Convex Optimization-based Static Analysis for Controllers

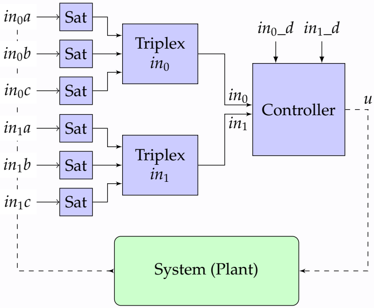
Pierre-Loïc Garoche – ONERA

November, 18th, 2016

Contents

Contents

CONTROLLERS



Typically:

- Linear core: $x_{k+1} = Ax_k + Bin_k$
- Non linearities:
  - Piecewise systems, ie. with modes: if $guard_i$ then $x_{k+1} = A_i x_k + B_i in_k$
  - Saturations on inputs/outputs
  - Linear Parameter Varying (LPV): linearization through gain interpolation
  - Polynomial update
  - Safety architecture (redundancy, voters, ...)

Hypothesis: everything is discrete, no continuous models (e.g. ODE)

Object under analysis – The Input

System:

- Code
  - set of functions, sequence of instructions, mix of boolean conditions, integer counters, floating point computations, pointers
  - no dynamic allocation (malloc), no nested loops
- Models
  - similar notions but simpler: no pointers, more types,
- knowledge can be provided on model components: a linear controller, an anti-windup, a saturation, etc

Safety property:

axiomatic semantics, aka predicate over values

- set of all reachable values is bounded
- a given bad region is unreachable
- high level properties: stability, robustness, bounded overshoot, etc
- …

How to verify safety properties

Let such a discrete system be defined as

- set of states Σ
- initial states $\mathit{Init} \subseteq \wp(\Sigma)$
- dynamics: $\mathit{Step} \subseteq \wp(\Sigma \times \Sigma)$

Almost all analyses are based on "induction"

How to verify safety properties

Let such a discrete system be defined as

- set of states Σ
- initial states $Init \subseteq \wp(\Sigma)$
- dynamics: $Step \subseteq \wp(\Sigma \times \Sigma)$

Almost all analyses are based on "induction"

- SMT-based model checking (k-induction, PDR), Deductive methods
  - encode system semantics S and property P as logical predicates
  - check inductiveness of P wrt S through calls to SMT solvers
  - loops: compute (inductive) loop invariants
- Static Analysis (Abstract interpretation)
  - express collecting semantics (reachable states) as a fixpoint
  - approximate the fixpoint by a larger set of states, the over-approximation
    inductive wrt the abstract transition relation (abstract domain).

In all cases: computation or need of **inductive invariants**

Abstract Interpretation: definitions

Definition

Abstract Interpretation is a constructive and sound theory for the approximation of semantics expressed as fixpoint of monotonic operators in a complete lattice.

Collecting semantics ($\mathcal{R}$) as a fixpoint

- transition system: $(\Sigma, \mathit{Init}, \mathit{Step})$
- monotonic function:

$$\begin{array}{rcl} F : \wp(\Sigma) & \rightarrow & \wp(\Sigma) \\ X & \mapsto & \{s' \in \wp(\Sigma) | s' \in \mathit{Init} \vee \exists s \in X, (s, s') \in \mathit{Step}\} \end{array}$$

- $\mathcal{R} = \mathrm{lfp}\ F$

Thanks to Tarski fixpoint theorem, it exists and is defined as the smallest postfixpoint

$$\mathcal{R} = \mathrm{lfp}\ F = \inf\{X | F(X) \subseteq X\}$$

Abstracting the fixpoint

Instead of computing $\mathcal{R}$, computation of $\mathcal{R}^{\#}$ such that $\gamma(\mathcal{R}^{\#}) \supseteq \mathcal{R}$ and

$$\mathcal{R}^{\#} = \mathrm{lfp}\, f^{\#} \text{ with } F^{\#} : X \mapsto \alpha(\mathit{Init}) \sqcup^{\#} \bigsqcup^{\#}_{\exists s' \in \Sigma, \exists s \in \gamma(X), \mathit{Step}(s,s')} \alpha(\{s'\})$$

where an abstract domain is defined by

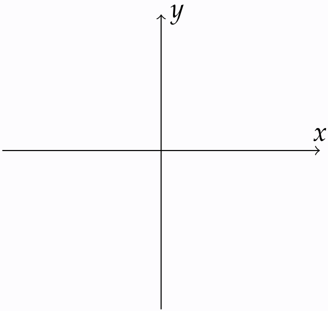
- $\langle \mathcal{D}, \sqsubseteq^{\#} \rangle$ a partially ordered set of abstract elements, $\perp$ its infimum.
- $\sqcup^{\#}$ a join operator
- $\alpha : \wp(\Sigma) \rightarrow \mathcal{D}$ an abstraction function
- $\gamma : \mathcal{D} \rightarrow \wp(\Sigma)$ a concretization function

E.g. interval abstraction, convex polyhedra, etc

CLASSICAL ABSTRACT FIXPOINT COMPUTATION: KLEENE ITERATIONS

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\perp)$

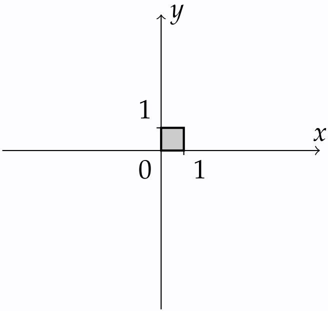
```
x := ?(0, 1); y := ?(0, 1);
while true do
   in := ?(0, 1);
   if 0.9 − in ≤ 0 then
      x := 10×in − 9;
      y := 10×in − 9
   else
      t := x;
      x := 0.2×t − 0.7×y + 0.5×in;
      y := 0.7×t + 0.2×y + 0.5×in
   fi
od
```



Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n\to+\infty} F^n(\perp)$

x **:= ?(**0, 1**)**; y **:= ?(**0, 1**)**;
while true **do**
  in **:= ?(**0, 1**)**;
  if 0.9 $-$ in $\leq$ 0 **then**
    x **:=** 10×in $-$ 9;
    y **:=** 10×in $-$ 9
  else
    t **:=** x;
    x **:=** 0.2×t $-$ 0.7×y $+$ 0.5×in;
    y **:=** 0.7×t $+$ 0.2×y $+$ 0.5×in
  fi
od

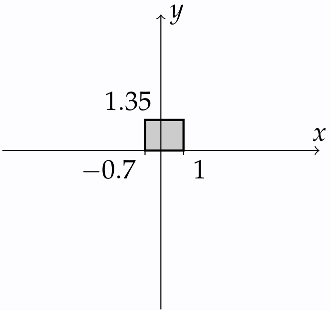


before entering the loop

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\bot)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
  in := ?(0, 1);
  if 0.9 − in ≤ 0 then
    x := 10×in − 9;
    y := 10×in − 9
  else
    t := x;
    x := 0.2×t − 0.7×y + 0.5×in;
    y := 0.7×t + 0.2×y + 0.5×in
  fi
od
```

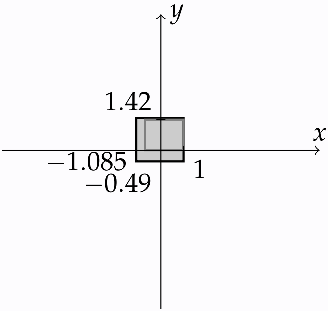


after a first iteration

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\perp)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
  in := ?(0, 1);
  if 0.9 − in ≤ 0 then
    x := 10×in − 9;
    y := 10×in − 9
  else
    t := x;
    x := 0.2×t − 0.7×y + 0.5×in;
    y := 0.7×t + 0.2×y + 0.5×in
  fi
od
```

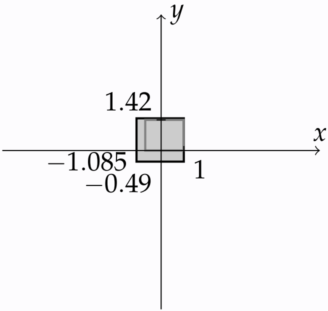


after a second iteration

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\perp)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
   in := ?(0, 1);
   if 0.9 − in ≤ 0 then
      x := 10×in − 9;
      y := 10×in − 9
   else
      t := x;
      x := 0.2×t − 0.7×y + 0.5×in;
      y := 0.7×t + 0.2×y + 0.5×in
   fi
od
```

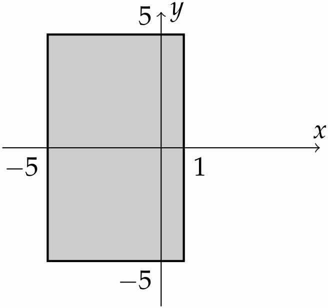


not stable → widening

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\bot)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
  in := ?(0, 1);
  if 0.9 − in ≤ 0 then
    x := 10×in − 9;
    y := 10×in − 9
  else
    t := x;
    x := 0.2×t − 0.7×y + 0.5×in;
    y := 0.7×t + 0.2×y + 0.5×in
  fi
od
```

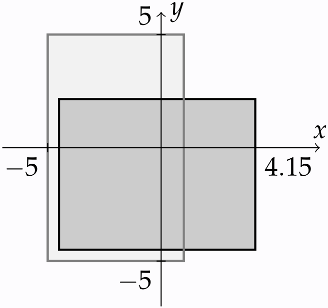


after widening

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\perp)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
  in := ?(0, 1);
  if 0.9 − in ≤ 0 then
    x := 10×in − 9;
    y := 10×in − 9
  else
    t := x;
    x := 0.2×t − 0.7×y + 0.5×in;
    y := 0.7×t + 0.2×y + 0.5×in
  fi
od
```

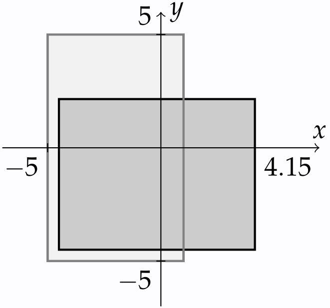


after another iteration

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\bot)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
   in := ?(0, 1);
   if 0.9 − in ≤ 0 then
      x := 10×in − 9;
      y := 10×in − 9
   else
      t := x;
      x := 0.2×t − 0.7×y + 0.5×in;
      y := 0.7×t + 0.2×y + 0.5×in
   fi
od
```

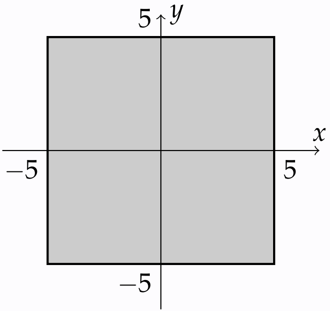


not stable → widening

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\bot)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
   in := ?(0, 1);
   if 0.9 − in ≤ 0 then
      x := 10×in − 9;
      y := 10×in − 9
   else
      t := x;
      x := 0.2×t − 0.7×y + 0.5×in;
      y := 0.7×t + 0.2×y + 0.5×in
   fi
od
```

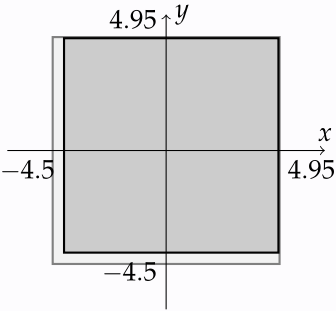


after widening

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\perp)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
  in := ?(0, 1);
  if 0.9 − in ≤ 0 then
    x := 10×in − 9;
    y := 10×in − 9
  else
    t := x;
    x := 0.2×t − 0.7×y + 0.5×in;
    y := 0.7×t + 0.2×y + 0.5×in
  fi
od
```

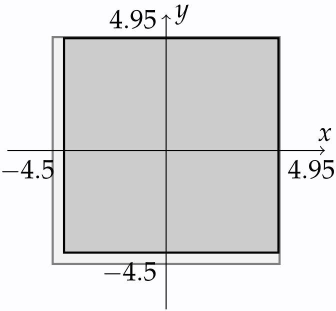


after another iteration

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n\to+\infty} F^n(\bot)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
  in := ?(0, 1);
  if 0.9 − in ≤ 0 then
    x := 10×in − 9;
    y := 10×in − 9
  else
    t := x;
    x := 0.2×t − 0.7×y + 0.5×in;
    y := 0.7×t + 0.2×y + 0.5×in
  fi
od
```



stable !

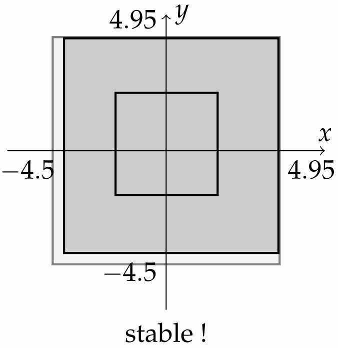
Remarks

- Worthwhile result: $x \in [-5, 5] \wedge y \in [-5, 5]$.
  But we were lucky with the widening.

Classical abstract fixpoint computation: Kleene iterations

When ascending chains admit least upper bounds, fixpoint can be computed iteratively using Kleene iterations lfp $F = lim_{n \to +\infty} F^n(\perp)$

```
x := ?(0, 1); y := ?(0, 1);
while true do
  in := ?(0, 1);
  if 0.9 − in ≤ 0 then
    x := 10×in − 9;
    y := 10×in − 9
  else
    t := x;
    x := 0.2×t − 0.7×y + 0.5×in;
    y := 0.7×t + 0.2×y + 0.5×in
  fi
od
```



Remarks

- Worthwhile result: $x \in [-5, 5] \wedge y \in [-5, 5]$.
  But we were lucky with the widening.
- Larger than least fixpoint: $x \in [-2.23, 2.27] \wedge y \in [-1.95, 2.55]$.

Analysis of controllers

Quadratic Lyapunov functions for linear systems

Let A be a square matrix. Define the linear system:

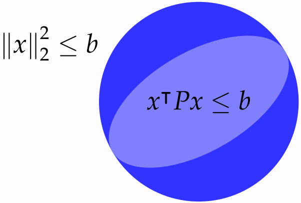
$$x^{k+1} = Ax^k, k \geq 0, \text{ a given } x^0$$

A matrix P satisfies Lyapunov conditions for the system iff:

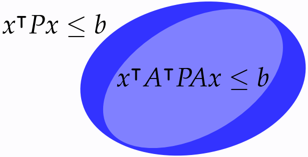
$$P - \mathrm{Id} \succeq 0 \,, \qquad P - A^\intercal P A \succeq 0 \;\;, \tag{1}$$

- Id is the identity matrix;
- $M \succeq 0$ means $M = M^\intercal$ and $\forall\, x, x^\intercal M x \geq 0$;

$P - \mathrm{Id} \succeq 0$ implies boundedness:



$P - A^\intercal P A \succeq 0$ guarantees the decrease:



Let A be a square matrix. Define the linear system:

$$x^{k+1} = Ax^k, k \geq 0, \text{ a given } x^0$$

A matrix P satisfies Lyapunov conditions for the system iff:

$$P - \mathrm{Id} \succeq 0\,, \qquad P - A^\intercal PA \succeq 0\,, \tag{1}$$

- Id is the identity matrix;
- $M \succeq 0$ means $M = M^\intercal$ and $\forall\, x, x^\intercal M x \geq 0$;

$P - \mathrm{Id} \succeq 0$ is equivalent to:

$$\forall\, \alpha \geq 0$$

$$\begin{pmatrix} -b & 0 \\ 0 & P \end{pmatrix} - \begin{pmatrix} -b & 0 \\ 0 & \mathrm{Id} \end{pmatrix} \succeq 0$$

$P - A^\intercal PA \succeq 0$ is equivalent to:

$$\forall\, b \geq 0$$

$$\begin{pmatrix} -b & 0 \\ 0 & P \end{pmatrix} - \begin{pmatrix} -b & 0 \\ 0 & A^\intercal PA \end{pmatrix} \succeq 0$$

Contents

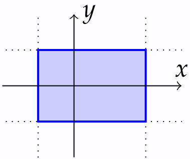
Template abstractions

Let x be a vector of program variables and $b_i \in \mathbb{R}$.

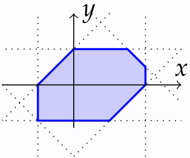
A template domain D is defined as: $\bigwedge_i p_i(x) \leq b_i$

- $p_i = \pm x_i \pm b_i$, octagons, for example $x - y \leq 3$
- p_i quadratic polynomials: ellipsoids
- p_i polynomials: basic semi-algebraic sets

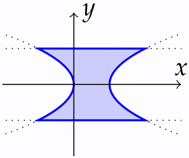
Once (p_i) fixed, an abstract element is only defined by the vector (b_i).



intervals



octagons



quadratic

$$\left\{ \begin{array}{rcl} x & \leq & 2 \\ -x & \leq & 1 \\ y & \leq & 1 \\ -y & \leq & 1 \end{array} \right.$$

$$\left\{ \begin{array}{rcl|rcl} x & \leq & 2 & x+y & \leq & 2.5 \\ -x & \leq & 1 & x-y & \leq & 2 \\ y & \leq & 1 & -x+y & \leq & 1 \\ -y & \leq & 1 & -x-y & \leq & 2 \end{array} \right.$$

$$\left\{ \begin{array}{rcl} y & \leq & 1 \\ -y & \leq & 1 \\ x - y^2 & \leq & 1 \\ -y^2 - x & \leq & 0 \end{array} \right.$$

Key contribution: Revisiting Tarski fixpoint definition as convex problem

Least fixpoint is the smallest postfixpoint

$$\text{lfp } F^{\#} = \mathit{inf}\left\{Y | F^{\#}(Y) \sqsubseteq Y\right\}$$

where $F^{\#}(Y) = \mathit{Init}^{\#} \sqcup f^{\#}(Y)$
Let C be a postfixpoint: $F^{\#}(C) \sqsubseteq C$. Then

$$\{F^{\#}(C) \sqsubseteq C\} = \left\{ C \;\middle|\; \begin{array}{l} \mathit{Init}^{\#} \sqsubseteq C \\ f^{\#}(C) \sqsubseteq C \end{array} \right\}$$

A postfixpoint, in a template domain, satisfies:

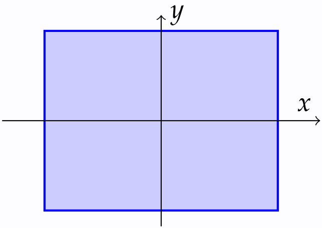
$$\begin{array}{l} \forall x \in \mathit{Init}, p(x) \le 0 \\ \forall (x, x') \in \mathit{Step}, p(x) \le 0 \implies p(x') \le 0 \end{array}$$

Lyapunov function: energy level decreases over trajectories:

$$p(x') \le p(x) \text{ is a sufficient condition for } p(x) \le 0 \implies p(x') \le 0$$

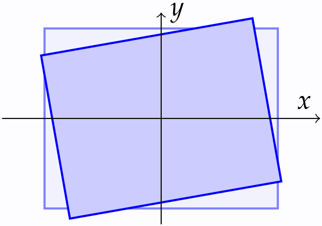
Quadratic templates for linear systems

- Linear invariants commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.



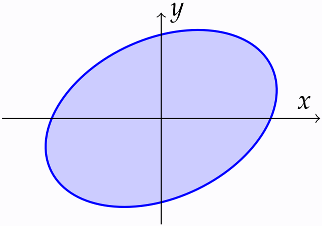
Quadratic templates for linear systems

- Linear invariants commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.



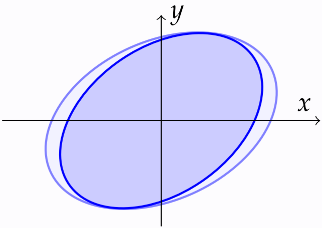
Quadratic templates for linear systems

- Linear invariants commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.
- Control theorists have known for a long time that quadratic invariants are a good fit for linear systems.



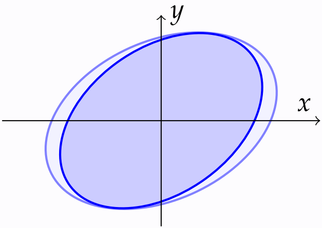
Quadratic templates for linear systems

- Linear invariants commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.
- Control theorists have known for a long time that quadratic invariants are a good fit for linear systems.

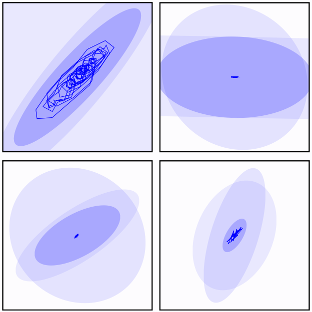


Quadratic templates for linear systems

- Linear invariants commonly used in static analysis are not well suited:
  - at best costly;
  - at worst no result.
- Control theorists have known for a long time that quadratic invariants are a good fit for linear systems.



Characterizing a small stable ellipsoid for a linear system:



- Convex expression: Linear Matrix Inequalities (LMI)

$$P - A^\intercal P A \succeq 0$$
$$P - \mathrm{Id} \succeq 0$$

- Different heuristics (encoding and optimization costs)
  - minimize condition number
  - preserve shape
  - consider inputs

Piecewise Linear Systems

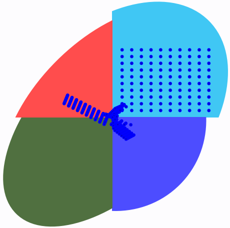
k-inductive quadratic invariants

For stable switched linear systems, a common Lyapunov function may not exists. Method by Morari *et al*, Rantzer and Johansson to compute piecewise quadratic LF.

- System defined as partition of zones: $X^i = \{c^i, T^i\}$.
- Build a set of local Lyapunov function P^i such that
  - $x \in X^i, T^i(x) \in X^j, x^\intercal P^i x \leq 0 \implies (T^i(x))^\intercal P^j T^i(x) \leq 0$
  - bound variable values in each zone
  - quadratic number of constraints in the LMI wrt number of zones.
- Reducing the set of possible zone transitions is performed using Motzkin transposition theorem

Extension to k-inductive invariants:

- Generate a set of paths in X^{i^*} of length $< k$
- Considering transitions between zones $i \to j$
  - Base cases with $|w| < k$: $T^{i\intercal} P^{w \cdot i \cdot j} T^i - P^{w \cdot i} \preceq 0$
  - Inductive cases with $|w| = k$: $T^{i\intercal} P^{tl(w \cdot i) \cdot j} T^i - P^{w \cdot i} \preceq 0$



Sum of Squares (SOS) Polynomials

Definition (SOS Polynomial)

A polynomial p is SOS if there are polynomials $q_1, \ldots, q_m$ s.t.

$$p = \sum_i q_i^2.$$

- If p SOS then $p \geq 0$

Sum of Squares (SOS) Polynomials

Definition (SOS Polynomial)

A polynomial p is SOS if there are polynomials $q_1, \ldots, q_m$ s.t.

$$p = \sum_i q_i^2.$$

- If p SOS then $p \geq 0$
- p SOS iff there exist $z := \left[1, x_0, x_1, x_0x_1, \ldots, x_n^d\right]$ and $Q \succeq 0$

$$p = z^T Q \, z.$$

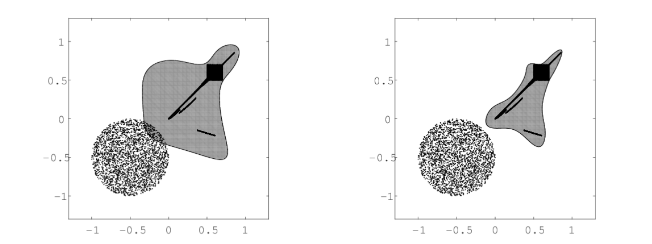
$\Rightarrow$ SOS can be encoded as semi-definite programming (SDP).

Polynomial invariants

Prop.-driven polynomial templates using SOS

Provided a property expressed as a sublevel set property $\kappa(x)$, search for polynomial p such that

- initial condition: $p(x) \leq 0, \forall x \in Init$
- inductiveness: $\forall\, i \in \mathcal{I}, p\left(T^i(x)\right) \leq p(x), \forall x \in X^i$
- property-driven, minimizing $w \in \mathbb{R}$ such that $\kappa(x) \leq w + p(x)$



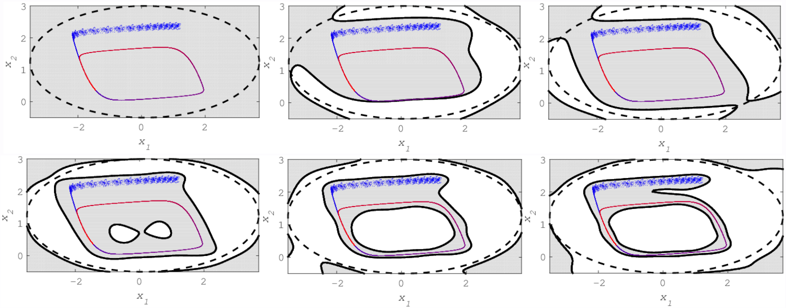
Expressions are convex and linear in p: Positiveness is ensured thanks to Sum of Square Programming (solving LMIs)

Polynomial invariants

Minimizing volume within compact set X using SOS

When provided an upper bound on variables value (a compact set X), one can minimize the volume of reachable states in that set:

- $p(x) \geq 0$ on initial states
- inductive positiveness (with damping scalar α): $\alpha\, p \circ T(x) - p(x)$ positive on X
- w positive on X and "strictly above" p: $w(x) \geq p(x) + 1$
- minimizing the volume of $w(x)$ in compact set X



Contents

Floating point issues

Floating point computation are inexact:

- approximate representation of constants
  e.g. 0.1 = 0.1000000000000000055511151231257827021182...
- sum/product of two floats is not necessarily a float
- results depend on order of evaluation (no distributivity, associativity)

Two main (and different) issues wrt floating-point arithmetic:

the analyzed controller performs its computations using floating-point arithmetic rather than real numbers

the analysis itself is performed in floating-point arithmetic, in particular the LMI/SOS is solved using approximate SDP solvers

Floating-Point Arithmetic in the Controller

Computations of the controller being performed using floating-point arithmetic, rounding errors unavoidably occur and x^c_{k+1} is not exactly equal to $f(x^c_k) = A_c x^c_k + B_c e_k$.

Using affine arithmetics or intervals, we bound the floating point error ϵ associated to the computation of $f(x^c_k)$ assuming x^c_k in a given interval $[a, b]$.

$$(f(x^c_k))_{fl} = f(x^c_k) \pm \epsilon$$

Inductiveness constraints in the LMI/SOS become

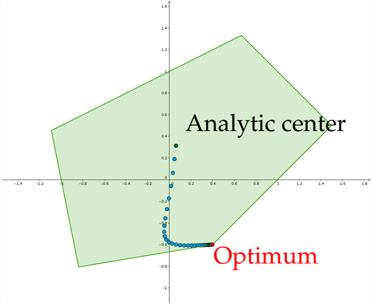
$$p \circ f(x) - p(x) + \epsilon \leq 0$$

In practice, for linear systems, $\epsilon \simeq 10^{-9}$ is small with respect to the ϵ already needed to compensate for the SDP solver precision.

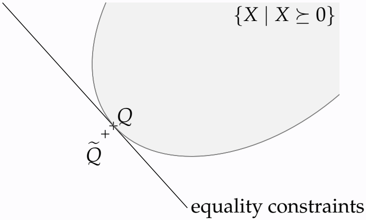
Floating-Point Arithmetic in the Analysis

- we solve a convex SDP optimization problem:

  linear objective + (LMI) constraints
- the SDP solver, implemented with floating-point arithmetic, computes an approximate solution
  - the solution is not the real optimum wrt objective
  - it may not strictly satisfy the constraints (ie. not a feasible solution)
  - more than often, returned values of P makes the LMI slightly not negative definite.



Interior point methods

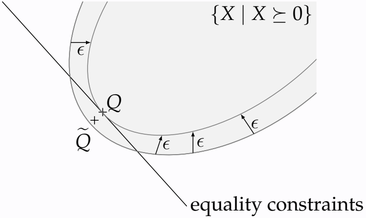


Infeasibility of the computed solution

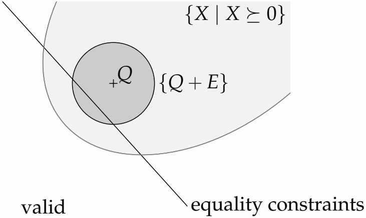
Floating-Point Arithmetic in the Analysis

Conservative check

- we "pad" the initial problem $M \prec 0$ into $M + \epsilon I \prec 0$ with ϵ greater than solver precision, e.g. $\epsilon := 10^{-7}$
- we check the soundness of the solution (P, γ) wrt the initial LMI.
  - LMI is instanciated into an exact matrix, computed with rational arithmetics
  - positiveness is checked with a conservative Cholesky decomposition using floats (algorithm proved in Coq)



Padding conic convex constraints



Checking feasibility

Conclusion

- Convex optimization is a powerful tool to perform automatic computation of non linear invariants
- Lyapunov function is the good approach to construct inductive invariants
- Applicable to large sets of programs, especially numerical controllers
- Floating point issues have to be carefully addressed
- Enable the analysis of control level properties at code level
- OSDP: Ocaml SDP library with integrated soundness checks

Conclusion

- Convex optimization is a powerful tool to perform automatic computation of non linear invariants
- Lyapunov function is the good approach to construct inductive invariants
- Applicable to large sets of programs, especially numerical controllers
- Floating point issues have to be carefully addressed
- Enable the analysis of control level properties at code level
- OSDP: Ocaml SDP library with integrated soundness checks

Thank you for your attention