Introduction to Hybrid Systems Verification

Davide Bresolin, Luca Geretti, Tiziano Villa

University of Verona, Italy

Summer School on Formal Methods for Cyber-Physical Systems
Second Edition
3-7 June 2019

Outline

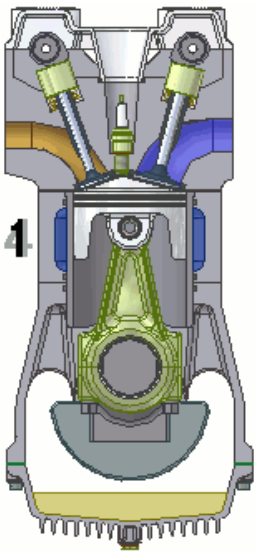
Hybrid systems

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 because they mix discrete and continuous behaviours

Example: 4-strokes engine

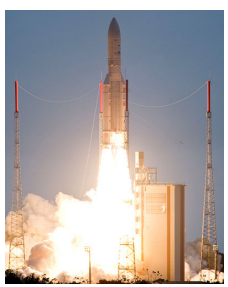


- Intake stroke: air and vaporized fuel are drawn in
- Compression stroke: fuel vapor and air are compressed and ignited
- Combustion stroke: fuel combusts and piston is pushed downwards
- Exhaust/Emission stroke: exhaust is driven out
- During 1st, 2nd and 4th stroke the piston is relying on the power and momentum generated by the pistons of the other cylinders

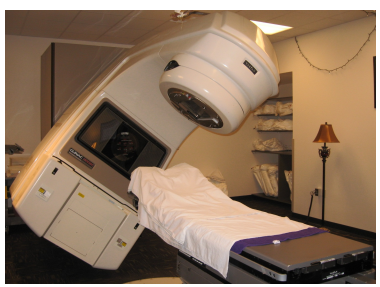
During the 4 strokes pression, temperature, . . . , vary continuously

The cost of errors and failures

- Hybrid Systems are safety-critical systems:
  - airplane control systems, medical care systems, train signalling systems, automotive systems, ...
- Bugs, design errors and failures can cause catastrophic loss of money, time or even human life:



- Ariane 5 explosion (1996, approx $500 million)



- Therac 25 accident (1985-87, six patients seriously injured or killed)

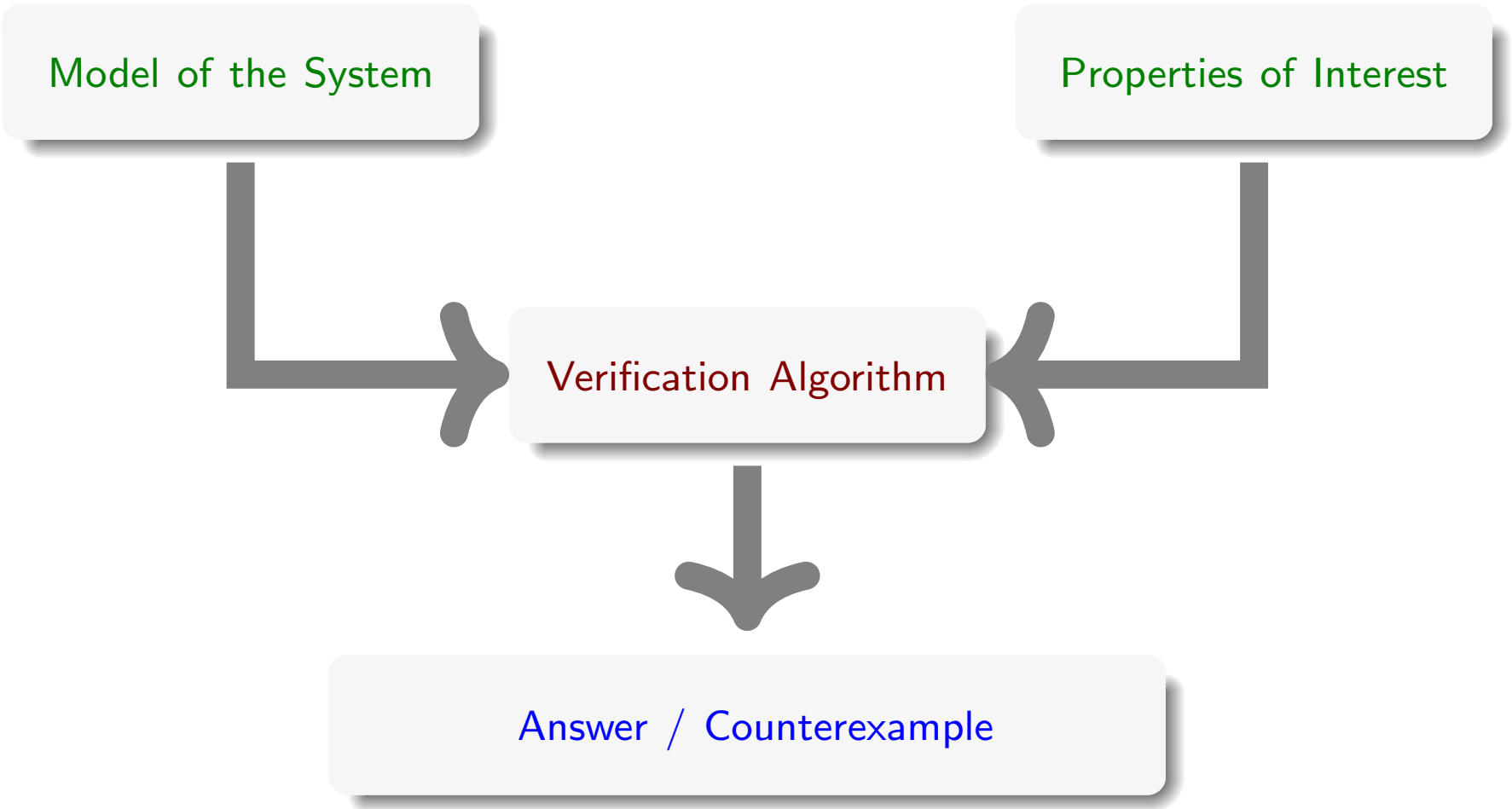
Testing and simulation

- The usual methods for the validation of complex engineering systems are:
  - Testing (using the system itself)
  - Simulation (using a model of the system)
- They can cover only a subset of possible behaviors of the system
- They can only prove the existence of errors, not their in-existence
- CPS are reactive systems: they maintain an ongoing interaction with the environment
  - errors and failures caused by the interaction with the environment can be very difficult to discover!

A different way to design correct systems

- Formal Methods (FM):
  - mathematical languages, techniques and tools ...
  - ... used to specify and verify systems ...
  - ... to help engineers construct more reliable systems
- A way to examine the entire state space of a design
  - Establish a correctness or safety property that is true for all possible inputs
- FM are standard practice of many hardware and software producers:
  - microprocessor design (Intel), critical software design (NASA), operating systems (Microsoft), ...
- Recently introduced also within the robotics and control communities

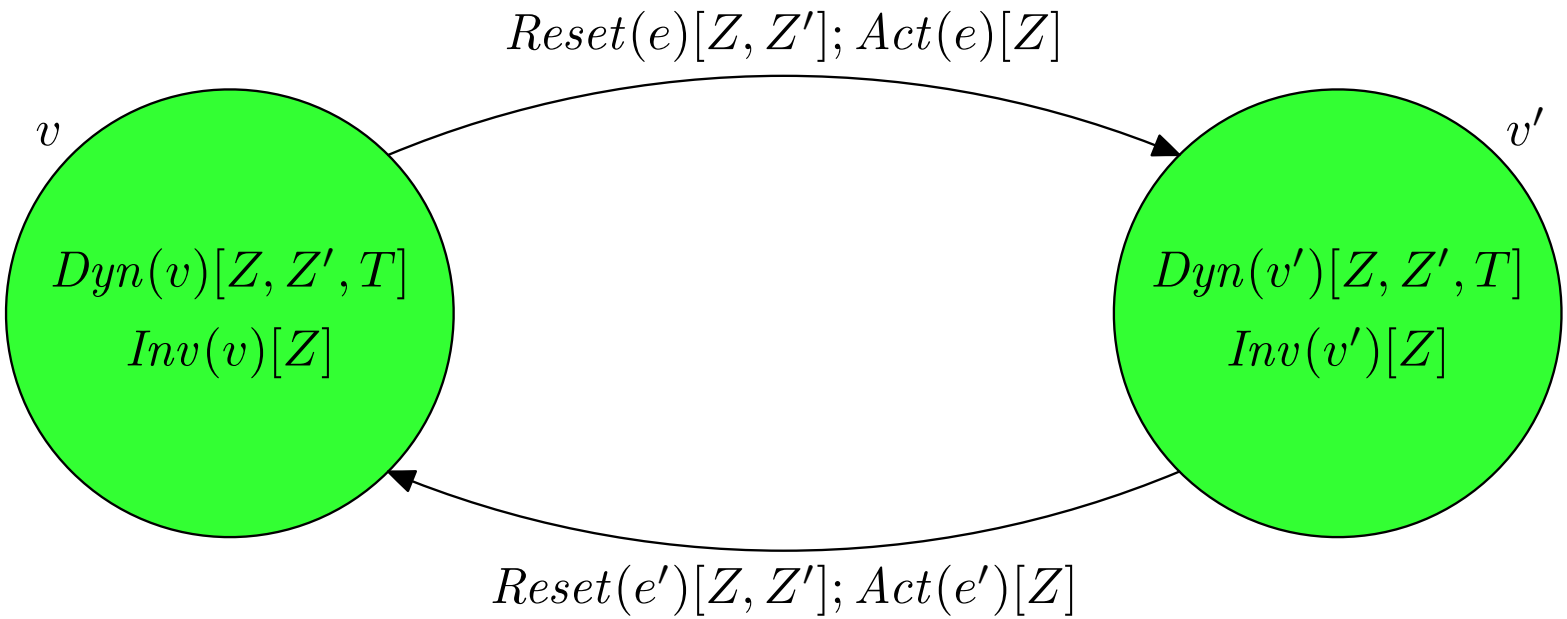
The formal verification flow



Modeling the system: Hybrid Automata

A hybrid automaton H is

a finite-state automaton with continuous variables Z



A state is a couple $\langle v, r \rangle$ where r is a valuation for Z

Syntax

Definition (Hybrid Automata (Piazza et al.))

A k-hybrid automaton $H = \langle Z, Z', \mathcal{V}, \mathcal{E}, Inv, Dyn, Act, Reset \rangle$ consists of the following components:

Syntax

Definition (Hybrid Automata (Piazza et al.))

A k-hybrid automaton $H = \langle Z, Z', \mathcal{V}, \mathcal{E}, Inv, Dyn, Act, Reset \rangle$ consists of the following components:

1. $Z = (Z_1, \ldots, Z_k)$ and $Z' = (Z'_1, \ldots, Z'_k)$ are two vectors of variables ranging over the reals;

Syntax

Definition (Hybrid Automata (Piazza et al.))

A k-hybrid automaton $H = \langle Z, Z', \mathcal{V}, \mathcal{E}, Inv, Dyn, Act, Reset \rangle$ consists of the following components:

1. $Z = (Z_1, \ldots, Z_k)$ and $Z' = (Z'_1, \ldots, Z'_k)$ are two vectors of variables ranging over the reals;
2. $\langle \mathcal{V}, \mathcal{E} \rangle$ is a finite directed graph;

Syntax

Definition (Hybrid Automata (Piazza et al.))

A k-hybrid automaton $H = \langle Z, Z', \mathcal{V}, \mathcal{E}, Inv, Dyn, Act, Reset \rangle$ consists of the following components:

1. $Z = (Z_1, \ldots, Z_k)$ and $Z' = (Z'_1, \ldots, Z'_k)$ are two vectors of variables ranging over the reals;
2. $\langle \mathcal{V}, \mathcal{E} \rangle$ is a finite directed graph;
3. Each $v \in \mathcal{V}$ is labeled by the two formulæ $Inv(v)[Z]$ and $Dyn(v)[Z, Z', T]$ such that if $Inv(v)[p]$ holds then $Dyn(v)[p, p, 0]$ holds as well;

Syntax

Definition (Hybrid Automata (Piazza et al.))

A k-hybrid automaton $H = \langle Z, Z', \mathcal{V}, \mathcal{E}, Inv, Dyn, Act, Reset \rangle$ consists of the following components:

1. $Z = (Z_1, \ldots, Z_k)$ and $Z' = (Z'_1, \ldots, Z'_k)$ are two vectors of variables ranging over the reals;
2. $\langle \mathcal{V}, \mathcal{E} \rangle$ is a finite directed graph;
3. Each $v \in \mathcal{V}$ is labeled by the two formulæ $Inv(v)[Z]$ and $Dyn(v)[Z, Z', T]$ such that if $Inv(v)[p]$ holds then $Dyn(v)[p, p, 0]$ holds as well;
4. Each $e \in \mathcal{E}$ is labeled by the formulæ $Act(e)[Z]$ and $Reset(e)[Z, Z']$.

Comments on the definition

- *Inv*, *Dyn*, *Act*, *Reset* are sets of formulae in a first-order language $\mathcal{L}$
  E.g., $\mathcal{L} = (+, *, <, 0, 1)$

Comments on the definition

- *Inv*, *Dyn*, *Act*, *Reset* are sets of formulae in a first-order language $\mathcal{L}$
  E.g., $\mathcal{L} = (+, *, <, 0, 1)$
- the formulae are evaluated over a model $\mathcal{M}$ of $\mathcal{L}$ in the domain $\mathbb{R}$
  E.g., $\mathcal{M} = (\mathbb{R}, +, *, <, 0, 1)$

Comments on the definition

- *Inv*, *Dyn*, *Act*, *Reset* are sets of formulae in a first-order language $\mathcal{L}$
  E.g., $\mathcal{L} = (+, *, <, 0, 1)$
- the formulae are evaluated over a model $\mathcal{M}$ of $\mathcal{L}$ in the domain $\mathbb{R}$
  E.g., $\mathcal{M} = (\mathbb{R}, +, *, <, 0, 1)$
- the nodes $\mathcal{V}$ are called locations (or *control modes*), the arcs $\mathcal{E}$ are called control switches

Comments on the definition

- *Inv*, *Dyn*, *Act*, *Reset* are sets of formulae in a first-order language $\mathcal{L}$
  E.g., $\mathcal{L} = (+, *, <, 0, 1)$
- the formulae are evaluated over a model $\mathcal{M}$ of $\mathcal{L}$ in the domain $\mathbb{R}$
  E.g., $\mathcal{M} = (\mathbb{R}, +, *, <, 0, 1)$
- the nodes $\mathcal{V}$ are called locations (or *control modes*), the arcs $\mathcal{E}$ are called control switches
- the variable T represents time

Comments on the definition

- *Inv*, *Dyn*, *Act*, *Reset* are sets of formulae in a first-order language $\mathcal{L}$
  E.g., $\mathcal{L} = (+, *, <, 0, 1)$
- the formulae are evaluated over a model $\mathcal{M}$ of $\mathcal{L}$ in the domain $\mathbb{R}$
  E.g., $\mathcal{M} = (\mathbb{R}, +, *, <, 0, 1)$
- the nodes $\mathcal{V}$ are called locations (or *control modes*), the arcs $\mathcal{E}$ are called control switches
- the variable T represents time
- $p \in \mathbb{R}^k$

Example: thermostat

Example (Thermostat)

Let us consider a room heated by a radiator controlled by a thermostat

- When the thermostat is on the temperature increases exponentially in time
- When the thermostat is off the temperature decreases exponentially in time
- The thermostat switches on the radiator when the temperature decreases below $19C$
- The thermostat switches off the radiator when the temperature increases above $21C$

Example: thermostat

Let us model the behaviour of the temperature in time by a hybrid automaton H with:

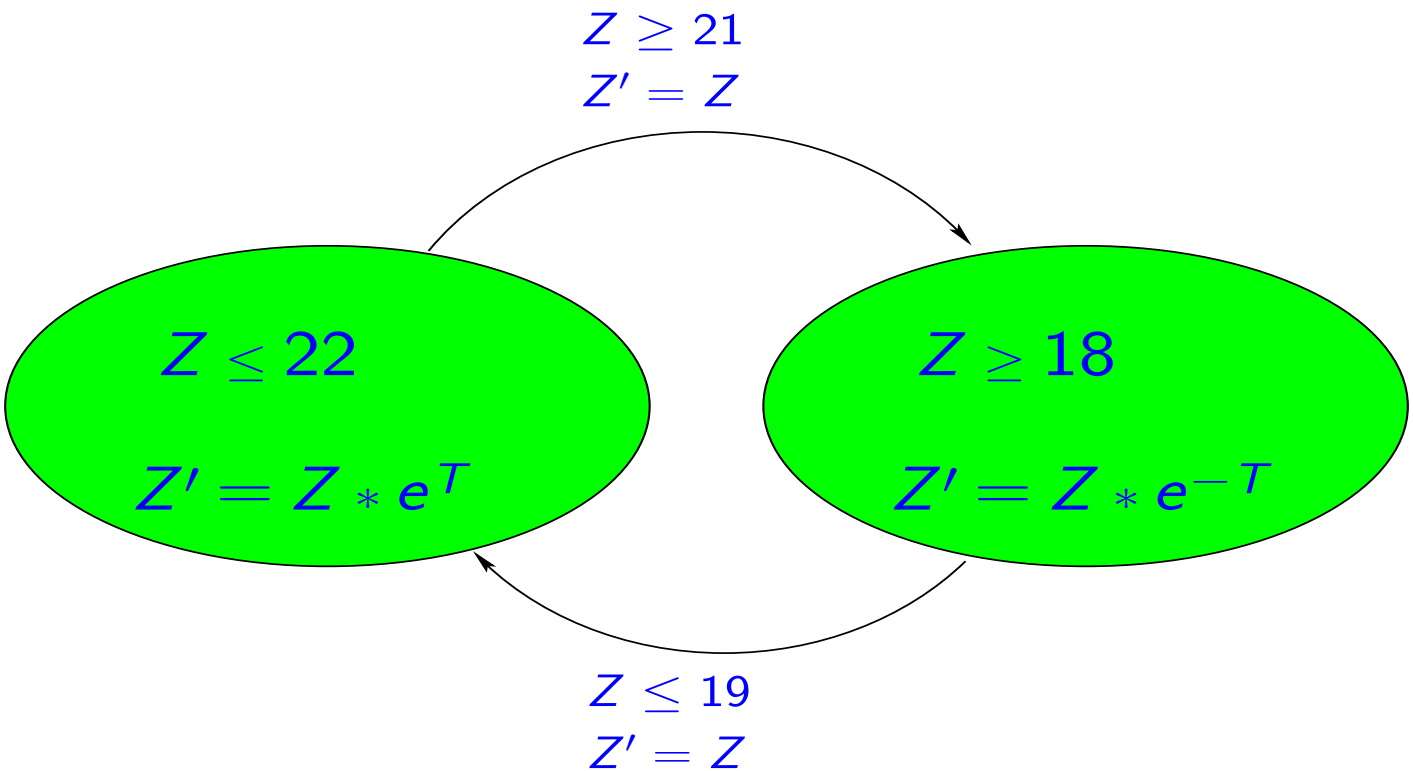
- 2 locations ON and OFF
- 2 arcs that join the two locations
- 1 continuous variable Z that represents the temperature

Example: thermostat

$H = \langle Z,\ Z',\ \mathcal{V},\ \mathcal{E},\ Inv,\ Dyn,\ Act,\ Reset \rangle$ such that:

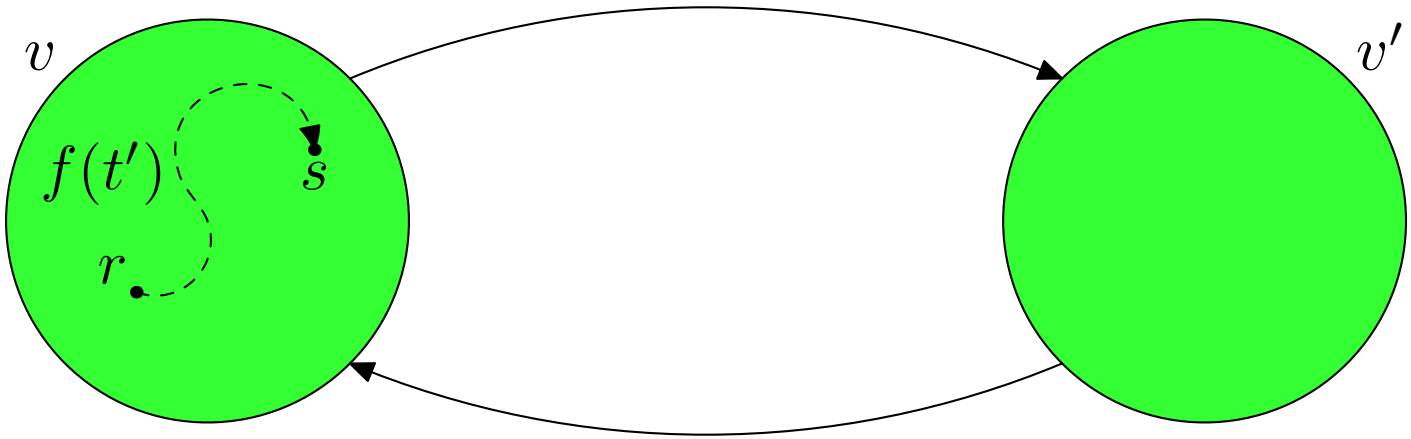
- Z e Z' are two variables
- $\mathcal{V} = \{ON, OFF\}$ and $\mathcal{E} = \{(ON, OFF), (OFF, ON)\}$
- $Inv(ON)[Z] := Z \leq 22$ and
  $Dyn(ON)[Z, Z', T] := Z' = Z * e^T$
- $Inv(OFF)[Z] := Z \geq 18$ and
  $Dyn(OFF)[Z, Z', T] := Z' = Z / e^T$
- $Act((ON, OFF))[Z] := Z \geq 21$ and
  $Reset((ON, OFF))[Z, Z'] := Z' = Z$
- $Act((OFF, ON))[Z] := Z \leq 19$ and
  $Reset((OFF, ON))[Z, Z'] := Z' = Z$

Example: thermostat



Semantics

$\ell = \langle v, r \rangle$ is *admissible* if $Inv(v)[r]$ holds

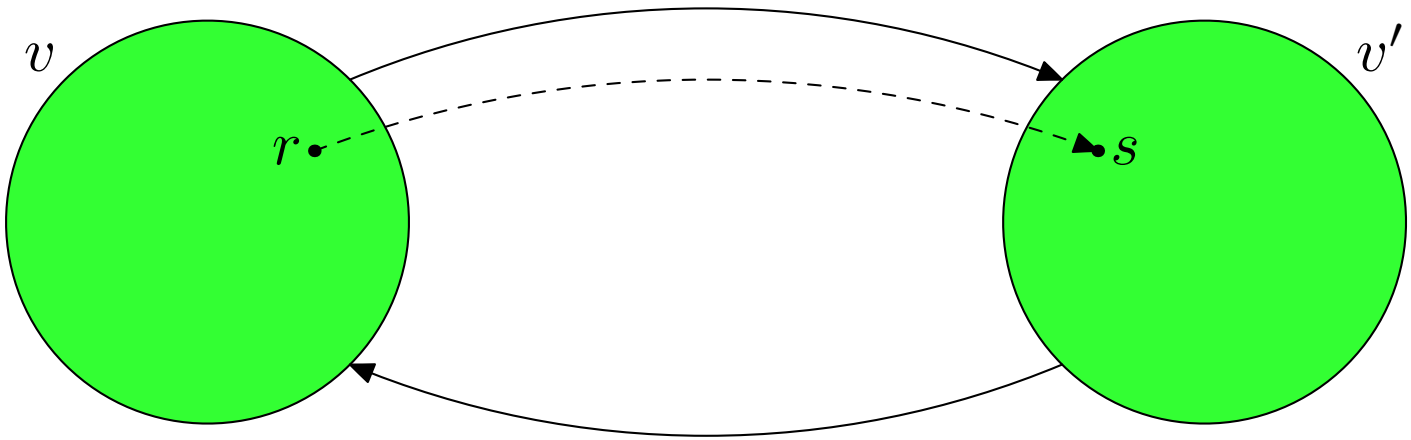


Definition (Continuous transitions)

$\langle v, r \rangle \xrightarrow{t}_C \langle v, s \rangle \iff$ There exists a continuous function $f : \mathbb{R}^+ \mapsto \mathbb{R}^k$ such that $r = f(0)$, $s = f(t)$ and for each $t' \in [0, t]$ the formulæ $Inv(v)[f(t')]$ and $Dyn(v)[r, f(t'), t']$ hold

Semantics

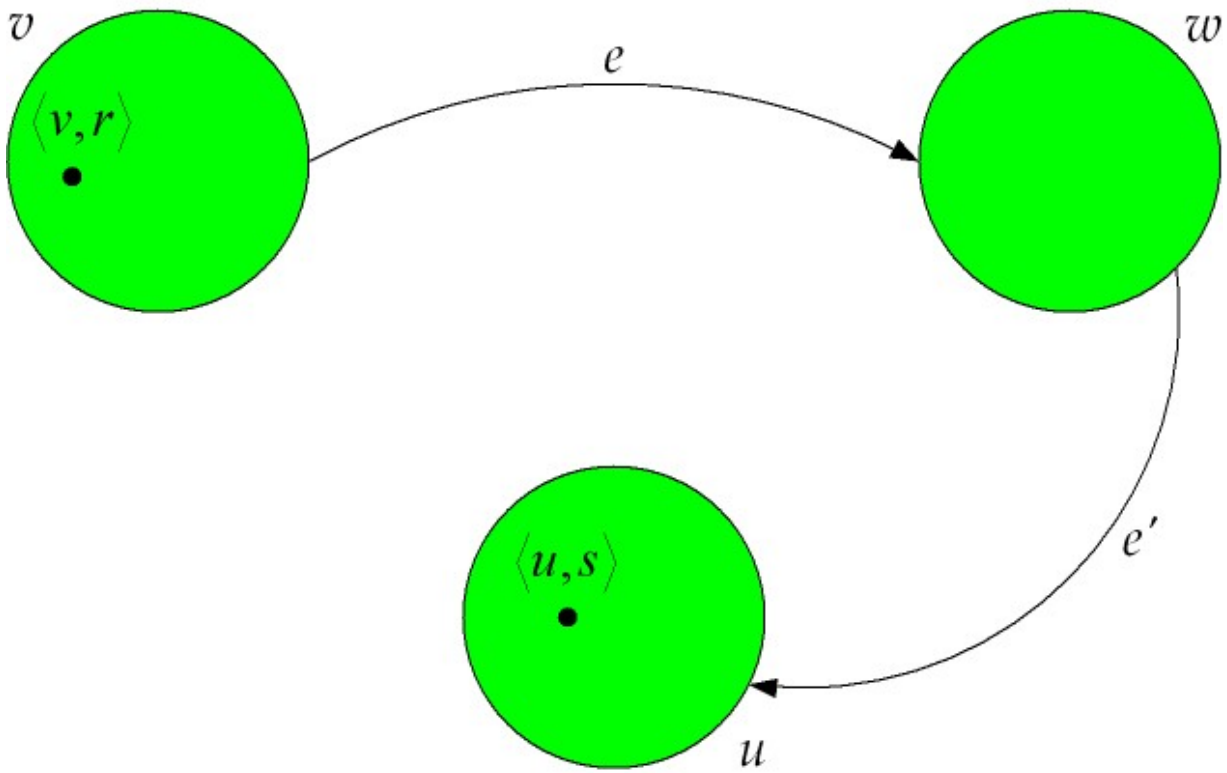
$\ell = \langle v, r \rangle$ is *admissible* if $Inv(v)[r]$ holds



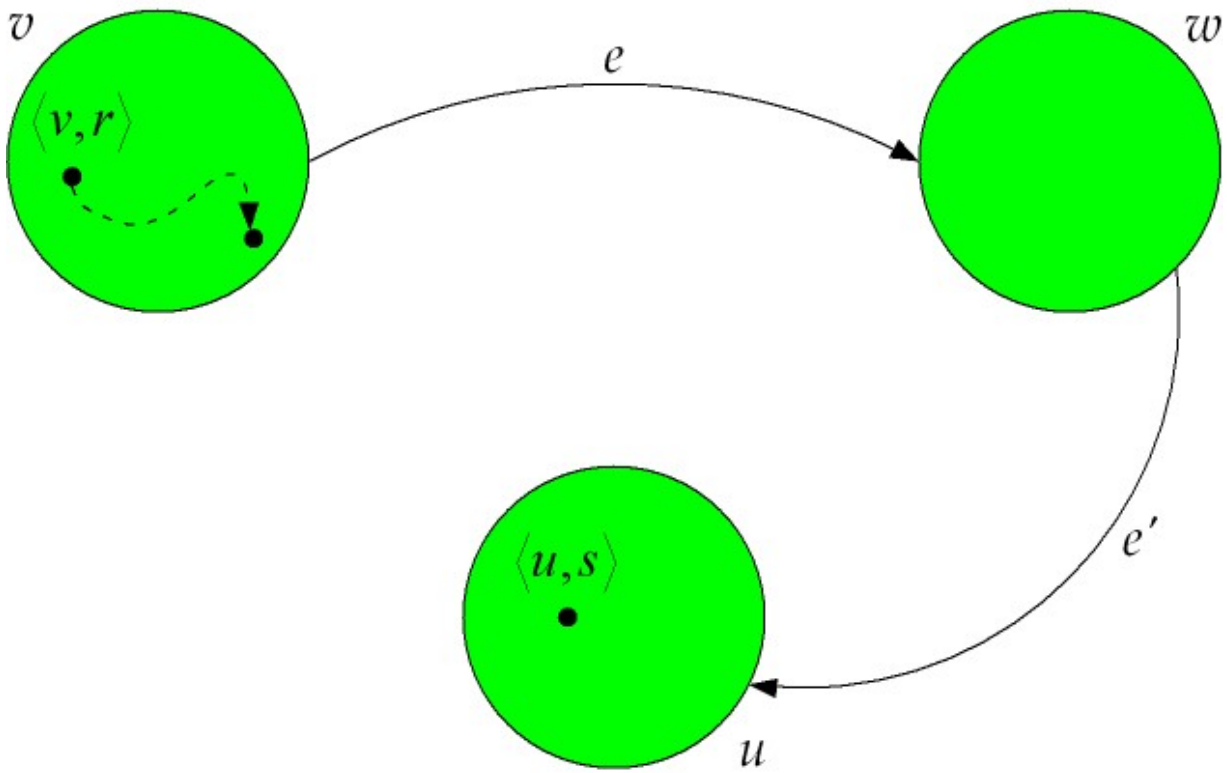
Definition (Discrete transitions)

$$\langle v, r \rangle \xrightarrow{\langle v, v' \rangle}_D \langle v', s \rangle \iff \begin{array}{l} \langle v, v' \rangle \in \mathcal{E}, \quad Inv(v)[r], \\ Act(\langle v, v' \rangle)[r], \\ Reset(\langle v, v' \rangle)[r, s] \quad \text{and} \\ Inv(v')[s] \text{ hold} \end{array}$$

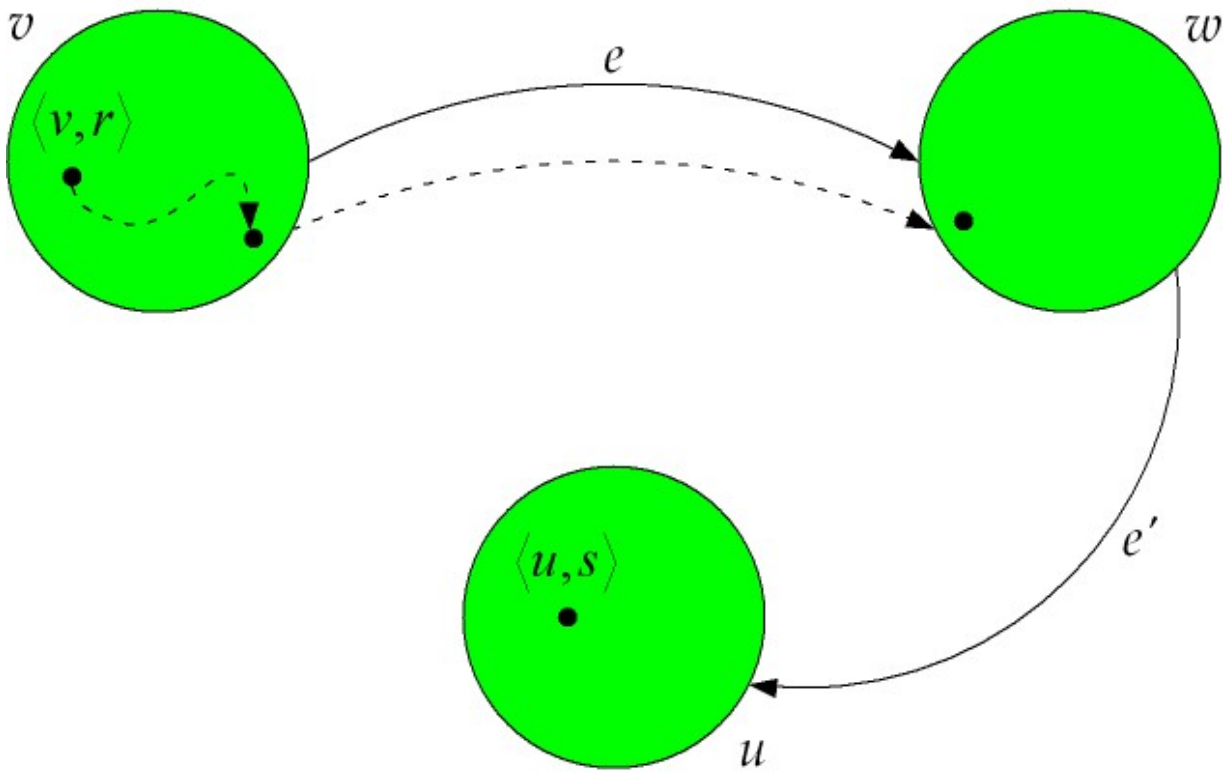
Reachability



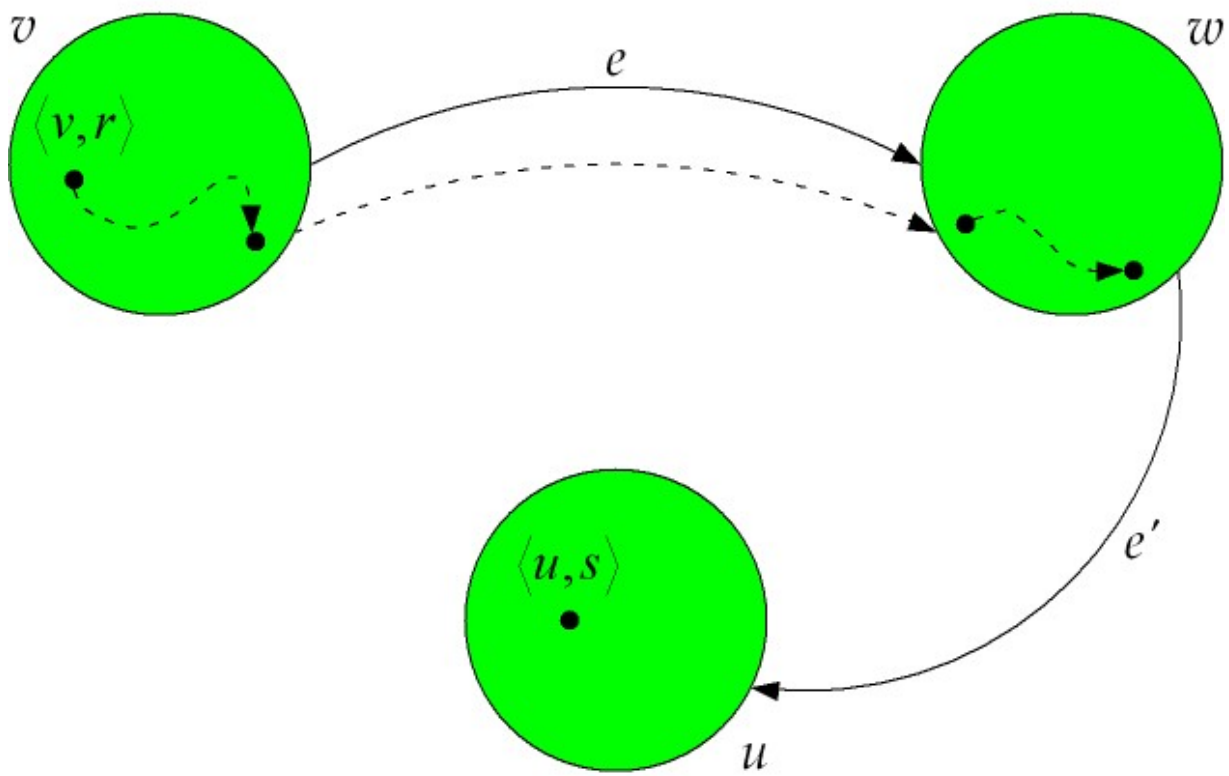
Reachability



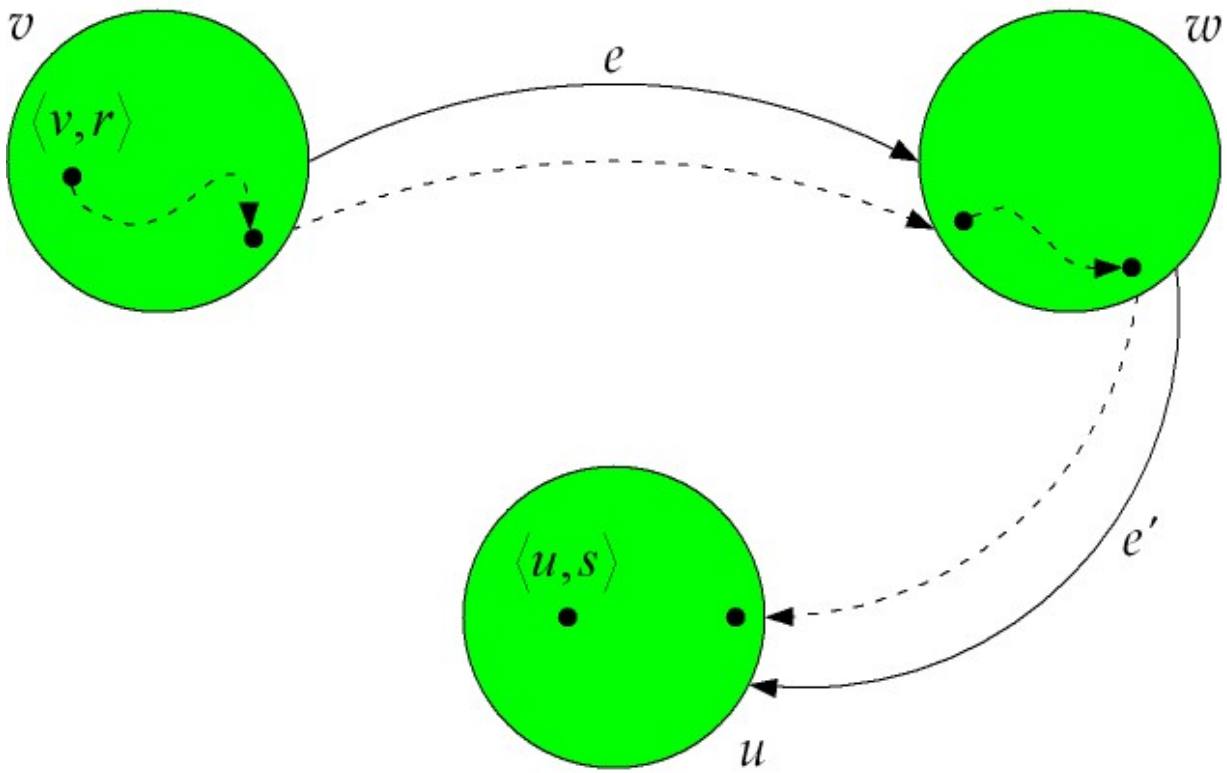
Reachability



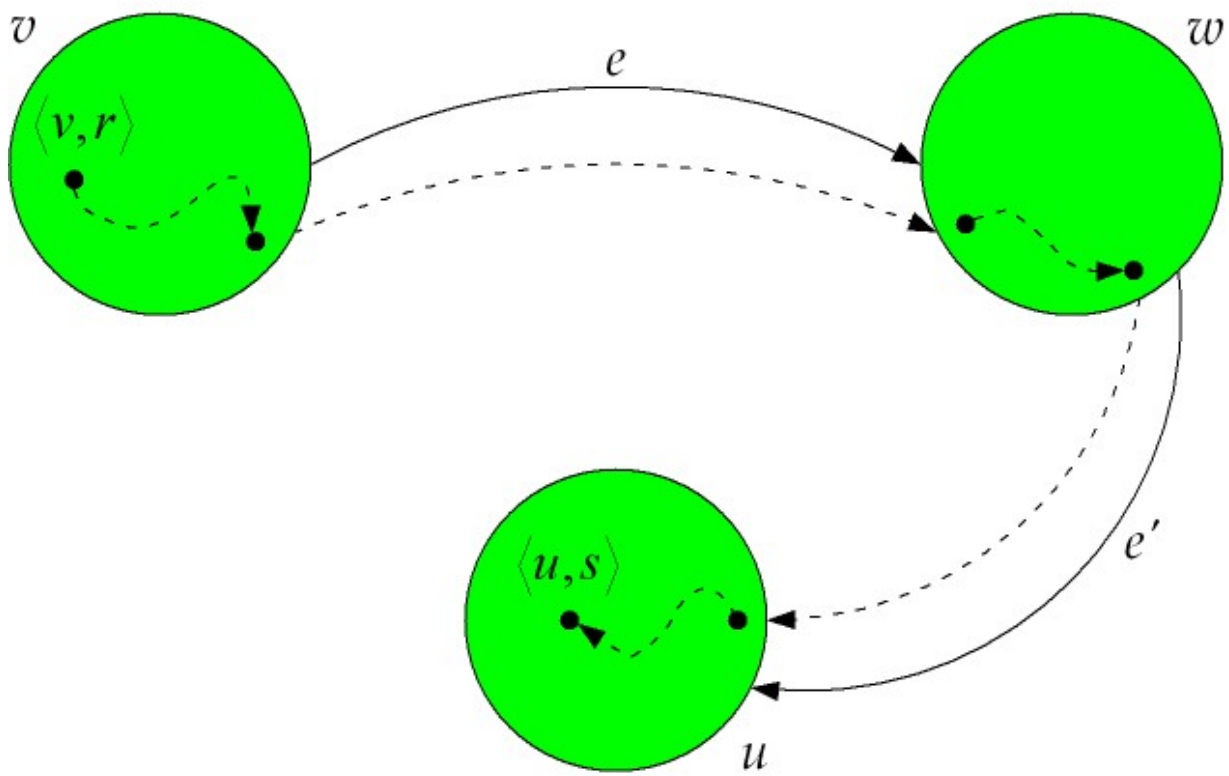
Reachability



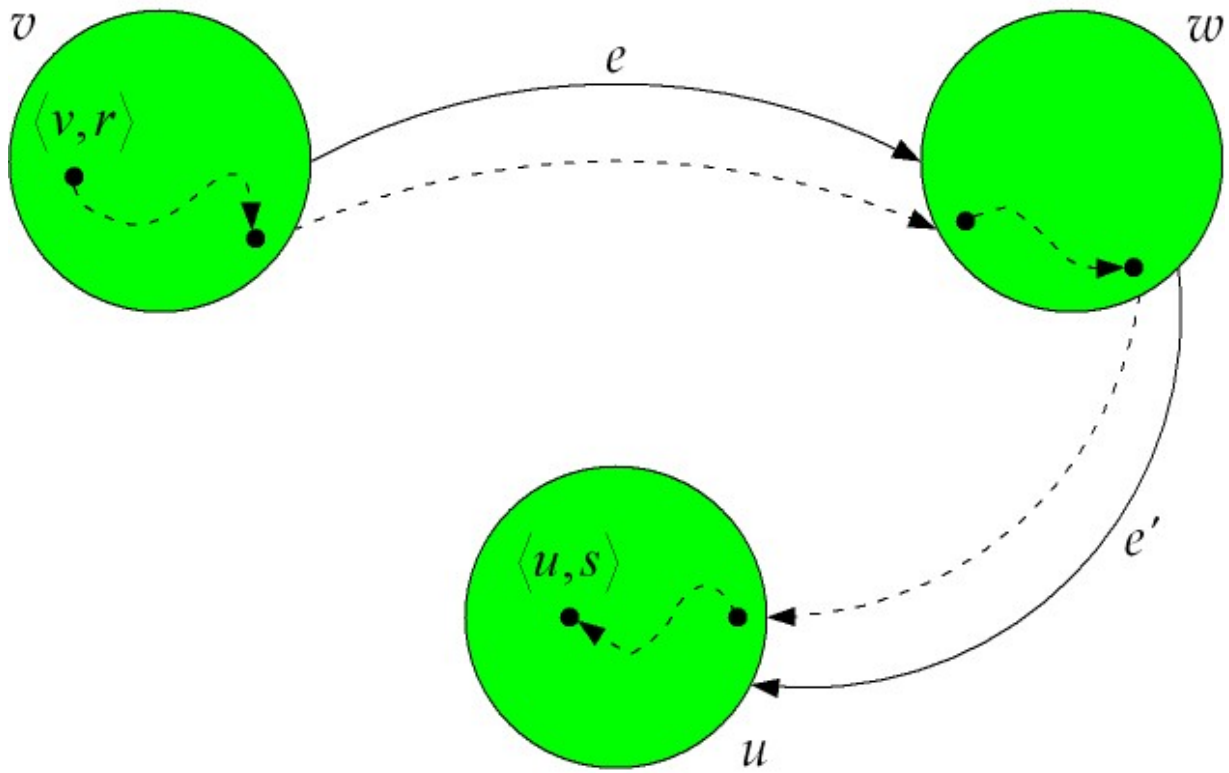
Reachability



Reachability



Reachability



Let $I, F \in \mathbb{R}^k$. Can we reach $\langle u, F \rangle$ from $\langle v, I \rangle$?

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

- Different locations may partially share the invariants

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

- Different locations may partially share the invariants
- Different continuous trajectories may leave from the same admissible state

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

- Different locations may partially share the invariants
- Different continuous trajectories may leave from the same admissible state
- There may be arcs that go to different locations but partially share the activation functions

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

- Different locations may partially share the invariants
- Different continuous trajectories may leave from the same admissible state
- There may be arcs that go to different locations but partially share the activation functions
- The activation functions are not necessarily on the frontiers of the invariants

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

- Different locations may partially share the invariants
- Different continuous trajectories may leave from the same admissible state
- There may be arcs that go to different locations but partially share the activation functions
- The activation functions are not necessarily on the frontiers of the invariants
- The reset functions are not necessarily deterministic

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

- Different locations may partially share the invariants
- Different continuous trajectories may leave from the same admissible state
- There may be arcs that go to different locations but partially share the activation functions
- The activation functions are not necessarily on the frontiers of the invariants
- The reset functions are not necessarily deterministic

Probabilistic generalisations add other levels of complexity:

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

- Different locations may partially share the invariants
- Different continuous trajectories may leave from the same admissible state
- There may be arcs that go to different locations but partially share the activation functions
- The activation functions are not necessarily on the frontiers of the invariants
- The reset functions are not necessarily deterministic

Probabilistic generalisations add other levels of complexity:

- Discrete transitions form a Markov process

Many sources of non-determinism

Hybrid automata may be non-deterministic since:

- Different locations may partially share the invariants
- Different continuous trajectories may leave from the same admissible state
- There may be arcs that go to different locations but partially share the activation functions
- The activation functions are not necessarily on the frontiers of the invariants
- The reset functions are not necessarily deterministic

Probabilistic generalisations add other levels of complexity:

- Discrete transitions form a Markov process
- Stochastic differential equations add probability to the continuous dynamics

Trace and reachability

A trace of H is a sequence of admissible states $[\ell_0, \ell_1, \ldots, \ell_i, \ldots, \ell_n]$ such that $\ell_{i-1} \to \ell_i$ holds $\forall i \in [1, n]$.

Definition (Reachability)

The automaton H reaches $\langle u, s \rangle$, $s \in \mathbb{R}^k$, from $\langle v, r \rangle$, $r \in \mathbb{R}^k$, if there exists a trace $tr = [\ell_0, \ldots, \ell_n]$ of H such that $\ell_0 = \langle v, r \rangle$ and $\ell_n = \langle u, s \rangle$.

Definition (Reachability problem)

Given an automaton H, a set of starting points $\langle v, I \rangle$, $I \subseteq \mathbb{R}^k$, and a set of ending points $\langle u, F \rangle$, $F \subseteq \mathbb{R}^k$, decide whether there exists a point in $\langle v, I \rangle$ from which a point in $\langle u, F \rangle$ is reachable.

Outline

Reachability and Verification

Reachability can be used to verify safety properties

φ is always true in H

if and only if

all states reachable from the initial states of H

are included in the safe set $Sat(\varphi)$.

Reachability and Verification

Reachability can be used to verify safety properties

φ is always true in H

if and only if

all states reachable from the initial states of H

are included in the safe set $Sat(\varphi)$.

Question

Is the reachability problem for Hybrid Automata decidable?

Reachability and Verification

Reachability can be used to verify safety properties

φ is always true in H
if and only if
all states reachable from the initial states of H
are included in the safe set $Sat(\varphi)$.

Question

Is the reachability problem for Hybrid Automata decidable?

Answer

No (Alur et al. 1995).

Many different approaches

Different approaches have been developed to solve the verification problem, at least in some cases:

1. Restrict to special classes of Hybrid Automata that admit exact verification
2. Compute approximations of the reachable set
3. Compute discrete abstractions of the system
4. Simulate the system to find counterexample
5. Use automated theorem proving techinques
6. Use quantitative finite abstractions and probabilistic model checking to verify stochastic hybrid systems

Exact verification of hybrid systems

- For systems with simple dynamics, the evolution can be computed exactly:
  - timed automata ($\dot{x} = 1, x \approx c$),
  - initialized rectangular automata ($\dot{x} \in [c_{min}, c_{max}], x \approx [c_{min}, c_{max}]$ needed when $\dot{x}$ changes).
  - restricted linear automata ($\dot{x} = c$, linear guards) for bounded reachability.
- Verification techniques for finite-state systems can be used
- The state space of the system can be computed

PROS: definite YES/NO answer, high performances

CONS: can verify only systems with simple dynamics

Uppaal

`http://www.uppaal.org/`

- developed by the Uppsala University, Sweden and the Aalborg University in Denmark
- written in Java and C++, graphical user interface
- limited to timed automata
- supports the composition of timed automata
- can verify complex properties in a subset of CTL temporal logic
- results are formally sound
- scalability: up to 100 continuous variables

HyTech

`http://embedded.eecs.berkeley.edu/research/hytech/`

- developed at Cornell and Berkeley by Tom Henzinger, Pei-Hsin Ho, and Howard Wong-Toi
- written in C++, textual scripting interface
- one of the first tools for the verification of hybrid systems
- limited to rectangular automata
- supports the composition of automata
- results are formally sound
- computation is not guaranteed to terminate
- scalability: up to 10 continuous variables

Approximation of the reachable set

- For systems with complex dynamics we cannot compute the exact evolution
- Approximations can give positive and negative answers:
  - overapproximations $\Rightarrow$ positive answers
  - underapproximations $\Rightarrow$ negative answers

PROS: approximation can be very tight, can simulate the system

CONS: requires complex numerical analysis techniques

PHAVer

`http://www-verimag.imag.fr/~frehse/phaver_web/`

- developed at Verimag by Goran Frehse
- written in C++, textual scripting interface
- one of the first tools that enabled verification of hybrid automata
- affine dynamics and guards ($\dot{x} = Ax + b$)
- supports the composition of hybrid automata
- computes over-approximations of the reachable set
- state space is represented using polytopes
- results are formally sound (rational arithmetic with unlimited precision)
- scalability: up to 10 continuous variables

SpaceEx

`http://spaceex.imag.fr/`

- developed at Verimag by a team led by Oded Maler and Goran Frehse
- written in C++, graphical user interface
- affine dynamics and guards ($\dot{x} = Ax + b$)
- supports the composition of hybrid automata
- computes over-approximations of the reachable set
- state space is represented using support functions and polytopes
- results are not guaranteed to be numerically sound (IEEE floating point arithmetic)
- one of the most scalable tools: up to a hundred continuous variables

COHO

`https://www.cs.ubc.ca/~chaoyan/cra.html`

- written by Yan Chao, Mark Greenstreet and Ian Mitchell at the University of British Columbia (Canada)
- CRA (COHO-Reach, COHO Reachability Analysis tool) developed for reachability analysis of analog mixed signal circuits (AMS), and for hybrid systems
- it models reachability analysis as solving differential inclusions $\dot{x} \in F(x), X_0 \subseteq \Omega$
- it overapproximates $F(x)$ by linear differential inclusions and Ω by projectagons
- projectagons represent high-dimensional polyhedra by their projections onto two-dimensional planes, where these projection polygons are not required to be convex

Flow*

`http://systems.cs.colorado.edu/research/cyberphysical/taylormodels`

- developed at RWTH Aachen University (Germany) and at UC Boulder (USA) by Xin Chen, Erika Ábrahám and Sriram Sankaranarayanan
- written in C++
- non-linear ODEs (polynomial dynamics inside modes, polyhedral guards on discrete transitions)
- computes over-approximations of the reachable set
- state space is represented by Taylor models
- results are guaranteed to be numerically sound
- scalability: up to a dozen variables

HyPro/HyDRA

`https://ths.rwth-aachen.de/research/projects/hypro/`

- developed at RWTH Aachen University, Germany, by a team led by Erika Ábrahám
- C++ library for state set representations
- linear dynamics and guards
- different representations supported, with conversion between different representations
- templated number type
- fast implementation of specialized reachability analysis methods
- results are numerically sound only for some representations

SAPO

`https://github.com/dreossi/sapo`

- authors: Tommaso Dreossi and Thao Dang (VERIMAG, Grenoble), Carla Piazza (University of Udine)
- C++ tool designed to solve the reachability and parameter synthesis problem for discrete-time polynomial dynamical systems.
- the sets reachable by the system can be represented with boxes, parallelotopes, while the parameter sets are represented by polytopes
- The reachability computation and parameters refinement are carried out representing the polynomials in Bernstein form and transforming the problems into linear programs

A glimpse of Bernstein polynomials

- Bernstein basis for polynomials of degree 2 over $[0, 1]$:
  $B_0^2(x) = \binom{2}{0} x^0 (1-x)^{2-0} = (1-x)^2$
  $B_1^2(x) = \binom{2}{1} x^1 (1-x)^{2-1} = 2x(1-x)$
  $B_2^2(x) = \binom{2}{2} x^2 (1-x)^{2-2} = x^2$
- A polynomial represented with this basis would be expressed as
  $P(x) = c_0 B_0^2(x) + c_1 B_1^2(x) + c_2 B_2^2(x)$
- For $n \to \infty$ this polynomial representation converges uniformly to the represented function

Ariadne

`http://www.ariadne-cps.org`

- developed by a joint team including the University of Maastricht and the University of Verona
- C++ library, with Python bindings
- non-linear dynamics and guards
- supports the composition of hybrid automata
- computes both over- and lower- approximations of the reachable set
- state space is represented using Taylor image sets and *k*d-trees
- results are guaranteed to be sound (rigorous interval arithmetic)
- scalability: up to 10 continuous variables

Verification by discrete abstraction

An alternative approach, which approximates the system instead of the reachable set:

- compute a discrete abstract system that overapproximates the behavior of the original system
- does the abstract system satisfy the property?
  - YES: the original system is safe
  - NO: reject spurious counterexamples, refine the abstraction and repeat the verification

PROS: can obtain a result in a few steps even for complex systems

CONS: no graphical output of the results, cannot simulate the system

HSOLVER

`http://hsolver.sourceforge.net/`

- developed by Stefan Ratschan
- written in C++, textual scripting interface
- non-linear dynamics and guards
- no support for the composition of hybrid automata
- uses constraint propagation techniques to approximate the system
- results are guaranteed to be sound (rational arithmetic with unlimited precision)
- scalability: up to 10 continuous variables

HybridSAL

`http://sal.csl.sri.com/hybridsal/`

- developed by Ashish Tiwari
- written in Java and LISP, textual scripting interface
- polynomial dynamics and guards
- supports the composition of hybrid automata
- uses predicate abstraction to abstract the discrete dynamics and qualitative reasoning to abstract the continuous dynamics
- results are guaranteed to be sound
- scalability: up to 10 continuous variables

dReach/dReal

`http://dreal.github.io/dReach/`

- developed at CMU by Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke
- dReach is a bounded reachability analysis tool for nonlinear hybrid systems
- it encodes reachability problems of hybrid systems to first-order formulas over real numbers, which are solved by delta-decision procedures in the SMT solver dReal
- dReach is able to handle a wide range of highly nonlinear hybrid systems
- it has scaled well on various realistic models from biomedical and robotics applications

Bounded δ-reachability - 1

Consider an hybrid system and SMT formulas that encode it, a numerical error bound δ, and for any formula ϕ a δ-perturbation of ϕ, written as ϕ^δ, yielding a perturbed system H^δ. Choose $n \in N$ as a bound on the number of discrete mode changes and $T \in R^+$ as an upper bound on the time duration. Let *unsafe* encode a subset of $X \times Q$, the state space of H.
The bounded δ-reachability problem asks for one of the following answers:

- *safe*: H cannot reach *unsafe* in n steps within time T
- δ-*unsafe*: H^δ can reach *unsafe*$^\delta$ in n steps within time T

Bounded δ-reachability - 2

These answers are not weaker than the precise ones. When *safe* is the answer, we know for certain that the system does not reach the unsafe region; when δ-*unsafe* is the answer, there exists some δ-bounded perturbation in the system that would make it unsafe. The error-bound δ can be chosen to be sufficiently small so that the δ-*unsafe* answer discovers robustness problems in the system, which should consequently be classified as *unsafe* for real.

HyCOMP

`https://es.fbk.eu/tools/hycomp/`

- developed at FBK by Alessandro Cimatti, Sergio Mover, Stefano Tonetta.
- written in C, textual scripting interface
- piece-wise linear and affine dynamics
- supports the composition of hybrid automata
- solves different tasks: verification of invariant and LTL properties, verification of scenario specifications, parameter synthesis.
- verification algorithms based on Satisfiability Modulo Theory
- results are sound (infinite-precision arithmetic)
- scalability: up to 60 continuous variables

nuXmv

https://nuxmv.fbk.eu/

- nuXmv is a symbolic model checker for the analysis of synchronous finite-state and infinite-state systems
- for the finite-state case, nuXmv features a strong verification engine based on state-of-the-art SAT-based algorithms
- For the infinite-state case, nuXmv features SMT-based verification techniques, implemented through a tight integration with MathSAT5

NuSMV

`https://nusmv.fbk.eu/`

- NuSMV is a symbolic model checker developed as a joint project between: FBK-IRST at Trento, Carnegie Mellon University, University of Genova and University of Trento
- NuSMV is a reimplementation and extension of SMV, the first model checker based on BDDs
- NuSMV2, combines BDD-based model checking (with the CUDD library developed by Fabio Somenzi at the Colorado University) and SAT-based model checking (RBC-based Bounded Model Checker, which can be connected to the Minisat SAT Solver and/or to the ZChaff SAT Solver)

Falsification by simulation

- Explore the state space of the system by computing a bunch of trajectories
- If one of the trajectories violates the property, a counterexample is found
- If no counterexample is found, no conclusion can be made on the truth of the property
- Can be used to verify black-box systems

PROS: can manage complex properties and black-box systems

CONS: can only falsify the property

Breach

`https://www.eecs.berkeley.edu/~donze/breach_page.html`

- developed by Alexandre Donzé
- Matlab/C++ toolbox
- Systematic simulation is used to compute an underapproximation of the reachable set
- supports complex properties in Signal Temporal Logic
- supports parametric systems

S-TaLiRo

`https://sites.google.com/a/asu.edu/s-taliro/`

- developed by Sriram Sankaranarayanan
- Matlab toolbox
- uses a robustness metric to search for a counterexample
- randomized testing and stochastic optimization techniques are used to maximize the chance of finding the counterexample
- supports complex properties in Metric Temporal Logic
- supports parametric systems

C2E2

`https://publish.illinois.edu/c2e2-tool/`

- developed by the C2E2 development team at University of Illinois at Urbana-Champaign
- can verify bounded-time invariants of hybrid automata and Stateflow models
- it generates numerical simulations, and it iteratively refines reach set over-approximations to prove invariants
- it can also find counterexamples or bug traces

HyLAA

`http://stanleybak.com/hylaa/`

- developed by Stanley Bak with input from Parasara Sridhar Duggirala
- verification tool for system models with linear ODEs, time-varying inputs, and hybrid dynamics
- computes simulation-equivalent reachability: the set of states that can be reached by any fixed-step simulation under any possible input
- results are exact (under some restrictions)
- can generate counter-example traces when an error is found

Verification by theorem proving

- System under verification represented by a logical formula *Sys*
- Properties of interest represented by a logical formula *Prop*
- The verification problem is reduced to testing whether the formula $Sys \rightarrow Prop$ is valid (a logical tautology)
- Systems can be parametric and/or partially specified
- Very complex properties can be tested
- User intervention is needed to guide the proof

PROS: can manage complex properties and partially specified systems

CONS: semi-automatic approach

KeYmaera X

`http://www.ls.cs.cmu.edu/KeYmaeraX/`

- developed by André Platzer
- written in Java, graphical user interface
- hybrid systems are specified using a programming language
- non-linear dynamics and guards
- supports complex properties written in Differential Dynamic Logic
- pre-defined and custom tactics drive the automated proof search

Verification of Stochastic Hybrid Systems

- Stochastic hybrid systems extend hybrid systems with probabilistic aspects
- Explicitly probabilistic models are useful in a number of applicative instances:
  - when the uncertainty of the system cannot be "averaged out"
  - when the knowledge of the system is too coarse
  - when stochastic mechanisms play a role in the system under study

PROS: a richer class of models that extends deterministic ones

CONS: more and deeper technicalities that are required in the analysis

StocHy

https://gitlab.com/natchi92/StocHy

- developed at the Oxford Control and Verification group (OXCAV) led by Alessandro Abate
- C++ tool
- support discrete time stochastic hybrid systems
- can parse well known state space models
- three analysis tasks:
  1. simulation of stochastic processes
  2. formal verification via abstractions
  3. strategy synthesis

Outline

Conclusions

- Formal Verification is a systematic way of checking whether all behaviors of a model of a system fulfill its specification
- Formal Verification is in use in the industry in several different areas, mainly for the design of discrete hardware/software systems
- Formal Verification can be applied to hybrid systems also, however, with limitations on the properties and/or on the dynamics of the system
- Different approaches are available. The choice of the best one is strongly dependent on the application domain, on the system under verification

A selection of publications

General references

R. Alur "Principles of Cyber-Physical Systems", MIT Press, 2015

E. A. Lee; S. A. Seshia "Introduction to Embedded Systems, A Cyber-Physical Systems Approach", Second Edition, MIT Press, 2017

P. Nuzzo; A. L. Sangiovanni-Vincentelli "Lets get physical: Computer science meets systems", in From Programs to Systems. The Systems perspective in Computing, ser. Lecture Notes in Computer Science, S. Bensalem, Y. Lakhneck, and A. Legay, Eds. Springer Berlin Heidelberg, 2014, vol. 8415, pp. 193208, 126(2), p. 183-235, 1995

Theory of hybrid automata

R. Alur; D. L. Dill "A theory of timed automata", Theoretical Computer Science, 126(2), p. 183-235, 1995

R. Alur; C. Courcoubetis; N. Halbwachs; T. A. Henzinger; P.- H. Ho; X. Nicollin; A. Olivero; J. Sifakis; S. Yovine "The algorithmic analysis of hybrid systems", Theoretical Computer Science, 138(1), p. 3-34, 1995

T. A. Henzinger; P. W. Kopke; A. Puri; P. Varaiya "What's Decidable about Hybrid Automata?", Journal of Computer and System Sciences, 57, p. 94-124, 1998

N. Lynch; R. Segala "Hybrid I/O Automata", Information and Computation, 185(1), p. 103-157, 2003

Survey on the state of the art

P. Nuzzo; A.L. Sangiovanni-Vincentelli; D. Bresolin; L. Geretti; T. Villa "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems", Proceedings of the IEEE, Volume 103, Issue 11, 2015, pg. 2104-2132, DOI: 10.1109/JPROC.2015.2453253