*. We define $E = I \cup A \cup O$;
4. $\mathrm{Edg} \subseteq \mathrm{Loc} \times E \times \mathrm{Loc}$ is a set of *discrete transitions* such that for every location $l \in \mathrm{Loc}$ and input action $i \in I$ there exists $(l, i, l') \in \mathrm{Edg}$, i.e., for every input action a transition must be defined. For every location l, $(l, \varepsilon, l) \in \mathrm{Edg}$ and if $l \neq l'$, $(l, \varepsilon, l') \notin \mathrm{Edg}$;
5. Dyn is a mapping that associates to every location $l \in \mathrm{Loc}$ a function $\mathrm{Dyn}(l) : \mathrm{Val}(V) \mapsto \mathrm{Val}(Z)$ describing the *dynamics* of l;
6. $\mathrm{Inv} \subseteq \mathrm{Loc} \times \mathrm{Val}(V)$ is a set of *invariants*. For every location $l \in \mathrm{Loc}$, we denote by $\mathrm{Inv}(l)$ the set $\{\mathbf{v} \in \mathrm{Val}(V) : (l, \mathbf{v}) \in \mathrm{Inv}\}$;
7. Act is a mapping that associates to every discrete transition $(l, e, l') \in \mathrm{Edg}$ an *activation set* $\mathrm{Act}(l, e, l') \subseteq \mathrm{Val}(V)$. For every input action $i \in I$ and every transition $(l, i, l') \in \mathrm{Edg}$ and

every $\mathbf{v} \in \mathrm{Val}(V)$, we require $\mathbf{v} \in \mathrm{Act}(l, i, l')$, i.e., the predicate enabling the transition must be true;

8. Res is a mapping that associates every discrete transition $(l, e, l') \in \mathrm{Edg}$ with its *reset function* $\mathrm{Res}(l, e, l') : \mathrm{Val}(V) \mapsto \mathrm{Val}(Z)$. For every transition $(l, \varepsilon, l) \in \mathrm{Edg}$ we impose $\mathrm{Res}(l, \varepsilon, l)(\mathbf{v}) = \mathbf{v} \downarrow Z$ for every $\mathbf{v} \in \mathrm{Val}(V)$.

When the set of input variables and of input actions of a hybrid automaton is empty (except for the ε action), the behaviour of the automaton is not influenced by the surrounding environment. We refer to such an automaton as being *input-free*.

Composition is defined as a partial binary operation on hybrid automata. Given two hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$, we say that they are *compatible* if and only if $A_1 \cap E_2 = A_2 \cap E_1 = \emptyset$, $X_1 \cap V_2 = X_2 \cap V_1 = \emptyset$ (disjointness of internal actions and variables), $Y_1 \cap Y_2 = O_1 \cap O_2 = \emptyset$ (disjointness of output actions and variables). We define the composition as follows.

Definition 2
Given two compatible hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$, we define their composition $\mathcal{H}_1 \| \mathcal{H}_2$ as the hybrid automaton $\mathcal{H} = \langle \mathrm{Loc}, (U, X, Y), (I, A, O), \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Inv}, \mathrm{Act}, \mathrm{Res} \rangle$ such that:

1. $\mathrm{Loc} = \mathrm{Loc}_1 \times \mathrm{Loc}_2$;
2. $Y = Y_1 \cup Y_2$, $U = (U_1 \cup U_2) \setminus Y$, and $X = X_1 \cup X_2$;
3. $O = O_1 \cup O_2$, $I = (I_1 \cup I_2) \setminus O$, $A = A_1 \cup A_2$;
4. $((l_1, l_2), e, (l'_1, l'_2)) \in \mathrm{Edg}$ iff *(i)* $e \in E_i$ and $l_i \xrightarrow{e} l'_i$, or *(ii)* $e \notin E_i$ and $l_i \xrightarrow{\varepsilon} l'_i,\ l_i = l'_i$, for $i = 1, 2$;
5. for every $(l_1, l_2) \in \mathrm{Loc}$ and $\mathbf{v} \in \mathrm{Val}(V)$, we have that $\mathrm{Dyn}((l_1, l_2))(\mathbf{v}) = \mathrm{Dyn}_1(l_1)(\mathbf{v} \downarrow V_1) \sqcup \mathrm{Dyn}_2(l_2)(\mathbf{v} \downarrow V_2)$;
6. $\mathrm{Inv} = \{((l_1, l_2), \mathbf{v}) : (l_1, \mathbf{v} \downarrow V_1) \in \mathrm{Inv}_1 \text{ and } (l_2, \mathbf{v} \downarrow V_2) \in \mathrm{Inv}_2\}$;
7. for every $((l_1, l_2), e, (l'_1, l'_2)) \in \mathrm{Edg}$, we have that:
   (i) $\mathrm{Act}((l_1, l_2), \varepsilon, (l'_1, l'_2)) = \mathrm{Val}(V)$,
   (ii) $\mathrm{Act}((l_1, l_2), e, (l'_1, l'_2)) \downarrow V_1 = \mathrm{Act}_1(l_1, a, l'_1)$, for all $e \in E_1$, and
   (iii) $\mathrm{Act}((l_1, l_2), e, (l'_1, l'_2)) \downarrow V_2 = \mathrm{Act}_2(l_2, e, l'_2)$, for all $e \in E_2$;
8. for every $((l_1, l_2), e, (l'_1, l'_2)) \in \mathrm{Edg}$, and $\mathbf{v} \in \mathrm{Val}(V)$, we have that:
   (i) $e \in E_i$ and $\mathrm{Res}((l_1, l_2), e, (l'_1, l'_2))(\mathbf{v}) \downarrow Z_i = \mathrm{Res}_i(l_i, e, l'_i)$, or
   (ii) $e \notin E_i$ and $\mathrm{Res}((l_1, l_2), e, (l'_1, l'_2))(\mathbf{v}) \downarrow Z_i = \mathbf{v} \downarrow Z_i$,
   for $i = 1, 2$.

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

Definition 3
Let $\mathcal{H}$ be a hybrid automaton. A *state* of $\mathcal{H}$ is a pair $(l, \mathbf{v})$, where $l \in \mathrm{Loc}$ is a location and $\mathbf{v} \in \mathrm{Val}(V)$ is a valuation for the continuous variables. A state $(l, \mathbf{v})$ is said to be *admissible* if $(l, \mathbf{v}) \in \mathrm{Inv}$.

Even though it seems more natural to define the state of a hybrid automaton by considering only controlled variables or internal variables (see, for instance, [21]), we choose to include also input variables to simplify the definition of the semantics. In our formalism, an execution of a hybrid automaton corresponds to a sequence of transitions from a state to another. Transitions can be either *continuous*, capturing the continuous evolution of the state, or *discrete*, capturing instantaneous and discontinuous changes of the state. Formally, they are defined as follows.

Definition 4
Let $\mathcal{H}$ 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:

$$(l, \mathbf{v}) \xrightarrow{t}_C (l, \mathbf{v}') \iff \exists \text{ continuous function } f : t \to V \text{ s.t. } f(0) = \mathbf{v},\ f(t) = \mathbf{v}', \text{ and} \\ \forall\, t' \in [0, t], f(t') \in \mathrm{Inv}(l) \wedge \frac{df}{dt}(t') \downarrow Z = \mathrm{Dyn}(l)(t'). \quad (1)$$

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

$$(l, \mathbf{v}) \xrightarrow{e}_D (l', \mathbf{v}') \iff (l, e, l') \in \mathrm{Edg} \wedge \mathbf{v} \in \mathrm{Inv}(l) \wedge \mathbf{v} \in \mathrm{Act}(l, e, l') \\ \wedge \mathrm{Res}(l, e, l')(\mathbf{v}) = \mathbf{v}' \downarrow Z \wedge \mathbf{v}' \in \mathrm{Inv}(l'). \quad (2)$$

This semantics allows us to define the notion of trajectory of a hybrid automaton and the reachability relation between states and sets of states.

Definition 5
Let $\mathcal{H}$ be a hybrid automaton, and let $(l, \mathbf{v})$ be a state of $\mathcal{H}$. A trajectory ξ of $\mathcal{H}$ from $(l, \mathbf{v})$ is a (finite or infinite) sequence $(l_i, \mathbf{v}_i)_{i \geq 0}$ of states such that $(l_0, \mathbf{v}_0) = (l, \mathbf{v})$ and either $(l_{i-1}, \mathbf{v}_{i-1}) \xrightarrow{t}_C (l_i, \mathbf{v}_i)$ or $(l_{i-1}, \mathbf{v}_{i-1}) \xrightarrow{e}_D (l_i, \mathbf{v}_i)$, for some $t \in \mathbb{R}^{\geq 0}$, and $e \in E$.

Definition 6
Let $\mathcal{H}$ be a hybrid automaton. A state $(l, \mathbf{v})$ *reaches* a state $(l', \mathbf{v}')$ if there exists a finite trajectory $\xi = (l_i, \mathbf{v}_i)_{0 \leq i \leq n}$ such that $(l_0, \mathbf{v}_0) = (l, \mathbf{v})$ and $(l_n, \mathbf{v}_n) = (l', \mathbf{v}')$. We use $\mathit{ReachSet}_{\mathcal{H}}(l, \mathbf{v})$ to denote the set of states reachable from $(l, \mathbf{v})$. Moreover, given a set of states $S_0 \subseteq \mathrm{Loc} \times \mathrm{Val}(V)$ we use $\mathit{ReachSet}_{\mathcal{H}}(S_0)$ to denote the set $\cup_{(l,\mathbf{v}) \in S_0} \mathit{ReachSet}_{\mathcal{H}}(l, \mathbf{v})$.

The above definition introduces the notion of *infinite-time reachable set* for hybrid automata. In this paper we will also consider another notion of reachable set, the so-called *finite-time reachable set*, defined as follows.

Definition 7
Let $\mathcal{H}$ be a hybrid automaton, $t \in \mathbb{R}$ and $n \in \mathbb{N}$. We say that a state $(l, \mathbf{v})$ *reaches a state* $(l', \mathbf{v}')$ *with time* (t, n) if there exists a finite trajectory $\xi = (l, \mathbf{v}) \xrightarrow{t_1}_C (l_1, \mathbf{v}_1) \xrightarrow{e_1}_D (l_2, \mathbf{v}_2) \xrightarrow{t_2}_C \ldots \xrightarrow{e_n}_D (l_{2n}, \mathbf{v}_{2n}) \xrightarrow{t_{n+1}}_C (l', \mathbf{v}')$ such that $\sum_{i=1}^{n+1} t_i = t$. Given a set of states $S_0 \subseteq \mathrm{Loc} \times \mathrm{Val}(V)$ we use $\mathit{ReachSet}_{\mathcal{H}}(S_0, t, n)$ to denote the set of states reachable from S_0 in time (t, n). This definition extends to the case when t and n are intervals over $\mathbb{R}$ and $\mathbb{N}$ in the usual way.

Intuitively, $\mathit{ReachSet}_{\mathcal{H}}(S_0, t, n)$ contains all states reachable from a state in S_0 with a trajectory of total time t and with n discrete transitions.

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 $\mathcal{H}$; in other words, that φ remains true for all possible executions starting from a set S_0 of initial states. Then we only need to prove

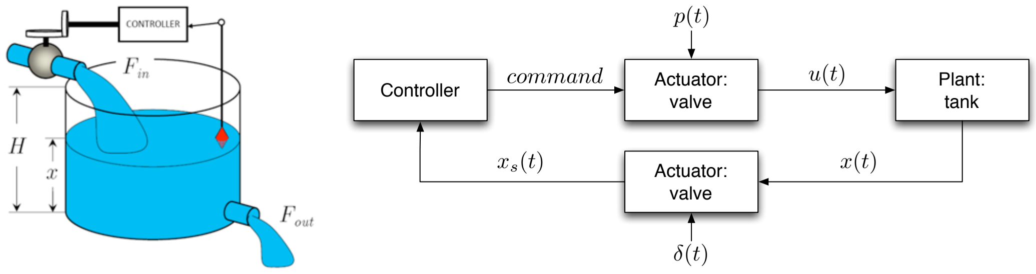


Figure 1. The water tank closed-loop system.

that $ReachSet_{\mathcal{H}}(S_0) \subseteq \mathrm{Sat}(\varphi)$, where $\mathrm{Sat}(\varphi)$ is the set of states where φ is true. Unfortunately, the reachability problem is not decidable in general, as proved in [22] by reducing the halting problem for two-counter machines to the emptiness problem for $ReachSet_{\mathcal{H}}(S_0)$. Hence, many tools for reachability analysis of general hybrid automata are based on approximation techniques. The challenge is to find the "best" approximations of continuous regions. In Section 4 we will show how Weihrauch's theory of computable analysis [19, 23] can be used to formally establish the conditions under which the reachable set of a hybrid automaton can be effectively approximated, and what is the "best" approximation to use.

3. A RUNNING EXAMPLE: THE WATERTANK

To show how the definition of Hybrid Automata given in the previous section can be used to model and verify hybrid systems, we consider an application from fluid dynamics. Suppose we want to design a complex system that controls the flow of some fluid in an industrial plant. Following the compositional approach, the whole system is first decomposed into a set of smaller components, each of which describes a different part of the system.

In this paper we concentrate our attention on a single component of the system: 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. To simplify the description, we provide the tank component as a set of smaller sub-components, depicted in Figure 1.

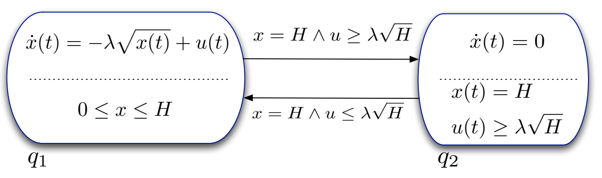
The tank component is modeled by the hybrid automaton in Figure 2(a), featuring the input variable u (representing the inlet flow) and the output variable x (water level). The outlet flow is proportional to the water level x under the nonlinear relation

$$F_{out}(t) = \lambda\sqrt{x(t)}. \tag{3}$$

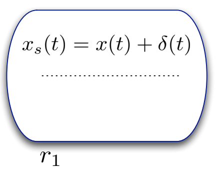
Location q_1 represents the nominal state of the tank, when the water level is under the overflow limit, while location q_2 represents overflow. When overflow occurs, the automaton stays in location q_2 until the inlet flow $u(t)$ is less than or equal to the outlet flow $\lambda\sqrt{H}$.

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

$$x_s(t) = x(t) + \delta(t), \tag{4}$$


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

(a) Tank



(b) Sensor

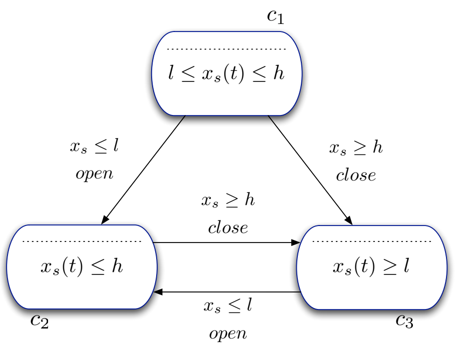
Figure 2. The hybrid automata for the tank and the sensor.

and can be modeled by the single location automaton of Figure 2(b), where x and δ are input variables, and x_s is the only output variable.

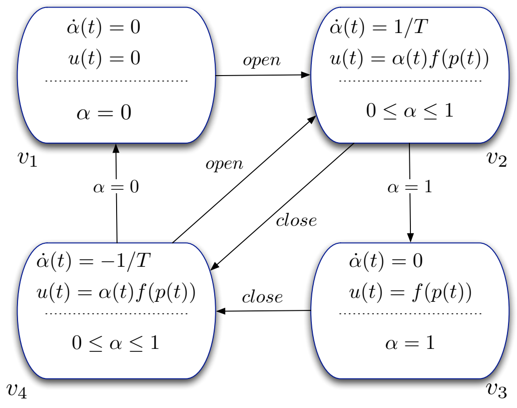
The automaton for the controller is depicted in Figure 3(a). It reads the water level $x_s(t)$ measured by the sensor and sends the position commands *open* and *close* to the valve following a simple hysteresis loop:

- 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).

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 x_s as the only input variable.



(a) Hysteretic controller



(b) Valve

Figure 3. The hybrid automata for the controller and the valve.

The inlet flow is controlled by a *valve* that receives *open* and *close* position commands from the controller. In response to a position 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 $\alpha(t) \in [0, 1]$:

$$u(t) = \alpha(t)f(p(t)). \tag{5}$$

Location v_1 models the dynamics of the inlet flow when the valve is completely closed ($\alpha = 0$), while location v_3 models the dynamics when the valve is completely opened ($\alpha = 1$). Locations

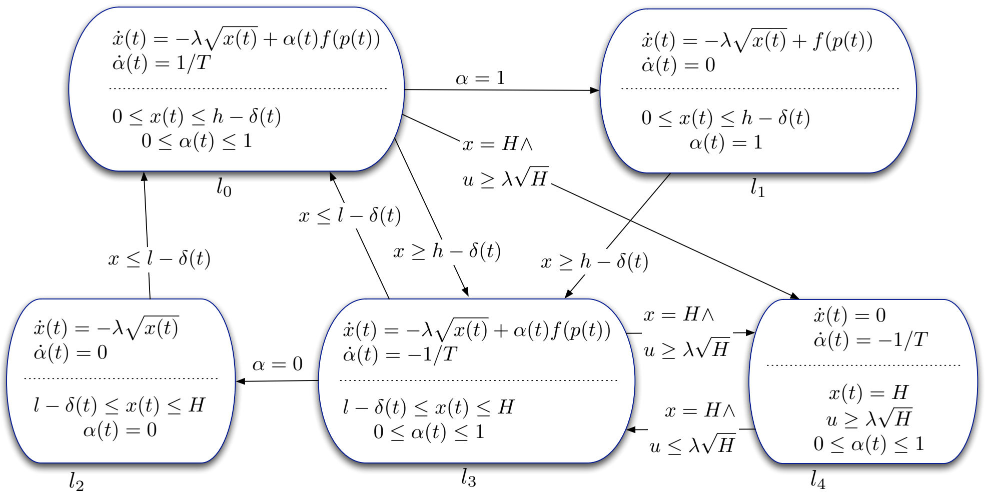


Figure 4. The hybrid automaton $\mathcal{H}_1$ for the watertank system with a hysteretic controller.

v_2 and v_4 correspond to the opening and closing of the valve, respectively. In the definition of the automaton, *open* and *closed* are input events, x_s and p are input variables, u is an output variable, and α an internal one.

Figure 4 depicts the hybrid automaton $\mathcal{H}_1$ that models the entire watertank system, obtained by composing the automata of the single components following Definition 2 and by using the algorithm described in [24] to remove the non-reachable locations. In the composed system the only input variables are the pressure p and the sensor error δ, while the only output variable is the water level x; α remains an internal variable, while the variables x_s and u are replaced by the expressions (4) and (5). Moreover, the discrete actions *open* and *close* became internal, and thus are removed from the picture.

Throughout the paper we will consider also a second version of the watertank system, where the controller and the valve have been replaced with different ones. In particular, with a valve that is slower than the previous one but that is paired with a smarter controller. Instead of sending open and close commands, the new controller can stabilize the valve aperture to any reference value $w(t) \in [0, 1]$. In response to the reference input $w(t)$ coming from the controller, the valve aperture $\alpha(t)$ changes following the first order dynamics $\dot{\alpha}(t) = \frac{1}{\tau}(w(t) - \alpha(t))$, for some constant $\tau > 0$. The controller computes the reference input $w(t)$ for the valve following the proportional control law

$$w(t) = K_P(R - x_s(t)), \text{ when } R - \frac{1}{K_P} \le x_s(t) \le R, \tag{6}$$

where $x_s(t)$ is the water level measure of the sensor and R is a design parameter representing the desired long-term water level. The valve input $w(t)$ is saturated to 1 when $x_s(t) \le R - \frac{1}{K_P}$ and to 0 when $x_s(t) \ge R$, in order to keep the aperture inside the $[0,\ 1]$ interval. Figure 5 depicts the hybrid automaton $\mathcal{H}_2$ modeling the watertank system with the proportional controller. In location c_0 there is no saturation, and the controller follows the given proportional law, while in locations c_1 and c_2 the valve input is saturated to 1 and to 0, respectively. Location c_3 models overflow.

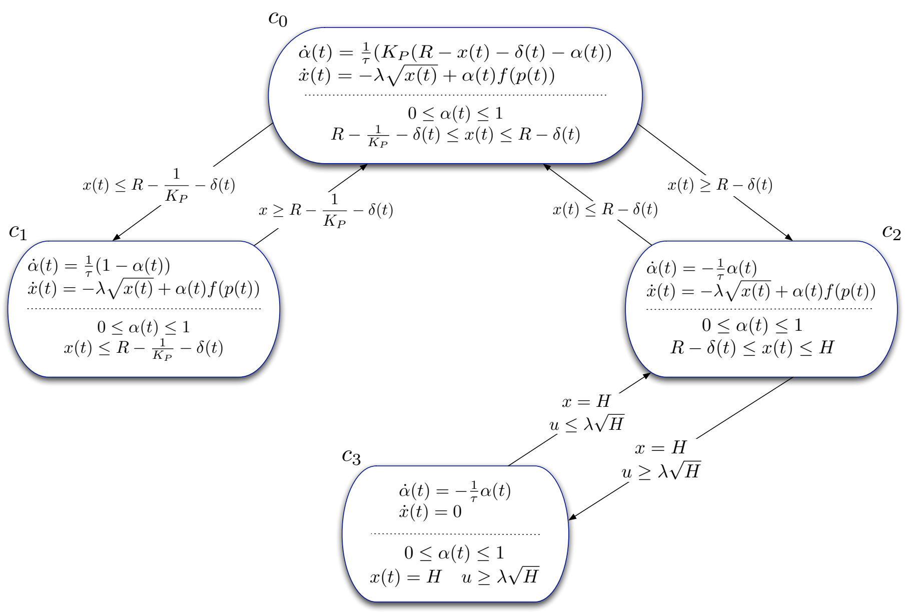

c_0
$\dot\alpha(t) = \frac{1}{\tau}(K_P(R - x(t) - \delta(t) - \alpha(t))$
$\dot x(t) = -\lambda\sqrt{x(t)} + \alpha(t)f(p(t))$
$0 \le \alpha(t) \le 1$
$R - \frac{1}{K_P} - \delta(t) \le x(t) \le R - \delta(t)$
$x(t) \le R - \frac{1}{K_P} - \delta(t)$
$x \ge R - \frac{1}{K_P} - \delta(t)$
$x(t) \le R - \delta(t)$
$x(t) \ge R - \delta(t)$
c_1
$\dot\alpha(t) = \frac{1}{\tau}(1 - \alpha(t))$
$\dot x(t) = -\lambda\sqrt{x(t)} + \alpha(t)f(p(t))$
$0 \le \alpha(t) \le 1$
$x(t) \le R - \frac{1}{K_P} - \delta(t)$
c_2
$\dot\alpha(t) = -\frac{1}{\tau}\alpha(t)$
$\dot x(t) = -\lambda\sqrt{x(t)} + \alpha(t)f(p(t))$
$0 \le \alpha(t) \le 1$
$R - \delta(t) \le x(t) \le H$
$x = H$
$u \le \lambda\sqrt{H}$
$x = H$
$u \ge \lambda\sqrt{H}$
c_3
$\dot\alpha(t) = -\frac{1}{\tau}\alpha(t)$
$\dot x(t) = 0$
$0 \le \alpha(t) \le 1$
$x(t) = H \quad u \ge \lambda\sqrt{H}$

Figure 5. The hybrid automaton $\mathcal{H}_2$ for the watertank system with a proportional controller.

In Section 7 we will show how to solve some meaningful verification problems on the two watertank systems, using the software package ARIADNE.

4. COMPUTABILITY RESULTS FOR HYBRID AUTOMATA

In this section we will recall some basic notions from the theory of *computable analysis* as developed by Weihrauch and co-workers [19], and we will show how it can be applied to find approximate algorithmic solutions to the reachability problem for hybrid automata [25, 26, 27]. In this theory, computation is performed by Turing Machines acting on (infinite) streams of data. These data streams encode a sequence of approximations to some quantity (such as a region of the state space, or a function describing the system). A given function or operator is *computable* in this theory if, given a data stream encoding a *sequence of approximations converging to the input*, it is possible to calculate a data stream encoding a *sequence of approximations converging to the output*. In this setting, finite computations can be obtained by terminating when a given accuracy criterion is satisfied, and the theory of computable analysis allows to determine whether the solution to a certain problem can be approximated to *any* desired accuracy.

Definition 8 ([28])
Given a function $f : X \mapsto Y$, we say that f is computable if there is an oracle Turing machine that, given any $k \in N$, may ask for arbitrarily good approximations of the input $x \in dom(f)$; i.e., it may ask finitely many questions of the kind "Give me a rational number p with $d(x,p) < 1/2^i$ ", where i may depend on the answers to the previous questions, and after finitely many steps, it writes a rational number q on the output tape with $|f(x) - q| < 2^{-k}$.

More simply, a Turing machine M computes a function f if, on input $x \in dom(f)$, M computes infinitely long and in the long run it writes the infinite sequence denoting the function value $f(x)$ on the one-way output tape; if $x \notin dom(f)$, then M does not produce an infinite output.

Even though the model of computation is based on ordinary Turing Machines, the main purpose of computable analysis is to deal with approximations. The fact that input data are interpreted to be approximate can drastically change the computability properties of problems, as shown by the following simple example.

Example 1
Consider the problem of determining whether the value of a certain polynomial $p(x)$ with rational coefficients is equal to 0 or not. If the input x is taken to be a rational number that is described exactly, the exact value for $p(x)$ can be computed and the problem is easily solvable. However, if the *only* information that we have about x is a sequence of approximations that converges to x, then the problem becomes semi-decidable. From an approximation $\tilde{x}$ of x it is possible to compute an approximation $\tilde{y}$ of $p(x)$ whose accuracy depends on the accuracy of $\tilde{x}$. Hence, when $p(x) \neq 0$ it is possible to find a sufficiently accurate $\tilde{x}$ that allows one to determine that $p(x)$ is different from 0. On the other hand, when $p(x) = 0$, no matter how accurate the current approximation $\tilde{x}$ is, we cannot exclude the possibility that $p(x) \neq 0$, and thus we cannot give a positive answer to the problem.

The representations used in computable analysis to encode inputs and outputs are strictly related to a topology on the set of objects under consideration, thus giving a direct link between approximability, continuity and effective computability. The fundamental theorem is that *only continuous functions can be computable* with respect to a given representation and to the corresponding topology [19]. Hence, if we can prove that a certain function is discontinuous, then it is uncomputable. It is worth emphasizing that a function that is uncomputable with respect to a given representation can be computable with respect to a representation based on a different topology. This corresponds to requiring more information on the inputs, or to requiring less information on the output of the function.

In the case of hybrid automata, discrete transitions can cause discontinuities in both space and time [25], even for simple systems. By the fundamental theorem of computable analysis, this means that the reachable set of hybrid automata is, in general, uncomputable.

Theorem 1 ([25])
For any coherent semantics of evolution[†], the finite-time reachable set of a hybrid automaton is uncomputable.

The above theorem proves that it is impossible to constrain the evolution near the discontinuity points to make the solution computable. However, it does not in itself rule out the possibility of regularizing the evolution in some way so that the evolution becomes at least approximable either *from above* or *from below*.

To formally define this notion of semicomputability we need to introduce some terminology first. In the following, we denote the standard topological closure and interior operators with $cl(\,\cdot\,)$ and $int(\,\cdot\,)$, respectively. Given an $r > 0$, and a point $x \in R^k$, $B(x, r)$ is the r-ball centered in x, i.e.,

[†]The formal definition of coherent semantics is given in [25]. Here it is sufficient to say that this condition eliminates all trivial approximations, such as the one that takes the entire state space.

$\{y \in R^k : |x-y| < r\}$. Given a set $S \subseteq R^k$, we define the r-neighbourhood $r\text{-}nhd(S)$ of S as the set $r\text{-}nhd(S) = \bigcup\{B(x,r) : x \in S\}$. The notions of upper and lower semicomputability are then defined as follows (notice that a subset of real numbers is compact when it is closed and bounded).

Definition 9 ([29])
Given a multivalued function $f : X \mapsto Y$, we say that f is

- *upper-semicomputable* if, given a (representation of a) compact set K, it is possible to compute a sequence of approximations L_n that converges to $f(K)$ *from outside*. To converge from outside then means that the sequence L_n must satisfy $cl(L_m) \subset int(L_n)$ whenever $m \geq n$ and $\bigcap_{n=1}^{\infty} L_n = f(K)$.
- *lower-semicomputable* if, given a (representation of a) closed set C, it is possible to compute a sequence of approximations A_n that converges to $f(C)$ *from below*. To converge from below then means that the sequence A_n must satisfy $A_n \subset \frac{1}{n}\text{-}nhd(A_m)$ whenever $m \geq n$ and $\bigcap_{n=1}^{\infty} \bigcup_{m=n}^{\infty} A_m = f(C)$.

A function $f : X \mapsto Y$ is *computable* if and only if it is both upper and lower semicomputable.

Since the evolution of hybrid automata is not fully computable, the conditions under which it is upper and lower semicomputable are different. A comprehensive discussion on the different semantics and conditions that must be imposed to obtain upper and lower semicomputability, and the complete proofs of the subsequent theorems can be found in [25]. We recall from mathematical analysis that a function f is *upper* (resp., *lower*) *semi-continuous* if, for every point x_0 of the domain, the function values for arguments near x_0 are either close to $f(x_0)$ or less than (resp., greater than) $f(x_0)$. Moreover, we say that f is *closed-valued* if $f(x)$ is a closed set for every point x of the domain, and that f is *compact-valued* when $f(x)$ is a compact set.

Definition 10
A hybrid automaton $\mathcal{H} = \langle \mathrm{Loc}, (U,X,Y), (I,A,O), \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Inv}, \mathrm{Act}, \mathrm{Res}\rangle$ is said to be *upper-semicontinuous* if Inv and Act are *closed*, Dyn and Res are *compact-valued upper-semicontinuous* multivalued functions.

Definition 11
A hybrid automaton $\mathcal{H} = \langle \mathrm{Loc}, (U,X,Y), (I,A,O), \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Inv}, \mathrm{Act}, \mathrm{Res}\rangle$ is said to be *lower-semicontinuous* if Inv and Act are *open*, Dyn and Res are *closed-valued lower-semicontinuous* multivalued functions.

Theorem 2
Let $\mathcal{H}$ be an input-free hybrid automaton.

- If $\mathcal{H}$ is upper-semicontinuous, then the finite-time reachable set starting from a *compact initial set* S_0 is an upper-semicomputable closed set.
- If $\mathcal{H}$ is lower-semicontinuous, then, the finite-time reachable set starting from a *closed initial set* S_0 is a lower-semicomputable closed set.

General hybrid automata of the form given by Definition 1 are not necessarily upper or lower semicontinuous. In order to compute upper or lower approximations to the solution, we need to convert the automaton into either upper or lower semicontinuous form. We can do this by forcing dynamics and reset functions to be continuous and by requiring invariants and guard sets to be open or closed.

Definition 12
Let $\mathcal{H} = \langle \mathrm{Loc}, (U, X, Y), (I, A, O), \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Inv}, \mathrm{Act}, \mathrm{Res}\rangle$ be an input-free hybrid automaton such that Dyn and Res are continuous. Then

- ξ is a trajectory of $\mathcal{H}$ using *upper-semantics* if ξ is a trajectory of the upper-semicontinuous automaton $\overline{\mathcal{H}} = \langle \mathrm{Loc}, (U, X, Y), (I, A, O), \mathrm{Edg}, \mathrm{Dyn}, cl(\mathrm{Inv}), cl(\mathrm{Act}), \mathrm{Res}\rangle$;
- ξ is a trajectory of $\mathcal{H}$ using *lower-semantics* if ξ is a trajectory of the lower-semicontinuous automaton $\underline{\mathcal{H}} = \langle \mathrm{Loc}, (U, X, Y), (I, A, O), \mathrm{Edg}, \mathrm{Dyn}, int(\mathrm{Inv}), int(\mathrm{Act}), \mathrm{Res}\rangle$.

The following result is a direct consequence of Theorem 2.

Corollary 1
Let $\mathcal{H} = \langle \mathrm{Loc}, (U, X, Y), (I, A, O), \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Inv}, \mathrm{Act}, \mathrm{Res}\rangle$ be a hybrid automaton such that Dyn and Res are continuous. Then the finite-time reachable set of $\mathcal{H}$ (starting from a compact initial set X_0) using upper semantics is closed and upper-semicomputable and the reachable set of $\mathcal{H}$ (starting from a closed initial set X_0) using lower semantics is closed and lower-semicomputable.

Moreover, it turns out that $\overline{\mathcal{H}}$ is the "smallest" hybrid automaton for which the evolution is upper-semicomputable and that $\underline{\mathcal{H}}$ is the "greatest" hybrid automaton for which the evolution is lower-semicomputable. This means that, in general, the approximations given by the upper and by the lower semantics are the "best" approximations for computing the evolution of hybrid automata, unless additional information is provided.

Unfortunately, the given definition of lower-semantics is too restrictive when modeling systems with *urgent* transitions. We say that a transition is urgent if it must be activated as soon as the guard becomes true, with no delays. Urgent transitions are usually modeled by pairing a guard $g(x)$ with the negated invariant $\neg g(x)$: in this way the automaton is forced to leave the current location as soon as the transition can be activated.

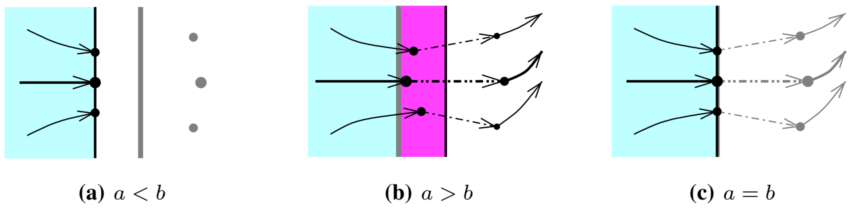


Figure 6. Different evolution scenarios of Example 2. Solid arrows represent continuous evolution, while dashed ones represent discontinuous jumps caused by discrete evolution.

Example 2
Consider a hybrid automaton with a location l_0 with invariant $x \leq a$ and a transition that leaves l_0 when $x \geq b$, with $a, b > 0$. If $a < b$, then the invariant is violated before the transition is active, and no further evolution is possible (Fig. 6(a)). If $a > b$, then the transition is active before the invariant is violated: a transition may occur at any time t such that $a \leq x(t) \leq b$ (Fig. 6(b)). If $a = b$ then a transition must occur exactly when $x(t) = a$ (Fig. 6(c)). However, equality is uncomputable in the theory of Computable Analysis. Hence, for the lower semantics we need to consider the possibility that $a < b$ and the evolution must be blocked, since this is the worst-case scenario. Notice that if we take a transition that does not exist in the exact system, we violate the requirements of lower semantics, whereas with upper semantics we are still over-approximating the reachable set.

The problem with lower semantics described by the above example is caused by the fact that the coupling between the invariants and the activations is not specified into the formalism. Hence, it is not possible to distinguish the case where $a = b$ due to a transition being urgent, from the case where $a = b$ by a coincidence caused by the particular approximation we are using (and under a small change in the parameters or in the approximation, the evolution may be blocked). Only by adding the additional information that the invariant and the activation boundaries lie exactly at the same point we can deduce that the evolution may continue. To this end, we need to extend the definition of hybrid automaton to explicitly list the transitions that are urgent.

Definition 13
A *hybrid automaton with urgent transitions* is a tuple $\mathcal{H} = \langle \mathrm{Loc}, (U, X, Y), (I, A, O), \mathrm{Edg}, \mathrm{Urg}, \mathrm{Dyn}, \mathrm{Inv}, \mathrm{Act}, \mathrm{Res}\rangle$ such that (i) $\langle \mathrm{Loc}, (U, X, Y), (I, A, O), \mathrm{Edg}, \mathrm{Dyn}, \mathrm{Inv}, \mathrm{Act}, \mathrm{Res}\rangle$ is a hybrid automaton, and (ii) $\mathrm{Urg} \subseteq \mathrm{Edg}$ is a set of *urgent transitions*.

The definition of continuous and discrete transition relations must take care of urgent transitions. For each location $l \in \mathrm{Loc}$ we define $\mathrm{UrgAct}(l)$ as the set $\bigcup_{(l,e,l')\in\mathrm{Urg}} \mathrm{Act}(l, e, l')$, i.e., the union of the activation sets of all the urgent transitions exiting from l.

Definition 14
Let H be a hybrid automaton with urgent transitions. 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:

$$(l, \mathbf{v}) \xrightarrow{t}_C (l, \mathbf{v}') \iff \exists \text{ continuous function } f : T \to V \text{ s.t. } f(0) = \mathbf{v},\ f(t) = \mathbf{v}', \text{ and}$$
$$\forall\, t' \in [0, t], f(t') \in \mathrm{Inv}(l) \wedge \frac{df}{dt}(t') \downarrow Z = \mathrm{Dyn}(l)(t'), \text{ and}$$
$$\forall t' \in [0, t), f(t') \notin \mathrm{UrgAct}(l). \quad (7)$$

The definition of the *discrete transition relation* $\xrightarrow{e}_D$ remains unchanged.

We are simply saying that we can stay in location l only if no urgent transition is enabled. This solves the problem discussed in Example 2, because now we do not need to evaluate whether the invariant and the urgent guards are equal, since the invariant is by construction the complement of the urgent guards.

The definitions of trajectory, of reachable set, and the computability results presented above for general hybrid automata easily translate to hybrid automata with urgent transitions.

5. THE ARIADNE SOFTWARE PACKAGE

ARIADNE is a development environment in which to construct space representations and algorithms for reachability analysis of hybrid systems. The original functionality, as described in [30, 31], was based on affine approximations of the continuous dynamics, and was limited to linear systems only. A new computational kernel was subsequently developed, based on rigorous numerical methods for working with real numbers, functions and sets in Euclidean space, and improving the previous one in the following three crucial aspects:

- it supports nonlinear dynamics, guards and reset functions;

- it supports composition of hybrid automata to build complex hybrid systems;
- it extends the approximation capabilities to support both upper and lower semantics for the evolution of hybrid systems (see Definition 12).

The tool is based on the rigorous computable analysis theory presented in Section 4, in order to achieve provable approximation bounds on the computations. In this section we first describe the computational engine of ARIADNE, then we discuss how we discretize the state space and, finally, how we combine continuous evolution and discretization to approximate the reachable set of the system under verification.

5.1. The computational engine

The computational engine of ARIADNE 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 [32]. Every component of a hybrid automaton with urgent transitions $\mathcal{H}$ 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 $\mathrm{Res} : \mathbb{R}^n \mapsto \mathbb{R}^n$.

Functions can be manipulated by using these basic operations:

- Given a function $f : \mathbb{R}^m \mapsto \mathbb{R}^n$ and an argument $x \in \mathbb{R}^m$, the result $f(x)$ can be *evaluated*.
- Two functions $f : \mathbb{R}^m \mapsto \mathbb{R}^n$ and $g : \mathbb{R}^n \mapsto \mathbb{R}^p$ can be *composed* to obtain the function $g \circ f : \mathbb{R}^m \mapsto \mathbb{R}^p$.
- Given $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, and a point $x \in \mathbb{R}^m$, an *approximation* of f over a box with center x can be computed. The result is a *polynomial expansion* of f to a given degree and an *error bound* on the approximation.
- Functions can be *numerically integrated*. This step is crucial to compute the flow tube of the continuous evolution of the hybrid automaton, i.e., a function $\Phi : \mathbb{R}^{n+1} \mapsto \mathbb{R}^n$ such that $\Phi(x_0, t)$ is the set of points reached from x_0 after t time units of continuous evolution.
- The time when the guard of a transition or an invariant is crossed can be established by using the so-called *implicit* function. Given a function $f : \mathbb{R}^{n+1} \mapsto \mathbb{R}^n$ and a function $g : \mathbb{R}^n \mapsto \mathbb{R}$, the *implicit function* of f and g is a function $\Gamma : \mathbb{R}^n \mapsto \mathbb{R}$ such that $g(f(x, \Gamma(x)) = 0$. If f is the flow tube of the continuous evolution in the current location, g is a guard or an invariant, and x_0 is the initial point, then $\Gamma(x_0)$ is the time when the guard or the invariant represented by g is crossed.

Given a hybrid automaton with urgent transitions $\mathcal{H}$, and an initial set of states S_0, the current reachability analysis engine of 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 $\mathit{ReachSet}_{\overline{\mathcal{H}}}(S_0)$ 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 $y \in \mathit{ReachSet}_{\underline{\mathcal{H}}}(S_0)$ such that $|x - y| < \varepsilon$.

Outer approximations cannot provide bounds on the approximation error, but are guaranteed to contain the exact reachable set. For this reason they are used to obtain *positive answers* to verification problems: for instance, if φ is the safety property we want to verify, and the outer approximation O is such that $O \subseteq \mathrm{Sat}(\varphi)$ (where $\mathrm{Sat}(\varphi)$ is the set of states where φ is true), then we can conclude that $\mathit{ReachSet}_{\overline{\mathcal{H}}}(S_0) \subseteq \mathrm{Sat}(\varphi)$ and thus that the system under verification satisfies φ. On the other hand, ε-lower approximations do provide bounds on the error, but are not guaranteed to contain the exact reachable set: the definition cannot exclude the existence of a point $x \in \mathit{ReachSet}_{\underline{\mathcal{H}}}(S_0)$ such that $x \notin L_\varepsilon$. For this reason, ε-lower approximations are used when we want to obtain *negative answers* to verification problems. Suppose that the computed L_ε is such that there exists a point $x \in L_\varepsilon$ such that the ε-ball centered in x is disjoint from $\mathrm{Sat}(\varphi)$: then, we can conclude that there exists a point $y \in \mathit{ReachSet}_{\underline{\mathcal{H}}}(S_0)$ that is not inside $\mathrm{Sat}(\varphi)$, and thus that the system under verification does not respect φ.

The class of hybrid automata managed by ARIADNE conforms to the restrictions of Corollary 1. Hence, both outer approximations and ε-lower approximations of the finite-time reachable set are computable, with the correct semantics. Approximations of the infinite-time reachable set can be obtained by iterating finite-time reachability until a solution is found. However, such an iteration is not guaranteed to terminate for all possible systems. As an example, if the dynamics of the system diverge the iteration goes on forever. To guarantee termination even in this case, a bound on the state space of the system and a limit on the approximation error must be provided by the user.

5.2. *Computing finite-time evolution*

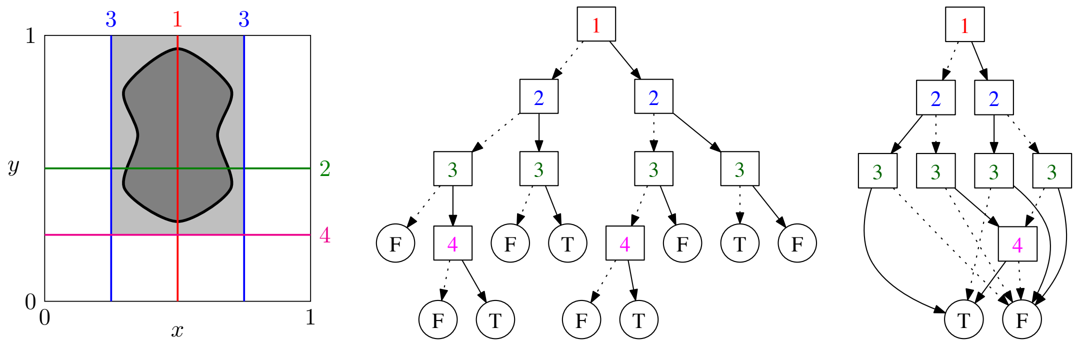
ARIADNE uses polynomial expansions with uniform error bounds as a canonical representation for functions on a bounded domain, similarly to the *Taylor models* of [17]. These *Taylor functions* implement the operations defined in Section 5.1.

Over-approximating *enclosures* to flow tubes are represented using *Taylor image sets*, which are Taylor models mapping a box $[-1, 1]^n$ to a subset of $\mathbb{R}^n$ that covers the desired region of space. Such sets allow for arbitrarily-accurate approximations to the exact flow regions. In particular, the "wrapping effect", which is the catastrophic loss of error which commonly occurs when over-approximating sets with a coordinate-aligned box, is avoided. The flow is computed via its Taylor series expansion, and composition of functions is used to compute reachable sets. Details can be found in [32].

5.3. Discretizing the state space

In order to implement the reachability algorithm in ARIADNE, we must be able to perform operations such as set unions and intersections in a spatially and temporally efficient way. Unfortunately, while the Taylor image set representation has good properties in terms of controlling the accuracy, it performs poorly on most elementary operations. For this reason, when dealing with state sets, Taylor sets must be replaced by a more malleable representation. Consequently, in ARIADNE the state space is partitioned with respect to a grid of chosen granularity: a reachable subset is *discretized*, i.e., overapproximated to the grid, thus losing its Taylor set representation in favor of a *cell* (i.e., a coordinated box) representation. This transformation necessarily introduces an error that combines with the error of the Taylor expansion of the subset. However, it is still possible to control the accuracy by choosing a sufficiently fine granularity for the grid.

ARIADNE uses a variable-step grid structure to discretize sets. This structure is represented by a n-dimensional box containing the set, called *root cell*, which is subdivided by a *binary partition tree*. Every node of the tree is recursively associated to a cell, starting from the root, which is associated to the root cell. Adding two children to a node corresponds to splitting the current cell into two sub-cells along a coordinate that is uniquely determined by the depth (distance from the root) of the node. Every leaf of the tree can be marked as *enabled* or not. The discretized set is then given by the union of the cells corresponding to enabled leaves, while the accuracy is controlled by limiting the maximum depth of the partition tree. To further improve efficiency and memory footprint, partition trees are kept in memory using Reduced Ordered Binary Decision Diagrams (ROBDDs), which turned out to be substantially more compact than the explicit representation of the tree and can be manipulated very efficiently [33, 34].



(a) Discretization **(b)** Binary partition tree **(c)** ROBDD

Figure 7. A discretization example in two-dimensions.

Figure 7 depicts an example of a two-dimensional set discretized using the root cell $[0, 1] \times [0, 1]$ and maximum depth 4 (corresponding to a smallest cell size of 0.25×0.25) together with the corresponding partition tree and ROBDD representations. Dotted arrows in the pictures correspond to left children, while solid arrows to right ones. Enabled leaves are labeled with "T", disabled ones with "F". Internal nodes are labelled with the depth. In Figure 7(a) the lines splitting the root cell are numbered with the depth of the corresponding nodes in the tree.

5.4. *The reachability algorithm*

The reachability algorithm implemented in ARIADNE takes as inputs the hybrid automaton $\mathcal{H}$, the initial set S_0 and, to control the accuracy of the approximations, the following parameters:

- a grid size that sets the desired precision of the discretizations;
- a time step t between two discretization events;
- an integration step h to control the accuracy of the continuous evolution;
- the maximum diameter D of the flow tube. This parameter controls when a set must be split (to decrease the error) when computing an outer-approximation, and fixes the value of $\varepsilon = D$ for lower approximations.

Then it proceeds as follows:

1. If the initial set S_0 is larger than a grid cell, outer-approximate it on the grid, and let $\mathcal{W}$ be the obtained set of cells. Otherwise, let $\mathcal{W} = \{S_0\}$.
2. Extract a new *working cell* W from $\mathcal{W}$:
   - Compute an approximation of the flow tube $R(W, t)$ and final set $F(W, t)$ for a time step t.
   - Map $R(W, t)$ and $F(W, t)$ on the grid, using the desired approximation type.
     - If computing an outer approximation, put the newly reached cells of $F(W, t)$ in $\mathcal{W}$.
     - If computing an ε-lower approximation and the diameter d of $F(W, t)$ is smaller than $\varepsilon = D$, put $F(W, t)$ into $\mathcal{W}$. Otherwise, do nothing.
3. Repeat recursively until $\mathcal{W}$ is empty or no new cells can be found.

Finally, the resulting reachability set is simply given by the union of the $R(W, t)$ sets obtained in the procedure above. Figure 8 depicts one step of the procedure.

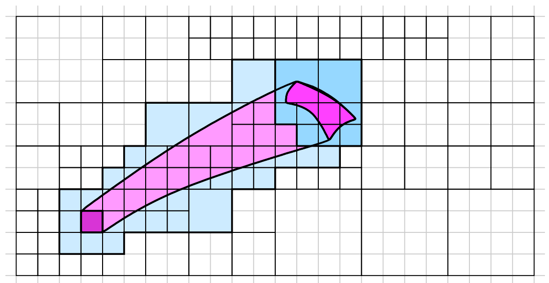


Figure 8. Step 2. of the reachability algorithm.

Note that the motivation for discarding the sets $F(W, t)$ having $d \geq D$ is that we would not be able to guarantee $|x - y| < \varepsilon$ anymore, for any $x, y \in F(W, t)$. Since by construction there exists at least one point $y \in F(W, t)$ that belongs to the true reachable set and y is not known, the condition $d < D = \varepsilon$ represents a sufficient condition for an L_ε approximation.

The output of the reachability routines converges to the "best possible" approximations given by the upper and lower semantics (Definition 12) when the size of the grid, the integration step h and the value of ε converge to zero. Termination of the algorithm is guaranteed even for systems that diverge by providing a bound on the evolution time (for finite-time reachability) or on the state space of the system (for infinite-time reachability). In the latter case, since the bounding set is divided into a finite number of cells by the underlying grid, after a finite number of steps either no more cells are marked or the whole search space is marked.

The current stable version of ARIADNE is freely available from [20] under the `crete-bdd` branch of the code repository. The tool has already been used to verify challenging industrial benchmark problems within the European Union projects COCONUT, CON4COORD and SPEEDS. The interested reader is referred to [20, 30, 31] for more information.

6. ASSUME-GUARANTEE REASONING

Assume-guarantee reasoning is a verification approach where the design process is seen as an assembly of components. A component is represented in terms of assumptions about its environment and guarantees about its behavior. In this way, the requirements of the whole system can be decomposed into a set of simpler requirements that, if satisfied, guarantee that the overall requirement is satisfied.

In this setting, each component of the system is equipped with a set of events and variables and its expected behaviour is described by some *assumptions* A and *guarantees* (or *promises*) G over these events and variables. The pair (A, G) can be viewed as a high-level specification of the component's functionalities and is usually referred to as the *contract* of the component. Contract composition formalizes how contracts related to different components have to be combined to represent the whole system. Every component of the system can then be verified separately and, if all components satisfy their "local" requirements, then the composition is well formed and the whole system is correct [35]. Contract-based modeling and assume-guarantee reasoning are key features of the *Heterogeneous Rich Components* meta-model developed inside the SPEEDS European project as a formalism to model and verify complex systems using a component-based approach, where ARIADNE is used as the formal verification tool when hybrid components are involved [36, 24].

Hybrid automata with inputs and outputs are a faithful formalism to model heterogeneous components mixing discrete and continuous behaviours in an assume-guarantee setting. Indeed, a component C can be represented by a hybrid automaton $\mathcal{H}_C$ with inputs U and outputs Y. Assumptions A of the contract specify the set of allowed input trajectories for $\mathcal{H}_C$, while requirements G describe the admissible output trajectories. In this setting, a component $\mathcal{H}_C$ respects the contract (A, G) if and only if for any trajectory ξ of $\mathcal{H}_C$, if the input trajectory $\xi \downarrow U, I \in A$ then the output trajectory $\xi \downarrow Y, O \in G$.

When composing two components C_1 and C_2 together, we have that some of the output variables and events of C_1 can be input variables and events for C_2, and vice-versa. In the assume-guarantee setting, assumptions and guarantees of the two components must be checked for compatibility before allowing the composition. Formally, we have that the composition $C_1 \| C_2$ is *admissible* if and only if the guarantees on the output variables and events of C_1 that are input variables or events for C_2 respect the assumptions for C_2, and vice-versa. For instance, in a serial composition where all the outputs of C_1 are inputs for C_2, $C_1 \| C_2$ is allowed only if $G_1 \subseteq A_2$. Under this definition, the replacement of the component C_1 with any other component C_3 that respects a stronger contract (A_3, G_3), that is, for which $A_3 \supseteq A_1$ and $G_3 \subseteq G_1$, is always safe: the set of behaviours of $C_3 \| C_2$ is a subset of the set of possible behaviours of $C_1 \| C_2$ determined by the contracts (A_1, G_1) and (A_2, G_2).

Usually assumptions and guarantees are described using some logical formalism, and can be arbitrarily complex. To simplify their algorithmic treatment, in this paper we restrict our attention to *uniform assumptions* modeled by another hybrid automaton $\mathcal{H}_A$, whose output variables correspond to the input variables of $\mathcal{H}_C$, and to *safety guarantees* specified as a subset of $\mathrm{Val}(Y)$. Then, we concentrate on the following verification problems, and we will show how the reachability analysis and approximation techniques of ARIADNE can be used to solve them.

- *Verification of Contract Satisfaction*. Given a component C and a contract (A, G), check whether the behaviour of C is contained in G under assumption A (i.e., it respects the contract). More formally, verify that for all possible input trajectories for $\mathcal{H}_C$ generated by $\mathcal{H}_A$, the values of the output variables are contained in G.
- *Dominance checking*. Given two components C_1 and C_2 defined over the same set of events and variables, and two assumptions A_1 and A_2, we define the corresponding *induced guarantees* $G(A_1, C_1)$ and $G(A_2, C_2)$ as the sets of outputs generated by C_1 and C_2 under the assumption A_1 and A_2, respectively. We say that C_1 *dominates* C_2 if and only if under weaker assumptions on C_1 ($A_2 \subseteq A_1$), stronger promises are guaranteed ($G(A_1, C_1) \subseteq G(A_2, C_2)$). If this is the case, the component C_2 can be replaced by C_1 in the system without affecting the whole system behaviour.

In our setting, we have that C respects the contract (A, G) if the outputs Y of the composition of $\mathcal{H}_C$ and $\mathcal{H}_A$ are contained in G. The contract satisfaction problem can be solved by checking if $ReachSet_{\mathcal{H}_A \| \mathcal{H}_C} \downarrow Y \subseteq G$. The approximation techniques of ARIADNE can be used to obtain both positive and negative answers for this problem, as stated by the following proposition.

Proposition 1
1. Let C be a component, (A, G) a contract for it, and O an outer-approximation of $ReachSet_{\mathcal{H}_A \| \mathcal{H}_C} \downarrow Y$. If $O \subseteq G$, then we can conclude that C respects the contract.
2. Conversely, let L_ε be a ε-lower approximation of $ReachSet_{\mathcal{H}_A \| \mathcal{H}_C} \downarrow Y$ for some $\varepsilon > 0$. If there exists a point $x \in L_\varepsilon$ such that the ε-ball centered in x, $B(x, \varepsilon)$, is disjoint from G, then we can conclude that C does not respect the contract.

Proof
By the definition of outer-approximation, we have that $ReachSet_{\mathcal{H}_A \| \mathcal{H}_C} \downarrow Y \subset O$ and thus property *1.* follows trivially.

To argue property *2.*, by the definition of ε-lower approximation, we have that $x \in L_\varepsilon$ implies that there exists $z \in B(x, \varepsilon)$ such that $z \in ReachSet_{\mathcal{H}_A \| \mathcal{H}_C} \downarrow Y$. Since $B(x, \varepsilon)$ is disjoint from G, we can conclude that $z \notin G$ and thus that C does not respect the contract. $\square$

As an immediate consequence of Theorem 3.20 in [23], it holds that the inclusion $O \subseteq G$ can be verified when O is a compact set and G is an open set. The first condition is guaranteed by the definition of outer approximation, the second one must be enforced by allowing only open guarantees for the contract (A, G).

The dominance checking problem can be solved by a similar approach: if the two components are represented by the hybrid automata $\mathcal{H}_{C_1}$ and $\mathcal{H}_{C_2}$, and the assumptions A_1 and A_2, with $A_2 \subseteq A_1$, are represented by the hybrid automata $\mathcal{H}_{A_1}$ and $\mathcal{H}_{A_2}$, respectively, we have that $\mathcal{H}_{C_1}$ dominates $\mathcal{H}_{C_2}$ if and only if $\mathit{ReachSet}_{\mathcal{H}_{A_1}\|\mathcal{H}_{C_1}} \downarrow Y \subseteq \mathit{ReachSet}_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$, where Y is the set of (common) output variables of $\mathcal{H}_{C_1}$ and $\mathcal{H}_{C_2}$. In contrast with the case of the contract satisfaction problem, now we are required to test the inclusion of two closed sets, which is an uncomputable problem [19]. However, it is possible to test the inclusion of a compact set in an open set, so it suffices to require that $\mathit{ReachSet}_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$ is sufficiently regular that testing inclusion of the closed ε-neighbourhood of $\mathit{ReachSet}_{\mathcal{H}_{A_1}\|\mathcal{H}_{C_1}} \downarrow Y$ in a ε-lower approximation of $\mathit{ReachSet}_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$ allows us to deduce inclusion of the original sets.

Now, given a set $S \subseteq \mathbb{R}^k$ and a $\gamma > 0$, we recall from Section 4 that the γ-neighbourhood $\gamma\text{-}\mathit{nhd}(S)$ of S it the set $\gamma\text{-}\mathit{nhd}(S) = \bigcup\{B(x,\gamma) : x \in S\}$. Symmetrically, we define the γ-interior $\gamma\text{-}\mathit{int}(S)$ as the set $\gamma\text{-}\mathit{int}(S) = \{x \in S : B(x,\gamma) \subseteq S\}$, i.e., the set of all points in S that are at a distance greater or equal than γ from the border. It is immediate that $S \subseteq \gamma\text{-}\mathit{int}(\gamma\text{-}\mathit{nhd}(S))$ for any set S. We say that S is *γ-regular* if $\gamma\text{-}\mathit{int}(\gamma\text{-}\mathit{nhd}(S)) = S$. The following lemma shows that if L_ε is a ε-lower approximation of a γ-regular set S then $\varepsilon\text{-}\mathit{int}(L_\varepsilon)$ is an under-approximation of S for every $\varepsilon \leq \gamma$.

Lemma 1
Let $R \subseteq \mathbb{R}^k$ be a γ-regular set (for some $\gamma > 0$), and let L_ε be an ε-lower approximation of R such that $\varepsilon \leq \gamma$. Then $\varepsilon\text{-}\mathit{int}(L_\varepsilon) \subseteq R$.

Proof
Let $x \in \varepsilon\text{-}\mathit{int}(L_\varepsilon)$. By the definition of ε-interior, we have that $B(x,\varepsilon) \subseteq L_\varepsilon$. From the definition of ε-lower approximation, it follows that for every point $x' \in B(x,\varepsilon)$ there exists a point $y' \in R$ such that $|x' - y'| < \varepsilon$. Consider now the γ-ball centered in x. Since $\varepsilon \leq \gamma$, we have that for every point $x' \in B(x,\gamma)$ either $x' \in B(x,\varepsilon)$, or there exists $x'' \in B(x,\varepsilon)$ such that $|x' - x''| \leq \gamma - \varepsilon$. In the latter case there exists $y' \in R$ such that $|x' - y'| = |x' - x'' + x'' - y'| \leq |x' - x''| + |x'' - y'| < \gamma - \varepsilon + \varepsilon = \gamma$. Consequently, in the general case we can assume $|x' - y'| < \gamma$. Since for every point $x' \in B(x,\gamma)$ we have proved that $x' \in \gamma\text{-}\mathit{nhd}(R)$, we can conclude that $B(x,\gamma) \subseteq \gamma\text{-}\mathit{nhd}(R)$. By the definition of γ-interior, $x \in \gamma\text{-}\mathit{int}(\gamma\text{-}\mathit{nhd}(R))$ holds. Under the assumption that R is γ-regular, we have that $x \in R$, and we have proved that $\varepsilon\text{-}\mathit{int}(L_\varepsilon) \subseteq R$. $\square$

The above Lemma enables us to give positive and negative answers to the dominance checking problem under the restriction that $\mathit{ReachSet}_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$ is γ-regular.

Proposition 2
Let C_1 and C_2 be two components, and let $A_2 \subseteq A_1$ be two assumptions. If $\mathit{ReachSet}_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$ is γ-regular for some $\gamma > 0$, the following properties hold.

1. Let O^1 be an outer approximation of $\mathit{ReachSet}_{\mathcal{H}_{A_1}\|\mathcal{H}_{C_1}} \downarrow Y$ and L^2_ε be an ε-lower approximation of $\mathit{ReachSet}_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$ with $\varepsilon \leq \gamma$. If $O^1 \subseteq \varepsilon\text{-}\mathit{int}(L^2_\varepsilon)$, then C_1 dominates C_2.

2. Conversely, let L^1_ε be an ε-lower approximation of $ReachSet_{\mathcal{H}_{A_1}\|\mathcal{H}_{C_1}} \downarrow Y$ and O^2 be an outer approximation of $ReachSet_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$. If there exists a point $x \in L^1_\varepsilon$ such that the ε-ball centered in x, $B(x, \varepsilon)$, is disjoint from O^2, then we can conclude that C_1 does not dominate C_2.

Proof
By Lemma 1, we have that if $ReachSet_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$ is γ-regular and $\varepsilon \leq \gamma$, then $\varepsilon\text{-}int(L^2_\varepsilon) \subseteq ReachSet_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$ and thus property *1.* follows trivially.

As for property 2., by the definition of ε-lower approximation we have that $x \in L^1_\varepsilon$ implies that there exists $z \in B(x, \varepsilon)$ such that $z \in ReachSet_{\mathcal{H}_{A_1}\|\mathcal{H}_{C_1}} \downarrow Y$. Since $B(x, \varepsilon)$ is disjoint from O^2, we can conclude that $z \notin ReachSet_{\mathcal{H}_{A_2}\|\mathcal{H}_{C_2}} \downarrow Y$ and thus that C_1 does not dominate C_2. □

In order to practically implement dominance checking based on Propositions 1 and 2, it is convenient to find sufficient conditions for γ-regularity that are easier to check.

One example is *convexity*. We recall from topology that a set $S \subseteq \mathbb{R}^k$ is *convex* if the line segment joining any pair of points of S lies entirely in S. Given $\alpha \in \mathbb{R}^k$ and $\beta \in \mathbb{R}$ we can define the corresponding *hyperplane* as the set $H = \{x \in \mathbb{R}^k : \alpha^T z = \beta\}$. A hyperplane divides $\mathbb{R}^k$ into the two half spaces $H_\leq = \{x \in \mathbb{R}^k : \alpha^T z \leq \beta\}$ and $H_\geq = \{x \in \mathbb{R}^k : \alpha^T z \geq \beta\}$. We say that two sets S_1 and S_2 are *separated* by H if and only if $S_1 \subseteq H_\leq$ and $S_2 \subseteq H_\geq$, or vice versa. A separation is *strict* if S_1 and S_2 are disjoint from H. Closed convex sets respect the following strict separation theorem.

Theorem 3 ([37, Theorem 11.4, p. 99])
Let $S \subseteq \mathbb{R}^k$ be a closed convex set, and let $v \notin S$. Then

1. there exists a unique point $x_0 \in S$ closest to v;
2. the hyperplane $H = \{z \in \mathbb{R}^k : \alpha^T z = \beta\}$, with $\alpha = x_0 - v$ and $\beta = (x_0 - v)^T v + |x_0 - v|^2/2$ strictly separates v from S.

The following lemma shows that every closed and convex set is γ-regular for all $\gamma > 0$.

Lemma 2
Let $R \subseteq \mathbb{R}^k$ be a closed convex set. Then R is γ-regular for all $\gamma > 0$.

Proof
Suppose by contradiction that there exists $\gamma > 0$ and $v \in \gamma\text{-}int(\gamma\text{-}nhd(R))$ such that $v \notin R$. Consider the unique point $x_0 \in R$ closest to v and the hyperplane H separating v from R given by Theorem 3. Let z be the intersection between the γ-sphere centered in v, $S(v, \gamma)$, and the infinite line connecting x_0 and v that lies on the same half space of v (see Figure 9). By the definition of γ-interior and $v \in \gamma\text{-}int(\gamma\text{-}nhd(R))$, we have that $z \in \gamma\text{-}nhd(R)$. This implies that there exists $z' \in R$ such that $z \in B(z', \gamma)$. Moreover, thanks again to the strict separation construction by Theorem 3 shown in Figure 9, we have that $z \notin R$ and that the unique point in R closest to z is exactly x_0. Since the distance between z and x_0 is greater than γ, we found a contradiction with the hypothesis that $z \in \gamma\text{-}nhd(R)$. Hence, v must belong to R and we have proved that R is γ-regular for all $\gamma > 0$. □

Propositions 1 and 2 have been exploited to implement a semi-automatic verification flow based on this assume-guarantee paradigm in ARIADNE. The user provides the automaton describing the component, the automaton describing the assumptions, and the region of space that corresponds to the requirements G, for the contract satisfaction problem, or the automata describing the components

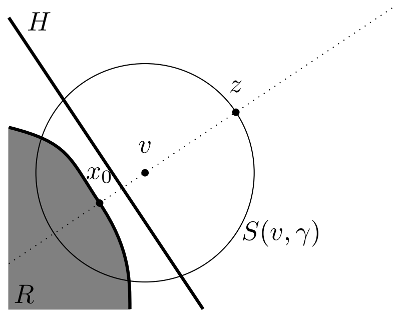


Figure 9. A graphical explanation of the proof of Lemma 2.

and the assumptions A_1 and A_2, for the dominance checking problem. The initial values of the parameters determining the accuracy of the computation (grid size, integration step, value of ε for the lower-approximation) can be either fine-tuned by the user, or automatically determined by the algorithm. Then, the verification loop proceeds by increasing the accuracy of the computation through a series of refinement steps until either a positive or negative answer is obtained, or the maximum specified accuracy is reached.

In the next Section we will apply our flow to the watertank systems described in Section 3. In particular, in the dominance case, we will show that the projections of the approximated sets on the output variables are convex, thus guaranteeing the regularity property regardless of the accuracy used.

7. VERIFICATION OF THE WATERTANK EXAMPLE IN ARIADNE

In this section we will show how the reachability analysis capabilities of ARIADNE can be exploited to solve some verification problems for the watertank example presented in Section 3, under the assume-guarantee setting defined in Section 6.

The actual behavior of the watertank can depend on environmental parameters that are outside the control of the designer, and for which there may be imperfect knowledge, like the pressure of the inlet flow or the sensor error, and on design parameters that can be fixed by the designer, but whose admissible values are not necessarily known a priori, like the thresholds for the hysteretic controller. Hence, the verification problems we have to tackle are two-fold: determine the value for the design parameters for which the component (in this case the controller) respects the guarantees, and do this for all possible values of the environmental parameters.

Verification of contract satisfaction. First of all, we want to be sure that both variants of the watertank system are safe, namely, that they keep the water level within a given interval.

Problem 1 (Verification of contract satisfaction for the hysteretic controller)
Suppose that the values of the constants for the water tank system with a hysteretic controller are the following: $\lambda = 0.065$, $T = 4.0$, $H = 10.0$. The constants l and h instead represent design parameters, with $l \in [5.25, 6.25]$ and $h \in [7.25, 8.25]$. Under the assumption that $f(p(t))$ has a constant unknown value in the $[0.3, 0.33]$ interval, that the sensor error $\delta(t)$ has a constant unknown

value in $[-0.1, 0.1]$, the initial water level $x_0 = x(0)$ can take any value in the $[6.5,\ 7.0]$ interval, and the initial locations are $\{l_0, l_1, l_2, l_3\}$, verify whether or not the controller guarantees a water level x between 5.25 and 8.25.

The extension of the values of certain variables/constants in Problem 1 to intervals require some clarification. The x_0 variable value, the $f(p(t))$ constant and the $\delta(t)$ constant are defined over intervals so to model imperfect knowledge of their values: as a consequence, the model for the controller under verification is more robust. On the other hand, the guarantee on x must necessarily hold over an interval, since the range of the water level can not be of zero measure otherwise the system should start and forever remain at the same value of x (hardly an interesting behavior). Finally, the intervals for l and h are used to perform verification inside a design space, i.e., to determine the values for the design parameters that satisfy the contract.

Problem 2 (Verification of contract satisfaction for the proportional controller)
Suppose that the values of the constants for the water tank system with a proportional controller are the following: $\lambda = 0.065$, $T = 4.0$, $H = 10.0$ and $\tau = 1.25$. The constants $K_P \in [0.2,\ 0.8]$ and $R \in [5.25,\ 8.25]$ instead represent design parameters. Under the assumption that $f(p(t))$ has a constant unknown value in the $[0.3,\ 0.33]$ interval, the sensor error $\delta(t)$ has a constant unknown value in $[-0.1, 0.1]$, the initial water level x_0 can take any value in the $[6.5,\ 7.0]$ interval, and the initial locations are $\{c_0, c_1, c_2\}$, verify whether or not the controller guarantees a water level between 5.25 and 8.25.

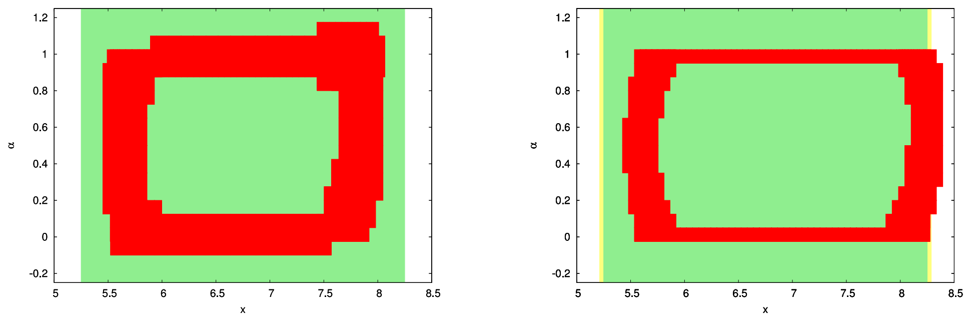
The same discussion as in Problem 1 regarding the use of intervals holds. Please note that the assumptions for the two contract satisfaction problems are the same: this choice was made to conform to the definition of the dominance checking problem.

As discussed in Section 6, the contract satisfaction problem can be reduced to the reachability problem, and the approximation capabilities of ARIADNE can be used to obtain both positive and negative answers to it. In the particular case of Problem 1, the watertank system $\mathcal{H}_1$ respects the contract if and only if $\mathit{ReachSet}_{\mathcal{H}_1} \!\downarrow\! x \subseteq \{x : 5.25 < x < 8.25\}$. The ARIADNE verification loop starts from some initial values of the parameters determining the accuracy of the computation and from a bound on the state space of the system and proceeds as follows.

1. Compute an outer-approximation O of $\mathit{ReachSet}_{\mathcal{H}_1}$ with the current accuracy parameters;
2. If $O \!\downarrow\! x \subseteq \{x : 5.25 < x < 8.25\}$, the system satisfies the contract. Exit with a *true* outcome;
3. Otherwise, compute an ε-lower approximation L_ε of $\mathit{ReachSet}_{\mathcal{H}_1}$;
4. If there exists at least a point in $L_\varepsilon \!\downarrow\! x$ that is outside $\{x : 5.25 < x < 8.25\}$ by more than ε, the system does not satisfy the contract. Exit with a *false* outcome;
5. Otherwise, halve the accuracy parameters (grid size, ε, $\ldots$) and restart from point 1.

For termination reasons, a maximum number of iterations N (which implies a maximum achievable accuracy) must be enforced beforehand. If no definite result is obtained after the given number of iterations, the verification outcome is *indeterminate* and no conclusion on the contract satisfaction is drawn. Specifically, we use $N = 7$, meaning that each accuracy parameter is reduced down to $1/128$ of its initial value. This number was chosen after a manual pre-evaluation of the performance of the verification loop on both systems, under the constraint of a worst-case verification time of

10 minutes. The results of this Section have been obtained by running ARIADNE on an Intel Xeon W3520 processor under a Linux 2.6.32 x86-64 platform.



(a) $l = [5.75, 5.781]$, $h = [7.75, 7.781]$ **(b)** $l = [5.75, 5.781]$, $h = [8.125, 8.156]$

Figure 10. ARIADNE results for two choices of the l and h thresholds, with a hysteretic controller.

The verification in Problem 1 is performed for different values of the h and l thresholds. Figure 10 shows the computed approximations of the reachable set at the end of the verification loop for two different choices of the thresholds. The water level is on the x-axis of the graphs, while the valve aperture is on the y-axis. The computed approximation is pictured in red, while the safe region is in green. For $l = [5.75, 5.78125]$ and $h = [7.75, 7.78125]$ (Figure 10(a)) the computed outer-approximation is inside the safe region, and thus the system is proved to be safe. Conversely, for $l = [5.75, 5.78125]$ and $h = [8.125, 8.15625]$ (Figure 10(b)) an ε-lower approximation is computed with $\varepsilon = 0.0375$. In this case the result lies outside the safe region by more than ε, and thus the system is proved to be unsafe.

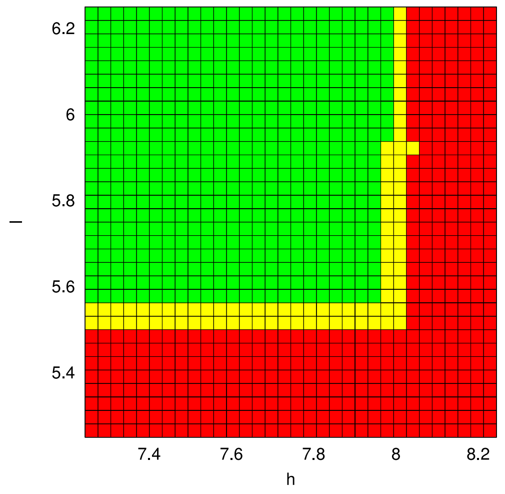


Figure 11. ARIADNE results for the contract satisfaction problem, with a hysteretic controller.

Figure 11 shows the complete verification results, where the interval for each threshold is split into 32 parts, thus yielding 1024 different interval couples (i.e. bidimensional boxes). For each of these boxes, the corresponding system is verified: a green color implies a *true* outcome, a red color is used for a *false* outcome, and a yellow color is chosen if the outcome is *indeterminate*. As it can

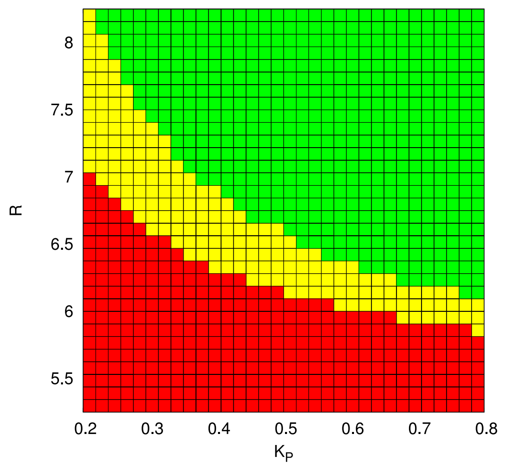


Figure 12. ARIADNE results for the contract satisfaction problem, with a proportional controller.

be seen, two definite regions are identified where a positive and negative outcome is obtained. Quite intuitively, the system turns out to satisfy the contract for low values of the upper threshold and high values of the lower threshold. Between the two regions, the outcome is indeterminate for the given maximum number of iterations. This is due to the fact that the nearer we are to the border between positive and negative results, the higher is the accuracy required to obtain a definite answer, possibly higher than the maximum allowed. More in general, an indeterminate outcome is possible even for boxes whose surrounding boxes yield a definite outcome. This situation, suggested in Fig. 11 by the "partially isolated" indeterminate box near $h = 8$ and $l = 6$, can be interpreted with the system being particularly sensitive to the two parameters in the given subregion. Since the expected outcome for the box is negative, the situation is the following: the over-approximation error when performing evolution under lower semantics is larger; this in turn causes the computation of L_ε to terminate earlier, yielding a smaller $\varepsilon\text{-}int(L_\varepsilon)$ which does still satisfy the contract. If it were the case of a positive expected outcome, then the larger over-approximation error when performing evolution under upper semantics would have produced a strictly larger O; such outer approximation would then have happened not to satisfy the contract.

As far as Problem 2 is concerned, the same considerations as in the previous Problem hold. In this case the chosen parameters are the proportional gain K_P and the reference water level R. The results of Figure 12 again show a clear separation between the positive outcome and negative outcome regions. In this case, however, the two parameters have a hyperbolic relation, implying that they depend on one another. On the contrary, the results of Figure 11 suggest that h and l affect quite independently the verification result.

As an additional comment, in the proportional case we can better observe how the sensitivity of the verification outcome depends on the actual values of the parameters. For low values of K_P (or equivalently, for high values of R), the indeterminate region is wider, hence the outcome is less definite and requires comparatively higher accuracy in order to be obtained.

Dominance checking. Once we have determined that both variants of the system are safe, we would like to compare them, and to know under which values of the design parameters the hysteretic controller can be replaced with the proportional one without affecting safety of the whole system.

Problem 3 (Dominance checking)
Suppose that the values of the constants for the two water tank systems are the following: $\lambda = 0.065$, $T = 4.0$, $l = 5.75$, $h = 7.75$, $H = 10.0$ and $\tau = 1.25$. The constants $K_P \in [0.2,\ 0.8]$ and $R \in [5.25,\ 8.25]$ instead represent design parameters for the system with a proportional controller. Under the assumption that $f(p(t))$ has a constant unknown value in the $[0.3,\ 0.33]$ interval, that the sensor error $\delta(t)$ has a constant unknown value in $[-0.1, 0.1]$, the initial water level x_0 is in the $[6.5,\ 7.0]$ interval, and the initial locations are $\{l_0, l_1, l_2, l_3\}$ and $\{c_0, c_1, c_2\}$, verify whether or not the proportional controller guarantees the dominance over the hysteretic controller.

Since no significant design parameters are shared between the two systems, in Problem 3 we choose to explore the values of constants for the supposedly *dominating* system (i.e. the water tank with a proportional controller) only, while the values of constants for the supposedly *dominated* system (i.e. the water tank with a hysteretic controller) are chosen as the midpoints of the respective intervals used in Problem 1. As an example, this can model the situation in which the designer wants to replace a legacy controller (for which no choice on the design parameters is possible) with a newer one (where design parameters can be tuned by the designer). However, this is not the only possible situation: another possible choice is to explore the values of the supposedly dominated controller while keeping the supposedly dominating one fixed, or explore the values of both controllers.

The dominance checking problem can be solved in ARIADNE with a strategy that is similar to the one described for Problems 1-2. The proportional watertank controller (in $\mathcal{H}_2$) dominates the hysteretic controller (in $\mathcal{H}_1$) if and only if $\mathit{ReachSet}_{\mathcal{H}_2}\downarrow x \subseteq \mathit{ReachSet}_{\mathcal{H}_1}\downarrow x$, i.e., with the same assumptions it guarantees stronger promises. As in the previous case, the ARIADNE verification loop starts from some initial values of the accuracy parameters and from a bound on the state space. Then, it proceeds as follows:

1. Compute an outer-approximation O_2 of $\mathit{ReachSet}_{\mathcal{H}_2}$ with the current accuracy parameters;
2. Compute an ε-lower approximation $L_{\varepsilon,1}$ of $\mathit{ReachSet}_{\mathcal{H}_1}$;
3. If $O_2\downarrow x \subset \varepsilon\text{-}\mathit{int}(L_{\varepsilon,1}\downarrow x)$, then $\mathcal{H}_2$ dominates $\mathcal{H}_1$. Exit with a *true* outcome;
4. Compute an outer-approximation O_1 of $\mathit{ReachSet}_{\mathcal{H}_1}$;
5. Compute an ε-lower approximation $L_{\varepsilon,2}$ of $\mathit{ReachSet}_{\mathcal{H}_2}$;
6. If $\varepsilon\text{-}\mathit{int}(L_{\varepsilon,2}\downarrow x) \nsubseteq O_1\downarrow x$, then $\mathcal{H}_2$ does not dominate $\mathcal{H}_1$. Exit with a *false* outcome;
7. Otherwise, halve the accuracy parameters (grid size, integration step, ε) and restart from 1.

Again, an *indeterminate* outcome is provided if no definite result can be obtained at the maximum accuracy. In this case, a maximum number of iterations $N = 6$ is chosen. Notice that this algorithm is correct under the assumptions required by Proposition 2, that is, that $\mathit{ReachSet}_{\mathcal{H}_1}\downarrow Y$ is γ-regular for some γ greater or equal to the value of ε at the first iteration. Since in the case of the watertank example the only output variable of the system is the water level x, and since the reset functions of $\mathcal{H}_1$ do not change the value of variables, we have that $\mathit{ReachSet}_{\mathcal{H}_1}\downarrow Y$ is convex and, by Lemma 2, γ-regular for every $\gamma > 0$.

It must be noted that the convexity of $\mathit{ReachSet}_{\mathcal{H}_1}\downarrow Y$ is useful for successfully checking $O_2\downarrow x \subset \varepsilon\text{-}\mathit{int}(L_{\varepsilon,1}\downarrow x)$. For example, any "holes" in $\mathit{ReachSet}_{\mathcal{H}_1}$ may likely prevent a positive

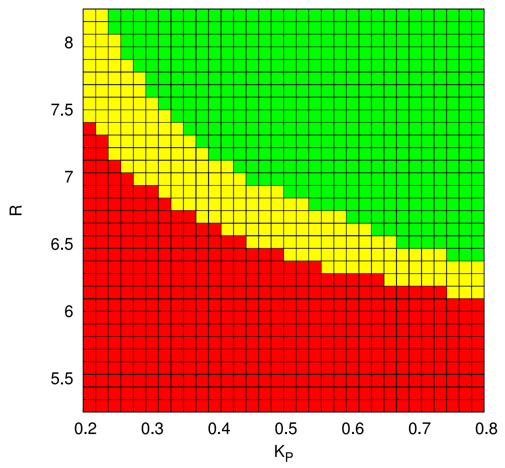


Figure 13. ARIADNE results for the dominance checking problem.

answer to the dominance problem. However, the specific requirement of convexity, while feasible in some cases, is rather stringent. Additionally, there may be situations where we do not actually care for the presence of small "holes" or even for some small error on the boundary of the reachable set. ARIADNE currently does not implement a solution to this issue. However, a possible approach would involve the introduction of a *tolerance* parameter μ for the comparison of $L_{\varepsilon,1}$ with O_2. In practice, we add an error to cancel irregularities in $ReachSet_{\mathcal{H}_1}$. The value of μ may be a function of the contract to be satisfied by both systems, for example a fraction of the radius of a convex unsafe set. We will then check $O_2 \downarrow x \subset (\mu - \varepsilon)\text{-}nhd(L_{\varepsilon,1} \downarrow x)$, where $(\mu - \varepsilon)\text{-}nhd(L_{\varepsilon,1} \downarrow x)$ is the lower approximation enlarged by a $\mu - \varepsilon$ quantity. The actual tolerance will be a quantity in the $[\mu - \varepsilon,\ \mu]$ interval, reaching the μ value for infinite accuracy. Therefore, if we want to guarantee that "holes" of a given radius r are ignored for any $\varepsilon \leq \varepsilon_0$, we must have an initial accuracy $\varepsilon_0 \leq \mu - r$.

In order to identify when $\mathcal{H}_2$ dominates $\mathcal{H}_1$, we keep $\mathcal{H}_1$ fixed and vary $\mathcal{H}_2$ in respect to the K_P and R parameters. Please note that in $\mathcal{H}_1$ we choose $h = 7.75$ and $l = 5.75$ in order for the controller to satisfy the contract of Problem 1 (see Figure 11). In fact, the practical objective of such a Problem is to identify a controller with better guarantees than a given one that already satisfies a contract.

We observe from Figure 13 that the outcomes bear a strong similarity to those of Figure 12, with a larger *false* region. Intuitively, this result means that for $\mathcal{H}_2$ to dominate $\mathcal{H}_1$, the system must necessarily satisfy the same contract of $\mathcal{H}_1$ *and* yield a narrower range for the x variable.

It is worth emphasizing that, to the best of our knowledge, ARIADNE is the only currently available tool for reachability analysis of hybrid systems that can verify both the contract satisfaction and the dominance checking problems. Other tools, like PhaVer [10] or HSolver [13] can compute over approximations of the reachable set of hybrid automata, and thus can be used to obtain positive answers to the contract satisfaction problem. However, they cannot compute lower or under approximations of the reachable set and thus cannot be used to obtain negative answers to the contract satisfaction problem or to solve the dominance checking problem.

8. CONCLUSIONS AND FUTURE WORK

In this paper we described the verification strategy implemented in ARIADNE to solve the contract satisfaction problem and the dominance checking problem for hybrid systems using assume-guarantee reasoning techniques.

Currently ARIADNE is one of the few tools available in the scientific community that can verify nonlinear hybrid systems. Compared to other approaches, it has two main advantages: it can manage more complex dynamics (most tools are limited to linear systems only) and it can compute both outer and lower approximations to the reachable set. The latter is a unique feature of our tool: all the other software packages we are aware of are limited to outer approximations. By combining outer and lower approximations, ARIADNE can address more complex verification problems, like the dominance checking problem discussed in Section 6 and Section 7.

However, this high expressivity is also the main reason for some shortcomings in our reachability routines. Handling nonlinearity requires a high accuracy and degree of control on the approximation errors. This is especially important for lower approximations, where large errors severely reduce the falsification capability of the tool. For this reason we chose to use a Taylor Set enclosure type, which is general enough to approximate with the required accuracy non-convex and non-linear flows. Unfortunately, there are some important operations, like computing unions and intersections, which are very difficult, if not impossible at all, to be implemented on Taylor Sets. This calls for an additional discretization step to approximate regions of space by means of a more tractable data structure. The current version of ARIADNE uses a variable-step grid structure to discretize the state space of the system under verification, represented using Binary Decision Diagrams to improve efficiency and memory footprint, as described in Section 5.3. In this setting, to determine the accuracy parameters and the frequency of discretization events in an efficient and effective way proves to be a challenging problem, limiting the scalability of the computational engine.

The recent introduction of support-function based set representation techniques increased substantially the scalability of the verification of linear hybrid systems [11], and showed that the choice of the correct representation is crucial for this kind of problems. For this reason we are exploring a number of directions for improving performances and scalability of our tool. One possibility is to change the representation of enclosures. The use of polytopes (like in PhaVer) is appealing, but it still does not offer a clear way to properly control the over-approximation error as a function of the (nonlinear) dynamics. A possible alternative is to use support functions (as in SpaceEx), but it is not clear whether they can be extended to the nonlinear case or not. Morever, we realized that the grid-based approach for discretization has drawbacks, and we will address this problem by using a more accurate data structure, like template polyhedra or other binary space partitioning techniques. Also, when using reachability analysis specifically for safety verification, we plan to adopt more efficient dedicated approaches, such as abstraction refinement, as done in HSOLVER for the nonlinear case.

In terms of additional features, we are working on extending the computational engine in order to support more complex ways of composing hybrid automata, on further automating the verification loop and on interfacing it with third-party description languages for hybrid systems.

ACKNOWLEDGEMENTS

This research was partly supported by the EU projects FP7-ICT-217069 COCONUT and FP7-ICT-223844 CON4COORD.

REFERENCES

1. Alur R, Courcoubetis C, Halbwachs N, Henzinger TA, h Ho P, Nicollin X, Olivero A, Sifakis J, Yovine S. The algorithmic analysis of hybrid systems. *Theoretical Computer Science* 1995; **138**:3–34.
2. Maler O, Manna Z, Pnueli A. From timed to hybrid systems. *Real-Time: Theory in Practice*, vol. 600, de Bakker JW, Huizing C, de Roever WP, Rozenberg G (eds.), Springer-Verlag, 1991; 447–484.
3. Collins P. Continuity and computability of reachable sets. *Theoretical Computer Science* 2005; **341**:162–195.
4. Yovine S. Kronos: a verification tool for real-time systems. *Int. J. on Software Tools for Technology Transfer* 1997; **1**(1–2):123–133.
5. Larsen KG, Pettersson P, Yi W. UPPAAL in a nutshell. *Int. J. on Software Tools for Technology Transfer* 1997; **1**(1–2):134–152.
6. Henzinger TA, Ho PH, Wong-Toi H. HYTECH: a model checker for hybrid systems. *Int. J. on Software Tools for Technology Transfer* 1997; **1**(1–2):110–122.
7. Asarin E, Dang T, Maler O. The d/dt tool for verification of hybrid systems. *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, Springer-Verlag, 2002; 365–370.
8. Botchkarev O, Tripakis S. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. *Proceedings of Hybrid Systems: Computation and Control (HSCC'00)*, *LNCS*, vol. 1790, Springer, 2000; 73–88.
9. Tiwari A. Abstractions for hybrid systems. *Formal Methods in System Design* 2008; **32**(1):57–83.
10. Frehse G. Phaver: algorithmic verification of hybrid systems past hytech. *International Journal on Software Tools for Technology Transfer (STTT)* 2008; **10**:263–279.
11. Frehse G, Le Guernic C, Donzé A, Cotton S, Ray R, Lebeltel O, Ripado R, Girard A, Dang T, Maler O. Spaceex: Scalable verification of hybrid systems. *Proc. 23rd International Conference on Computer Aided Verification (CAV 2011)*, *LNCS*, vol. 6806. Springer Berlin / Heidelberg, 2011; 379–395.
12. Clarke E, Fehnker A, Han Z, Krogh B, Ouaknine J, Stursberg O, Theobald M. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Internat. J. Found. Comput. Sci.* 2003; **14**(4):583–604.
13. Ratschan S, She Z. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Transactions in Embedded Computing Systems* 2007; **6**(1).
14. Platzer A, Quesel JD. Keymaera: A hybrid theorem prover for hybrid systems. *Proc. of the Third International Joint Conference on Automated Reasoning (IJCAR 2008)*, *Lecture Notes in Computer Science*, vol. 5195. Springer Berlin / Heidelberg, 2008; 171–178.
15. Donzé A. Breach, a toolbox for verification and parameter synthesis of hybrid systems. *Computer Aided Verification*, *Lecture Notes in Computer Science*, vol. 6174, Touili T, Cook B, Jackson P (eds.), Springer, 2010; 167–170.
16. Dellnitz M, Froyland G, Junge O. The algorithms behind GAIO-set oriented numerical methods for dynamical systems. *Ergodic theory, analysis, and efficient simulation of dynamical systems*. Springer, 2001; 145–174, 805–807.
17. Makino K, Berz M. Cosy infinity version 9. *Nuclear Instruments and Methods* 2006; **A558**:346–350.
18. Tomlin C, Mitchell I, Bayen A, Oishi M. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE* 2003; **91**(7):986–1001.
19. Weihrauch K. *Computable analysis - An introduction*. Texts in Theoretical Computer Science, Springer-Verlag: Berlin, 2000.
20. Ariadne: An open tool for hybrid system analysis. `http://ariadne.parades.rm.cnr.it`.
21. Lynch N, Segala R, Vaandrager F. Hybrid I/O automata. *Information and Computation* 2003; **185**(1):105 – 157, doi:10.1016/S0890-5401(03)00067-1.
22. Henzinger TA, Kopke PW, Puri A, Varaiya P. What's decidable about hybrid automata? *Journal of Computer and System Sciences* 1998; **57**(1):94 – 124, doi:10.1006/jcss.1998.1581.
23. Collins P. Computable analysis with applications to dynamic systems. *Technical Report MAC-1002*, Centrum voor Wiskunde en Informatica (CWI) 2010. URL `http://oai.cwi.nl/oai/asset/16557/16557D.pdf`.
24. Benvenuti L, Ferrari A, Mazzi E, Sangiovanni-Vincentelli AL. Contract-based design for computation and verification of a closed-loop hybrid system. *HSCC '08: Proceedings of the 11th international workshop on Hybrid*

Systems, *LNCS*, vol. 4981, Springer-Verlag, 2008; 58–71, doi:10.1007/978-3-540-78929-1_5.

25. Collins P. Semantics and computability of the evolution of hybrid systems. *SIAM J. Control Optim.* 2011; **49**:890–925.
26. Collins P, Lygeros J. Computability of finite-time reachable sets for hybrid systems. *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005; 4688–4693.
27. Goebel R, Teel A. Solutions to hybrid inclusions via set and graphical convergence with stability theory applications. *Automatica* 2006; **42**(4):573 – 587, doi:10.1016/j.automatica.2005.12.019.
28. Brattka V, Hertling P, Weihrauch K. A tutorial on computable analysis. *New Computational Paradigms*, Cooper SB, Löwe B, Sorbi A (eds.). Springer, 2008; 425–491.
29. Li M, Vitányi PM. *An Introduction to Kolmogorov Complexity and Its Applications*. Texts in Computer Science, Springer, 2008.
30. Balluchi A, Casagrande A, Collins P, Ferrari A, Villa T, Sangiovanni-Vincentelli A. Ariadne: a framework for reachability analysis of hybrid automata. *Proc. of the 17th Int. Symp. on Mathematical Theory of Networks and Systems (MTNS 2006)*, 2006.
31. Benvenuti L, Bresolin D, Casagrande A, Collins P, Ferrari A, Mazzi E, Sangiovanni-Vincentelli A, Villa T. Reachability computation for hybrid systems with Ariadne. *Proc. of the 17th IFAC World Congress*, Seul, South Korea, 2008; 8960–8965.
32. Collins P, Bresolin D, Geretti L, Villa T. Computing the evolution of hybrid systems using rigorous function calculus. *Proc. of the 4th IFAC Conference on Analysis and Design of Hybrid Systems (ADHS12)*, Eindhoven, The Netherlands, 2012; 284–290.
33. Bryant R. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* 1986; **C-35**(8):677–691, doi:10.1109/TC.1986.1676819.
34. BuDDy: A BDD package. `http://buddy.sourceforge.net/`.
35. Gössler G, Graf S, Majster-Cederbaum M, Martens M, Sifakis J. An approach to modelling and verification of component based systems. *SOFSEM 2007: Theory and Practice of Computer Science* 2007; :295–308.
36. Benvenuti L, Ferrari A, Mangeruca L, Mazzi E, Passerone R, Sofronis C. A contract-based formalism for the specification of heterogeneous systems. *Proc. of the Forum on Specification, Verification and Design Languages FDL'08.*, 2008; 142–147, doi:10.1109/FDL.2008.4641436.
37. Rockafellar RT. *Convex Analysis*. Princeton Landmarks in Mathematics and Physics, Princeton University Press, 1996.