Rigorous computation of viability kernel

Benjamin Martin[1] Olivier Mullier[2]

[1]LIX, École Polytechnique, Université Paris - Saclay
[2]U2IS, ENSTA Paristech, Université Paris - Saclay

26 August 2017
International Workshop on Methods and Tools for Distributed Hybrid Systems, Aalborg

Research financially supported by the DGA MRIS

Outline

Controlled dynamical system and viability kernel

We consider the following controlled nonlinear system:

$$(S) \begin{cases} \dot{x} = f(x, u) \\ x(0) = x_0 \end{cases}$$

with $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ the input and $x(t) \in \mathbb{R}^n$ the states, for all $t \geqslant 0$.

Controlled dynamical system and viability kernel

We consider the following controlled nonlinear system:

$$(S) \begin{cases} \dot{x} = f(x, u) \\ x(0) = x_0 \end{cases}$$

with $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ the input and $x(t) \in \mathbb{R}^n$ the states, for all $t \geqslant 0$.

Goal

Given a set of safe states K, find its viability kernel: the set of all initial conditions for which there always exists a trajectory remaining inside K for an indefinite amount of time.

Controlled dynamical system and viability kernel

We consider the following controlled nonlinear system:

$$(S)\begin{cases}\dot{x} = f(x,u)\\ x(0) = x_0\end{cases}$$

with $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$ the input and $x(t) \in \mathbb{R}^n$ the states, for all $t \geqslant 0$.

Viability Kernel

The viability kernel of a set $K \in \mathbb{R}^n$, is defined by:

$$\text{Viab}_S(K) := \{x_0 \in K | (\forall t \in [0,+\infty])(\exists u(t) \in \mathcal{U})(\varphi(x_0,u(t),t) \in K)\},$$

$\varphi(x_0,u(.),.)$ being a solution of (S) starting from x_0 with control input $u(.)$.

Computing the viability kernel

Some approaches from the literature:

Computing the viability kernel

Some approaches from the literature:

- discretization of time and states, low dimensional systems, not guaranteed (e.g. [Muhammad et al., 2015])

Computing the viability kernel

Some approaches from the literature:

- discretization of time and states, low dimensional systems, not guaranteed (e.g. [Muhammad et al., 2015])
- high dimensional (linear) system, Lagrangian methods, ellipsoids, finite time horizon (e.g. [Kaynama et al., 2012])

Computing the viability kernel

Some approaches from the literature:

- discretization of time and states, low dimensional systems, not guaranteed (e.g. [Muhammad et al., 2015])
- high dimensional (linear) system, Lagrangian methods, ellipsoids, finite time horizon (e.g. [Kaynama et al., 2012])
- polynomial system, iterative reachability analysis via Lyapunov-like functions (e.g. [She and Xue, 2013])

Computing the viability kernel

Some approaches from the literature:

- discretization of time and states, low dimensional systems, not guaranteed (e.g. [Muhammad et al., 2015])
- high dimensional (linear) system, Lagrangian methods, ellipsoids, finite time horizon (e.g. [Kaynama et al., 2012])
- polynomial system, iterative reachability analysis via Lyapunov-like functions (e.g. [She and Xue, 2013])

In [Monnet et al., 2015], a two phase framework implemented by interval-based methods:

1. build an initial inner-approximation of the viability kernel (implemented by construction of Lyapunov-like functions)

Computing the viability kernel

Some approaches from the literature:

- discretization of time and states, low dimensional systems, not guaranteed (e.g. [Muhammad et al., 2015])
- high dimensional (linear) system, Lagrangian methods, ellipsoids, finite time horizon (e.g. [Kaynama et al., 2012])
- polynomial system, iterative reachability analysis via Lyapunov-like functions (e.g. [She and Xue, 2013])

In [Monnet et al., 2015], a two phase framework implemented by interval-based methods:

I. build an initial inner-approximation of the viability kernel (implemented by construction of Lyapunov-like functions)

II. improving the inner-approximation by computing iteratively its capture basin (implemented by guaranteed numerical integration).

Illustration

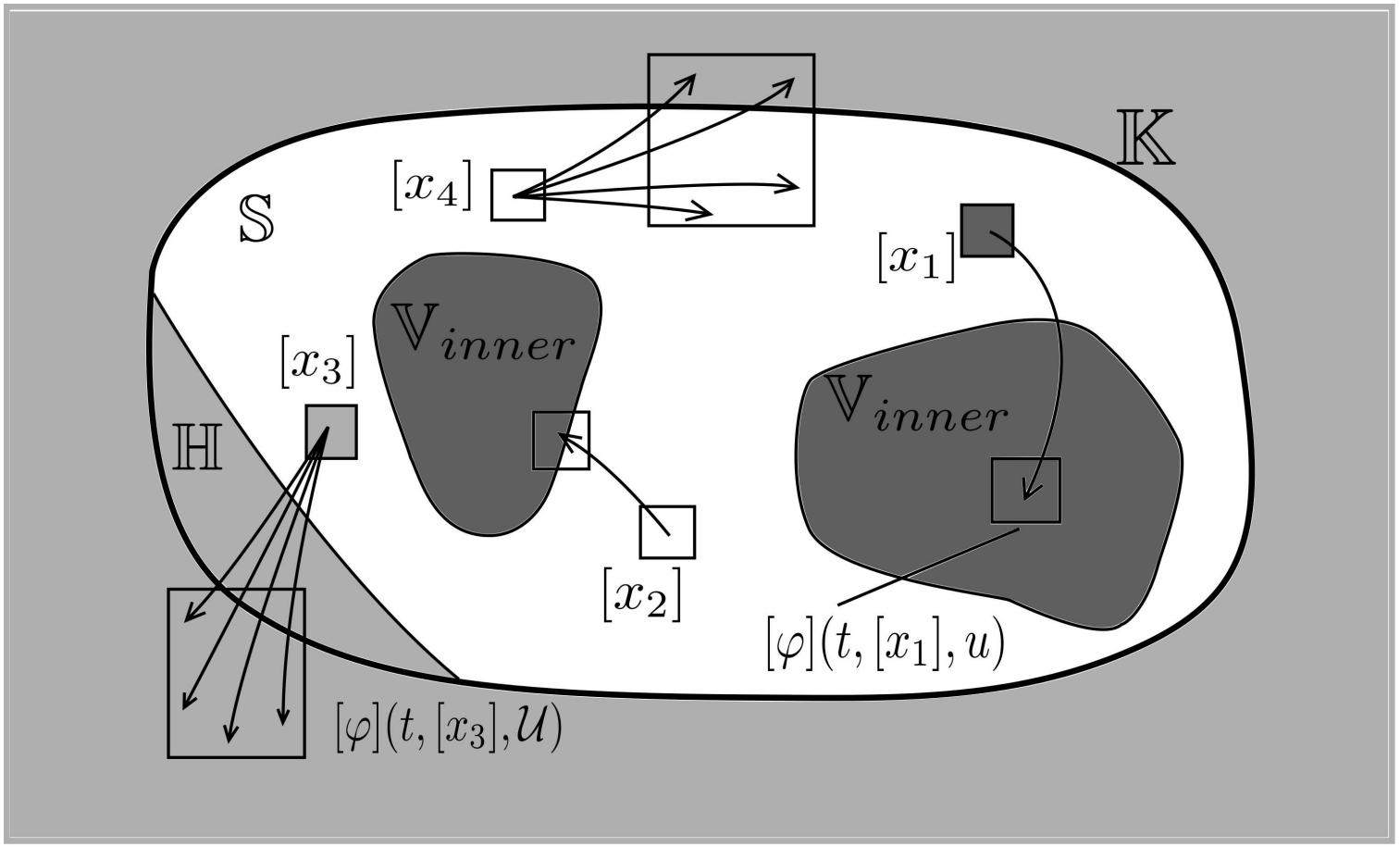
Capture basin

Given a target set T, the capture basin of T within K for a time horizon t_{end} is defined by:

$$\text{Capt}_S^{t_{end}}(K,T) := \left\{ x_0 \in K \;\middle|\; \begin{array}{c} (\exists \tilde{t} \in [0, t_{\text{end}}])(\exists \tilde{u} \in \mathcal{U})(\varphi(x_0, \tilde{u}(\tilde{t}), \tilde{t}) \in T \\ \wedge (\forall t \in [0, \tilde{t}])(\varphi(x_0, \tilde{u}(t), t) \in K)) \end{array} \right\}$$

Main idea: If $T \subseteq \text{Viab}_S(K)$, then $\text{Capt}_S^{t_{end}}(K,T) \subseteq \text{Viab}_S(K)$ for any $t_{\text{end}} \geqslant 0$.

Illustration



Taken from [Monnet et al., 2015]

Interval analysis

Intervals

An interval $[x]$ denotes a connected set of real values, e.g.

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\}.$$

Vectors of intervals are called boxes. The set of real intervals is denoted $\mathbb{IR}$.

Interval analysis

Intervals

An interval $[x]$ denotes a connected set of real values, e.g.

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\}.$$

Vectors of intervals are called boxes. The set of real intervals is denoted $\mathbb{IR}$.

Intervals replace real computations, allowing analysis over set of values

Interval analysis

Intervals

An interval $[x]$ denotes a connected set of real values, e.g.

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\}.$$

Vectors of intervals are called boxes. The set of real intervals is denoted $\mathbb{IR}$.

Intervals replace real computations, allowing analysis over set of values

- safe over-approximation by interval extension of functions (via interval arithmetic)

Interval analysis

Intervals

An interval $[x]$ denotes a connected set of real values, e.g.

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\}.$$

Vectors of intervals are called boxes. The set of real intervals is denoted $\mathbb{IR}$.

Intervals replace real computations, allowing analysis over set of values

- safe over-approximation by interval extension of functions (via interval arithmetic)
- proofs of existence and uniqueness of solutions to system of equations e.g. via interval Newton methods

Interval analysis

Intervals

An interval $[x]$ denotes a connected set of real values, e.g.

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\}.$$

Vectors of intervals are called boxes. The set of real intervals is denoted $\mathbb{IR}$.

Intervals replace real computations, allowing analysis over set of values

- safe over-approximation by interval extension of functions (via interval arithmetic)
- proofs of existence and uniqueness of solutions to system of equations e.g. via interval Newton methods
- safe removal of inconsistent values via interval contractors

Goal

Starting from the framework of [Monnet et al., 2015]:

Goal

Starting from the framework of [Monnet et al., 2015]:

- enhancing the improvement phase by guaranteed numerical integration (done by Olivier Mullier, not detailed in this talk)

Goal

Starting from the framework of [Monnet et al., 2015]:

- enhancing the improvement phase by guaranteed numerical integration (done by Olivier Mullier, not detailed in this talk)
- enhancing the first phase by generating large inner-approximations (this talk)

Goal

Starting from the framework of [Monnet et al., 2015]:

- enhancing the improvement phase by guaranteed numerical integration (done by Olivier Mullier, not detailed in this talk)
- enhancing the first phase by generating large inner-approximations (this talk)

Motivation

Taking more time for computing larger initial inner-approximation may reduce significantly the cost of the improvement phase.

First phase: state-of art approach (1)

The core idea is to find a function $h : \mathbb{R}^n \to \mathbb{R}$ of the states satisfying

$$\forall x \in K, \exists u \in \mathcal{U},\ h(x) \neq 0 \vee \langle \nabla_x h(x), f(x,u) \rangle < 0, \tag{1}$$

which is a Quantified Constraint Satisfaction Problem (QCSP).

First phase: state-of art approach (1)

The core idea is to find a function $h : \mathbb{R}^n \to \mathbb{R}$ of the states satisfying

$$\forall x \in K, \exists u \in \mathcal{U},\ h(x) \neq 0 \vee \langle \nabla_x h(x), f(x,u) \rangle < 0, \tag{1}$$

which is a Quantified Constraint Satisfaction Problem (QCSP).

Property

Let h be a function satisfying (1), define $\mathcal{H} := \{x \in K |\ h(x) \leqslant 0\}$. Then $\mathcal{H} \subset \mathrm{Viab}_S(K)$, provided the boundary of K and $\mathcal{H}$ does not intersects.

First phase: state-of art approach (1)

The core idea is to find a function $h : \mathbb{R}^n \to \mathbb{R}$ of the states satisfying

$$\forall x \in K, \exists u \in \mathcal{U},\ h(x) \neq 0 \vee \langle \nabla_x h(x), f(x,u) \rangle < 0, \tag{1}$$

which is a Quantified Constraint Satisfaction Problem (QCSP).

Property

Let h be a function satisfying (1), define $\mathcal{H} := \{x \in K |\ h(x) \leqslant 0\}$. Then $\mathcal{H} \subset \mathrm{Viab}_S(K)$, provided the boundary of K and $\mathcal{H}$ does not intersects.

The last part can be checked in the same manner:

$$\forall x \in K,\ x \notin \delta K \vee h(x) > 0, \tag{2}$$

where δK is the boundary of K.

First phase: original approach (2)

Implemented as follows:

1. choose a particular control $u \in \mathcal{U}$,

First phase: original approach (2)

Implemented as follows:

1. choose a particular control $u \in \mathcal{U}$,
2. compute equilibrium points of S_u, i.e. x^* such that $f(x^*, u) = 0$,

First phase: original approach (2)

Implemented as follows:

1. choose a particular control $u \in \mathcal{U}$,
2. compute equilibrium points of S_u, i.e. x^* such that $f(x^*, u) = 0$,
3. linearize S_u around x^*, obtain

$$(\tilde{S}_u) \begin{cases} \dot{\tilde{x}} = A\tilde{x} \ , \end{cases}$$

with $\tilde{x} = (x - x^*)$,

First phase: original approach (2)

Implemented as follows:

1. choose a particular control $u \in \mathcal{U}$,
2. compute equilibrium points of S_u, i.e. x^* such that $f(x^*, u) = 0$,
3. linearize S_u around x^*, obtain

$$(\tilde{S}_u) \begin{cases} \dot{\tilde{x}} = A\tilde{x} \ , \end{cases}$$

   with $\tilde{x} = (x - x^*)$,
4. find quadratic Lyapunov function of $\tilde{S}_u$. If found, x^* is stable,

First phase: original approach (2)

Implemented as follows:

1. choose a particular control $u \in \mathcal{U}$,
2. compute equilibrium points of S_u, i.e. x^* such that $f(x^*, u) = 0$,
3. linearize S_u around x^*, obtain

$$(\tilde{S}_u) \begin{cases} \dot{\tilde{x}} = A\tilde{x} \ , \end{cases}$$

   with $\tilde{x} = (x - x^*)$,
4. find quadratic Lyapunov function of $\tilde{S}_u$. If found, x^* is stable,
5. if not found try to find another control such that x^* is stable and compute the corresponding quadratic Lyapunov function,

First phase: original approach (2)

Implemented as follows:

1. choose a particular control $u \in \mathcal{U}$,
2. compute equilibrium points of S_u, i.e. x^* such that $f(x^*, u) = 0$,
3. linearize S_u around x^*, obtain

$$(\tilde{S}_u) \begin{cases} \dot{\tilde{x}} = A\tilde{x} \ , \end{cases}$$

   with $\tilde{x} = (x - x^*)$,
4. find quadratic Lyapunov function of $\tilde{S}_u$. If found, x^* is stable,
5. if not found try to find another control such that x^* is stable and compute the corresponding quadratic Lyapunov function,
6. noting by $V(x)$ this Lyapunov function, find the largest r such that $h(x) := V(x) - r$ satisfy (1) (dichotomic search with interval tests).

First phase: brief analysis

Original implementation

The proposed implementation is fast, but may fail to produce large inner-approximations.

First phase: brief analysis

Original implementation

The proposed implementation is fast, but may fail to produce large inner-approximations.

We propose to get back to the original statement. Given a family of parametric template functions $h_p : \mathbb{R}^n \to \mathbb{R}$, find $p \in \mathcal{P} \subset \mathbb{R}^q$ such that:

$$\forall x \in K, \exists u \in \mathcal{U},\ h_p(x) = 0 \vee \langle \nabla_x h(x), f(x,u) \rangle < 0 \tag{3}$$

First phase: brief analysis

Original implementation

The proposed implementation is fast, but may fail to produce large inner-approximations.

We propose to get back to the original statement. Given a family of parametric template functions $h_p : \mathbb{R}^n \to \mathbb{R}$, find $p \in \mathcal{P} \subset \mathbb{R}^q$ such that:

$$\forall x \in K, \exists u \in \mathcal{U},\ h_p(x) = 0 \vee \langle \nabla_x h(x), f(x,u) \rangle < 0 \tag{3}$$

Additionally, find a $p \in \mathcal{P}$ verifying (3) that maximizes the volume of $\mathcal{H}_p := \{x \in K |\ h_p(x) \leqslant 0\}$.

How to solve (3)

Barrier function

In [Djaballah et al., 2017], an interval method is proposed for computing barrier functions, which requires to satisfy (3) (and extra other constraints). Uncontrolled case.

How to solve (3)

Barrier function

In [Djaballah et al., 2017], an interval method is proposed for computing barrier functions, which requires to satisfy (3) (and extra other constraints). Uncontrolled case.

Computable Sufficient Condition - Feasible Point Searcher (CSC-FPS) algorithm.

- FPS (branching algorithm on p): explore the set of parameters, each iteration applying the CSC step. Stops once a feasible parameter p is found;

How to solve (3)

Barrier function

In [Djaballah et al., 2017], an interval method is proposed for computing barrier functions, which requires to satisfy (3) (and extra other constraints). Uncontrolled case.

Computable Sufficient Condition - Feasible Point Searcher (CSC-FPS) algorithm.

- FPS (branching algorithm on p): explore the set of parameters, each iteration applying the CSC step. Stops once a feasible parameter p is found;
- CSC (branching algorithm on x): checks whether (a.) an initially selected point in a given box $[p]$ satisfy the universally quantified constraints, (b.) that for all $p \in [p]$ there exists an x which contradicts the constraints, (c.) not decidable.

How to solve (3)

Barrier function

In [Djaballah et al., 2017], an interval method is proposed for computing barrier functions, which requires to satisfy (3) (and extra other constraints). Uncontrolled case.

Computable Sufficient Condition - Feasible Point Searcher (CSC-FPS) algorithm.

- FPS (branching algorithm on p): explore the set of parameters, each iteration applying the CSC step. Stops once a feasible parameter p is found;
- CSC (branching algorithm on x): checks whether (a.) an initially selected point in a given box $[p]$ satisfy the universally quantified constraints, (b.) that for all $p \in [p]$ there exists an x which contradicts the constraints, (c.) not decidable.

Use of contractors [Chabert and Jaulin, 2009] to reduce further the boxes $[p]$ and $[x]$.

CSC-FPS: discussion

Pros and Cons

A branching algorithm on the parameters nesting another branching algorithm on the states.

- Computationally expensive (low dimensions), but it is not necessary to explore all parameters.
- Numerically guaranteed and handles transcendental functions.

CSC-FPS: discussion

Pros and Cons

A branching algorithm on the parameters nesting another branching algorithm on the states.

- Computationally expensive (low dimensions), but it is not necessary to explore all parameters.
- Numerically guaranteed and handles transcendental functions.

Limitations for our application

I. I. CSC cannot contradict (3), i.e. prove that a p satisfy:

$$\exists x \in K,\ h_p(x) = 0 \wedge \langle \nabla_x h(x), f(x,u) \rangle \geqslant 0.$$

II. Does not consider the "quality" of the returned parameter.
III. Uncontrolled case.

What we propose for I.

I. : Counter-examples

In [Ishii et al., 2012], a Branch & Prune method for solving problems as negation of (3) (uncontrolled case) $\rightarrow$ Use of parametric interval Newton for finding counter-examples to (3).

What we propose for I.

I. : Counter-examples

In [Ishii et al., 2012], a Branch & Prune method for solving problems as negation of (3) (uncontrolled case) → Use of parametric interval Newton for finding counter-examples to (3).

Definition

A parametric interval newton operator with respect to a system of equation $h(.,.) = 0$ $(h : \mathbb{R}^{q+n} \to \mathbb{R}^n)^a$ and a parameter box $[p] \in \mathbb{IR}^q$, written $\mathcal{N}_{[p]}(.) : \mathbb{IR}^n \to \mathbb{IR}^n$, is an operator satisfying:

$$\forall [x] \in \mathbb{IR}^n, \mathcal{N}_{[p]}([x]) \subset [x] \implies \forall p \in [p], \exists x \in [x], h(p,x) = 0$$

[a] With fewer equations than n, select some states x_i as parameters.

What we propose for II

II. : Optimizing via Branch & Bound

Embedding the FPS loop into a Branch & Bound optimization algorithm:

- Each iteration selects a preferred box $[p]$ from the branching
- If a feasible point $p \in [p]$ is found, eliminates the boxes from the branching whose elements are necessarily worse
- Stops when a sufficiently good point $\hat{p}$ is found

What we propose for II

II. : Optimizing via Branch & Bound

Embedding the FPS loop into a Branch & Bound optimization algorithm:

- Each iteration selects a preferred box $[p]$ from the branching
- If a feasible point $p \in [p]$ is found, eliminates the boxes from the branching whose elements are necessarily worse
- Stops when a sufficiently good point $\hat{p}$ is found

Difficulties

- Definition of a "best" parameter p (which objective function to use for which template)
- Efficient exploration strategy. Following [Neveu et al., 2016], we chose two orderings of the parameter boxes: min LB and max UB. Emphasis on likely feasible parameters ?

What we propose for III.

III. Relaxation by sampling feasible controls

Relaxation of (3): consider a finite set of control functions $\{u_1, u_2, \ldots, u_k\}$, then (3) can be relaxed to:

$$\forall x \in K \; h_p(x) \neq 0 \bigvee_{i=1}^{k} \langle \nabla_x h_p(x), f(x, u_i) \rangle < 0.$$

Choices for the u_i: constant (interval) function, state feedback, ...

Numerical results

Two phase algorithm implemented in Ibex [Chabert and Jaulin, 2009] using DynIbex [2] providing rigorous numerical integration tools using Runge-Kutta methods [Alexandre dit Sandretto and Chapoutot, 2016] for the improvement phase.

[2] `http://perso.ensta-paristech.fr/~chapoutot/dynibex/`

Car on the hill

Consider the system:

$$\begin{cases} \dot{x} = v \\ \dot{v} = -9.81 \sin\left(\frac{1.1\sin(1.2x) - 1.2\sin(1.1x)}{2}\right) - 0.7v + u, \end{cases} \tag{4}$$

with $K = ([-1, 13] \times [-6, 6])^T$ and $u \in [-3, 3]$.

Car on the hill

Consider the system:

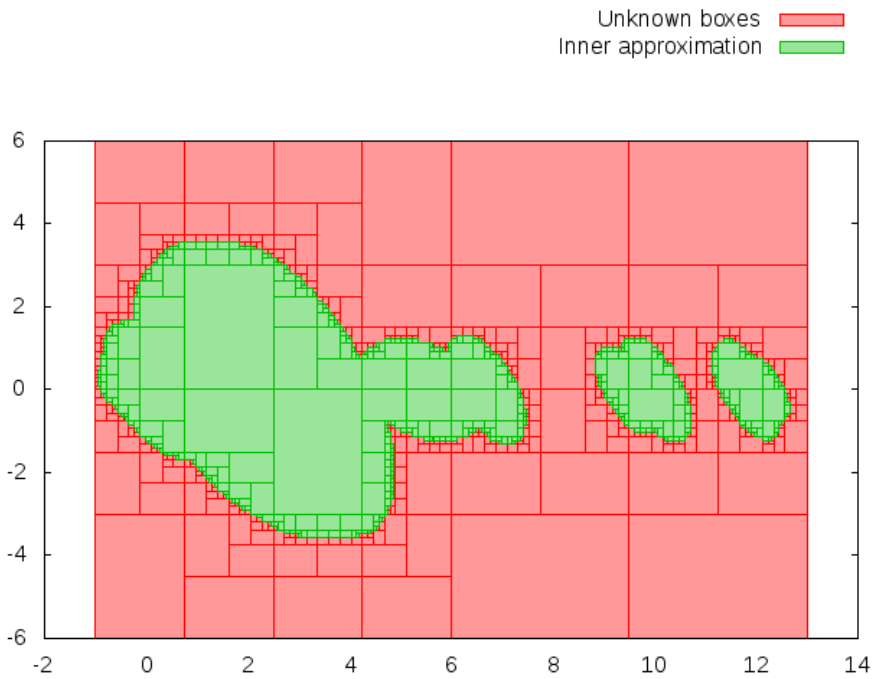
$$\begin{cases} \dot{x} = v \\ \dot{v} = -9.81 \sin\left(\frac{1.1\sin(1.2x) - 1.2\sin(1.1x)}{2}\right) - 0.7v + u, \end{cases} \tag{4}$$

with $K = ([-1, 13] \times [-6, 6])^T$ and $u \in [-3, 3]$.
Considering the equilibrium points $(x^*, 0)$ with control $u = 0$, we use the following 3-parametric template function:

$$h_p(x, v) := p_1(x - x^*)^2 + p_2 v^2 + p_3(x - x^*)v - 1, \tag{5}$$

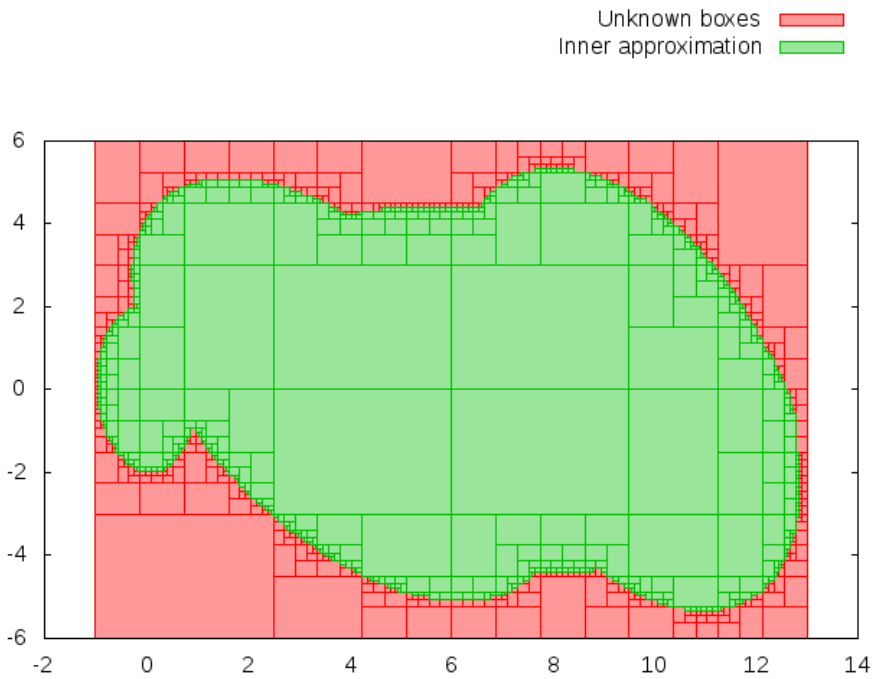
and we impose its positive definiteness, i.e. h_p is a quadratic convex function (additional constraints on p). We minimize the norm of (p_1, p_2).
First phase stopped after 10 seconds for each equilibrium points.

Initial inner-approximation



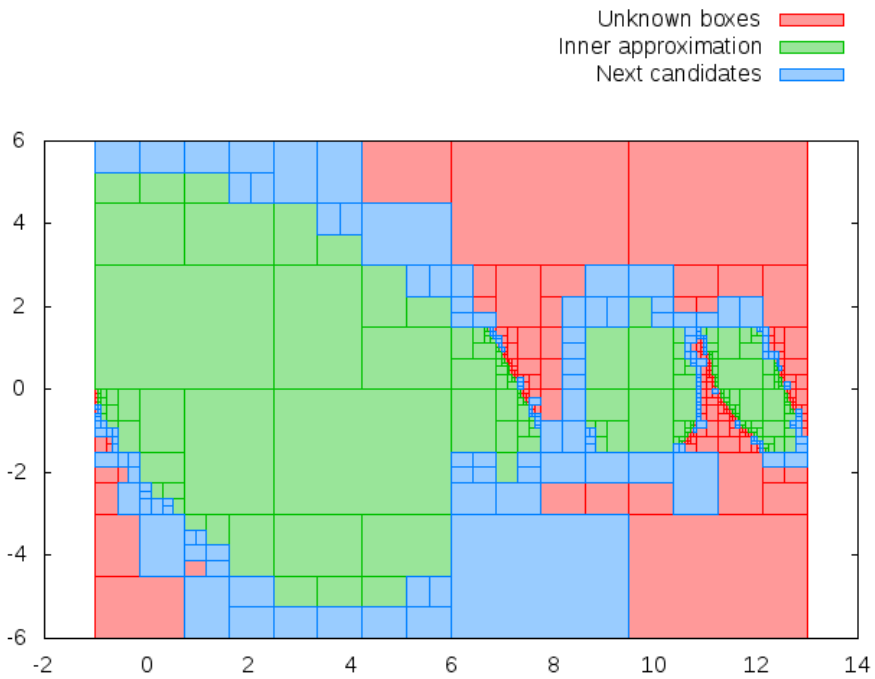
Original method

Initial inner-approximation



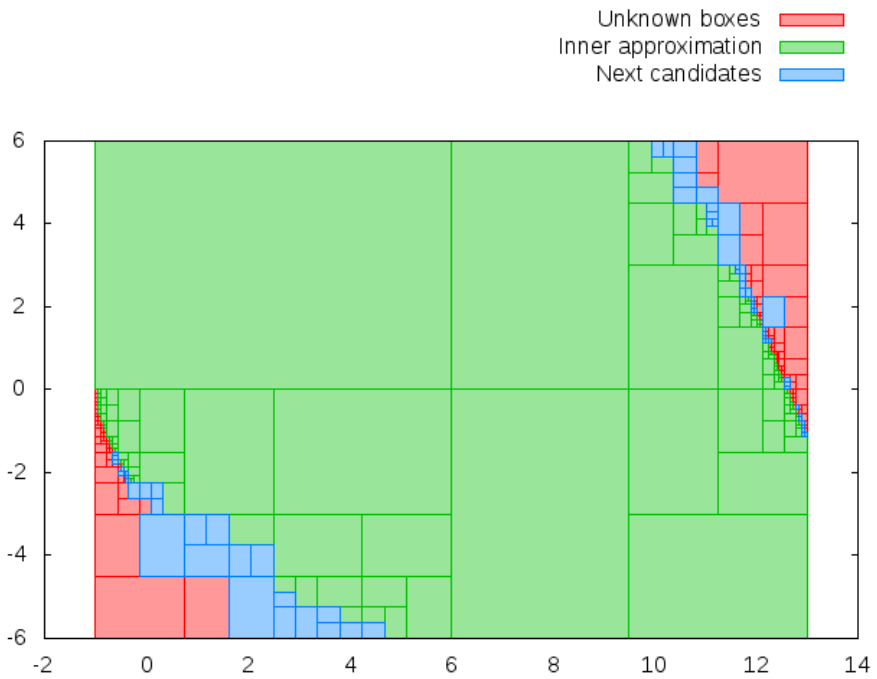
New method

After 5 iterations of the improvement procedure



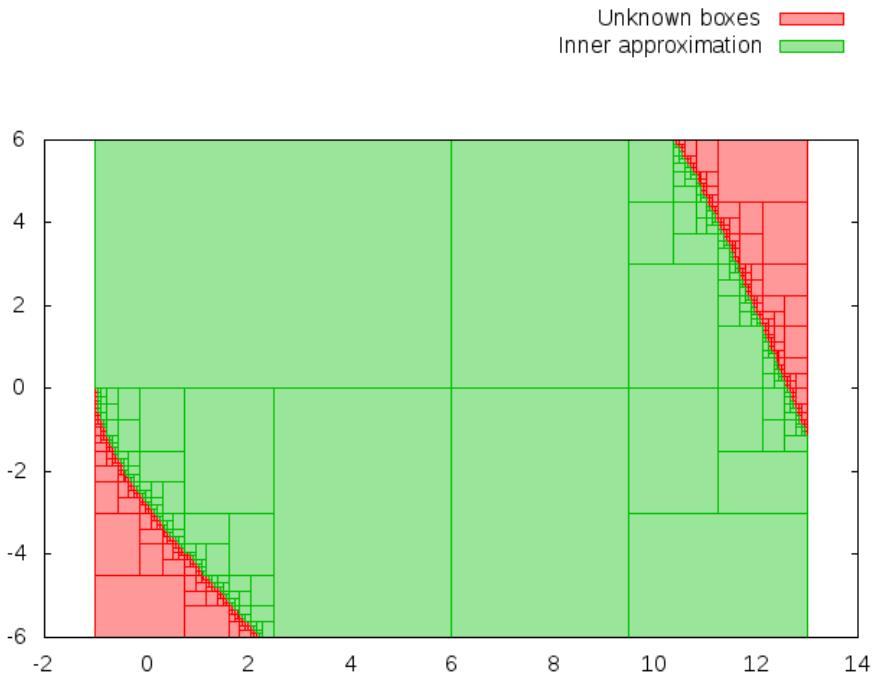
Original method

After 5 iterations of the improvement procedure



New method

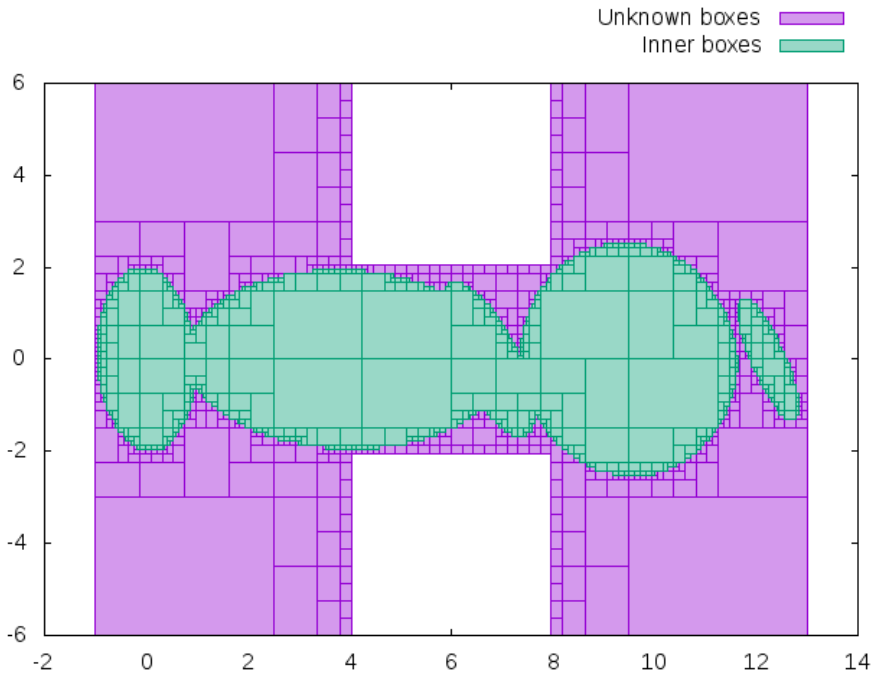
Final inner-approximation



Analysis

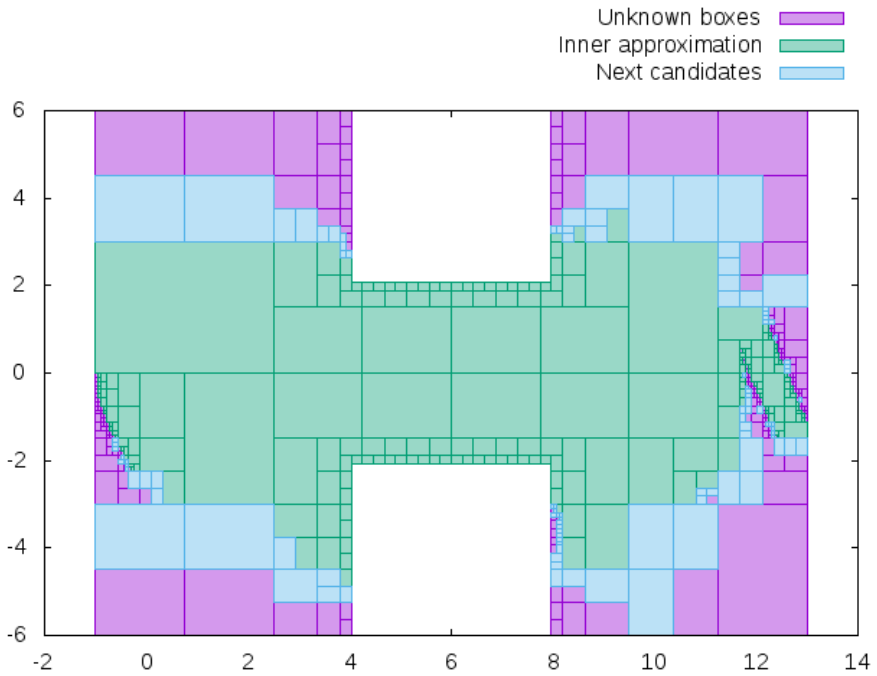
Original method: 27 iterations of improvement procedure, about 360 seconds (3 seconds for the first phase)
New method: 18 iterations of improvement procedure about 175 seconds (34 seconds for the first phase)

Non-convex K



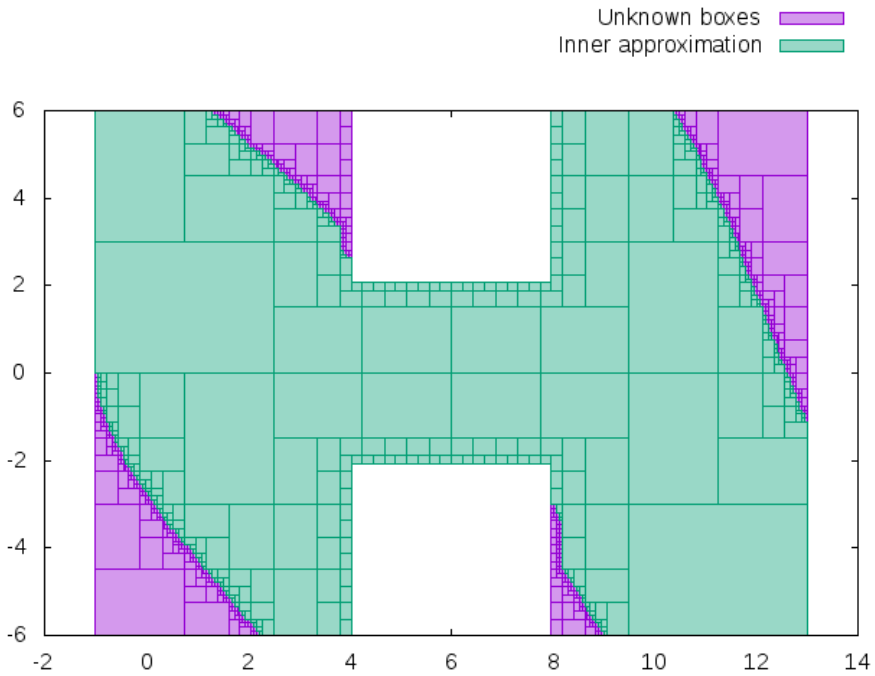
Initial inner-approximation

Non-convex K



After 5 iteration of improvement phase

Non-convex K



Final inner-approximation: 22 iterations, 305 seconds (50s for the first phase)

Conclusion

Another approach for finding an initial inner-approximation of the viability kernel:

Conclusion

Another approach for finding an initial inner-approximation of the viability kernel:

- generalization of the method from [Monnet et al., 2015];

Conclusion

Another approach for finding an initial inner-approximation of the viability kernel:

- generalization of the method from [Monnet et al., 2015];
- much higher cost, but trades favorably with the gain in the improvement phase

Conclusion

Another approach for finding an initial inner-approximation of the viability kernel:

- generalization of the method from [Monnet et al., 2015];
- much higher cost, but trades favorably with the gain in the improvement phase

Future work:

- scalability ? different templates (e.g. semi-algebraic sets) ?

Conclusion

Another approach for finding an initial inner-approximation of the viability kernel:

- generalization of the method from [Monnet et al., 2015];
- much higher cost, but trades favorably with the gain in the improvement phase

Future work:

- scalability ? different templates (e.g. semi-algebraic sets) ?
- measuring likeliness of feasibility of $[p]$: possibility of more efficient exploration strategy and finding better solutions quicker;

Conclusion

Another approach for finding an initial inner-approximation of the viability kernel:

- generalization of the method from [Monnet et al., 2015];
- much higher cost, but trades favorably with the gain in the improvement phase

Future work:

- scalability ? different templates (e.g. semi-algebraic sets) ?
- measuring likeliness of feasibility of $[p]$: possibility of more efficient exploration strategy and finding better solutions quicker;
- a better method for finding counter-examples or feasible parameters (see e.g. [Hladík and Ratschan, 2014]);

Conclusion

Another approach for finding an initial inner-approximation of the viability kernel:

- generalization of the method from [Monnet et al., 2015];
- much higher cost, but trades favorably with the gain in the improvement phase

Future work:

- scalability ? different templates (e.g. semi-algebraic sets) ?
- measuring likeliness of feasibility of $[p]$: possibility of more efficient exploration strategy and finding better solutions quicker;
- a better method for finding counter-examples or feasible parameters (see e.g. [Hladík and Ratschan, 2014]);
- (meta)heuristic based optimization loop.

Thank you for your attention

References I

[Alexandre dit Sandretto and Chapoutot, 2016] Alexandre dit Sandretto, J. and Chapoutot, A. (2016).
Validated explicit and implicit runge-kutta methods.
Reliable Computing, 22:79–103.

[Chabert and Jaulin, 2009] Chabert, G. and Jaulin, L. (2009).
Contractor programming.
Artificial Intelligence, 173(11):1079 – 1100.

[Djaballah et al., 2017] Djaballah, A., Chapoutot, A., Kieffer, M., and Bouissou, O. (2017).
Construction of parametric barrier functions for dynamical systems using interval analysis.
Automatica, 78:287 – 296.

References II

[Hladík and Ratschan, 2014] Hladík, M. and Ratschan, S. (2014).
Efficient solution of a class of quantified constraints with quantifier prefix exists-forall.
Mathematics in Computer Science, 8(3):329–340.

[Ishii et al., 2012] Ishii, D., Goldsztejn, A., and Jermann, C. (2012).
Interval-based projection method for under-constrained numerical systems.
Constraints, 17(4):432–460.

[Kaynama et al., 2012] Kaynama, S., Maidens, J., Oishi, M., Mitchell, I. M., and Dumont, G. A. (2012).
Computing the viability kernel using maximal reachable sets.
In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '12, pages 55–64, New York, NY, USA. ACM.

References III

[Monnet et al., 2015] Monnet, D., Jaulin, L., Ninin, J., Chapoutot, A., and Alexandre-dit Sandretto, J. (2015).
Viability kernel computation based on interval methods.
In *SWIM (Summer Workshop on Interval Analysis)*.

[Muhammad et al., 2015] Muhammad, B., Fraichard, T., and Fezari, M. (2015).
Safe motion using viability kernels.
In *ICRA 2015-IEEE Int. Conf. on Robotics and Automation*.

[Neveu et al., 2016] Neveu, B., Trombettoni, G., and Araya, I. (2016).
Node selection strategies in interval Branch and Bound algorithms.
Journal of Global Optimization, 64(2):289–304.

References IV

[She and Xue, 2013] She, Z. and Xue, B. (2013).

Computing an invariance kernel with target by computing lyapunov-like functions.

IET Control Theory & Applications, 7(15):1932–1940.

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;
5. if feasible parameter found, stop algorithm and returns it;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;
5. if feasible parameter found, stop algorithm and returns it;
6. else if no feasible parameter exists go to 1.;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;
5. if feasible parameter found, stop algorithm and returns it;
6. else if no feasible parameter exists go to 1.;
7. otherwise branch on $[p]_0$, go to 1.;

CSC-FPS (1)

FPS loop consists of a branching algorithm on an initial box $[p]$:
Inputs: boxes $[p]$, $[x]$
Main loop (while $Q \neq 0$):

1. $[p]_0 \leftarrow \mathrm{pop}(Q)$;
2. Contract $[p]_0$ using some samples in $[x]$ (proj-inter contractor [Chabert and Jaulin, 2009]);
3. if empty go to 1. otherwise continue;
4. apply the CSC procedure on $[p]_0$;
5. if feasible parameter found, stop algorithm and returns it;
6. else if no feasible parameter exists go to 1.;
7. otherwise branch on $[p]_0$, go to 1.;

If never stopped by 5., limit precision is reached then it is not decidable, otherwise no feasible parameter in $[p]$.

CSC-FPS (2)

CSC loop consists of a branching algorihtm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;

CSC-FPS (2)

CSC loop consists of a branching algorihtm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;

CSC-FPS (2)

CSC loop consists of a branching algorihtm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;
3. if constraint locally satisfied for $\tilde{p}$, go to 1.;

CSC-FPS (2)

CSC loop consists of a branching algorihtm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;
3. if constraint locally satisfied for $\tilde{p}$, go to 1.;
4. try to find a $x \in [x]_0$ not satisfying the constraints for all $p \in [p]$, if one found returns "no feasible parameter";

CSC-FPS (2)

CSC loop consists of a branching algorihtm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;
3. if constraint locally satisfied for $\tilde{p}$, go to 1.;
4. try to find a $x \in [x]_0$ not satisfying the constraints for all $p \in [p]$, if one found returns "no feasible parameter";
5. otherwise branch on $[x]_0$, go to 1.

CSC-FPS (2)

CSC loop consists of a branching algorihtm on an initial box $[x]$:
Inputs: boxes $[p]$, $[x]$, a point $\tilde{p} \in [p]$
Main loop (while $S \neq 0$)

1. $[x]_0 \leftarrow \mathrm{pop}(S)$;
2. Contract $[x]_0$ with respect to $\tilde{p}$;
3. if constraint locally satisfied for $\tilde{p}$, go to 1.;
4. try to find a $x \in [x]_0$ not satisfying the constraints for all $p \in [p]$, if one found returns "no feasible parameter";
5. otherwise branch on $[x]_0$, go to 1.

If never stopped by 4., limit precision reached then it is not decidable, otherwise $\tilde{p}$ satisfies the QCSP.