Computation of Parametric Barrier Functions for Dynamical Systems using Interval Analysis

Olivier Bouissou, Alexandre Chapoutot, Adel Djaballah, Michel Kieffer

Computation of Parametric Barrier Functions for Dynamical Systems using Interval Analysis

Olivier Bouissou[1], Alexandre Chapoutot[2], Adel Djaballah[3] and Michel Kieffer[4]

***Abstract*— The formal verification of safety properties for hybrid systems is an important but challenging problem. Recently, barrier functions have been introduced to prove safety without requiring the computation of the reachable set of continuous or hybrid dynamical systems.**

This paper presents a new approach for the construction of barrier functions for safety verification of nonlinear dynamical systems. The proposed method is based on the search for the parameters of a parametric barrier function using interval analysis. This technique allows considering complex dynamics without needing any relaxation of constraints in the barrier function.

I. INTRODUCTION

Formal verification aims at proving that a certain behavior or property is fulfilled by a system. Verifying, *e.g.*, safety properties consists in ensuring that the system will never reach a dangerous or an unwanted configuration. Safety verification is usually translated into a reachability problem [1]. Starting from an initial region, a system must not reach some unsafe region. Different methods have been considered to address with this problem [2], [3]. One way is to explicitly compute the reachable region and to determine whether it contains the unsafe region [4]. Alternatively, one may compute an invariant for the system, *i.e.*, a region where the system will always evolve [5]. This paper considers a class of invariants delimited by *barrier functions*.

A barrier function [6] defines an uncrossable frontier between the initial region and the unsafe region. When the system initial state is in the initial region, the barrier function guarantees that the system will never reach the unsafe region, providing a *barrier certificate* for the safety of the system. The main challenge lies in the computation of barrier functions. In [6] polynomial barrier functions are efficiently designed for polynomial systems. Here, the aim is to extend the class of problems addressed by [6] to non-polynomial systems, and non-polynomial barrier functions. In [6], hybrid systems are considered. This paper focuses on continuous-time systems; the extension to hybrid systems will be consider in future works.

The barrier function has to satisfy some constraints which are formulated as quantified constraints to be solved using interval analysis. The proposed approach involves validation and invalidation of constraints based on a technique introduced in [7]. It is improved by the use of constraint propagation techniques as introduced in [8], [9].

The paper is organized as follows. Section II briefly recalls some related work. Section III introduces barrier functions and formulates the constraints they should satisfy to provide a barrier certificate. Section IV presents the framework developed to solve the constraints. Experimental results are given in Section V before drawing some conclusions in Section VI.

Small italic letters x represent real variables while real vectors $\mathbf{x}$ are in bold. Intervals $[x]$ and interval vectors (boxes) $[\mathbf{x}]$ are represented between brackets. Data structure or sets $\mathcal{S}$ are in upper-case calligraphic. The derivative of a function x with respect to time t is denoted $\dot{x}$.

II. RELATED WORK

To prove the safety of a system, different approaches have been proposed [10]. One way is to explicitly compute an approximation of the reachable region from the initial region. The system is safe when the reachable region does not intersect the unsafe region. In [3], [4], [11] the reachable region is computed for linear hybrid systems for a finite time horizon using geometric representations such as polyhedra. The reachable region for non-linear systems is computed in [2] using an abstraction of the non-linear systems by a linear system expressed in a new system of coordinates. The problem of reachability of non-linear systems is formulated as an optimization problem in [12]. In [13], a Picard iterator is combined with Taylor models to find the reachable region for non-linear hybrid systems.

An other way to address the safety problem is by exhibiting an invariant region in which the system remains. If the invariant does not intersect the unsafe region then the safety of the system is proved. One way to find such an invariant is by using stability properties of a dynamical system [14] and searching for a Lyapunov function. In [15], a sum-of-square decomposition and semi-definite programming are employed to find a Lyapunov function for a system with polynomial dynamics. A template approach is considered in [16] to find Lyapunov functions using a branch-and-relax scheme and linear programming to solve the constraints induced. In [6], instead of looking for a function that fulfills some stability

This work was partly supported by Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02) and by the ANR INS Project CAFEIN (ANR-12-INSE-0007).

[1]Olivier Bouissou, CEA Saclay Nano-INNOV Institut CARNOT, 91191 Gif-sur-Yvette olivier.bouissou@cea.fr

[2]Alexandre Chapoutot, ENSTA ParisTech, 91762 Palaiseau, alexandre.chapoutot@ensta.fr

[3]Adel Djaballah, ENSTA ParisTech, 91762 Palaiseau, adel.djaballah@ensta.fr

[4]Michel Kieffer, L2S, CNRS, Supélec, Univ. Paris-Sud 91192 Gif-sur-Yvette and Institut Universitaire de France 75005 Paris, michel.kieffer@lss.supelec.fr

conditions, a function is searched that separates the initial region from the unsafe region. Then, [17] extends the idea to search for invariants in conjunctive normal form for hybrid systems.

III. Formulation

A. Safety for continuous-time system

Consider the autonomous and time-invariant continuous-time dynamical system described by

$$\dot{\mathbf{x}} = f(\mathbf{x}), \tag{1}$$

where $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$. We assume that classical hypotheses on f are satisfied so that (1) has a unique solution $\mathbf{x}(t, \mathbf{x}_0) \in \mathcal{X}$ for a given initial value $\mathbf{x}_0 \in \mathcal{X}$ at time $t = 0$.

Definition 1: Consider an initial region $\mathcal{X}_0 \subseteq \mathcal{X}$, an unsafe region $\mathcal{X}_u \subseteq \mathcal{X}$. The system (1) is safe if $\forall \mathbf{x} \in \mathcal{X}_0$ and $\forall t \geq 0$, $\mathbf{x}(t, \mathbf{x}_0) \notin \mathcal{X}_u$, *i.e.*, the system never reaches $\mathcal{X}_u$ from $\mathcal{X}_0$.

B. Barrier certificates

One way to prove that (1) is safe is by the barrier certificate approach introduced in [6]. A barrier is a differentiable function $B : \mathcal{X} \to \mathbb{R}$ that partitions the state-space $\mathcal{X}$ into $\mathcal{X}_-$ where $B(\mathbf{x}) \leq 0$ and $\mathcal{X}_+$ where $B(\mathbf{x}) > 0$ such that $\mathcal{X}_0 \subseteq \mathcal{X}_-$ and $\mathcal{X}_u \subseteq \mathcal{X}_+$. Moreover, B has to be such that $\forall \mathbf{x}_0 \in \mathcal{X}_0,\ \forall t \geq 0\ B(\mathbf{x}(t, \mathbf{x}_0)) \leq 0$. Proving that $B(\mathbf{x}(t, \mathbf{x}_0)) \leq 0$ requires an evaluation of the solution of (1) for all $\mathbf{x}_0 \in \mathcal{X}_0$. Alternatively, [6] proposes the following theorem providing some constraints a barrier function has to satisfy to prove the safety of a dynamical system.

Theorem 1: Consider f defined in (1), and the sets $\mathcal{X}$, $\mathcal{X}_0$ and $\mathcal{X}_u$. If there exists a function $B : \mathcal{X} \to \mathbb{R}$ such that:

$$\begin{cases} \forall \mathbf{x} \in \mathcal{X}_0, \quad B(\mathbf{x}) \leq 0, & \text{(2a)} \\ \forall \mathbf{x} \in \mathcal{X}_u, \quad B(\mathbf{x}) > 0, & \text{(2b)} \\ \forall \mathbf{x} \in \mathcal{X}\ , B(\mathbf{x}) = 0 \Rightarrow \left\langle \dfrac{\partial B}{\partial \mathbf{x}}(\mathbf{x}), f(\mathbf{x}) \right\rangle < 0, & \text{(2c)} \end{cases}$$

then the dynamical system defined by f is safe.

Note that $\langle ., . \rangle$ stands for the scalar product in $\mathbb{R}^n$. In Theorem 1, (2a) and (2b) ensure that $\mathcal{X}_0 \subseteq \mathcal{X}_-$, and $\mathcal{X}_u \subseteq \mathcal{X}_+$, while (2c) states that when $\mathbf{x}$ is on the border between $\mathcal{X}_-$ and $\mathcal{X}_+$ (*i.e.*, $B(\mathbf{x}) = 0$), then the dynamics f pushes the state back in $\mathcal{X}_-$.

C. Parametric barrier functions

The search for a barrier B is challenging since it is over a functional space. One of the idea to address this problem is to consider barriers belonging to a family of parametric functions (or templates) $B(\mathbf{x}, \mathbf{p})$ depending on a fixed number of parameters $\mathbf{p} \in \mathcal{P} \subseteq \mathbb{R}^n$. Then one may search for the parameter values such that $B(\mathbf{x}, \mathbf{p})$ satisfies (2a)-(2c). Theorem 1 may then be reformulated by replacing $B(\mathbf{x})$ by $B(\mathbf{x}, \mathbf{p})$.

If there is no $\mathbf{p} \in \mathcal{P}$ such that $B(\mathbf{x}, \mathbf{p})$ satisfies (2a)-(2c), this does not mean that the system is not safe: Other structures of functions $B(\mathbf{x}, \mathbf{p})$ could provide a barrier certificate.

IV. Characterization using interval analysis

In this section we present an approach to find a parametric barrier $B(\mathbf{x}, \mathbf{p})$ that satisfies the constraints in Theorem 1. For that purpose, we use tools from interval analysis [9], [18] and reformulate the constraints of Theorem 1 so that they can be efficiently handled by constraint propagation techniques.

A. Interval analysis

An interval $[\underline{x}, \overline{x}] = \{x \in \mathbb{R} | \underline{x} \leq x \leq \overline{x}\}$ is defined by its lower and upper bounds $\underline{x}$ and $\overline{x}$. An interval vector (or box) $[\mathbf{x}]$ is a Cartesian product of intervals $[\underline{x}_0, \overline{x}_0] \times ... \times [\underline{x}_n, \overline{x}_n]$. $\mathbb{IR}$ denotes the set of bounded intervals over $\mathbb{R}$. All classical arithmetic operations are extended to intervals. An inclusion function $[f] : \mathbb{IR} \to \mathbb{IR}$ for $f : \mathbb{R} \to \mathbb{R}$ satisfies

$$\forall [x] \in \mathbb{IR}\ \{f(x) \mid x \in [x]\} \subseteq [f]([x]). \tag{3}$$

A natural inclusion function $[f]$ is obtained by substituting all variables and operations involved in f by their interval counterpart. The evaluation of the range of functions over intervals using inclusion function may introduce some over approximation, see [9], [18] for more details.

B. Interval formulation

Interval analysis is employed to find a parametric barrier function which satisfies the constraints introduced by Theorem 1. In what follows we assume that $\mathcal{X}_0$, $\mathcal{X}_u$, $\mathcal{X}$, and $\mathcal{P}$ are boxes denoted $[\mathbf{x}_0]$, $[\mathbf{x}_\mathrm{u}]$, $[\mathbf{x}_\mathrm{s}]$, and $[\mathbf{p}]$. The property $(p \implies q) \iff (\neg p \vee q)$ is used to rewrite Constraint (2c) in a form more amenable to a solution via interval analysis as follows

$\exists \mathbf{p} \in [\mathbf{p}]$,

$$\begin{cases} \forall \mathbf{x} \in [\mathbf{x}_0],\ B(\mathbf{x}, \mathbf{p}) \leq 0, & \text{(4a)} \\ \forall \mathbf{x} \in [\mathbf{x}_\mathrm{u}],\ B(\mathbf{x}, \mathbf{p}) > 0, & \text{(4b)} \\ \forall \mathbf{x} \in [\mathbf{x}_\mathrm{s}],\ B(\mathbf{x}, \mathbf{p}) \neq 0 \vee \left\langle \dfrac{\partial B}{\partial \mathbf{x}}(\mathbf{x}), f(\mathbf{x}) \right\rangle < 0. & \text{(4c)} \end{cases}$$

C. Solving the constraints

Different approaches exist to handle the quantified constraints (4a)-(4c). A branch-and-prune approach is presented in [19]. It performs branching over constraints and eliminates all irrelevant points. A method to solve quantified semi-algebraic constraints is described in [16]. In addition of branching over the constraints, the existential constraint ($\exists$) are solved using linear programming methods. Both methods are implemented in *RSolver* [20].

To address problems with non-polynomial dynamics and design non-polynomial barrier functions, we use the CSC-FPS algorithm introduced in [7]. Here, CSC-FPS is supplemented by constraint propagation techniques to improve its efficiency.

The algorithm is based on two procedures FPS and CSC. FPS (*Feasible Point Searcher*) branches over the search box $[\mathbf{p}]$ in parameter space. Branching is decided depending on the result of CSC (*Computable Sufficient Conditions*), which evaluate the constraints over $[\mathbf{p}]$. CSC returns `true` if there exists some vector of parameters $\mathbf{p} \in [\mathbf{p}]$ that

satisfies the constraints. It returns `false` if no vector of parameters satisfies the constraints. If no decision can be made, `unknown` is returned.

The two first constraints (4a)-(4b) may be easily reformulated as

$$\exists \mathbf{p} \in [\mathbf{p}], \forall \mathbf{x} \in [\mathbf{x}] \quad f(\mathbf{x}, \mathbf{p}) \in [\mathbf{z}]. \tag{5}$$

This formulation is more suitable for the use of constraint propagation techniques. The constraint (4c) requires a specific treatment.

1) CSC: CSC, see Algorithm 1, first verifies if (5) cannot be satisfied by determining if there exists $\mathbf{x}' \in [\mathbf{x}]$ such that

$$\forall \mathbf{p} \in [\mathbf{p_0}] \; f(\mathbf{x}', \mathbf{p}) \notin [\mathbf{z}]. \tag{6}$$

If (6) is satisfied, CSC returns `false`. Then, CSC evaluates whether there exists $\mathbf{p}' \in [\mathbf{p_0}]$ such that

$$\forall \mathbf{x} \in [\mathbf{x}] \; f(\mathbf{x}, \mathbf{p}') \in [\mathbf{z}]. \tag{7}$$

If such $\mathbf{p}'$ exists, (5) is satisfied. CSC returns `true` only when (5) is satisfied for all boxes in the stack $\mathcal{S}$ and for the *same* value $\mathbf{p}' \in [\mathbf{p_0}]$.

Different strategies may be considered to choose $\mathbf{p}'$ and $\mathbf{x}'$. Here we take the center of the boxes $[\mathbf{p}]$ and $[\mathbf{x}]$ denoted by mid($[\mathbf{p}]$), mid($[\mathbf{x}]$).

To verify (6) and (7), one may use an inclusion function $[\mathbf{f}]$ of f. If

$$[\mathbf{f}]\left([\mathbf{x}], \mathbf{p}'\right) \subseteq [\mathbf{z}] \tag{8}$$

then, thanks to (3), (7) is satisfied. Conversely, if

$$[\mathbf{f}]\left([\mathbf{x}], \mathbf{p}'\right) \cap [\mathbf{z}] = \emptyset \tag{9}$$

then, using again (3), (7) cannot be satisfied. A similar reasoning can be made to verify (6).

Due to the over-approximation of the range of a function over an interval provided by inclusion functions, one may sometimes not be able to conclude. Branching over $[\mathbf{x}]$ is then performed by CSC as long as the width of the box $[\mathbf{x}]$ is larger than a given parameter ε_1. Otherwise, to ensure the termination of CSC, one considers that no conclusion can be made for $[\mathbf{p}]_0$.

2) FPS: FPS, see Algorithm 2, searches for parameters that satisfy (5) by calling iteratively CSC. If CSC returns `true`, a valid parameter value and thus a barrier function has been found. When `false` is returned the current box of parameters is deleted. Finally, `unknown` means that bisections in the parameter space are required and CSC has to be called on the resulting boxes.

As for CSC, to ensure the termination of FPS, boxes in the parameter space are bisected as long as their width is larger than ε_2.

D. Implementation

Theorem 1 is defined on the conjunction of three Constraints (4a)-(4c). Three versions of CSC are thus jointly considered: CSCInit, CSCunsafe, and CSCborder dedicated respectively to Constraints (4a), (4b), and (4c).

When CSC is called by FPS, CSCInit, CSCunsafe, and CSCborder are called consecutively. Clearly, CSC returns `false` as soon as one of the CSC functions returns `false`. Two strategies may then be considered. In Strategy 1, all CSC functions are called, except when one of the two first CSC functions returns `false`. In Strategy 2, CSC returns `unknown` as soon as a CSC function returns `unknown`. Strategy 1 may be more efficient at eliminating boxes, while Strategy 2 does not spend unnecessary time in testing CSC functions when the global result is likely to be `unknown`.

E. Using contractors

Due to bisections, the proposed approach can be computationally demanding. This motivates the use of *contractors*,

Algorithm 1: CSC

Input: $[\mathbf{p}]_0, [\mathbf{x}]$
Stack $S := [\mathbf{x}]$;
Flag:=`true`;
while $S \neq \emptyset$ **do**
  pop $[\mathbf{x}]_0$ out of S ;
  if $[f](mid([\mathbf{x}]_0), [\mathbf{p}]_0) \cap [\mathbf{z}] = \emptyset$ **then**
    return(`false`);
  end
  if $[f]([\mathbf{x}]_0, mid([\mathbf{p}]_0)) \subseteq [\mathbf{z}]$ **then**
    continue;
  else
    if *(width*$([\mathbf{x}]_0) \leq \varepsilon_1$*)* **then**
      Flag:=`unknown`;
    else
      $([\mathbf{x}]_1, [\mathbf{x}]_2)$=bisect($[\mathbf{x}]_0$);
      push $[\mathbf{x}]_1$ in S;
      push $[\mathbf{x}]_2$ in S;
    end
  end
end
return(Flag);

Algorithm 2: FPS

Input: $[\mathbf{p}]$,$[\mathbf{x}]$
queue $\mathcal{Q} := [\mathbf{p}]$;
Flag := `true`;
while $\mathcal{Q} \neq \emptyset$ **do**
  dequeue $[\mathbf{p}]_0$ from $\mathcal{Q}$;
  code := CSC($[\mathbf{p}]_0, [\mathbf{x}]$);
  if *code*=`true` **then**
    return(mid($[\mathbf{p}]_0$);
  else
    if *code*=`unknown` **then**
      if *width*$([\mathbf{p}]_0) \leq \varepsilon_2$ **then**
        Flag=`false` ;
      else
        $([\mathbf{p}]_1, [\mathbf{p}]_2) =$ bisect($[\mathbf{p}]_0$);
        enqueue $[\mathbf{p}]_1$ in $\mathcal{Q}$;
        enqueue $[\mathbf{p}]_2$ in $\mathcal{Q}$;
      end
    end
  end
end
if *Flag*=`true` **then**
  return($\emptyset$);
else
  return(`unknown`);
end

see [21] or [9, Chap. 4].

Definition 2: A contractor $\mathcal{C}_c$ associated to the generic constraint

$$c: \quad f(\mathbf{x}) \in [\mathbf{z}] \tag{10}$$

is a function taking a box $[\mathbf{x}]$ as input and returning a box satisfying

$$\mathcal{C}_c([\mathbf{x}]) \subseteq [\mathbf{x}] \tag{11}$$

and

$$f([\mathbf{x}]) \cap [\mathbf{z}] = f(\mathcal{C}_c([\mathbf{x}])) \cap [\mathbf{z}]. \tag{12}$$

For a given box $[\mathbf{x}]$, (11) translates the fact that $\mathcal{C}_c$ eliminates parts of $[\mathbf{x}]$ that are not consistent with (10), but without loosing any consistent solution, as indicated by (12).

Various types of contractors have been proposed in the literature, for example, the contractors by interval constraint propagation, by parallel linearization, the Newton contractor, the Krawczyk contractor, see [9, Chap. 4] for more details.

Contractors can be used by CSC to prove that (5) cannot be satisfied using the following proposition.

Proposition 1: Consider the constraint

$$c: \quad f(\mathbf{x}, \mathbf{p}) \in [\mathbf{z}] \tag{13}$$

and a contractor $\mathcal{C}_c$ for this constraint. For the pair of boxes $([\mathbf{x}], [\mathbf{p}])$, let $\left([\mathbf{x}]', [\mathbf{p}]'\right) = \mathcal{C}_c([\mathbf{x}], [\mathbf{p}])$. If $[\mathbf{x}]' \neq [\mathbf{x}]$, then there is some $\mathbf{x}_0 \in [\mathbf{x}]$ such that $\forall \mathbf{p} \in [\mathbf{p}]$, $f(\mathbf{x}_0, \mathbf{p}) \notin [\mathbf{z}]$. A consequence of Proposition 1 is that if $[\mathbf{x}]' \neq [\mathbf{x}]$, then (13) cannot be satisfied and CSC can directly return `false`.

Contractors can also be used by CSC to prove that (5) is satisfied for some $\mathbf{p}' \in [\mathbf{p}]$. For that purpose, consider the negation $\bar{c}$ of the constraint (10) defined as

$$\bar{c}: \quad f(\mathbf{x}) \notin [\mathbf{z}]. \tag{14}$$

A contractor $\mathcal{C}_{\bar{c}}$ for $\bar{c}$ may then be useful to characterize some $[\tilde{\mathbf{x}}] \subset [\mathbf{x}]$ such that $\forall \mathbf{x} \in [\tilde{\mathbf{x}}]$, $f(\mathbf{x}) \in [\mathbf{z}]$.

Proposition 2: Consider a box $[\mathbf{x}]$ and $\mathcal{C}_{\bar{c}}([\mathbf{x}])$ then,

$$\forall \mathbf{x} \in [\mathbf{x}] \setminus \mathcal{C}_{\bar{c}}([\mathbf{x}]), \text{ one has } f(\mathbf{x}) \in [\mathbf{z}],$$

where $[\mathbf{x}] \setminus \mathcal{C}_{\bar{c}}([\mathbf{x}])$ denotes the box $[\mathbf{x}]$ deprived from $\mathcal{C}_{\bar{c}}([\mathbf{x}])$, which is not necessarily a box.

Consider now the constraint

$$c: \quad f(\mathbf{x}, \mathbf{p}') \in [\mathbf{z}]. \tag{15}$$

for some $\mathbf{p}' \in [\mathbf{p}]$. If $\mathcal{C}_{\bar{c}}([\mathbf{x}]) = \emptyset$ for (15), then, according to Proposition 2, $\forall \mathbf{x} \in [\tilde{\mathbf{x}}]$, $f(\mathbf{x}, \mathbf{p}') \in [\mathbf{z}]$. CSC may then use this contractor as an alternative way to determine whether $f([\mathbf{x}]_0, \text{mid}([\mathbf{p}]_0)) \subseteq [\mathbf{z}]$.

Contractor can also be used in FPS to prune some part of $[\mathbf{p}]$ using the following proposition

Proposition 3: Consider some $\mathbf{x}' \in [\mathbf{x}]$, the constraint

$$c: \quad f(\mathbf{x}', \mathbf{p}) \in [\mathbf{z}], \tag{16}$$

and an associated contractor $\mathcal{C}_c$. For the box $[\mathbf{p}]$, one may evaluate $[\mathbf{p}]' = \mathcal{C}_c([\mathbf{p}])$. Then

$$\forall \mathbf{p} \in [\mathbf{p}] \setminus [\mathbf{p}]' \; f(\mathbf{x}', \mathbf{p}) \notin [\mathbf{z}] \quad . \tag{17}$$

The consequence of Proposition 3 is all the parameters in the set $[\mathbf{p}] \setminus [\mathbf{p}']$ cannot satisfy Constraint (5). They can be safely discarded from the parameter search space.

From an algorithmic point of view, using Propsition 3 can reduce the size of the parameter space in order to reduce the amount of bisections necessary to get a valid result. Propsition 3 may be applied considering many different $\mathbf{x}'$ to further reduce the size of the parameter search space.

V. Examples

A. Experimental conditions

This section presents some results provided by CSC-FPS presented in Section IV. CSC-FPS is implemented in *C++* using the IBEX library, see [21], [22]. The computing time and the number of bisections made by FPS are provided for each example. Table I shows the results obtained without contractor, while Table II shows the benefits of contractors.

Experiments are conducted using an Intel core i7 at $1.70GHz$. One chooses $\varepsilon_1 = 10^{-1}$ for CSC and $\varepsilon_2 = 10^{-5}$ for FPS. In the following examples, $[\mathbf{p}] = [-10, 10]^m$ where m is the number of parameters. TIMEOUT is indicated in the tables after 48 hours of computation. The template are chosen iteratively. First one checks if a linear template can be found. If not the search is done on more complex templates.

Example 1: (P0) Consider the unstable system:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ x_1 x_2 - 0.5 x_2^2 \end{pmatrix}$$

with $[\mathbf{x}_0] = [0, 0.2] \times [-0.2, 0]$, $[\mathbf{x}_\text{u}] = [-2, -1] \times [-2, -1]$ and $[\mathbf{x}_\text{s}] = [-5, 0.25] \times [-5, 5]$. The chosen template is

$$B(\mathbf{x}, \mathbf{p}) = p_1 \ln(p_2 x_1 + p_3) + p_4 x_2 + p_5.$$

Example 2: (sixdim) Here a 6-dimension system is considered. It is taken from [16]. To the best of our knowledge no barrier certificate has been computed for this system.

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{pmatrix} = \begin{pmatrix} -x_1^3 + 4x_2^3 - 6x_3 x_4 \\ -x_1 - x_2 + x_5^3 \\ x_1 x_4 - x_3 + x_4 x_6 \\ x_1 x_3 + x_3 x_6 - x_4^3 \\ -2x_2^3 - x_5 + x_6 \\ -3x_3 x_4 - x_5^3 - x_6 \end{pmatrix}$$

with $[\mathbf{x}_0] = [3, 3.1]^6$, $[\mathbf{x}_\text{u}] = [4, 4.1] \times [4.1, 4.2] \times [4.2, 4.3] \times [4.3, 4.4] \times [4.4, 4.5] \times [4.5, 4.6]$ and $[\mathbf{x}_\text{s}] = [0, 10] \times [0, 10] \times [2, 10] \times [0, 10^4] \times [0, 10] \times [0, 10]$. The chosen template is

$$B(\mathbf{x}, \mathbf{p}) = p_1 x_1^2 + p_2 x_2^4 + p_3 x_3^2 + p_4 x_4^2 + p_5 x_5^4 + p_6 x_6 + p_7.$$

Example 3: (Parillo) [23] For the system:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -x_1 + x_1 x_2 \\ -x_2 \end{pmatrix}$$

with $[\mathbf{x}_0] = [1, 1.25] \times [0.5, 0.75]$, $[\mathbf{x}_\text{u}] = [0.75, 1] \times [0.05, 0.2]$ and $[\mathbf{x}_\text{s}] = [-100, 100] \times [-100, 100]$, no polynomial template has been found. The template used is

$$B(\mathbf{x}, \mathbf{p}) = \ln(p_1 x_1) - \ln(x_2) + p_2 x_2 + p_3.$$

Example 4: (P3) [6] For the disturbed system

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix} = \begin{pmatrix}x_2\\ -x_1 + \frac{d}{3}x_1^3 - x_2\end{pmatrix}$$

with $[\mathbf{x}_0] = [1,2]\times[-0.5,0.5]$, $[\mathbf{x}_\mathrm{u}] = [-1.4,-0.6]\times[-1.4,-0.6]$ and $[\mathbf{x}_\mathrm{s}] = [-100,100]\times[-10,10]$ and $d \in [0.9,1.1]$, the template

$$B(\mathbf{x},\mathbf{p}) = p_1x_1^2 + p_2x_2^2 + p_3x_1x_2 + p_4x_1 + p_5x_2 + p_6.$$

is considered. The disturbance here is taken over a box $[\mathbf{d}]$ and it is considered as time independent.

In this case, Constraint (4c) has to be valid $\forall \mathbf{d} \in [\mathbf{d}]$. It is rewritten as follows

$$\begin{aligned}&\exists \mathbf{p} \in [\mathbf{P}]\,,\ \forall \mathbf{x} \in [\mathbf{x}_\mathrm{s}],\\ &B(\mathbf{x},\mathbf{p}) \neq 0 \vee \left(\forall \mathbf{d} \in [\mathbf{D}]\,, \left\langle \frac{\partial B}{\partial \mathbf{x}}(\mathbf{x}), f(\mathbf{x},d)\right\rangle < 0\right).\end{aligned} \tag{18}$$

Inclusion functions can be used to evaluate the range of the perturbed f.

Example 5: (E5) [24] Consider the unstable system:

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix} = \begin{pmatrix}x_2\\ \frac{\sqrt{(1+x_2^2)}}{25-x1}\end{pmatrix}$$

with $[\mathbf{x}_0] = [0.75,0.8]\times[-0.1,-0.05]$, $[\mathbf{x}_\mathrm{u}] = [0.5,0.55]\times[-0.15,-0.1]$ and $[\mathbf{x}_\mathrm{s}] = [-100,100]\times[-1,1]$. The considered template

$$B(\mathbf{x},\mathbf{p}) = p_1x_1^2 + p_2x_2^2 + p_3x_1x_2 + p_4x_1 + p_5x_2 + p_6.$$

Example 6: (Lorenz) [25] We consider the Lorenz system with a limit cycle. The initial region is taken inside the limit cycle and the unsafe outside of it.

$$\begin{pmatrix}\dot{x_1}\\ \dot{x_2}\\ \dot{x_3}\end{pmatrix} = \begin{pmatrix}10(x_2-x_1)\\ x_1(28-x_3)-x_2\\ x_1x_2 - \frac{8}{3}x_3\end{pmatrix}$$

with $[\mathbf{x}_0] = [-15,-14]\times[-15,-14]\times[10,15]$, $[\mathbf{x}_\mathrm{u}] = [-17,-16]\times[-15,-14]\times[0,5]$ and $[\mathbf{x}_\mathrm{s}] = [-20,20]\times[-20,-10]\times[-20,20]$. We consider the template

$$B(\mathbf{x},\mathbf{p}) = p_1x_1 + p_2x_2 + p_3x_3 + p_4.$$

Example 7: (Saturation) [26]

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix} = \begin{pmatrix}x_2\\ -\frac{x_1+x_2}{\sqrt{1+(x_1+x_2)^2}}\end{pmatrix} \tag{19}$$

with $[\mathbf{x}_0] = [-0.5,0.5]\times[-0.5,0.5]$, $[\mathbf{x}_\mathrm{u}] = [2.5,3]\times[-0.5,0]$, and $[\mathbf{x}_\mathrm{s}] = [-10^3,10^3]\times[-100,100]$. The quadratic template

$$B(\mathbf{x},\mathbf{p}) = p_1x_1^2 + p_2x_2^2 + p_3x_1x_2 + p_4x_1 + p_5x_2 + p_6.$$

is considered. Figure 1 illustrates the barrier function computed with the proposed approach.

TABLE I

RESULTS WITHOUT CONTRACTORS

	Time		Bisections	
Example	Strategy 1	Strategy 2	Strategy 1	Strategy 2
P0	$2603s$	$544s$	298462	389026
sixdim	TIMEOUT	TIMEOUT	/	/
parillo	TIMEOUT	TIMEOUT	/	/
Saturation	$6830s$	$1711s$	9290	10931
P3	$1224s$	$326s$	12111	14456
E5	TIMEOUT	TIMEOUT	/	/
Lorenz	$766s$	$120s$	191	195

TABLE II

RESULTS USING CONTRACTORS

	Time		Bisections	
Example	Strategy 1	Strategy 2	Strategy 1	Strategy 2
P0	$7s$	$3s$	1527	1541
sixdim	$50s$	$1s$	3897	3897
parillo	$9s$	$2s$	317	317
Saturation	$1s$	$1s$	6	6
P3	$1036s$	$212s$	17646	19004
E5	$247s$	$51s$	42304	42304
Lorenz	$343s$	$77s$	188	188

B. Discussion

Tables I and II show that CSC-FPS supplemented with contractors is much more efficient in term of computing time and number of necessary bisections. The reason for this difference is that the contractor found in FPS reduces the size of the parameter space by eliminating invalid parameters, without calling CSC. Moreover, the contractor found in CSC speeds up the validation and invalidation process. For some examples (E5 or Parillo), the version without contractor fails in finding a valid solution. This is mainly due to the non-linearities of the system or of the barrier function.

Comparing the two strategies for managing the results of the CSC functions, Strategy 1, which evaluates all CSC functions, except if one of them returns `false` takes more time due to the cost of every CSC check. As expected, it reduces the number of bisections compared to Strategy 2.

C. RSolver

As mentioned in Section IV-C, other tools exist to solve quantified constraints. We report in this section the results produced by RSolver [20] on our particular problem. As RSolver uses linear programming to compute solutions of quantified constraint problem, we are limited on the form of parametric barrier functions which must be linear in the parameters. Moreover, RSolver relies on a language to describe quantified constraints which does not handle the division operation so, we are unable to compute barrier functions for problems E5 and Saturation. For the remaining examples, RSolver usually terminates without producing satisfying answers even when the polynomial parametric barrier functions are considered. If we use linear parametric barrier functions, RSolver is able to compute very efficiently a result only on two examples: Parillo and Lorenz: no bisections are required and the solution is obtained in few milliseconds.

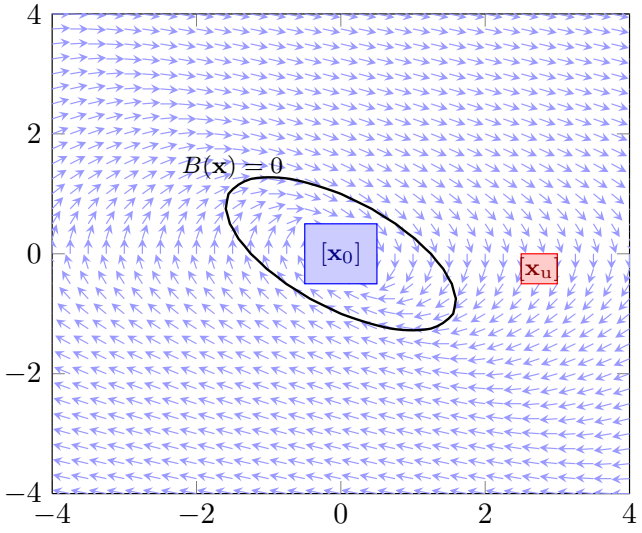

Fig. 1. Barrier function for the example Saturation

VI. Conclusion

This paper presents a new method to find parametric barriers and provide barrier certificates for nonlinear dynamical systems. It is based on the search for the parameters of a barrier function with a given template using interval analysis. The main benefit of the barrier certificate technique is that there is no need for an explicit computation of the reachable state space which is difficult for nonlinear dynamic. The proposed technique has no restriction regarding the dynamics nor the template of the barrier function. It is able to find barrier certificates for a large class of dynamical systems.

Future work, will focus on better strategies to choose the vector parameter tested in CSCs procedure for the validation.

Extensions to hybrid systems will also be investigated using an approach similar to that considered in [6]. A set of quantified constraints may be defined for each location of the hybrid automaton and one may search for templates satisfying the constraints associated to the transitions.

References

[1] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *Computer Aided Verification*, pp. 379–395, Springer, 2011.

[2] S. Sankaranarayanan, H. B. Sipma, and Z. Manna, "Constructing invariants for hybrid systems," in *Hybrid Systems: Computation and Control*, pp. 539–554, Springer, 2004.

[3] E. Asarin, O. Bournez, T. Dang, and O. Maler, "Approximate reachability analysis of piecewise-linear dynamical systems," in *Hybrid Systems: Computation and Control*, pp. 20–31, Springer, 2000.

[4] A. Tiwari, "Approximate reachability for linear systems," in *Hybrid Systems: Computation and Control*, pp. 514–525, Springer, 2003.

[5] A. Chutinan and B. H. Krogh, "Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations," in *Hybrid Systems: Computation and Control*, pp. 76–90, Springer, 1999.

[6] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *Hybrid Systems: Computation and Control*, pp. 477–492, Springer, 2004.

[7] L. Jaulin and É. Walter, "Guaranteed tuning, with application to robust control and motion planning," *Automatica*, vol. 32, no. 8, pp. 1217–1221, 1996.

[8] P. Van Hentenryck, D. McAllester, and D. Kapur, "Solving polynomial systems using a branch and prune approach," *SIAM Journal on Numerical Analysis*, vol. 34, no. 2, pp. 797–827, 1997.

[9] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*. Springer, 1 ed., 2001.

[10] H. Guéguen and J. Zaytoon, "On the formal verification of hybrid systems," *Control Engineering Practice*, vol. 12, no. 10, pp. 1253 – 1267, 2004.

[11] A. Girard, C. Le Guernic, and O. Maler, "Efficient computation of reachable sets of linear time-invariant systems with inputs," in *Hybrid Systems: Computation and Control*, pp. 257–271, Springer, 2006.

[12] A. Chutinan and B. H. Krogh, "Computational techniques for hybrid system verification," *Automatic Control, IEEE Transactions on*, vol. 48, no. 1, pp. 64–75, 2003.

[13] X. Chen, E. Abrahám, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," in *Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd*, pp. 183–192, IEEE, 2012.

[14] R. Genesio, M. Tartaglia, and A. Vicino, "On the estimation of asymptotic stability regions: State of the art and new proposals," *Automatic Control, IEEE Transactions on*, vol. 30, no. 8, pp. 747–755, 1985.

[15] P. A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," *Mathematical programming*, vol. 96, no. 2, pp. 293–320, 2003.

[16] S. Ratschan and Z. She, "Providing a basin of attraction to a target region by computation of lyapunov-like functions," in *Computational Cybernetics, 2006. ICCC 2006. IEEE International Conference on*, pp. 1–5, IEEE, 2006.

[17] S. Gulwani and A. Tiwari, "Constraint-based approach for analysis of hybrid systems," in *Computer Aided Verification*, pp. 190–203, Springer, 2008.

[18] R. E. Moore and R. Moore, *Methods and applications of interval analysis*, vol. 2. SIAM, 1979.

[19] S. Ratschan, "Efficient solving of quantified inequality constraints over the real numbers," *ACM Transactions on Computational Logic (TOCL)*, vol. 7, no. 4, pp. 723–748, 2006.

[20] "RSolver sourceforge." `http://rsolver.sourceforge.net/`. Accessed: 2014-09-16.

[21] G. Chabert and L. Jaulin, "Contractor programming," *Artificial Intelligence*, vol. 173, no. 11, pp. 1079–1100, 2009.

[22] I. Araya, G. Trombettoni, B. Neveu, and G. Chabert, "Upper bounding in inner regions for global optimization under inequality constraints," *Journal of Global Optimization*, pp. 1–20, 2012.

[23] A. A. Ahmadi, M. Krstic, and P. A. Parrilo, "A globally asymptotically stable polynomial vector field with no polynomial lyapunov function.," in *CDC-ECE*, pp. 7579–7580, 2011.

[24] W. H. Enright and J. D. Pryce, "Two fortran packages for assessing initial value methods," *ACM Transactions on Mathematical Software (TOMS)*, vol. 13, no. 1, pp. 1–27, 1987.

[25] A. Vaněček and S. Čelikovskỳ, *Control systems: from linear analysis to synthesis of chaos*. Prentice Hall International (UK) Ltd., 1996.

[26] A. Papachristodoulou and S. Prajna, "Analysis of non-polynomial systems using the sum of squares decomposition," in *Positive Polynomials in Control*, pp. 23–43, Springer, 2005.