Finding Errors of Hybrid Systems by Optimising an Abstraction-Based Quality Estimate

Stefan Ratschan Jan-Georg Smaus

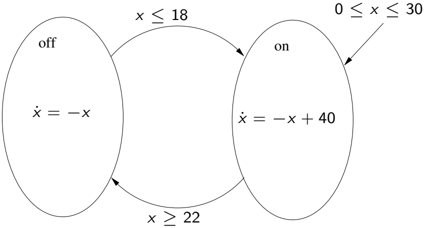
Institute of Computer Science of the Czech Academy of Sciences

Albert-Ludwigs-Universität Freiburg

July 3, 2009

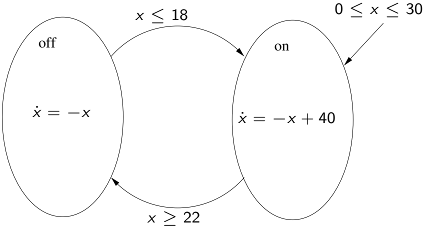
Hybrid Systems

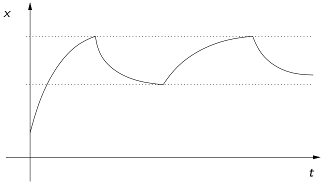
Thermostat:



Hybrid Systems

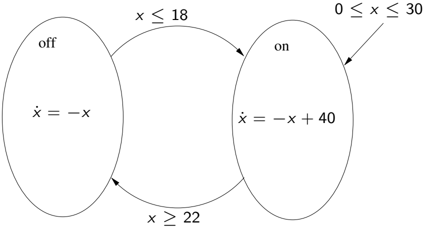
Thermostat:

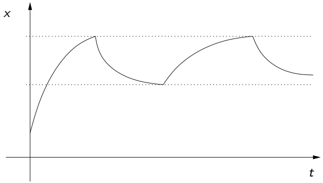




Hybrid Systems

Thermostat:

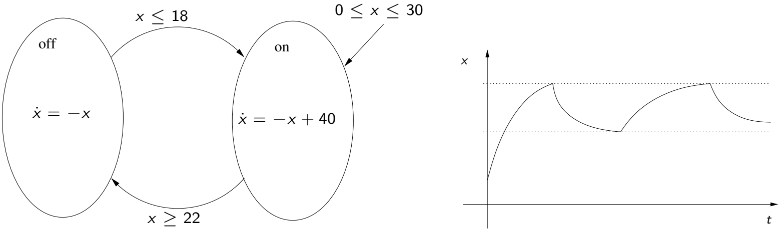




Dynamical system with both continuous and discrete state and evolution.

Hybrid Systems

Thermostat:

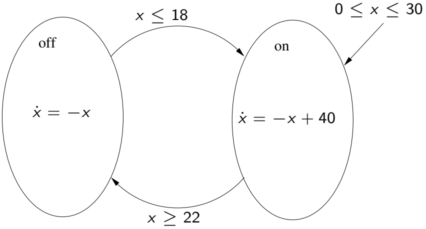


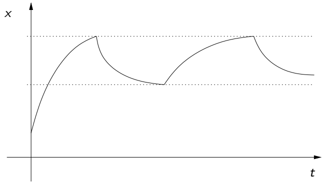
Dynamical system with both continuous and discrete state and evolution.

Also continuous state can jump discontinuously (state updates)

Hybrid Systems

Thermostat:





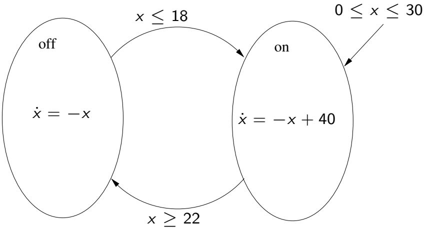
Dynamical system with both continuous and discrete state and evolution.

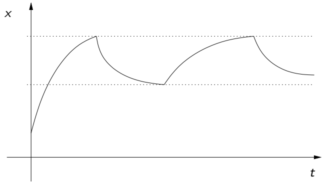
Also continuous state can jump discontinuously (state updates)

Non-linearity (differential equations, updates)

Hybrid Systems

Thermostat:





Dynamical system with both continuous and discrete state and evolution.

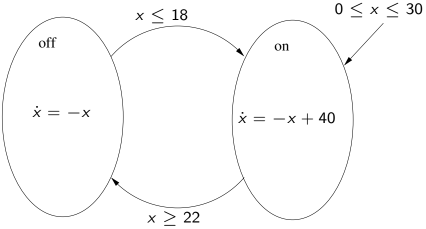
Also continuous state can jump discontinuously (state updates)

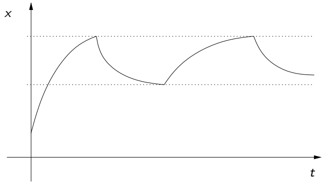
Non-linearity (differential equations, updates)

In illustrations: systems with just one control mode.

Hybrid Systems

Thermostat:





Dynamical system with both continuous and discrete state and evolution.

Also continuous state can jump discontinuously (state updates)

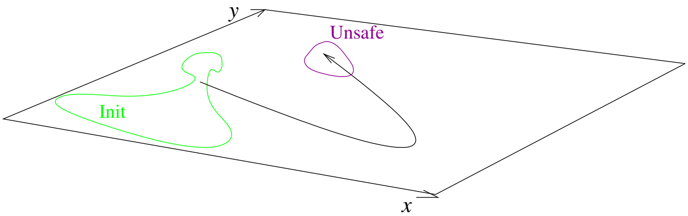
Non-linearity (differential equations, updates)

In illustrations: systems with just one control mode.

Motivation: embedded systems, motor gears, ...

System Correctness

Error trajectory: trajectory from initial to unsafe state



System is *correct (safe)* if it does not contain an error trajectory

Problem Definition

Observation:

- for ordinary differential equations forward reachability computation (as used in most verification algorithms) only with over-approximation.
- So: from this, no (systematic) detection of error trajectories

Problem Definition

Observation:

- for ordinary differential equations forward reachability computation (as used in most verification algorithms) only with over-approximation.
- So: from this, no (systematic) detection of error trajectories

So: design algorithm to detect incorrectness (*falsification* algorithm)

Illustration of the Problem

Illustration of the Problem

Assumptions:

- deterministic evolution: for a given initial state, unique trajectory

Illustration of the Problem

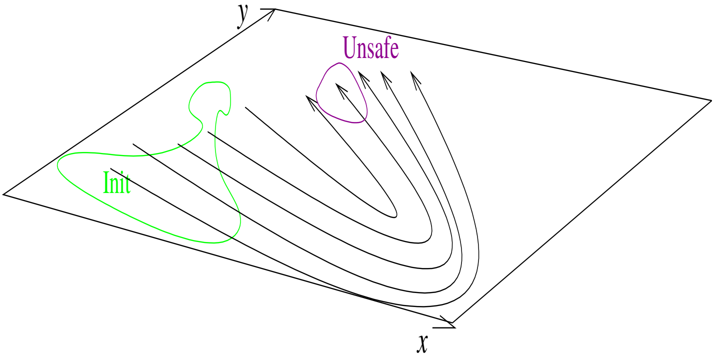
Assumptions:

- deterministic evolution: for a given initial state, unique trajectory
- bounded state space

Illustration of the Problem

Assumptions:

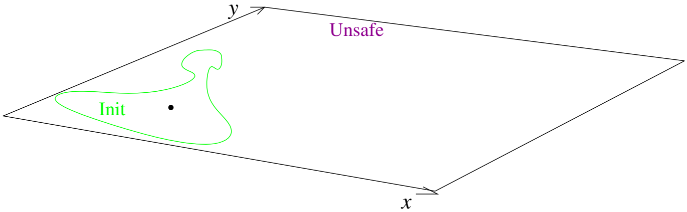
- deterministic evolution: for a given initial state, unique trajectory
- bounded state space



So, problem: finding a startpoint of an error trajectory.

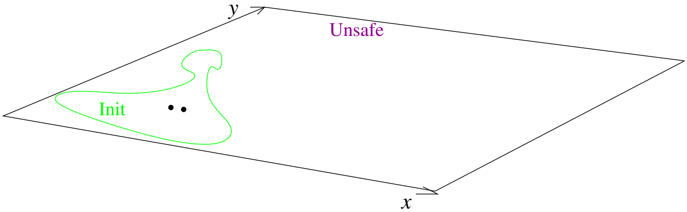
Computing trajectories

simulation



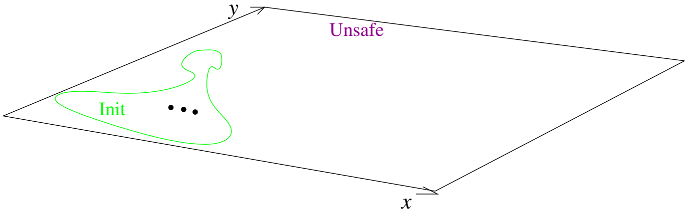
Computing trajectories

simulation



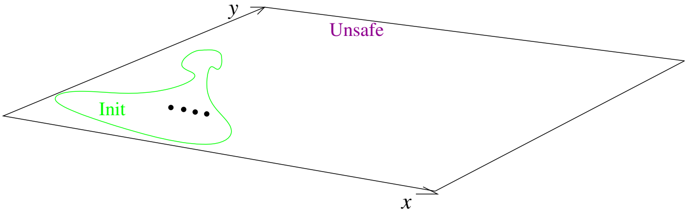
Computing trajectories

simulation



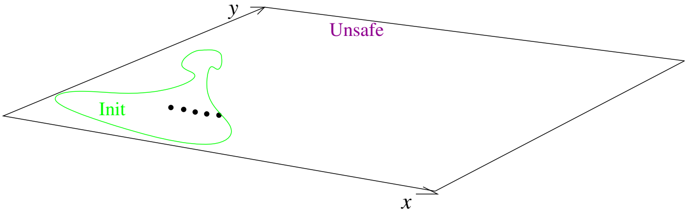
Computing trajectories

simulation



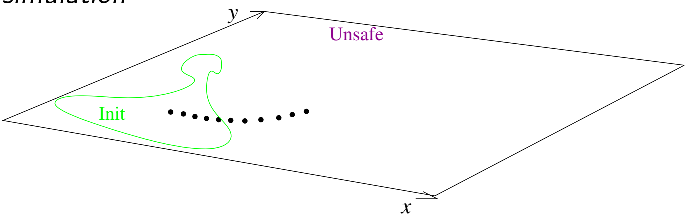
Computing trajectories

simulation



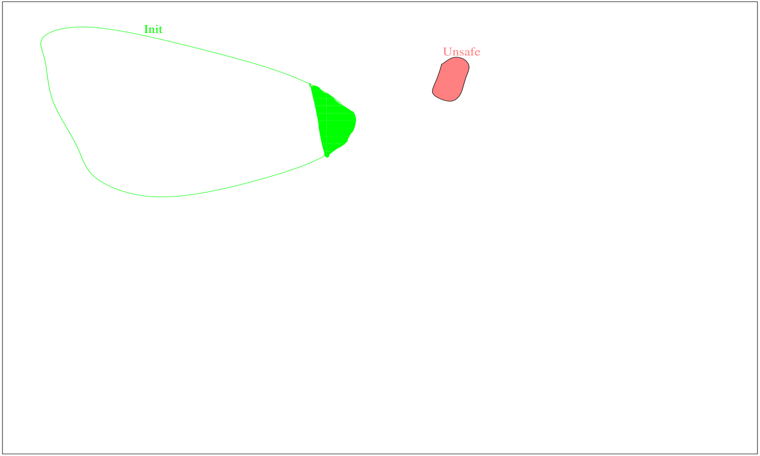
Computing trajectories

simulation



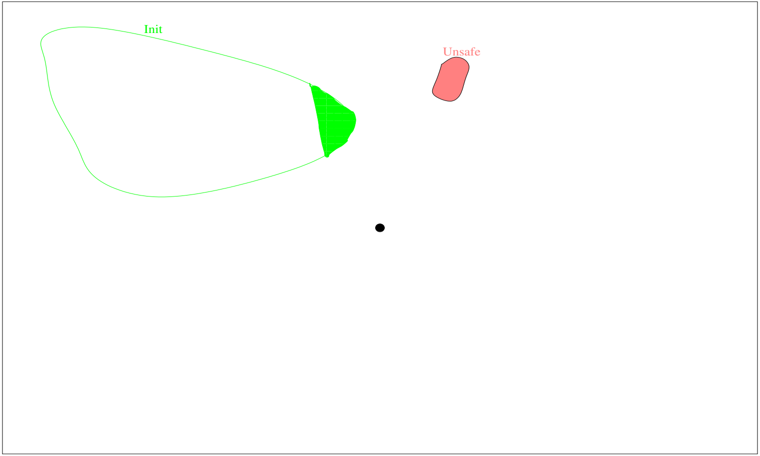
Naïve Method

starting points of simulations



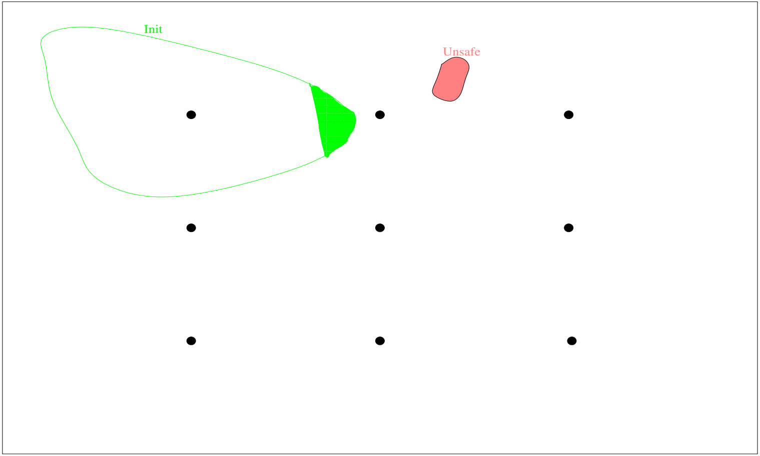
Naïve Method

starting points of simulations



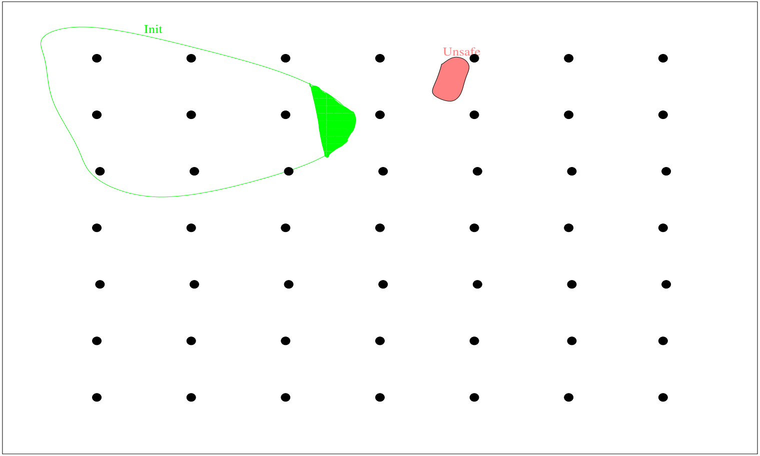
Naïve Method

starting points of simulations



Naïve Method

starting points of simulations



What to Do about the Naïve Method?

Naïve because:

- It runs forever on safe systems.
- It runs simulations evenly distributed on the whole statespace.
- Each individual simulation runs for a pre-determined amount of time.

What to Do about the Naïve Method?

Naïve because:

- It runs forever on safe systems.
- It runs simulations evenly distributed on the whole statespace.
- Each individual simulation runs for a pre-determined amount of time.

Therefore we will . . .

- . . . alternate verification and falsification cycles;
- . . . prefer the more promising simulations;
- . . . cancel simulations when they do not look promising anymore.

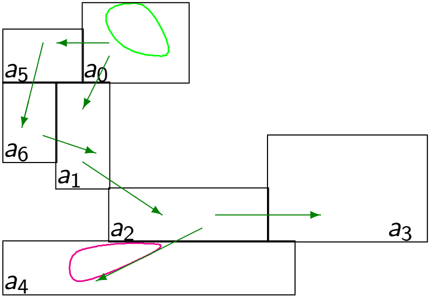
HSolver Abstraction

Verification tool: HSolver
(`http://hsolver.sourceforge.net/`)

HSOLVER Abstraction

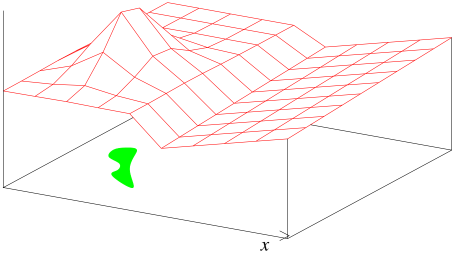
Verification tool: HSOLVER
(`http://hsolver.sourceforge.net/`)

- The statespace is partitioned into finitely many boxes.
- Interval arithmetic is used to compute the abstract transitions.
- Overapproximation used for verification.



- If necessary, the abstraction is refined by splitting a box.
- State space pruning

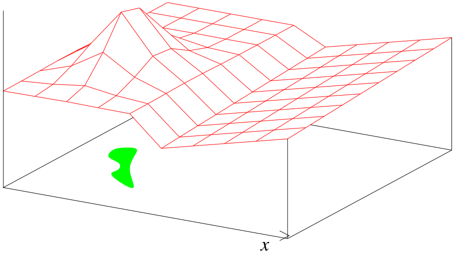
Our Method: Main Idea



Use real-valued *quality estimate* to approximate "given point is close to an initial point of an error trajectory".

Optimise the quality estimate.

Our Method: Main Idea



Use real-valued *quality estimate* to approximate "given point is close to an initial point of an error trajectory".

Optimise the quality estimate.

- How to define this function?
- How to find the optimum?

Defining the Quality Estimate

Overall approach:

- Start a simulation
- Compute closeness to error trajectory on the fly
- Cancel if no new information gained

Defining the Quality Estimate

Overall approach:

- Start a simulation
- Compute closeness to error trajectory on the fly
- Cancel if no new information gained

Problems: a-priori, length of error trajectories unbounded, and

- the longer we simulate, the more information about quality, but simulation costs
- a simulation that looks bad at the beginning, might turn out good much later

Defining the Quality Estimate

Overall approach:

- Start a simulation
- Compute closeness to error trajectory on the fly
- Cancel if no new information gained

Problems: a-priori, length of error trajectories unbounded, and

- the longer we simulate, the more information about quality, but simulation costs
- a simulation that looks bad at the beginning, might turn out good much later

Solution: Use information from abstraction, s.t. fine enough abstraction will result in reliable quality estimate

Closeness to Error Trajectory

A simulation is close to an error trajectory iff

- its first point is initial

Closeness to Error Trajectory

A simulation is close to an error trajectory iff

- its first point is initial
- it stays inside of abstraction as much as possible

Closeness to Error Trajectory

A simulation is close to an error trajectory iff

- its first point is initial
- it stays inside of abstraction as much as possible
- it gets close to unsafe state

Closeness to Error Trajectory

A simulation is close to an error trajectory iff

- its first point is initial
- it stays inside of abstraction as much as possible
- it gets close to unsafe state

Note: leaving abstraction means "no error trajectory"

Closeness to Error Trajectory

A simulation is close to an error trajectory iff

- its first point is initial
- it stays inside of abstraction as much as possible
- it gets close to unsafe state

Note: leaving abstraction means "no error trajectory"

But: there might still be an error trajectory nearby

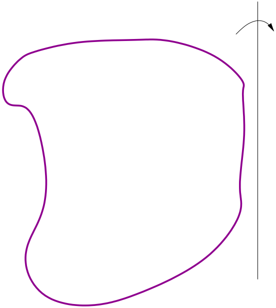
Closeness to Unsafe State

Maximal closeness of any individual simulation point

Closeness to Unsafe State

Maximal closeness of any individual simulation point

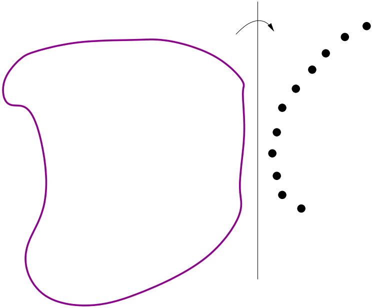
This is not
Euclidean closeness



Closeness to Unsafe State

Maximal closeness of any individual simulation point

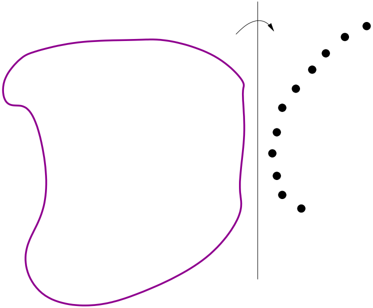
This is not
Euclidean closeness



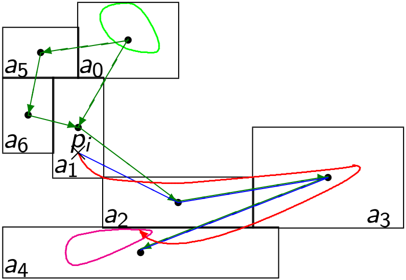
Closeness to Unsafe State

Maximal closeness of any individual simulation point

This is not
Euclidean closeness



Measure closeness using
abstraction



Cancellation Strategy

Goal: Cancel if no interesting new information gained

Cancellation Strategy

Goal: Cancel if no interesting new information gained

Problem: based on future (when new information might be gained)

Cancellation Strategy

Goal: Cancel if no interesting new information gained

Problem: based on future (when new information might be gained)

Cancel if

- unsafe state hit,
- outside of abstraction for too long, or
- no improvement of quality for too long.

"too long": parameter *sim_cnc*

Cancellation Strategy

Goal: Cancel if no interesting new information gained

Problem: based on future (when new information might be gained)

Cancel if

- unsafe state hit,
- outside of abstraction for too long, or
- no improvement of quality for too long.

"too long": parameter *sim_cnc*

Observation: last two items improve with abstraction

Cancellation Strategy

Goal: Cancel if no interesting new information gained

Problem: based on future (when new information might be gained)

Cancel if

- unsafe state hit,
- outside of abstraction for too long, or
- no improvement of quality for too long.

"too long": parameter *sim_cnc*

Observation: last two items improve with abstraction

Hope: abstraction eventually good enough for reliable strategy

Optimising the Quality Estimate

From boxes that might contain initial states,
start numerical local optimisation.

Optimising the Quality Estimate

From boxes that might contain initial states,
start numerical local optimisation.

Numerical optimisation usually needs derivatives.

Optimising the Quality Estimate

From boxes that might contain initial states,
start numerical local optimisation.

Numerical optimisation usually needs derivatives.

Not available! *direct search* methods

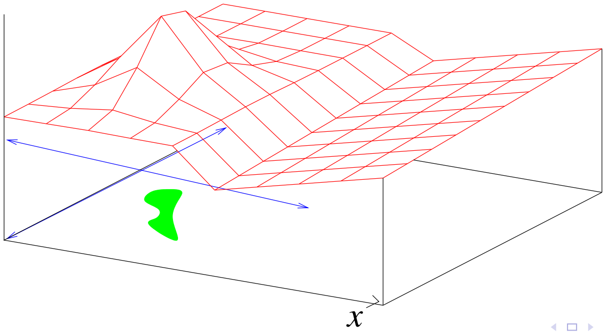
Optimising the Quality Estimate

From boxes that might contain initial states,
start numerical local optimisation.

Numerical optimisation usually needs derivatives.

Not available! *direct search* methods

Compass method



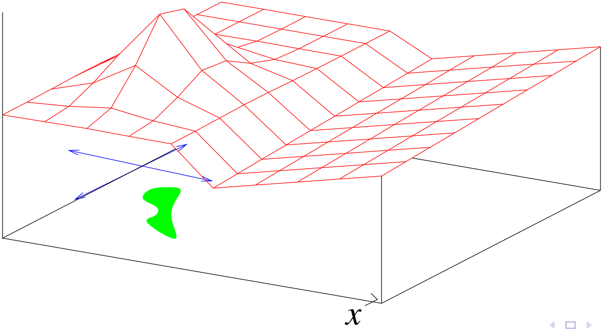
Optimising the Quality Estimate

From boxes that might contain initial states,
start numerical local optimisation.

Numerical optimisation usually needs derivatives.

Not available! *direct search* methods

Compass method



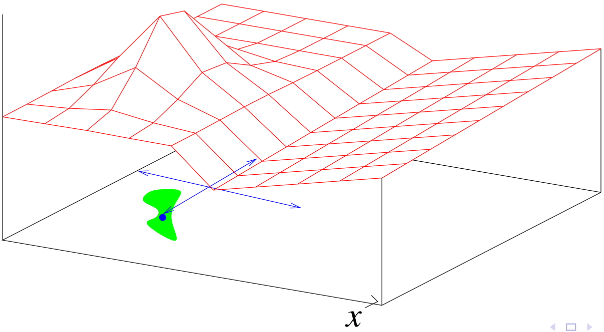
Optimising the Quality Estimate

From boxes that might contain initial states,
start numerical local optimisation.

Numerical optimisation usually needs derivatives.

Not available! *direct search* methods

Compass method



Experiments

	our algorithm				naïve algorithm	
Example	*sim_cnc*	time	ref.	sim.	time	sim.
convoi	200	0.04	0	1	∞	∞
eco	400	0.1	0	1	0.1	1
eco	200	2.1	10	63	0.1	1
focus	200	0.1	0	10	0.04	1
focus	20	29.7	434	288	0.04	1
parabola	105	0.0	0	1	∞	∞
parabola	30	18.0	353	113	∞	∞

Jumps

- For simplicity, we did not explain here how the quality estimate is defined in the presence of jumps.
- Our current implementation did find error trajectories with up to 2 (necessary) jumps.
- Encouraging first result, but topic for future work.

Conclusion

Main Observations:

- Local search can help to find error trajectories.
- Even for too small value of *sim_cnc*, simulations will eventually "survive" long enough thanks to the refinement of the abstraction and improving faithfulness of the quality function.

Conclusion

Main Observations:

- Local search can help to find error trajectories.
- Even for too small value of *sim_cnc*, simulations will eventually "survive" long enough thanks to the refinement of the abstraction and improving faithfulness of the quality function.

Future work:

- More mathematical intelligence (e.g., derivatives)
- Reasoning forward and backward
- Non-deterministic evolution