Controller Synthesis for Linear Hybrid Systems

SynCoP and PV
April 14th, 2018



Marco Faella
Università di Napoli “Federico II”, Italy

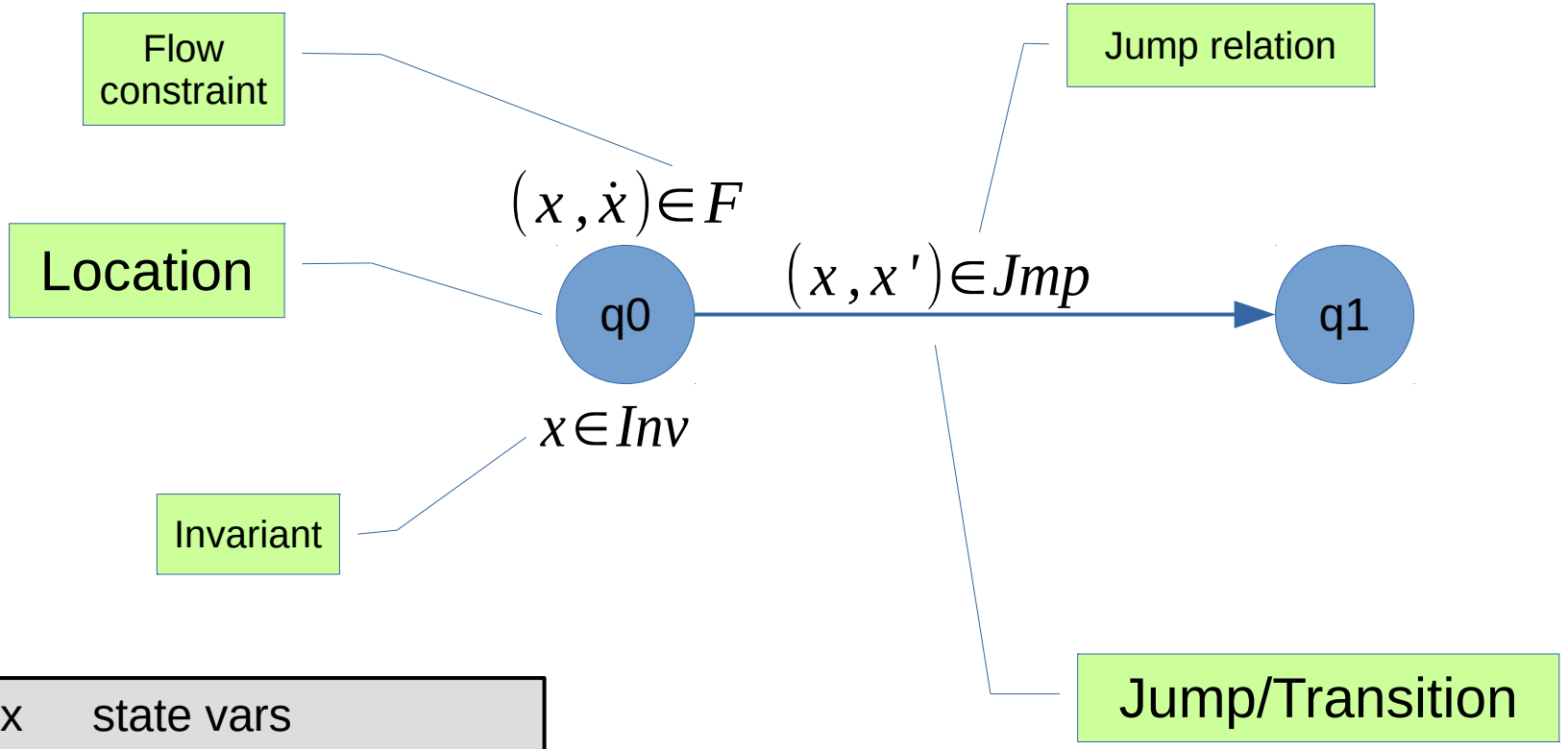
Summary

- Hybrid systems
- Controller synthesis
  - safety control
  - reachability control
- Tool demo

Models of Hybrid Systems

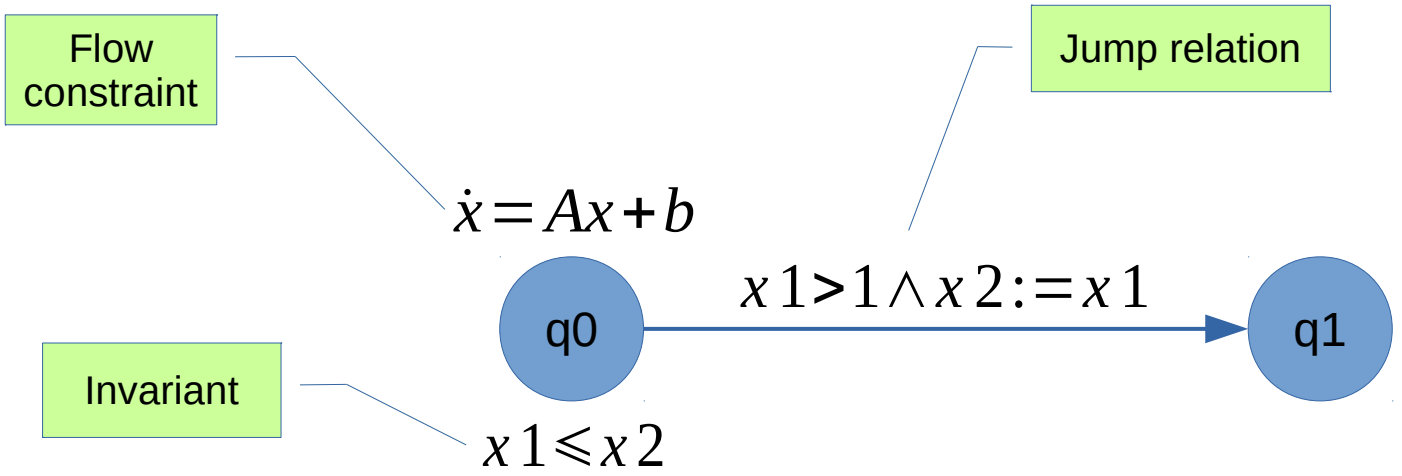
- In control engineering:
  - Switched systems
  - Piecewise affine systems
- In computer science:
  - **Hybrid Automata**
  - generalize Timed Automata by allowing *more general dynamics*

Hybrid Automata



x	state vars
ẋ	first derivatives
x’	state vars after jump

A Hybrid Automaton



x	state vars
ẋ	first derivatives
x’	state vars after jump

Runs

- Semantics based on *runs*
- Sequences of alternating **timed steps** and **discrete steps**
- Timed step:
  - a continuous-time **trajectory** in $\mathbb{R}^n$, which satisfies the invariant and the flow constraint of the current location
  - location remains fixed
- Discrete step (i.e., jump):
  - an **instantaneous change of location**
  - variables change according to jump relation

What Kind of Flow Constraints?

1. Affine Hybrid Automaton: *linear* diff. eq.

- $\dot{x} = A\,x + B\,d$
- A and B are constant matrices and $d \in$ D are *disturbances*

2. Linear (sic!) Hybrid Automaton (LHA): *polyhedral* diff. inclusions

- $\dot{x} \in F$
- where *F* is a convex polyhedron
- i.e., *a set of linear constraints on the derivatives*
- special case of affine, with A=0 (no dependency on the current state)

3. Rectangular Hybrid Automaton: rectangular diff. incl.

- as above, but *F* is a hyper-rectangle (Cartesian product of intervals)

Verification Problems

- We seek to solve the following:
  - **Forward reachability problem (FRP)**: given an effective set of *initial states*, compute the set of states that are reachable from them
  - **Backward reachability problem (BRP)**: given an effective set of *error states*, compute the set of states that can reach them

- BRP and FRP are inter-reducible provided the model supports inversion of time (LHAs do)
- In this talk, effective = polyhedral

Decidability

- Decidability requires:
  - very simple dynamics (initialized rectangular), or
  - very simple transitions (o-minimal)

- LHAs are undecidable

Algorithms

Two approaches:

- Finite bisimulation quotient (a.k.a. *indirect* approach)
  - decidable models
- On-the-fly exploration of the state-space (*direct* approach)
  - decidable or undecidable models

The Direct Approach

- On-the-fly exploration of the state space

- Two sub-approaches:
  - Surrender exactness ⇨ approximate algorithms
  - Surrender termination ⇨ **exact semi-algorithms**

Exact Semi-Algorithms

- An algorithm that may or may not terminate
- When it terminates, it provides the exact answer

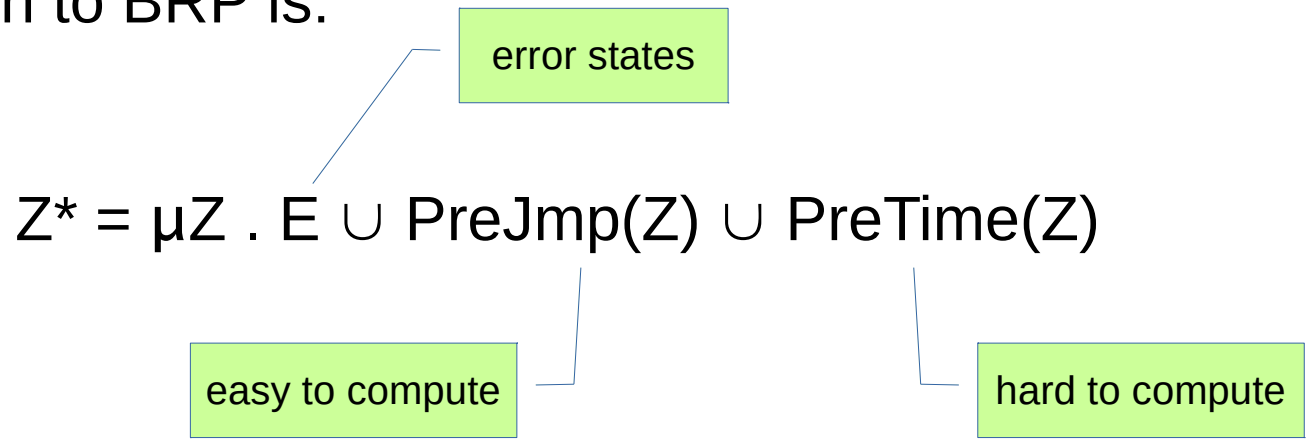
- It may be stopped after a deadline
- It provides the exact answer up to a **fixed number of discrete steps**
  - *bounded horizon reachability*

An Exact Semi-Algorithm for BRP on LHAs

- Given a polyhedral set of states E, simulate discrete steps and timed steps backwards (*symbolic execution*)
- For discrete steps:
  - **PreJmp**(E) = states that can reach E via a discrete step
- For timed steps:
  - **PreTime**(E) = states that can reach E via a timed step

An exact semi-algorithm for BRP on LHAs

- The solution to BRP is:



- **Fact**: When Z is polyhedral, **PreJmp** and **PreTime** can be effectively computed and their result is polyhedral

- However, the above sequence may not converge in a finite number of steps

Computing PreTime: The Reach-While-Avoiding Operator

Fix a location q and the corresponding flow constraint Flow(q)

Definition. Given two polyhedra U and V,

RWA(U, V) is the set of points from which there is a trajectory that:

- reaches U
- while avoiding V at all times.

- a.k.a.: *flow_avoid* in [Wong-Toi,97], *Reach* in [Tomlin et al.,00]
- in temporal logic (CTL): $\exists\, \overline{\mathbf{V}}$ ***Until*** **U**

PreTime and RWA

PreTime is a special case of RWA, i.e.:

$$\text{PreTime}(U) = \text{RWA}(U, \overline{\text{inv}(q)})$$

where U is a polyhedron in $\mathbb{R}^n$

Let's analyze the basic properties of RWA...

Properties of RWA

Assume w.l.o.g. that U and V are disjoint.

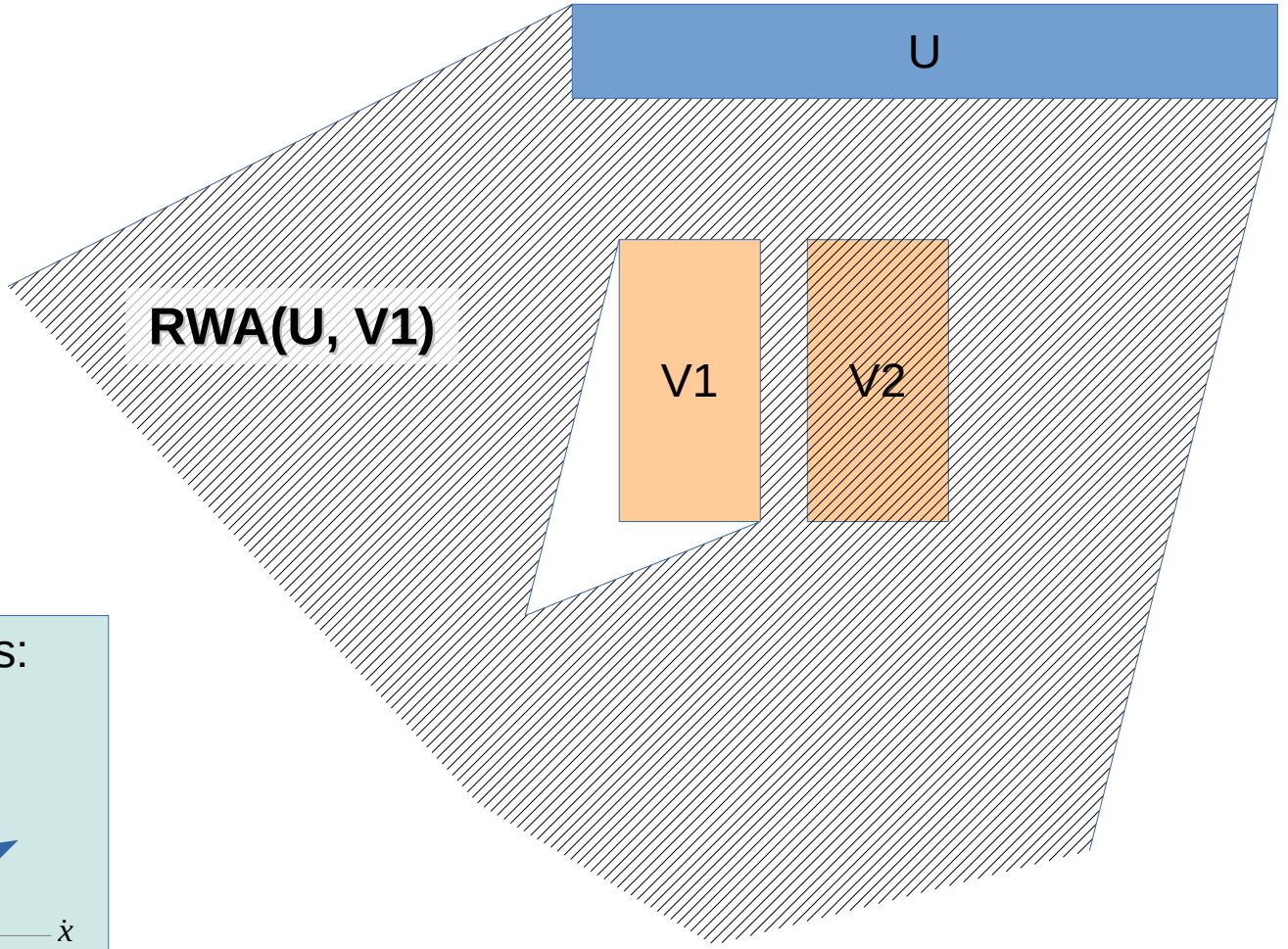
[*distributivity 1st arg*]

$$RWA(U1 \cup U2, V) = RWA(U1, V) \cup RWA(U2, V)$$

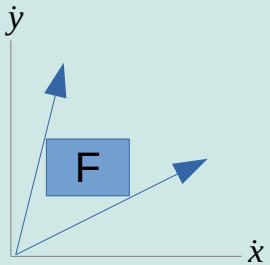
[*non-distrib. 2nd arg*]

$$RWA(U, V1 \cup V2) \supseteq RWA(U, V1) \cap RWA(U, V2)$$

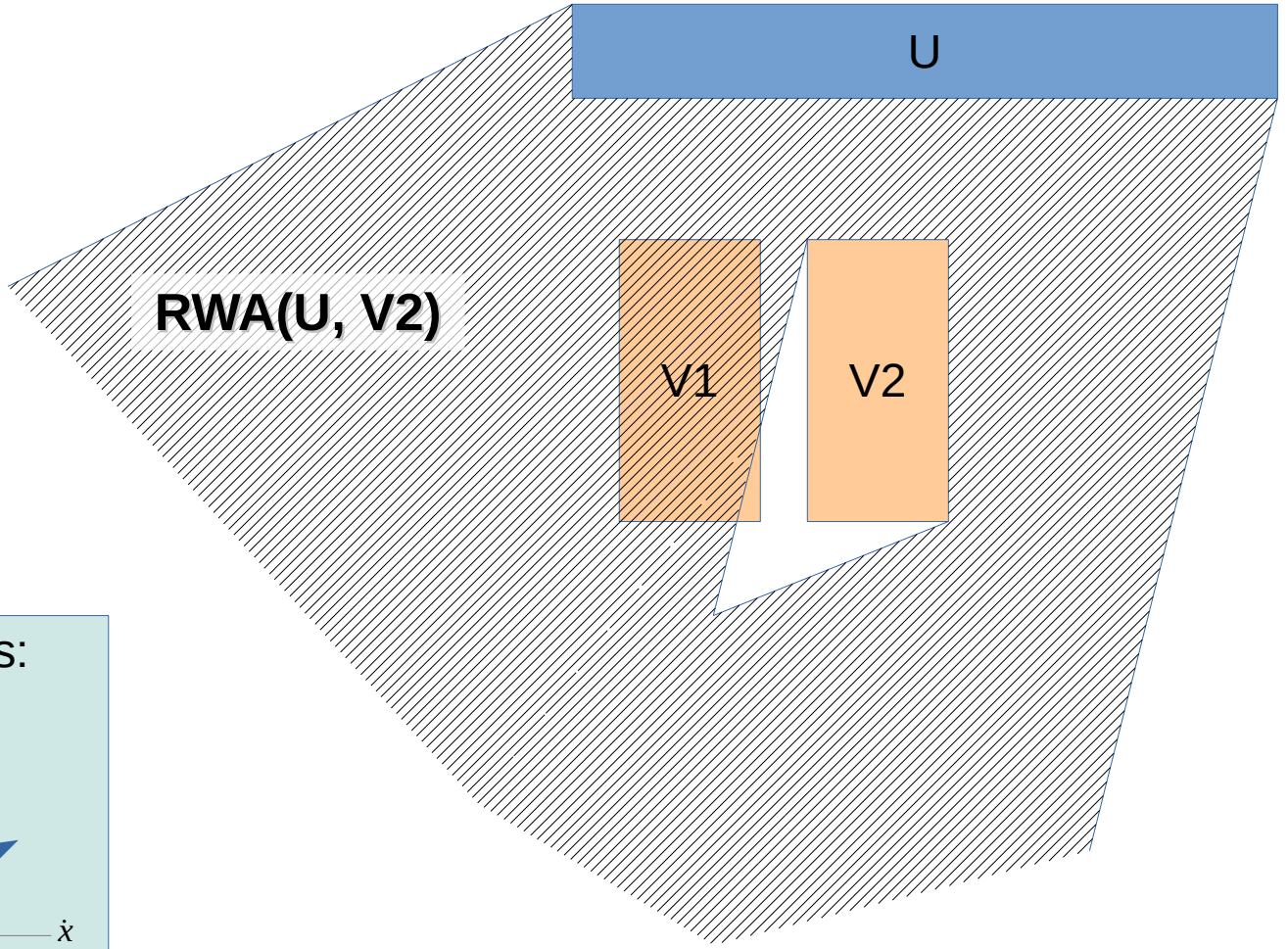
Non-distributivity



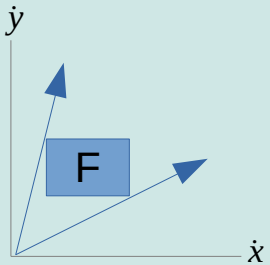
Dynamics:



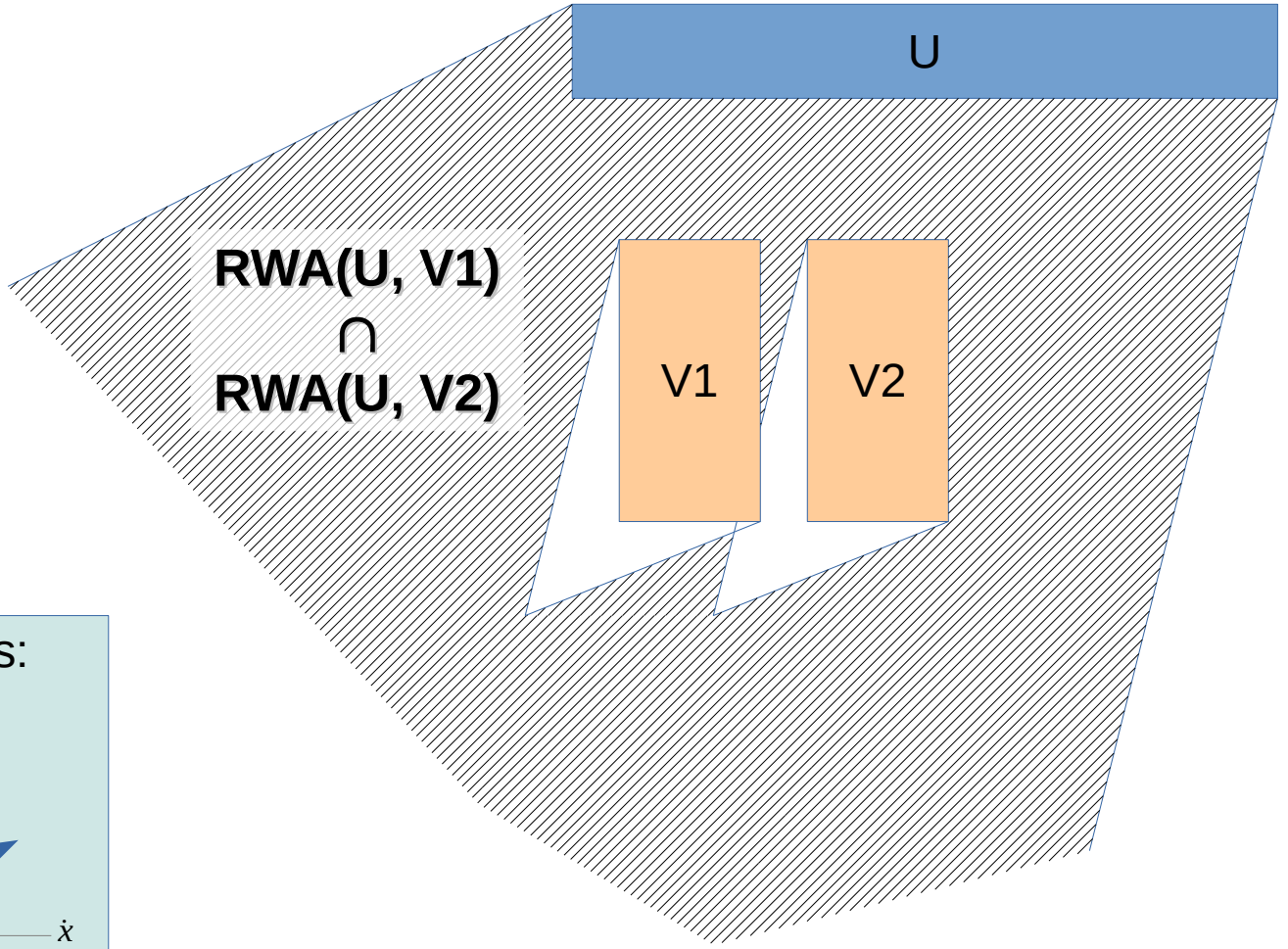
Non-distributivity

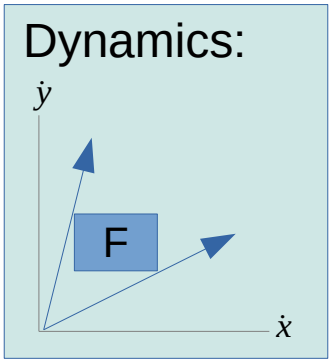


Dynamics:

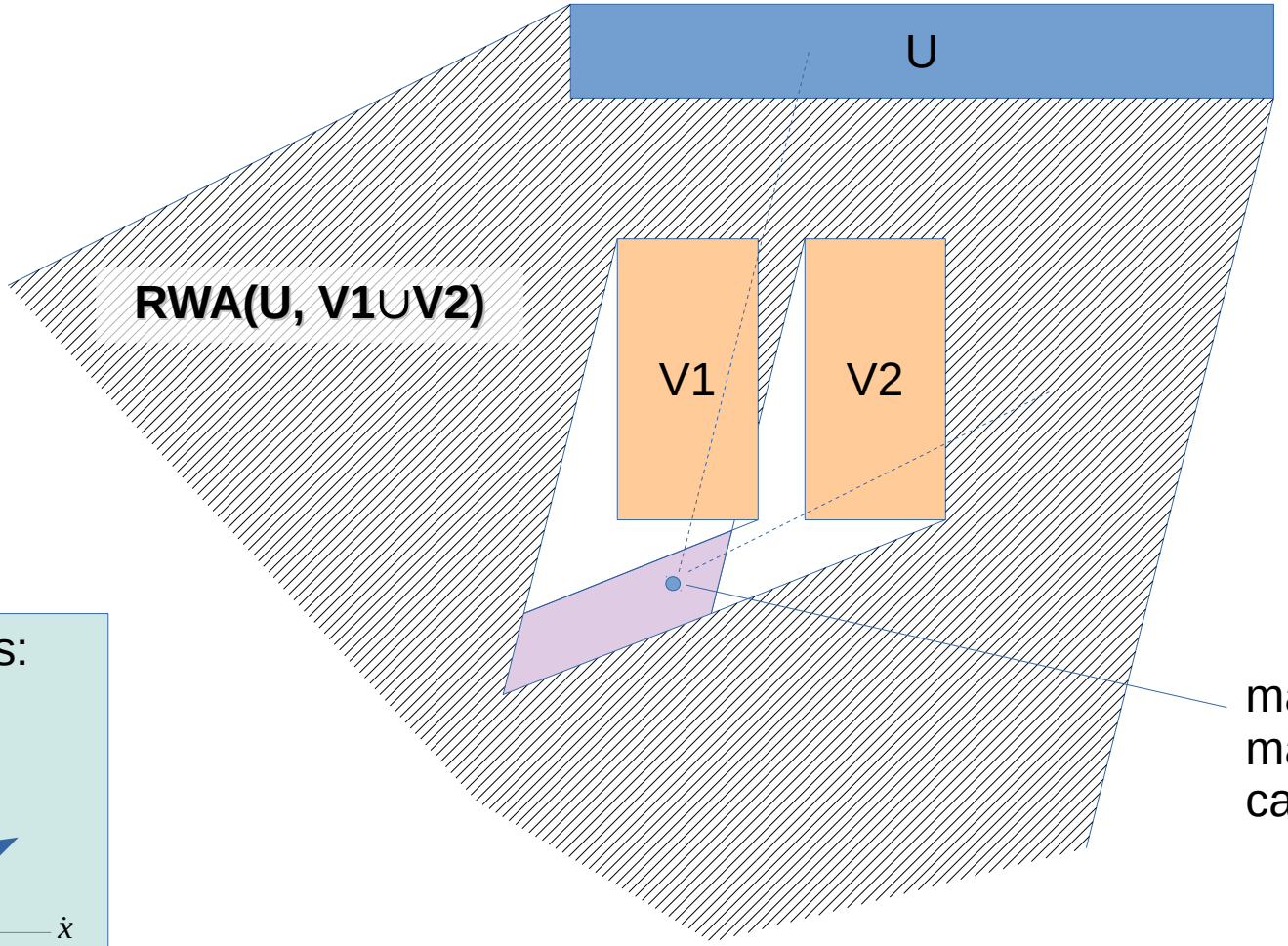


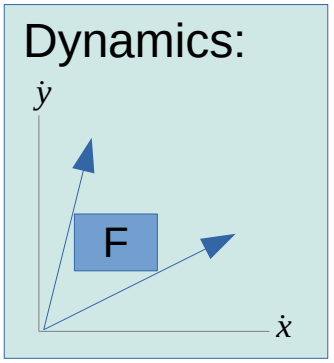
Non-distributivity





Non-distributivity

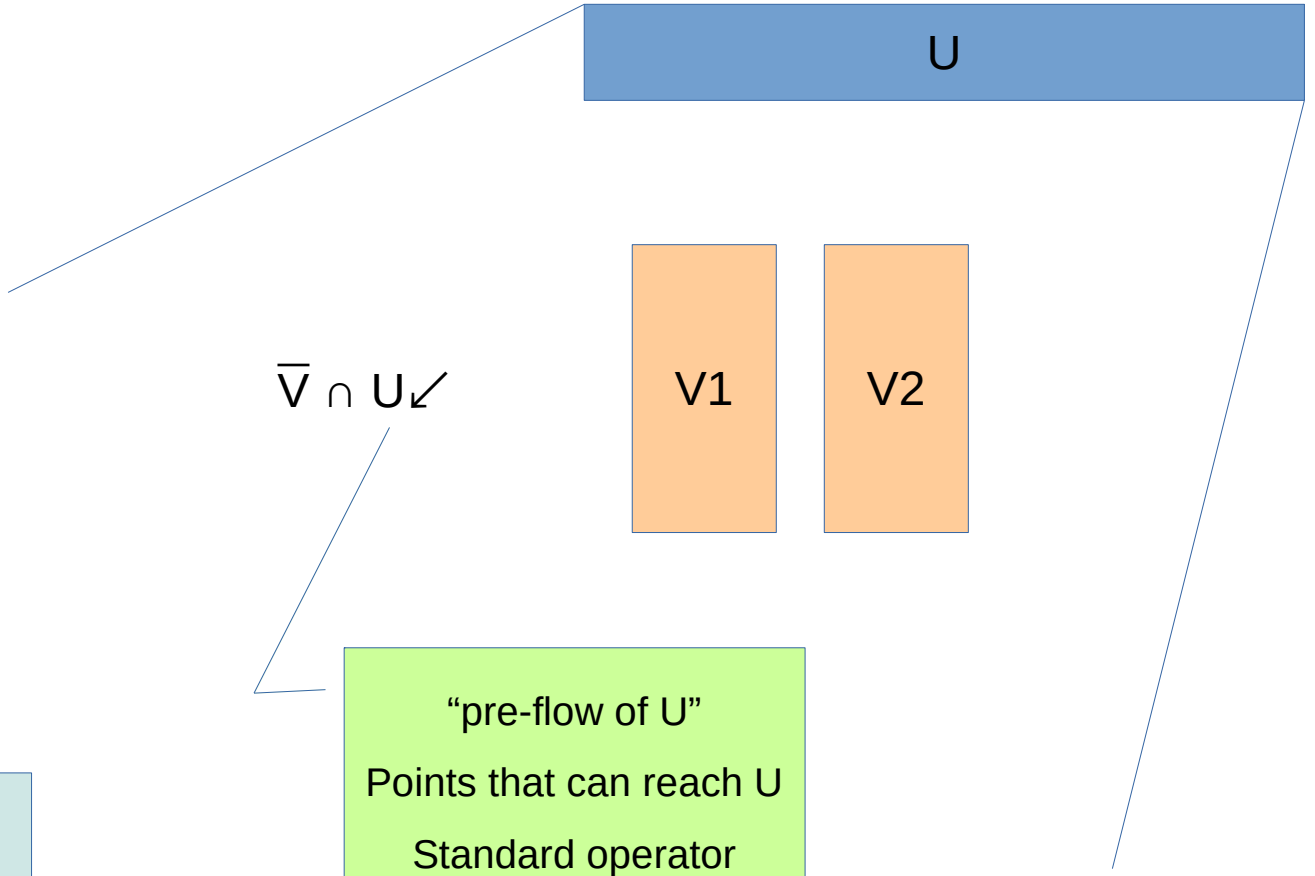


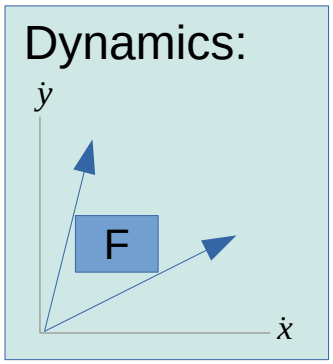


may avoid V1
may avoid V2
can't avoid V1 ∪ V2

Computing RWA(U,V)

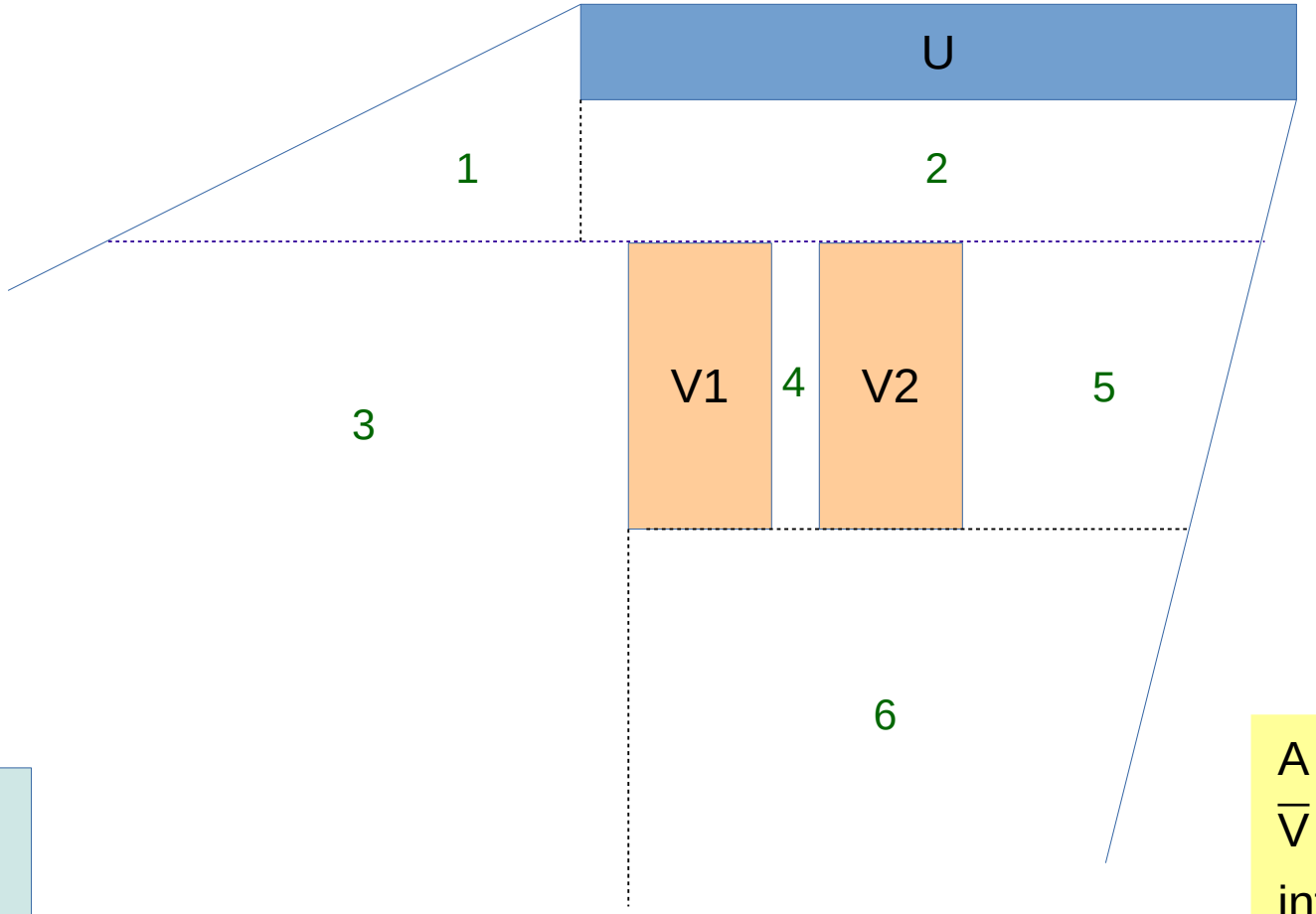
$V = V1 \cup V2$





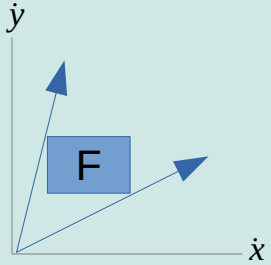
Computing RWA(U,V)

$V = V1 \cup V2$

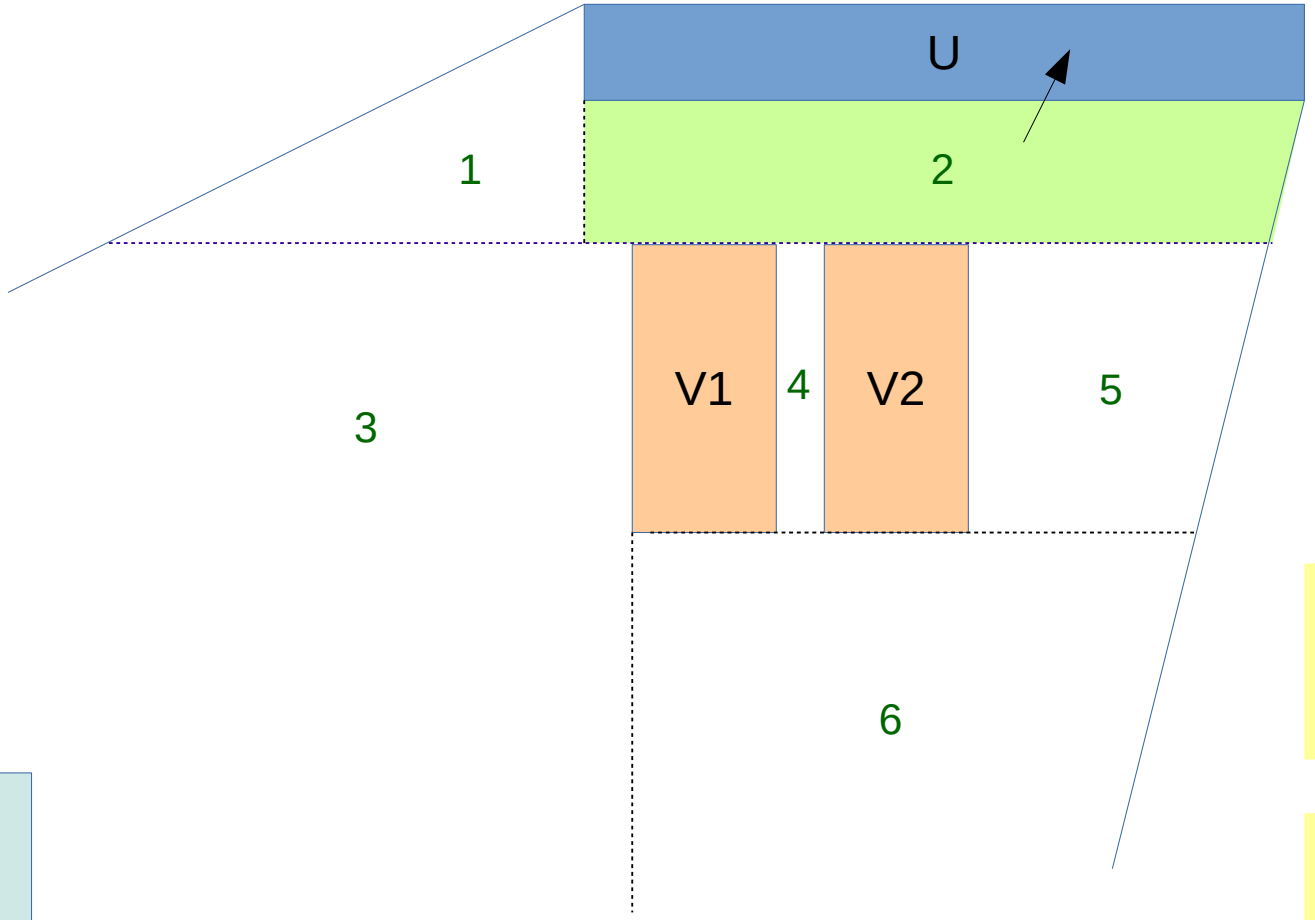


A partition of $\overline{V} \cap U\swarrow$ into 6 convex polyhedra

Dynamics:

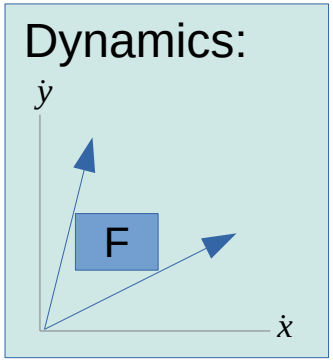


Computing RWA(U,V)

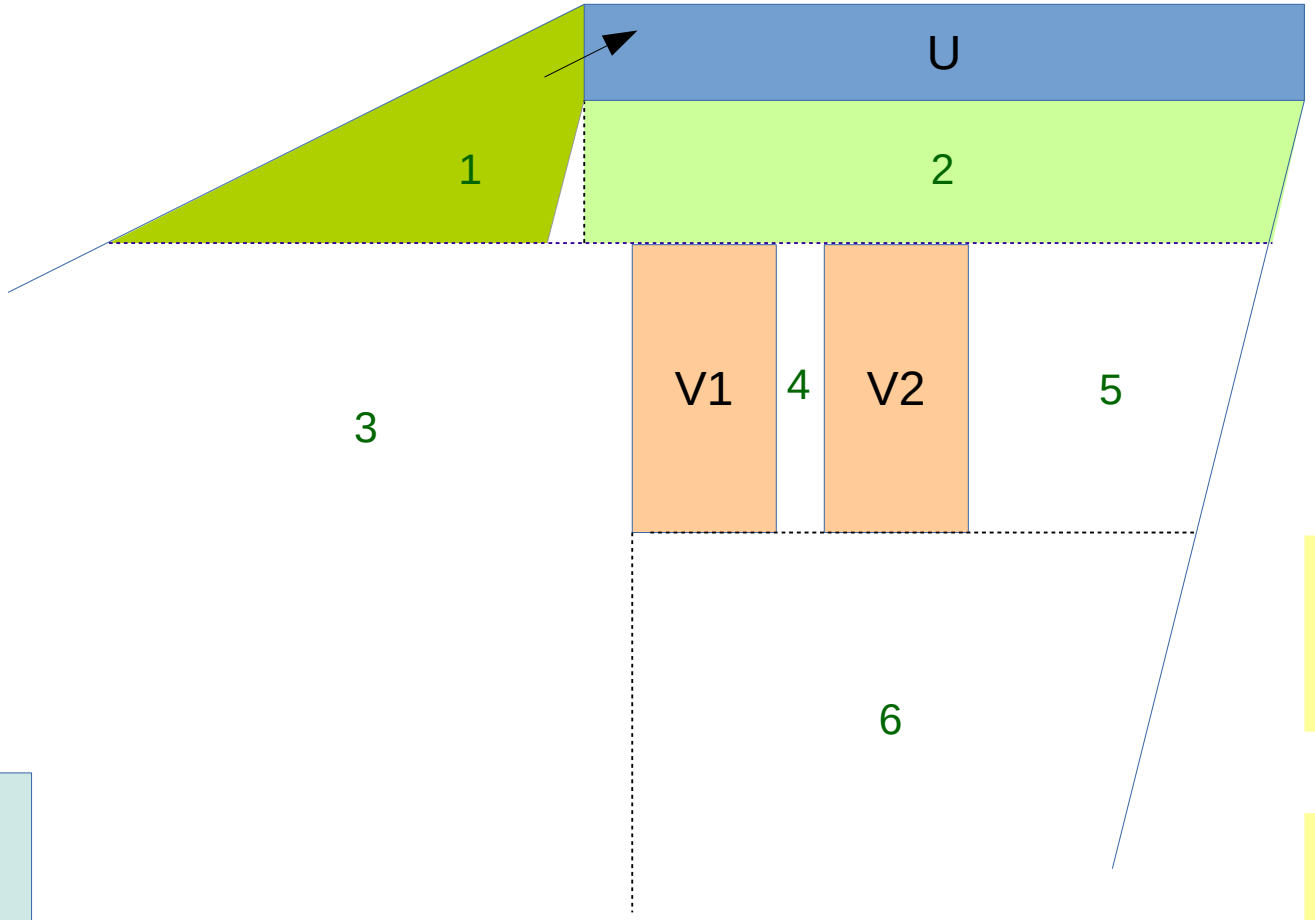


All points of P2 go directly into U

They are added to the result



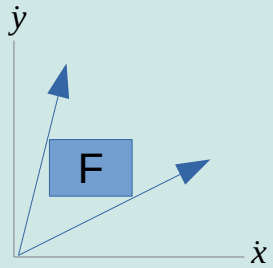
Computing RWA(U,V)



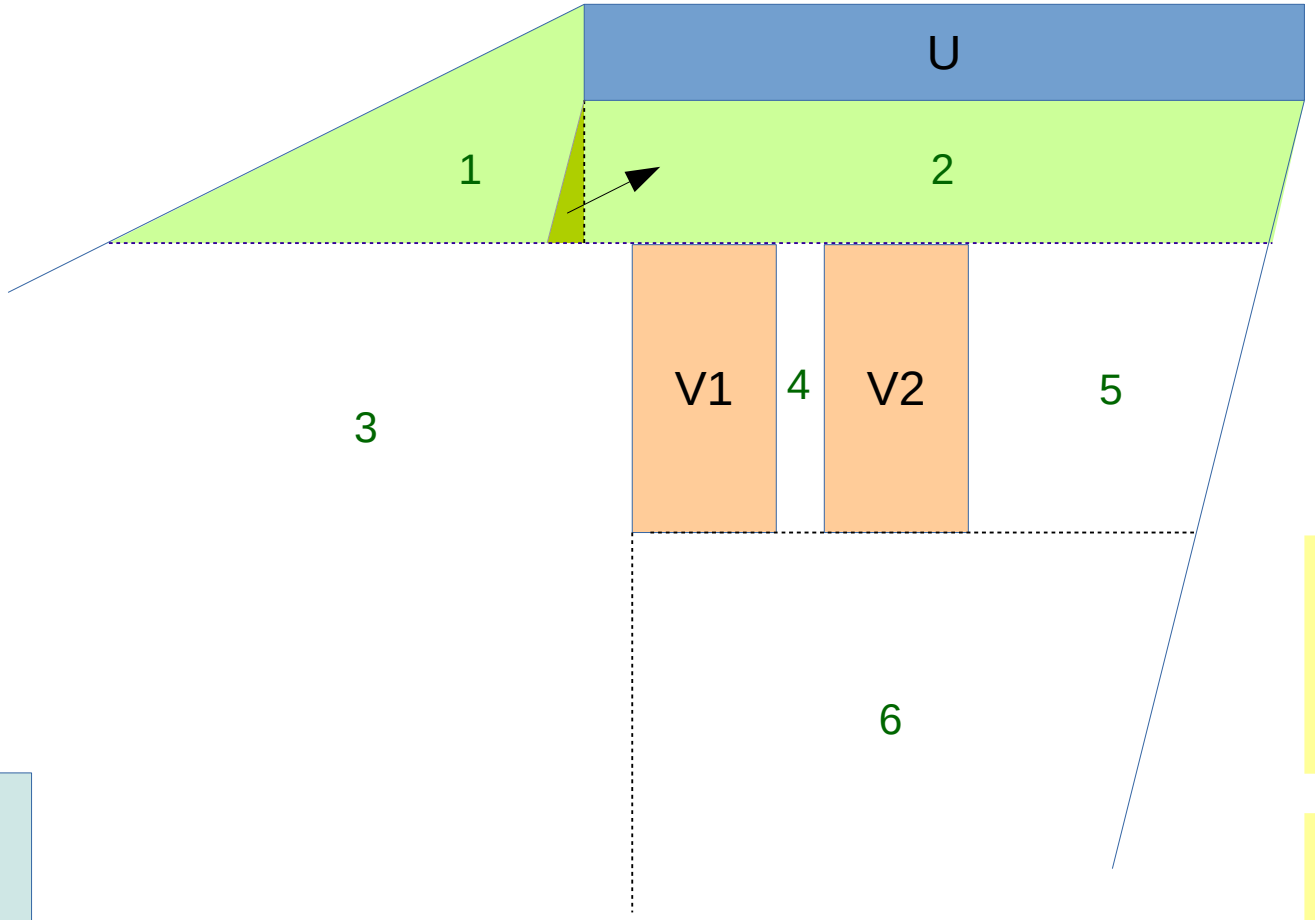
Some points of P1 go directly into U

They are added to the result

Dynamics:

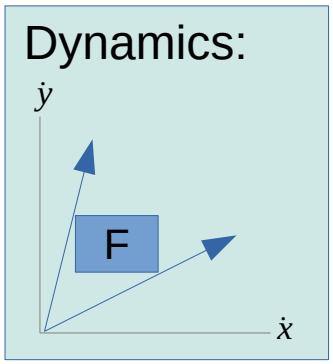


Computing RWA(U,V)

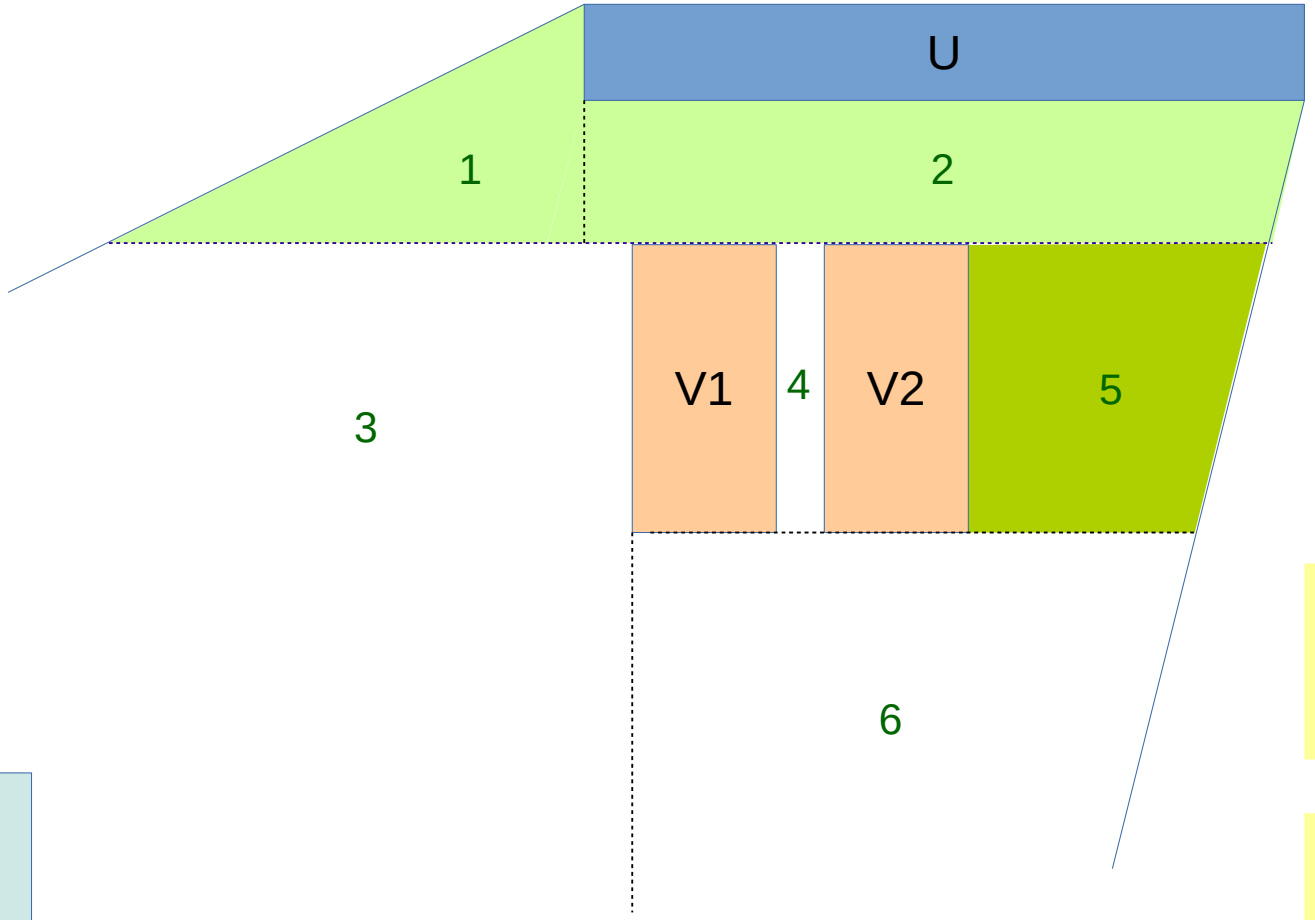


The other points of P1 go directly into P2

They are added to the result



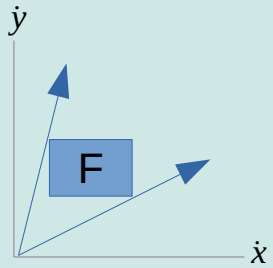
Computing RWA(U,V)



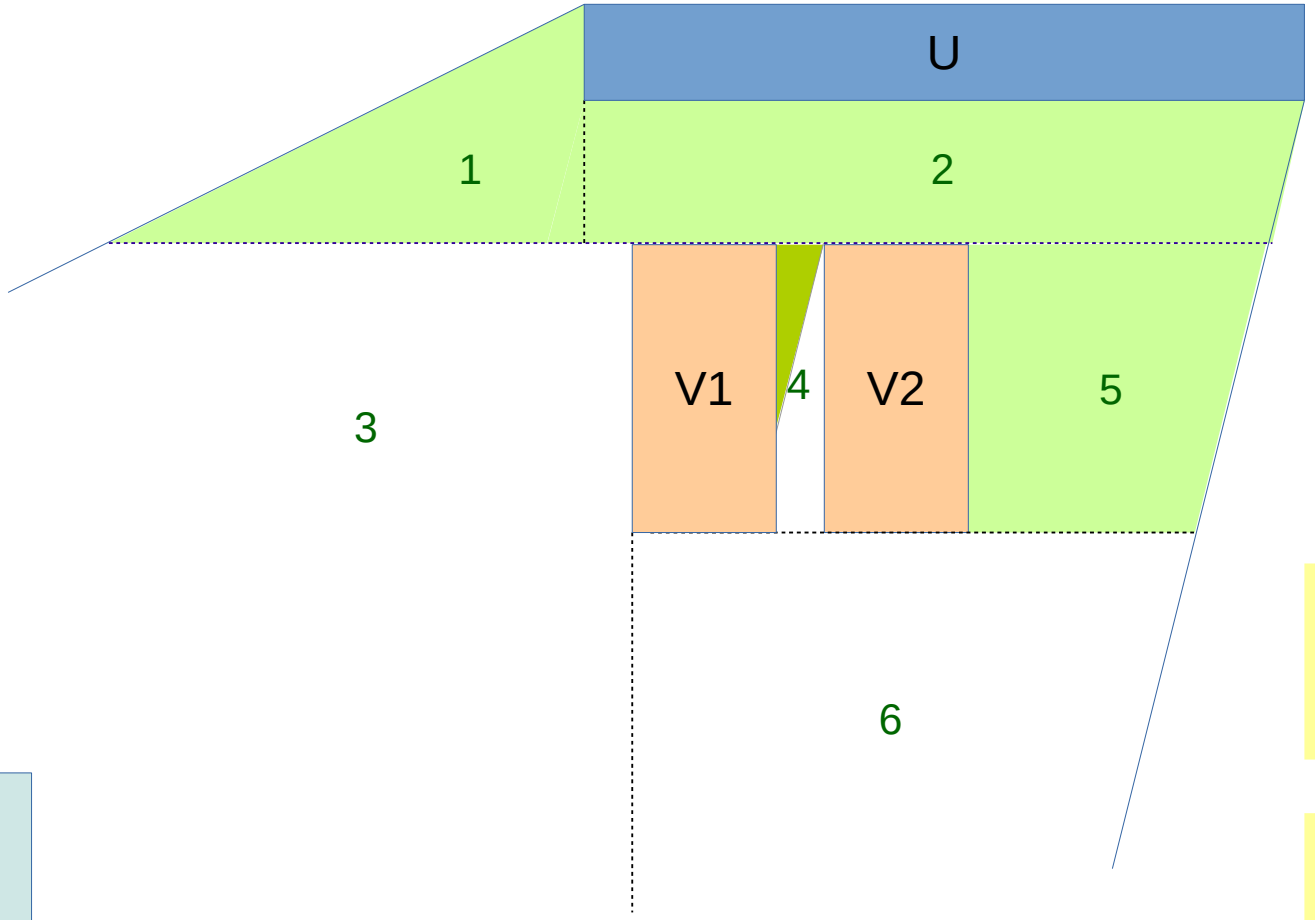
All points of P5 go directly into P2

They are added to the result

Dynamics:



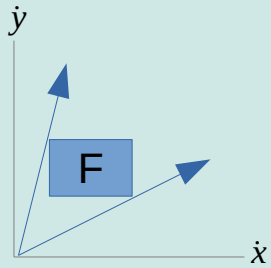
Computing RWA(U,V)



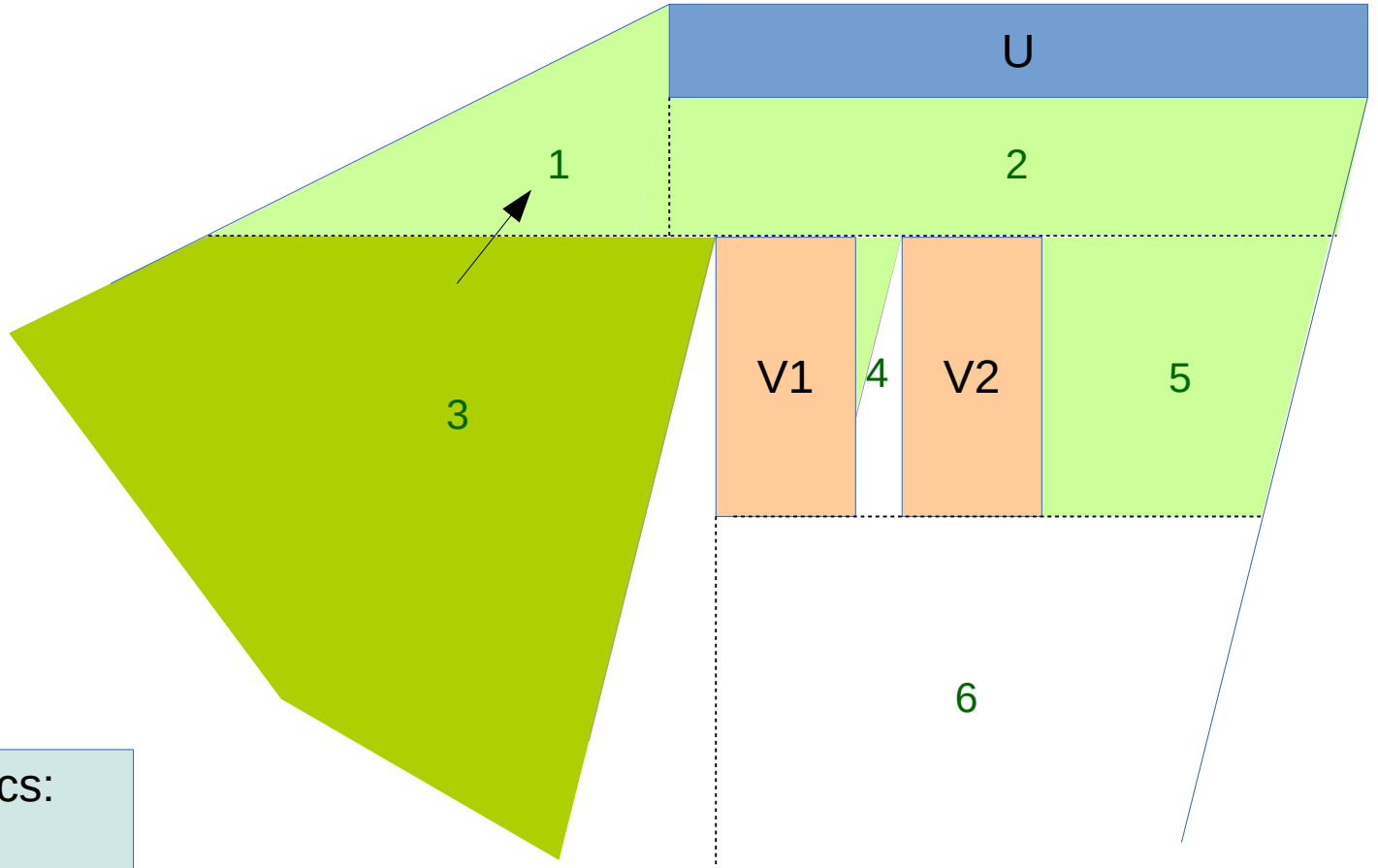
Some points of P4 go directly into P2

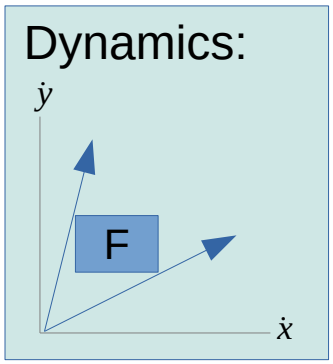
They are added to the result

Dynamics:

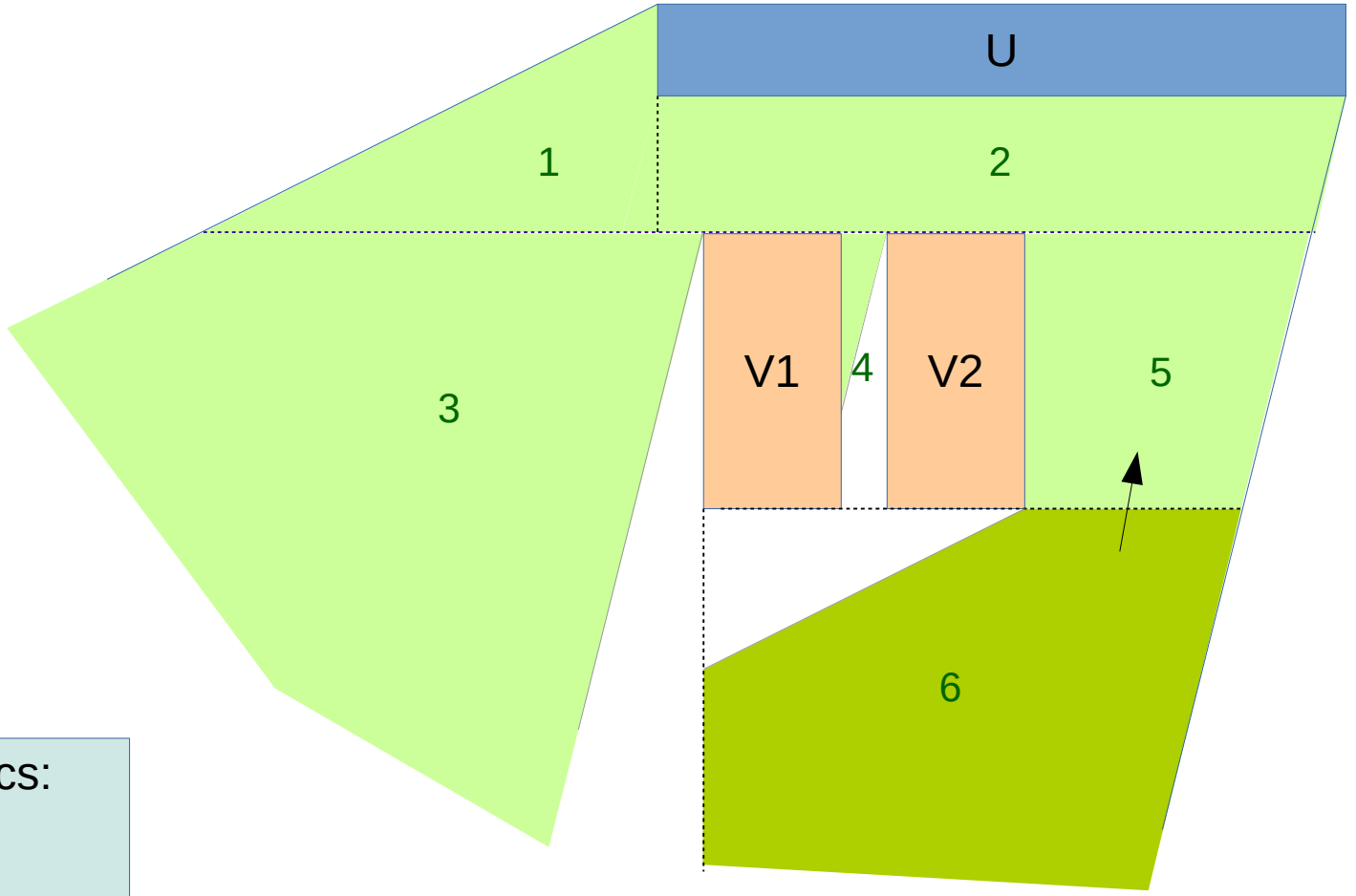


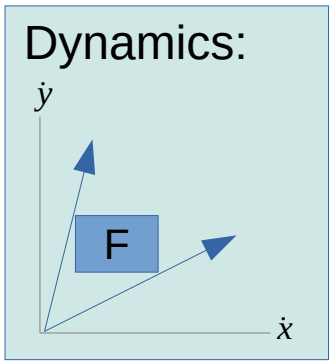
Computing RWA(U,V)



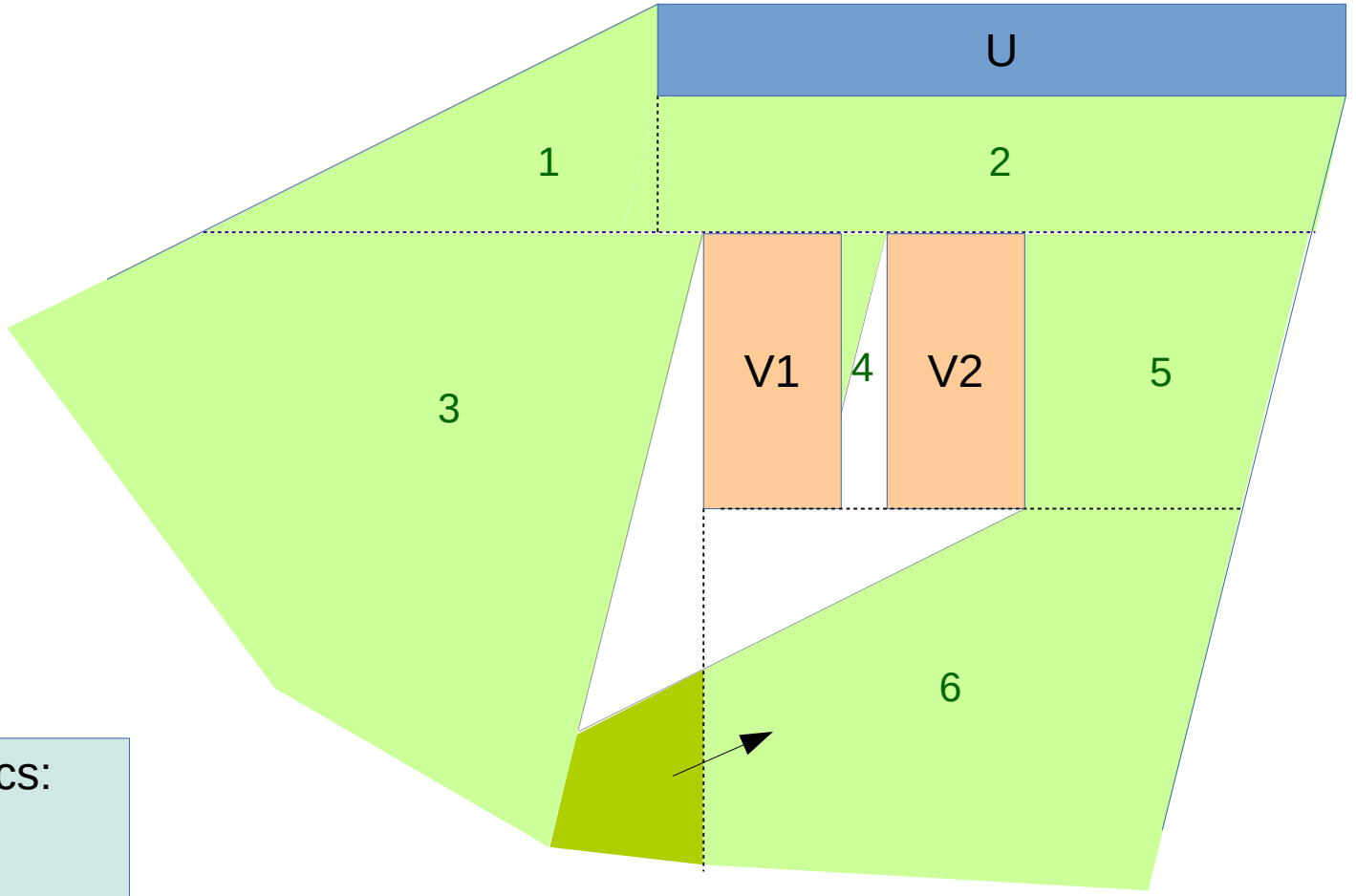


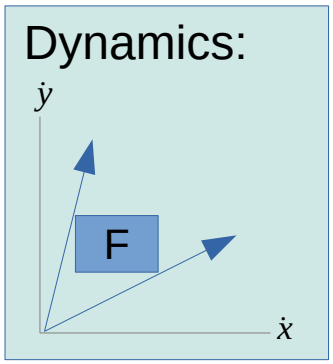
Computing RWA(U,V)



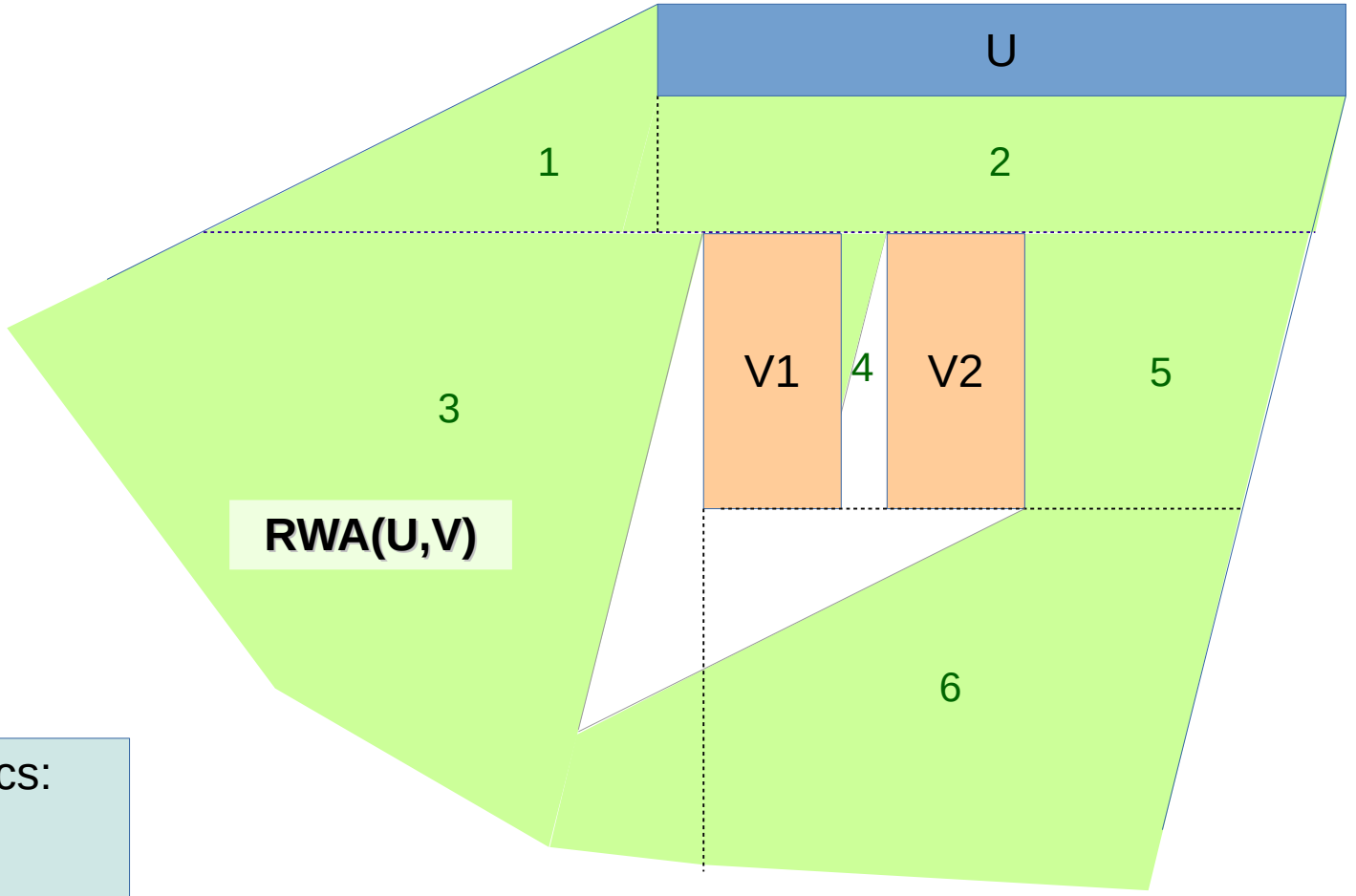


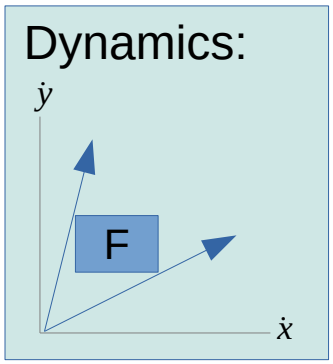
Computing RWA(U,V)





Computing RWA(U,V)





Algorithm for RWA(U, V)

$$\mu R\,.\,U \cup \bigcup_{P \in [\![\overline{V} \cap U_{\swarrow}]\!]} \bigcup_{P' \in [\![R]\!]} \Big(P \cap \big(bndry(P, P') \cap P'_{\swarrow} \big)_{\swarrow} \Big),$$

where

$$bndry(P, P') = (P \cap cl(P')) \cup (cl(P) \cap P').$$

and [[A]] is the representation of A as a finite set of convex polyhedra

- Interesting implementation issues
- Details in [Benerecetti et al., TCS 2013]

Controller Synthesis

Hybrid Games

A hybrid automaton whose transitions are divided between *controllable* and *uncontrollable*

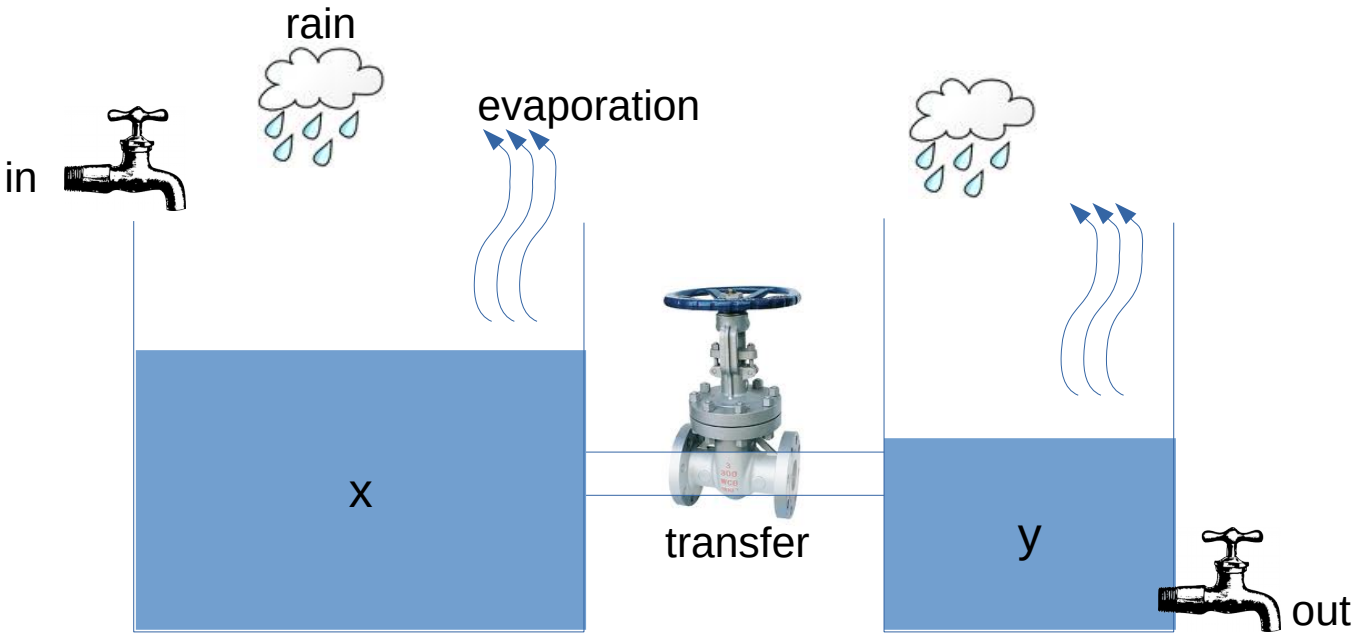
i.e., the controller can only take certain transitions

- it does not directly influence the continuous behavior
- a.k.a. *switching controller*

Control goals:

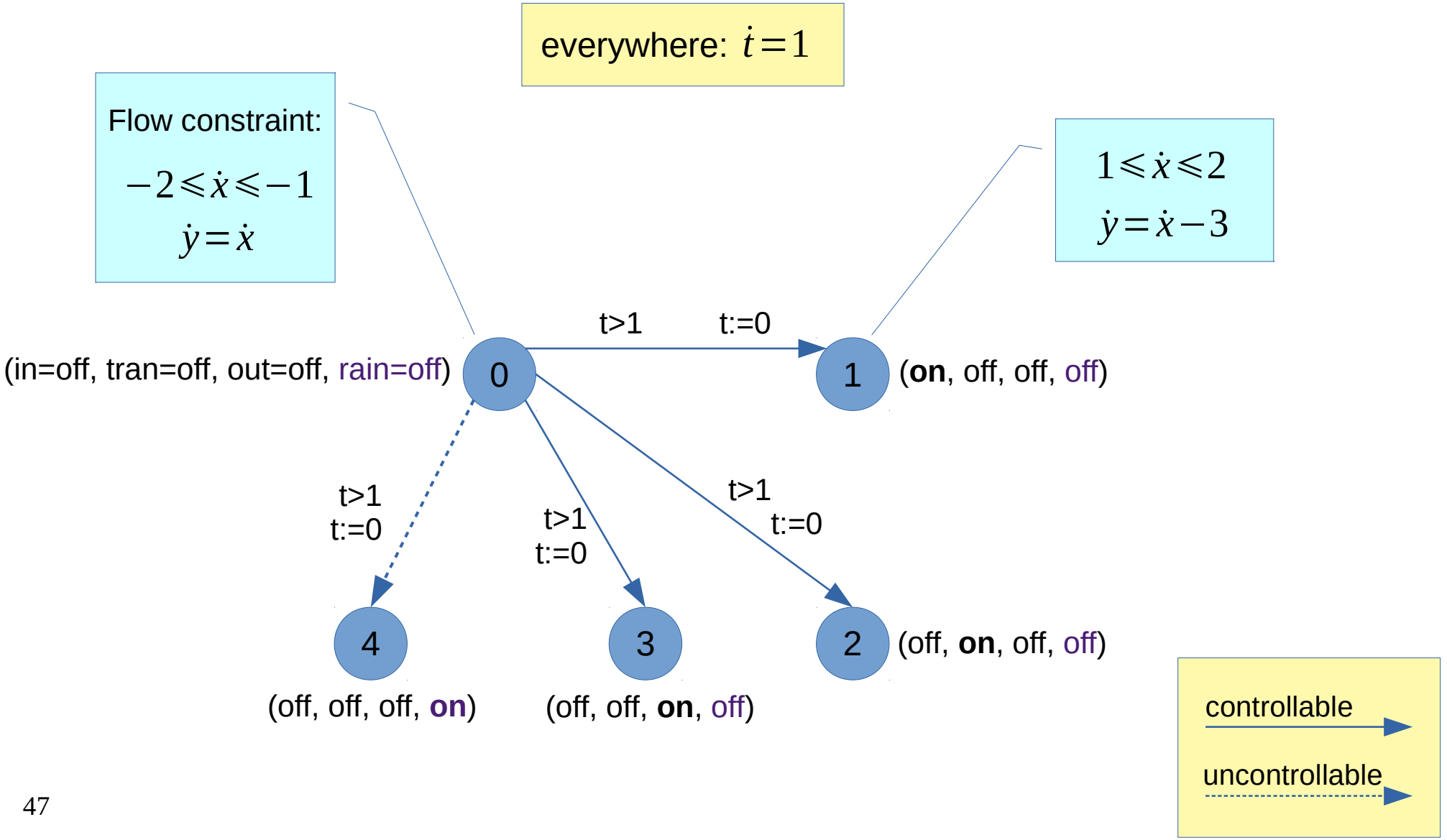
- safety
- reachability

Example: Two Open-air Tanks



- We **control** 3 discrete valves (on/off): *in*, *transfer*, and *out*
- **Rain** is an uncontrollable discrete event (on/off)
- **Evaporation** rate varies within given bounds
- Control **goal**: keep the level in both tanks within bounds (safety)
- At least one time unit between any two transitions (prevents *Zenoness*)

HG Fragment for Two Open-air Tanks



The Safety Control Problem

- A (control) **strategy** is a function from states to moves of the controller
  - **possible moves: take an enabled controllable transition or do nothing**
  - a form of *closed-loop* control

- A **strategy** is **winning** if it constrains the system within the safe set
- **Problem:** Given a HG and a safe set, compute the set of states from which the controller has a winning strategy (**winning states**)

The General Algorithm for Safety Games

- Inspired by finite-state games
- Based on the “controllable predecessors” operator

$$CPre : 2^S \rightarrow 2^S$$

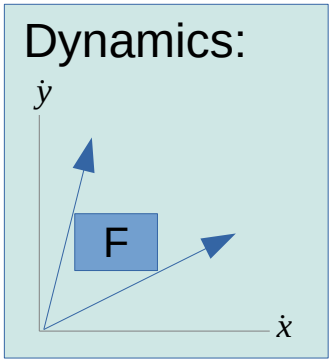
Definition. Given a set of states Z, **CPre(Z)** contains the states from which the controller can ensure that the system **remains in Z until the next discrete transition (included).**

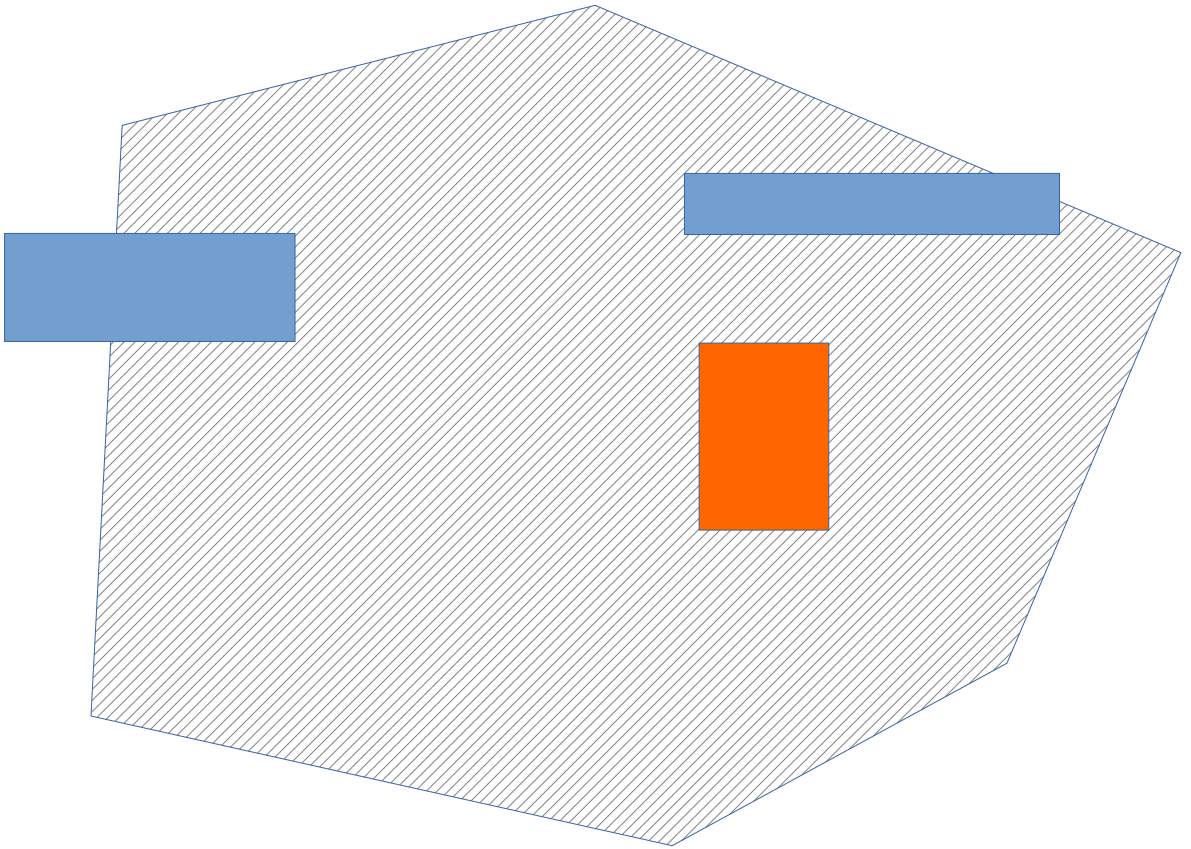
The General Algorithm for Safety Games

Given the set of safe states R, the set of winning states is:

$$\nu Z . R \cap CPre(Z)$$

Computing CPre(Z) on LHGs



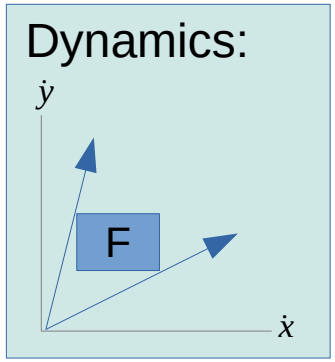


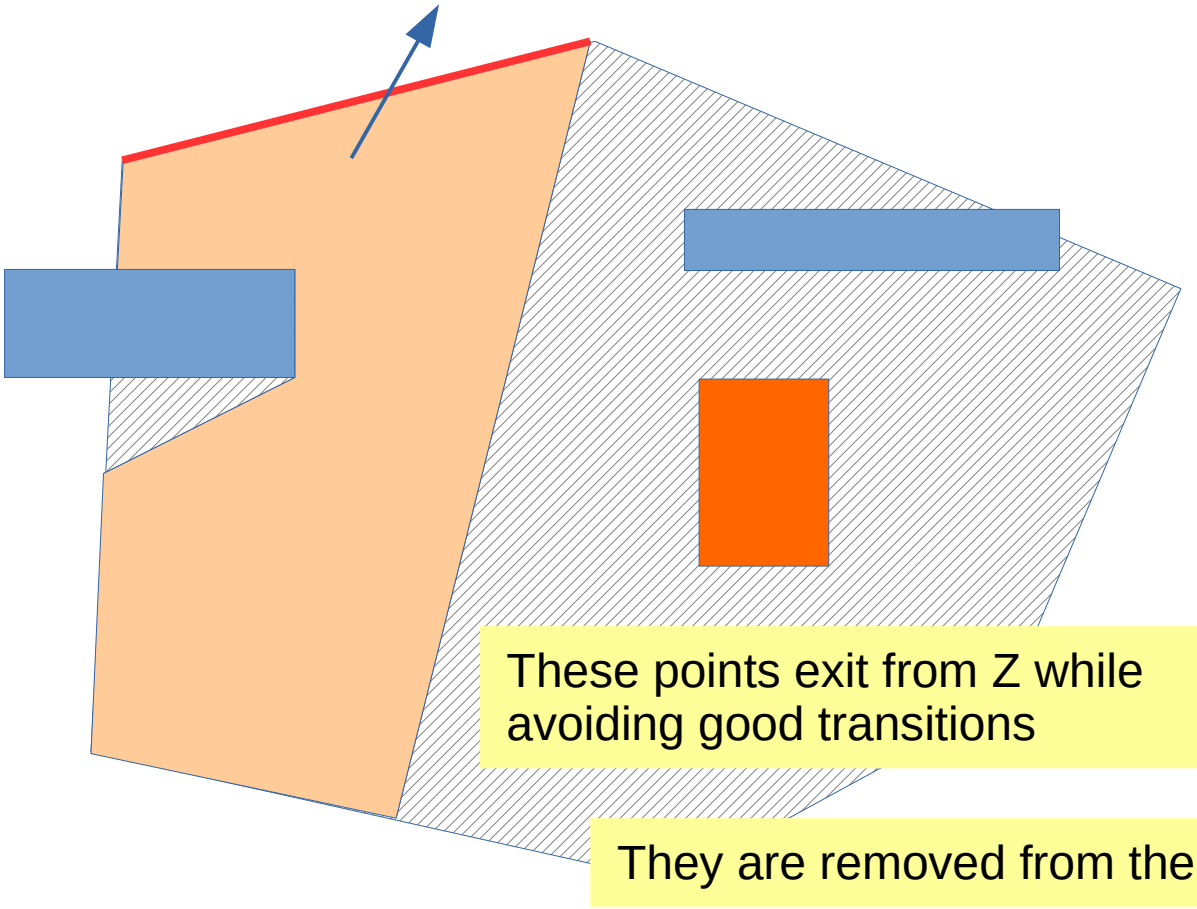
Z

controllable trans. leading into Z (good)

uncontrollable trans. leading outside Z (bad)

Computing CPre(Z) on LHGs





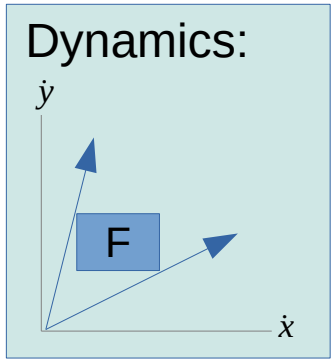
These points exit from Z while avoiding good transitions

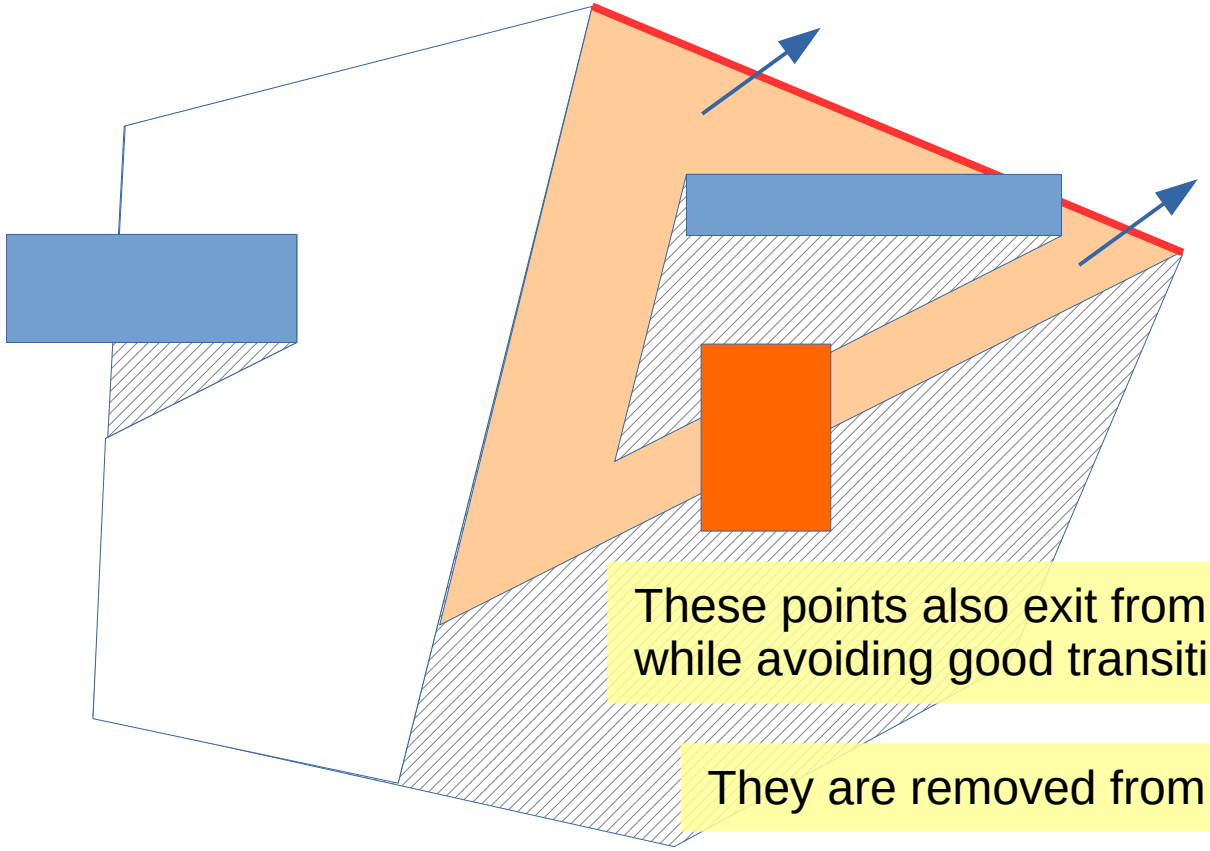
They are removed from the result

controllable trans. leading into Z (good)

uncontrollable trans. leading outside Z (bad)

Computing CPre(Z) on LHGs





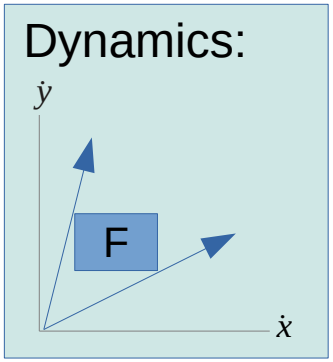
These points also exit from Z
while avoiding good transitions

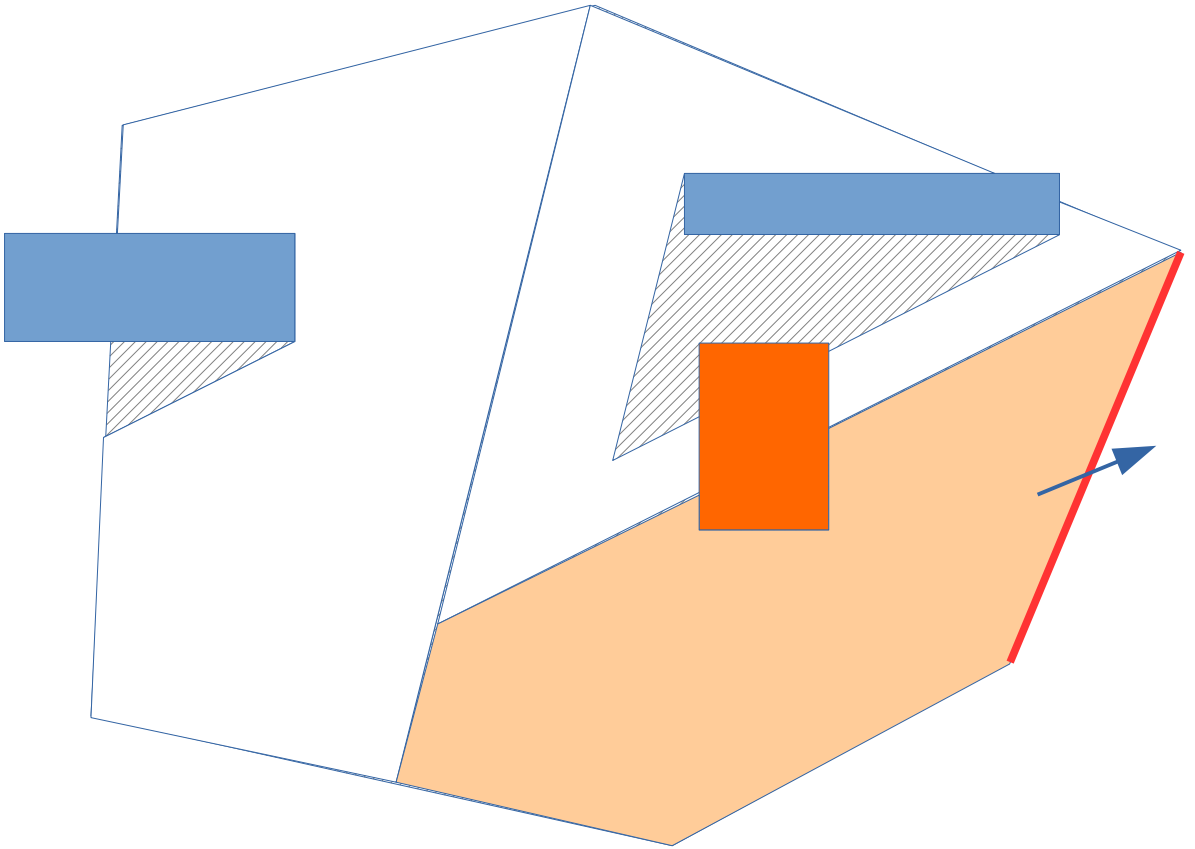
They are removed from the result

controllable trans. leading into Z (good)

uncontrollable trans. leading outside Z (bad)

Computing CPre(Z) on LHGs

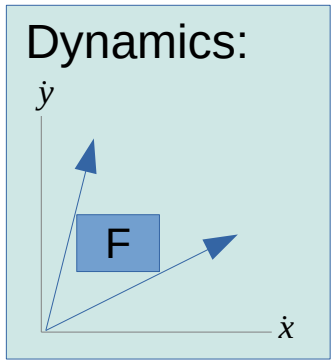


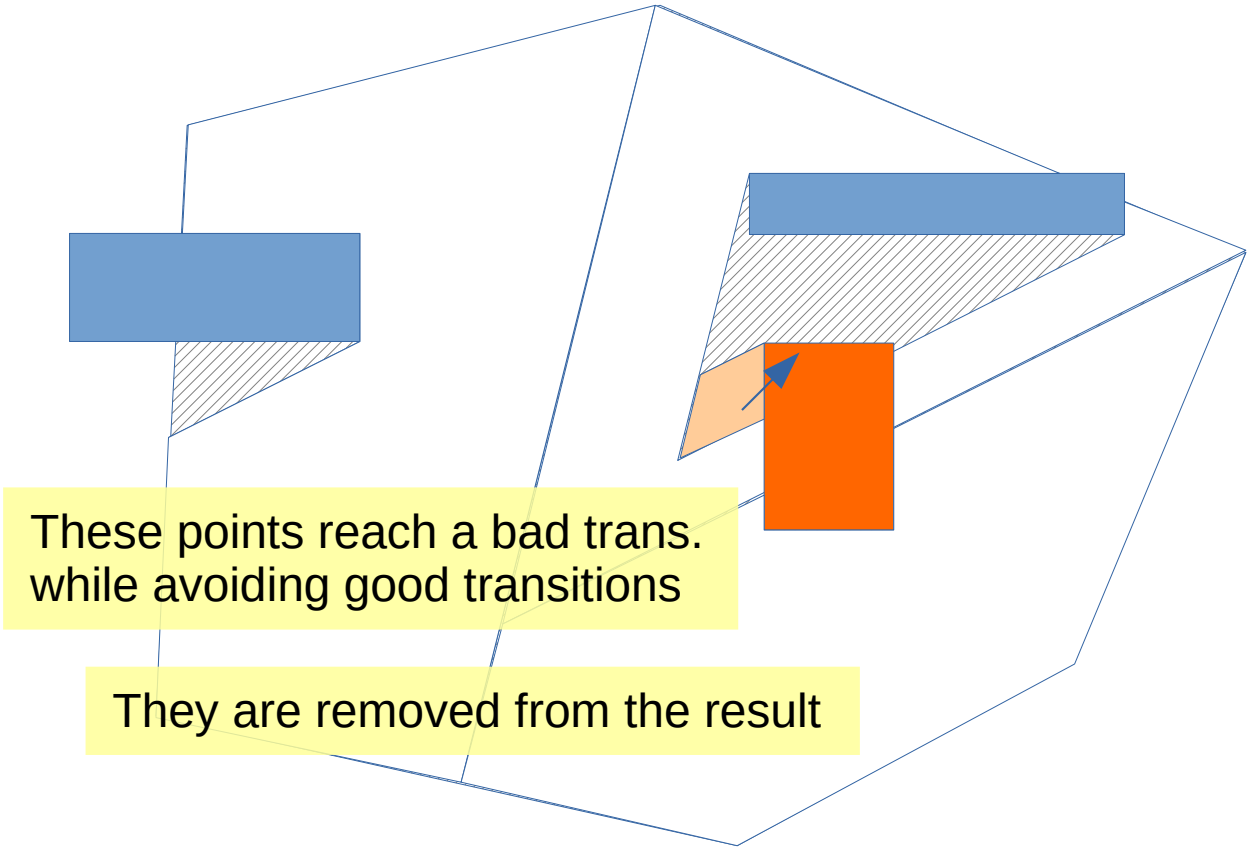


controllable trans. leading into Z (good)

uncontrollable trans. leading outside Z (bad)

Computing CPre(Z) on LHGs

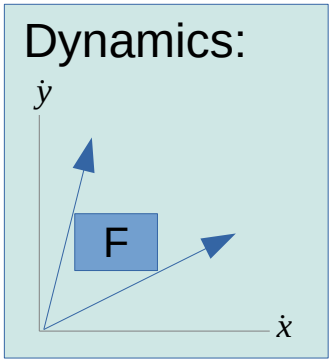


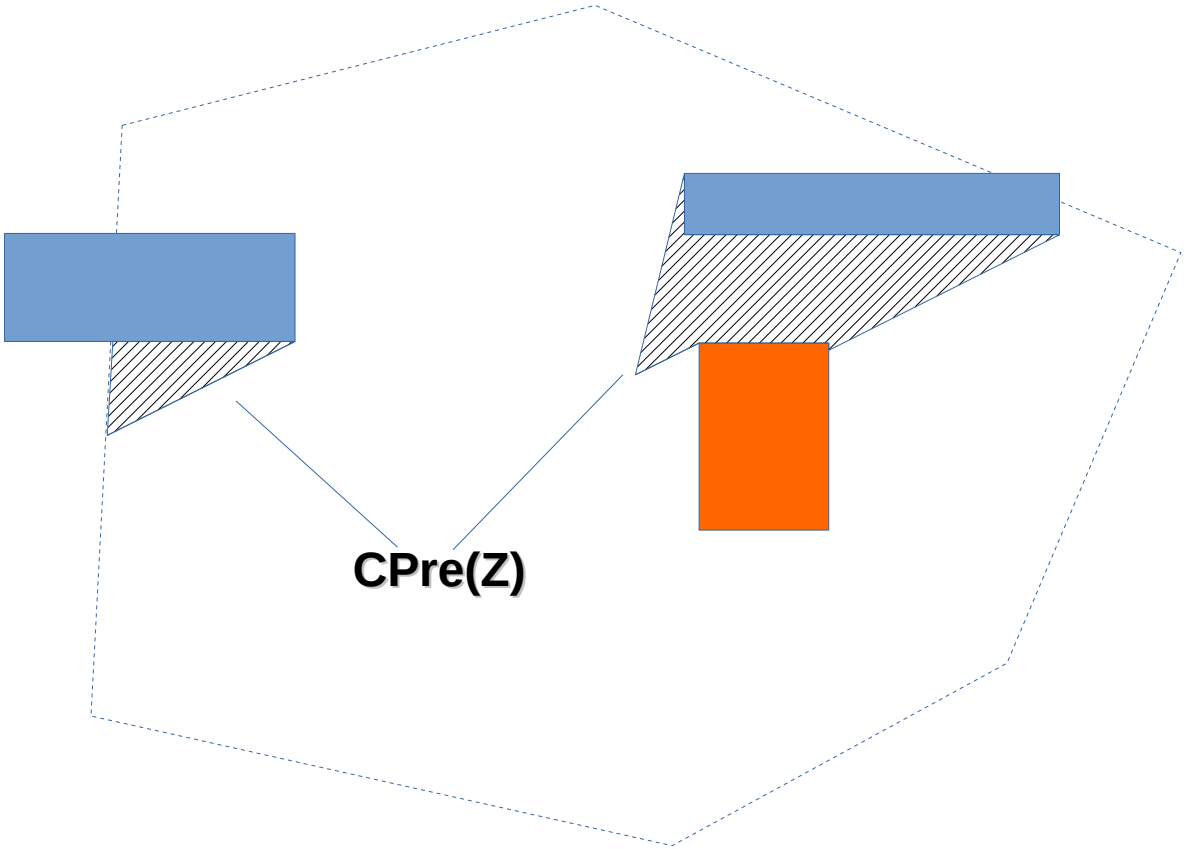


controllable trans. leading into Z (good)

uncontrollable trans. leading outside Z (bad)

Computing CPre(Z) on LHGs

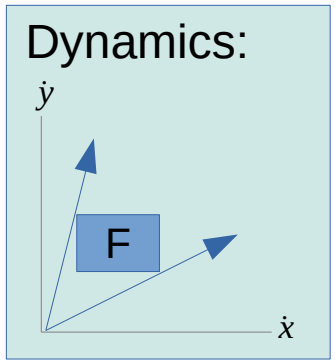


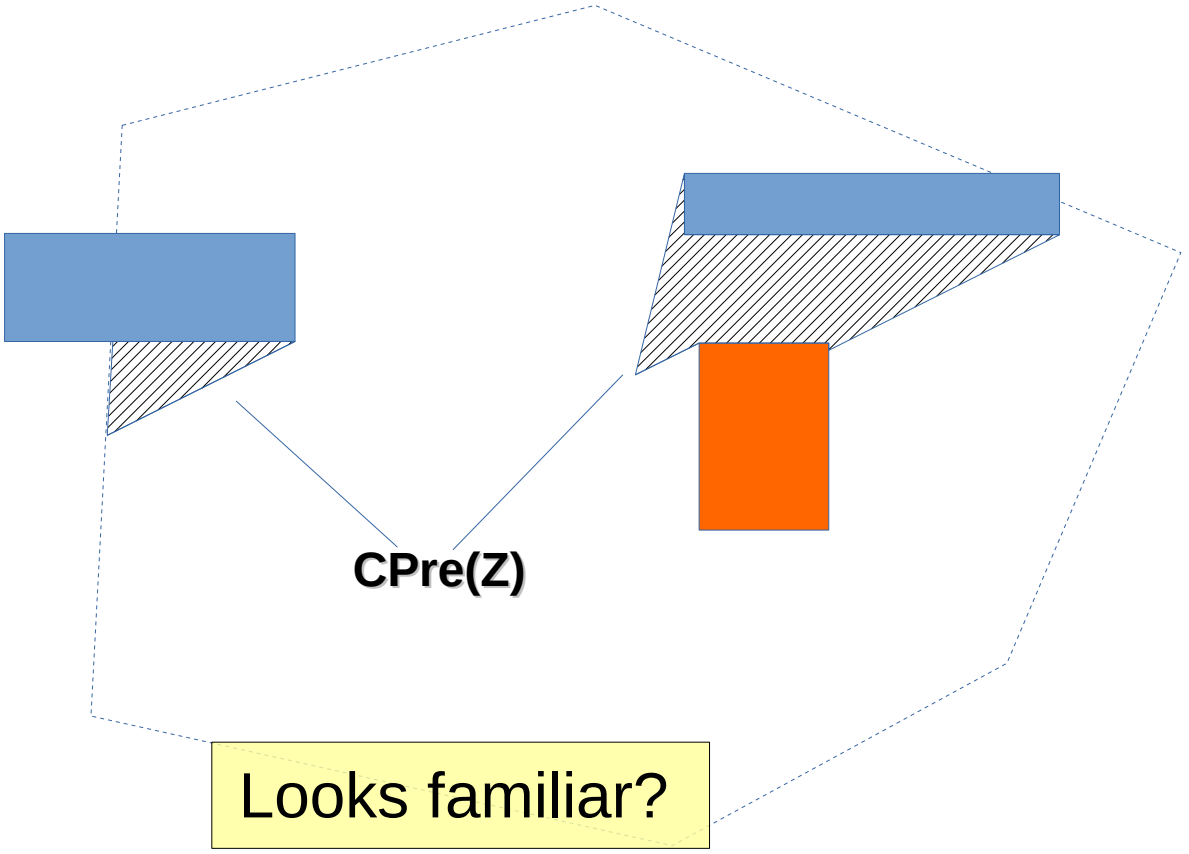


controllable trans. leading into Z (good)

uncontrollable trans. leading outside Z (bad)

Computing CPre(Z) on LHGs





controllable trans. leading into Z (good)

uncontrollable trans. leading outside Z (bad)

Computing CPre(Z) on LHGs

We can compute CPre using RWA!

$$\mathrm{CPre}(Z) = Z \setminus \mathbf{RWA}(\overline{Z} \cup \mathrm{Bad}, \mathrm{Good})$$

where:

Bad = $\mathrm{PreJmp}_{\mathbf{u}}(\overline{Z})$ predecessors under **u**ncontrollable trans.

Good = $\mathrm{PreJmp}_{\mathbf{c}}(Z)$ predecessors under **c**ontrollable trans.

Warning: there are special effects when a trajectory exits from Z and from the invariant at the same time.

Proof Sketch in Temporal Logic

$$RWA(U,V) = \exists\, \overline{V}\ \textit{Until}\ U$$

$$CPre(Z) = Z \cap \forall\, (Z \cap \overline{Bad})\ \textit{WUntil}\ Good$$

weak until

Recall the classical equivalence:

$$\forall\, A\ \textit{WUntil}\ B \ \equiv\ \neg\, \exists\, \overline{B}\ \textit{Until}\ \overline{A}$$

Hence,

$$CPre(Z) = Z \setminus \mathbf{RWA}(\overline{Z} \cup Bad, Good)$$

The Reachability Control Problem

The Reachability Control Problem

- Control **goal**: bring system into a desired set of target states
  - *reachability game*

- **Problem.** Given a HG and a target set, compute the set of states from which the controller has a winning strategy (**winning states**)

(non-) Duality

- In most types of finite-state games, a reachability game can be reduced to its *dual safety game*
- I.e.,
  - to solve the game with goal “F R”...
  - ...exchange the players...
  - ...and solve the game with goal “G $\overline{R}$”

- This does not work for LHGs
- LHGs are **asymmetric**: continuous flow (timed steps) is always adversarial
- We need a **direct algorithm** for reachability games

The General Algorithm for Reachability Games

Based on a variant to the controllable predecessors operator

$$CPreReach : 2^S \rightarrow 2^S$$

Definition. Given a set of states Z, **CPreReach(Z)** contains the states from which the controller can ensure that **the system reaches Z within the next discrete transition (included).**

The General Algorithm for Reachability Games

Given the target set R, the set of winning states is:

$$\mu Z \,.\, R \cup CPreReach(Z)$$

Computing CPreReach

All trajectories should reach Z ∪ Good while avoiding Bad.

$$\text{CPreReach}(Z) = \forall\ \overline{\text{Bad}}\ \textit{Until}\ (Z \cup \text{Good})$$

where as before:

Bad = $\text{PreJmp}_{\mathbf{u}}(\bar{Z})$

Good = $\text{PreJmp}_{\mathbf{c}}(Z)$

Another Type of RWA!

Fix a location q and the corresponding flow constraint Flow(q).

Definition. Given two (possibly non-convex) polyhedra **U** and **V**, **must-RWA(U,V)** is the set of points from which **all trajectories reach U while avoiding V.**

in temporal logic (CTL): $\forall\, \overline{\forall}$ *Until* U

CPreReach(Z) = **must-RWA**(Z ∪ Good, Bad)

Computing must-RWA

Unfortunately, forall-until (must-RWA) cannot be reduced to exists-until (RWA).

However, we can use exists-until to do the *heavy lifting*.

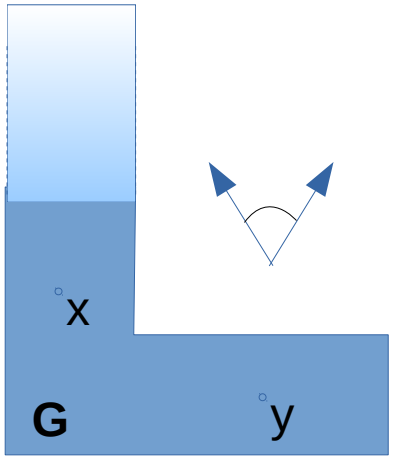
Idea:

- First, compute an **over-approximation** of must-RWA
- Then, **use RWA to refine it** (remove unwanted points)

Boundedness

Given a location *q* and a (possibly non-convex) polyhedron G:

- **A point p** in G is **q-*bounded*** if all trajectories starting from p eventually exit from G
- **G is q-bounded** if all of its points are



- y is q-bounded
- x is not q-bounded
- G is not q-bounded

Computing must-RWA: the Idea in Detail

Assume w.l.o.g. that U and V are **disjoint**.

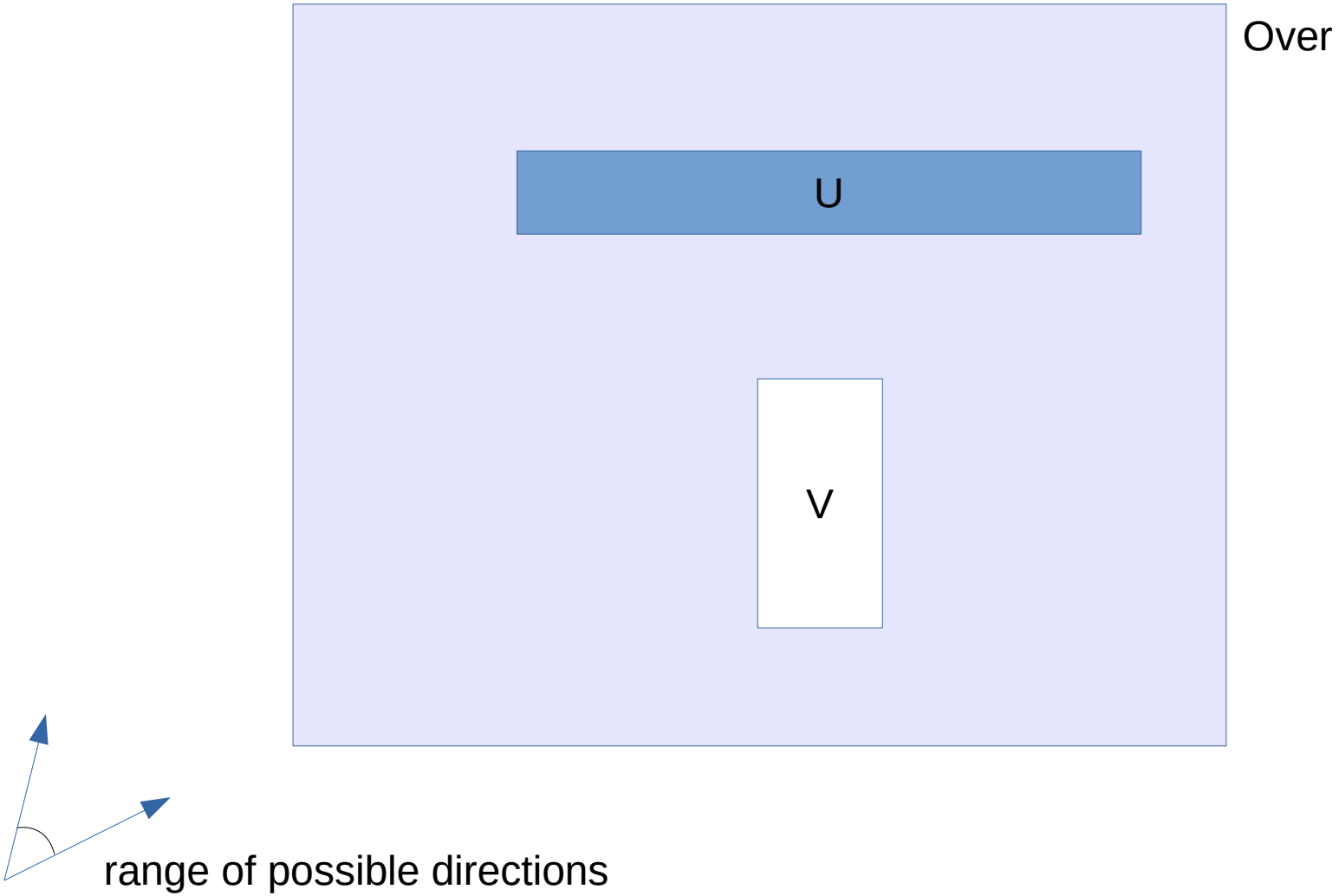
Clearly, U ⊆ must-RWA(U,V).

Let **Over** be a set such that:

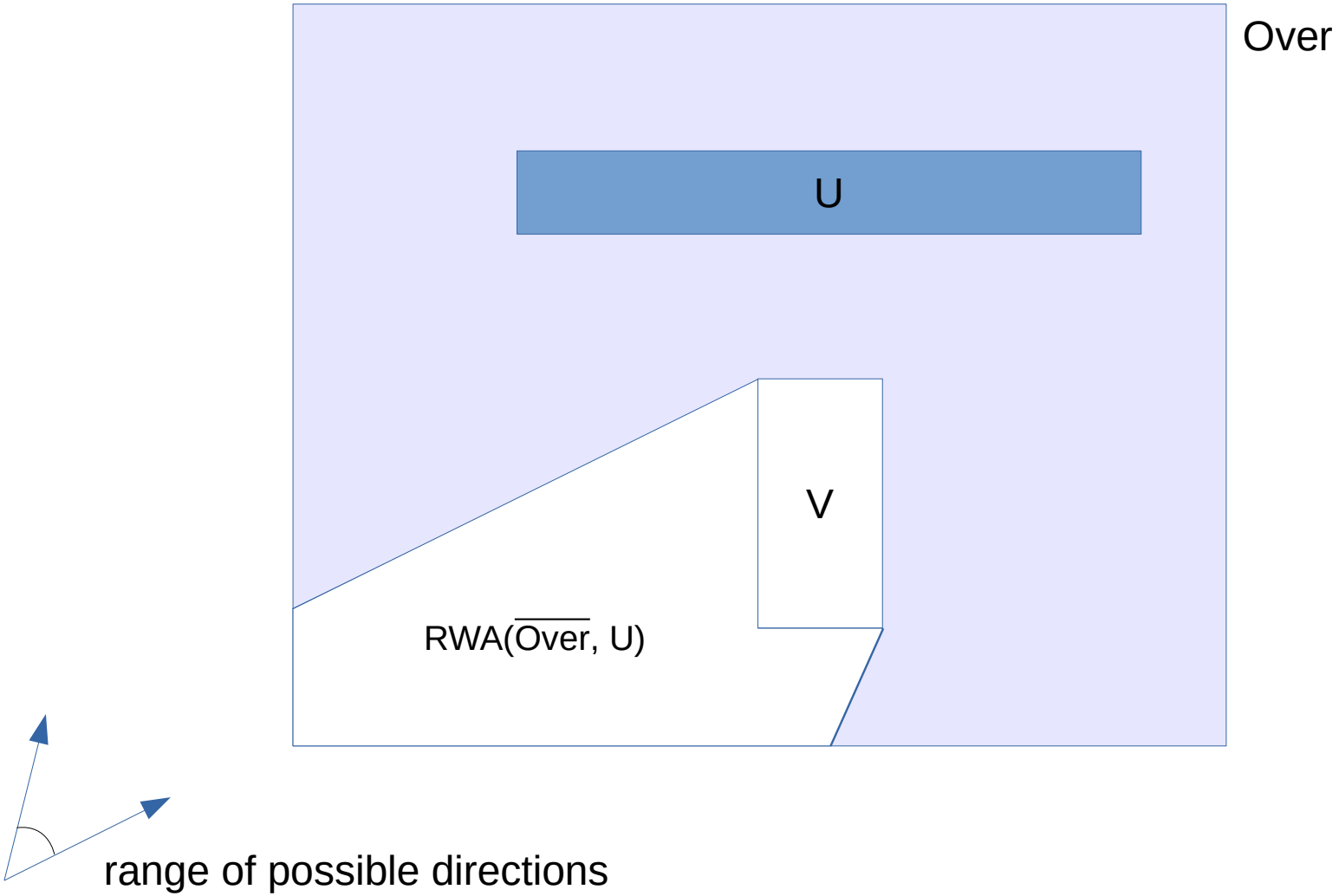
1) must-RWA(U,V) ⊆ Over (it is an over-approximation)
2) Over ∩ V = ∅ (it does not contain trivially "losing" points)
3) Over \ U is q-*bounded* (the "unsafe" part of Over is q-bounded)

Theorem: must-RWA(U,V) = Over \ RWA**($\overline{\text{Over}}$, U)**

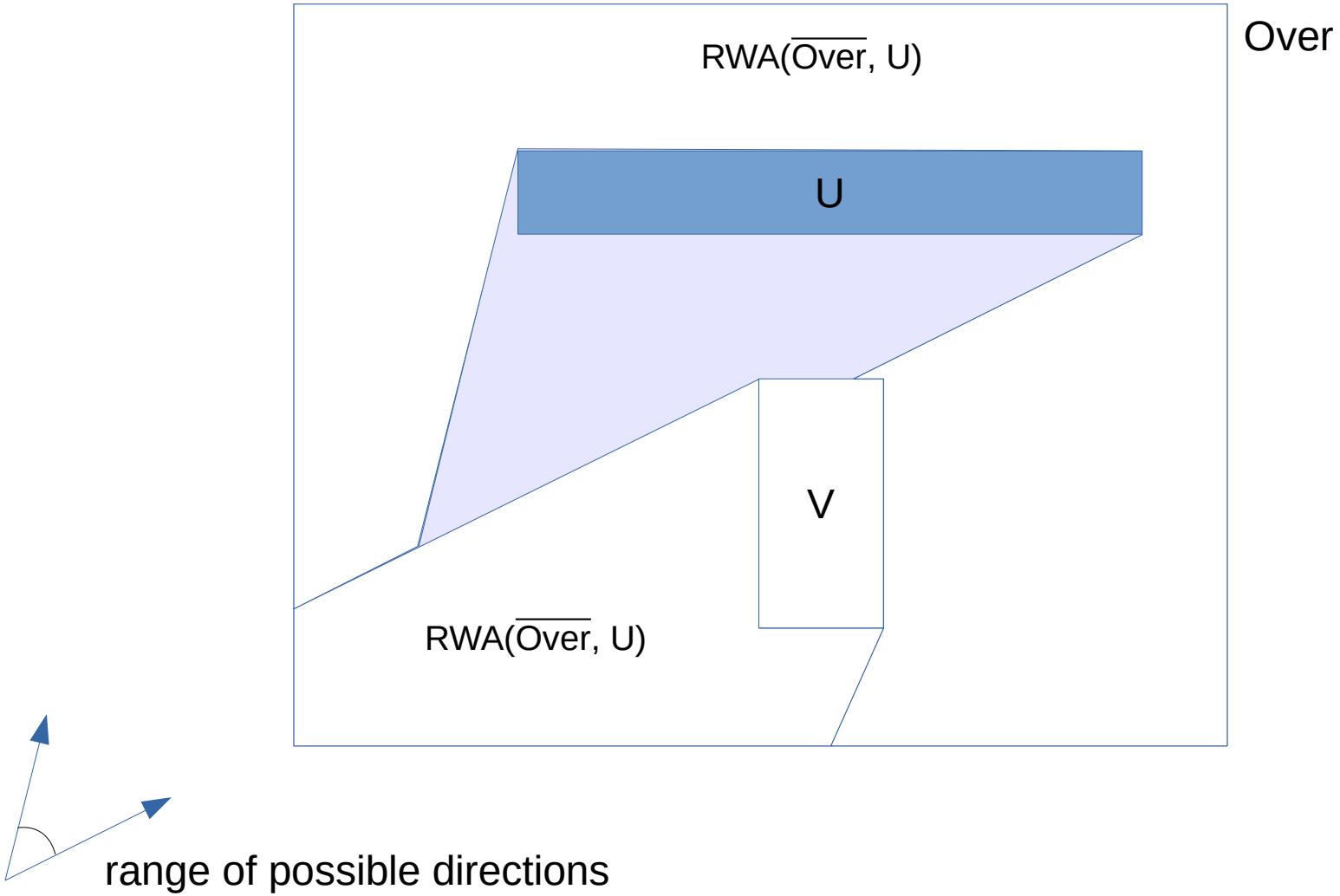
Computing must-RWA Using RWA



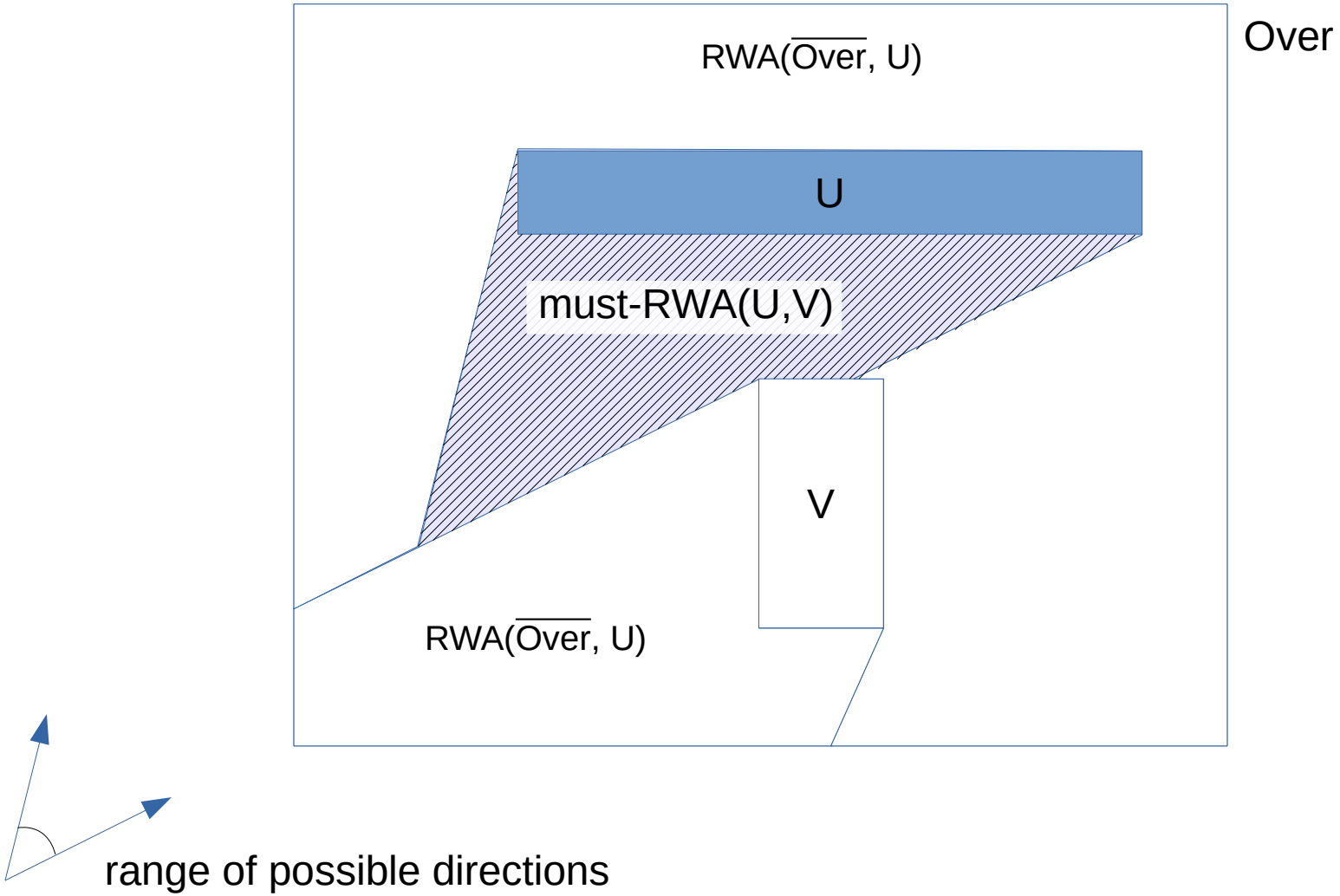
Computing must-RWA Using RWA



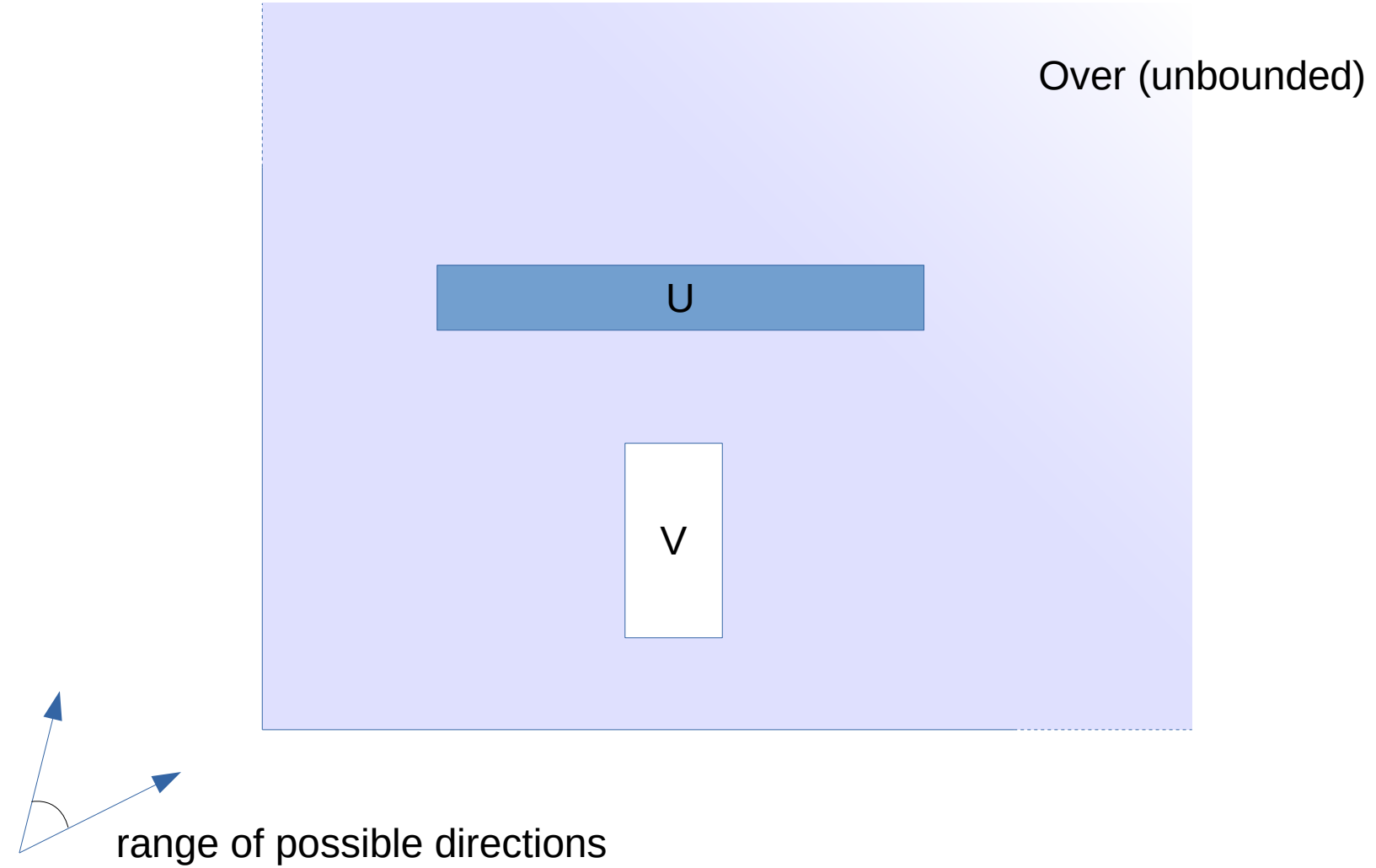
Computing must-RWA Using RWA



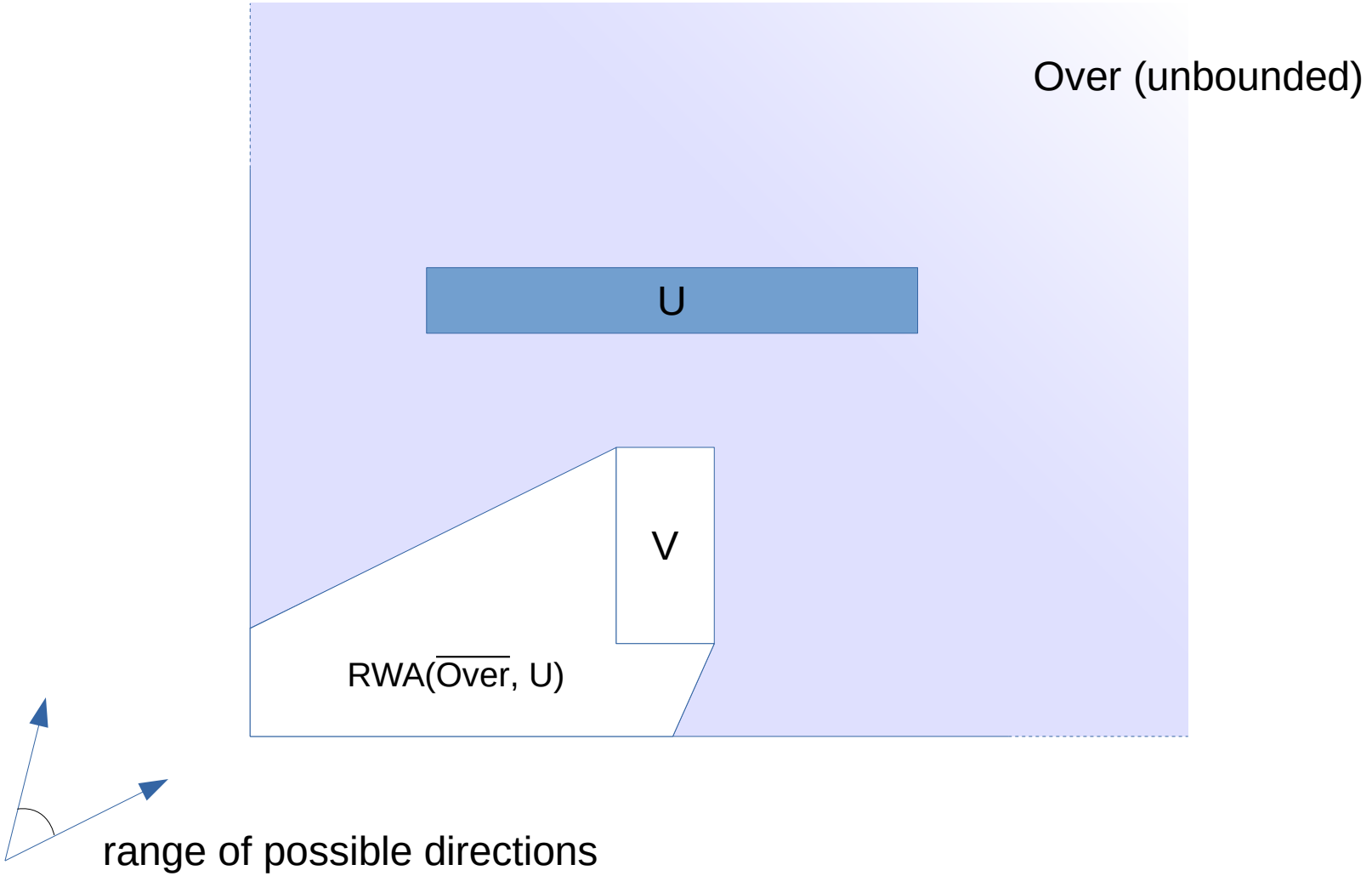
Computing must-RWA Using RWA



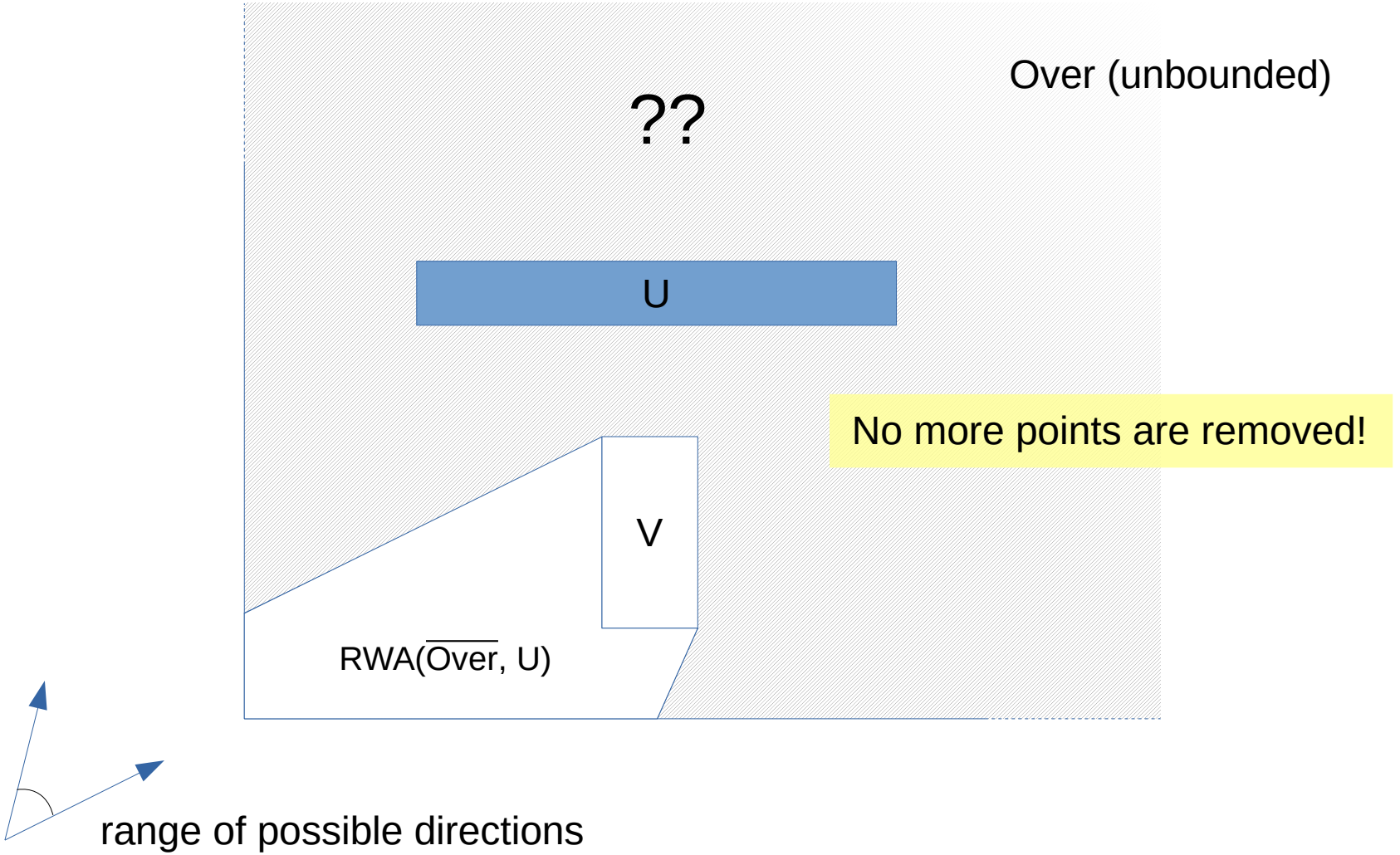
Why the Over-approximation Must Be Bounded



Why the Over-approximation Must Be Bounded



Why the Over-approximation Must Be Bounded



Computing the Over-approximation

- “not V” is an obvious over-approximation
  - however, its unsafe part “Over \ U” is not necessarily q-**bounded**

- How do we make it q-bounded while preserving all good points?
  - *How do we distinguish good points?*

- **Idea sketch:**

  - Luckily, the **good points** in Over \ U are already q-**bounded**
    - because they can reach U
  - Hence, **preserve all the q-bounded points!**

more info in [Benerecetti & F., ACM TECS, 2017]

References

- M. Benerecetti, M.F. **Tracking Smooth Trajectories in Linear Hybrid Systems.** *Information & Computation*, 257, 2017.
- M. Benerecetti, M.F. **Automatic Synthesis of Switching Controllers for Linear Hybrid Systems: Reachability Control**. *ACM Trans. on Embedded Computing Systems*, 16(4), 2017.
- M. Benerecetti, M.F., S. Minopoli. **Automatic Synthesis of Switching Controllers for Linear Hybrid Systems: Safety Control**. *Theoretical Computer Science*, 493, 2013.

Also: HSCC 2013, HSCC 2012, IEEE CDC 2011, GandALF 2011.

Tool Demo

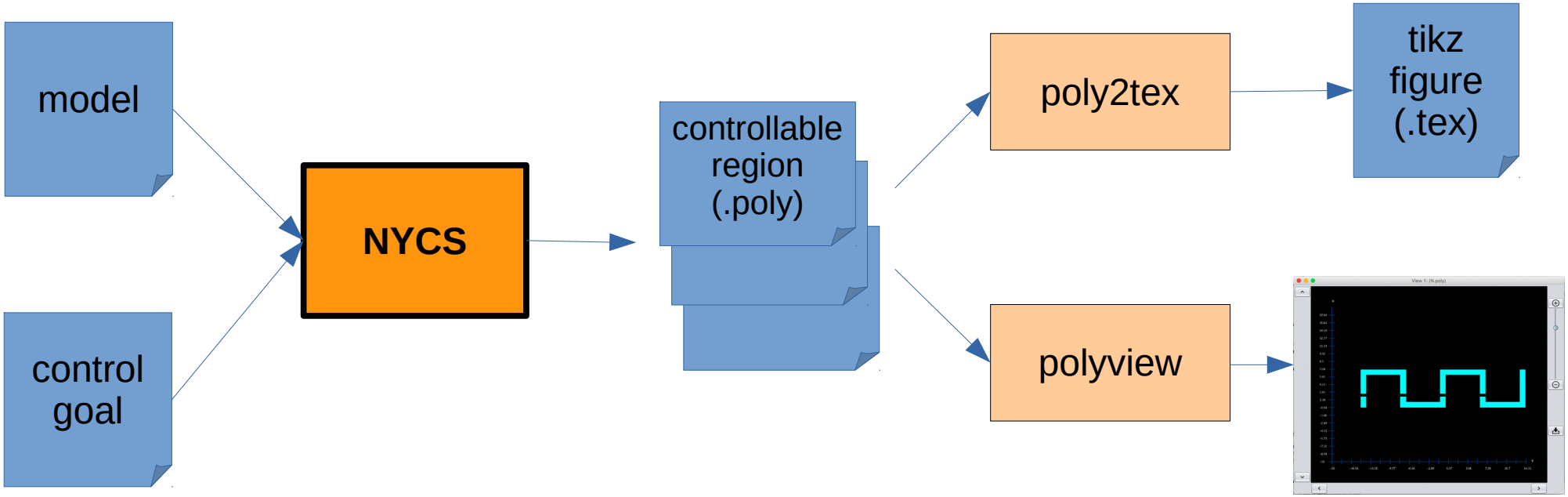
NYCS
Naples hYbrid Controller Synthesis

Features:

- controllability region w.r.t. **safety** goal
- controllability region w.r.t. **reachability** goal
- optional *smooth* semantics (instead of a.e. differentiable)

`http://wpage.unina.it/m.faella/nycs`

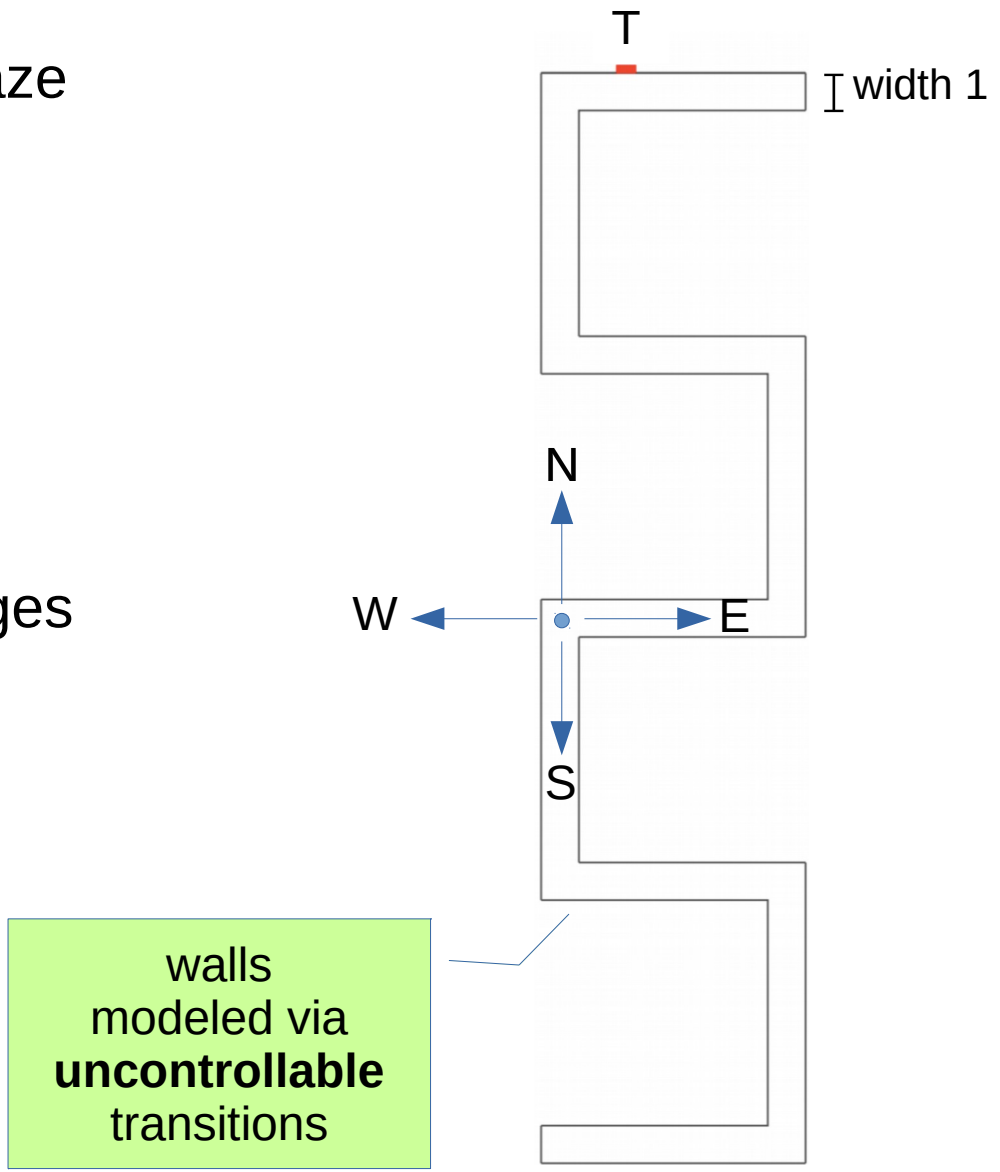
NYCS Schematics



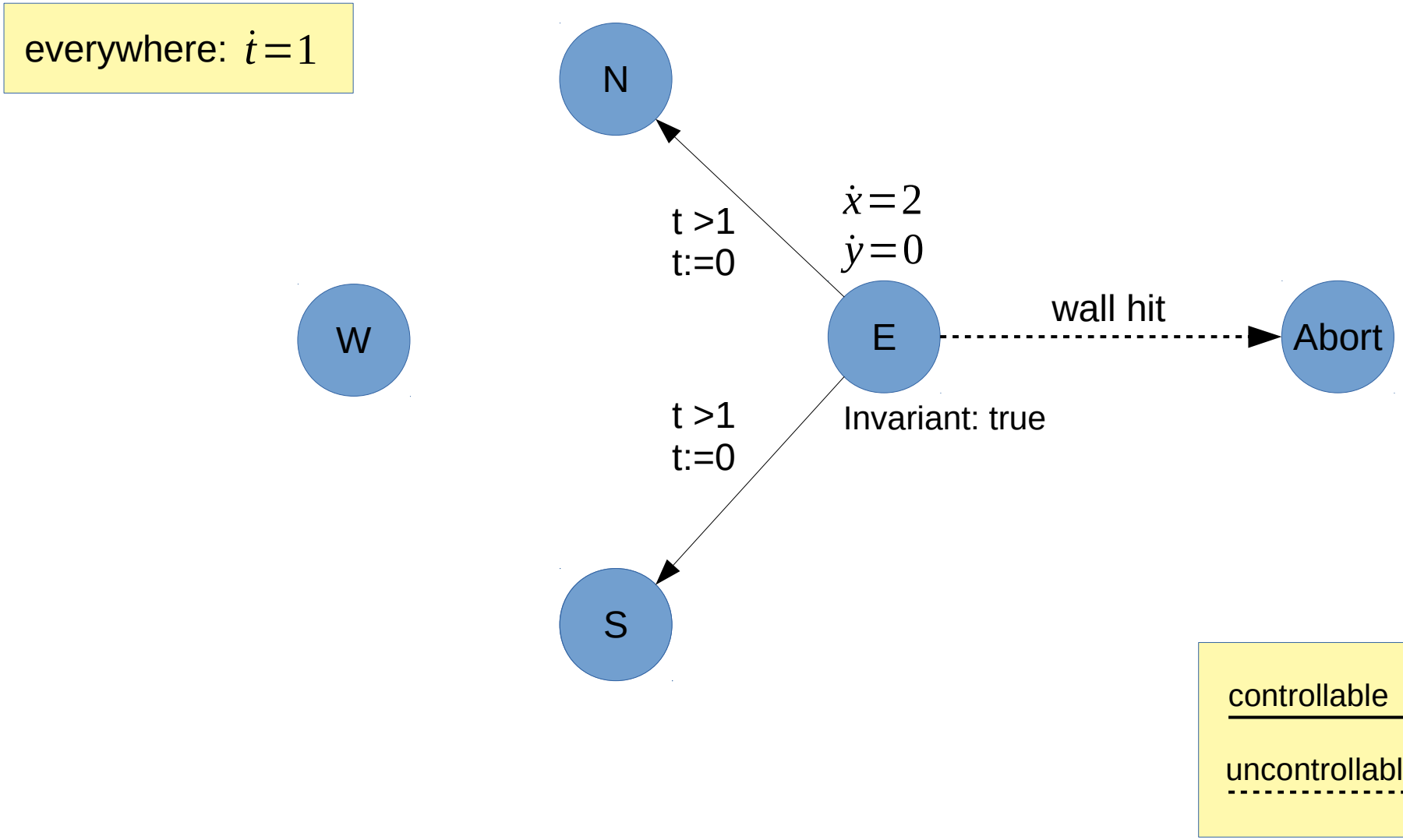
The “Maze” Example

Navigate a point vehicle in a maze

- control goal: **reach target T**
- 4 cardinal directions
- speed = 2
- 1 time unit between two changes of dir. (non-Zeno)



The “Maze” Example



Demo

```
nycs maze.xml maze.cfg.xml
```