Formal verification of nonlinear hybrid systems with Ariadne

Luca Geretti and Tiziano Villa

May 2, 2024

The ARIADNE library in a nutshell

- Developed by a joint team led by the University of Verona
- Written in C++, with additional Python bindings
- Managed as a CMake project with minimal dependencies
- Supported under Linux and macOS, using Clang or GCC

- Website: `http://www.ariadne-cps.org`
- Repository: `https://github.com/ariadne-cps/ariadne`

The ARIADNE Golden Four requirements

1. Define rigorous mathematical semantics for the analysis of continuous and hybrid systems.
2. Numerical soundness in all operations.
3. Allow arbitrary accuracy by handling nonlinear behavior directly.
4. Allow proving and disproving of properties of a system.

Outline

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:
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

The fundamental theorem

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 approximation of the reachable set from above are different from the ones for approximation from below
- we can only obtain a semi-decidable problem

Upper and lower semantics

Definitions

Theorem

Given a Hybrid Automaton with continuous dynamics and reset functions:

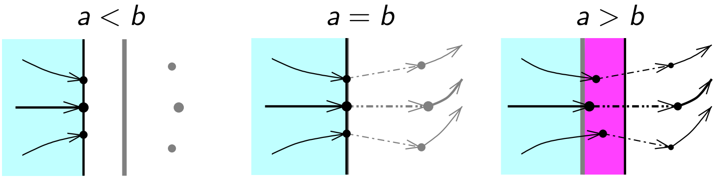
Upper semantics *if guards and invariants are closed, then the finite-time reachable set is approximable from above;*

Lower semantics *if guards and invariants are open, then the finite-time reachable set is approximable from below.*

Upper and lower semantics

An example

Consider a location l_0 with invariant $x \leq a$ and a transition that leaves l_0 when $x \geq b$



Upper: No Transition	Upper: Transition	Upper: Transition
Lower: No Transition	Lower: No Transition	Lower: Transition

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 up to a given time t (including the infinite-time case):

- an outer approximation O of the states reached by H starting from I such that:

$$Re \subset O$$

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 up to a given time t (including the infinite-time case):

- an outer approximation O of the states reached by H starting from I such that:

$$Re \subset O$$

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

$$\exists x \in Re \; s.t. ||x - L_\varepsilon|| \leq \varepsilon$$

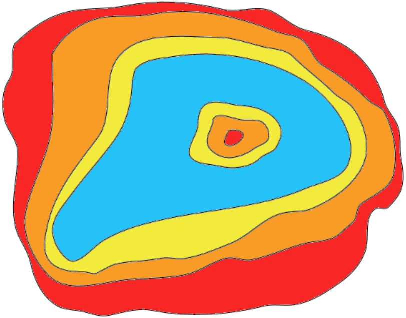
L_ε is an overapproximation of a subset of Re.

Outer approximation O

- Blue: reachable set

This is a sequence of approximations from above:

- Red + Orange + Yellow + Blue: coarse O
- Orange + Yellow + Blue: finer O
- Yellow + Blue: finest O

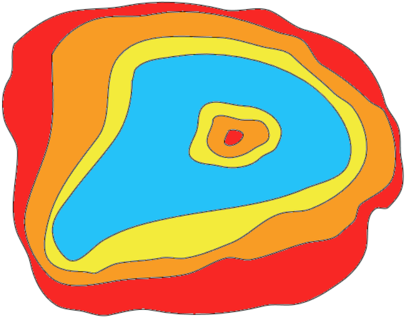


Outer approximation O

- Blue: reachable set

This is a sequence of approximations from above:

- Red + Orange + Yellow + Blue: coarse O
- Orange + Yellow + Blue: finer O
- Yellow + Blue: finest O



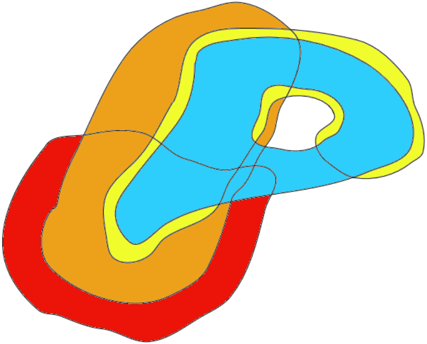
A valid, albeit useless, O is the whole continuous space.

ε-lower approximation L_ε

- Blue: reachable set

This is a sequence of approximations from below:

- Interior of outline of Red: coarse L_ε
- Interior of outline of Orange: finer L_ε
- Interior of outline of Yellow: finest L_ε

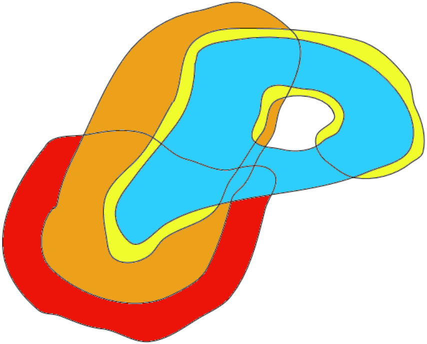


ε-lower approximation L_ε

- Blue: reachable set

This is a sequence of approximations from below:

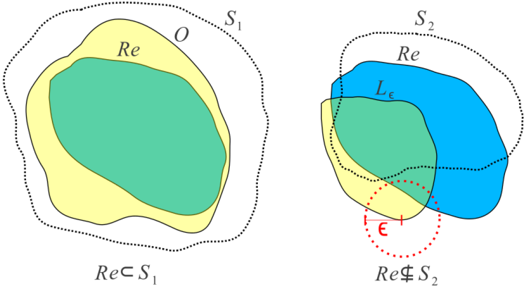
- Interior of outline of Red: coarse L_ε
- Interior of outline of Orange: finer L_ε
- Interior of outline of Yellow: finest L_ε



A valid, albeit useless, L_ε is the empty set.

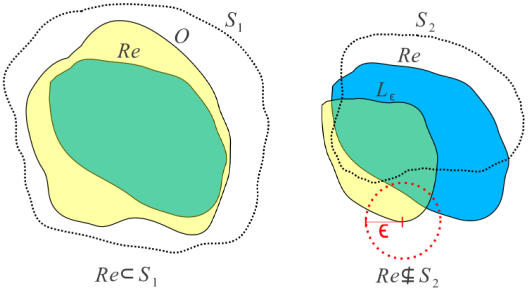
How to use approximations to verify properties

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



How to use approximations to verify properties

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



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

→ (possibly infinite) sequence of approximations

Switching between representations might be required

- Accurate representations are useful for frequent events (such as continuous steps of evolution), in order to limit accumulation of overapproximation error;

Switching between representations might be required

- Accurate representations are useful for frequent events (such as continuous steps of evolution), in order to limit accumulation of overapproximation error;
- Coarse representations are useful for sporadic events, where operations such as intersection, joining and splitting are required and would be inefficient/ineffective on accurate representations.

The role of functions

Functions can be used to represent Hybrid Automata:

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

A function along with a finite domain can also specify a region of space for the evolution of an automaton.

Representing functions in the nonlinear case

We represent f from a function mapping a parameter space into the state space: $p : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e., $\{p_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^{m}$.

- We want $p_j = \sum_{i=0}^{N_j} c_{ij}\beta_{ij}$, i.e., a linear combination of terms in a basis $\{\beta\}$ in the parameter space.

Representing functions in the nonlinear case

We represent f from a function mapping a parameter space into the state space: $p : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e., $\{p_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^m$.

- We want $p_j = \sum_{i=0}^{N_j} c_{ij}\beta_{ij}$, i.e., a linear combination of terms in a basis $\{\beta\}$ in the parameter space.
- Most common basis: monomials from all cross products, e.g. x_1, x_2, $x_1^2 x_3$, etc. which produce a Taylor Model.
    - Other bases for models: Chebyshev, Bernstein

Representing functions in the nonlinear case

We represent f from a function mapping a parameter space into the state space: $p : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e., $\{p_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^m$.

- We want $p_j = \sum_{i=0}^{N_j} c_{ij}\beta_{ij}$, i.e., a linear combination of terms in a basis $\{\beta\}$ in the parameter space.
- Most common basis: monomials from all cross products, e.g. x_1, x_2, $x_1^2 x_3$, etc. which produce a Taylor Model.
  - Other bases for models: Chebyshev, Bernstein
- Coefficients c_{ij} may be singleton reals or real intervals.

Representing functions in the nonlinear case

We represent f from a function mapping a parameter space into the state space: $p : \mathbb{R}^n \mapsto \mathbb{R}^m$, i.e., $\{p_j : \mathbb{R}^n \mapsto \mathbb{R}\}_{j=1}^m$.

- We want $p_j = \sum_{i=0}^{N_j} c_{ij}\beta_{ij}$, i.e., a linear combination of terms in a basis $\{\beta\}$ in the parameter space.
- Most common basis: monomials from all cross products, e.g. x_1, x_2, $x_1^2 x_3$, etc. which produce a Taylor Model.
  - Other bases for models: Chebyshev, Bernstein
- Coefficients c_{ij} may be singleton reals or real intervals.

Finite representation

Since the exact representation of f in general would require infinite terms, and since we need to overapproximate, we add a uniform error term e to the expansion for enclosure.

$\rightarrow$ Ultimately we have $f_j \subset p_j + e_j$ for the j-th component of f.

Advantages and drawbacks of a nonlinear basis

Advantages

- We are not limited to a convex representation;
- We can approximate arbitrarily close by increasing the number of terms and/or the number of parameters (i.e., curve segments in the boundary);
- Algebraic operations between sets use results from Interval Analysis: efficient.

Advantages and drawbacks of a nonlinear basis

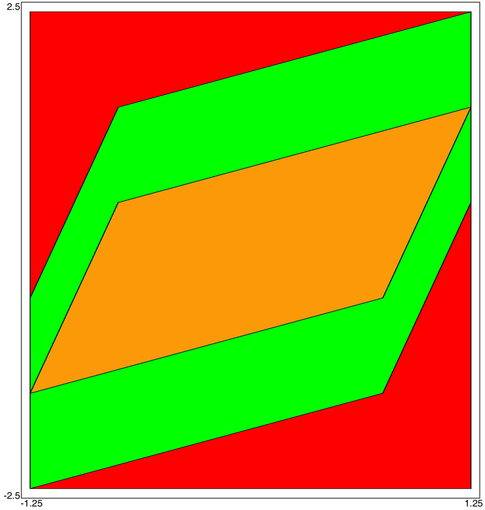
Advantages

- We are not limited to a convex representation;
- We can approximate arbitrarily close by increasing the number of terms and/or the number of parameters (i.e., curve segments in the boundary);
- Algebraic operations between sets use results from Interval Analysis: efficient.

Drawbacks

- Observation of a set is limited: we can efficiently evaluate only the bounds of the function over a domain
  - $\rightarrow$ We need to iteratively split the function to improve evaluation.
- Splitting is done on the domain space: the larger the dimension of the domain space, the more overlapping the resulting split sets.

Sets from Taylor Models: a linear example



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

$x = p_0 + 0.25 p_1 \pm 0$

$y = 0.5 p_0 + p_1 \pm 0$

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

$x = p_0 + 0.25 p_1 \pm 0$

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

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

Its bounding box: $[-1,1]^2 \mapsto \mathbb{R}^2$

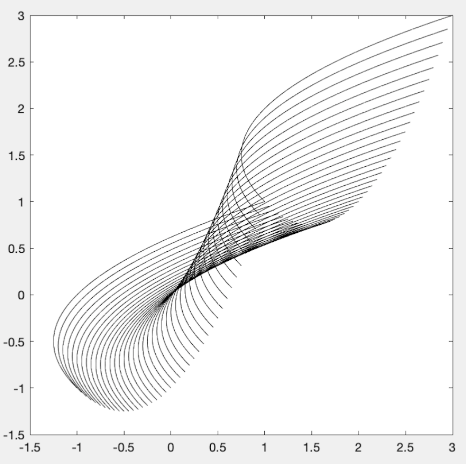
$x = 1.25 p_0 \pm 0$

$y = 2.5 p_1 \pm 0$

Sets from Taylor Models: a nonlinear example

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

$$x = p_0 + p_1 + p_1^2$$
$$y = p_0 + p_1 + p_0^2$$

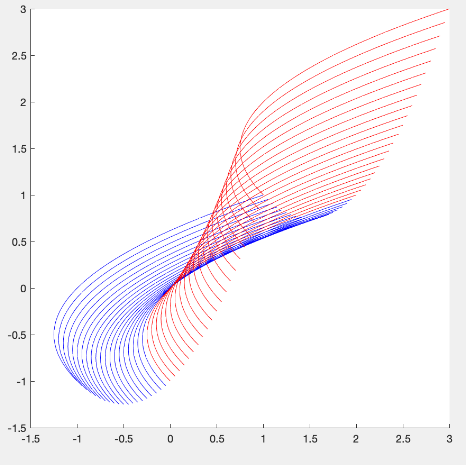


Sets from Taylor Models: a nonlinear example

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

$$x = p_0 + p_1 + p_1^2$$
$$y = p_0 + p_1 + p_0^2$$

By splitting along p_0, i.e. $p_0' = [-1,0]$ and $p_0'' = [0,1]$ we obtain partially overlapping sets.



Accuracy control

Numerical parameters available

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

1. Maximum polynomial order
2. Minimum coefficient value

Accuracy control

Numerical parameters 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 accuracy and domain space complexity

a. Convert e into an additional parameter $\rightarrow$ increase n

b. Sweep all terms where a parameter appears into e $\rightarrow$ reduce n

Representation of sets using a grid

Definition (Grid)

A coordinate-aligned discrete partitioning of a root hyper-rectangle in the variables space, which identifies cells of different sizes.

Definition (Grid set)

A marking of cells locked to a grid.

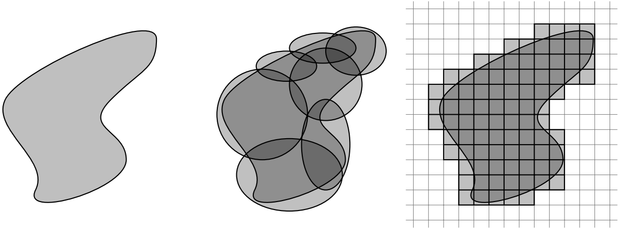
Representation of sets using a grid

Definition (Grid)

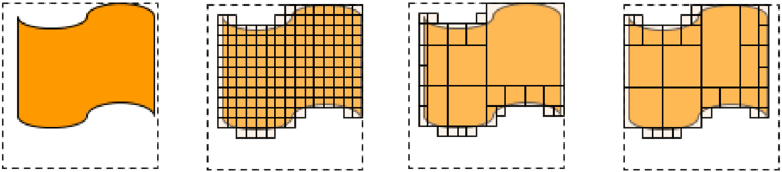
A coordinate-aligned discrete partitioning of a root hyper-rectangle in the variables space, which identifies cells of different sizes.

Definition (Grid set)

A marking of cells locked to a grid.

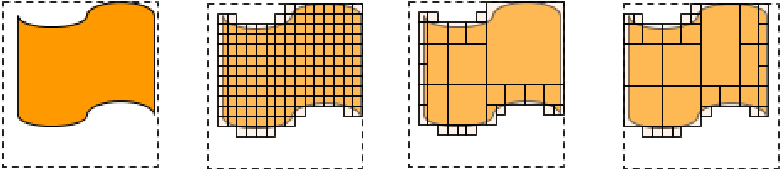


Example of paving a set with a grid set



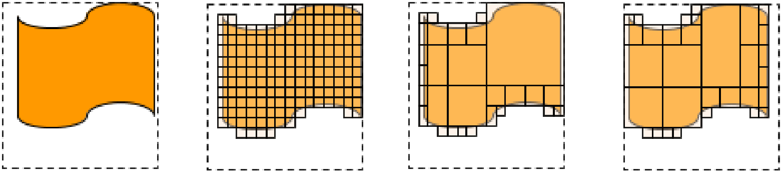
- The grid set in the second figure is useful for splitting a large set into smaller, more numerically amenable, subsets

Example of paving a set with a grid set



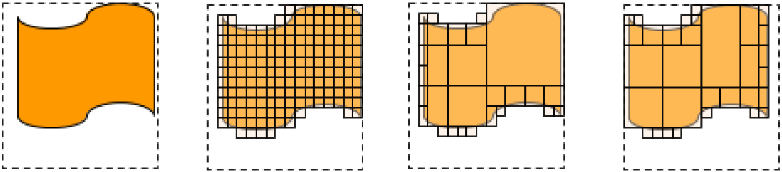
- The grid set in the second figure is useful for splitting a large set into smaller, more numerically amenable, subsets
- The grid set on the third figure is the most efficient when evolution is not considered

Example of paving a set with a grid set



- The grid set in the second figure is useful for splitting a large set into smaller, more numerically amenable, subsets
- The grid set on the third figure is the most efficient when evolution is not considered
- The choice of the root cell (which can be any rectangle centered anywhere) is essential to the efficiency of the grid set approximation

Example of paving a set with a grid set



- The grid set in the second figure is useful for splitting a large set into smaller, more numerically amenable, subsets
- The grid set on the third figure is the most efficient when evolution is not considered
- The choice of the root cell (which can be any rectangle centered anywhere) is essential to the efficiency of the grid set approximation
  - In the third figure we have 28 cells, in the fourth 31
  - However, if we want to combine sets, the root cell must be common to all sets in the reachable set

Grid sets - pros and cons

Pros

- Converts easily from/to a polynomial model;
- Allows a compact internal representation, e.g. markings on a binary tree or a binary decision diagram;
- Cells can be split/joined by changing the depth of the markings;
- Union, intersection, difference and inclusion can be performed very efficiently.

Grid sets - pros and cons

Pros

- Converts easily from/to a polynomial model;
- Allows a compact internal representation, e.g. markings on a binary tree or a binary decision diagram;
- Cells can be split/joined by changing the depth of the markings;
- Union, intersection, difference and inclusion can be performed very efficiently.

Cons

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

Hybrid evolution of a set

Continuous step

1. From the starting set, given a time step h, construct the flow set
2. Apply to the whole $[0, h]$ time interval to get the reached set
3. Apply to the h time value to get the finishing set

Hybrid evolution of a set

Continuous step

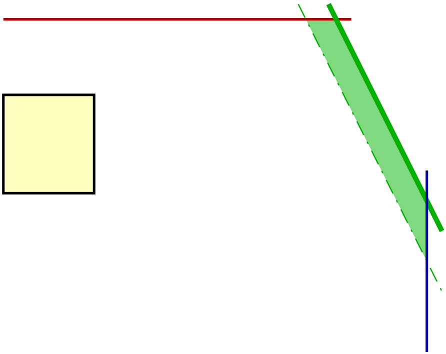
1. From the starting set, given a time step h, construct the flow set
2. Apply to the whole $[0, h]$ time interval to get the reached set
3. Apply to the h time value to get the finishing set

Discrete step

1. From the flow set, identify the presence of intersections with guard sets
2. Compute crossing times with the guards
3. Compute intersections with the guards
4. Apply the resets

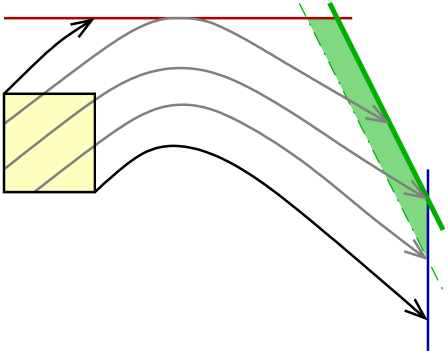
Finite-time evolution

A sequence of continuous and discrete steps



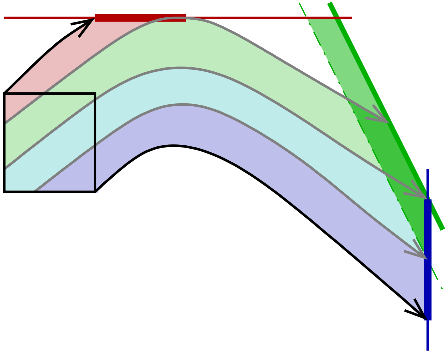
Finite-time evolution

A sequence of continuous and discrete steps



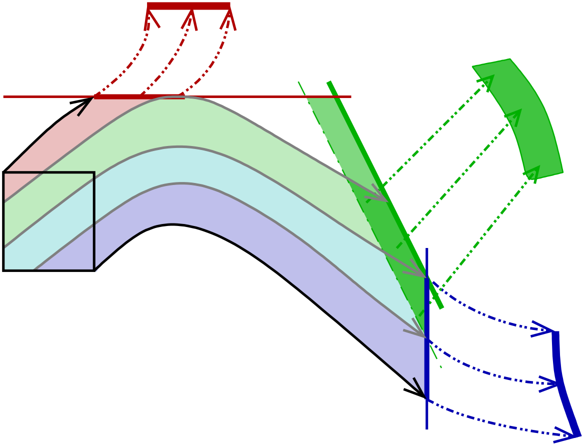
Finite-time evolution

A sequence of continuous and discrete steps



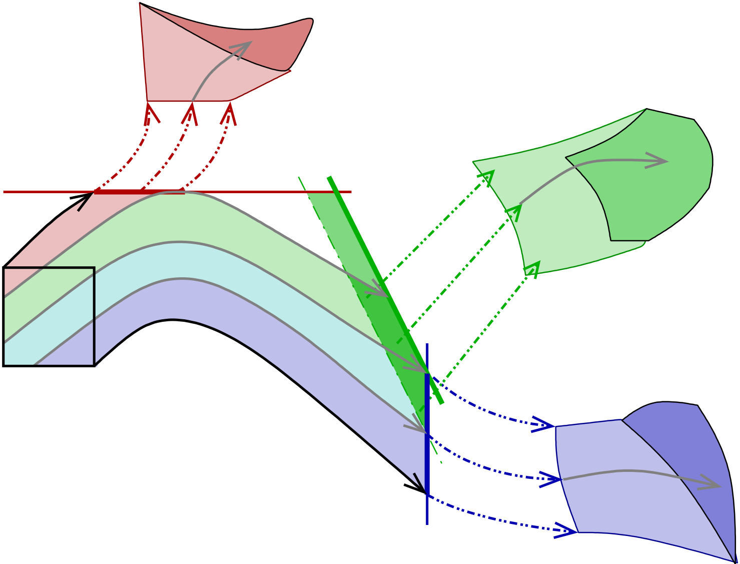
Finite-time evolution

A sequence of continuous and discrete steps



Finite-time evolution

A sequence of continuous and discrete steps



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.

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.

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 would need a set representation with

- operations like subtraction, intersection, splitting and merging;
- small memory usage, fast operations and good scalability;
- small 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 would need a set representation with

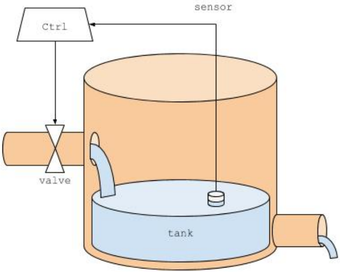
- operations like subtraction, intersection, splitting and merging;
- small memory usage, fast operations and good scalability;
- small overapproximation error.

We use Taylor Sets to respect 2., switching temporarily to Grid Sets for 1.

Infinite-time reachability at a glance

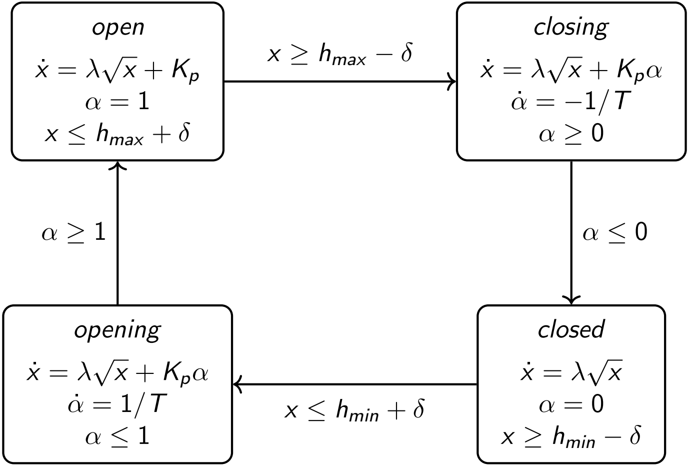
1. Identify a bounding set B to constrain evolution;
2. Approximate non-rigorously the evolution (by points) to identify reasonable lock-to-grid times when the grid set representation should be updated;
3. Compute the finite-time hybrid evolution of the automaton up to the next lock-to-grid time;
4. 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.

The watertank example

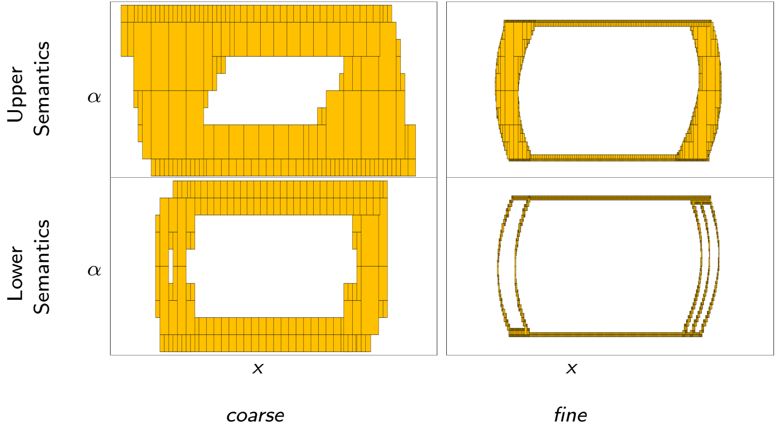


- Outlet flow F_{out} depends on the water level $x(t)$: $F_{out}(t) = \lambda\sqrt{x(t)}$
- Inlet flow F_{in} is controlled by the valve position $\alpha(t)$: $F_{in}(t) = K_p \cdot \alpha(t)$
- The controller senses the water level and sends the appropriate commands to the valve.

The watertank automaton

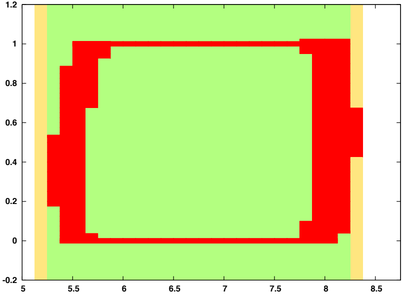


Reachability at different accuracies



Safety verification

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



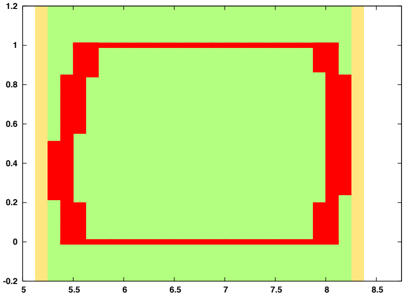
First iteration:
grid $1/8 \times 1/80$
(x-axis: $x(t)$, y-axis: $\alpha(t)$).

Outer reach is not safe, try lower reach.

Green: safe set Orange: ε-tolerance Red: computed set

Safety verification

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



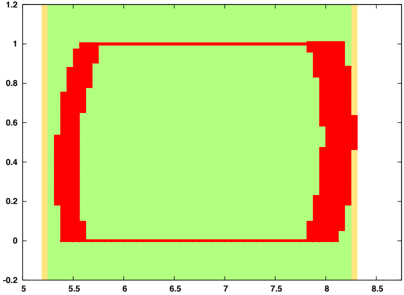
First iteration:
grid $1/8 \times 1/80$
(x-axis: $x(t)$, y-axis: $\alpha(t)$).

Lower reach is not unsafe, refine grid.

Green: safe set Orange: ε-tolerance Red: computed set

Safety verification

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



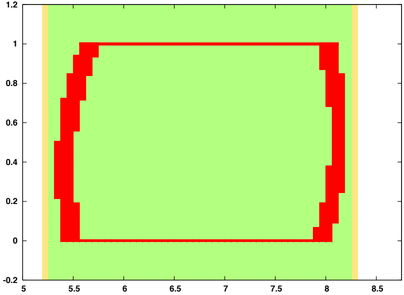
Second iteration: grid $1/16 \times 1/160$ (x-axis: $x(t)$, y-axis: $\alpha(t)$).

Outer reach is not safe, try lower reach.

Green: safe set Orange: ε-tolerance Red: computed set

Safety verification

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



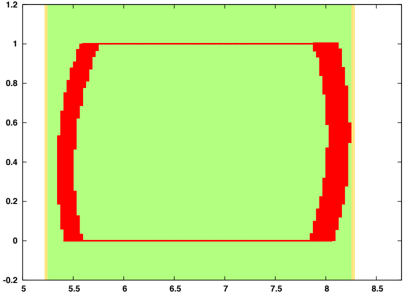
Second iteration: grid $1/16 \times 1/160$ (x-axis: $x(t)$, y-axis: $\alpha(t)$).

Lower reach is not unsafe, refine grid.

Green: safe set Orange: ε-tolerance Red: computed set

Safety verification

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: ε-tolerance   Red: computed set

Open issues on the topic

- Identify proper values of the numerical parameters and sensible refinement strategies;

Open issues on the topic

- Identify proper values of the numerical parameters and sensible refinement strategies;
- Identify improved data structures for geometrical representation;

Open issues on the topic

- Identify proper values of the numerical parameters and sensible refinement strategies;
- Identify improved data structures for geometrical representation;
- Address scalability with respect to the number of variables;

Future directions of development

1. Complete the Python interface to cover the dynamics module;

Future directions of development

1. Complete the Python interface to cover the dynamics module;
2. Introduce parallelism in the handling of multiple trajectories;

Future directions of development

1. Complete the Python interface to cover the dynamics module;
2. Introduce parallelism in the handling of multiple trajectories;
3. Introduce exploration of the numerical parameters space using operations research approaches;

Future directions of development

1. Complete the Python interface to cover the dynamics module;
2. Introduce parallelism in the handling of multiple trajectories;
3. Introduce exploration of the numerical parameters space using operations research approaches;
4. Implement grid sets by developing a parallel BDD library;

Future directions of development

1. Complete the Python interface to cover the dynamics module;
2. Introduce parallelism in the handling of multiple trajectories;
3. Introduce exploration of the numerical parameters space using operations research approaches;
4. Implement grid sets by developing a parallel BDD library;
5. Interface to known tools and frameworks such as MATLAB and ROS;

Future directions of development

1. Complete the Python interface to cover the dynamics module;
2. Introduce parallelism in the handling of multiple trajectories;
3. Introduce exploration of the numerical parameters space using operations research approaches;
4. Implement grid sets by developing a parallel BDD library;
5. Interface to known tools and frameworks such as MATLAB and ROS;
6. Develop verification procedures in the context of robotic surgery and smart manufacturing.

References

- Collins, P.; Bresolin, D.; Geretti, L.; Villa, T. "Computing the evolution of hybrid systems using rigorous function calculus", 4th IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'12), June 2012, pg. 284-290, DOI: 10.3182/20120606-3-NL-3011.00046
- Benvenuti, L; Bresolin, D.; Collins, P.; Ferrari, A.; Geretti, L.; Villa, T. "Assume-guarantee verification of nonlinear hybrid systems with Ariadne", International Journal of Robust and Nonlinear Control, Volume 24, Issue 4, Mar. 2014, pg. 699-724, ISSN: 1049-8923, DOI: 10.1002/RNC.2914