Modeling and Analysis of Hybrid Systems

Erika Ábrahám

RWTH Aachen University, Germany

Beijing, September 2013

Contents

1 Modeling

2 Reachability analysis

3 Counterexample generation

Contents

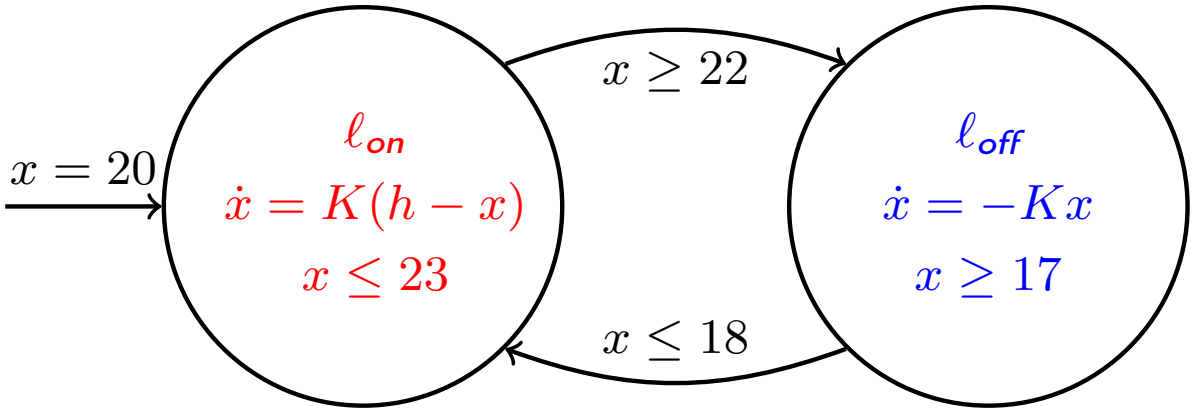
1 Modeling

2 Reachability analysis

3 Counterexample generation

Modeling with hybrid automata

Thermostat example



Some interesting subclasses of hybrid automata

subclass	derivatives	conditions	bounded reachability	unbounded reachability
timed automata	$\dot{x} = 1$	$x \sim c$	decidable	decidable
initialized rectangular automata	$\dot{x} \in [c_1, c_2]$ reset by derivative change	$x \sim [c_1, c_2]$	decidable	decidable
linear hybrid automata I	$\dot{x} = c$	$x \sim g_{linear}(\vec{x})$	decidable	undecidable
linear hybrid automata II	$\dot{x} = f_{linear}(\vec{x})$	$x \sim g_{linear}(\vec{x})$	undecidable	undecidable
general hybrid automata	$\dot{x} = f(\vec{x})$	$x \sim g(\vec{x})$	undecidable	undecidable

[Henzinger et al., 1998]

Contents

1 Modeling

2 Reachability analysis

3 Counterexample generation

Some tools

- Uppaal [Behrmann et al., 2004]
- HyTech [Henzinger et al., 1997]
- PHAVer [Frehse, 2005]
- SpaceEx [Frehse et al., 2011]
- d/dt [Asarin et al., 2002]
- Ellipsoidal toolbox [Kurzhanski et al., 2006]
- MATISSE [Girard et al., 2007]
- Multi-Parametric Toolbox [Kvasnica et al., 2004]
- Flow* [Chen et al., 2012]

The two most popular techniques for reachability analysis

Given: hybrid automaton + set of unsafe states

Abstraction

Iterative forward/backward search

Iterative forward search

Iterative forward search

We need a (possibly over-approximative) state set representation and operations on them like intersection, union, linear transformation and Minkowski sum.

Iterative forward search

We need a (possibly over-approximative) state set representation and operations on them like intersection, union, linear transformation and Minkowski sum.

The representation is crucial for

- the representation size,
- efficiency and
- accuracy.

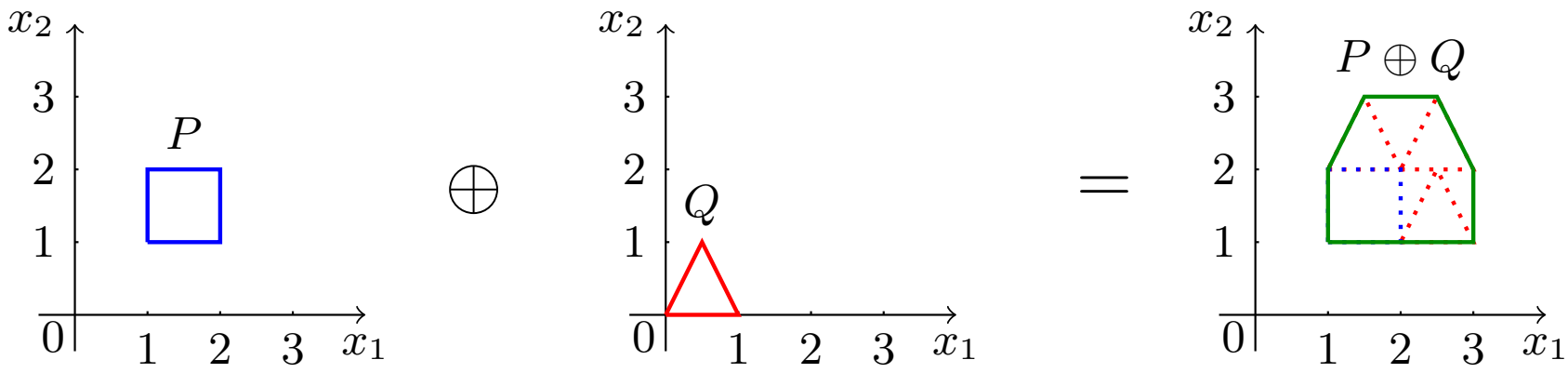
Iterative forward search

We need a (possibly over-approximative) state set representation and operations on them like intersection, union, linear transformation and Minkowski sum.

The representation is crucial for

- the representation size,
- efficiency and
- accuracy.

Minkowski sum



$$P \oplus Q = \{p + q \mid p \in P \text{ and } q \in Q\}$$

Most well-known state set representations

Geometric objects:

- hyperrectangles [Moore et al., 2009]
- oriented rectangular hulls [Stursberg et al., 2003]
- convex polyhedra [Ziegler, 1995] [Chen at el, 2011]
- orthogonal polyhedra [Bournez et al., 1999]
- template polyhedra [Sankaranarayanan et al., 2008]
- ellipsoids [Kurzhanski et al., 2000]
- zonotopes [Girard, 2005])

Other symbolic representations:

- support functions [Le Guernic et al., 2009]
- Taylor models [Berz and Makino, 1998, 2009] [Chen et al., 2012]

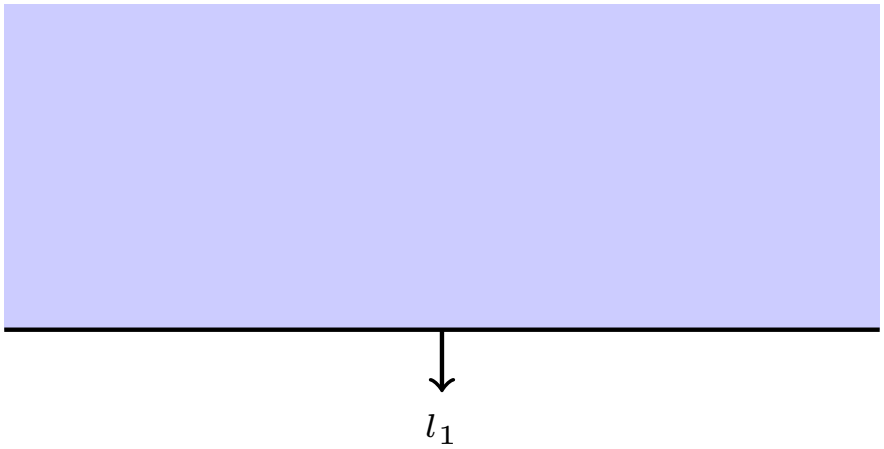
Example: Polytopes

Example: Polytopes

- Halfspace: set of points satisfying $l \cdot x \leq z$

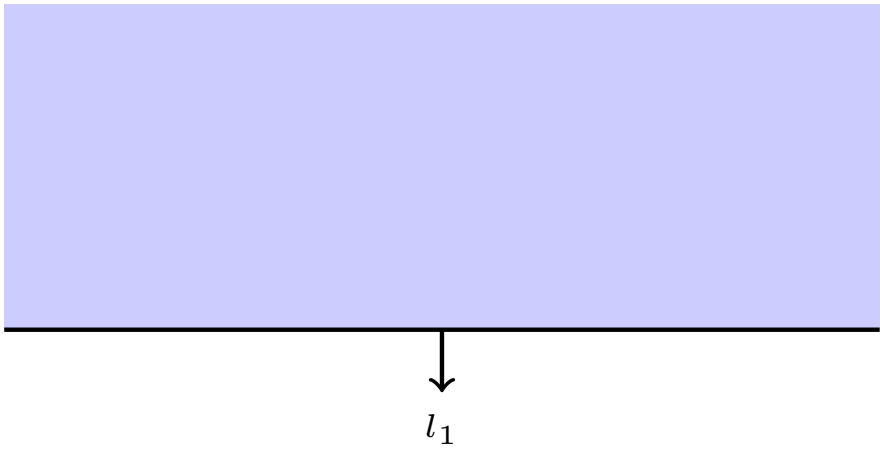
Example: Polytopes

- Halfspace: set of points satisfying $l \cdot x \leq z$



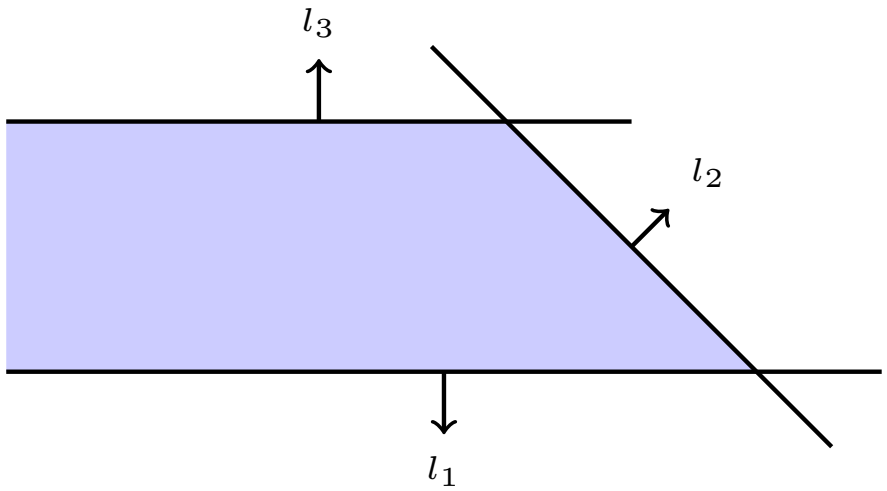
Example: Polytopes

- Halfspace: set of points satisfying $l \cdot x \leq z$
- Polyhedron: an intersection of finitely many halfspaces



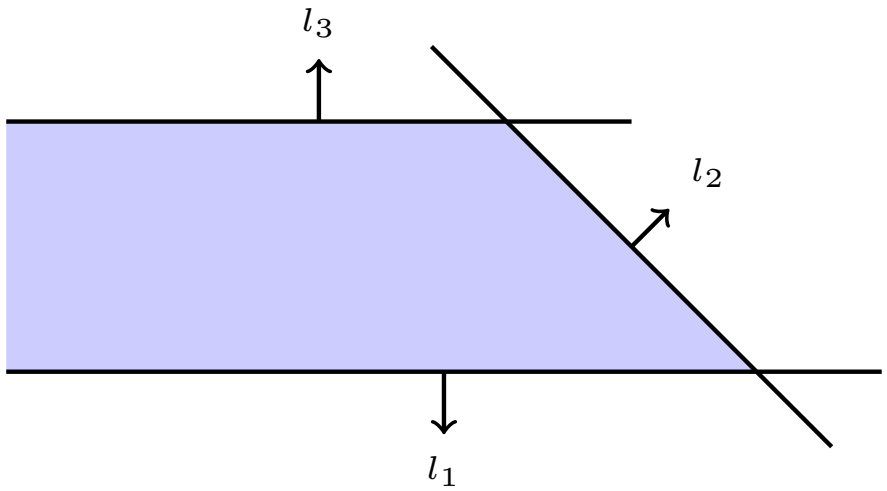
Example: Polytopes

- Halfspace: set of points satisfying $l \cdot x \leq z$
- Polyhedron: an intersection of finitely many halfspaces



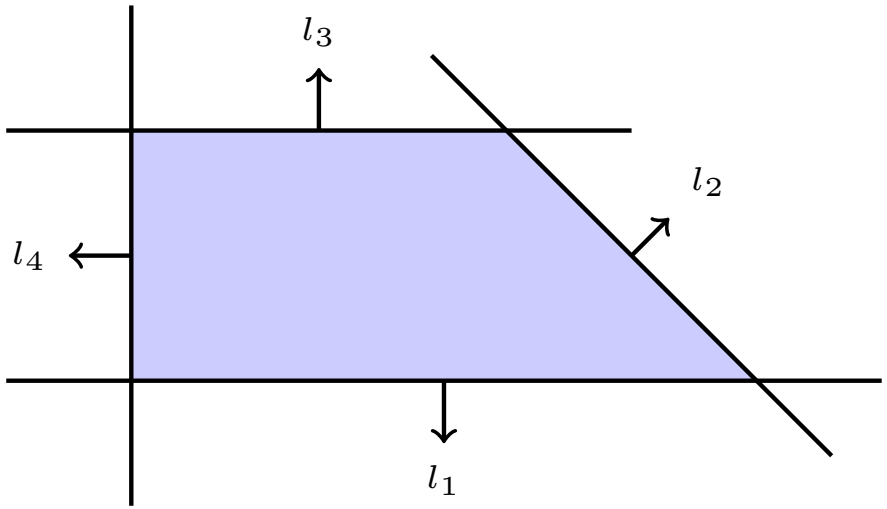
Example: Polytopes

- Halfspace: set of points satisfying $l \cdot x \leq z$
- Polyhedron: an intersection of finitely many halfspaces
- Polytope: a bounded polyhedron



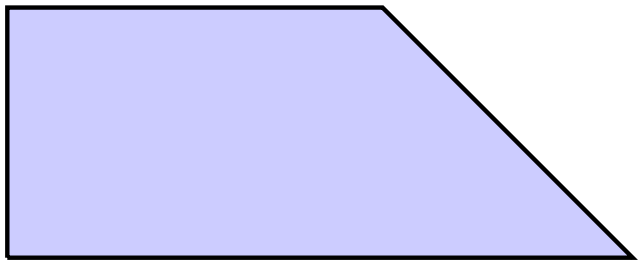
Example: Polytopes

- Halfspace: set of points satisfying $l \cdot x \leq z$
- Polyhedron: an intersection of finitely many halfspaces
- Polytope: a bounded polyhedron



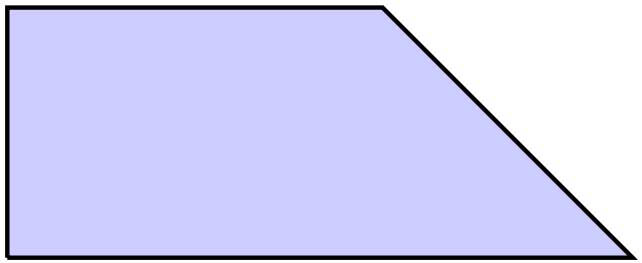
Example: Polytopes

- Halfspace: set of points satisfying $l \cdot x \leq z$
- Polyhedron: an intersection of finitely many halfspaces
- Polytope: a bounded polyhedron



Example: Polytopes

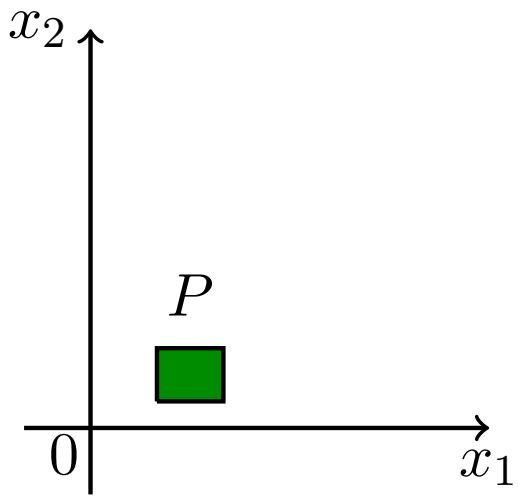
- Halfspace: set of points satisfying $l \cdot x \leq z$
- Polyhedron: an intersection of finitely many halfspaces
- Polytope: a bounded polyhedron



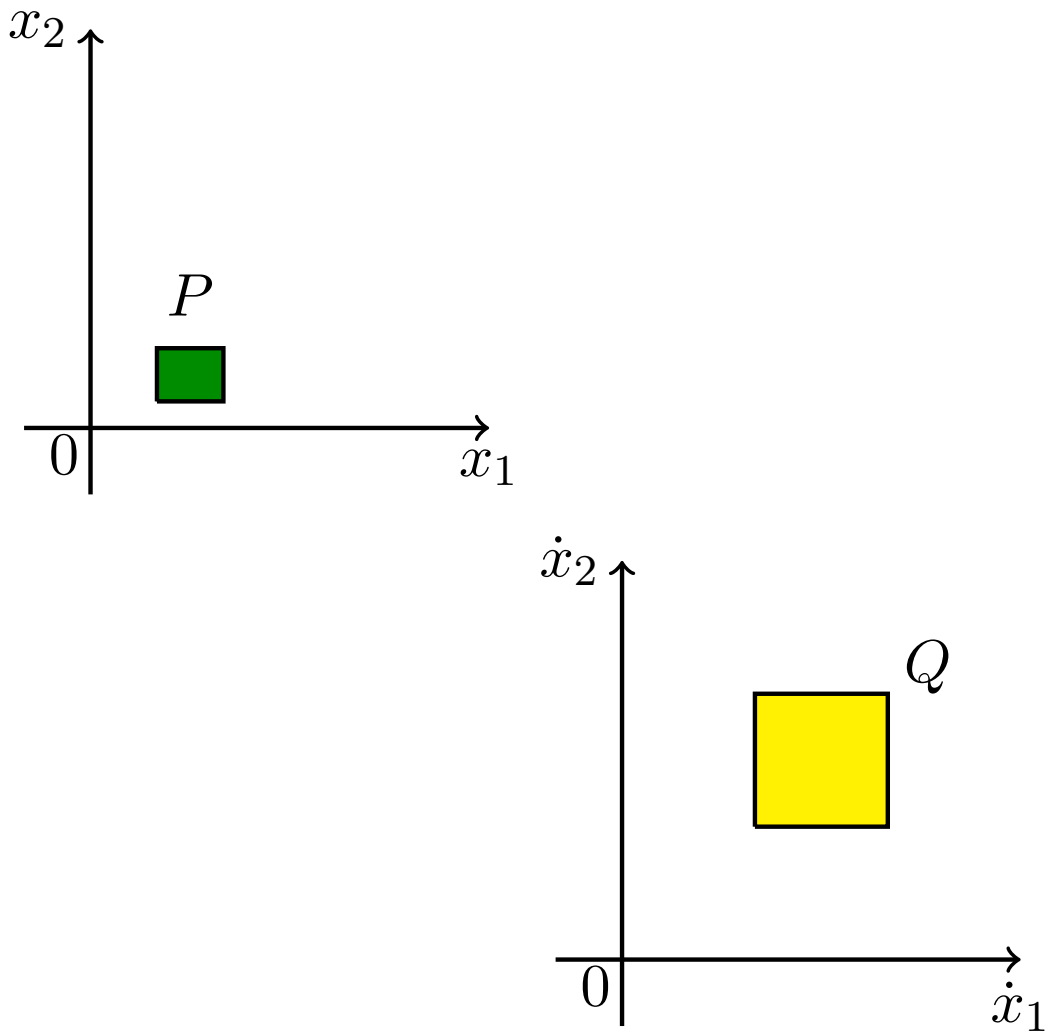
representation	union	intersection	Minkowski sum
$\mathcal{V}$-representation by vertices	easy	hard	easy
$\mathcal{H}$-representation by facets	hard	easy	hard

Linear hybrid automata I: Time evolution

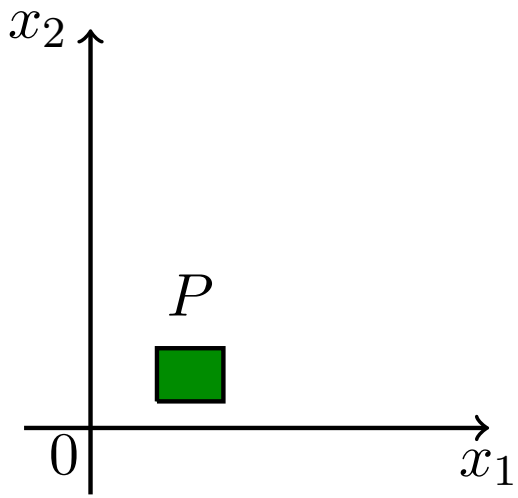
Linear hybrid automata I: Time evolution

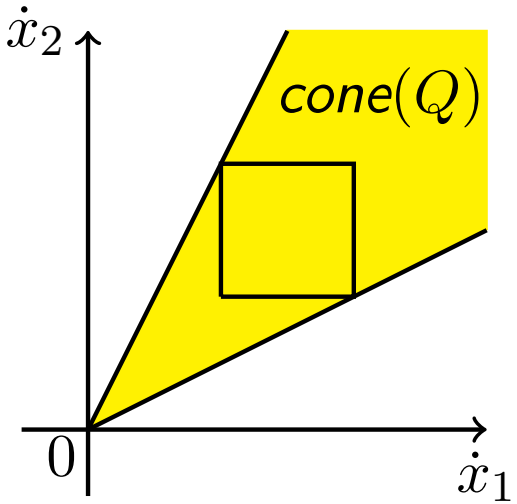


Linear hybrid automata I: Time evolution

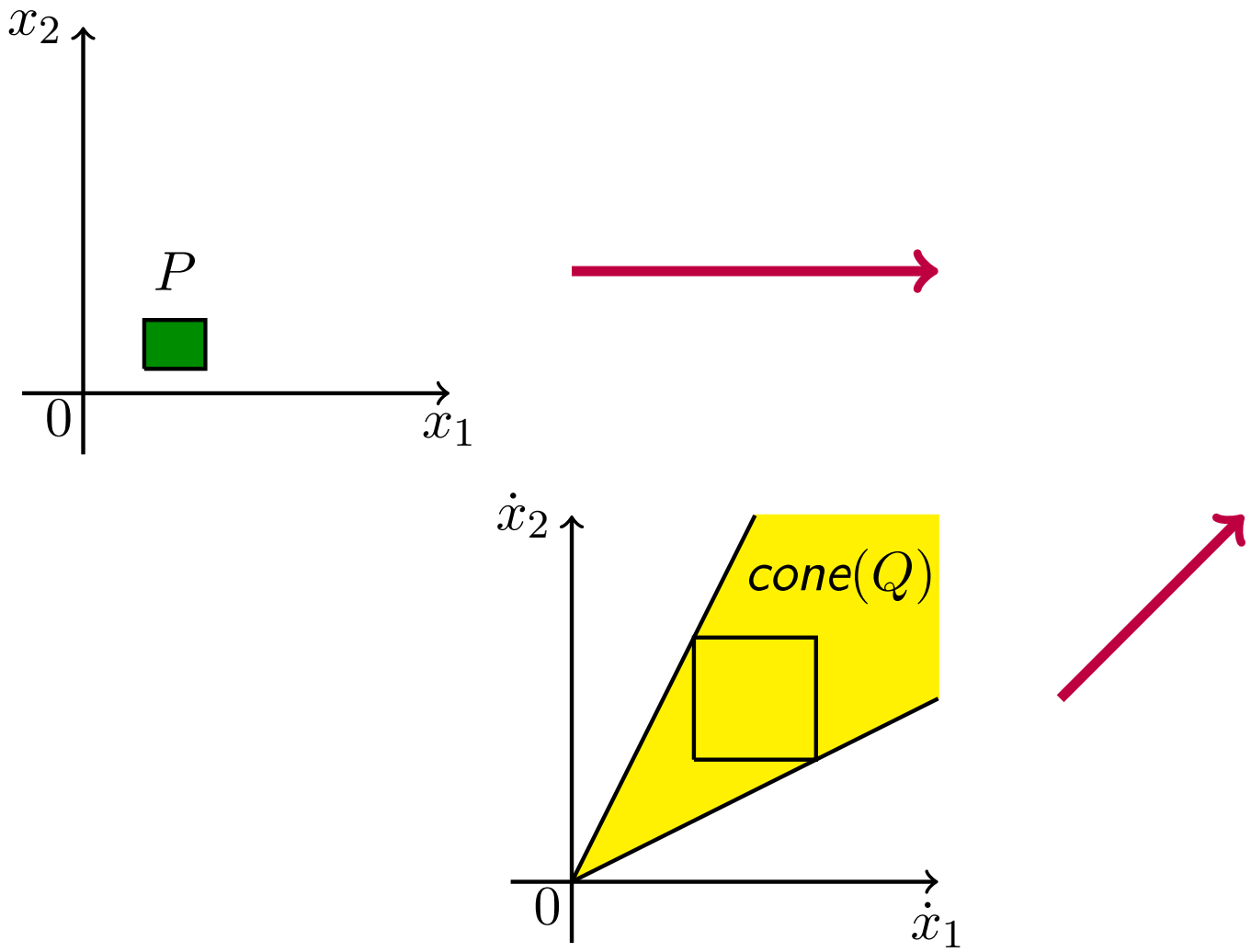


Linear hybrid automata I: Time evolution

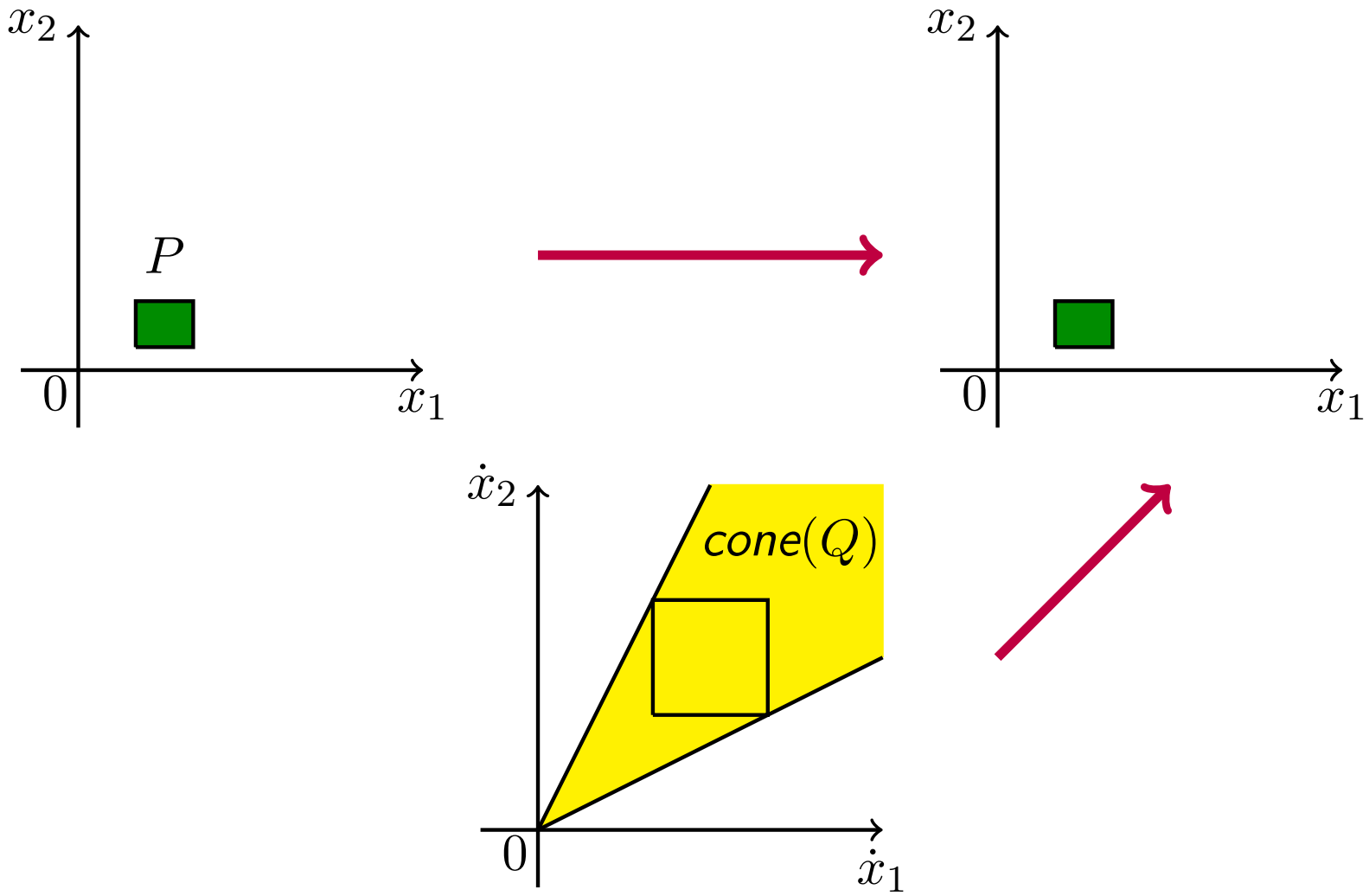




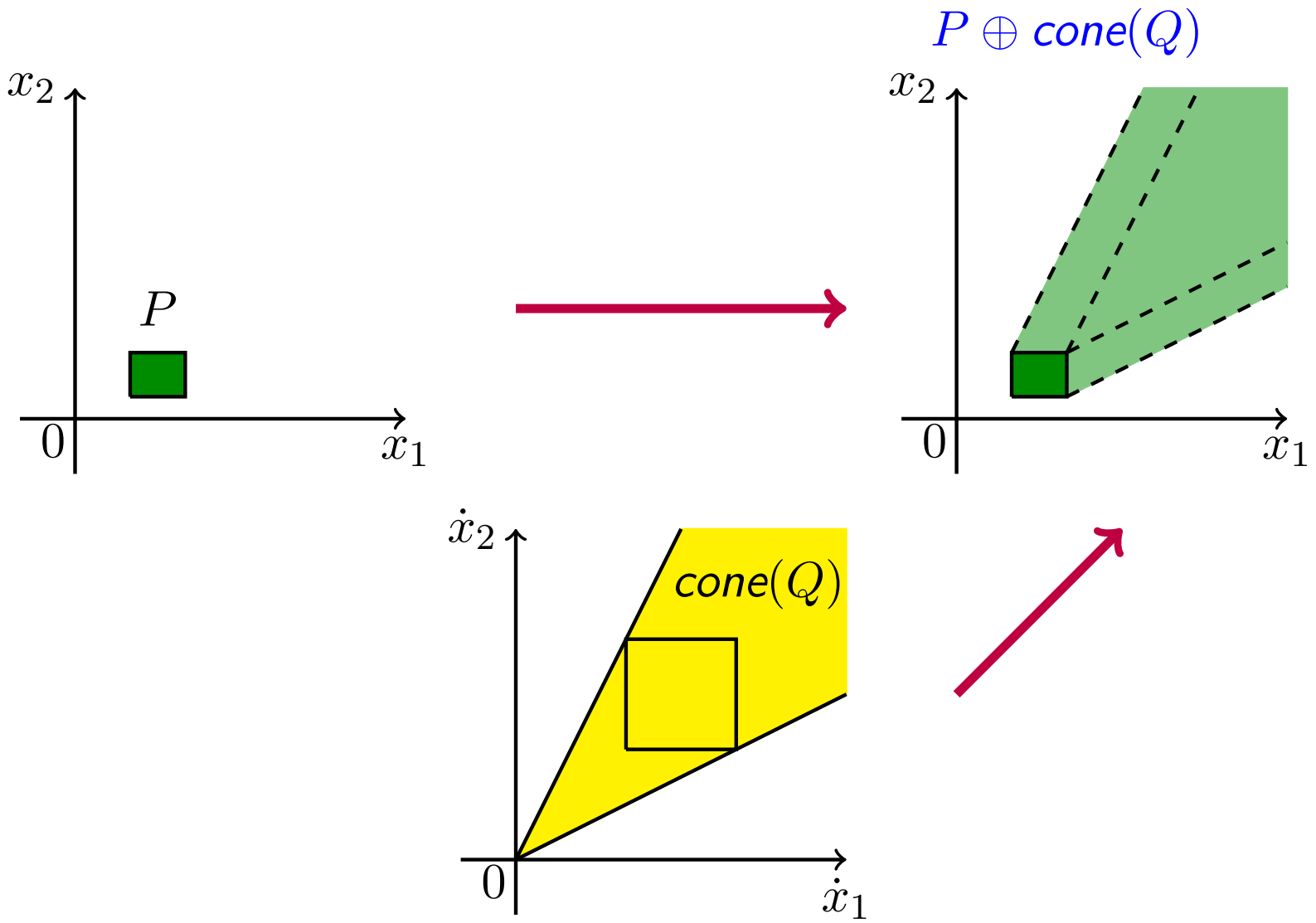
Linear hybrid automata I: Time evolution



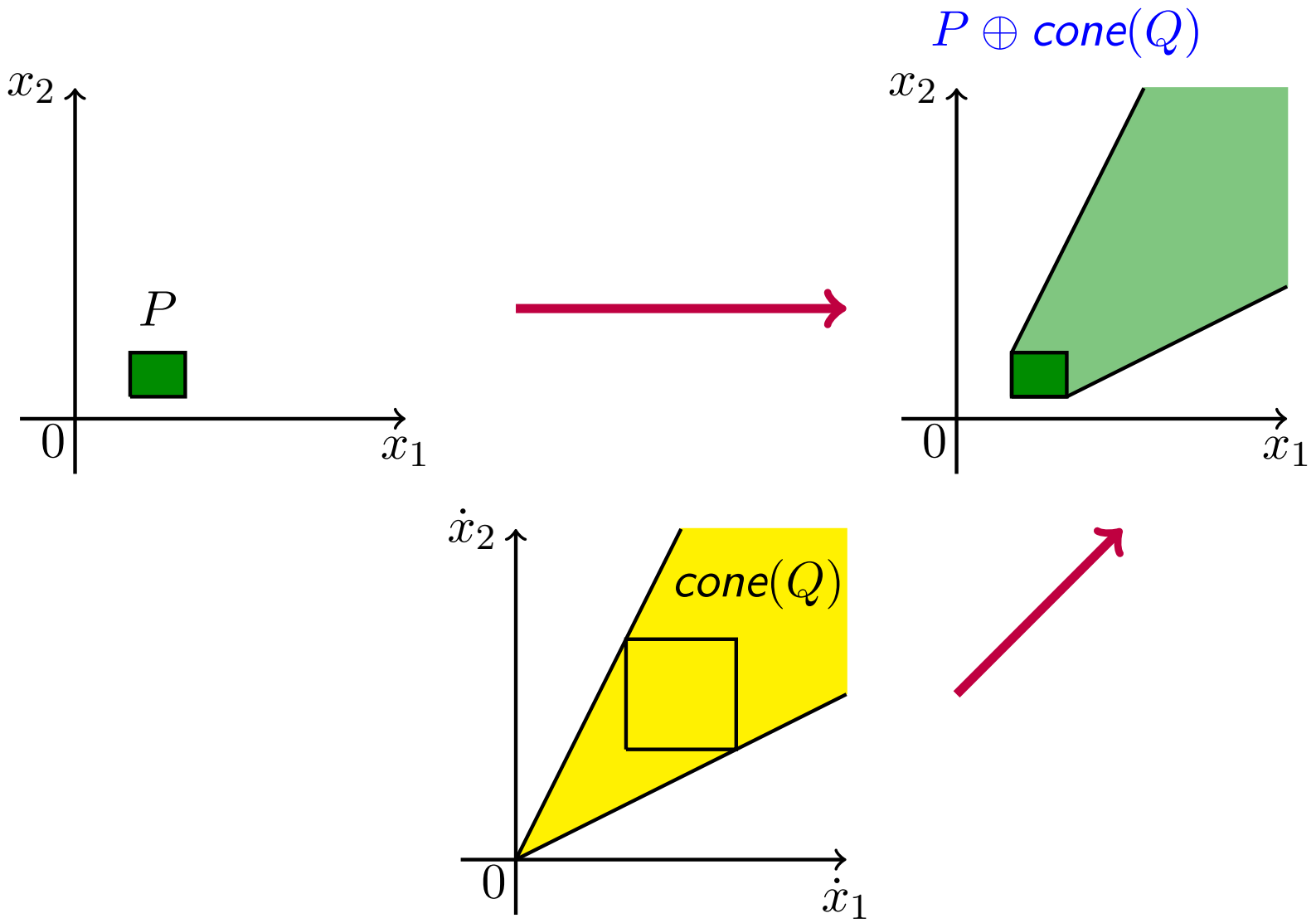
Linear hybrid automata I: Time evolution



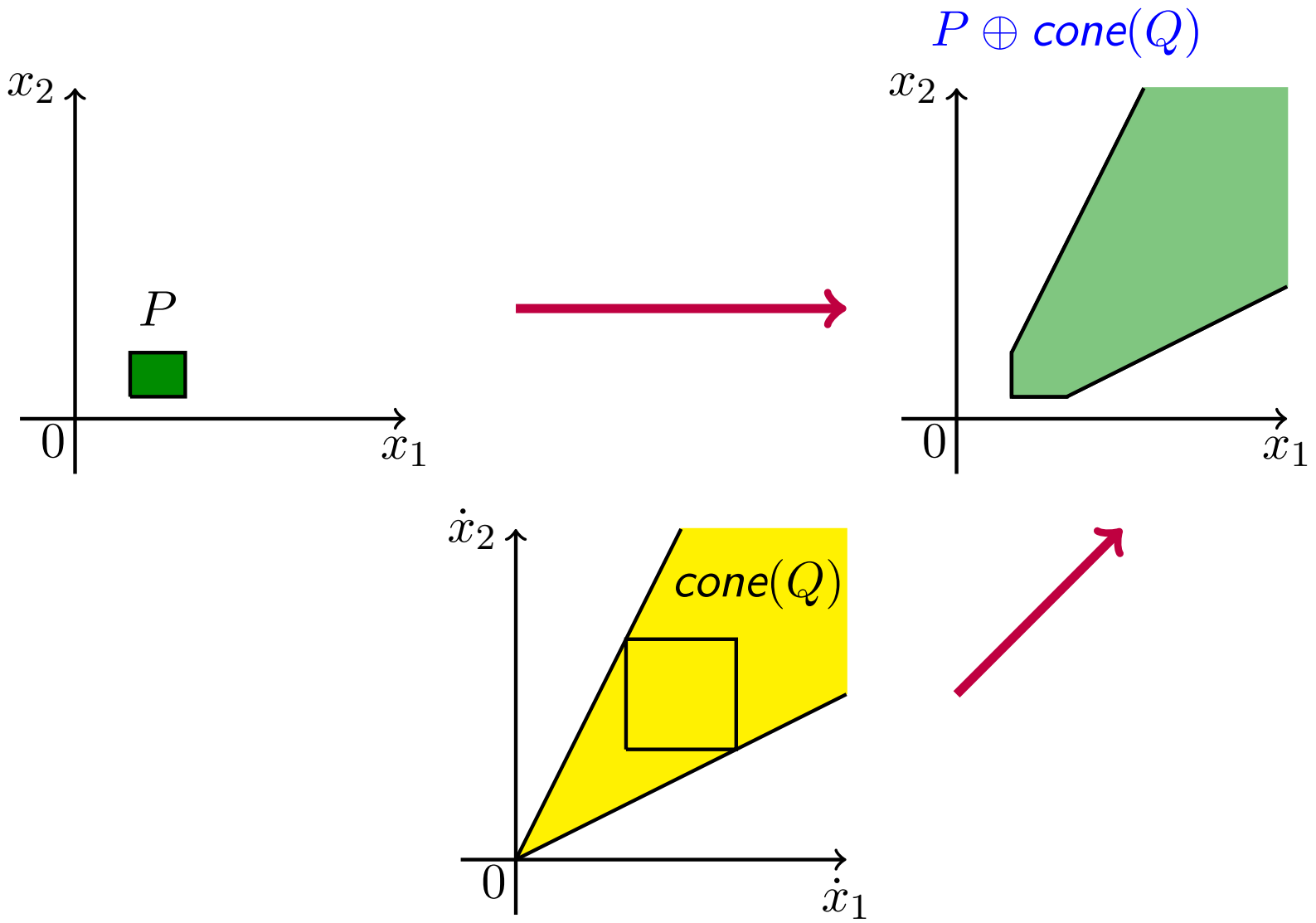
Linear hybrid automata I: Time evolution



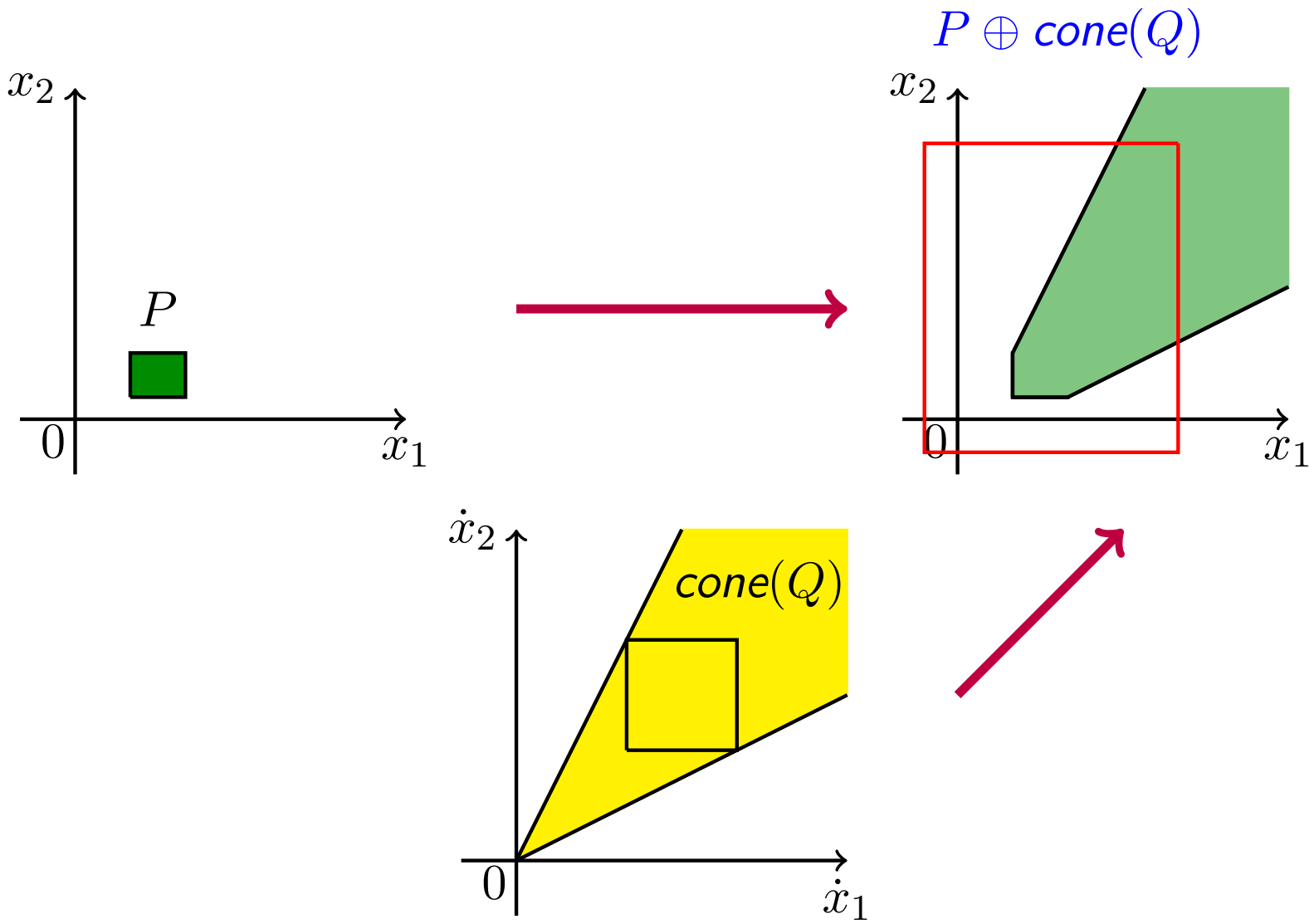
Linear hybrid automata I: Time evolution



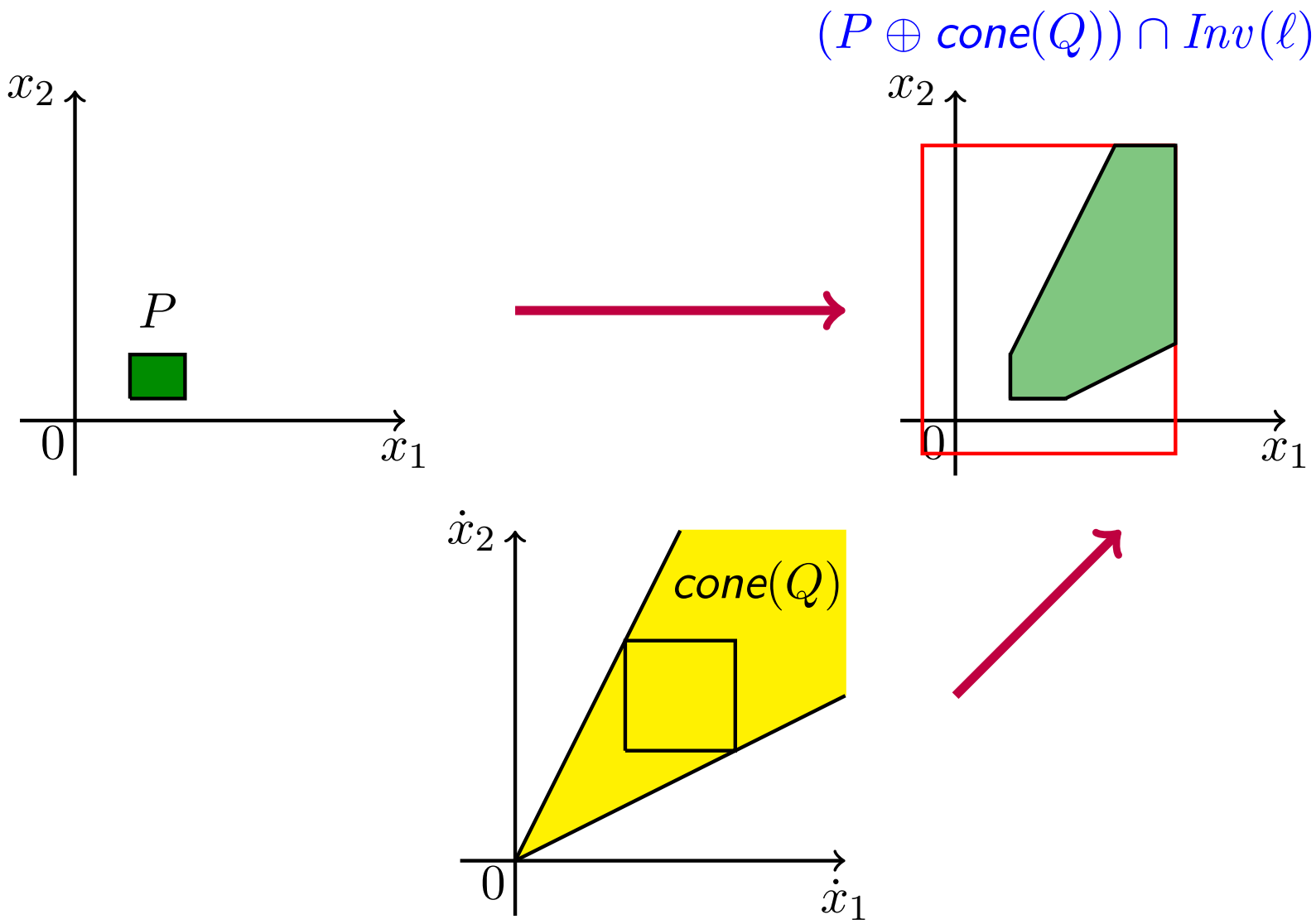
Linear hybrid automata I: Time evolution



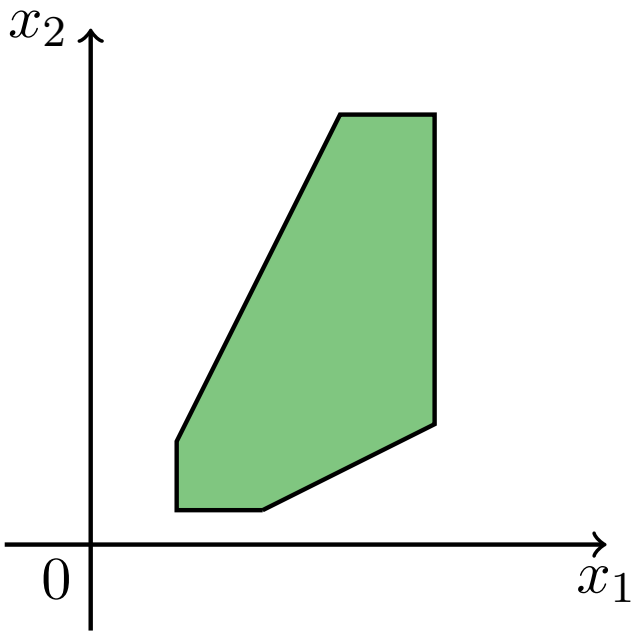
Linear hybrid automata I: Time evolution



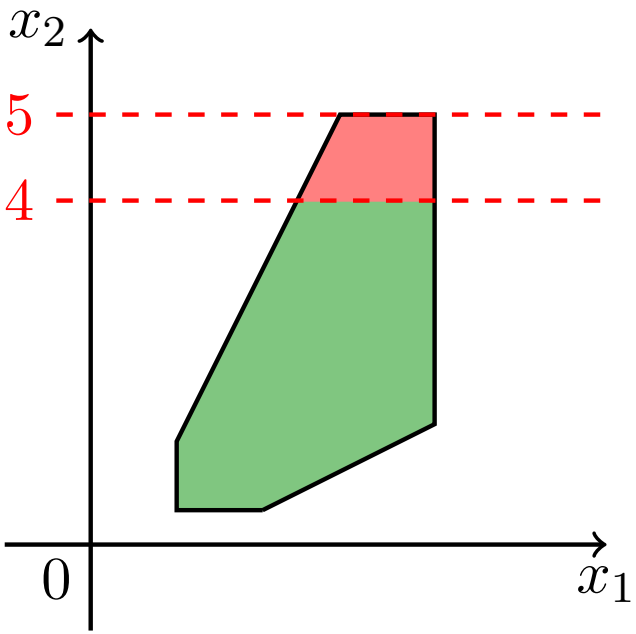
Linear hybrid automata I: Time evolution



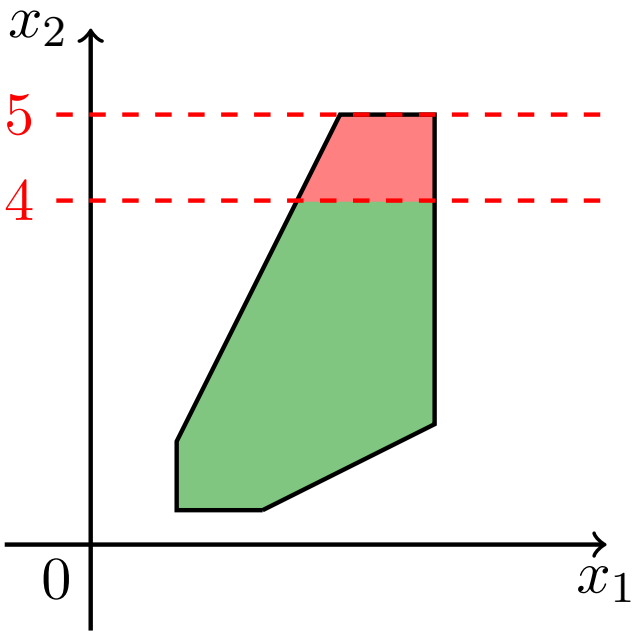
Linear hybrid automata I: Discrete steps (jumps)



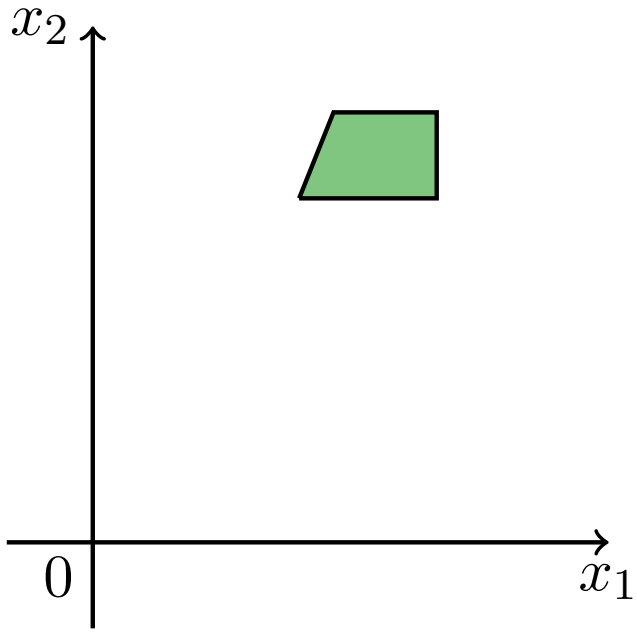
Linear hybrid automata I: Discrete steps (jumps)



Linear hybrid automata I: Discrete steps (jumps)

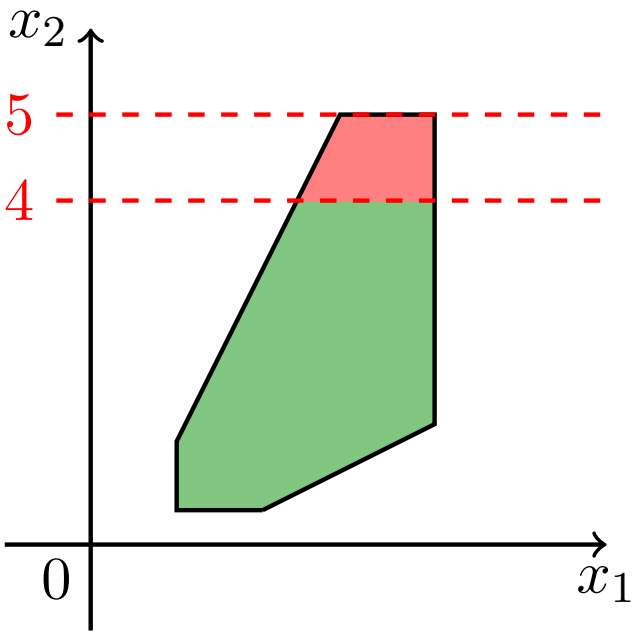


ℓ

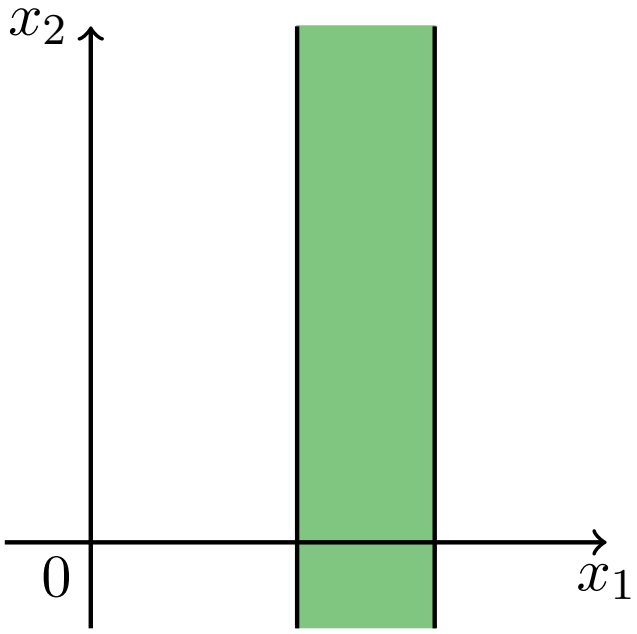


ℓ'

Linear hybrid automata I: Discrete steps (jumps)

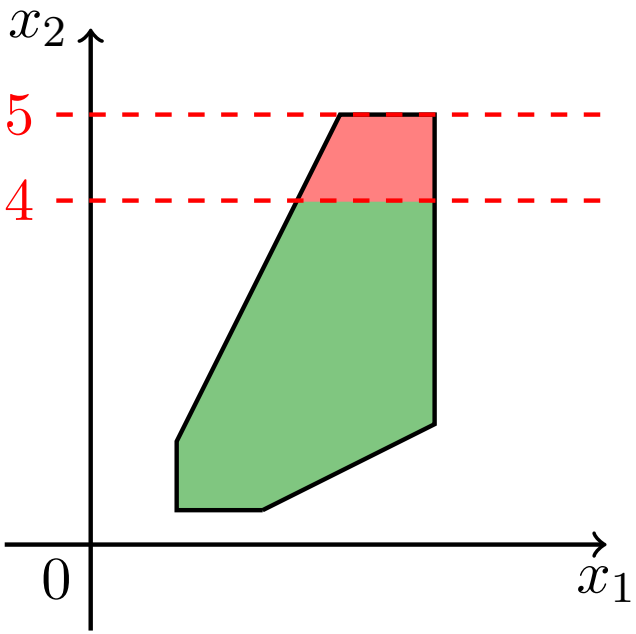


ℓ

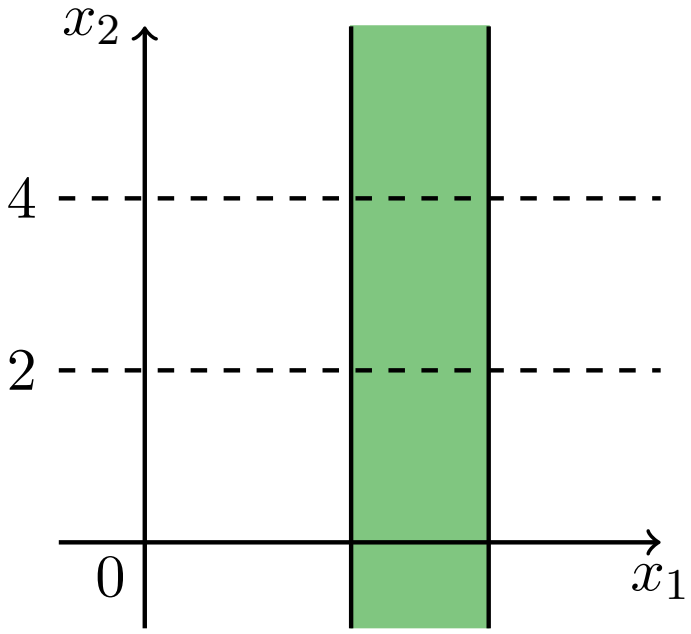


ℓ'

Linear hybrid automata I: Discrete steps (jumps)

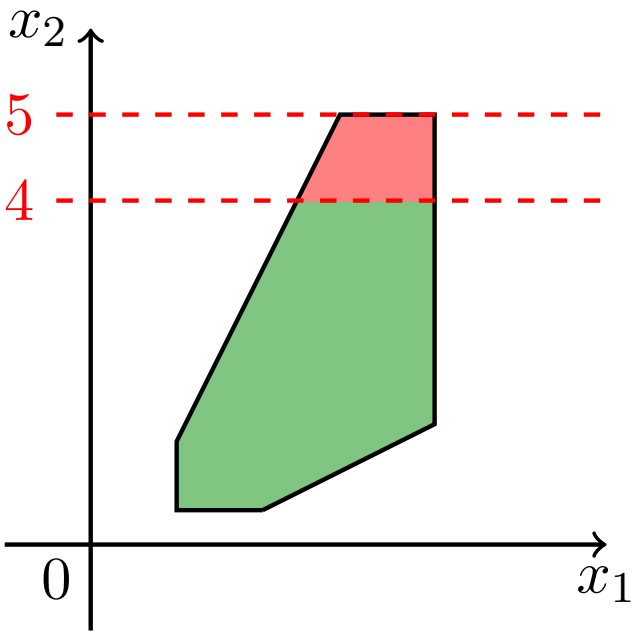


ℓ

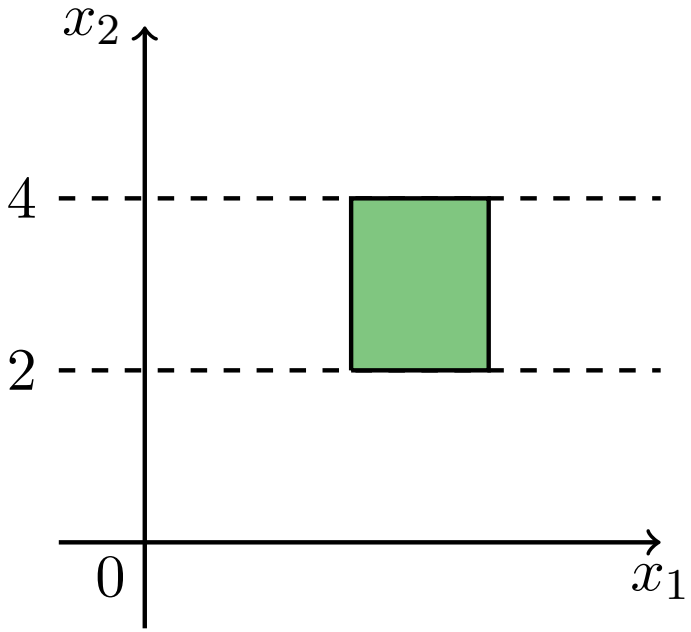


ℓ'

Linear hybrid automata I: Discrete steps (jumps)

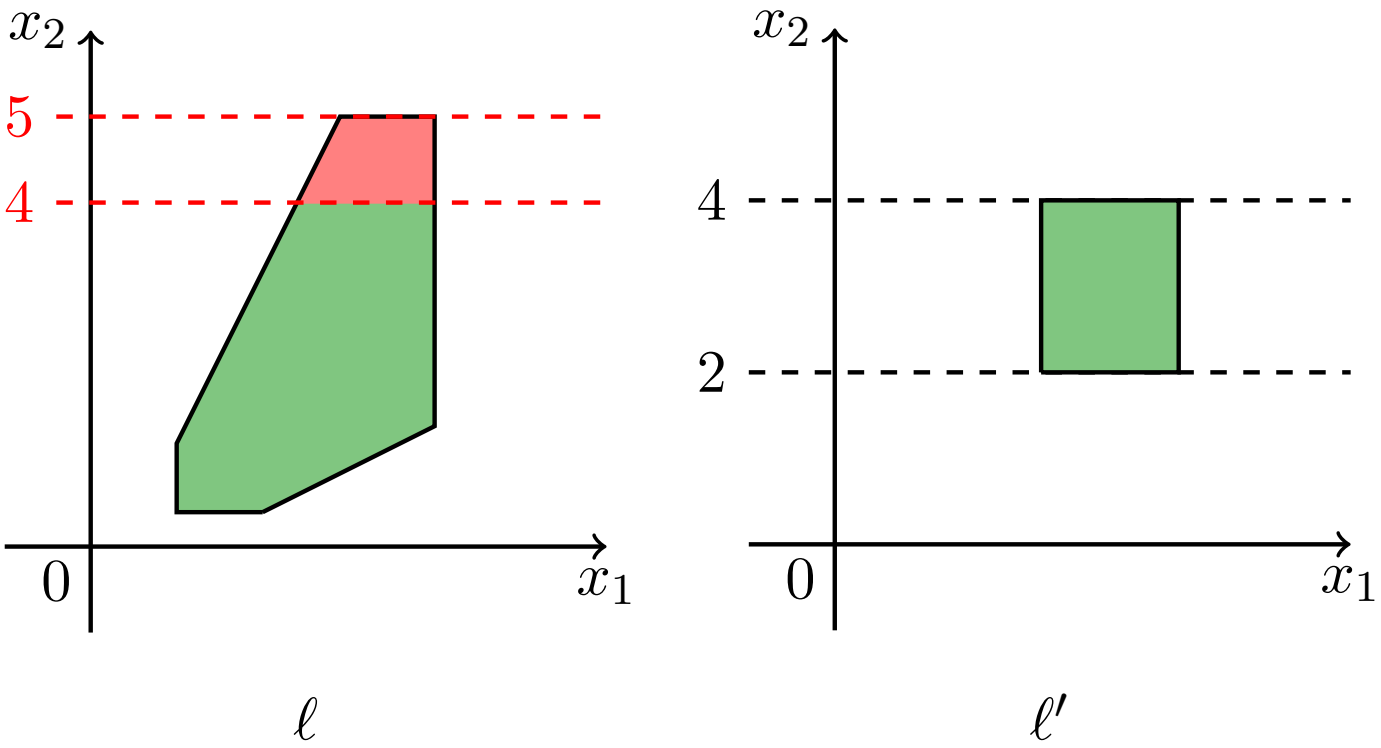


ℓ



ℓ'

Linear hybrid automata I: Discrete steps (jumps)

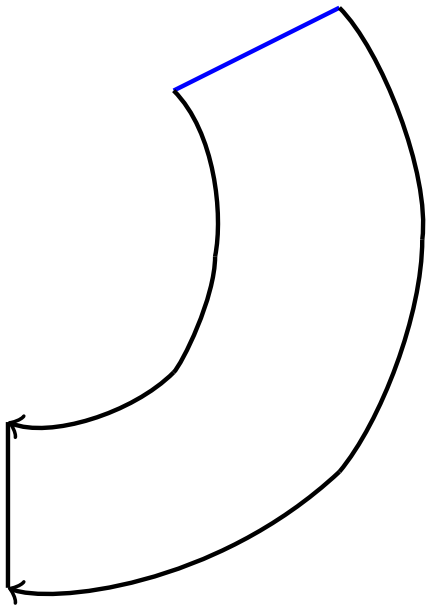


- Computed via projection and Minkowski sum.

Linear hybrid automata II: Time evolution

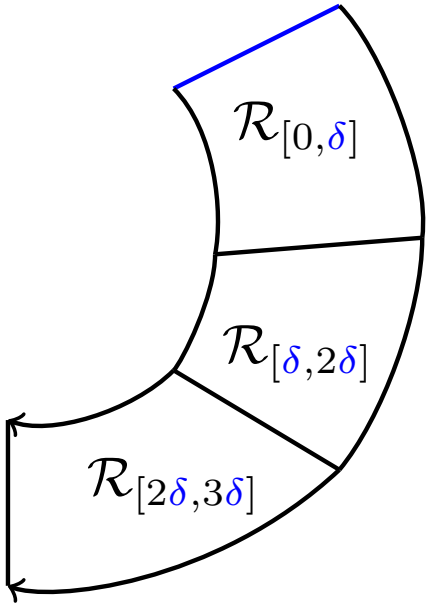
Linear hybrid automata II: Time evolution

- Assume $\dot{x} = Ax + Bu$



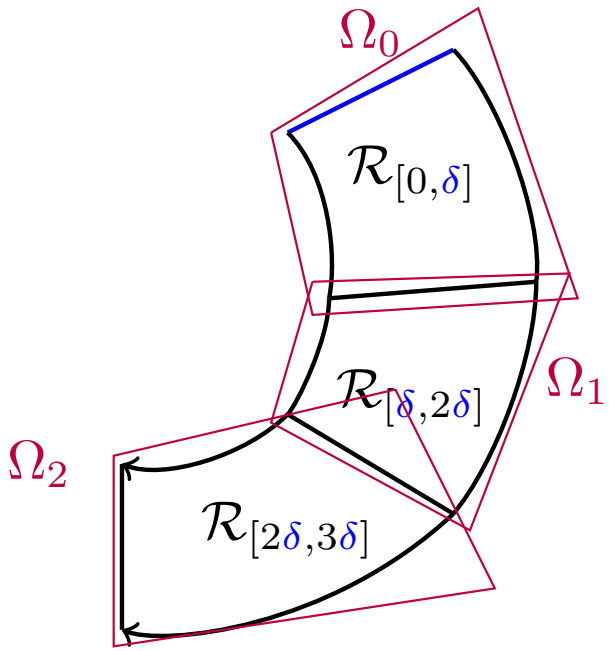
Linear hybrid automata II: Time evolution

- Assume $\dot{x} = Ax + Bu$



Linear hybrid automata II: Time evolution

- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$



Linear hybrid automata II: Time evolution

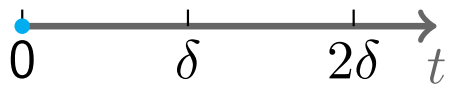
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

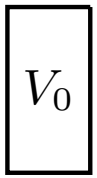
Linear hybrid automata II: Time evolution

- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

Linear hybrid automata II: Time evolution

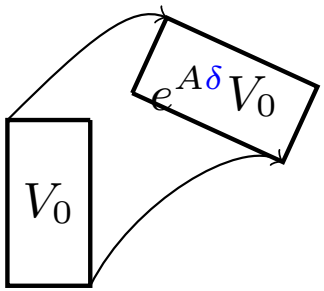
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

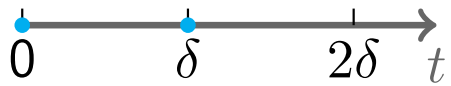




Linear hybrid automata II: Time evolution

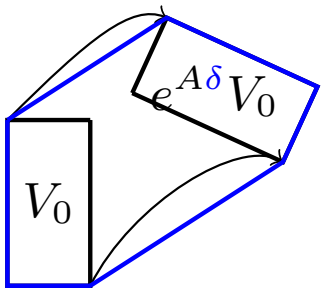
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

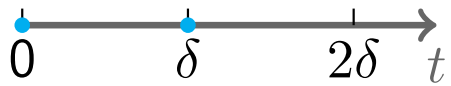




Linear hybrid automata II: Time evolution

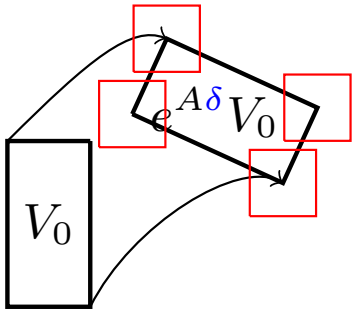
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

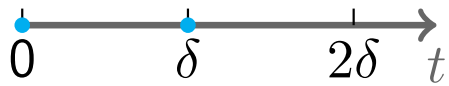




Linear hybrid automata II: Time evolution

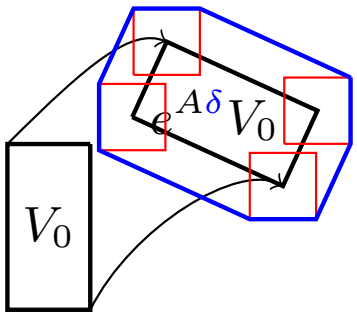
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

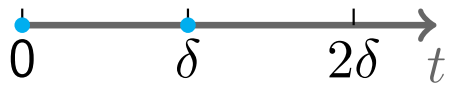




Linear hybrid automata II: Time evolution

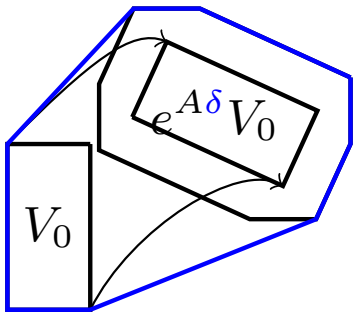
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

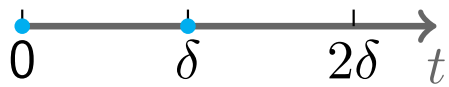




Linear hybrid automata II: Time evolution

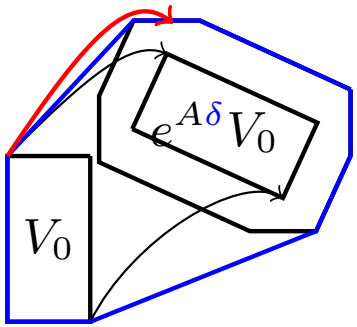
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

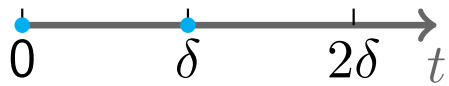




Linear hybrid automata II: Time evolution

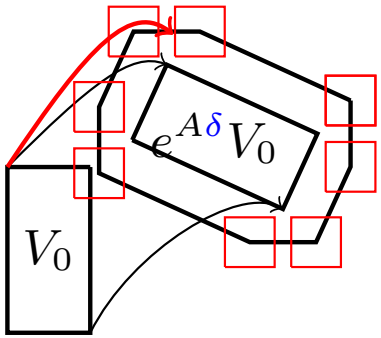
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

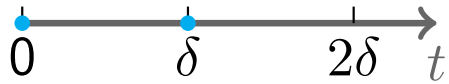




Linear hybrid automata II: Time evolution

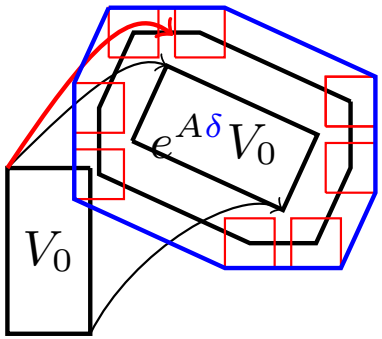
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

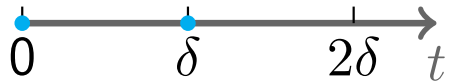




Linear hybrid automata II: Time evolution

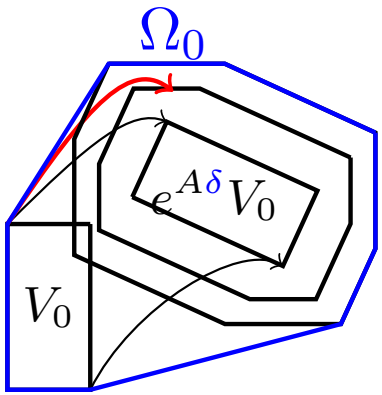
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

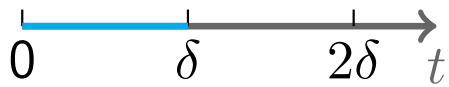




Linear hybrid automata II: Time evolution

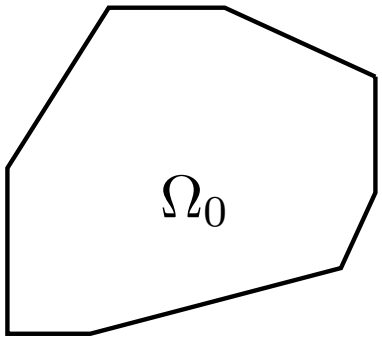
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

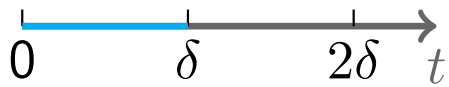




Linear hybrid automata II: Time evolution

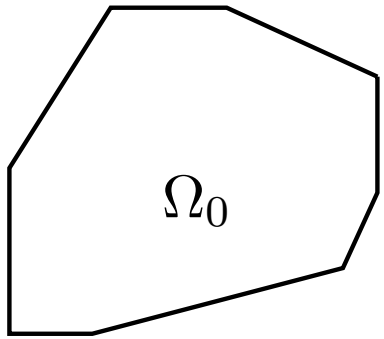
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:

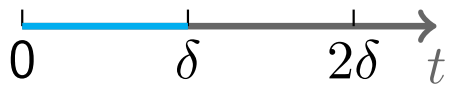




Linear hybrid automata II: Time evolution

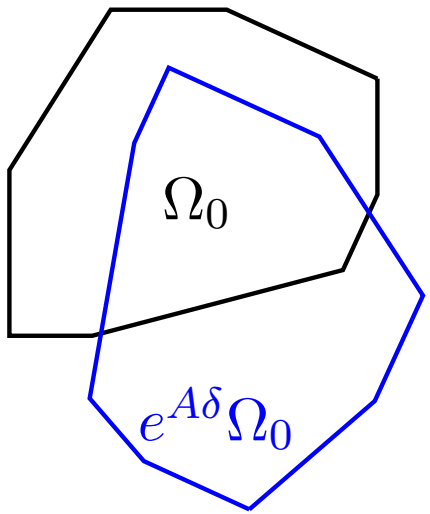
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:
- The remaining ones:

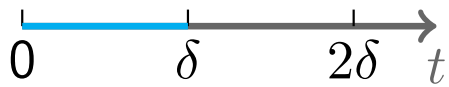




Linear hybrid automata II: Time evolution

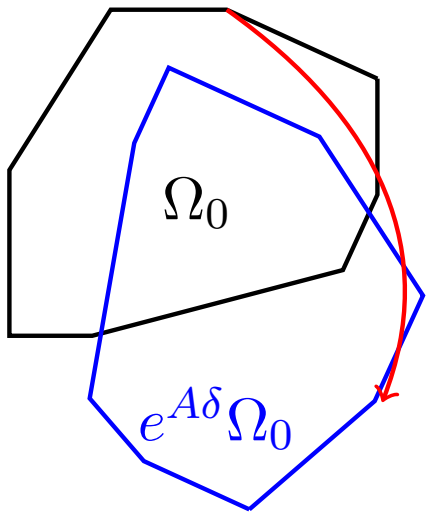
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:
- The remaining ones:

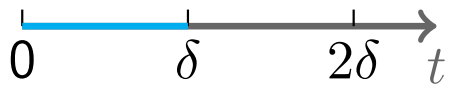




Linear hybrid automata II: Time evolution

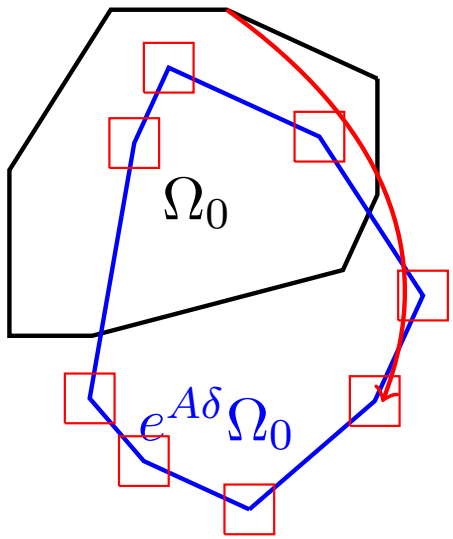
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:
- The remaining ones:

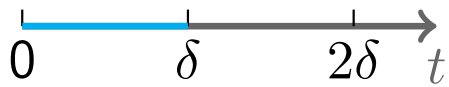




Linear hybrid automata II: Time evolution

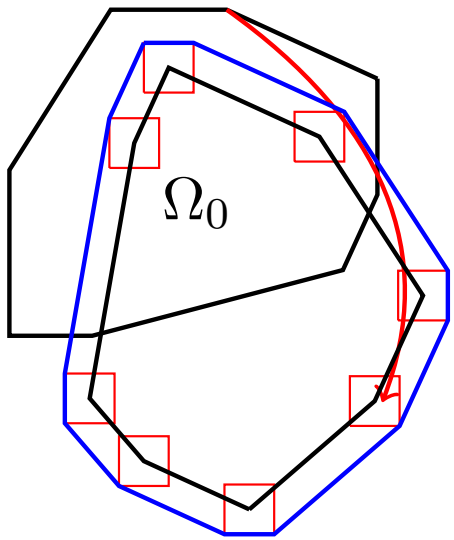
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:
- The remaining ones:

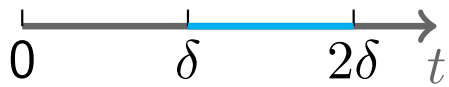




Linear hybrid automata II: Time evolution

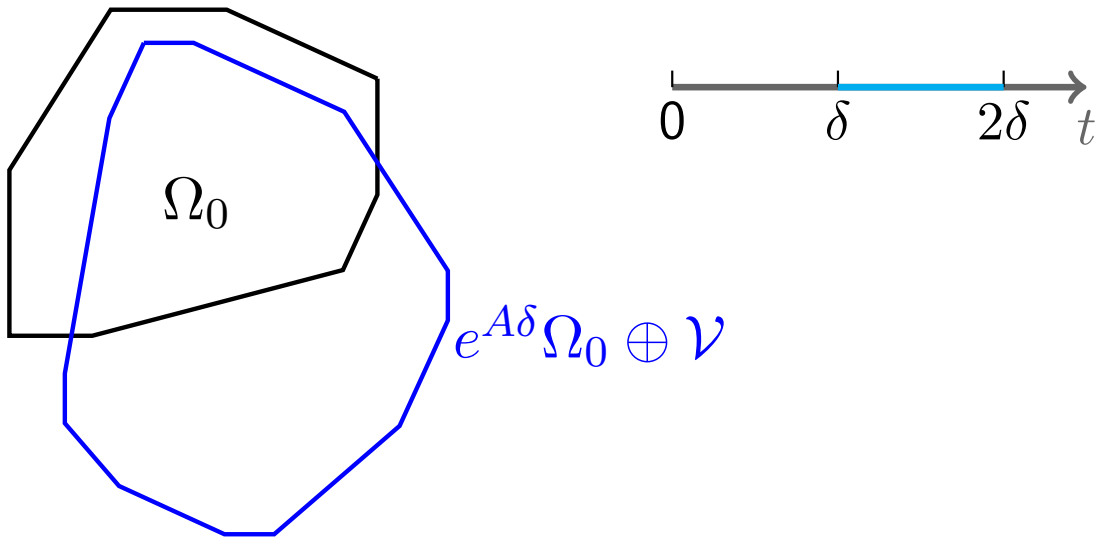
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:
- The remaining ones:



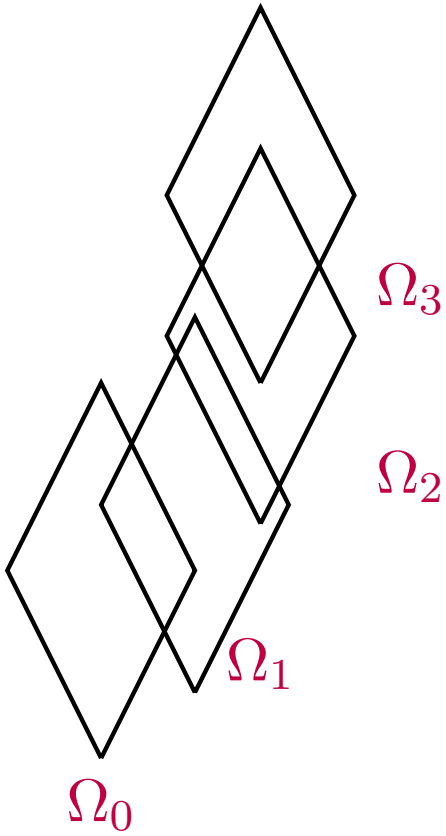


Linear hybrid automata II: Time evolution

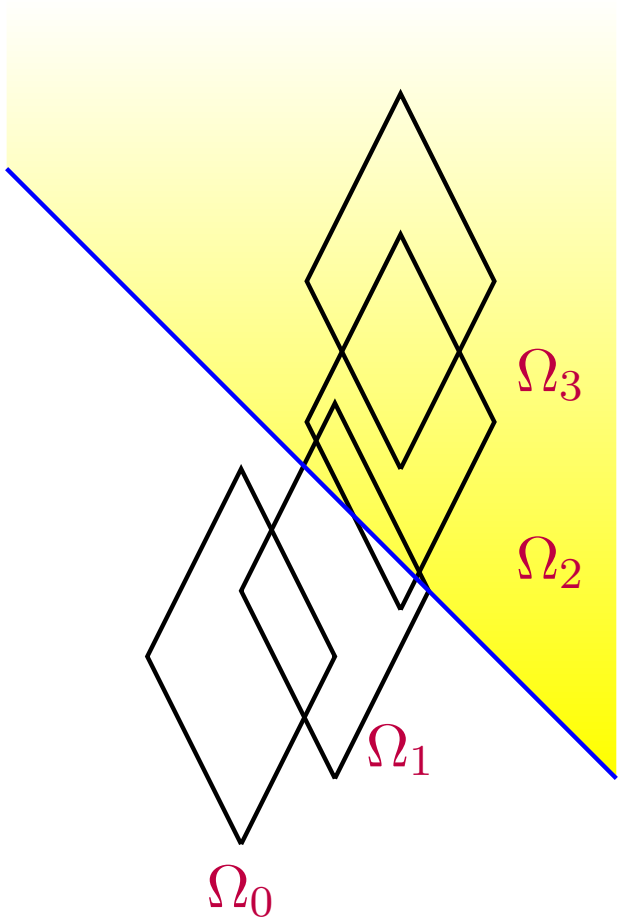
- Assume $\dot{x} = Ax + Bu$
- Compute $\Omega_0, \Omega_1, \ldots$ such that $\mathcal{R}_{[i\delta,(i+1)\delta]} \subseteq \Omega_i$
- The first flowpipe segment:
- The remaining ones:



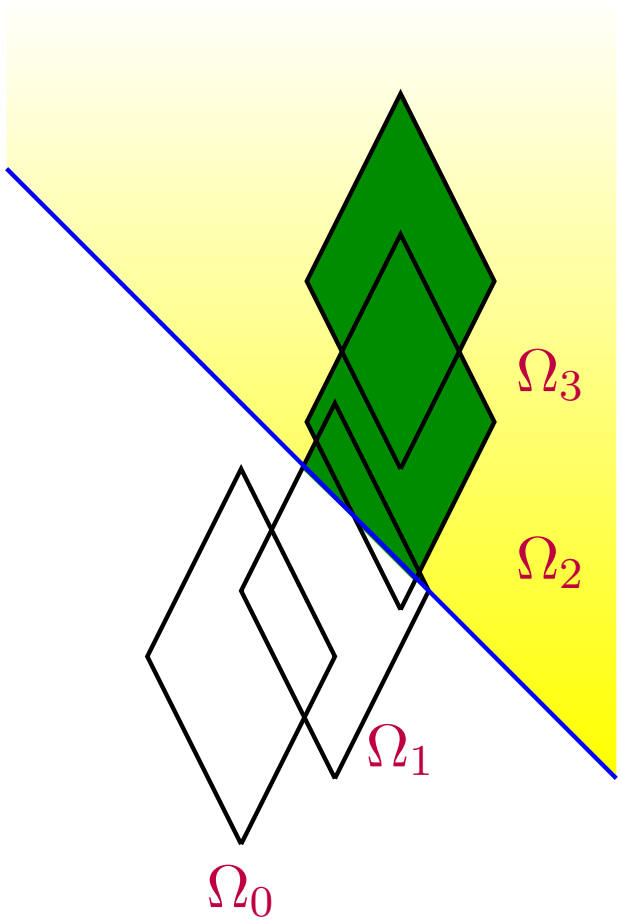
Linear hybrid automata II: Discrete steps (jumps)



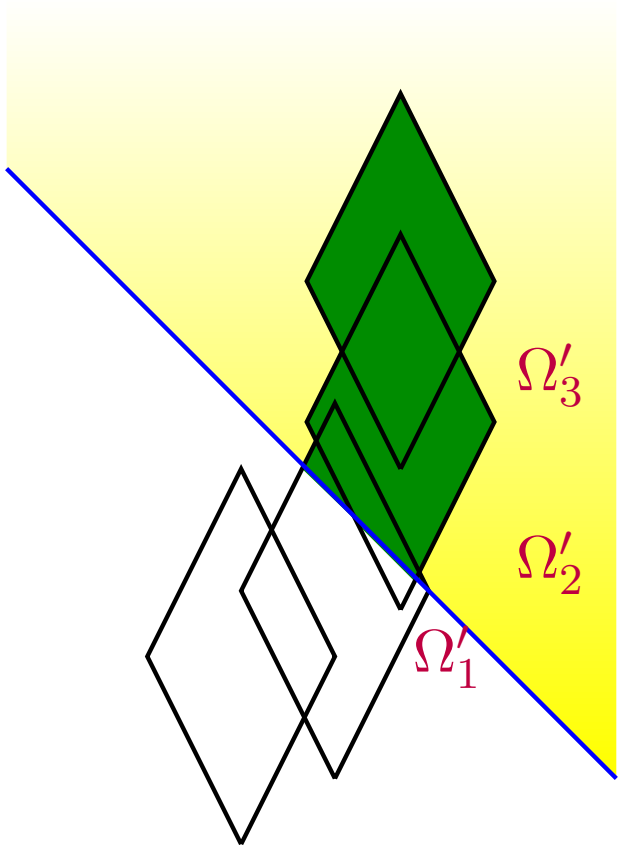
Linear hybrid automata II: Discrete steps (jumps)



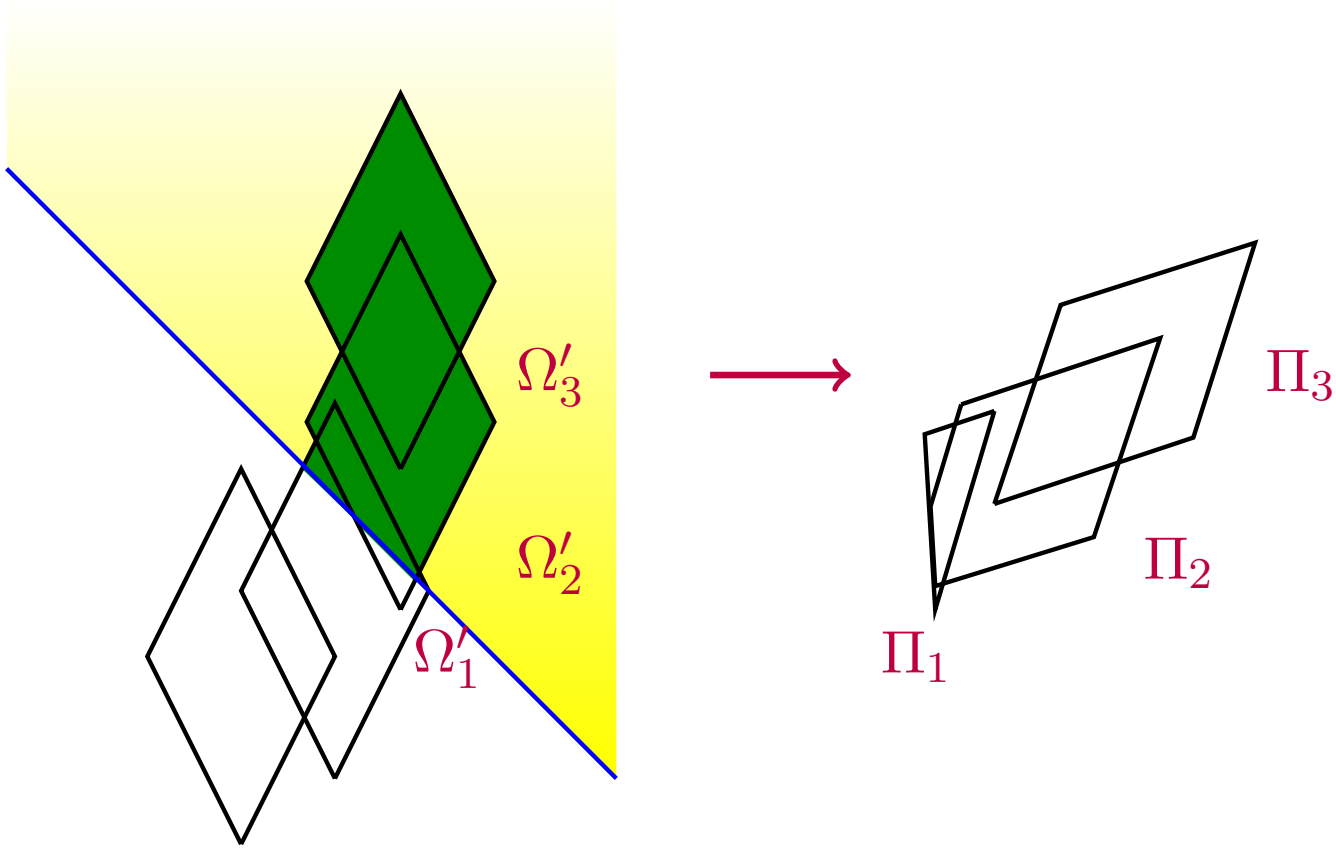
Linear hybrid automata II: Discrete steps (jumps)



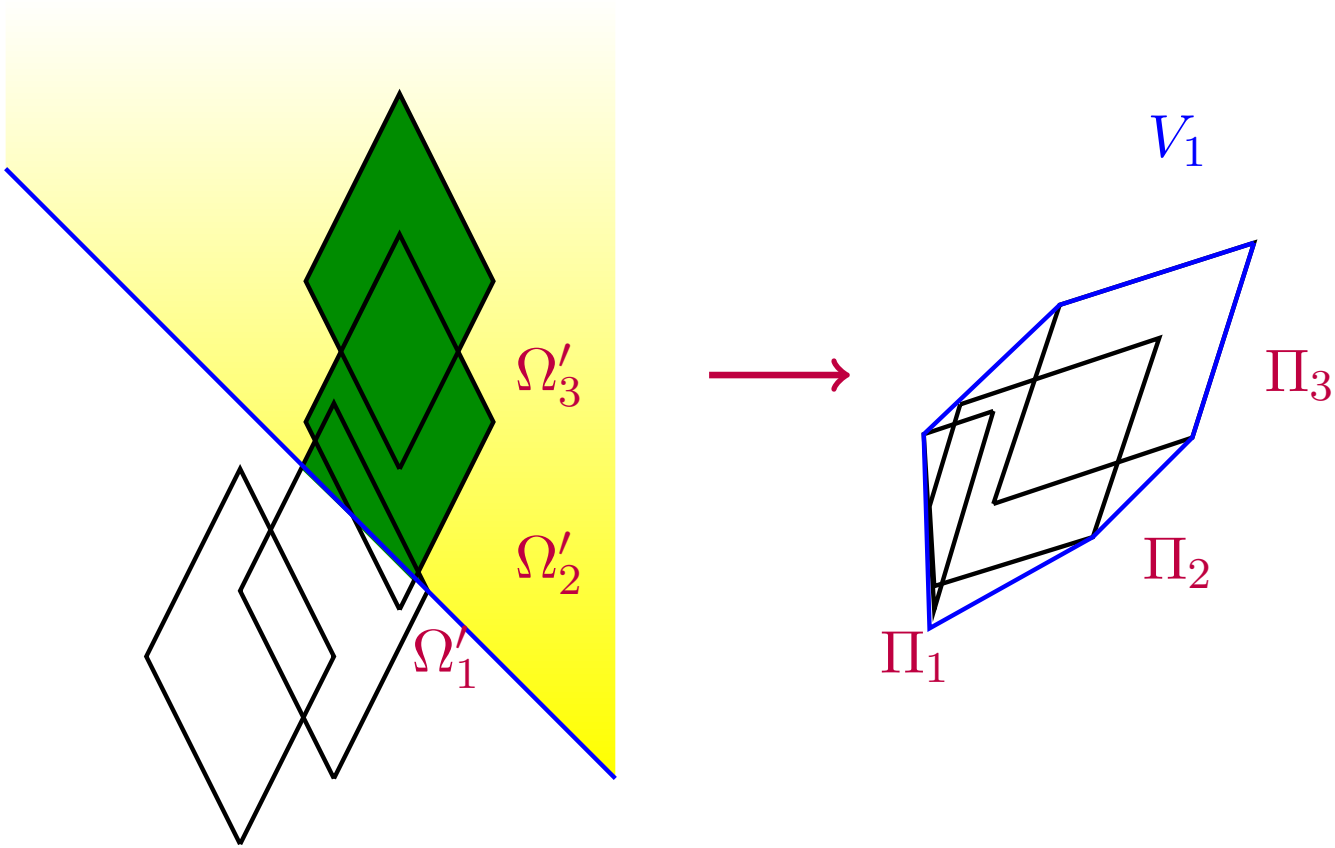
Linear hybrid automata II: Discrete steps (jumps)



Linear hybrid automata II: Discrete steps (jumps)



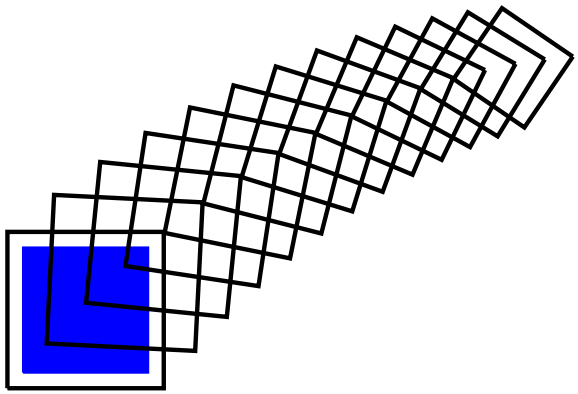
Linear hybrid automata II: Discrete steps (jumps)



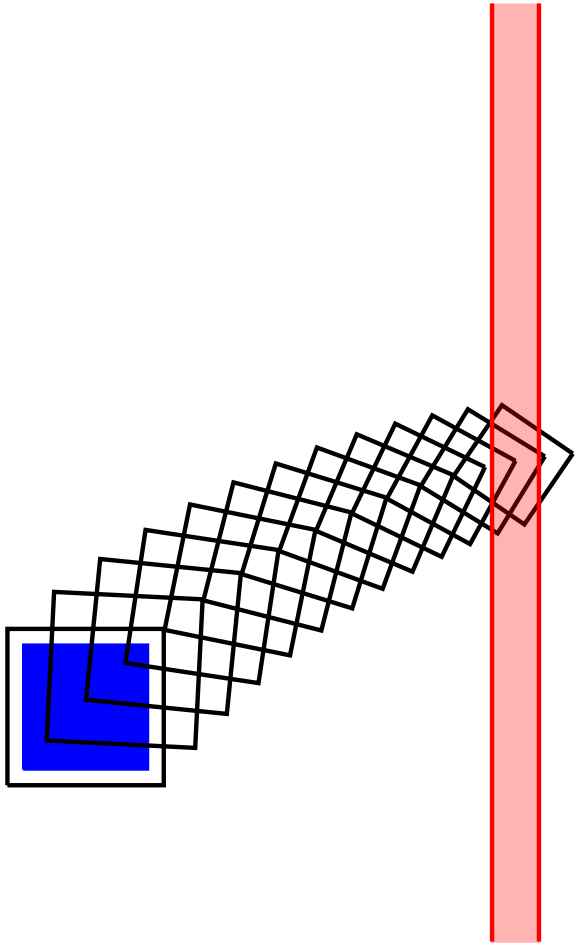
Linear hybrid automata II: The global picture

Linear hybrid automata II: The global picture

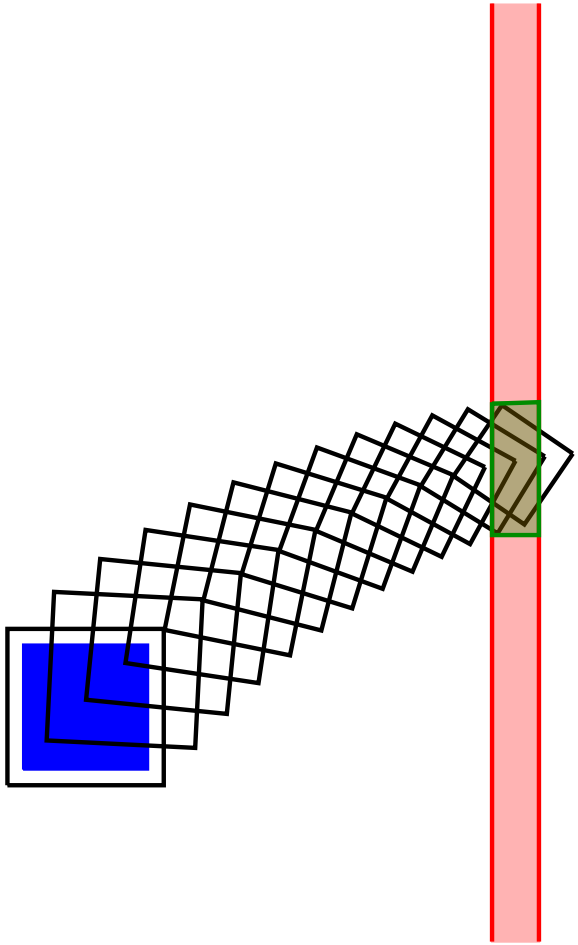
Linear hybrid automata II: The global picture



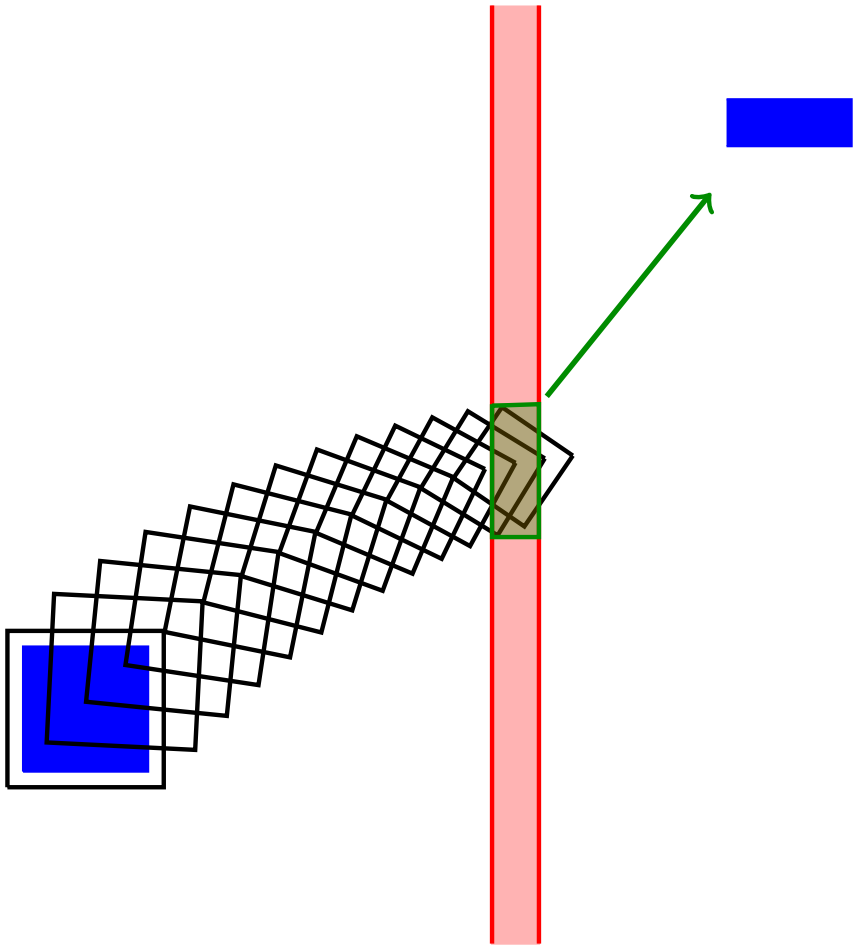
Linear hybrid automata II: The global picture



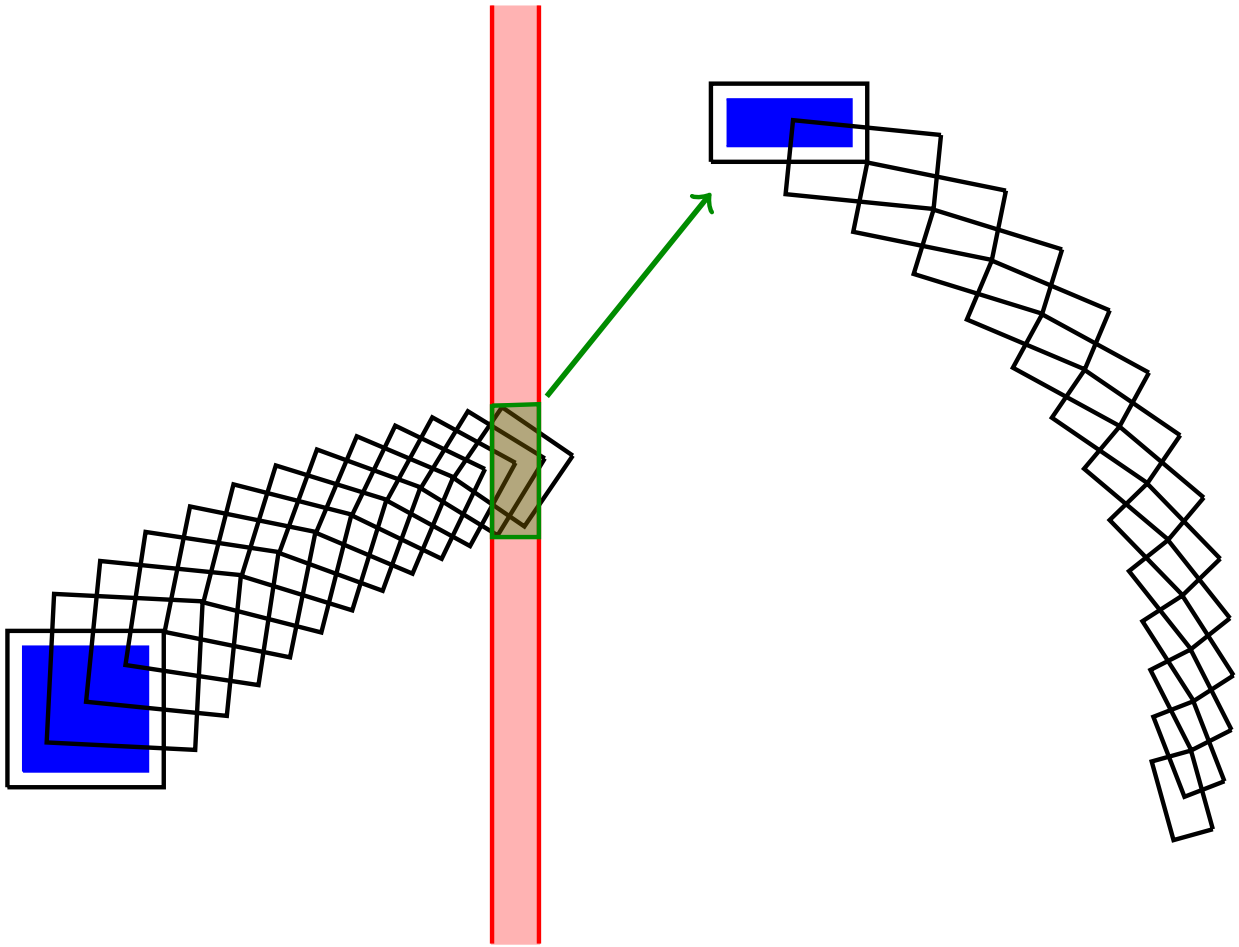
Linear hybrid automata II: The global picture



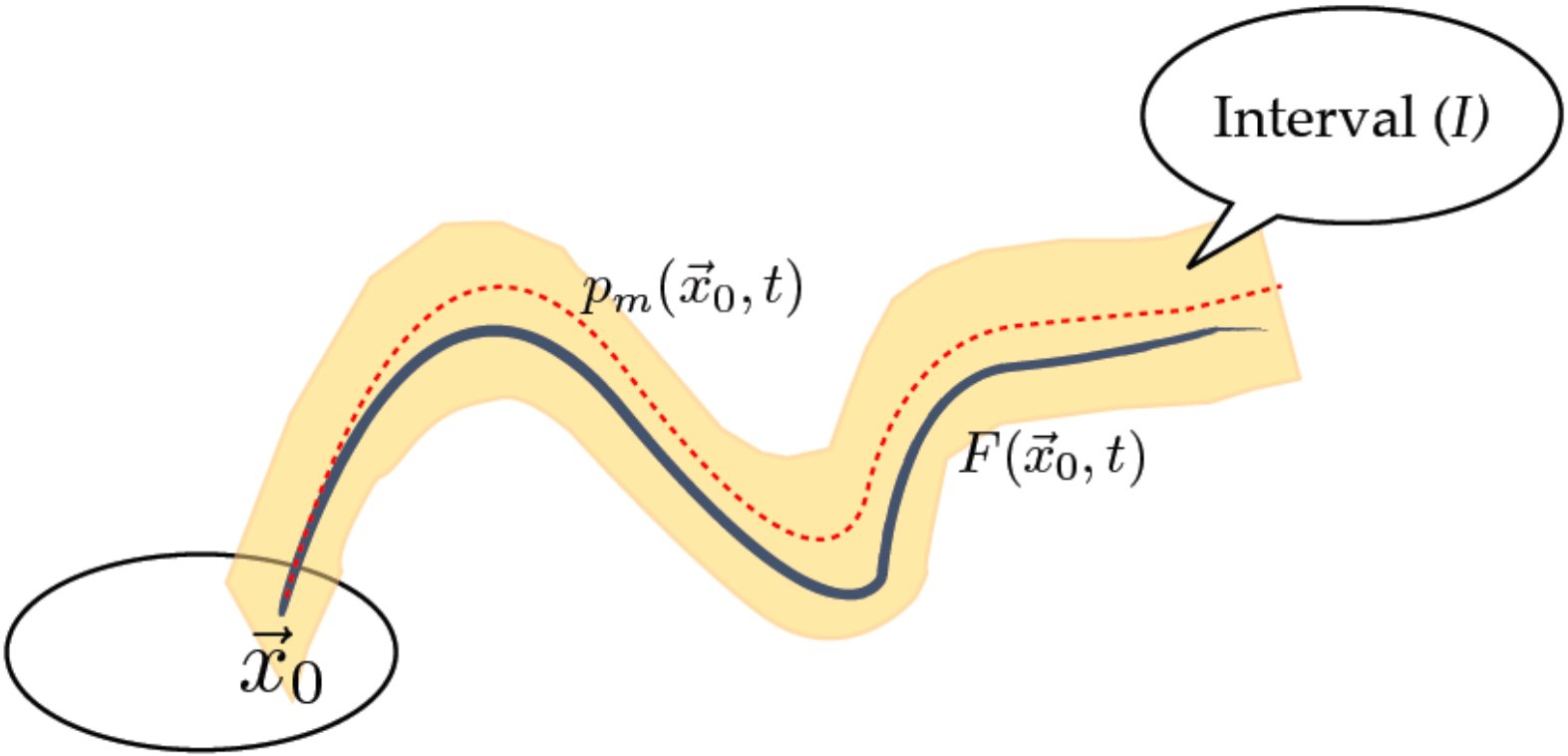
Linear hybrid automata II: The global picture



Linear hybrid automata II: The global picture



Our contribution: Taylor model representation of state sets



Fetch the tool

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

Flow*: Taylor Model-Based Analyzer for Hybrid Systems

What is Flow*?

Flow* is a tool which computes *Taylor model flowpipes* for a given continuous or hybrid systems. The current version of Flow* is able to handle hybrid systems with

- continuous dynamics defined by polynomial ordinary differential equations (ODEs),
- mode invariants and jump guards defined by conjunctions of polynomial constraints,
- jump resets defined by polynomial mappings.

What are flowpipes?

There are various definitions on *flowpipes*. Here, a flowpipe means an over-approximation of the reachable states in a time interval (or step).

Why Taylor models?

A Taylor model is the set defined by a polynomial (over an interval domain) bloated by an interval. The flow of a continuous system can be tightly enclosed by Taylor models. With proper interval-based techniques, we may construct Taylor model flowpipes for non-linear hybrid systems.

How to use Flow*?

A user manual can be found here.

Source code

The source code is released under the GNU General Public License (GPL). We are happy to release the code under a license that is more (or less) permissive upon request. source code

Some case studies on Flow* is available now. link

Publications

- Xin Chen, Erika Abraham and Sriram Sankaranarayanan. Flow*: An Analyzer for Non-Linear Hybrid Systems. Computer Aided Verification (CAV), 2013.
- Xin Chen, Erika Abraham and Sriram Sankaranarayanan. Taylor Model Flowpipe Construction for Non-linear Hybrid Systems. IEEE Real-Time Systems Symposium (RTSS), 2012.
- Yan Zhang, Xin Chen, Erika Abraham and Sriram Sankaranarayanan. Empirical Taylor Model Flowpipe Construction for Analog Circuits (Abstract). Frontiers of Analog Computation Workshop, 2013. (slides will be posted soon).

People

Constructing Flowpipes for Continuous and Hybrid Systems: Case-Studies.

Introduction

We present a set of benchmarks of continuous and hybrid systems as long as their running results on the tool Flow*. These studies are intended to benchmark the performance of Flow* tool and serve as a basis of comparison with other tools.

All these studies are run on the following computational platform.

CPU: Intel Core i7-860 Processor (2.80 GHz)
Memory: 4096 MB
System: Ubuntu 12.04 LTS

Continuous-Time Case Studies

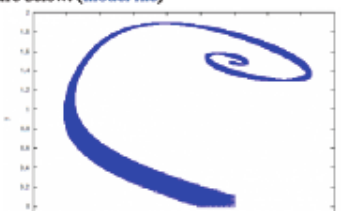
(A) Brusselator

The Brusselator system is a "chemical oscillator" (see here for more details).

The dynamics of a Brusselator are given by

$$\begin{cases} \dot{x} &= A + x^2 \cdot y - B \cdot x - x \\ \dot{y} &= B \cdot x - x^2 \cdot y \end{cases}$$

wherein A=1 and B=1.5 in our tests. We let Flow* compute the Taylor model flowpipes for the time horizon **[0,15]**. We first choose the initial set **x in [0.9,1]** and **y in [0,0.1]**, Flow* costs **7 seconds** to generate the flowpipes shown in the figure below. (model file)



Contents

1 Modeling

2 Reachability analysis

3 Counterexample generation

Counterexamples for hybrid automata

- Only a few approaches are available, mostly for rectangular automata
- Possible techniques?

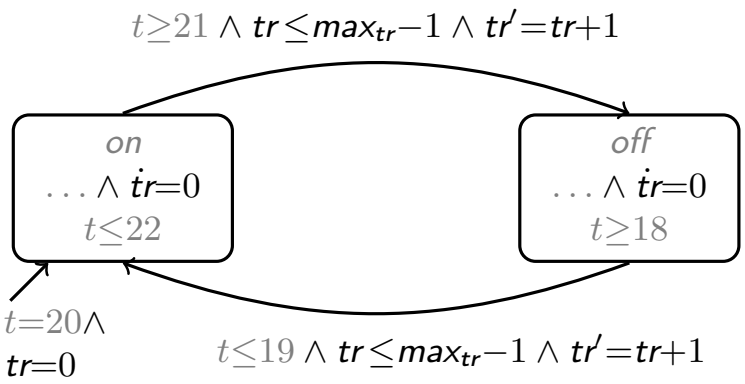
Simulation-based approach

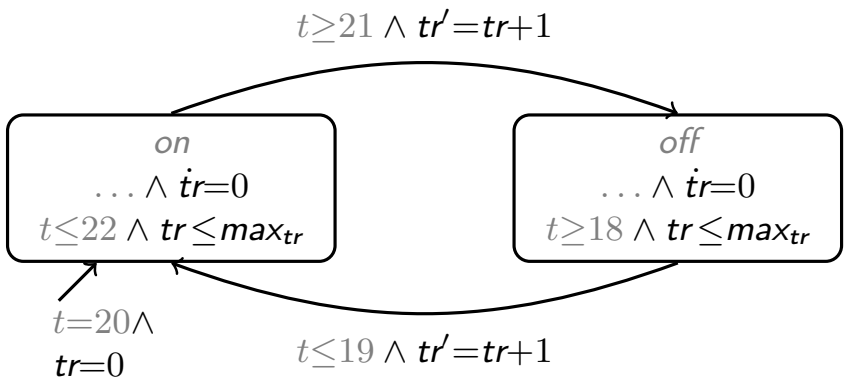
- Identify timed traces as possible counterexamples

$$l_0 \xrightarrow{[t_0,t_0'],e_0} l_1 \xrightarrow{[t_1,t_1'],e_1} \dots$$

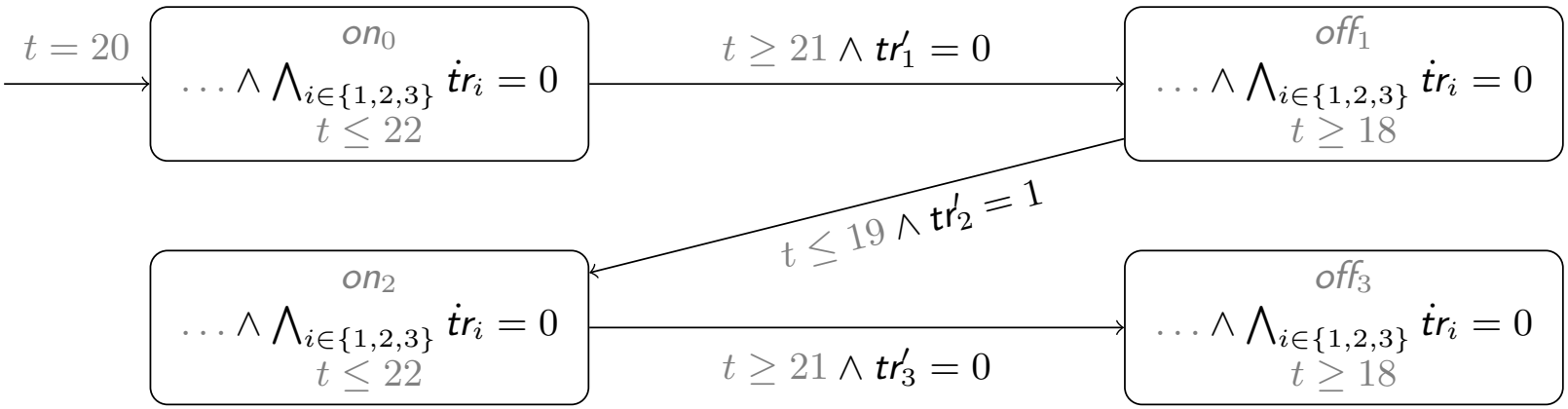
- Validate them using simulation

Extracting the path length

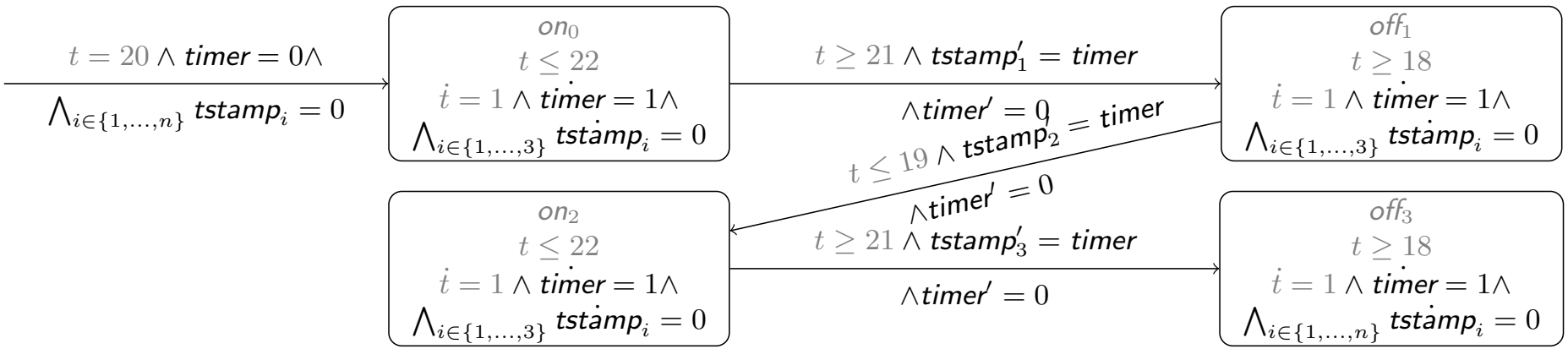




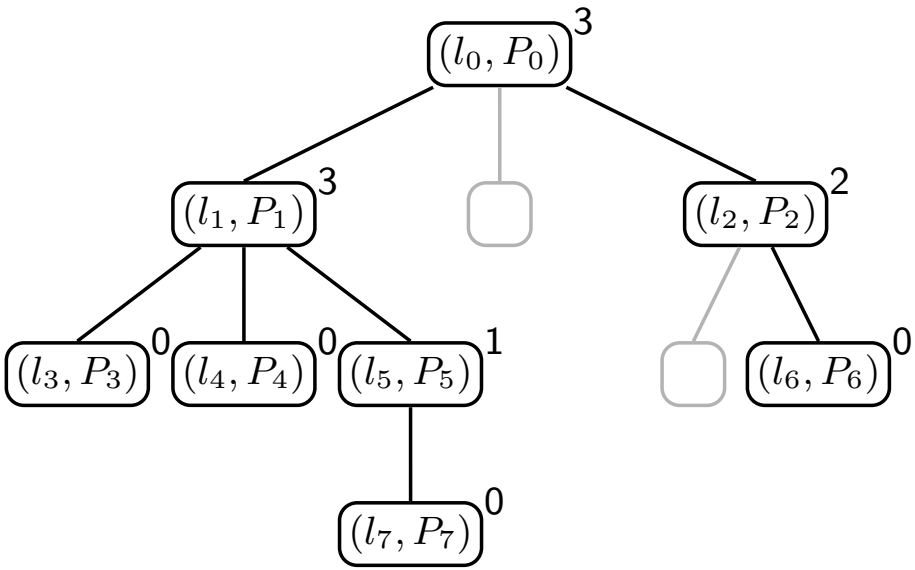
Naive trace encoding



Getting time information

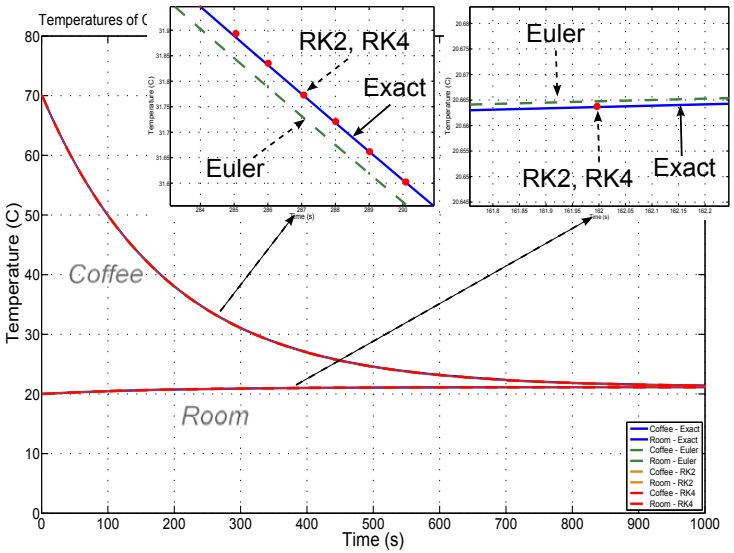


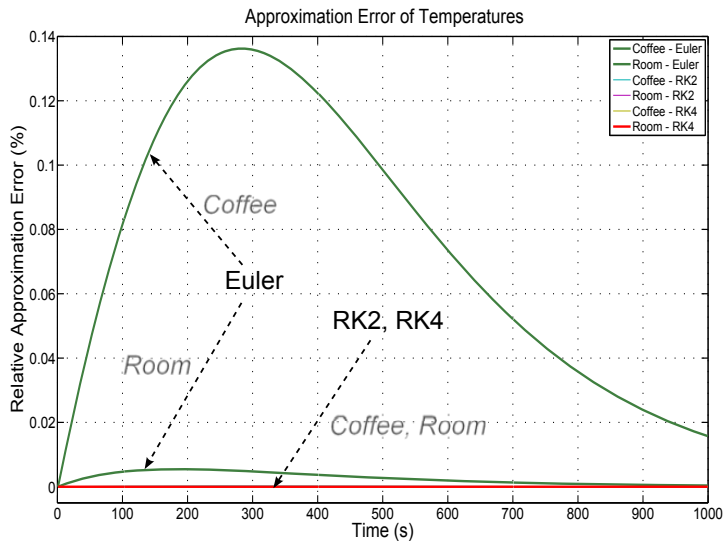
The search tree



Simulation-based validation

- Domain contraction
- Linear ODEs: time successor computations closed to exact
- Non-liner ODEs: numerical methods
- Problems: invariants, hit equality guards
- Search heuristics
- Validity metrics





Under-approximative reachability analysis

Provably correct counterexamples using Ariadne