On Control with Bounded Computational Resources

Oded Maler

CNRS-VERIMAG

Grenoble, France

Joint work with Bruce Krogh (CMU) and Moez Mahfoudh

Control and Computation

Computation, like sensors and actuators, is one of the resources used in control.

Control quality usually improves when more computation is used:

- Sampling rate: higher sampling rates results in better control. They imply more computations per time interval.

- More sophisticated control algorithm consume more computation. For example: the decision horizon in model-predictive control.

- State estimation and noise filtering.

- Dynamic plant identification and re-calibration.

The Traditional Approach

In "classical" control theory functions are abstract mathematical objects. When implemented by computers, their evaluation consumes time and other resources.

Design by separation of concerns:

- The control engineer determines the control laws: the feed-back functions and their invocation rate.
- The informatician implements them on a computer architecture. Intended functionality lost – just tasks with release times and deadlines.

Advantage: specialization, "modularity".

Shortcomings: inefficient use of resources when controlling a complex and fast plant by a cheap computer.

The Appropriate Level of Abstraction

Two extreme approaches for treating computation in control design:

- Ignore it.
- Combine implementation details, such as programs, scheduling policies, with plant description.

Alternative modeling methodology: computation is not ignored but modeled in a sufficiently abstract way (a process that takes time).

Computational constraints can be taken into account early in the control design process, before implementation.

Application: Adaptive Allocation of Resources

Two models that relate the allocation of resources with plant performance.

Can be used by designers to:

- check whether performance requirements can be met on a given architecture

- to design schedulers which are sensitive to the evolution of the plant.

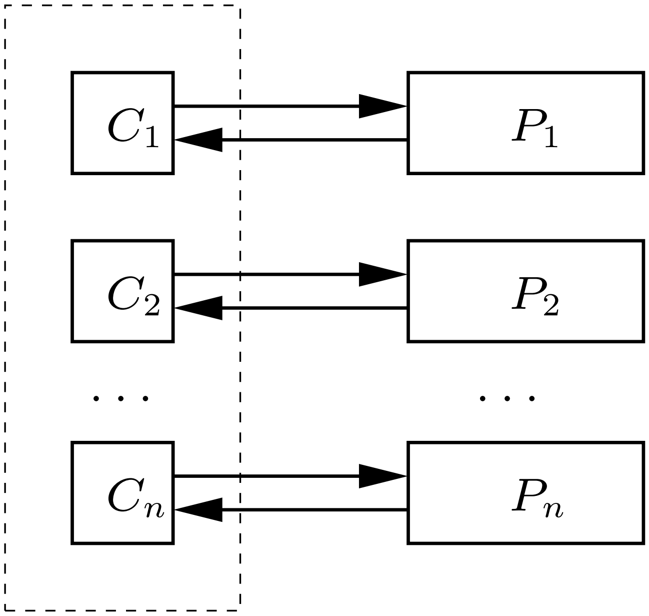
Slogan: breaking the wall separating the control and software engineer.

Anti-slogan: toy models, preliminary "proofs of concept". A large gap to be bridged before moving to practice.

The Setting

We have n independent sub-plants $P_1, \ldots, P_n$, controlled by n independent controllers $C_1, \ldots C_n$.

All controllers use the same processor and hence compete for the same resource.



How to allocate the resource according to the situation of the plant. How to pay attention to the more burning issues?

Plant Performance

For each controlled plant $S_i = P_i||C_i$, a performance measure indicating how well the plant is doing.

We associate with S_i a scalar variable x_i tracking the evolution of the performance index

Examples: distance from a reference point, smoothness of the trajectory in some time window, ...

To use this measure in the design we need a model of its dynamics (like models used in control design).

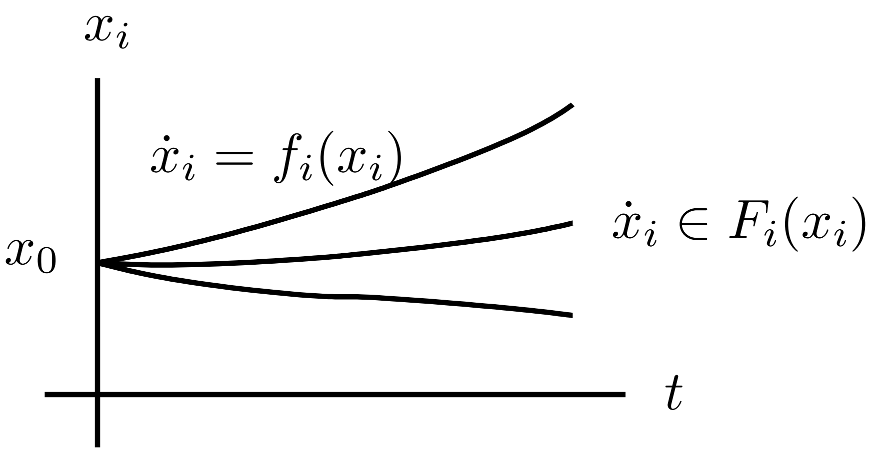
To use it during the execution as a basis for adaptive scheduling, the value of x_i should be computable from available observations within a bounded delay.

Dynamics of the Performance Index I

- For every plant P_i, x_i lives in $[0, \infty)$ where $x_i = 0$ is good and $x_i \geq m_i$ is very bad.

- Its evolution is governed by the differential inclusion $\dot{x}_i \in F_i(x_i)$ capturing all possible behaviors under all disturbances.

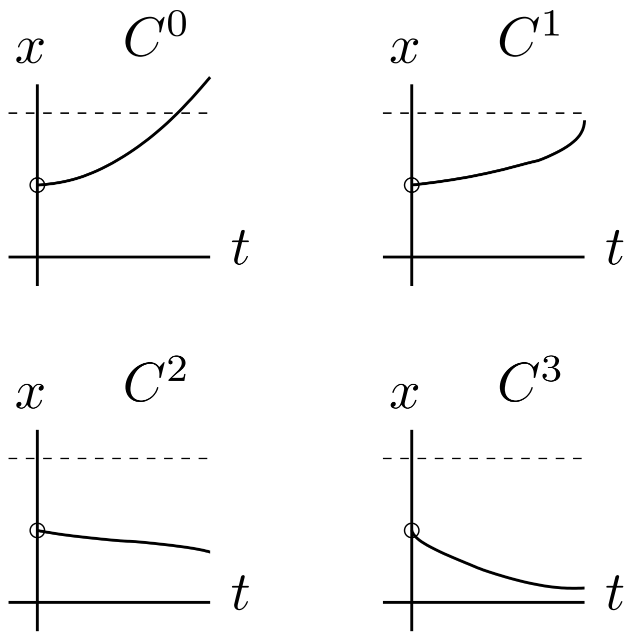
- All these behaviors are *bounded from above* by a *worst-case* dynamics: $\dot{x}_i = f_i(x_i)$.



Dynamics of the Performance Index II

In the short-term: when C is a good controller x converges to zero. When C is bad (for example, no control at all) x diverges.

$$C^0 \prec C^1 \prec C^2 \prec C^3$$



In the long-run: even for a good controller x starts diverging unless some investment is done in re-calibration.

Some Notations

To speak of a combination of continuous evolution and discrete transition we need notations that capture both.

- $\mathbf{x}_0 \xrightarrow{f,t} \mathbf{x}_1$: the trajectory starting from $\mathbf{x}_0$, leads to $\mathbf{x}_1$ within t time.
- $\mathbf{x}_0 \xrightarrow{f,t} G$: the trajectory reaches some point $\mathbf{x}_1 \in G$.
- $\mathbf{x}_0 \xrightarrow[H]{f,t} \mathbf{x}_1$, $\mathbf{x}_0 \xrightarrow[H]{f,t} G$: as above and the trajectory stays in $H \subseteq X$ during the interval $[0, t)$.

By trajectories we mean solutions of the differential equation $\dot{\mathbf{x}} = f(\mathbf{x})$.

Model I: Re-calibrating Controllers

Long-term behavior: the performance of all plants deteriorates Occasionally one needs to re-calibrate some controller C_i.

This takes d_i time and then x_i is reset to zero.

Resetting Dynamical System (RDS):
$\mathcal{A} = (X, f, \mathbf{d})$ where $X = [0, \infty)^n$, $f : X \to \mathbb{R}^n$ is a vector field and $\mathbf{d} = (d_1, \ldots, d_n)$ is a vector of delay constants.

f is diagonal and positive, i.e. $f(\mathbf{x}) = (f_1(x_1), \ldots f_n(x_n))$ and $f_i(x_i) > 0$ for all i.

Reset Function: $R_i : X \to X$

$$R_i(x_1, \ldots, x_i, \ldots, x_n) = (x_1, \ldots, 0, \ldots, x_n)$$

Its inverse is $R_i^{-1} : X \to 2^X$.

Update Strategies

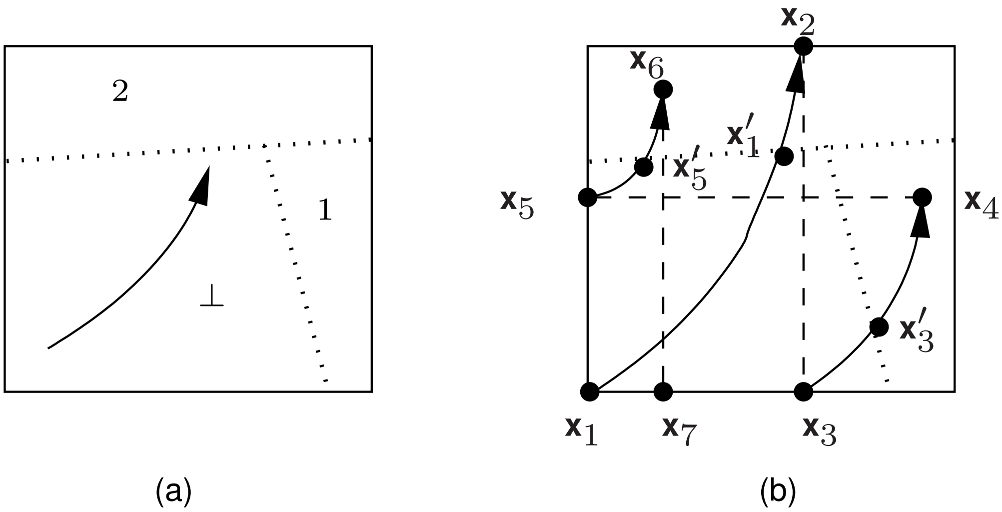
Update Strategy: $s : X \to \{1, \ldots, n, \perp\}$.
In a state **x** it tells the system either to reset some x_i or continue as usual ($\perp$).

A run following a strategy s:

$$\mathbf{x} \xrightarrow{t_1} \mathbf{x}' \xrightarrow{d_i} \mathbf{x}'' \xrightarrow{0} R_i(\mathbf{x}'') \xrightarrow{t_2} \ldots$$

such that $\mathbf{x} \xrightarrow[s^{-1}(\perp)]{t_1} \mathbf{x}'$ and $s(\mathbf{x}') = i$.

When $s(\mathbf{x}) = i$ the system continues to evolve for d_i and then it resets x_i.



$$\mathbf{x}_1 \xrightarrow{t_1} \mathbf{x}_1' \xrightarrow{d_2} \mathbf{x}_2 \xrightarrow{0} \mathbf{x}_3 \xrightarrow{t_2} \mathbf{x}_3' \xrightarrow{d_1}$$
$$\mathbf{x}_4 \xrightarrow{0} \mathbf{x}_5 \xrightarrow{t_3} \mathbf{x}_5' \xrightarrow{d_2} \mathbf{x}_6 \xrightarrow{0} \mathbf{x}_7$$

The Problem

Switching Synthesis Problem for RDS:

Given a RDS $\mathcal{A}$ and a hyper-rectangle $G \subseteq X$ containing **0**, find the maximal (controlled) invariant subset F of G.

The maximal F for which there exists a switching strategy $s : F \rightarrow \{1, \ldots n, \perp\}$ such that all trajectories starting in F stay in F.

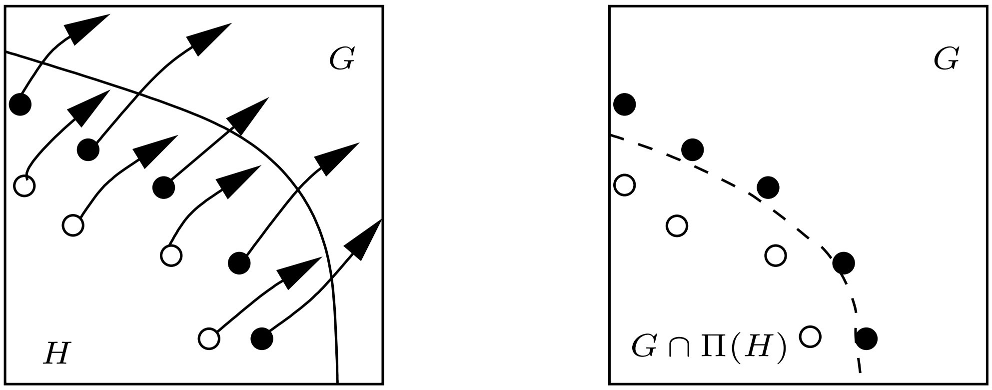
Of course we want to compute s as well.

When F is empty or too small it means that we cannot keep **x** bounded and we need a faster computer (smaller **d**).

The Solution I

Let H be a subset of X. The dealyed f-predecessors of H:

$$\Pi_{(f,d)}(H) = \{\mathbf{x} : \exists t \geq d\ \mathbf{x} \xrightarrow{f,t} H\}$$



The points from from which the system can reach H after evolving for at least d time according to the dynamics f.

Π is distributive over union:

$$\Pi(H_1 \cup H_2) = \Pi(H_1) \cup \Pi(H_2)$$

The Solution II

The following iterative algorithm computes F:

Winning Update Strategies for RDS:

$F^0 := G$
repeat
$F^{k+1} := \bigcup_{i\in\{1,\ldots,n\}} \Pi_{(f,d_i)}(R_i^{-1}(F^k))$
until $F^{k+1} = F^k$
$F := F^k$

x is in F^{k+1} if after d_i time and resetting x_i we reach F^k.

The Algorithm generates a decreasing sequence $F^0, F^1, F^2, \ldots$ whose limit F is the maximal invariant set.

Proof: let $d = \min\{d_i\}$. Then **x** $\in F^k$ if and only if starting from **x** the system can survive in G for at least dk time.

Model II: Different Mixes of Quality

Each plant P_i admits a family of controllers of varying quality

$$\mathcal{C}_i = \{C_i^1 \prec C_i^2 \ldots \prec C_i^p\}$$

At least one of them is good (makes x_i converge to zero).

The set of global controllers is $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$. Each $C \in \mathcal{C}$ is called a control “mode” of the system.

We want to maintain the global performance **x** inside a bounded rectangle $G \subseteq \mathbb{R}_+^n$.

Due to resource constraints, only a subset $\mathcal{C}' \subseteq \mathcal{C}$ of control modes is feasible. In each feasible mode at least one plant is controlled badly.

The only way to keep **x** bounded is to switch between modes.

Motivation: Sampling Rates

For every plant P_i the family of controllers consists of digital realizations of the same continuous controller with an increasing sampling rate.

The computation time of C_i^j is D_i and its frequency is φ_i^j with $\varphi_i^j < \varphi_i^{j+1}$.

A mode $C = (C_1^{j_1}, \ldots C_n^{j_n})$ is feasible if the schedulability condition (Liu and Layland) is satisfied :

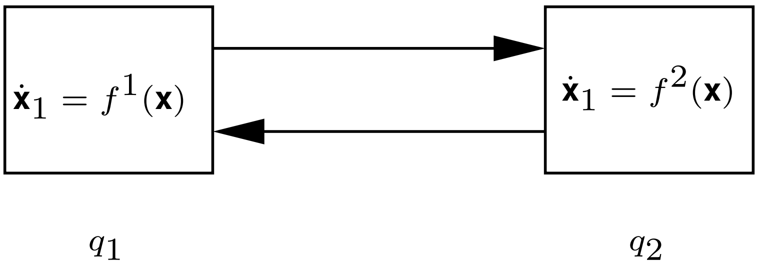
$$\sum_{i=1}^{n} D_i \cdot \varphi_i^{j_i} \leq 1$$

If control is impossible, a faster processor will reduce D_i, increase the set of feasible modes, and can make the system controllable.

We assume it takes at most d time to switch between modes (to complete some cycles in the current schedule and to start the new one).

Multi-Mode Hybrid Automata

Multi-Mode Hybrid Automaton (MMHA): $\mathcal{A} = (Q, X, f, d)$ where Q is a set of modes, $X = [0, \infty)^n$, $f : Q \to (X \to \mathbb{R})$ is a family of vector fields and $d > 0$.



For every q, f^q is diagonal and its flow is component-wise monotone, i.e.

$$f^q(\mathbf{x}) = (f_1^q(x_1), \ldots f_n^q(x_n))$$

for every i, the sign of f_i^q is uniform all over X.

A subset $\mathcal{F} \subseteq Q \times X$ can be written as $\{(q_1, F_1), \ldots (q_n, F_n)\}$.

Switching Strategies

A switching strategy $\mathcal{A}$ is a function $s : Q \times X \to Q$, a rule that tells the controller when to switch modes.

A trajectory following a strategy s is a sequence of the form

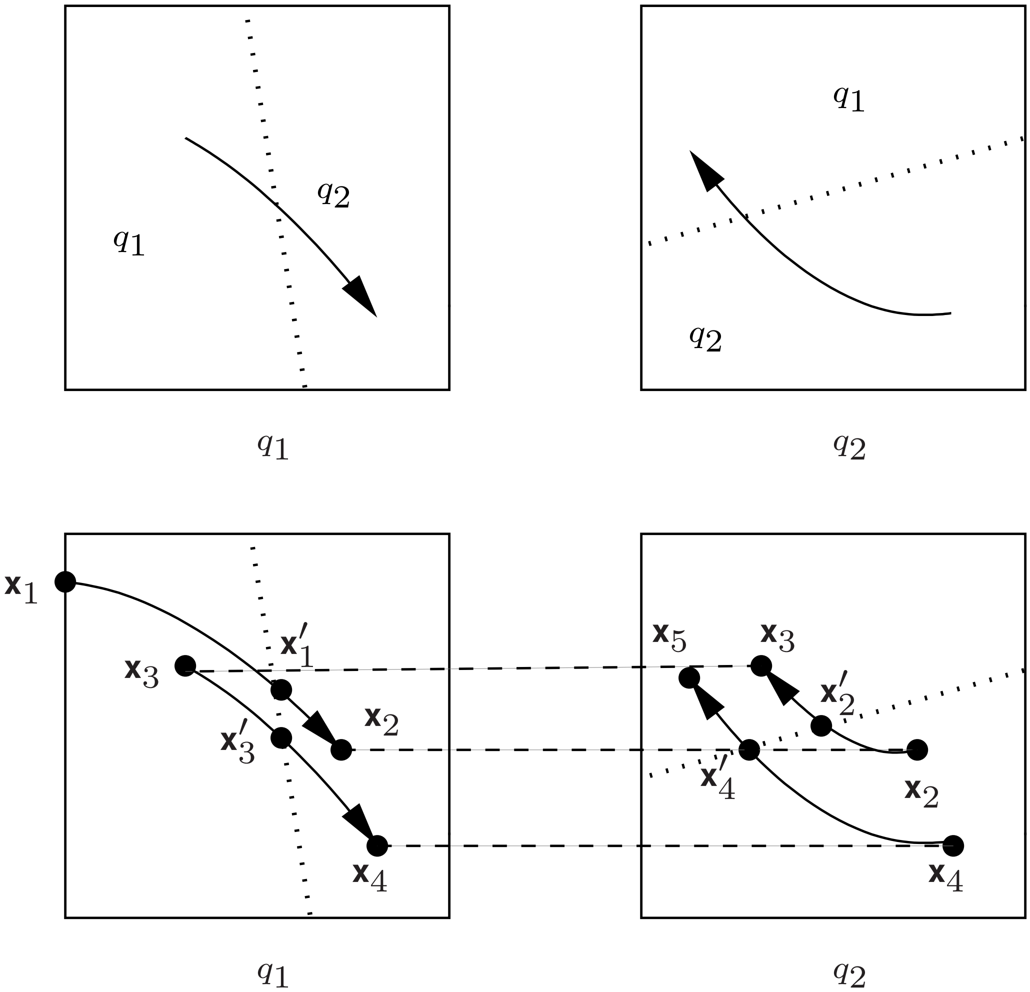
$$(q, \mathbf{x}) \xrightarrow{t_1} (q, \mathbf{x}') \xrightarrow{d} (q, \mathbf{x}'') \xrightarrow{0} (q', \mathbf{x}'') \xrightarrow{t_2} \ldots$$

such that $\mathbf{x} \xrightarrow[s^{-1}(q)]{f_q, t} \mathbf{x}'$, $s(q, \mathbf{x}') = q'$ and $\mathbf{x}' \xrightarrow{f_q, d} \mathbf{x}''$.

$\mathbf{x}$ evolves for t_1 time following the f_q dynamics until $\mathbf{x}'$ where $s(q, \mathbf{x}') = q'$. Then the system still evolves for d time at q and then switches to q'.

Example

A system with two plants and two modes, one good for x_2 and the other for x_1.



$$(q_1, \mathbf{x}_1) \xrightarrow{t_1} (q_1, \mathbf{x}_1') \xrightarrow{d} (q_1, \mathbf{x}_2) \xrightarrow{0} (q_2, \mathbf{x}_2) \xrightarrow{t_2} (q_2, \mathbf{x}_2') \xrightarrow{d} (q_2, \mathbf{x}_3) \xrightarrow{0}$$
$$(q_1, \mathbf{x}_3) \xrightarrow{t_3} (q_1, \mathbf{x}_3') \xrightarrow{d} (q_1, \mathbf{x}_4) \xrightarrow{0} (q_2, \mathbf{x}_4) \xrightarrow{t_4} (q_2, \mathbf{x}_4') \xrightarrow{d} (q_2, \mathbf{x}_5)$$

The Problem and the Solution

Switching Synthesis Problem for MMHA:

Given a MMHA $\mathcal{A}$ and a hyper-rectangle $G \subseteq X$ containing **0**, find the maximal subset $\mathcal{F}$ of $Q \times G$ for which there exists a switching strategy $s : \mathcal{F} \to Q$ such that all the trajectories starting in $\mathcal{F}$ stay in $\mathcal{F}$.

The algorithm:

$$\forall q \in Q \qquad \begin{array}{l} F_q^0 := G \\ \textbf{repeat} \\ \quad F_q^{k+1} := G \cap \bigcup_{q' \in Q} \Pi_{(f_q,d)}(F_{q'}^k) \\ \textbf{until } F_q^{k+1} = F_q^k \\ F_q := F_q^k \end{array}$$

The algorithm generates a decreasing sequence $\mathcal{F}^0, \mathcal{F}^1, \mathcal{F}^2, \ldots$ whose limit $\mathcal{F}$ is the maximal invariant set.

Proof: Similarly to [AMPS98], [ABDMP00] and the previous algorithm.

Strategy Extraction

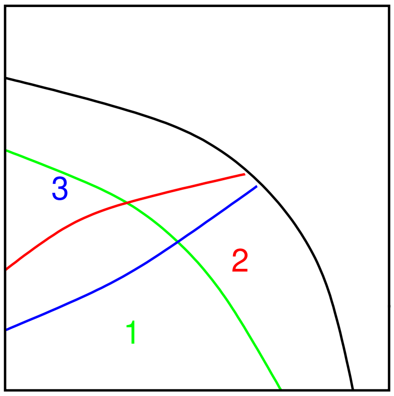
Once the algorithm converges a switching strategy can be extracted.

First we compute a non-deterministic "least-restrictive" strategy $\hat{s} : \mathcal{F} \to 2^Q$:

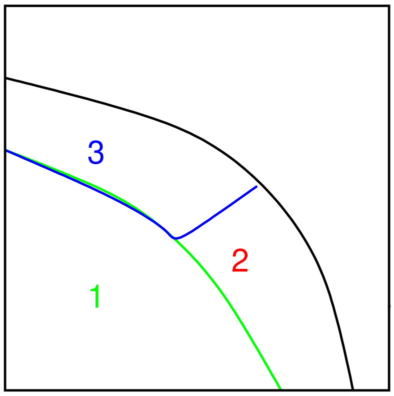
$$\hat{s}(q, \mathbf{x}) = \{q' : \mathbf{x} \xrightarrow{f_q, d} F_{q'}\}$$

all the modes to which it is possible to switch safely from $(q, \mathbf{x})$.

Then take s to be any subset of $\hat{s}$.



$\hat{s}(q, .)$

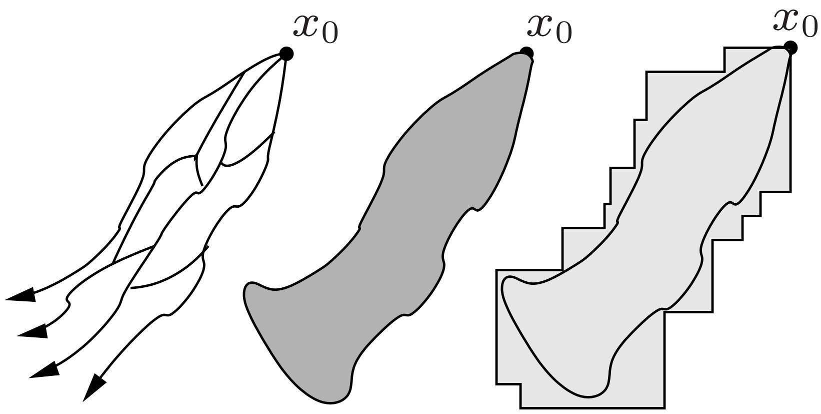


$s(q, .)$

Computational Issues

For non-trivial dynamics f the set $pred f, d(H)$ cannot be computed effectively.

Tools such as CheckMate and $\mathbf{d/dt}$ can approximate it using polyhedra.



The controller is a module on top of the digital control system. It monitors the performance of the systems and switches between modes according to simple rules.

It is guaranteed to keep the system in G assuming the *worst-case* dynamics f in each mode.

The Case of Constant Slopes

In every mode the derivative of **x** is constant. Motivation:

- Can serve as an approximation of f.
- When f is linear, i.e. $f(\mathbf{x}) = A\mathbf{x}$, the derivative of $\log \mathbf{x}$ is constant.
- The Π operator can be computed exactly with linear algebra (HyTech).
- Potential convergence after finitely many steps (decidability)? (Theoreticians exposed).

To keep variables non-negative:

$$\lceil c, x \rceil = \begin{cases} c & \text{if } x > 0 \text{ or } c > 0 \\ 0 & \text{otherwise} \end{cases}$$

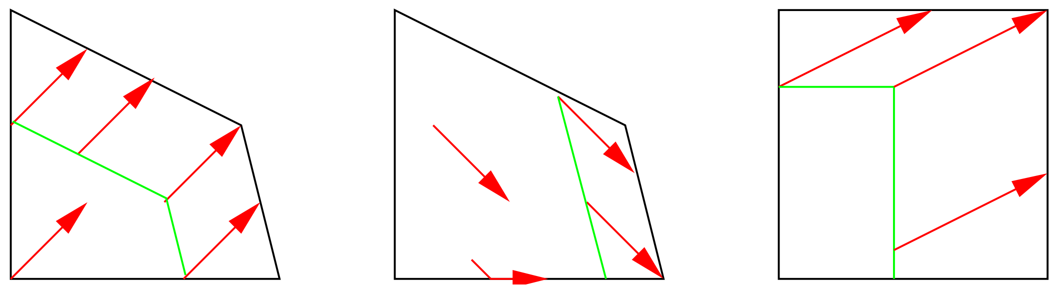
The evolution of **x** in q is $\dot{\mathbf{x}} = \lceil \mathbf{c}_q, \mathbf{x} \rceil$. All slope vectors and d are integer.

Preservation of Polyhedra

The predecessor operator specializes into:

$$\Pi_{(\mathbf{c},d)}(H) = \{\mathbf{x} : \exists t \geq d \ \ \lceil \mathbf{x} + \mathbf{c}t \rceil \in H\}.$$

- If H is a convex polyhedron so is $\Pi_{(\mathbf{c},d)}(H)$.
- When $\mathbf{c}$ is positive, and H is a hyper-rectangle with integer endpoints, so are $R^{-1}(H)$ and $\Pi_{(\mathbf{c},d)}(H)$.



Effectiveness and Convergence

- All the steps in the two algorithms can be computed effectively.

- The synthesis algorithm for RDS always terminates.

Proof: the set of rectangles with integer endpoints is finite and the iteration is monotone.

Open question: what about synthesis for MMHA?

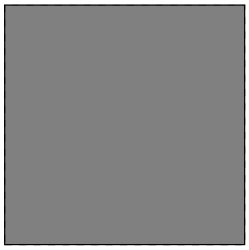
Necessary condition for controllability: there are non-negative constants $\{\lambda_q\}_{q \in Q}$ such that $\sum_{q \in Q} \lambda_q = 1$ and

$$\sum_{q \in Q} \lambda_q \mathbf{c}_q \leq \mathbf{0} \tag{1}$$

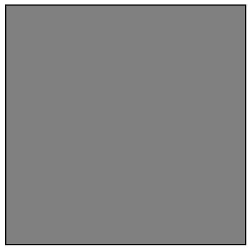
(1) gives "average direction". Not sufficient because of the effect of saturation and delay.

Two Dimensions

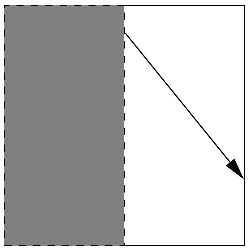
$$\mathbf{c}^1 = \begin{pmatrix} a_1 \\ -b_1 \end{pmatrix} \qquad \mathbf{c}^2 = \begin{pmatrix} -a_2 \\ b_2 \end{pmatrix}$$



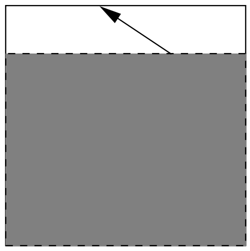
$F_1^0 = G$



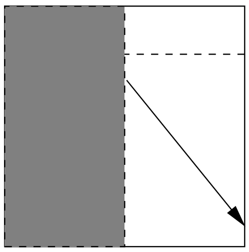
$F_2^0 = G$



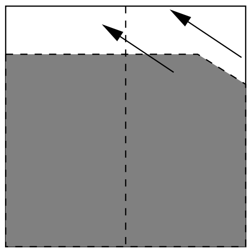
$F_1^1 = \Pi_{(\mathbf{c}_1,1)}(F_2^0)$



$F_2^1 = \Pi_{(\mathbf{c}_2,1)}(F_1^0)$



$F_1^2 = \Pi_{(\mathbf{c}_1,1)}(F_2^1)$



$F_2^2 = \Pi_{(\mathbf{c}_2,1)}(F_1^1)$

$$F_1^2 = \begin{array}{l} x_1 \leq m - a_1 \\ b_1 x_1 + a_1 x_2 \leq (b_1 + a_1)m - a_1 b_2 \end{array}$$

$$F_2^1 = \begin{array}{l} x_2 \leq m - b_2 \\ b_2 x_1 + a_2 x_2 \leq (a_2 + b_2)m - a_1 b_2 \end{array}$$

Empirically it converges on higher dimensions but we have no proof, neither a proof of undecidability.

Summary

A new methodology that relates computation time and other constraints with the quality of control.

Two toy problems that can be solved approximately.

An intriguing (but practically unimportant) decidability question for hybrid automata with constant slopes.