Formal verification of hybrid systems by reachability analysis with Ariadne

Tiziano Villa\textsuperscript{1} and Davide Bresolin\textsuperscript{2}

\textsuperscript{1}University of Verona, Italy
\textsuperscript{2}University of Padova, Italy

Hybrid systems: Computation and Control
La Sapienza Rome University
September 22, 2018
Outline

1. Introduction
2. Computability of hybrid automata
3. Hybrid evolution with Ariadne
4. Infinite-time evolution with Ariadne
5. Verification
6. Extending Ariadne to differential inclusions
7. Conclusions and Future Work
The formal verification flow

Model of the System
(using hybrid automata)

Properties of Interest
(using a specification language)

Verification Algorithm
(using reachability analysis)

Answer / Counterexample
Modeling a real system

Many real systems have a double nature. They:

- evolve in a **continuous** fashion
- are controlled by a **discrete** system

Such systems are called **hybrid systems** and can be modeled using the formalism of **hybrid automata**
A hybrid automaton $H$ is a finite-state automaton with continuous variables $Z$

A state is a couple $(\ell, r)$ where $r$ is a valuation for $Z$
Hybrid automata

Functional representation

- **dynamics** $\text{Dyn}|_{\ell}$: evolution of the variables in location $\ell$
- **invariant** $\text{Inv}|_{\ell}$: conditions under which continuous evolution is allowed in location $\ell$
- **guard** $\text{Gua}|_{e}$: conditions under which discrete evolution is allowed according to event $e$
- **reset** $\text{Res}|_{e}$: transformation of the continuous state after event $e$
Reachability analysis

Reachability

It represents the set $Re$ of states reached by the evolution of a system (usually cast as an Initial Value Problem).

How to use it

If we can convert properties into sets, we can compare them with $Re$ (for inclusion or intersection) to prove or disprove the properties.
Outline

1. Introduction
2. Computability of hybrid automata
3. Hybrid evolution with ARIADNE
4. Infinite-time evolution with ARIADNE
5. Verification
6. Extending ARIADNE to differential inclusions
7. Conclusions and Future Work
Computing on continuous spaces

“Classical” computability theory
- is a function on the natural numbers $f : \mathbb{N}^n \mapsto \mathbb{N}^m$
  computable by a Turing Machine?

What happens for functions on continuous spaces?
- e.g. function on the reals $f : \mathbb{R}^n \mapsto \mathbb{R}^m$
- how do we represent inputs and outputs?
- how are computations performed?
- which classes of functions are computable? And which are not?
Computable Analysis
A different notion of computability

- Introduced by Klaus Weihrauch and co-workers
- Computation is performed by Turing Machines acting on infinite streams of data
- Data streams encode a sequence of approximations to some quantity
- A function 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
- Finite computations are obtained by terminating when a given accuracy criterion is satisfied:
  - computable functions can be approximated to any desired accuracy
A simple problem

Let \( p(x) \) be a polynomial with rational coefficients: given a value for \( x \), is \( p(x) = 0 \) ?

- **Classical computability**: if \( x \) is a rational, then the problem is decidable.
- **Computable analysis**: if \( x \) is a real number, then the problem is semi-decidable:
  - when \( p(x) \neq 0 \) we can find a sufficiently accurate \( \tilde{x} \) to give a negative answer
  - when \( p(x) = 0 \), no matter how accurate \( \tilde{x} \) is, we cannot exclude the possibility that \( p(x) \neq 0 \), and thus we cannot give a positive answer
Only **continuous functions** are computable, with respect to a given representation for the data and to the corresponding topology.

- **a necessary** (but not sufficient) condition:
  - if a function is discontinuous, then it is uncomputable
  - a continuous function may be uncomputable

- **The choice of the representation** is essential:
  - we can make a function computable by requiring **more information on the inputs**, and/or **less information on the outputs**
Are hybrid automata computable?

**Theorem (Collins 2011)**

*For any coherent semantics of evolution, the finite-time reachable set of a hybrid automaton is uncomputable.*

- Discrete transitions can cause discontinuities in both space and time, even for simple systems.
- By the fundamental theorem of computable analysis, this means that the reachable set of hybrid automata is, in general, uncomputable.
Can we recover computability?

- By imposing restrictions on dynamics, reset functions, guards and invariants we can regularize the evolution to make it approximable either from above or from below.

... however ...

- The conditions for over-approximating the reachable set are different from the ones for under-approximating it.
- We can only obtain a semi-decidable problem.
Upper and lower semantics

Definitions

Theorem

Given an Hybrid Automaton with continuous dynamics and reset functions:

Upper semantics if guards and invariants are closed, then the finite-time reachable set is approximable from above;

Lower semantics if guards and invariants are open, then the finite-time reachable set is approximable from below.
Consider a location $l_0$ with invariant $x \leq a$ and a transition that leaves $l_0$ when $x \geq b$
Approximations to the reachable set

Given a hybrid automaton $H$ and an initial set $I$, it is possible to compute two approximations of the reachable set $Re$:

- Using **upper semantics**: an outer approximation $O$ of the states reached by $H$ starting from $I$ such that:

  $Re \subset O$

- Using **lower semantics**: for a given $\varepsilon > 0$, an $\varepsilon$-lower approximation $L_{\varepsilon}$ of the states reached by $H$ starting from $I$ such that:

  $\forall x \in L_{\varepsilon} \exists y \in Re \; \|x - y\| \leq \varepsilon$

$L_{\varepsilon}$ is an overapproximation of a subset of $Re$. 
Approximations to the reachable set

Given a hybrid automaton $H$ and an initial set $I$, it is possible to compute two approximations of the reachable set $Re$:

- Using **upper semantics**: an outer approximation $O$ of the states reached by $H$ starting from $I$ such that:

  $$Re \subset O$$

- Using **lower semantics**: for a given $\varepsilon > 0$, an $\varepsilon$-lower approximation $L_\varepsilon$ of the states reached by $H$ starting from $I$ such that:

  $$\forall x \in L_\varepsilon \exists y \in Re \text{ s.t. } ||x - y|| \leq \varepsilon$$

$L_\varepsilon$ is an overapproximation of a subset of $Re$. 
Property satisfaction in terms of sets

- $S_1$, $S_2$ are sets within which a property is satisfied

- $O \subset S_1 \rightarrow Re \subset S_1$

- $||L_\epsilon - S_2|| > \epsilon \rightarrow Re \not\subset S_2$
**Property satisfaction in terms of sets**

- $S_1$, $S_2$ are sets within which a property is satisfied
- $O \subset S_1 \rightarrow Re \subset S_1$
- $||L_\varepsilon - S_2|| > \varepsilon \rightarrow Re \not\subseteq S_2$

If for a given set of accuracy parameters no answer is found, we can recalculate the approximations with a finer accuracy.
Outline

1. Introduction
2. Computability of hybrid automata
3. Hybrid evolution with Ariadne
4. Infinite-time evolution with Ariadne
5. Verification
6. Extending Ariadne to differential inclusions
7. Conclusions and Future Work
Introduction to **ARIADNE**

- Developed by a joint team including the University of Verona, the University of Maastricht and the University of Padova.

- Based on a rigorous mathematical semantics for the numerical analysis of continuous and hybrid systems.

- Exploits reachability analysis to prove properties of nonlinear systems, especially for safety verification.

Functions are used to represent Hybrid Automata:

- For every discrete location, a function \( \text{Dyn} : \mathbb{R}^n \mapsto \mathbb{R}^n \) is used to represent the continuous dynamics.

- Invariants are represented using single-valued functions \( \text{Inv} : \mathbb{R}^n \mapsto \mathbb{R} \) that are negative exactly when the invariant is true.

- Discrete transitions are represented using a function \( \text{Act} : \mathbb{R}^n \mapsto \mathbb{R} \) that is positive when the guard of the transition is true (and negative otherwise), and a reset function \( \text{Res} : \mathbb{R}^n \mapsto \mathbb{R}^n \).

And also regions of space for the evolution of an automaton.
Representing functions with Taylor sets

At a glance

- Defined by a polynomial $p : [-1, 1]^n \mapsto \mathbb{R}^m$
- We represent a function or a region of space as a Taylor set plus an uniform error term $e$, which accumulates any overapproximation error

Operations

- Algebraic operations between Taylor sets use results from Interval Analysis: efficient
- Inclusion and intersection checks may require iterative splitting of the set: inefficient
Taylor sets: an example

Set: $[-1, 1]^2 \mapsto \mathbb{R}^2$

$x = p_0 + 0.25p_1 \pm 0$

$y = 0.5p_0 + p_1 \pm 0$

Set: $[-1, 1]^3 \mapsto \mathbb{R}^2$

$x = p_0 + 0.25p_1 \pm 0$

$y = 0.5p_0 + p_1 + p_2 \pm 0$

or $y = 0.5p_0 + p_1 \pm 1$

Its bounding box:

$[-1, 1]^2 \mapsto \mathbb{R}^2$

$x = 1.25p_0 \pm 0$

$y = 2.5p_1 \pm 0$
Accuracy control of a Taylor set

Numerical thresholds available

Allow to decide if a polynomial term should be added into $e$

1. Maximum polynomial order
2. Minimum coefficient value
Accuracy control of a Taylor set

Numerical thresholds available

Allow to decide if a polynomial term should be added into $e$

1. Maximum polynomial order
2. Minimum coefficient value

Reconditioning operations

Trade between domain space complexity $n$ and accuracy

a. Convert $e$ into an additional parameter $\rightarrow$ increase $n$
b. Sweep all terms where a parameter appears into $e$ $\rightarrow$ reduce $n$
Accuracy control of a Taylor set

Numerical thresholds available
- Allow to decide if a polynomial term should be added into $e$
  - 1. Maximum polynomial order
  - 2. Minimum coefficient value

Reconditioning operations
- Trade between domain space complexity $n$ and accuracy
  - a. Convert $e$ into an additional parameter $\rightarrow$ increase $n$
  - b. Sweep all terms where a parameter appears into $e$ $\rightarrow$ reduce $n$

Splitting and joining operations
- Trade between accuracy and scalability
  - Splitting issue: the subsets are usually not disjoint
  - Joining issue: it may introduce a large overapproximation
Continuous step

1. From the starting set, given a time step $h$, identify the bounding box of the flow (using Picard iteration)
2. Construct the flow function from the bounding box and the dynamics (using automatic differentiation)
3. Apply to the whole $[0, h]$ time interval to get the reached set
4. Apply to the $h$ time value to get the final set of the step
Hybrid evolution of a set

**Continuous step**

1. From the starting set, given a time step $h$, identify the bounding box of the flow (using Picard iteration)
2. Construct the flow function from the bounding box and the dynamics (using automatic differentiation)
3. Apply to the whole $[0, h]$ time interval to get the reached set
4. Apply to the $h$ time value to get the final set of the step

**Discrete step**

1. From the flow function, identify intersections with guard sets
2. Compute the crossing time with the guards
3. Compute the intersection with the guards
4. Apply the reset and change the location
Finite-time evolution
A sequence of continuous and discrete steps
Finite-time evolution
A sequence of continuous and discrete steps
Finite-time evolution
A sequence of continuous and discrete steps
Finite-time evolution
A sequence of continuous and discrete steps
Finite-time evolution
A sequence of continuous and discrete steps
Outline

1. Introduction
2. Computability of hybrid automata
3. Hybrid evolution with ARIADNE
4. Infinite-time evolution with ARIADNE
5. Verification
6. Extending ARIADNE to differential inclusions
7. Conclusions and Future Work
Infinite vs finite time evolution

Infinite-time evolution in practice

A sequence of finite-time evolutions, which terminates if no additional state space can be reached after a while.

Finite time is simple, but may not be usable

Using finite time evolution to verify a system which evolves for infinite time requires the manual identification of a time interval that still gives formal guarantees.

- Example: if the behavior is guaranteed to be periodic, analyze only one period.

In general, to verify some properties of the system we need to evolve the system for infinite time.
To obtain convergence, we have two requirements:

1. Be able to identify when no new state space is reached;
2. Control the growth of the overapproximation error.
Convergence for infinite-time evolution

To obtain convergence, we have two requirements:

1. Be able to identify when no new state space is reached;
2. Control the growth of the overapproximation error.

We need a set representation with

- operations like subtraction, intersection, splitting and merging;
- small memory usage, fast operations and good scalability;
- small overapproximation error.
### Definition (Grid)

A coordinate-aligned discrete partitioning of the variables space, which identifies **cells** of different sizes.

### Definition (Grid set)

A marking of cells locked to a grid.
Representation of sets using a grid

**Definition (Grid)**
A coordinate-aligned discrete partitioning of the variables space, which identifies cells of different sizes.

**Definition (Grid set)**
A marking of cells locked to a grid.
Grid sets - pros and cons

Pros

- Converts easily from/to a Taylor Set;
- Allows a compact internal representation, e.g. a binary tree or a binary decision diagram;
- Cells $2^N$ times larger/smaller than a basic cell can be marked;
- Union, intersection, difference and inclusion can be performed efficiently.
## Grid sets - pros and cons

### Pros

- Converts easily from/to a Taylor Set;
- Allows a compact internal representation, e.g. a binary tree or a binary decision diagram;
- Cells $2^N$ times larger/smaller than a basic cell can be marked;
- Union, intersection, difference and inclusion can be performed efficiently.

### Cons

- They are coarse when using large cell sizes;
- Using small cell sizes is computationally demanding, especially for a large number of variables.
Infinite-time reachability at a glance

1. Identify a bounding set $B$ to constrain evolution;
2. Solve numerically the system (using point-based evolution) to identify the times when the grid set representation must be updated;
3. Compute the finite-time hybrid evolution of the automaton up to the next grid update time;
4. If the reached set is partially outside the bounding set, stop with failure;
5. If new cells have been found in this iteration, resume from (3);
6. Stop with success.
Specific comments on computing $L_\varepsilon$ (1)

We cannot split a set

We would lose the information on which cells contain points of the reachable set: $||Re - L_\varepsilon||$ would be untrackable. This means we cannot resume evolution from a discretized final set.

- We maintain a separate discretized reached set, which is updated in parallel only to check if no new cells are reached. Hence we do not resume from the final cells, but from the original final Taylor set.
Specific comments on computing $L_\varepsilon$ (1)

We cannot split a set

We would lose the information on which cells contain points of the reachable set: $||Re - L_\varepsilon||$ would be untrackable. This means we cannot resume evolution from a discretized final set.

- We maintain a separate discretized reached set, which is updated in parallel only to check if no new cells are reached. Hence we do not resume from the final cells, but from the original final Taylor set.

We do not allow spurious transitions

Again, $||Re - L_\varepsilon||$ would be untrackable. Therefore multiple or undecidable transitions must stop evolution.
Specific comments on computing $L_\varepsilon$ (2)

$L_\varepsilon$ of a subset of $\mathbb{R}e$ is still a valid $L_\varepsilon$ of $\mathbb{R}e$

An empty set for example is valid, while uninformative.

- The termination clause can be relaxed: we may stop as soon as the set width becomes as large as $\varepsilon$
- For efficiency purposes we can evolve subsets of initial sets: particularly useful when large sets would be involved
Specific comments on computing $L_\varepsilon$ (2)

$L_\varepsilon$ of a subset of $Re$ is still a valid $L_\varepsilon$ of $Re$

An empty set for example is valid, while uninformative.

- The termination clause can be relaxed: we may stop as soon as the set width becomes as large as $\varepsilon$
- For efficiency purposes we can evolve subsets of initial sets: particularly useful when large sets would be involved

An invalid $B$ can be detected

If the reached set is outside $B$ more than $\varepsilon$, then $B$ is too restrictive in respect to $Re$ and should be enlarged.
The watertank example

- Outlet flow $F_{out}$ depends on the water level $x(t)$:
  $$F_{out}(t) = \lambda \sqrt{x(t)}$$

- Inlet flow $F_{in}$ is controlled by the valve position $\alpha(t)$:
  $$F_{in}(t) = K_p \cdot \alpha(t)$$

- The controller senses the water level and sends the appropriate commands to the valve.
The watertank control loop

Controller

Actuator: valve

Plant: tank

Sensor

command

\( x_s(t) \)

\( u(t) \)

\( x(t) \)

\( p(t) \)

\( \delta \)
Modeling the water tank

### Tank Automaton

\[ \dot{x}(t) = -\lambda \sqrt{x(t)} + u(t) \]

- \( x(0) \leq H \leq \lambda \sqrt{H} \)
- \( x(t) = H \)
- \( u(t) \geq \lambda \sqrt{H} \)

### Sensor Automaton

\[ x_s(t) = x(t) + \delta(t) \]

### Controller Automaton

\[ \dot{\alpha}(t) = \begin{cases} 0 & \text{if } 0 \leq \alpha \leq 1 \end{cases} \]

\[ \dot{\alpha}(t) = \begin{cases} 1/T & \text{if } \alpha = 0 \end{cases} \]

\[ \dot{\alpha}(t) = \begin{cases} -1/T & \text{if } \alpha \text{ is open} \end{cases} \]

\[ \dot{\alpha}(t) = \begin{cases} 0 & \text{if } \alpha \text{ is close} \end{cases} \]

### Valve Automaton

\[ \dot{\alpha}(t) = \begin{cases} 1/T & \text{if } \alpha = 0 \end{cases} \]

\[ \dot{\alpha}(t) = \begin{cases} -1/T & \text{if } \alpha \text{ is open} \end{cases} \]

\[ \dot{\alpha}(t) = \begin{cases} 0 & \text{if } \alpha \text{ is close} \end{cases} \]
The water tank automaton

\[ \dot{x}(t) = -\lambda \sqrt{x(t)} + u(t) \]

\[ 0 \leq x \leq H \]

\[ x = H \land u \geq \lambda \sqrt{H} \]

\[ \dot{x}(t) = 0 \]

\[ x(t) = H \land u \leq \lambda \sqrt{H} \]

\[ x(t) = H \land u \geq \lambda \sqrt{H} \]

\[ x(t) \text{ is the water level, } u(t) \text{ is the inlet flow.} \]

\[ q_1 \text{ represents the situation when there is no water overflow.} \]

\[ q_2 \text{ represents the situation when there is water overflow.} \]
The water tank automaton

- $\lambda \sqrt{H}$ is the largest outflow $\lambda \sqrt{x}$ when $x = H$.
- $x = H \land u > \lambda \sqrt{H}$ is the case when the water is at the top level $H$ and the inflow $u$ is larger than the largest outflow $\lambda \sqrt{H}$.
- when in $q_2$, if (by the action of the controller) the valve angle decreases, then $u$ decreases to the point that the invariant $x = H \land u > \lambda \sqrt{H}$ is not true anymore, and so the transition to $q_1$ is taken under the guard $x = H \land u \leq \lambda \sqrt{H}$.
The sensor automaton

\[ x_s(t) = x(t) + \delta(t) \]

- The input is the real water level \( x(t) \) provided by the tank.
- The output is the measured water level \( x_s(t) = x(t) + \delta \) for the controller (where \( \delta \) is an interval \( (-\delta_1, \delta_1) \)).
A simple hysteresis controller

- The input is the measured water level $x_s(t)$ provided by the sensor.
- The output is the command signal $\text{open}$ or $\text{close}$ for the valve.

- The controller produces the $\text{open}$ command when $x_s(t) \leq l$, and it produces the $\text{close}$ command when $x_s(t) \geq h$.
- $l$ and $h$ are lower and upper water levels.
In response to a command, the valve aperture changes linearly in time with rate $1/T$. 
The valve automaton

- The pressure \( p(t) \) is assumed to be any constant value in an interval \([p_1, p_2]\), where \( p_1 \) and \( p_2 \) are respectively the minimum and the maximum of \( p(t) \) over a time interval of interest.

- One may assume \( f(p(t)) = k \sqrt{p} \), where \( p \) is a constant value from an interval (see above) and so it can be used also in a linear model.
The complete watertank automaton

\[ \dot{x}(t) = -\lambda \sqrt{x(t)} + \alpha(t) f(p(t)) \]
\[ \dot{\alpha}(t) = 1/T \]
\[ 0 \leq x(t) \leq h - \delta(t) \]
\[ 0 \leq \alpha(t) \leq 1 \]

**l\(_0\)**

\[ \dot{x}(t) = -\lambda \sqrt{x(t)} \]
\[ \dot{\alpha}(t) = 0 \]
\[ l - \delta(t) \leq x(t) \leq H \]
\[ \alpha(t) = 0 \]

**l\(_2\)**

\[ \dot{x}(t) = -\lambda \sqrt{x(t)} + \alpha(t) f(p(t)) \]
\[ \dot{\alpha}(t) = -1/T \]
\[ l - \delta(t) \leq x(t) \leq H \]
\[ 0 \leq \alpha(t) \leq 1 \]

\[ \alpha = 0 \]

\[ x = H \wedge \]
\[ u \geq \lambda \sqrt{H} \]

**l\(_3\)**

\[ \dot{x}(t) = -\lambda \sqrt{x(t)} + \alpha(t) f(p(t)) \]
\[ \dot{\alpha}(t) = 0 \]
\[ 0 \leq x(t) \leq h - \delta(t) \]
\[ \alpha(t) = 1 \]

\[ x = H \wedge \]
\[ u \geq \lambda \sqrt{H} \]

**l\(_4\)**

\[ \dot{x}(t) = 0 \]
\[ \dot{\alpha}(t) = -1/T \]
\[ x = H \wedge \]
\[ u \leq \lambda \sqrt{H} \]
\[ 0 \leq \alpha(t) \leq 1 \]
The simplified watertank automaton
No overflow, reduced transitions

**open**
\[ \dot{x} = \lambda \sqrt{x} + K_p \alpha \]
\[ \alpha = 1 \]
\[ x \leq h_{max} + \delta \]

**closing**
\[ \dot{x} = \lambda \sqrt{x} + K_p \alpha \]
\[ \dot{\alpha} = -\frac{1}{T} \]
\[ \alpha \geq 0 \]

\[ x \geq h_{max} - \delta \]

**opening**
\[ \dot{x} = \lambda \sqrt{x} + K_p \alpha \]
\[ \dot{\alpha} = 1/T \]
\[ \alpha \leq 1 \]

\[ x \leq h_{min} + \delta \]

**closed**
\[ \dot{x} = \lambda \sqrt{x} \]
\[ \alpha = 0 \]
\[ x \geq h_{min} - \delta \]

\[ \alpha \leq 0 \]
Reachability results

On the watertank example

Upper Semantics

$\alpha$

Lower Semantics

$\alpha$

coarse

fine
1. Introduction
2. Computability of hybrid automata
3. Hybrid evolution with ARIADNE
4. Infinite-time evolution with ARIADNE
5. Verification
6. Extending ARIADNE to differential inclusions
7. Conclusions and Future Work
Assume-guarantee system specification

- The system is specified as a set of components.
- Every component is annotated with a pair \((A_i, G_i)\) of assumptions and guarantees.
- The requirements \((A, G)\) of the whole system are decomposed into a set of simpler requirements \((A_i, G_i)\) that, if satisfied, guarantee that the overall requirements \((A, G)\) are satisfied.
Safety checking

Let C be a component of the system, annotated with assumptions A and guarantees G. With Ariadne we can verify whether the component C respects the safety guarantees G or not given the assumptions A.

- Represent the component C by a hybrid automaton \( H \) with inputs and outputs.
- Assumptions A are represented by a hybrid automaton \( H_A \) that specifies the possible inputs \( U \) for \( H \).
- Guarantees G specify the possible outputs \( Y \) of automaton \( H \).

This is a reachability analysis problem:

\[
\text{Reach}(H \| H_A) \subseteq \text{Sat}(G).
\]
1. Compute an outer-approximation $O$ of $\text{Reach}(H\|H_A)$ using a grid of a given size.

2. If $O \subseteq \text{Sat}(G)$, the system is verified to be safe. Exit with success.

3. Otherwise, compute an $\varepsilon$-lower approximation $L_\varepsilon$ of $\text{Reach}(H\|H_A)$. The value of $\varepsilon$ depends on the size of the grid (typically, $\varepsilon$ is a small multiple of the size of a grid cell).

4. If there exists at least a point in $L_\varepsilon$ that is outside $\text{Sat}(G)$ by more than $\varepsilon$, the system is verified to be unsafe. Exit with failure.

5. Otherwise, set the grid to a finer size and restart from point 1.
Verifying the water tank

Safety property: the water level between 5.25 and 8.25 meters.

First iteration:
grid $1/8 \times 1/80$ ($x$-axis: $x(t)$, $y$-axis: $\alpha(t)$).

Outer reach is not safe, try lower reach.

Green: safe set  Orange: $\varepsilon$-tolerance  Red: computed set
Verifying the water tank

Safety property: the water level between 5.25 and 8.25 meters.

First iteration:
grid $1/8 \times 1/80$
($x$-axis: $x(t)$, $y$-axis: $\alpha(t)$).

Lower reach is not unsafe, refine grid.

Green: safe set  Orange: $\varepsilon$-tolerance  Red: computed set
Verifying the water tank

**Safety property:** the water level between 5.25 and 8.25 meters.

Second iteration:
grid $1/16 \times 1/160$
($x$-axis: $x(t)$, $y$-axis: $\alpha(t)$).

Outer reach is not safe, try lower reach.

**Green:** safe set  **Orange:** $\varepsilon$-tolerance  **Red:** computed set
Verifying the water tank

Safety property: the water level between 5.25 and 8.25 meters.

Second iteration: grid $1/16 \times 1/160$ ($x$-axis: $x(t)$, $y$-axis: $\alpha(t)$).

Lower reach is not unsafe, refine grid.

Green: safe set  Orange: $\varepsilon$-tolerance  Red: computed set
Verifying the water tank

Safety property: the water level between 5.25 and 8.25 meters.

Third iteration: grid $1/32 \times 1/320$ ($x$-axis: $x(t)$, $y$-axis: $\alpha(t)$).

Outer reach is safe, system is proved safe.

Green: safe set  Orange: $\varepsilon$-tolerance  Red: computed set
Verifying the water tank

1. In this example, we could prove safety by outer reach.

2. Variations of the parameters could yield systems where lower reach would prove unsafety or where no conclusions could be drawn (smallest precision of the parameters reached without proving safety or unsafety).
Dominance checking

**Definition**

Given two components $C_1$ and $C_2$, with assumptions and guarantees $(A_1, G_1)$ and $(A_2, G_2)$, we say that $C_1$ dominates $C_2$ if and only if under **weaker assumptions** ($A_2 \subseteq A_1$), **stronger promises** are guaranteed ($G_1 \subseteq G_2$).

If this is the case, the component $C_2$ can be replaced with $C_1$ in the system without affecting the whole system behaviour.

Intuitively, the component $C_1$ dominates $C_2$ if it issues sharper outputs ($G_1 \subseteq G_2$) with looser inputs ($A_2 \subseteq A_1$), e.g., a dominating controller can issue a subset of the control commands to cope with an environment which is allowed more freedom.
Dominance checking by reachability analysis

1. Represent the two components by two hybrid automata $H_1$ and $H_2$ with inputs and outputs.

2. Assumptions $A_1$ and $A_2$ are represented by hybrid automata $H_{A_1}$ and $H_{A_2}$ that specify the possible inputs $U_1, U_2$ for the components.

3. Guarantees $G_1$ and $G_2$ specify the possible outputs $Y_1, Y_2$ of the automata $H_1$ and $H_2$.

4. $H_1$ dominates $H_2$ if and only if $G_1 \subseteq G_2$ and $A_2 \subseteq A_1$.

This is a *reachability analysis* problem:

$$\text{Reach}(H_{A_1} || H_1)|_{Y_1} \subseteq \text{Reach}(H_{A_2} || H_2)|_{Y_2}.$$
Dominance checking in ARIADNE

The approximate reachability routines of ARIADNE can be used to test dominance of components:

1. Compute an $\varepsilon$-lower approximation $L_2^\varepsilon$ of $\text{Reach}(H_{A_2} \parallel H_2)_{|Y_2}$.
2. Remove a border of size $\varepsilon$ from $L_2^\varepsilon$.
3. Compute an outer approximation $O_1$ of $\text{Reach}(H_{A_1} \parallel H_1)_{|Y_1}$.
4. If $O_1 \subseteq L_2^\varepsilon - \varepsilon$ then $\text{Reach}(H_{A_1} \parallel H_1)_{|Y_1} \subseteq \text{Reach}(H_{A_2} \parallel H_2)_{|Y_2}$ and thus $H_1$ dominates $H_2$.
5. If not, we cannot say anything about $H_1$ and $H_2$, and we retry with a finer approximation.
The proof of correctness of the procedure relies on the following steps:

1. $\text{Reach}(H_{A_1} \| H_1) \mid Y_1 \subseteq O_1$ by definition.
2. $O_1 \subseteq L^\varepsilon_2 - \varepsilon$ to be verified.
3. $L^\varepsilon_2 - \varepsilon \subseteq \text{Inner}_2$ under suitable hypotheses.
4. $\text{Inner}_2 \subseteq \text{Reach}(H_{A_2} \| H_2) \mid Y_2$ by definition.

Therefore $\text{Reach}(H_{A_1} \| H_1) \mid Y_1 \subseteq \text{Reach}(H_{A_2} \| H_2) \mid Y_2$ and thus $H_1$ dominates $H_2$.

A sufficient hypothesis to guarantee that $L^\varepsilon_2 - \varepsilon \subseteq \text{Inner}_2$ is that $\text{Reach}(H_{A_2} \| H_2) \mid Y_2$ is a $\varepsilon$-regular set, i.e., there are no holes “smaller than $\varepsilon$” in the set.
The water tank again

We want to replace the controller and the valve.

- The valve is slower than the previous one
- The controller is smarter and can fix the valve aperture to any value $w(t) \in [0, 1]$

Does the system still operate correctly?
The water tank again

Application of dominance relation in this example:

1. The automaton $H_1$ represents the whole system with new components (proportional controller, slower valve, sensor, plant).

2. The automaton $H_2$ represents the whole system with old components (hysteresis controller, original faster valve, sensor, plant).

3. $A_1$ and $A_2$ specify the same external input $U_1 = U_2 = p(t)$, i.e. the pressure on the valve, so it is $A_2 = A_1$.

4. $G_1$ and $G_2$ specify the same output $Y_1 = Y_2 = x(t)$, i.e., the water level of the tank, for which it is requested $G_1 \subseteq G_2$. 
A proportional controller

- The input is the measured water level $x_s(t)$ provided by the sensor.
- The output is a command signal $w(t) \in [0, 1]$ for the valve position regulation.
- The controller computes the output $w(t)$ from the measured level $x_s(t)$ and the water level reference $R$.
- In response to a command $w(t)$ the valve aperture $a(t)$ varies with the first-order linear dynamics $\dot{a}(t) = \frac{1}{\tau}(w(t) - a(t))$. 
A proportional controller

1. Location $c_0$ models when the controller saturates the opening valve command to $w(t) = 1$.

2. Location $c_1$ models when the controller tracks the water reference level $R$.

3. Location $c_2$ models when the controller saturates the closing valve command to $w(t) = 0$. 
Results

$\varepsilon$-lower approximation of the reachable set of the hysteresis controller:

Assumptions:

- Inlet pressure $p$ between 50 and 60 KPa (KiloPascal)
- Sensor’s error between $-0.05$ and $0.05$ m
Results

Outer approximation of the reachable set of the proportional controller:

Assumptions:
- Inlet pressure $p$ between 50 and 60 KPa (KiloPascal)
- Sensor's error between $-0.05$ and $0.05$ m

The proportional controller dominates the hysteresis controller.
A system can be *partially specified*

- **environmental parameters** outside the control of the designer and for which there may be imperfect knowledge
- **design parameters** that can be fixed by the designer, but whose admissible values are not necessarily known a priori

**ARIADNE** allows *parametric verification*

- exhaustively check all possible values of the parameters
- determine the value for the design parameters for which the component respects the guarantees, for all possible values of the environmental parameters
Obtained for different values of two parameters: the gain $K_P$ and the reference height $R$ of the proportional controller.

- **Green**: proportional dominates hysteretic for all points;
- **Red**: proportional does not dominate hysteretic in at least one point;
- **Yellow**: insufficient accuracy to obtain a result.
Outline

1. Introduction
2. Computability of hybrid automata
3. Hybrid evolution with Ariadne
4. Infinite-time evolution with Ariadne
5. Verification
6. Extending Ariadne to differential inclusions
7. Conclusions and Future Work
Overview of differential inclusions

The dynamics of the continuous state $x$ is such that

$$\dot{x} \in f(x, V)$$

where $V$ is a compact set that introduces **uncertainty** in the dynamics.

In other words, differential inclusions (DI) in general allow to model **unknown inputs** whose derivatives are bounded.
How to exploit differential inclusions

Modeling of noise

If the real system has uncertain parameters, we can analyze a family of behaviors of the system by using DIs to model such parameters.

+ We better match a given real system, consequently we expand the scope of rigorous numerical methods.

Decoupling from input variables

If the system model is large, we would prefer that the components of the system be analyzed separately. When a component depends on another component, we need to decouple them. Replacing dependencies with DIs satisfies such requirement.

+ We increase the size of systems that we can analyze efficiently.
An example of a system

Overview

Objective
Keep the water level $x$ in each tank within a required range, using a valve aperture control $u$.

a. The first component $C_1$ can be analyzed in isolation

b. The second component $C_2$ can be analyzed in isolation if we replace its input from $C_1$ with a DI (as resulting from the analysis of a.)

c. The sensor reading or the valve actuation can have a noise component that extends the generality of the analytical results
An example of a system
Model details

For the two components and a given discrete location, we have:

\[
\begin{align*}
\dot{x}_1 &= \alpha_1 u_1 - \beta_1 \sqrt{x_1} \\
\dot{u}_1 &= \psi_1(x_1, s, v_{1,c}) \\
\dot{x}_2 &= \alpha_2 u_2 x_1 - \beta_2 \sqrt{x_2} \\
\dot{u}_2 &= \psi_2(x_2, s, v_{2,c})
\end{align*}
\]

with \(x_{1,s} = x_1 + v_{1,s}\) and \(x_{2,s} = x_2 + v_{2,s}\)

where the constants \(\alpha, \beta\) depend on the tank and the pipe sections, \(\psi\) depends on the specific controller (e.g., hysteretic, proportional), and \(v_c, v_s\) are noise sources (see c.).

To remove the dependency from \(x_1\), we introduce an additional \(v_{2,d}\) source that overapproximates the behavior of \(C_1\) in respect to \(x_1\) (see b.).

We model \(v = \{v_{1,c}, v_{2,c}, v_{1,s}, v_{2,s}, v_{2,d}\}\) using a DI, i.e.: \(|v| \leq V\).
An example of a system
Analyzing the decoupled component

1. First, we analyze $C_1$ separately, obtaining the reachable set $Re_1$.

2. Then, we analyze the decoupled $C_2$:

   \[ \dot{z} = v_{2,d} \]
   \[ \dot{x}_2 = \alpha_2 u_2 z - \beta_2 \sqrt{x_2} \]
   \[ \dot{u}_2 = \psi_2(x_2,s,v_2,c) \]

   with $x_{2,s} = x_2 + v_{2,s}$

   where we set $v_{2,d} \in \{ \dot{x}_1(x_1,u_1) \}_{(x_1,u_1) \in Re_1}$ and $z(0) = x_1(0)$.

While $z$ is an extra variable, it allows to set an invariant on $z$: $z \in Re_1|_{x_1}$. 

The approach within Ariadne

In general, we can address input-affine systems of the form

\[ \dot{x}(t) = f(x(t)) + \sum_{i=1}^{m} g_i(x(t))v_i(t), \quad x(0) = x_0 \]

with \( \{f, g_i\} \) non-linear, \( v_i(t) \) measurable, with \( |v_i(t)| \leq V_i \).

We replace each unspecified \( v_i \) with an auxiliary function \( w_i \)

- Example: \( w_i(t) = a_i^{(0)} + a_i^{(1)}(t - (t_k + \Delta t/2)) \)
- Each auxiliary function introduces parameters \( a_i \) that increase the complexity of the set representation

Being \( w_i \) an approximation of \( v_i \), we need to compute analytical error bounds that we use to correctly overapproximate the resulting set.
The algorithm for a single continuous step

Let \( S_k = \{ h_k(s) \pm e_k \mid s \in [-1, 1]^{p_k} \} \) be an over-approximation of the set \( S(x_0, t_k) \).

1. Compute the flow \( \tilde{\phi}_k(x_k, a_k) \) of

\[
\dot{x}(t) = f(x(t)) + \sum_{i=1}^{m} g_i(x(t))w(a_{k,i}, t)
\]

for \( t \in [t_k, t_{k+1}] \), \( x_k = x(t_k) \in S_k \), and \( a_k \in V \);

2. Add the analytical error bound \( \epsilon \);

3. Compute the set \( S_{k+1} \) which approximates \( S(x_0, t_{k+1}) \).
Open issues

Handle the number of auxiliary parameters

Auxiliary functions applied on each continuous step increase the number of parameters used to represent the set.

- It is necessary to dynamically trade between complexity of representation and overapproximation error.

Handle the continuous step size

Choosing a large continuous step size, while more efficient, enlarges the reached sets and increases the overapproximation error.

- It is necessary to dynamically choose the step size.
Outline

1. Introduction
2. Computability of hybrid automata
3. Hybrid evolution with Ariadne
4. Infinite-time evolution with Ariadne
5. Verification
6. Extending Ariadne to differential inclusions
7. Conclusions and Future Work
The problem and where we stand

Reachability analysis for nonlinear systems is hard

- We can compute approximations at chosen accuracies, but
- There is no guarantee that properties can be proven with a finite accuracy, and
- There is no representation that is both scalable and accurate
The problem and where we stand

Reachability analysis for nonlinear systems is hard
- We can compute approximations at chosen accuracies, but
- There is no guarantee that properties can be proven with a finite accuracy, and
- There is no representation that is both scalable and accurate

The extension to uncertain dynamics is even harder
- Representations become more complex
- Accuracy control is paramount
Experimental evaluation

- **ARIADNE** applied to proving safety of plans in robotic surgery tasks (see Geraldes, Geretti, Bresolin, Muradore, Fiorini, Mattos, Villa, “Formal Verification of Medical CPS: a Laser Incision Case Study”, Accepted by ACM Transactions on Cyber-Physical Systems, expected publication 2018)

- Other industrial test cases
**ARIADNE: work in progress**

**Experimental evaluation**

- **ARIADNE** applied to proving safety of plans in robotic surgery tasks (see Geraldes, Geretti, Bresolin, Muradore, Fiorini, Mattos, Villa, “Formal Verification of Medical CPS: a Laser Incision Case Study”, Accepted by ACM Transactions on Cyber-Physical Systems, expected publication 2018)

- Other industrial test cases

**Extension of ARIADNE to noisy systems by solving differential inclusions**

- Continuous evolution implemented,
- Hybrid evolution under way.
ARIADNE: future work

Modeling stochastic behaviors

To extend to probabilistic requirements and system descriptions.
**ARIADNE: future work**

**Modeling stochastic behaviors**

To extend to probabilistic requirements and system descriptions.

**Extension to world automata (Segala and Capiluppi)**

To model agent systems and their implicit communication through perturbation of the environment.
ARIADNE: future work

Modeling stochastic behaviors
To extend to probabilistic requirements and system descriptions.

Extension to world automata (Segala and Capiluppi)
To model agent systems and their implicit communication through perturbation of the environment.

Synthesis
To go beyond verification to synthesize correct-by-construction controllers that satisfy some properties.
A selection of publications

- **Hybrid evolution**

- **Differential inclusions**

- **Assume-guarantee verification**

- **Contract-based design and analysis of the state of the art**