From paths to traces: discussion about Eulerian discretization

Damien Massé
with Luc Jaulin and Thomas Le Mézo

LabSTICC
Université de Bretagne Occidentale
Brest, France

MRIS seminar

Outline

Differential inclusion, invariant sets

Let's consider a (deterministic) differential equation (with $x \in \mathbb{R}^n$ a state vector):

$$\dot{x} = f(x)$$

or a (non-deterministic) differential inclusion:

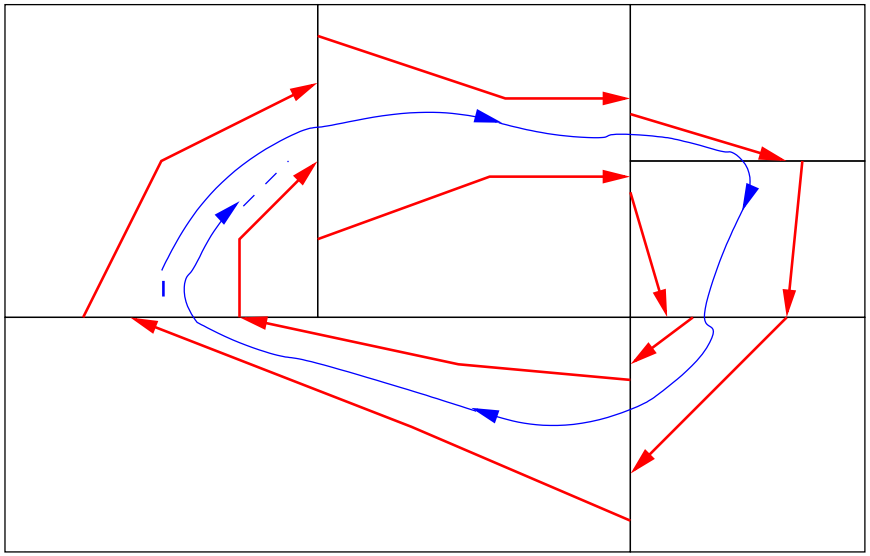
$$\dot{x} \in F(x)$$

We consider solutions of these equations in $\mathbb{R}^+ \to \mathbb{R}^n$ (*trajectories*), and define:

- a *positive invariant* **B** is a set of states from which all trajectories stay in **B**;
- a *capture basin* **C** of a target **T** is a set is states from which at least one trajectory goes to **T**;
- a *viability kernel* **K** is a set of states from which at least one trajectory stays in **K**.

Eulerian approach

Eulerian approach: decompose the state space and approximate the sub-paths restricted to each subset of the state space.



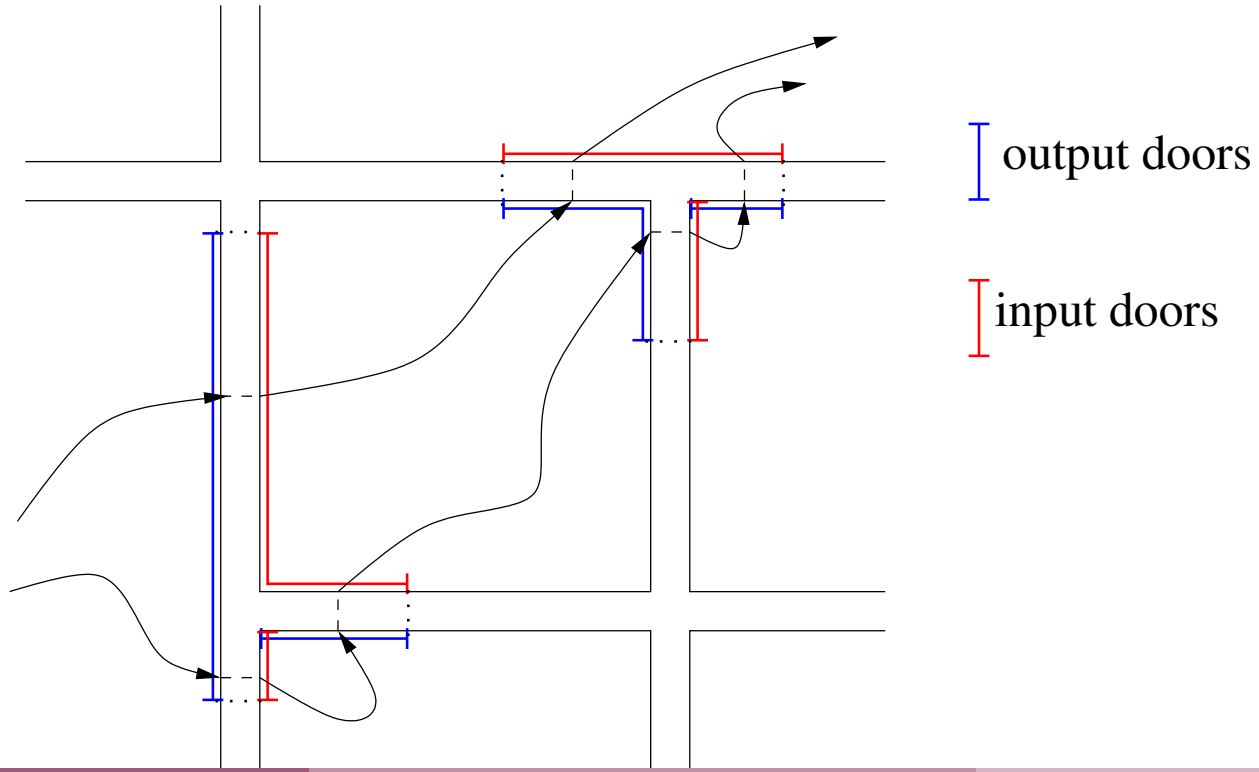
A simpler approach (proposed by Luc Jaulin and Thomas Le Mézo) would be to represent only the transitions between each subset.

Maze (Luc Jaulin and Thomas Le Mézo)

The modelisation (abstraction) of the state space uses a *paving* $\mathcal{P}$ of boxes (which can be bisected to increase the precision), and doors at the boundary of each box:

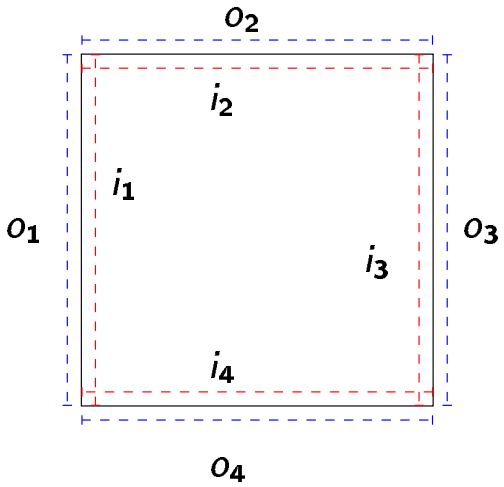
- *input doors* for ingoing oriented paths;
- *output doors* for outgoing oriented paths.

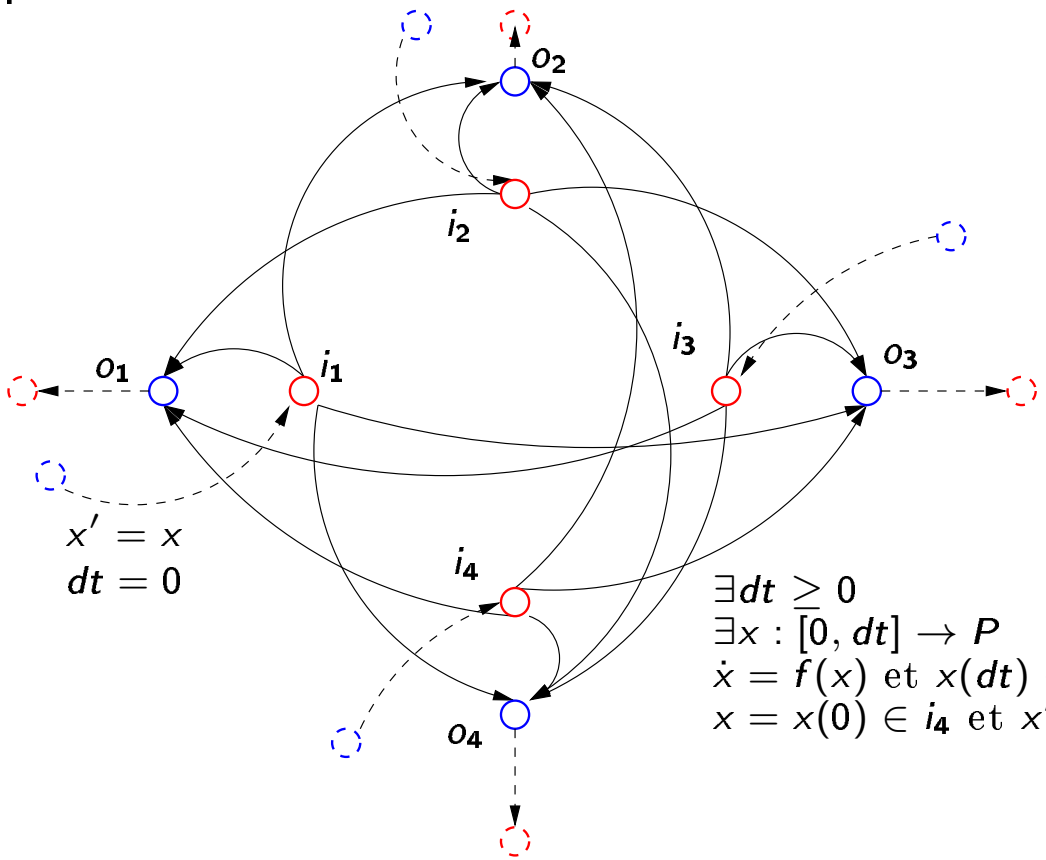
With a valuation of the doors, the result is a *maze*.



Maze and control flow graph

Given the differential equation/inclusion and a paving gives something similar to a control flow graph, where each door is a vertex:

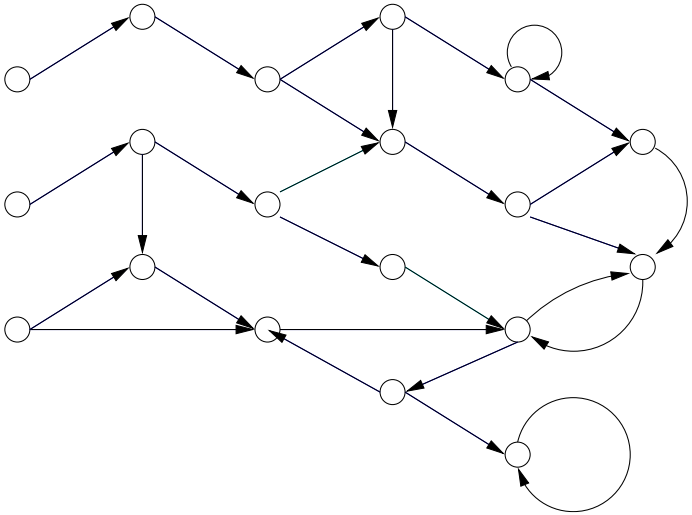




The result is a **discretization approach** with arbitrary time between transitions from input to output doors (transitions from output to input doors are timeless).

Program semantics

Static analysis by abstract interpretation consider programs as transition systems over an *infinite* set of states Σ (to simplify, we consider the transition relation τ to be *total*).



Note: here, one vertex = one state, not one program point.

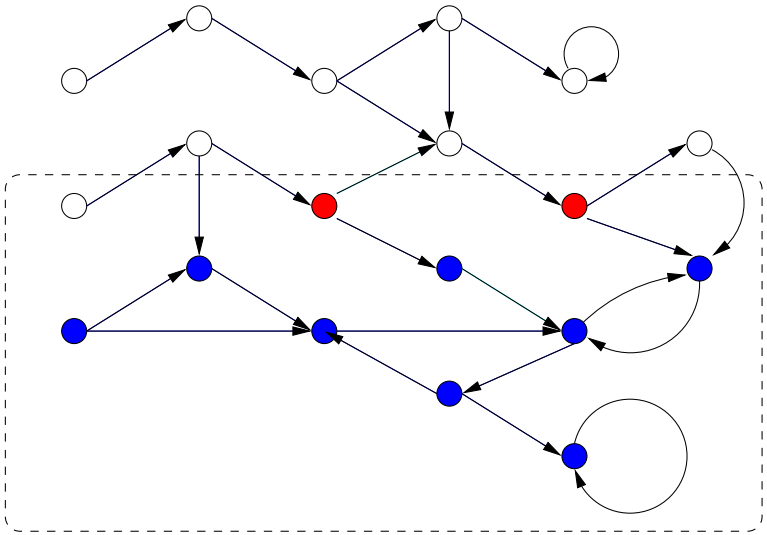
A *trace* is an (infinite) sequence of successive states (i.e. in $\mathbb{N} \to \Sigma$). The discretization approach assumes the existence of a relation between traces and oriented paths.

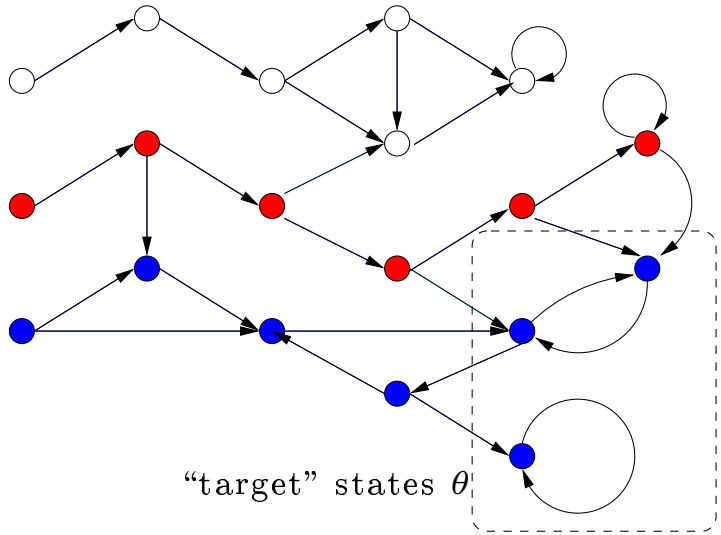
Temporal properties

Properties generally analysed are *temporal properties* (reachability, termination...), computed for set of traces.
Examples of **state-based temporal properties** (and their CTL expression):

- states from which one trace leads to θ (**EF**θ) ($\Rightarrow$ "capture basin").
- states from which all traces lead to θ (**AF**θ);
- states from which there exists one trace staying in σ (**EG**σ) ($\rightarrow$ "viability kernel").
- states from which all traces stay in σ (**AG**σ) ($\rightarrow$ "positive invariant").

Example





AGσ

EGσ (+blue)

AFθ

EFθ (+blue)

Note that $\overline{\mathbf{EF}\theta} = \mathbf{AG}\overline{\theta}$ and $\overline{\mathbf{AF}\theta} = \mathbf{EG}\overline{\theta}$.

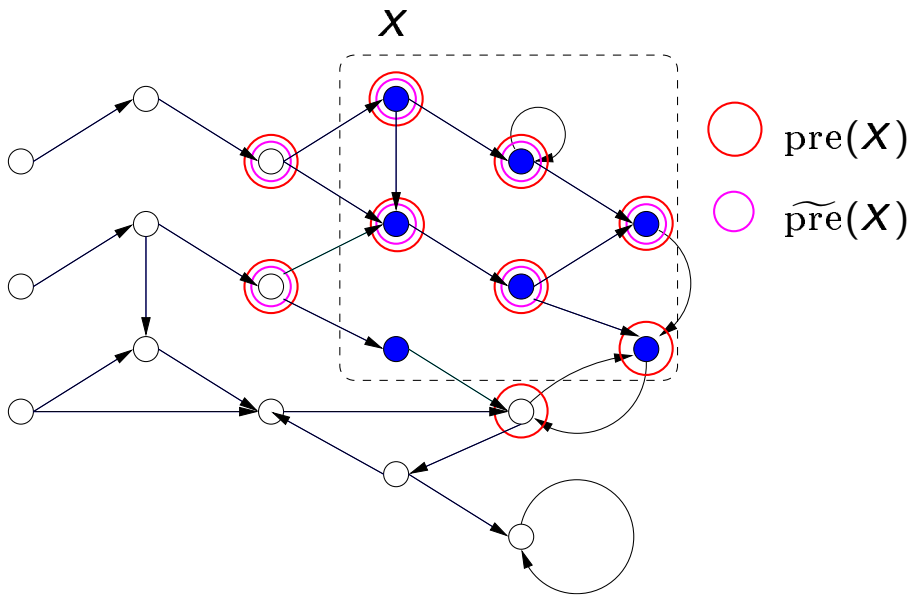
All these properties can be expressed using **fixpoint semantics** over monotone operators (called *predicate transformers*).

Predicate transformers

Two operators pre and $\widetilde{\mathrm{pre}}$ on $\wp(\Sigma)$ are commonly used:

$$\begin{aligned}\mathrm{pre}(X) &= \{s \mid \exists s' \in X, s \xrightarrow{\tau} s'\}\\ \widetilde{\mathrm{pre}}(X) &= \{s \mid \forall s' \in \Sigma, s \xrightarrow{\tau} s' \Rightarrow s' \in X\}\end{aligned}$$

- pre and $\widetilde{\mathrm{pre}}$ are monotonic (note that $\widetilde{\mathrm{pre}}$ is anti-monotone w.r.t. τ);
- $\forall X, \widetilde{\mathrm{pre}}(X) = \Sigma \setminus \mathrm{pre}(\Sigma \setminus X)$.
- when τ is *deterministic*, $\mathrm{pre} = \widetilde{\mathrm{pre}}$;



Fixpoint semantics

If s satisfies $\mathbf{EF}\theta$, then its predecessors satisfy $\mathbf{EF}\theta$. So $\mathbf{EF}\theta$ is (almost) a **fixpoint** of pre.

More precisely:

- $\mathbf{EF}\theta$ is the smallest set (*least fixpoint*) stable by the function $X \mapsto (\theta \cup \mathrm{pre}(X))$:

$$\mathbf{EF}\theta \;=\; \mathrm{lfp}(\theta \cup \mathrm{pre})$$

- $\mathbf{EG}\sigma$ is the largest set (*greatest fixpoint*) stable by the function $X \mapsto (\sigma \cap \mathrm{pre}(X))$:

$$\mathbf{EG}\sigma \;=\; \mathrm{gfp}(\sigma \cap \mathrm{pre})$$

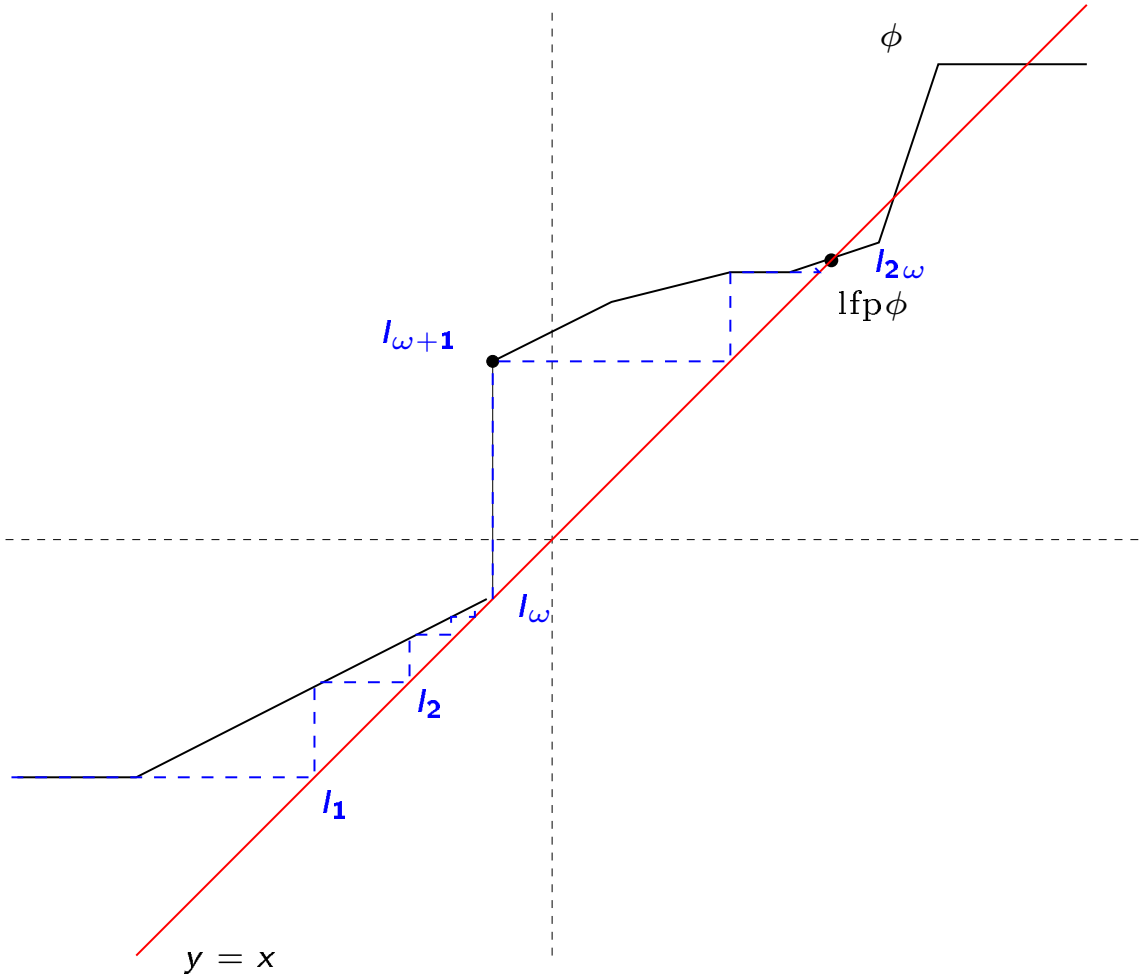
$\mathbf{AF}\theta$ and $\mathbf{AG}\sigma$ are similar, using $\widetilde{\mathrm{pre}}$:

$$\mathbf{AF}\theta \;=\; \mathrm{lfp}(\theta \cup \widetilde{\mathrm{pre}})$$

$$\mathbf{AG}\sigma \;=\; \mathrm{gfp}(\sigma \cap \widetilde{\mathrm{pre}})$$

Construction of fixpoint semantics

The common constructive approach for fixpoints use the constructive form of the *Knaster-Tarski theorem*, e.g., starting with $\emptyset$ (of Σ), to repeatedly apply the function ("Kleene iterations") until convergence.



However:

- convergence is not guaranteed, even after ω iterations;
- the fixpoint may not (even) be memory representable.

Abstraction of fixpoint semantics

Using abstract domains, one can safely *over*-approximate the fixpoint semantics, e.g. with $\mathrm{pre}^\sharp(x) \supseteq [\mathrm{pre}[x]]$:

$$\mathrm{lfp}(\theta \cup \mathrm{pre}^\sharp) \supseteq \mathrm{lfp}(\theta \cup \mathrm{pre})$$

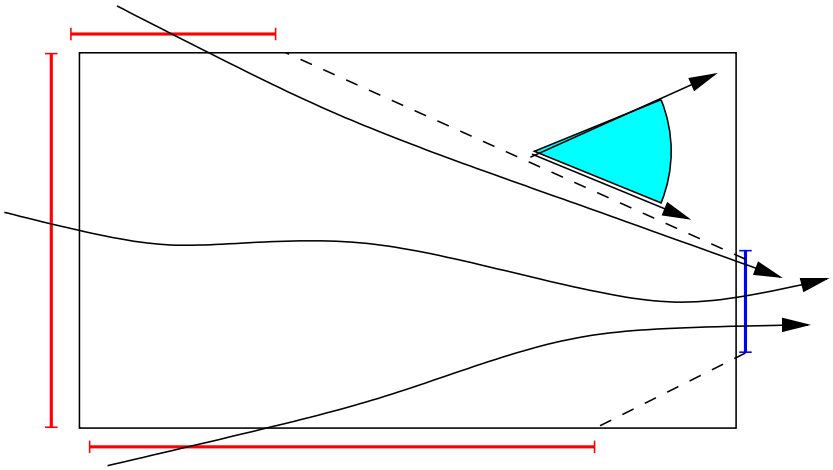
Abstract domains use (almost always) over-approximations. Hence *under*--approximations requires to compute the complement, e.g.:

$$\mathrm{lfp}(\theta \cup \mathrm{pre}) \supseteq \Sigma \setminus (\mathrm{gfp}(\overline{\theta} \cap \widetilde{\mathrm{pre}}^\sharp))$$

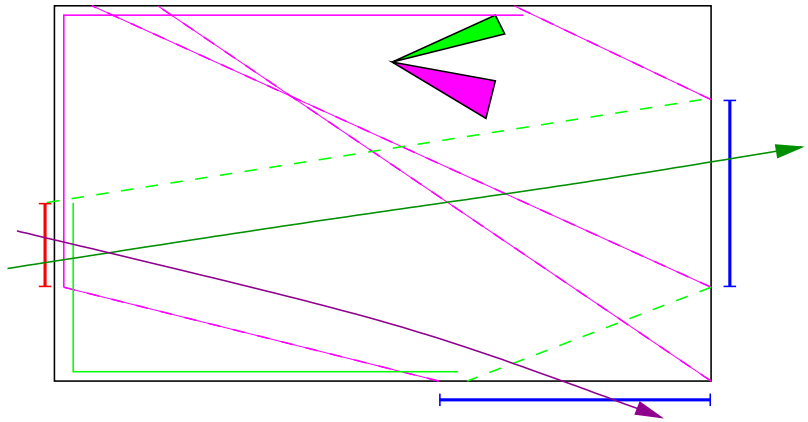
Now the (abstract) fixpoints are memory representable, but the convergence is still not guaranteed.

Abstraction of the predicate transformers

An simple abstraction uses cones to represent all possibles directions of $f(x)$ inside a box $[x]$.



abstract pre operator (from blue to red)



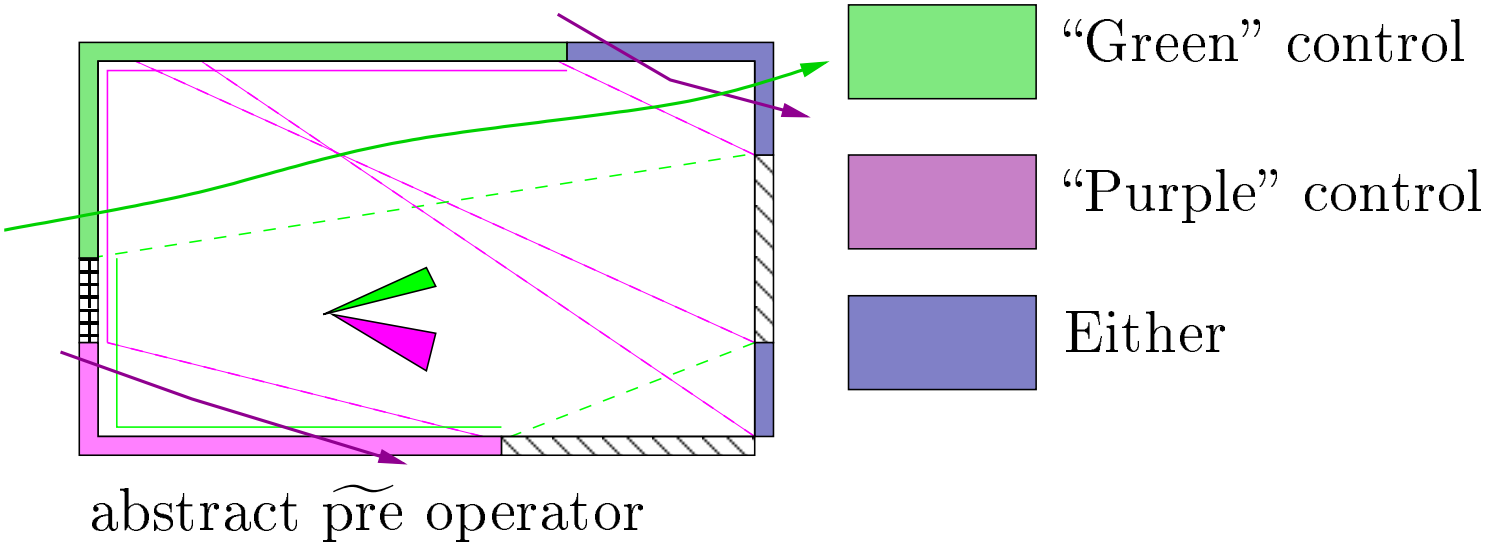
abstract $\widetilde{\text{pre}}$ operator

To abstract $\widetilde{\text{pre}}$, we restrict the differential inclusion to a finite set of functions (controls). This restriction is itself an overapproximation (for the $\widetilde{\text{pre}}$ operator).

The approximated transition relation defines an **affine program**: transitions between states follow affine (polyhedral) constraints (of the form $A \cdot \begin{pmatrix} X \\ X' \end{pmatrix} \leq B$).

Abstraction of $\widetilde{\mathrm{pre}}$ and viability kernel

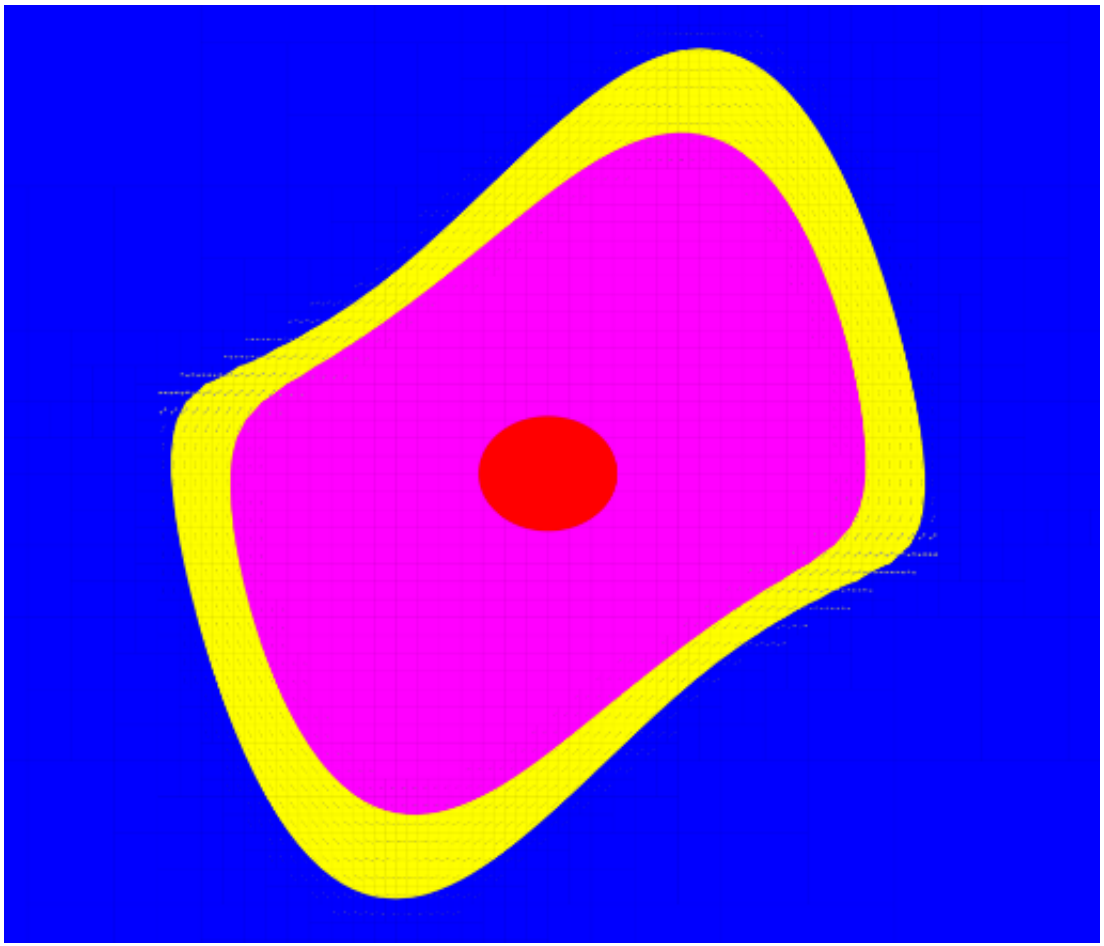
An **under-approximation** of the viability kernel is computed by abstracting $\mathrm{lfp}(\theta \cup \widetilde{\mathrm{pre}})$ (i.e. what is computed is in fact an over-approximation of the complement). The abstract $\widetilde{\mathrm{pre}}$ operator "selects" a possible "control" to ensure staying *outside* the maze (hence *inside* the viability kernel).



abstract $\widetilde{\mathrm{pre}}$ operator

In this approach, the "control" is fixed at the entrance of each box. Bisection must be used to increase the possibilities.

And so...

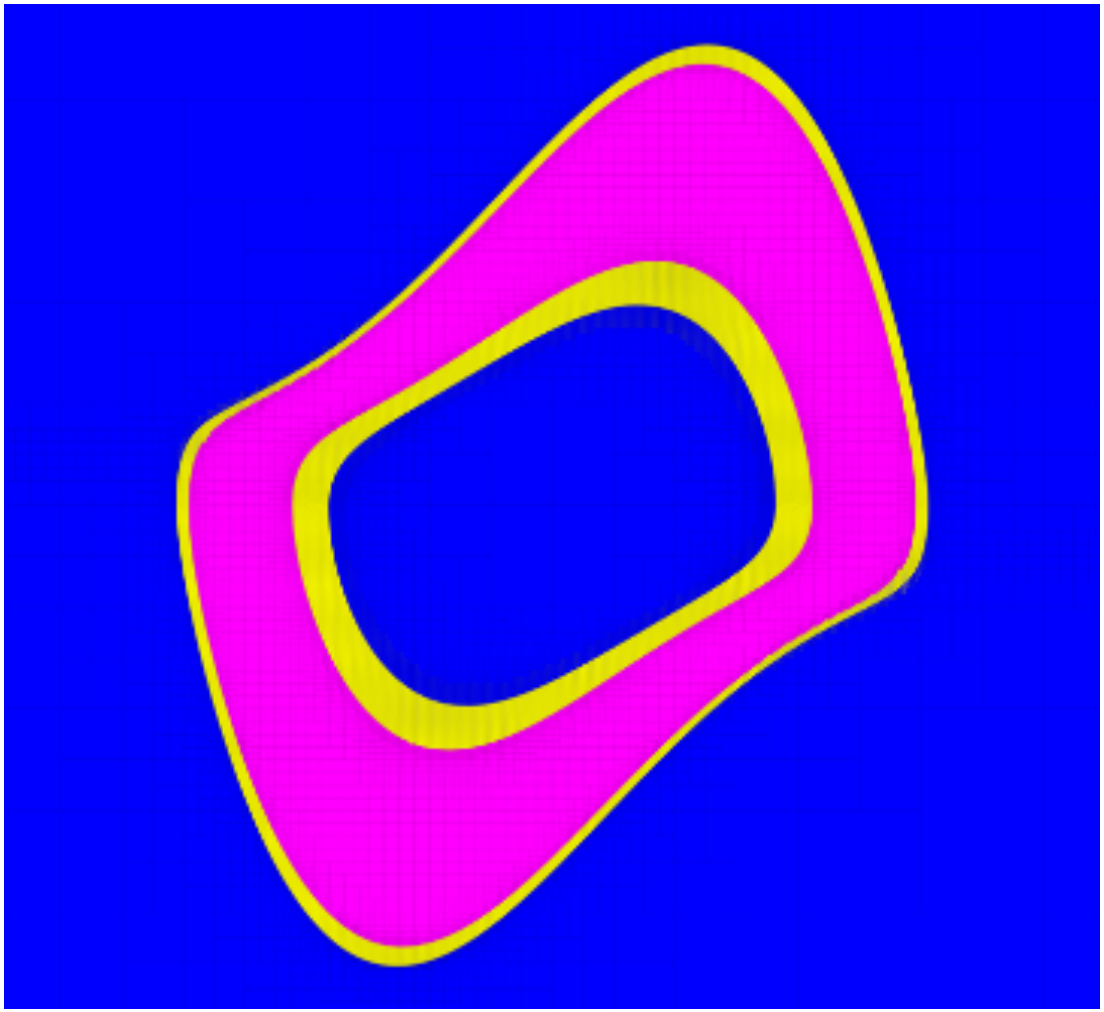


Van Der Pol basin of capture (target θ in red), deterministic:

- underapproximation by overapproximating $EG\overline{\theta}$ (greatest fixpoint, starting from the negation of the target);
- overapproximation using $EF\theta$ (least fixpoint, starting from the target)

Note: bisection is applied on the yellow zone to increase the precision.

And so (2)...



Van Der Pol viability kernel with two controls, σ being the whole square without the center:

- underapproximation by overapproximating $AF\overline{\sigma}$ (least fixpoint using $\widetilde{\mathrm{pre}}$);
- overapproximation using $EG\sigma$ (greatest fixpoint using pre).

Current experiments

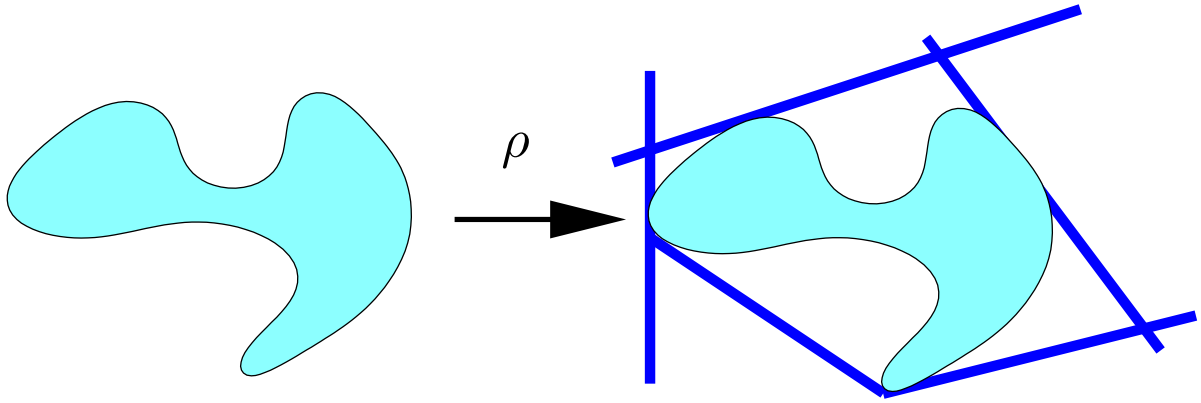
Extension to 3D: doors are two-dimensional, hence the discussion about which abstract domain can be used?

- with intervals, the convergence is fast but the result seems too imprecise;
- with polyhedra, widening/narrowing are needed (for lfp) even for a "simple" (theoretically acyclic) system $\Rightarrow$ questions about the positioning/use of widening points;
- restrictions of polyhedra (template polyhedral domains).

Template polyhedral domains[Sankaranarayanan *et al.*,2005]

Abstract domain: **template polyhedral domains**.
Example with:

$$T = \begin{pmatrix} -1 & 0 \\ -1 & 3 \\ 4 & 3 \\ 1 & -4 \\ -2 & -3 \end{pmatrix}$$



Advantages: possible to customize the template.
Drawbacks: in general, linear programming must be used to compute convex union, intersection.
Boxes and octagons are particular cases.

Relation paths/traces

The whole computation relies on a (assumed) relation between the semantics of the graph (traces) and the semantics of the differential equation:

1. every oriented path can be associated to an (infinite) trace;
2. every trace can be associated to an oriented path.

Unfortunately, this assumption does not hold: a path may be associated to a finite trace, and infinite traces may represent sub-path (on limited time).

From paths to traces

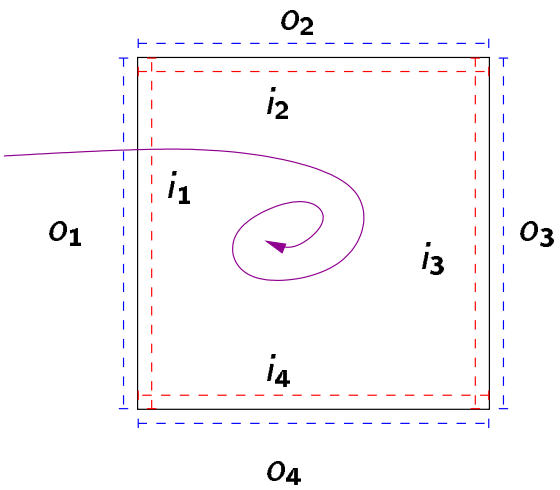
Path: $p : \mathbb{R}^+ \longrightarrow \mathbb{R}^n$.
Doors (D_k) represent closed subsets of $\mathbb{R}^n$. Each door is included in (at least) two (closed) boxes of the paving.
For each k, we denote $\delta_k \subseteq \mathbb{R}^k$ the boundary of $p^{-1}(D_k)$, and $\delta = \cup_k \delta$.
Then:

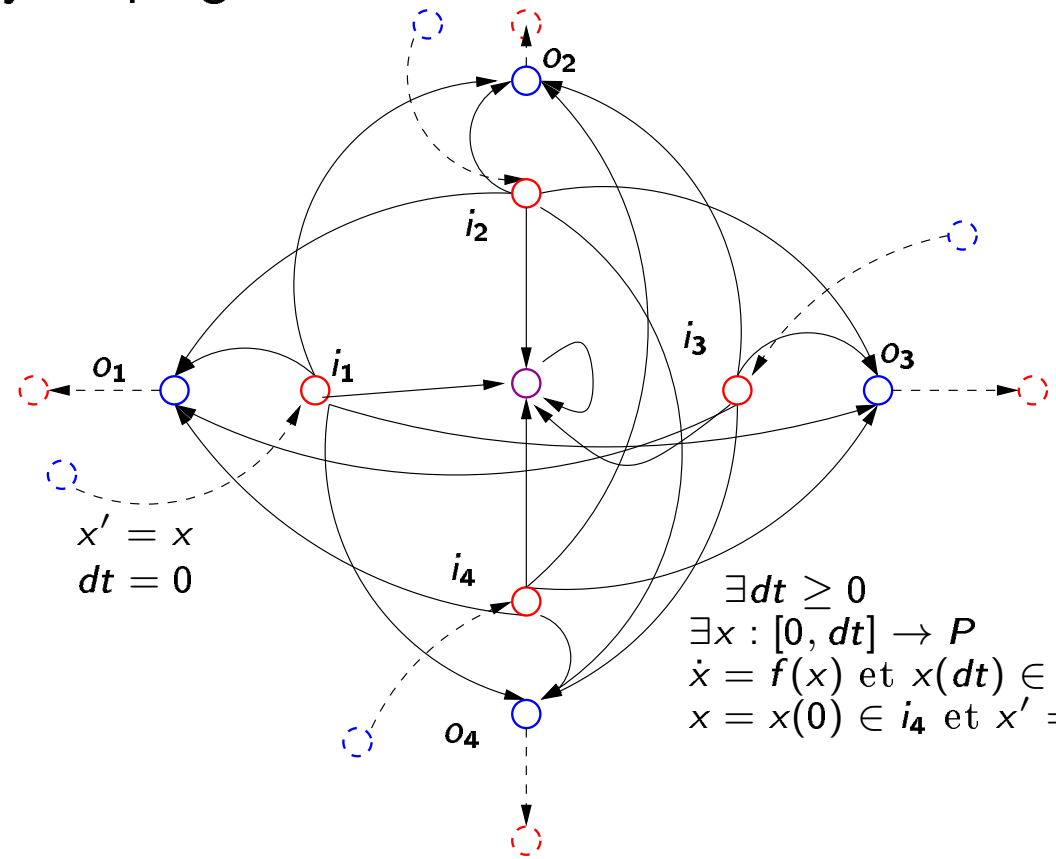
- δ is closed and its interior is empty;
- on each component of $\mathbb{R}^n \setminus \delta$, p is included in one box (thus, each component of $\mathbb{R}^n \setminus \delta$ appears as a "transition" of a trace);
- for each $t \in \delta$, one can find a finite timeless subtrace from the box associated to $\pi(t^-)$ to the one associated to $\pi(t^+)$.

Hence, p can be associated to an infinite trace if δ can be ordered into an infinite sequence (or if δ is infinite and has no accumulation point).

When δ is finite

Easy case: when a path does not cross an infinite number of doors, it means that it "terminates" inside a box.
$\Rightarrow$ we just add a new infinitely looping state for this box.

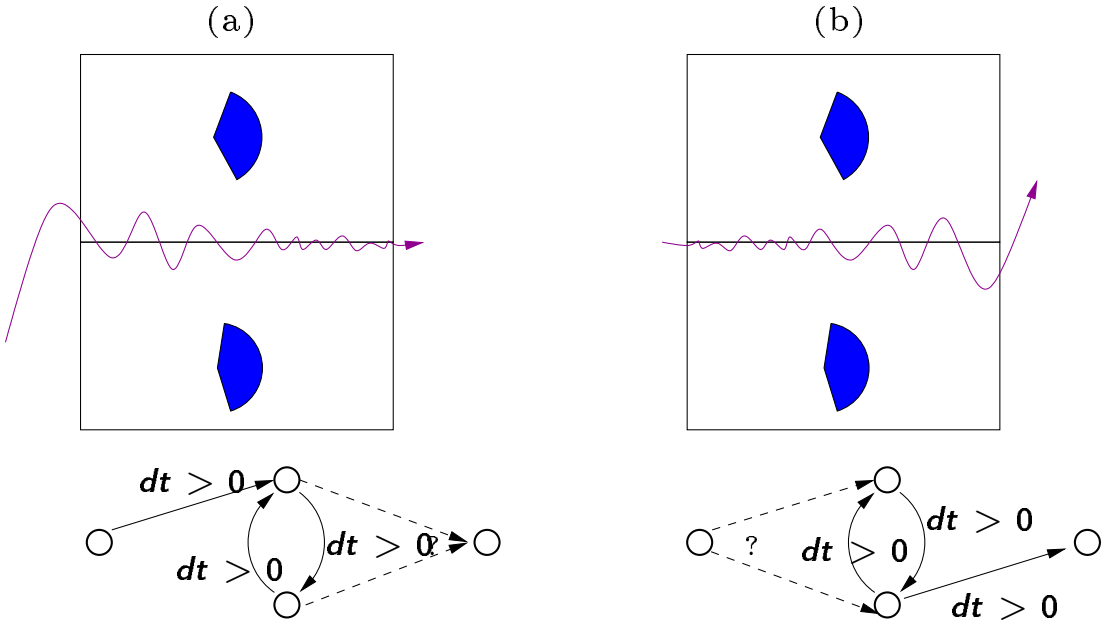




Boxes with this state are those for which the "cone" has opposite vectors.

When δ has accumulation points

Accumulation points have an infinite number of points before (a) and/or after (b). No (timed) transition can be defined before (or after) one.

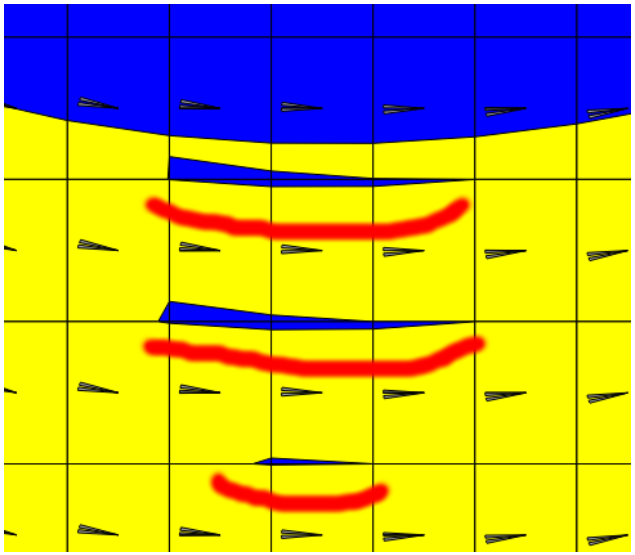


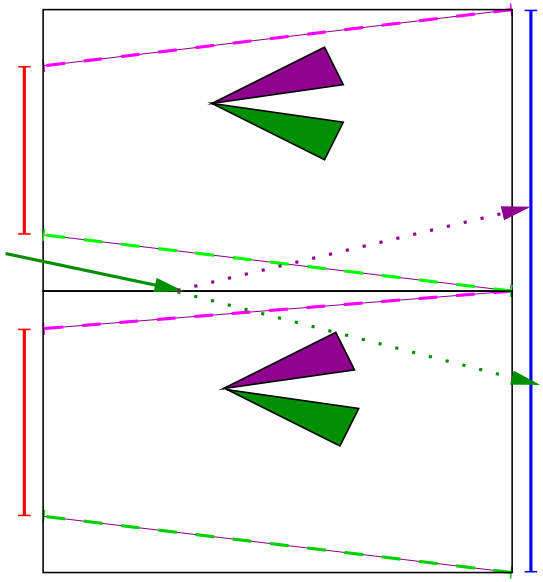
Not only these paths may (exceptionally) exist, but some are "created" by the abstraction of the predicate transformer. We can discard them by assuming that every point of δ must have a predecessor and a successor:

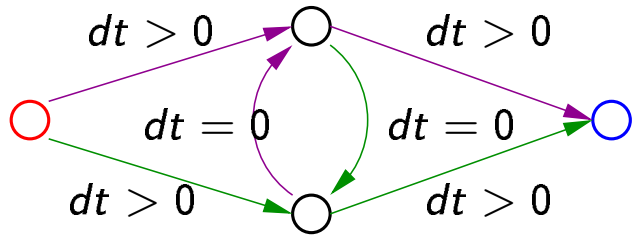
- "real" paths will probably have a "normal" neighbour, hence a closed set abstractions can "capture" missing points;
- and it prevents associating an infinite trace to a finite path.

Special case of infinite timeless sequence

Our transition system also admits many timeless cycles, which generate infinite timeless traces. This problem appears specifically with $\mathrm{lfp}\,\widetilde{\mathrm{pre}}$ analysis, where these cycles generate false results.







No obvious way to solve this problem, especially when the dimension increases.

Unsoundness

EFt $(\mathrm{lfp},\mathrm{pre})$
- Unsound when t is reachable from a state x only by "pathological paths".
- Unlikely (even more with closed abstractions).

EGσ $(\mathrm{gfp},\mathrm{pre})$
- Unsound if the only paths from a state x staying in σ cannot be associated (at least partially) to an infinite trace.
- Quite impossible.

AFt $(\mathrm{lfp},\widetilde{\mathrm{pre}})$
- Unsound if from x all paths lead to t, but some infinite traces do not.
- Common (with timeless loops).

AGσ $(\mathrm{gfp},\widetilde{\mathrm{pre}})$
- Unsound if from x all paths stays in σ, but an infinite trace does not.
- Impossible.

The potential unsoundness of **AF**t analyses implies that **EG**σ analyses may be incomplete.

Convergence issues

To summarize, two theoretical problems with practical consequences:

- problem with the convergence of fixpoint computations (though it seems to work with intervals);
- spurious traces are taken into account.

Removing these traces would mean getting a fixpoint (the set is still stable by the predicate operators), but *not* the least (or greatest) fixpoint (trying to get "more than infinite" traces).
In this case, Kleene iterations are not usable (they cannot go "past" a fixpoint), so is there other methods to compute fixpoints?

Policy iteration

Policy iteration (in the context of abstract interpretation) is a technique which uses convex optimization to compute *exact* abstract fixpoint for specific kind of programs, in finite time.
More specifically, for **affine programs** (checked) and **template polyhedral abstract domain** (checked), one can, using linear programming:

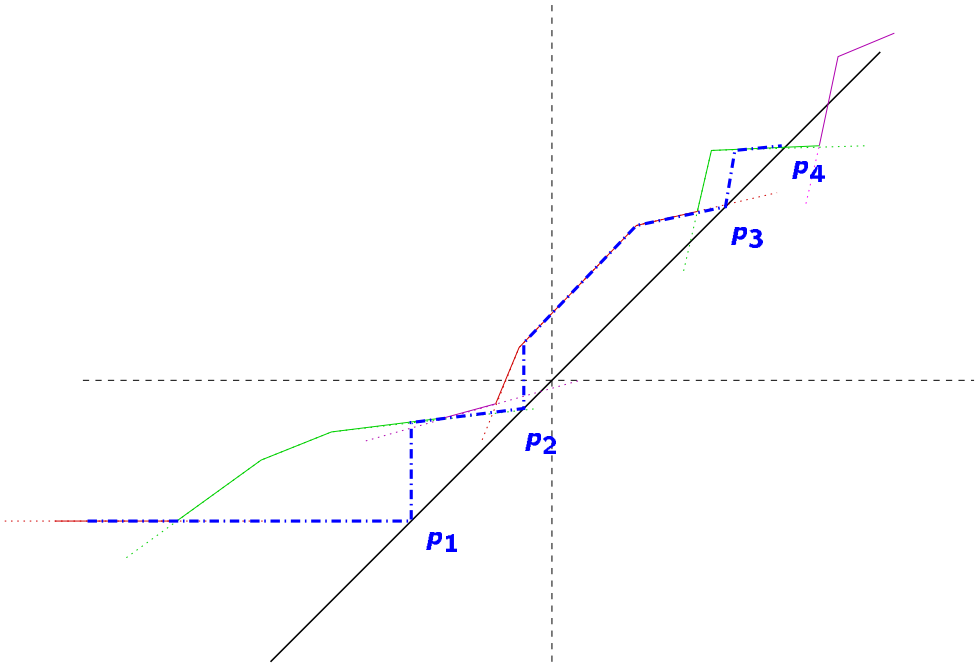
- compute the *least fixpoint* of the $\mathrm{pre}^\sharp$ operator in finite time [Gawlitza and Seidl, 2007];
- and also the *greatest fixpoint* of the $\mathrm{pre}^\sharp$ operator in finite time [Massé, 2012].

The extension to $\widetilde{\mathrm{pre}}^\sharp$ is quite easy.

Policy iteration (intuition for the lfp)

Policy iteration can be seen as an extension of Newton method. Let's consider a monotonic function $f : \mathbb{R} \to \mathbb{R}$ with $f = \max\ f_i$ where each f_i is **concave** and the fixpoints of f_i are computable (f_i are the *policies*). Then:

- each fixpoint of f is a fixpoint of (at least one) f_i;
- the least fixpoint of f is computable.



- p_1, p_2, p_3 and p_4 are successive policy fixpoints, creating an increasing chain of pre-fixpoints;
- no policy can be selected twice;
- the algorithm stops when it reaches a fixpoint.

Policy iteration extends this idea to $\mathbb{R}^n$.

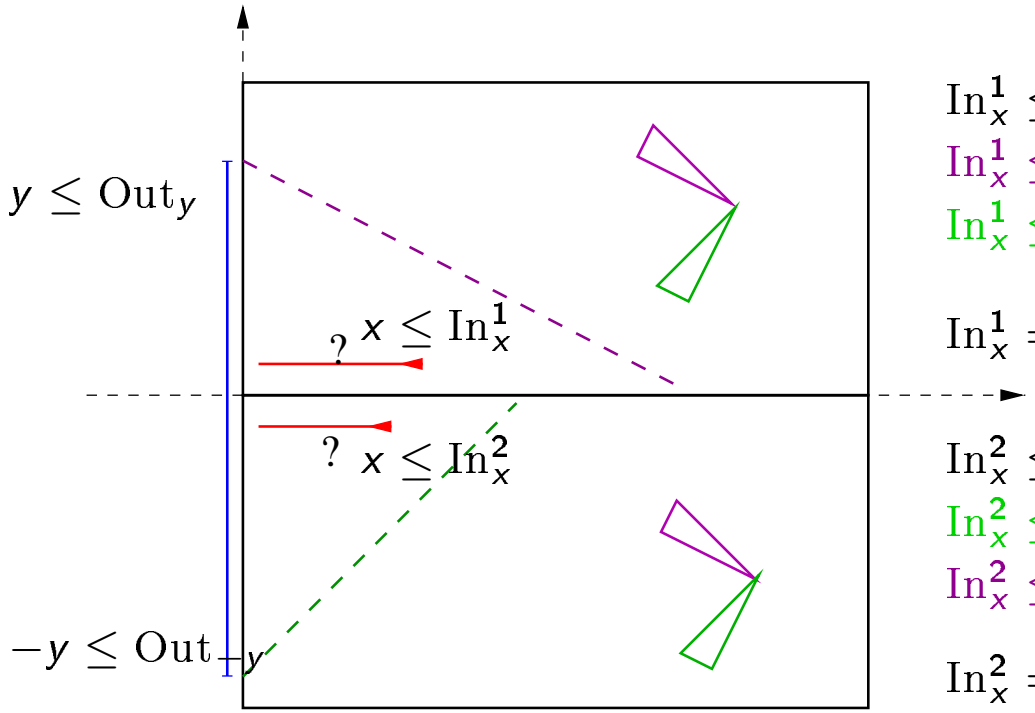
Policy iteration ($\widetilde{\mathrm{pre}}$)

The abstract domain associates to each door d several (abstract) variables $x_1^d, \ldots$. Predecessor operators can be translated as a set of equations of the form $x_i^d := e$ with:

$$e := a \mid x_i^d \mid \min(e, \ldots, e) \mid \max(e, \ldots, e) \mid \mathrm{LP}_{A,b}(e, \ldots, e)$$

min appears with different "controls" (meet operations), max with different doors (join operations). $\mathrm{LP}_{A,b}$ represent linear programs and abstract transitions (and can be replaced as min operations of affine expressions). Note that min (and $\mathrm{LP}_{A,b}()$) are **concave** whereas max is convex. Policy selection selects expressions between max operations.

Simple example



$$\mathrm{In}_x^1 \leq 2$$
$$\mathrm{In}_x^1 \leq 2\mathrm{Out}_y$$
$$\mathrm{In}_x^1 \leq \max(0, \mathrm{In}_x^2)$$
$$\mathrm{In}_x^1 = \min(2, 2\mathrm{Out}_y, \max(0, \mathrm{In}_x^2))$$

$$\mathrm{In}_x^2 \leq 2$$
$$\mathrm{In}_x^2 \leq \mathrm{Out}_{-y}$$
$$\mathrm{In}_x^2 \leq \max(0, \mathrm{In}_x^1)$$
$$\mathrm{In}_x^2 = \min(2, \mathrm{Out}_{-y}, \max(0, \mathrm{In}_x^1))$$

We consider here just the computation of In_x^1 and In_x^2 given Out_y and Out_{-y}. Our equation system assumes that Out_y and Out_{-y} are positive.

Policies

Policy selection select between max alternative to get a set of concave equations. In this case, the least fixpoint is given by the policy:

$$\begin{aligned} \mathrm{In}_x^1 &= \min(2, 2\mathrm{Out}_y, 0) \\ \mathrm{In}_x^2 &= \min(2, \mathrm{Out}_{-y}, 0) \end{aligned}$$

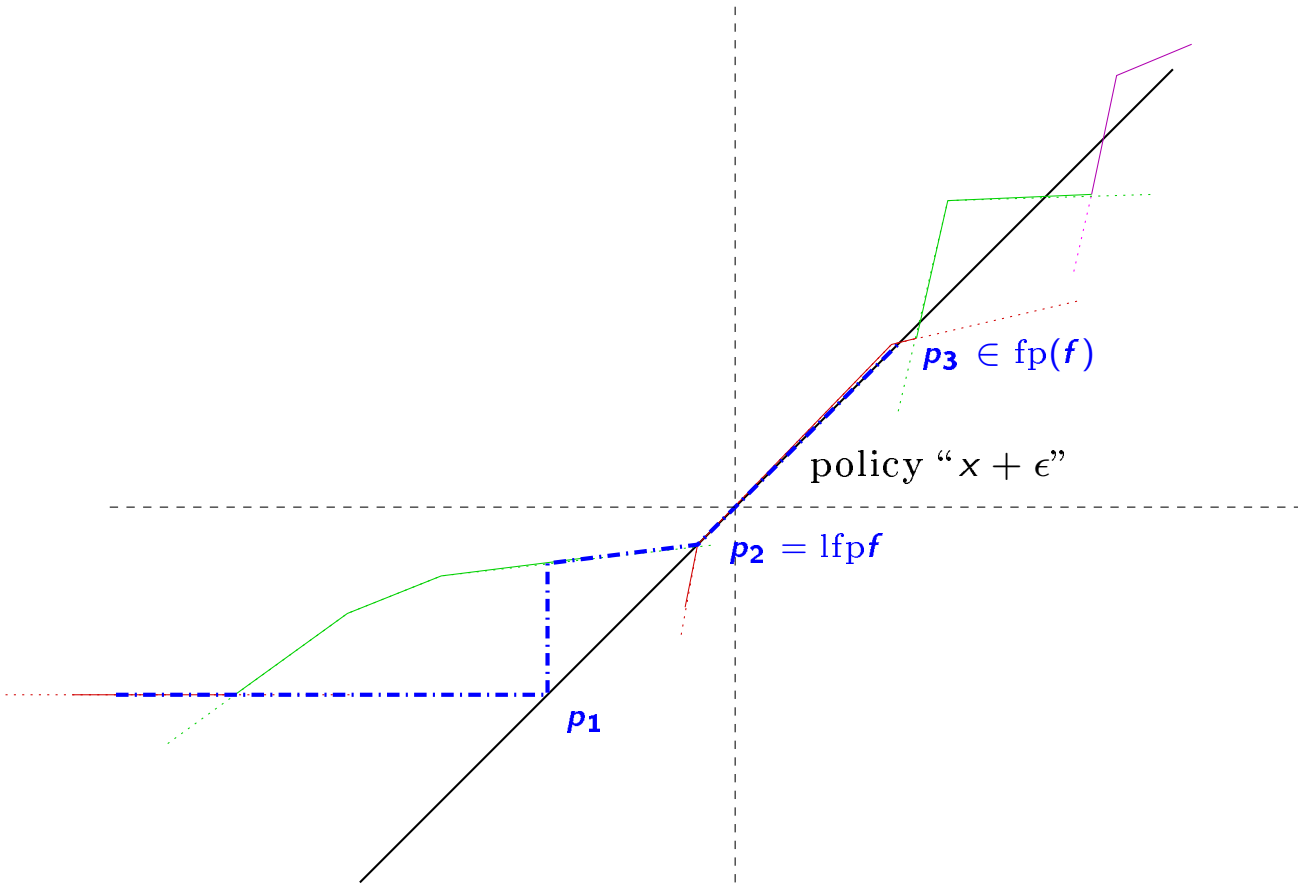
which gives the fixpoint $\mathrm{In}_x^1 = \mathrm{In}_x^2 = 0$.
The "correct" policy, taking into account time, would be:

$$\begin{aligned} \mathrm{In}_x^1 &= \min(2, 2\mathrm{Out}_y, \mathrm{In}_x^2) \\ \mathrm{In}_x^2 &= \min(2, \mathrm{Out}_{-y}, \mathrm{In}_x^1) \end{aligned}$$

which gives (we compute the greatest finite fixpoint)
$\mathrm{In}_x^1 = \mathrm{In}_x^2 = \min(2, 2\mathrm{Out}_y, \mathrm{Out}_{-y})$.

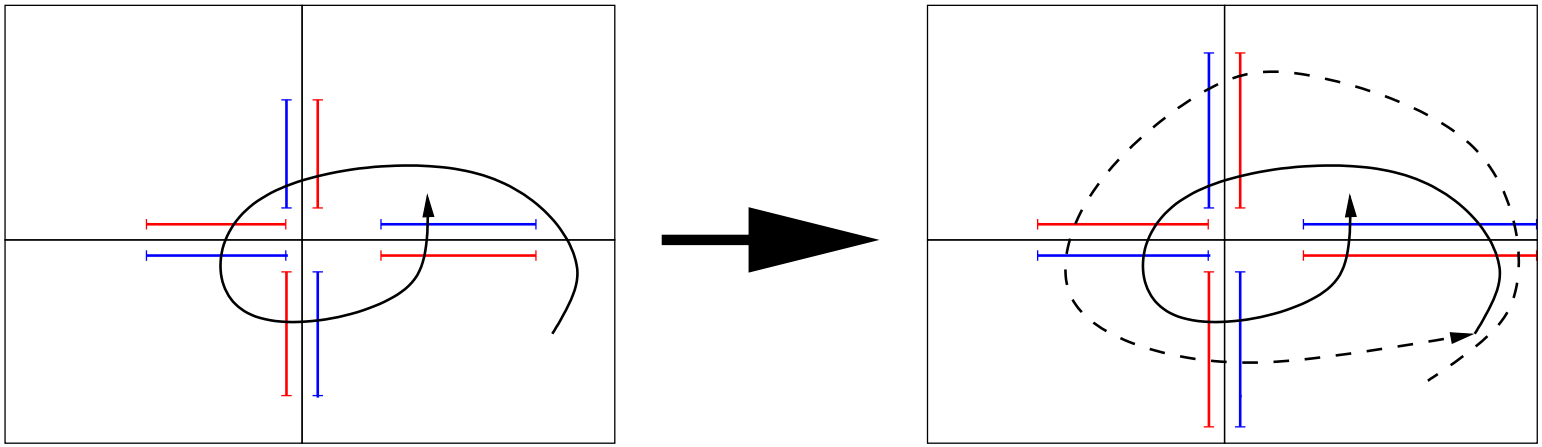
Beyond the least fixpoint

Since timeless cycles do not modify the current state values, they correspond to specific policies, locally equal to the identity function. These policies are never selected when computing the least fixpoint, but would enable to go beyond it (how?).



Policy selection

In the case of $\mathrm{lfp}\ \mathrm{pre}$, a policy can be seen as an "increasing" cycle of doors (containing a subpath going inside the current set).



cycle of doors with an ingoing path

after computing the policy fixpoint,
all paths looping there must come from the current set
(other paths are not concerned)

Policies for $\mathrm{lfp}\ \widetilde{\mathrm{pre}}$ are more abstract, creating different cycles for different controls, but represent broadly the same principle.

Using bounded model checking to select a policy

Approach suggested in 2011 by Monniaux and Gawlitza for $\mathrm{lfp}\ \mathrm{post}$: use bounded model checking (SMT solving) to:

- select a relevant strategy;
- improve the precision of the computation (path focusing).

The principle is to modelise as a bounded problem the existence of an "increasing" path and using this path as a strategy improvement (only for a restricted set of program points, traversing all cycles).

Application to our problem

A similar approach would be to formalise the following problem, given a current maze $\mathcal{M}$:

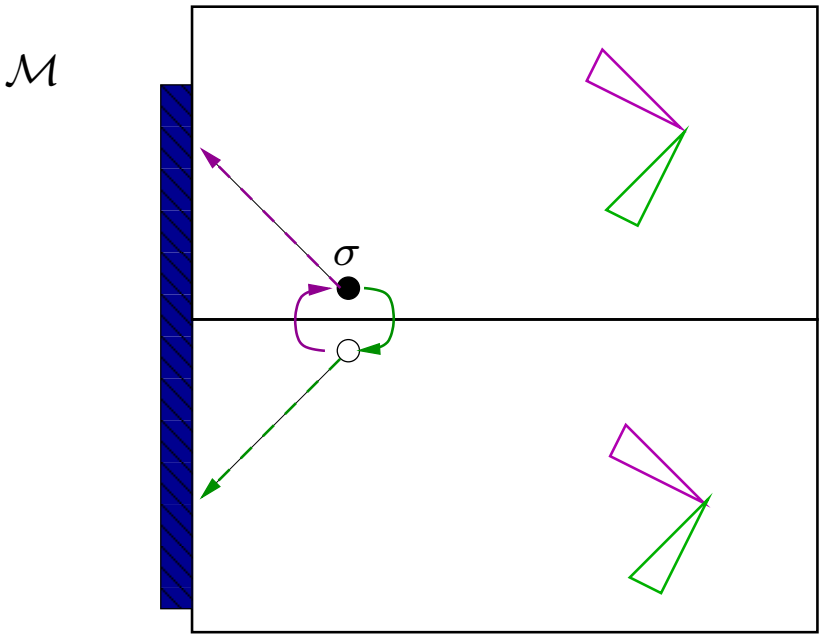
Invariant problem

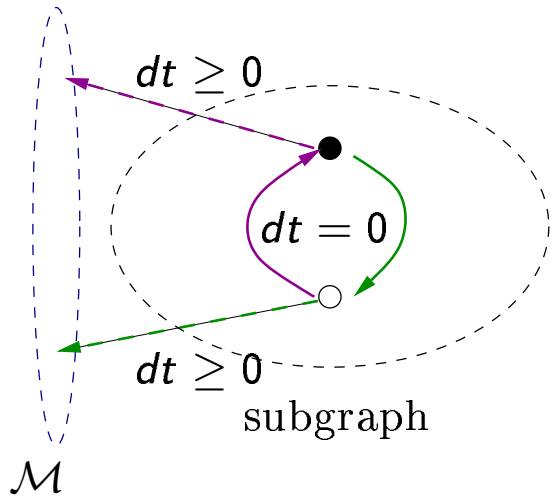
Is there a state $\sigma \notin \mathcal{M}$ such that each timed transition from σ (following any sequence of timeless transitions) leads to $\mathcal{M}$?

A solution of this problem can be seen a sub-graph of paths, with only timeless cycles, all times transitions leading directly to $\mathcal{M}$.
The problem can be modelised as a satisfaction problem for a mix of Boolean and linear constraints:

1. Boolean variables expresses the inclusion of σ w.r.t. doors, what are the output doors after σ for each control, and which transition is timeless;
2. numerical constraints expresses things like $\sigma \notin \mathcal{M}$, as well as the fact that all timed transitions lead to $\mathcal{M}$.

Working example



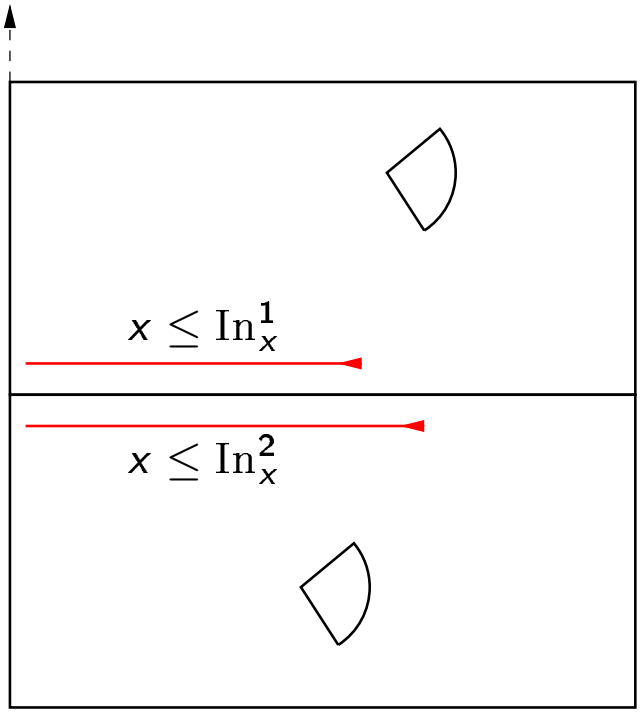


From the valuations of the Boolean variables (which state which outgoing doors are associated to each couple (ingoing door, control)), one can deduce the next strategy.
Hence:

- only relevant strategy are selected;
- this modelisation represents an "external" checking of the soundness of the algorithm.

Policy iteration and greatest fixpoint

For greatest fixpoint, we use dual solutions for the linear programs to generate the potential policies:



$$\mathrm{In}_x^1 = max\{x \mid \exists t \geq 0, x \leq 2 \wedge x + \lambda t \leq 2 \wedge x + \lambda t \leq \mathrm{In}_x^2\}$$

$$\mathrm{In}_x^1 = min(2, \mathrm{In}_x^2)$$

$$\mathrm{In}_x^2 = max\{x \mid \exists t \geq 0, x \leq 2 \wedge x + \lambda t \leq 2 \wedge x + \lambda t \leq \mathrm{In}_x^1\}$$

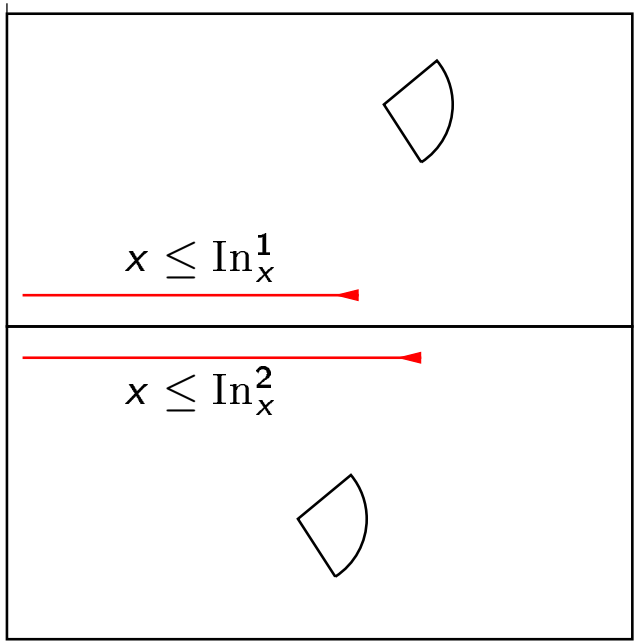
$$\mathrm{In}_x^2 = min(2, \mathrm{In}_x^1)$$

Starting policy : $\mathrm{In}_x^1 = 2$, $\mathrm{In}_x^2 = 2$, fixpoint reached.

No easy way to modelise unstable “timeless” loops.

Intuitive (but not so satisfying) idea

We can try to select only timed transition (if available), which implies strict polyhedra and strict inequalities.



$$\mathrm{In}_x^1 = max\{x \mid \exists t > 0, x \leq 2 \wedge x + \lambda t \leq 2 \wedge x + \lambda t \leq \mathrm{In}_x^2\}$$

$$\mathrm{In}_x^1 = min(2 - \varepsilon, \mathrm{In}_x^2 - \varepsilon)$$

$$\mathrm{In}_x^2 = max\{x \mid \exists t > 0, x \leq 2 \wedge x + \lambda t \leq 2 \wedge x + \lambda t \leq \mathrm{In}_x^1\}$$

$$\mathrm{In}_x^2 = min(2 - \varepsilon, \mathrm{In}_x^1 - \varepsilon)$$

Starting policy : $\mathrm{In}_x^1 = 2 - \varepsilon$, $\mathrm{In}_x^2 = 2 - \varepsilon$.

Next policy : $\mathrm{In}_x^1 = \mathrm{In}_x^2 - \varepsilon$, $\mathrm{In}_x^2 = \mathrm{In}_y^2 - \varepsilon$, fixpoint $= -\infty$.

Conclusions

Relationships between approximating differential inclusion sets and temporal property sets.
Lot of work to do, theoretical:

- better formalisation of this approach, for timeless and limited time cycles;
- specific case of the greatest fixpoint;
- what about the modification of the paving (bisection)? other abstract domains (and higher dimensions)?

and practical:

- current implementations of policy iteration not suitable for the maze framework;
- for initial computations, Kleene iterations may be more efficient; how to switch to policy iteration when needed?