Construction of parametric barrier functions for dynamical systems using interval analysis

Adel Djaballah, Alexandre Chapoutot, Michel Kieffer, Olivier Bouissou

Construction of Parametric Barrier Functions for Dynamical Systems using Interval Analysis ⋆

A. Djaballah [a], A. Chapoutot [a], M. Kieffer [b], O. Bouissou [c]

[a] *U2IS, ENSTA ParisTech, Université Paris-Saclay, 828 bd des Maréchaux, 91762 Palaiseau Cedex*

[b] *L2S, CNRS, Supélec, Univ. Paris-Sud 91192 Gif-sur-Yvette Cedex and Institut Universitaire de France 75005 Paris,*

[c] *CEA Saclay Nano-INNOV Institut CARNOT, 91191 Gif-sur-Yvette Cedex*

Abstract

Recently, barrier certificates have been introduced to prove the safety of continuous or hybrid dynamical systems. A barrier certificate needs to exhibit some barrier function, which partitions the state space in two subsets: the safe subset in which the state can be proved to remain and the complementary subset containing some unsafe region. This approach does not require any reachability analysis, but needs the computation of a valid barrier function, which is difficult when considering general nonlinear systems and barriers. This paper presents a new approach for the construction of barrier functions for nonlinear dynamical systems. The proposed technique searches for the parameters of a parametric barrier function using interval analysis. Complex dynamics with bounded perturbations can be considered without needing any relaxation of the constraints to be satisfied by the barrier function.

Key words: Formal verification, Dynamic systems, Intervals, Constraint satisfaction problem

1 Introduction

Formal verification aims at proving that a certain behavior or property is fulfilled by a system. Verifying, *e.g.*, the safety property for a system consists in ensuring that it will never reach a dangerous or an unwanted configuration. Safety verification is usually translated into a reachability analysis problem [5,11,14,38,39]. Starting from an initial region, a system must not reach some unsafe region. Different methods have been considered to address this problem. One may explicitly compute the reachable region and determine whether the system reaches the unsafe region [18]. An alternative idea is to compute an invariant for the system, *i.e.*, a region in which the system is guaranteed to stay [11]. This paper considers a class of invariants determined by *barrier functions*.

A barrier function [29,30] partitions the state space and isolates an unsafe region from the part of the state space containing the initial region. In [30] polynomial barriers are considered for polynomial systems and semi-definite programming is used to find satisfying barrier functions. Our aim is to extend the class of considered problems to non-polynomial systems and to non-polynomial barriers. This paper focuses on continuous-time systems.

The design of a barrier function is formulated as a quantified constraints satisfaction problem (QCSP) [8,32]. Interval analysis is then used to find the parameters of a barrier function such that the QCSP is satisfied. More specifically,

⋆ This research was partially 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), by the ANR INS Project CAFEIN (ANR-12-INSE-0007), and by the iCODE Institute, research project of the IDEX Paris-Saclay.

Email addresses: adel.djaballah@ensta-paristech.fr (A. Djaballah), alexandre.chapoutot@ensta-paristech.fr (A. Chapoutot), michel.kieffer@lss.supelec.fr (M. Kieffer), olivier.bouissou@cea.fr (O. Bouissou).

the algorithm presented in [21] for robust controller design is adapted and supplemented with some of the pruning schemes found in [9] to solve the QCSP associated to the barrier function design.

The paper is organized as follows. Section 2 introduces some related work. Section 3 defines the notion of barrier functions and formulates the design of barrier functions as a QCSP. Section 4 presents the framework developed to solve the QCSP. Design examples are presented in Section 5. Section 6 concludes the work.

In what follows 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. We denote by $\mathbb{IR}$ the set of closed intervals over $\mathbb{R}$, the set of real numbers. Data structures or sets $\mathcal{S}$ are in upper-case calligraphic. The derivative of a function x with respect to time t is denoted by $\dot{x}$.

2 Related work

The verification of the safety properties for dynamical systems has been an active field of research in the last years. This related work review focuses on methods involving the computation of *invariants* for dynamical systems. Alternative methods based on the computation of reachable sets are described in [10,14] and in the references therein.

An invariant is a part of the state space in which the state of a dynamical system can be proved to remain. Invariants are very useful to prove the safety of dynamical systems. If an invariant does not contain unsafe regions, then the dynamical system is safe. Methods to characterize invariants have been intensively studied for linear and polynomial dynamics but much work has still to be done for non-linear dynamical systems.

Coming from the community interested in *hybrid systems*, a set of methods has been defined to compute invariants for various classes of systems, for example for linear or affine systems [39] or for polynomial systems [18, 22, 36, 41]. These methods introduce a candidate parametric function, which parameter vector has to be adjusted to define an invariant of the considered dynamical system. Various techniques are then employed to determine satisfying parameter vectors. For example, in [36], theory of ideal over polynomials and Gröbner bases are used to define constraints to be satisfied by the parameter vector to be found. These constraints are then solved numerically using tools such as those introduced in [12]. In [18], quantified polynomial constraints are introduced for the design of parameters. Then, satisfying parameter vectors are found using Farkas' Lemma and solvers from sat-modulo theory [6]. Sum-of-Squares (SoS) polynomials are used in [22]. The design involves various system simulations and selection of candidate parameter vectors using linear programming. A final validation of the selected parameter vectors is then performed with Mathematica and using interval analysis with dReal [15]. Note that our algorithm presented in Section 4.3 could aso be used as validation method for the approach presented in [22]. Bilinear SoS programming is considered in [41].

An alternative way to find such an invariant is by considering tools such as Lyapunov functions to prove the stability properties of dynamical systems [16]. For example, [25] considers parametric functions to find a Lyapunov function for a system with polynomial dynamics formed by SoS polynomials and employs semidefinite programming (SdP) for the parameter synthesis. In [34] Lyapunov functions are designed via a branch and relax approach and linear programming to solve the induced constraints. In [17] Darboux polynomials are used to design specific forms of Lyapunov functions involving rational functions, logarithmic and exponential terms. Similar invariants have been also considered in [35].

Safety property verification may also be directly introduced in the design phase, instead of being verified *a posteriori*, as done in the previous approaches. In [26, 27], theorem-proving approaches are employed using symbolic-numeric techniques to synthesize invariants for differential (continuous and hybrid) systems. In particular, quantifier elimination techniques are intensively used and more recently a combination with the approach presented in [22] has been defined in [4]. Alternatively, techniques searching for *barrier certificates* aim at determining some parametric function, called *barrier*, defining an hyper-surface in the state-space which is never crossed by the dynamics of the system, see [13, 28, 30, 31, 37]. A parameter vector of this barrier has to be found such that the barrier separates the region in which the initial state belongs from the unsafe region. In [28, 30, 31], polynomial dynamics and barrier function are considered and parameters are designed with SdP, which requires some relaxation to obtain a convex design problem. In [13], two candidate functions are combined to define more sophisticated barriers, which parameters are again found via SdP. In [37], linear matrix inequalities and SoS are used to generate the barrier functions for hybrid dynamical systems with polynomial dynamics.

Our work follows this approach for non-linear and possibly non-polynomial continuous-time dynamical systems with bounded perturbations and uses interval analysis for the barrier parameter vector search phase.

3 Formulation

This section recalls the safety characterization introduced in [30] for continuous-time systems using barrier functions.

3.1 Safety for continuous-time systems

Consider the autonomous continuous-time perturbed dynamical system

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

where $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state vector and $\mathbf{d} \in \mathcal{D}$ is a constant and bounded disturbance. The set of possible initial states at $t = 0$ is denoted $\mathcal{X}_0 \subset \mathcal{X}$. There is some unsafe subset $\mathcal{X}_u \subseteq \mathcal{X}$ that shall not be reached by the system, given any $\mathbf{x}_0 \in \mathcal{X}_0$ at time $t = 0$ and any $\mathbf{d} \in \mathcal{D}$. We assume that classical hypotheses (see, *e.g.*, [7]) on f are satisfied so that (1) has a unique solution $\mathbf{x}(t, \mathbf{x}_0, \mathbf{d}) \in \mathcal{X}$ for any given initial value $\mathbf{x}_0 \in \mathcal{X}_0$ at time $t = 0$ and any $\mathbf{d} \in \mathcal{D}$.

Definition 1 *The dynamical system* (1) *is* safe *if* $\forall \mathbf{x}_0 \in \mathcal{X}_0$, $\forall \mathbf{d} \in \mathcal{D}$ *and* $\forall t \geqslant 0$, $\mathbf{x}(t, \mathbf{x}_0, \mathbf{d}) \notin \mathcal{X}_u$.

3.2 Barrier certificates

A way to prove that (1) is safe is by the barrier certificate approach introduced in [30]. A barrier is a differentiable function $B : \mathcal{X} \to \mathbb{R}$ that partitions the state space $\mathcal{X}$ into $\mathcal{X}_-$ where $B(\mathbf{x}) \leqslant 0$ 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 $\forall \mathbf{x}_0 \in \mathcal{X}_0$, $\forall \mathbf{d} \in \mathcal{D}$, $\forall t \geqslant 0$, $B(\mathbf{x}(t, \mathbf{x}_0, \mathbf{d})) \leqslant 0$.

Proving that $B(\mathbf{x}(t, \mathbf{x}_0, \mathbf{d})) \leqslant 0$ requires an evaluation of the solution of (1) for all $\mathbf{x}_0 \in \mathcal{X}_0$ and $\mathbf{d} \in \mathcal{D}$. Alternatively, [30] provides some sufficient conditions a barrier function has to satisfy to prove the safety of a dynamical system, see Theorem 1.

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

$$\forall \mathbf{x} \in \mathcal{X}_0, \quad B(\mathbf{x}) \leqslant 0, \tag{2}$$

$$\forall \mathbf{x} \in \mathcal{X}_u, \quad B(\mathbf{x}) > 0, \tag{3}$$

$\forall \mathbf{x} \in \mathcal{X}$, $\forall \mathbf{d} \in \mathcal{D}$,

$$B(\mathbf{x}) = 0 \implies \left\langle \frac{\partial B(\mathbf{x})}{\partial \mathbf{x}}, f(\mathbf{x}, \mathbf{d}) \right\rangle < 0, \tag{4}$$

then (1) *is safe.*

In (4) $\langle ., . \rangle$ stands for the dot product in $\mathbb{R}^n$. In Theorem 1, (2) and (3) ensure that $\mathcal{X}_0 \subseteq \mathcal{X}_-$, and $\mathcal{X}_u \subseteq \mathcal{X}_+$, while (4) states that if $\mathbf{x}$ 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 $\mathcal{X}_-$ whatever the value of the disturbance $\mathbf{d}$.

3.3 Parametric barrier functions

The search for a barrier B is challenging since it is over a functional space. As in [30], this paper considers barriers belonging to a family of parametric functions (or templates) $B(\mathbf{x}, \mathbf{p})$ depending on a parameter vector $\mathbf{p} \in \mathcal{P} \subseteq \mathbb{R}^m$. Then one may search for some parameter value $\mathbf{p}$ such that $B(\mathbf{x}, \mathbf{p})$ satisfies (2)-(4).

If there is no $\mathbf{p} \in \mathcal{P}$ such that $B(\mathbf{x}, \mathbf{p})$ satisfies (2)-(4), this does not mean that the system is not safe: other structures of functions $B(\mathbf{x}, \mathbf{p})$ could provide a barrier certificate.

4 Characterization using interval analysis

This section presents an approach to find a barrier function that fulfills the constraints of Theorem 1. These constraints are first reformulated to cast the design of a barrier function as a quantified constraint satisfaction problem (QCSP) [32].

4.1 Constraint satisfaction problem

Assume that there exist some functions $g_0 : \mathcal{X} \to \mathbb{R}$ and $g_u : \mathcal{X} \to \mathbb{R}$, such that

$$\mathcal{X}_0 = \{\mathbf{x} \in \mathcal{X} \,|\, g_0(\mathbf{x}) \leqslant 0\} \tag{5}$$

and

$$\mathcal{X}_u = \{\mathbf{x} \in \mathcal{X} \,|\, g_u(\mathbf{x}) \leqslant 0\}. \tag{6}$$

Theorem 1 may be reformulated as follows.

Proposition 2 *If* $\exists \mathbf{p} \in \mathcal{P}$ *such that* $\forall \mathbf{x} \in \mathcal{X}$, $\forall \mathbf{d} \in \mathcal{D}$

$$\begin{aligned} \xi\left(\mathbf{x}, \mathbf{p}, \mathbf{d}\right) = &\left(g_0(\mathbf{x}) > 0 \vee B(\mathbf{x}, \mathbf{p}) \leqslant 0\right) && (7)\\ &\wedge \left(g_u(\mathbf{x}) > 0 \vee B(\mathbf{x}, \mathbf{p}) > 0\right) && (8)\\ &\wedge \left(B(\mathbf{x}, \mathbf{p}) \neq 0 \vee \left\langle \frac{\partial B}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{p}), f(\mathbf{x}, \mathbf{d}) \right\rangle < 0\right) && (9) \end{aligned}$$

holds true, then the dynamical system (1) *is safe.*

PROOF. The first component of $\xi\left(\mathbf{x}, \mathbf{p}, \mathbf{d}\right)$,

$$\xi_0\left(\mathbf{x}, \mathbf{p}\right) = \left(g_0(\mathbf{x}) > 0 \vee B(\mathbf{x}, \mathbf{p}) \leqslant 0\right) \tag{10}$$

may be rewritten as

$$\xi_0\left(\mathbf{x}, \mathbf{p}\right) = \left(g_0(\mathbf{x}) \leqslant 0 \implies B(\mathbf{x}, \mathbf{p}) \leqslant 0\right),$$

see, *e.g.*, [19]. If $\xi_0\left(\mathbf{x}, \mathbf{p}\right)$ holds true for some $\mathbf{p} \in \mathcal{P}$ and $\mathbf{x} \in \mathcal{X}$, then one has either $\mathbf{x} \in \mathcal{X}_0$ and $B(\mathbf{x}, \mathbf{p}) \leqslant 0$, or $\mathbf{x} \notin \mathcal{X}_0$. In both cases, (2) is satisfied. Similarly, the second component of $\xi\left(\mathbf{x}, \mathbf{p}, \mathbf{d}\right)$ representing (3) may be rewritten as

$$\xi_{\mathrm{u}}(\mathbf{x}, \mathbf{p}) = \left(g_{\mathrm{u}}(\mathbf{x}) \leqslant 0 \implies B(\mathbf{x}, \mathbf{p}) > 0\right). \tag{11}$$

If $\xi_{\mathrm{u}}\left(\mathbf{x}, \mathbf{p}\right)$ holds true for some $\mathbf{p} \in \mathcal{P}$ and $\mathbf{x} \in \mathcal{X}$, then one has either $\mathbf{x} \in \mathcal{X}_{\mathrm{u}}$ and $B(\mathbf{x}, \mathbf{p}) \leqslant 0$, or $\mathbf{x} \notin \mathcal{X}_{\mathrm{u}}$. In both cases, (3) is satisfied. Now, one may rewrite the last component of $t\left(\mathbf{x}, \mathbf{p}, \mathbf{d}\right)$,

$$\xi_b(\mathbf{x}, \mathbf{p}, \mathbf{d}) = \left(B(\mathbf{x}, \mathbf{p}) \neq 0 \vee \left\langle \frac{\partial B}{\partial x}(\mathbf{x}, \mathbf{p}), f(\mathbf{x}, \mathbf{d}) \right\rangle < 0\right) \tag{12}$$

as

$$\xi_b(\mathbf{x}, \mathbf{p}, \mathbf{d}) = \left(B(\mathbf{x}, \mathbf{p}) = 0 \implies \left\langle \frac{\partial B}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{p}), f(\mathbf{x}, \mathbf{d}) \right\rangle < 0\right), \tag{13}$$

which corresponds to (4). If $\exists \mathbf{p} \in \mathcal{P}$ such that $\forall \mathbf{x} \in \mathcal{X}$, $\forall \mathbf{d} \in \mathcal{D}$, $\xi\left(\mathbf{x}, \mathbf{p}, \mathbf{d}\right)$ holds true, then the conditions of Theorem 1 are satisfied and (1) is safe. □

In [30], (9) is relaxed into

$$\xi_{\mathrm{b}}\left(\mathbf{x}, \mathbf{p}, \mathbf{d}\right) = \left(\left\langle \frac{\partial B}{\partial \mathbf{x}}(\mathbf{x}, \mathbf{p}), f(\mathbf{x}, \mathbf{d}) \right\rangle < 0\right), \tag{14}$$

with the consequence of possible elimination of barrier functions that would satisfy (9) for some $\mathbf{p}$ but not (14). Our aim in this paper is to design barrier functions without resorting to this relaxation by considering methods from interval analysis [20] which allow to consider strongly nonlinear dynamics and barrier functions.

4.2 Solving the constraints

To find a valid barrier function one needs to find some $\mathbf{p} \in \mathcal{P}$ such that $B(\mathbf{x}, \mathbf{p})$ satisfies the conditions of Proposition 2. For that purpose, the *Computable Sufficient Conditions-Feasible Point Searcher* (CSC-FPS) algorithm [21] is adapted.

In what follows, we assume that $\mathcal{X}$, $\mathcal{D}$, and $\mathcal{P}$ are boxes, *i.e.*, $\mathcal{X} = [\mathbf{x}]$, $\mathcal{D} = [\mathbf{d}]$, and $\mathcal{P} = [\mathbf{p}]$. CSC-FPS may also be applied when $\mathcal{X}$, $\mathcal{D}$, and $\mathcal{P}$ consist of a union of non-overlapping boxes.

Consider some function $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^k$ and some box $[\mathbf{z}] \in \mathbb{IR}^k$. CSC-FPS is designed to determine whether

$$\exists \mathbf{p} \in [\mathbf{p}]\,,\ \forall \mathbf{x} \in [\mathbf{x}]\,,\ g\,(\mathbf{x}, \mathbf{p}) \in [\mathbf{z}] \tag{15}$$

and to provide some satisfying $\mathbf{p}$. We extend CSC-FPS to handle conjunctions and disjunctions of constraints and supplement it with efficient pruning techniques involving contractors provided by interval analysis [20].

FPS branches over the parameter search box $[\mathbf{p}]$. Branching is performed based on the results provided by CSC. For a given box $[\mathbf{p}]_0 \subseteq [\mathbf{p}]$, CSC returns `true` when it manages to prove that (15) is satisfied for some $\mathbf{p} \in [\mathbf{p}]_0$. CSC returns `false` when it is able to show that there is no $\mathbf{p} \in [\mathbf{p}]_0$ satisfying (15). CSC returns `unknown` in the other cases.

In Proposition 2, $\xi\,(\mathbf{x}, \mathbf{p}, \mathbf{d})$ consists of the conjunction of three terms of the form

$$\tau\,(\mathbf{x}, \mathbf{p}, \mathbf{d}) = (u(\mathbf{x}, \mathbf{p}) \in \mathcal{A}) \vee (v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \in \mathcal{B})\,. \tag{16}$$

For $\xi_0(\mathbf{x}, \mathbf{p})$, defined in (7),

$$\mathcal{A} =]0, +\infty[\text{ and } \mathcal{B} =]-\infty, 0]\,; \tag{17}$$

for $\xi_u(\mathbf{x}, \mathbf{p})$, defined in (8),

$$\mathcal{A} =]0, +\infty[\text{ and } \mathcal{B} =]0, +\infty[\,; \tag{18}$$

for $\xi_b(\mathbf{x}, \mathbf{p}, \mathbf{d})$, defined in (9),

$$\mathcal{A} =]-\infty, 0[\cup]0, +\infty[\text{ and } \mathcal{B} =]-\infty, 0[\,. \tag{19}$$

To illustrate the main ideas of CSC-FPS combined with contractors, one focuses on the generic QCSP

$$\exists \mathbf{p} \in [\mathbf{p}]\,, \forall \mathbf{x} \in [\mathbf{x}]\,, \forall \mathbf{d} \in [\mathbf{d}]\,,\ \tau\,(\mathbf{x}, \mathbf{p}, \mathbf{d}) \text{ holds true.} \tag{20}$$

Finding a solution for such QCSP involves three steps: validation, reduction of the parameter and state spaces, and bisection.

4.2.1 Validation

In the validation step, one tries to prove that some vector $\mathbf{p} \in [\mathbf{p}]$ is such that $\forall \mathbf{x} \in [\mathbf{x}]$, $\forall \mathbf{d} \in [\mathbf{d}]$, $\tau\,(\mathbf{x}, \mathbf{p}, \mathbf{d})$ holds true. By definition of $\tau\,(\mathbf{x}, \mathbf{p}, \mathbf{d})$, one has to prove that

$$\exists \mathbf{p} \in [\mathbf{p}]\,, \forall \mathbf{x} \in [\mathbf{x}]\,, \forall \mathbf{d} \in [\mathbf{d}]\,,\quad (u(\mathbf{x}, \mathbf{p}) \in \mathcal{A}) \vee (v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \in \mathcal{B})\,. \tag{21}$$

For that purpose, one chooses some arbitrary $\mathbf{p} \in [\mathbf{p}]$ and evaluates the set of values $u\,([\mathbf{x}]\,, \mathbf{p}) = \{u\,(\mathbf{x}, \mathbf{p}) \mid \mathbf{x} \in [\mathbf{x}]\}$ and $v\,([\mathbf{x}]\,, \mathbf{p}, [\mathbf{d}]) = \{v\,(\mathbf{x}, \mathbf{p}, \mathbf{d}) \mid \mathbf{x} \in [\mathbf{x}], \mathbf{d} \in [\mathbf{d}]\}$ taken by $u(\mathbf{x}, \mathbf{p})$ and $v(\mathbf{x}, \mathbf{p}, \mathbf{d})$ for all $\mathbf{x} \in [\mathbf{x}]$ and for all $\mathbf{d} \in [\mathbf{d}]$. Outer-approximations of $u\,([\mathbf{x}]\,, \mathbf{p})$ and $v\,([\mathbf{x}]\,, \mathbf{p}, [\mathbf{d}])$ are easily obtained using *inclusion functions* provided by interval analysis.

Definition 3 *An inclusion function* $[f] : \mathbb{IR}^n \to \mathbb{IR}^k$ *for a function* $f : \mathbb{R}^n \to \mathbb{R}^k$ *satisfies for all* $[\mathbf{x}] \in \mathbb{IR}^n$,

$$f([\mathbf{x}]) = \{f(\mathbf{x}) \mid \mathbf{x} \in [\mathbf{x}]\} \subseteq [f]([\mathbf{x}]). \tag{22}$$

The *natural* inclusion function is the simplest to obtain: all occurrences of the real variables are replaced by their interval counterpart and all arithmetic operations are evaluated using interval arithmetic. More sophisticated inclusion functions such as the centered form, or the Taylor inclusion function may also be used, see [20].

Using inclusion functions, one may evaluate whether

$$[u]([\mathbf{x}], \mathbf{p}) \subseteq \mathcal{A} \quad \text{or} \quad [v]([\mathbf{x}], \mathbf{p}, [\mathbf{d}]) \subseteq \mathcal{B} \quad \text{holds true}$$

for the various $\mathcal{A}$ and $\mathcal{B}$ defined in (17), (18), and (19).

Different choices can be considered for $\mathbf{p}$: one can take a random point in $[\mathbf{p}]$, the middle, or one of the edges of $[\mathbf{p}]$. Here, only the middle of $[\mathbf{p}]$ is considered.

4.2.2 Reduction of the parameter and state spaces

To facilitate the search for $\mathbf{p} \in [\mathbf{p}]$ one may previously eliminate parts of $[\mathbf{p}]$ which may be proved not to contain any $\mathbf{p}$ satisfying (21). The elimination process can be done by evaluation or by using *contractors* [9].

4.2.2.1 Evaluation Considering the negation of (21), one deduces that a box $[\mathbf{p}]$ can be eliminated, *i.e.*, shown not to contain any $\mathbf{p}$ satisfying (16), if

$$\forall \mathbf{p} \in [\mathbf{p}], \exists \mathbf{x} \in [\mathbf{x}], \exists \mathbf{d} \in [\mathbf{d}], \quad u(\mathbf{x}, \mathbf{p}) \notin \mathcal{A} \wedge v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \notin \mathcal{B}, \tag{23}$$

If, using again inclusion functions, one is able to prove that

$$\exists \mathbf{x} \in [\mathbf{x}], \exists \mathbf{d} \in [\mathbf{d}], \quad [u](\mathbf{x}, [\mathbf{p}]) \cap \mathcal{A} = \emptyset \wedge [v](\mathbf{x}, [\mathbf{p}], \mathbf{d}) \cap \mathcal{B} = \emptyset \tag{24}$$

then (23) holds true and one can thus eliminate $[\mathbf{p}]$.

In general, $[u](\mathbf{x}, [\mathbf{p}])$ and $[v](\mathbf{x}, [\mathbf{p}], \mathbf{d})$ are not degenerated intervals, *i.e.*, are not reduced to a real value. When $\mathcal{A}$ and $\mathcal{B}$ are half-lines, as is the case for (7) and (8), one may be able to show that (24) holds true. Nevertheless, this will be impossible to show for (9) since in this case $\mathcal{A} = \mathbb{R} \setminus \{0\}$ and $[u](\mathbf{x}, [\mathbf{p}])$ is not a degenerated interval. To show that $[u](\mathbf{x}, [\mathbf{p}]) \cap \mathcal{A} = \emptyset$, one needs to have $[u](\mathbf{x}, [\mathbf{p}]) = [0, 0]$, which is impossible in general.

4.2.2.2 Contractors Consider some function $g : \mathbb{R}^n \to \mathbb{R}^k$ and some set $\mathcal{Z} \subset \mathbb{R}^k$.

Definition 4 *A contractor* $\mathcal{C}_c : \mathbb{IR}^n \to \mathbb{IR}^n$ *associated to the generic constraint*

$$c \equiv g(\mathbf{x}) \in \mathcal{Z} \tag{25}$$

is a function taking a box $[\mathbf{x}]$ *as input and returning a box* $\mathcal{C}_c([\mathbf{x}])$ *satisfying*

$$\mathcal{C}_c([\mathbf{x}]) \subseteq [\mathbf{x}] \tag{26}$$

and

$$g([\mathbf{x}]) \cap \mathcal{Z} = g(\mathcal{C}_c([\mathbf{x}])) \cap \mathcal{Z}. \tag{27}$$

$\mathcal{C}_c$ provides a box containing the set $\{\mathbf{x} \in [\mathbf{x}] \mid g(\mathbf{x}) \in \mathcal{Z}\}$ of solutions of $g(\mathbf{x}) \in \mathcal{Z}$ included in $[\mathbf{x}]$: (26) ensures that the returned box is included in $[\mathbf{x}]$ and (27) ensures that no solution of $g(\mathbf{x}) \in \mathcal{Z}$ in $[\mathbf{x}]$ is lost.

Example 1 *Consider the constraint*

$$c \equiv x_1 \exp(-x_2) \in [4, \infty[\tag{28}$$

and the initial box $[\mathbf{x}] = ([x_1], [x_2])^T = ([0.1, 10], [0.1, 10])^T$. *From (28), one deduces that*

$$x_1 \in ([4, \infty[\,/\, \exp(-[x_2])) \tag{29}$$

Moreover, since one also has $x_1 \in [x_1]$, *one deduces that* $x_1 \in \mathcal{C}_1([\mathbf{x}])$, *with*

$$\mathcal{C}_1([\mathbf{x}]) = ([4, \infty[\,/\, \exp(-[x_2])) \cap [x_1] \tag{30}$$

Similarly one has $x_2 \in \mathcal{C}_2([\mathbf{x}])$, *with*

$$\mathcal{C}_2([\mathbf{x}]) = -\log([4, \infty[\,/\, [x_1]) \cap [x_2] \tag{31}$$

A contractor $\mathcal{C}_c$ *associated to* c *is then*

$$\mathcal{C}_c([\mathbf{x}]) = (\mathcal{C}_1([\mathbf{x}]), \mathcal{C}_2([\mathbf{x}]))^T. \tag{32}$$

This contractor for $[\mathbf{x}] = ([0.1, 10], [0.1, 10])^T$ *provides* $\mathcal{C}_1([\mathbf{x}]) = ([0.1165, 10], [0.1, 0.9163])^T$. *This contractor corresponds to the generic forward-backward contractor, also called HC4-revise in [9]. It may be simply implemented,* e.g., *using IBEX as follows.*

```
Variable x1,x2;
NumConstraint c(x1, x2, x1*exp(-x2) >= 4.0);
CtcFwdBwd contractor(c);
IntervalVector x(2);
x[0]=Interval(0.1,10);
x[1]=Interval(0.1,10);
contractor.contract(x);
```

Various contractors have been proposed in the literature, *e.g.*, the forward-backward contractor, the contractor by parallel linearization, the Newton contractor, the Krawczyk contractor, *etc.* [20]. Most of them are available, *e.g.*, in IBEX. Example 1 shows that a contractor may be easily implemented from the expression of a generic constraint.

Consider now two functions $g_1 : \mathbb{R}^n \to \mathbb{R}^{k_1}$ and $g_2 : \mathbb{R}^n \to \mathbb{R}^{k_2}$, two sets $\mathcal{Z}_1 \subset \mathbb{R}^{k_1}$ and $\mathcal{Z}_2 \subset \mathbb{R}^{k_2}$, and the associated constraints $c_1 \equiv g_1(\mathbf{x}) \in \mathcal{Z}_1$ and $c_2 \equiv g_2(\mathbf{x}) \in \mathcal{Z}_2$. Assume that two contractors $\mathcal{C}_{c_1}$ and $\mathcal{C}_{c_2}$ are available for c_1 and c_2. A contractor $\mathcal{C}_{c_1 \wedge c_2}$ for the conjunction $c_1 \wedge c_2$ of c_1 and c_2 may be obtained as

$$\mathcal{C}_{c_1 \wedge c_2}([\mathbf{x}]) = \mathcal{C}_{c_1}([\mathbf{x}]) \cap \mathcal{C}_{c_2}([\mathbf{x}]), \tag{33}$$

or by composition of contractors

$$\mathcal{C}_{c_1 \wedge c_2}([\mathbf{x}]) = \mathcal{C}_{c_2}(\mathcal{C}_{c_1}([\mathbf{x}])). \tag{34}$$

A contractor $\mathcal{C}_{c_1 \vee c_2}$ for the disjunction $c_1 \vee c_2$ of c_1 and c_2 may be obtained as follows

$$\mathcal{C}_{c_1 \vee c_2}([\mathbf{x}]) = \square\{\mathcal{C}_{c_1}([\mathbf{x}]) \cup \mathcal{C}_{c_2}([\mathbf{x}])\}, \tag{35}$$

see [9], with $\square\{\cdot\}$ the interval hull of a set.

Using a contractor $\mathcal{C}_c$ for (25), one is able to characterize some $[\tilde{\mathbf{x}}] \subset [\mathbf{x}]$ such that $\forall \mathbf{x} \in [\tilde{\mathbf{x}}]$, $g(\mathbf{x}) \notin \mathcal{Z}$.

Proposition 5 *Consider a box* $[\mathbf{x}]$, *the elementary constraint* (25), *and the contracted box* $\mathcal{C}_c([\mathbf{x}]) \subseteq [\mathbf{x}]$. *Then,*

$$\forall \mathbf{x} \in [\mathbf{x}] \setminus \mathcal{C}_c([\mathbf{x}]), \text{ one has } g(\mathbf{x}) \notin \mathcal{Z}, \tag{36}$$

where $[\mathbf{x}] \setminus \mathcal{C}_c([\mathbf{x}])$ *denotes the box* $[\mathbf{x}]$ *deprived of* $\mathcal{C}_c([\mathbf{x}])$, *which is not necessarily a box.*

PROOF. Consider $\mathbf{x} \in [\mathbf{x}] \setminus \mathcal{C}_c([\mathbf{x}])$ and assume that $g(\mathbf{x}) \in \mathcal{Z}$. Since $g(\mathbf{x}) \in \mathcal{Z}$ and $\mathbf{x} \in [\mathbf{x}]$, one should have $\mathbf{x} \in \mathcal{C}_c([\mathbf{x}])$, according to (27), which contradicts the fact that $\mathbf{x} \in [\mathbf{x}] \setminus \mathcal{C}_c([\mathbf{x}])$. □

Proposition 5 can be used to eliminate $[\mathbf{p}]$ or a part of $[\mathbf{p}]$ for which it is not possible to find any $\mathbf{p}$ satisfying (21). Consider the constraint

$$\tau \equiv (u(\mathbf{x}, \mathbf{p}) \in \mathcal{A}) \vee (v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \in \mathcal{B}) \tag{37}$$

and a contractor $\mathcal{C}_\tau$ for this constraint. It involves elementary contractors for the components of the disjunction in (37), combined as in (35). For the boxes $[\mathbf{x}]$, $[\mathbf{p}]$, and $[\mathbf{d}]$, one gets

$$\left([\mathbf{x}]', [\mathbf{p}]', [\mathbf{d}]'\right) = \mathcal{C}_\tau\left([\mathbf{x}], [\mathbf{p}], [\mathbf{d}]\right), \tag{38}$$

where $[\mathbf{x}]'$, $[\mathbf{p}]'$, and $[\mathbf{d}]'$ are the contracted boxes. Three cases may then be considered.

(1) If $[\mathbf{p}] \setminus [\mathbf{p}]' \neq \emptyset$, then $\forall \mathbf{p} \in [\mathbf{p}] \setminus [\mathbf{p}]'$, $\forall \mathbf{x} \in [\mathbf{x}]$, $\forall \mathbf{d} \in [\mathbf{d}]$,

$$u(\mathbf{x}, \mathbf{p}) \notin \mathcal{A} \wedge v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \notin \mathcal{B}, \tag{39}$$

and there is no $\mathbf{p} \in [\mathbf{p}] \setminus [\mathbf{p}]'$ such that (37) holds true for all $\mathbf{x} \in [\mathbf{x}]$ and for all $\mathbf{d} \in [\mathbf{d}]$. Consequently, the search space for $\mathbf{p}$ can be reduced to $[\mathbf{p}]'$, see Figure 1 (a).

(2) If $[\mathbf{x}] \setminus [\mathbf{x}]' \neq \emptyset$ then, from Proposition 5, one has $\forall \mathbf{p} \in [\mathbf{p}]$, $\forall \mathbf{x} \in [\mathbf{x}] \setminus [\mathbf{x}]'$, $\forall \mathbf{d} \in [\mathbf{d}]$,

$$u(\mathbf{x}, \mathbf{p}) \notin \mathcal{A} \wedge v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \notin \mathcal{B}, \tag{40}$$

and there is no $\mathbf{p} \in [\mathbf{p}]$ such that (37) holds true for all $\mathbf{x} \in [\mathbf{x}]$, see Figure 1 (b).

(3) If $[\mathbf{d}] \setminus [\mathbf{d}]' \neq \emptyset$, then $\forall \mathbf{p} \in [\mathbf{p}]$, $\forall \mathbf{x} \in [\mathbf{x}]$, $\forall \mathbf{d} \in [\mathbf{d}] \setminus [\mathbf{d}]'$,

$$u(\mathbf{x}, \mathbf{p}) \notin \mathcal{A} \wedge v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \notin \mathcal{B}, \tag{41}$$

and there is no $\mathbf{p} \in [\mathbf{p}]$ such that (37) holds true for all $\mathbf{d} \in [\mathbf{d}]$, see Figure 1 (b).

One can reduce the size of the sets for the state $\mathbf{x}$ and the disturbance $\mathbf{d}$ on which (37) has to be verified using the contraction on the negation of this constraint. Consider the negation $\overline{\tau}$ of τ

$$\overline{\tau} \equiv \left(u(\mathbf{x}, \mathbf{p}) \in \overline{\mathcal{A}}\right) \wedge \left(v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \in \overline{\mathcal{B}}\right) \tag{42}$$

and a contractor $\mathcal{C}_{\overline{\tau}}$ for this constraint. Assume that after applying $\mathcal{C}_{\overline{\tau}}$ for the boxes $[\mathbf{x}]$, $[\mathbf{p}]$, and $[\mathbf{d}]$, one gets

$$\left([\mathbf{x}]", [\mathbf{p}]", [\mathbf{d}]"\right) = \mathcal{C}_{\overline{\tau}}\left([\mathbf{x}], [\mathbf{p}], [\mathbf{d}]\right). \tag{43}$$

From Proposition 5, one knows that

$$\forall (\mathbf{x}, \mathbf{p}, \mathbf{d}) \in ([\mathbf{x}] \times [\mathbf{p}] \times [\mathbf{d}]) \setminus ([\mathbf{x}]" \times [\mathbf{p}]" \times [\mathbf{d}]"), \quad u(\mathbf{x}, \mathbf{p}) \in \mathcal{A} \vee v(\mathbf{x}, \mathbf{p}, \mathbf{d}) \in \mathcal{B}. \tag{44}$$

Indeed, if $[\mathbf{p}]" = [\mathbf{p}]$, one can focus on the search for some $\mathbf{p} \in [\mathbf{p}]$ satisfying (37) by considering only $[\mathbf{x}]" \times [\mathbf{d}]"$, since for all $(\mathbf{x}, \mathbf{d}) \in ([\mathbf{x}] \times [\mathbf{d}]) \setminus ([\mathbf{x}]" \times [\mathbf{d}]")$, (37) is satisfied for all $\mathbf{p} \in [\mathbf{p}]$, see Figure 2 (a).

Now, if $[\mathbf{p}]" \neq [\mathbf{p}]$, then any $\mathbf{p} \in [\mathbf{p}] \setminus [\mathbf{p}]"$ will satisfy (37) for all $(\mathbf{x}, \mathbf{d}) \in ([\mathbf{x}] \times [\mathbf{d}])$, see Figure 2 (b).

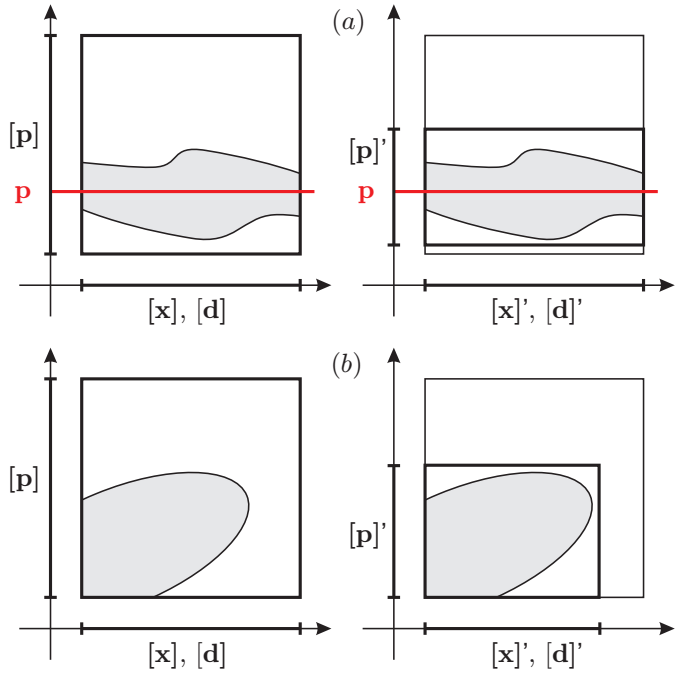

Fig. 1. Contractions using $\mathcal{C}_\tau$; the set for which (37) is satisfied is in gray; (a) $[\mathbf{p}] \setminus [\mathbf{p}]' \neq \emptyset$ and the search space for satisfying $\mathbf{p}$ can be reduced to $[\mathbf{p}]'$; (b) $[\mathbf{x}]' \neq [\mathbf{x}]$ or $[\mathbf{d}]' \neq [\mathbf{d}]$, it is thus not possible to find some $\mathbf{p} \in [\mathbf{p}]$ such that (37) is satisfied for all $\mathbf{x} \in [\mathbf{x}]$ and all $\mathbf{d} \in [\mathbf{d}]$.

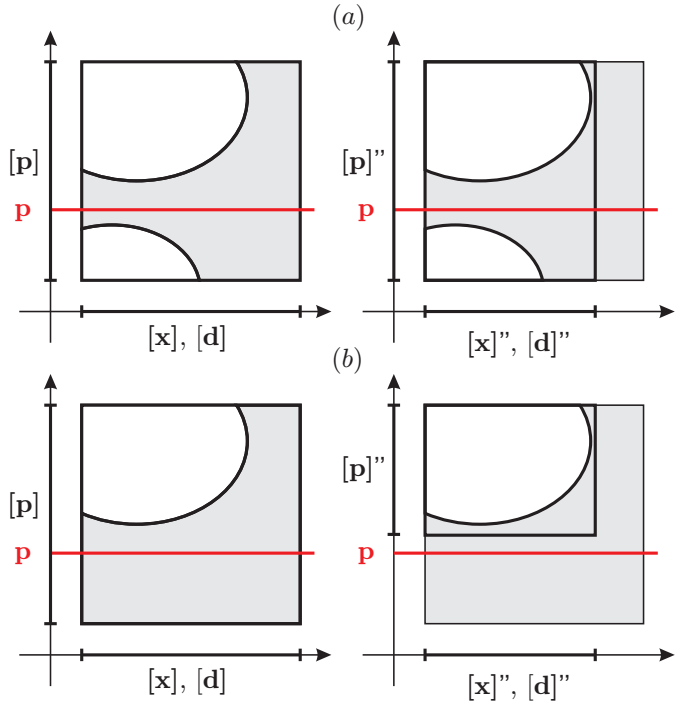

Fig. 2. Contractions using $\mathcal{C}_{\overline{\tau}}$; the set for which (42) is satisfied is in white; (a) $[\mathbf{x}]" \neq [\mathbf{x}]$ and/or $[\mathbf{d}]" \neq [\mathbf{d}]$ and one has only to find some suitable $\mathbf{p} \in [\mathbf{p}]$ such that (37) is satisfied for all $\mathbf{x} \in [\mathbf{x}]"$ and all $\mathbf{d} \in [\mathbf{d}]"$; (b) $[\mathbf{p}]" \neq [\mathbf{p}]$, one may choose any $\mathbf{p} \in [\mathbf{p}] \setminus [\mathbf{p}]"$ (for example the value of $\mathbf{p}$ indicated in red) and (37) will hold true for all $(\mathbf{x}, \mathbf{d}) \in ([\mathbf{x}] \times [\mathbf{d}])$.

4.2.3 Bisection

One is unable to decide whether some $\mathbf{p} \in [\mathbf{p}]$ satisfies (21) when

$$[u]([\mathbf{x}], \mathbf{p}) \cap \mathcal{A} \neq \emptyset \text{ and } [u]([\mathbf{x}], \mathbf{p}) \not\subseteq \mathcal{A} \tag{45}$$

or when

$$[v]([\mathbf{x}],\mathbf{p},[\mathbf{d}])\cap\mathcal{B}\neq\emptyset\,\text{and}\,[v]([\mathbf{x}],\mathbf{p},[\mathbf{d}])\not\subseteq\mathcal{B}. \tag{46}$$

This situation occurs in two cases. First, when the selected $\mathbf{p}$ does not satisfy (21) for all $\mathbf{x}\in[\mathbf{x}]$ and for all $\mathbf{d}\in[\mathbf{d}]$. Second, when inclusion functions introduce some *pessimism*, *i.e.*, they provide an over-approximation of the range of functions over intervals.

To address both cases, one may perform bisections of $[\mathbf{x}]\times[\mathbf{d}]$ and try to verify (21) on the resulting sub-boxes for the *same* $\mathbf{p}$. Bisection allows to isolate subsets of $[\mathbf{x}]\times[\mathbf{d}]$ on which one may show that (23) holds true. Bisections also reduce pessimism, and may thus facilitate the verification of (21). The bisection of $[\mathbf{x}]\times[\mathbf{d}]$ is performed within CSC as long as the width of the bisected boxes are larger than some $\varepsilon_{\mathrm{x}}>0$. When CSC is unable to prove (21) or (23) and when all bisected boxes are smaller than ε_{x}, CSC returns `unknown`.

FPS performs similar bisections on $[\mathbf{p}]$ and stops when the width of all bisected boxes are smaller than $\varepsilon_{\mathrm{p}}>0$.

4.2.4 Composition of constraints

To prove the safety of the dynamical system (1), Proposition 2 shows that one has to find some $\mathbf{p}\in[\mathbf{p}]$ such that $\forall\mathbf{x}\in[\mathbf{x}],\forall\mathbf{d}\in[\mathbf{d}],\xi(\mathbf{x},\mathbf{p},\mathbf{d})$ holds true. Since $\xi(\mathbf{x},\mathbf{p},\mathbf{d})$ consists of the conjunction of three elementary constraints of the form (20), validation requires the verification of (21) for the *same* $\mathbf{p}$ considering (7), (8), and (9) *simultaneously*. Invalidation may be performed as soon as one is able to prove that one of the constraints (7), (8), or (9) does not hold true using (23). Contraction may benefit from the conjunction or disjunction of these constraints, as introduced in (34) and (35).

4.3 CSC-FPS algorithms with contractors

The CSC-FPS algorithm, presented in [21] is supplemented with the contractors introduced in Section 4.2 to improve its efficiency. No specific contractor is indicated in this section. Nevertheless, the forward-backward contractor used in the experimental part provides good results and is provided directly from the explicit expression of the constraint using tools such as IBEX.

FPS, described in Algorithm 1, searches for some *satisfying* $\mathbf{p}\in[\mathbf{p}]$, *i.e.*, some $\mathbf{p}\in[\mathbf{p}]$ such that $\xi(\mathbf{x},\mathbf{p},\mathbf{d})$ introduced in Proposition 2 holds true for all $\mathbf{x}\in[\mathbf{x}]$ and all $\mathbf{d}\in[\mathbf{d}]$. This may require to bisect $[\mathbf{p}]$ into subboxes stored in a queue $\mathcal{Q}$, which initial content is $[\mathbf{p}]$.

A subbox $[\mathbf{p}]_0\subseteq[\mathbf{p}]$ is extracted from $\mathcal{Q}$ at Line 6. A reduction of $[\mathbf{p}]_0$ is then performed at Line 8 to eliminate values of $\mathbf{p}\in[\mathbf{p}]_0$ which cannot be satisfying. To facilitate contraction, *e.g.*, with the classical forward-backward contractor, specific $\mathbf{x}\in[\mathbf{x}]$ are chosen; here only the midpoint $\mathrm{m}([\mathbf{x}])$ of $[\mathbf{x}]$ is considered.

At Line 9, if $[\mathbf{p}]_0'$ is empty, the next box in $\mathcal{Q}$ has to be explored. Otherwise, at Lines 13-15, CSC is called for each constraint t_0, t_u, and t_b to verify whether $\mathrm{m}\left([\mathbf{p}]_0'\right)$, the midpoint of $[\mathbf{p}]_0'$, is satisfying. When all calls of CSC return `true` at Line 17, a barrier function with parameter $\mathrm{m}\left([\mathbf{p}]_0'\right)$ is found. When one of the calls of CSC returns `false` at Line 21, $[\mathbf{p}]_0'$ is proved not to contain any satisfying $\mathbf{p}$. In all other cases, when $\mathrm{w}([\mathbf{p}]_0')$, the width of $[\mathbf{p}]_0'$, is larger than ε_{p}, $[\mathbf{p}]_0'$ is bisected and the resulting subboxes are stored in $\mathcal{Q}$, see Lines 24-28. When $[\mathbf{p}]_0'$ is too small, even if one was not able to decide whether it contains a satisfying $\mathbf{p}$, it is not further considered to ensure termination of FPS in finite time. The price to be paid in such situation is the impossibility to state whether the initial box $[\mathbf{p}]$ contains some satisfying $\mathbf{p}$. This is done by setting flag to `unknown` at Line 31. Finally, when $\mathcal{Q}=\emptyset$, no satisfying $\mathbf{p}$ has been found. Whether $[\mathbf{p}]$ may however contain some satisfying $\mathbf{p}$ in an unexplored subbox $[\mathbf{p}]_0'$ depends on the value of flag.

CSC, described in Algorithm 2, determines whether $\mathrm{m}([\mathbf{p}])$ satisfies (16) by showing that $\tau(\mathbf{x},\mathrm{m}([\mathbf{p}]),\mathbf{d})$ holds true for all $\mathbf{x}\in[\mathbf{x}]$ and all $\mathbf{d}\in[\mathbf{d}]$. Alternatively, CSC may prove that there is no $\mathbf{p}\in[\mathbf{p}]$ satisfying (16) for all $\mathbf{x}\in[\mathbf{x}]$ and all $\mathbf{d}\in[\mathbf{d}]$.

For that purpose, due to the pessimism of inclusion functions, it may be necessary to bisect $[\mathbf{x}]\times[\mathbf{d}]$ in subboxes stored in a stack $\mathcal{S}$ initialized with $[\mathbf{x}]\times[\mathbf{d}]$.

Algorithm 1 FPS

1: **procedure** FPS(ξ_0, ξ_u, ξ_b, $[\mathbf{p}]$,$[\mathbf{x}]$, $[\mathbf{d}]$)
2: ▷ ξ_0, ξ_u, and ξ_b associated to (7), (8), and (9)
3: queue $\mathcal{Q} := [\mathbf{p}]$
4: flag := `true`
5: **while** $\mathcal{Q} \neq \emptyset$ **do**
6: $[\mathbf{p}]_0$:= dequeue($\mathcal{Q}$)
7: ▷ Reduction of $[\mathbf{p}]_0$ using (34), (38), and (39)
8: $[\mathbf{p}]'_0 := \mathcal{C}_{\xi_0 \wedge \xi_u}(\mathrm{m}([\mathbf{x}]), [\mathbf{p}]_0)$
9: **if** $[\mathbf{p}]'_0 = \emptyset$ **then**
10: continue ▷ There is no satisfying $\mathbf{p} \in [\mathbf{p}]_0$
11: **end if**
12: ▷ Call CSC for each constraint (7), (8), and (9)
13: r_0:= CSC(ξ_0,$[\mathbf{p}]'_0$, $[\mathbf{x}]$,$[\mathbf{d}]$)
14: r_u:= CSC(ξ_u,$[\mathbf{p}]'_0$, $[\mathbf{x}]$,$[\mathbf{d}]$)
15: r_b:= CSC(ξ_b,$[\mathbf{p}]'_0$, $[\mathbf{x}]$,$[\mathbf{d}]$)
16: ▷ $\mathrm{m}([\mathbf{p}]'_0)$ is satisfying if all CSCs hold true
17: **if** (r_0=`true`)$\wedge$(r_u=`true`)$\wedge$(r_b=`true`) **then**
18: return(`true`,$\mathrm{m}([\mathbf{p}]'_0)$)
19: **end if**
20: ▷ no solution in $[\mathbf{p}]'_0$ if one CSC holds false
21: **if** (r_0=`false`)$\vee$(r_u=`false`)$\vee$(r_b=`false`) **then**
22: continue
23: **end if**
24: **if** $\mathrm{w}([\mathbf{p}]'_0) \geqslant \varepsilon_{\mathrm{p}}$ **then**
25: ▷ no conclusion for $[\mathbf{p}]'_0$ and large enough to be bisected
26: $([\mathbf{p}]_1, [\mathbf{p}]_2)$:=bisect($[\mathbf{p}]_0{}'$)
27: enqueue($[\mathbf{p}]_1$) in $\mathcal{Q}$
28: enqueue($[\mathbf{p}]_2$) in $\mathcal{Q}$
29: **else**
30: ▷ no conclusion for $[\mathbf{p}]'_0$ and too small to be bisected
31: flag:=`unknown`
32: **end if**
33: **end while**
34: **if** flag=`unknown` **then**
35: ▷ a small $[\mathbf{p}]'_0$ was not explored, impossible to state whether there is no valid solution in $[\mathbf{p}]$
36: return(`unknown`,$\emptyset$)
37: **else**
38: return(`false`,$\emptyset$) ▷ no valid solution in $[\mathbf{p}]$
39: **end if**
40: **end procedure**

For each subbox $[\mathbf{x}]_0 \times [\mathbf{d}]_0 \subseteq [\mathbf{x}] \times [\mathbf{d}]$, CSC determines at Line 7 whether $\mathrm{m}([\mathbf{p}]_0)$ is satisfying. Alternatively, CSC tries to prove that $[\mathbf{p}]_0$ does not contain any satisfying $\mathbf{p}$ for all $(\mathbf{x}, \mathbf{d}) \in [\mathbf{x}]_0 \times [\mathbf{d}]_0$. This is done in two steps. First, at Line 11, one applies (24) considering some $(\mathbf{x}, \mathbf{d}) \in [\mathbf{x}]_0 \times [\mathbf{d}]_0$, here taken as the midpoint of $[\mathbf{x}]_0 \times [\mathbf{d}]_0$. Second, one applies Proposition 5 at Lines 15-17 using the result of a contractor for τ, as described in (40) and (41).

When one is not able to conclude and provided that $\mathrm{w}([\mathbf{x}]_0 \times [\mathbf{d}]_0)$ is larger than ε_{x}, some parts of $[\mathbf{x}]_0 \times [\mathbf{d}]_0$ for which $\mathrm{m}([\mathbf{p}]_0)$ is satisfying are removed at Line 23, *e.g.*, using the forward-backward contractor, before performing a bisection and storing the resulting subboxes in $\mathcal{S}$. When $\mathrm{w}([\mathbf{x}]_0 \times [\mathbf{d}]_0)$ is less than ε_{x}, to ensure completion of CSC in a finite time, $[\mathbf{x}]_0 \times [\mathbf{d}]_0$ is not further explored. The price to be paid is an impossibility to determine whether $\mathrm{m}([\mathbf{p}]_0)$ is satisfying for all $(\mathbf{x}, \mathbf{d}) \in [\mathbf{x}]_0 \times [\mathbf{d}]_0$. This is indicated by setting flag to `unknown` at Line 30. Nevertheless, one may still prove that $[\mathbf{p}]_0$ does not contain any satisfying $\mathbf{p}$ considering other subboxes of $[\mathbf{x}] \times [\mathbf{d}]$.

5 Examples

This section presents experiments for the characterization of barrier functions. The considered dynamical systems are described first before providing numerical results, comparison of different approaches and discussions.

Algorithm 2 CSC

procedure CSC(τ,$[\mathbf{p}]_0$,$[\mathbf{x}]$,$[\mathbf{d}]$) ▷ τ is of the form (16)
  stack $S := [\mathbf{x}] \times [\mathbf{d}]$
  flag:=`true`
  while $S \neq \emptyset$ **do**
    $[\mathbf{x}]_0 \times [\mathbf{d}]_0$:=pop(S)
    ▷ Validation of m($[\mathbf{p}]_0$) using (21)
    if $[u]([\mathbf{x}]_0, \mathrm{m}([\mathbf{p}]_0)) \subseteq \mathcal{A} \vee [v]([\mathbf{x}]_0, \mathrm{m}([\mathbf{p}]_0), [\mathbf{d}]_0) \subseteq \mathcal{B}$ **then**
      continue
    end if
    ▷ Elimination of $[\mathbf{p}]_0$ using (24)
    if $[u]\left(\mathrm{m}\left([\mathbf{x}]_0\right), [\mathbf{p}]_0\right) \cap \mathcal{A} = \emptyset \wedge [v]\left(\mathrm{m}\left([\mathbf{x}]_0\right), [\mathbf{p}]_0, \mathrm{m}\left([\mathbf{d}]_0\right)\right) \cap \mathcal{B} = \emptyset$ **then**
      return(`false`)
    end if
    ▷ Reduction of $[\mathbf{p}]_0$ using (38)
    $([\mathbf{x}]_0', [\mathbf{p}]_0', [\mathbf{d}]_0') := \mathcal{C}_\tau([\mathbf{x}]_0, [\mathbf{p}]_0, [\mathbf{d}]_0)$
    ▷ Prove with Proposition 5 that $[\mathbf{p}]_0$ does not contain any satisfying $\mathbf{p}$
    if $[\mathbf{x}]_0' \neq [\mathbf{x}]_0 \vee [\mathbf{d}]_0' \neq [\mathbf{d}]_0$ **then**
      return(`false`)
    end if
    ▷ $[\mathbf{x}]_0 \times [\mathbf{d}]_0$ undetermined and large enough to be bisected
    if $(\mathrm{w}([\mathbf{x}]_0 \times [\mathbf{d}]_0) \geqslant \varepsilon_\mathrm{x})$ **then**
      ▷ Reduction of the state space using (43) and (44)
      $([\mathbf{x}]_0'', [\mathbf{d}]_0'') := \mathcal{C}_{\overline{\tau}}([\mathbf{x}]_0, \mathrm{m}([\mathbf{p}]_0), [\mathbf{d}]_0)$
      ▷ Bisection
      $([\mathbf{x}]_1 \times [\mathbf{d}]_1, [\mathbf{x}]_2 \times [\mathbf{d}]_2)$:=bisect($[\mathbf{x}]_0'' \times [\mathbf{d}]_0''$)
      push($[\mathbf{x}]_1 \times [\mathbf{d}]_1$) in S
      push($[\mathbf{x}]_2 \times [\mathbf{d}]_2$) in S
    else
      ▷ $[\mathbf{x}]_0 \times [\mathbf{d}]_0$ too small to be further explored
      flag:=`unknown`
    end if
  end while
  return(flag)
end procedure

5.1 Dynamical system descriptions

For the following examples, one provides the dynamics of the system, the constraints g_0 and g_u for the definition of the sets $\mathcal{X}_0$ and $\mathcal{X}_u$, the state space $[\mathbf{x}]$, and the parametric expression of the barrier function. In all cases, the parameter space is chosen as $[\mathbf{p}] = [-10, 10]^m$ where m is the number of parameters.

Example 2 *Consider the 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 $g_0(\mathbf{x}) = (x_1+1.25)^2+(x_2-1.25)^2-0.05$, $g_u(\mathbf{x}) = (x_1+2.5)^2+(x_2-0.8)^2-0.05$, *and* $[\mathbf{x}] = [-10^3, 0]\times[-10^3, 10^3]$. *The parametric barrier function is* $B(\mathbf{x}, \mathbf{p}) = \frac{p_1 p_2 (x_0+p_3)}{(x_0+p_3)^2+p_2^2} + x_1 + p_4$.

Example 3 *Consider the system from [2]*

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

with $g_0(\mathbf{x}) = (x_1 - 1.125)^2 + (x_2 - 0.625)^2 - 0.0125$, $g_u(\mathbf{x}) = (x_1 - 0.875)^2 + (x_2 - 0.125)^2 - 0.0125$, *and* $[x] = [-100, 100] \times [-100, 100]$. *The parametric barrier function used is* $B(\mathbf{x}, \mathbf{p}) = \ln(p_1 x_1) - \ln(x_2) + p_2 x_2 + p_3$.

Example 4 *Consider the system from [24]*

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\end{pmatrix} = \begin{pmatrix} x_2 \\ -\frac{x1+x2}{\sqrt{1+(x1+x2)^2}}\end{pmatrix}$$

with $g_0(\mathbf{x}) = {x_1}^2 + {x_2}^2 - 0.5$, $g_u(\mathbf{x}) = (x_1-3.5)^2+(x_2-0.5)^2-0.5$, *and* $[\mathbf{x}] = [-10^3, 10^3]\times[-100, 100]$. *A quadratic parametric barrier function is chosen* $B(\mathbf{x},\mathbf{p}) = p_1x_1^2 + p_2x_2^2 + p_3x_1x_2 + p_4x_1 + p_5x_2 + p_6$.

Example 5 *Consider the disturbed system from [30]*

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

with $g_0(\mathbf{x}) = (x_1 - 1.5)^2 + {x_2}^2 - 0.25$, $g_u(\mathbf{x}) = (x_1 + 0.8)^2 + (x_2 + 1)^2 - 0.25$, $[\mathbf{x}] = [-100, 100]\times[-10, 10]$, *and* $d \in [0.9, 1.1]$. *The parametric barrier function* $B(\mathbf{x},\mathbf{p}) = p_1x_1^2 + p_2x_2^2 + p_3x_1x_2 + p_4x_1 + p_5x_2 + p_6$ *is considered.*

Example 6 *Consider the system with a limit cycle*

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

with $g_0(\mathbf{x}) = (x_1-1)^2+(x_2+1.5)^2-0.05$, $g_u(\mathbf{x}) = (x_1+0.6)+(x_2-1)^2-0.05$, *and* $[\mathbf{x}] = [-10^3, 10^3]\times[-10^3, 10^3]$. *The parametric barrier function used is* $B(\mathbf{x},\mathbf{p}) = \left(\frac{x_1+p_1}{p_2}\right)^2 + \left(\frac{x_2+p_3}{p_4}\right)^2 - 1$.

Example 7 *Consider the Lorenz system from [40]*

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{pmatrix} = \begin{pmatrix} 10(x_2 - x_1) \\ x_1(28 - x_3) - x_2 \\ x_1x_2 - \frac{8}{3}x_3\end{pmatrix}$$

with $g_0(\mathbf{x}) = (x_1+14.5)^2+(x_2+14.5)^2+(x_3-12.5)^2-0.25$, $g_u(\mathbf{x}) = (x_1+16.5)^2+(x_2+14.5)^2+(x_3-2.5)^2-0.25$, *and* $[\mathbf{x}] = [-20, 20]\times[-20, 0]\times[-20, 20]$. *The considered parametric barrier function is* $B(\mathbf{x},\mathbf{p}) = p_1x_1^2+p_2x_1+p_3x_3+p_4$.

Example 8 *Consider the system from [23]*

$$\begin{pmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\\ \dot{x}_4\\ \dot{x}_5\\ \dot{x}_6\end{pmatrix} = \begin{pmatrix} -x_1 + 4x_2 - 6x_3x_4 \\ -x_1 - x_2 + x_5^3 \\ x_1x_4 - x_3 + x_4x_6 \\ x_1x_3 + x_3x_6 - x_4^3 \\ -2x_2^3 - x_5 + x_6 \\ -3x_3x_4 - x_5^3 - x_6\end{pmatrix}$$

with $g_0(\mathbf{x}) = (x_1 - 3.05)^2 + (x_2 - 3.05)^2 + (x_3 - 3.05)^2 + (x_4 - 3.05)^2 + (x_5 - 3.05)^2 + (x_6 - 3.05)^2 - 0.0001$, $g_u(\mathbf{x}) = (x_1 - 7.05)^2 + (x_2 - 3.05)^2 + (x_3 - 7.05)^2 + (x_4 - 7.05)^2 + (x_5 - 7.05)^2 + (x_6 - 7.05)^2 - 0.0001$, *and* $[\mathbf{x}] = [0, 10]\times[0, 10]\times[2, 10]\times[0, 10]\times[0, 10]\times[0, 10]$. *The considered parametric barrier function is* $B(\mathbf{x},\mathbf{p}) = p_1x_1^2 + p_2x_2^4 + p_3x_3^2 + p_4x_4^2 + p_5x_5^4 + p_6x_6^2 + p_7$.

5.2 Experimental conditions and results

CSC-FPS, presented in Section 4, has been implemented using the IBEX library [3, 9]. The selection of candidate barrier functions is performed choosing polynomials with increasing degree, except for Examples 2, 3, and 5, where parametric functions taken from [1, 2] are considered.

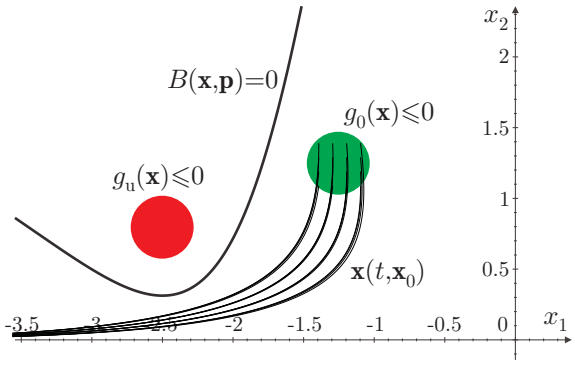

Fig. 3. Results for Example 2.

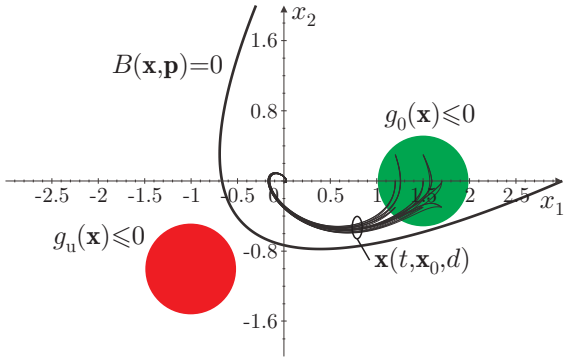

Fig. 4. Results for Example 5 with various values of the disturbance d.

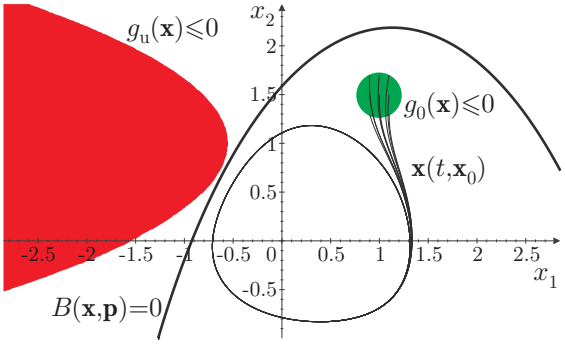

Fig. 5. Results for Example 6.

For each example, the computing time to get a valid barrier function and the number of bisections of the search box $[\mathbf{p}]$ are provided. Table 1 summarizes the results for the versions of CSC-FPS with and without contractors. As in [21], we choose $\varepsilon_{\mathrm{x}} = 10^{-1}$ and $\varepsilon_{\mathrm{p}} = 10^{-5}$. Computations were done on an Intel core 1.7 GHz processor with 8 GB of RAM. If after 30 minutes of computations no valid barrier function has been found, the search is stopped. This is denoted by T.O. (time out) in Table 1. Moreover, for Examples 2, 5, and 6, graphical representation of the computed barrier functions are provided. In Figures 3, 4, and 5, $\mathcal{X}_0$ is in green, $\mathcal{X}_u$ is in red, the bold line is the barrier function and some trajectories starting from $\mathcal{X}_0$ are also represented.

The results in Table 1 show the importance of contractors which are beneficial in all cases. Thanks to contractors, valid barrier functions were obtained for all examples, which is not the case employing the original version of CSC-FPS proposed in [21]. In theory, both FPS and CSC are of exponential complexity in the dimension m of the parameter space and n of the state space. In practice, contractors allow, on the considered examples, to get solutions in a reasonable amount of time, even for systems with a larger number of states and parameters.

The search for barrier functions using the relaxed version (14) of the constraint (12) as in [30], has also been

Table 1
Results of CSC-FPS without and with contractors

			Without contr.		With contr.	
Example	n	m	time	bisect.	time	bisect.
1	2	4	36s	4520	16s	4553
2	2	3	T.O.	/	1s	159
3	2	6	1133s	20388	1s	6
4	2	6	253s	14733	7s	435
5	2	4	T.O.	/	98s	4072
6	3	4	167s	1753	21s	47
7	6	7	697s	67600	1s	261

performed using the version of our algorithms with contractors considering the same parametric barrier functions. We were not able to find a valid barrier function in less than 30 minutes. This shows the detrimental effect of the relaxation (14) on the search technique.

Some examples were also addressed using RSolver, which is a tool to solve some classes of QCSP [32]. Nevertheless, this requires some modifications of the examples, since RSolver does not address dynamics or constraints involving divisions [33]. Moreover, the type of constraints processed by RSolver, for problems such as (20), allows only parametric barrier functions that are linear in the parameters. Outer-approximating boxes for the initial and the unsafe regions $\mathcal{X}_0$, $\mathcal{X}_u$ defined by g_0 and g_u are given to Rsolver. As a consequence, Examples 2, 3 and 4 could not be considered. Only Examples 5, 6, and 7 were tested by Rsolver, which was able to find a satisfying barrier function only for Lorenz in less than 1s. RSolver was unable to find a solution for Examples 5 and 6. Rsolver is well designed for problems with parametric barriers linear in the parameters, but has difficulties for non-linear problem and non-convex constraints such as (12).

Table 2 provides results obtained using dReal [15], which is sat-modulo-theory solver [6], based on interval analysis tools to handle first-order logic formulas over the real numbers. In the tested version (3.15.11.03) of dReal, the support for QCSP such as (20) does not benefit from contractors. A randomization heuristic is used in the search of satisfying parameter vectors. This leads to results obtained in a amount of time varying with the runs. A precision parameter of 0.001 (a tentative to have a fair comparison with respect to values of ε_{X} and ε_{P} used in our algorithm) was employed. Due to the non-deterministic behavior of dReal, each example has be run 10 times and we report minimal and maximal values for the execution time and the number of bisections. When no solution is found in less than 30 minutes, the value T.O. (time out) is indicated.

Table 2 shows that dReal can handle all considered problems despite the absence of contractors. In particular, one observes that randomization speed up the search for valid parameters, as for example, in case of examples 2, 5, 6 and 7. Nevertheless, as a consequence, the overall algorithm has a varying execution time. The absence of contractors may increase the parameter search time, as for example, in the case of examples 1, 3, and 4. Combining randomized methods and contractor programming should be very beneficial to efficiently solve QCSP.

Table 2
Results of dReal for 10 executions with precision of 0.001

	Time (in s)		Bisections	
Example	Min	Max	Min	Max
1	285.1	T.O.	31337	/
2	0.06	0.19	35	94
3	113.3	T.O.	61	/
4	533.3	T.O.	51	/
5	0.05	0.06	34	39
6	0.08	0.08	54	55
7	0.4	0.4	107	109

6 Conclusion

This paper presents a new method to find parametric barrier functions for nonlinear continuous-time perturbed dynamical systems. The proposed technique addresses the design of quite general barrier functions for continuous-time systems described by nonlinear dynamics that are also general. The search for barrier functions is formulated as an interval quantified constraint satisfaction problem. A branch-and-prune algorithm proposed in [21] has been supplemented with contractors to address this problem. Contractors are instrumental in solving problems with large number of parameters. The proposed approach can thus find barrier functions for a large class of possibly perturbed dynamical systems.

Alternative techniques based on RSolver or dReal may be significantly more efficient for some specific classes of problems, *e.g.*, where the parameters appear linearly in the parametric barrier functions. Combining ideas from RSolver, dReal and our approach may be very useful to further improve the global efficiency of barrier function characterization.

Future work will be dedicated to the search for the class of parametric barrier functions to consider. This may be done by exploring a library of candidate barrier functions. In our approach rejection of a candidate function occurs mainly after a timeout. Even if contractors aiming at eliminating some parts of the parameter space were defined, their efficiency is limited. Better contractors for that purpose may be very helpful.

An other future research direction is to extend the proposed method to hybrid systems as done in [30], *i.e.*, to consider a set of quantified constraints for each location of an hybrid automaton and the constraints associated to the transitions.

Acknowledgements

Authors are thankful to Sicun Gao and Soonho Kong for their valuable help on dReal.

References

[1] Mathcurve. http://www.mathcurve.com/.

[2] Amir A. Ahmadi, Miroslav Krstic, and Pablo A. Parrilo. A globally asymptotically stable polynomial vector field with no polynomial Lyapunov function. In *Conference on Decision and Control and European Control Conference*, pages 7579–7580, 2011.

[3] Ignacio Araya, Gilles Trombettoni, Bertrand Neveu, and Gilles Chabert. Upper bounding in inner regions for global optimization under inequality constraints. *Journal of Global Optimization*, 60(2):145–164, 2012.

[4] Nikos Aréchiga, James Kapinski, Jyotirmoy V. Deshmukh, André Platzer, and Bruce Krogh. Forward invariant cuts to simplify proofs of safety. In *Conference on Embedded Software*, pages 227–236. IEEE Press, 2015.

[5] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control*, pages 20–31. Springer, 2000.

[6] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*. 2015. In preparation.

[7] Richard E. Bellman and Kenneth L. Cooke. Differential-difference equations. *RAND Corporation*, 1963.

[8] Frédéric Benhamou and Frédéric Goualard. Universally quantified interval constraints. In *Principles and Practice of Constraint Programming*, pages 67–82. Springer, 2000.

[9] Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.

[10] Xin Chen, Erika Abrahám, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Real-Time Systems Symposium*, pages 183–192. IEEE, 2012.

[11] Alongkrit Chutinan and Bruce H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control*, pages 76–90. Springer, 1999.

[12] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299 – 328, 1991.

[13] Liyun Dai, Ting Gan, Bican Xia, and Naiun Zhan. Barrier certificates revisited. preprint, 2013.

[14] Goran Frehse, Colas Le Guernic, Alexandre Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395. Springer, 2011.

[15] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *Conference on Automated Deduction*, volume 7898 of *LNCS*, pages 208–214. Springer, 2013.

[16] Roberto Genesio, Michele Tartaglia, and Antonio Vicino. On the estimation of asymptotic stability regions: State of the art and new proposals. *Transaction on Automatic Control*, 30(8):747–755, 1985.

[17] Éric Goubault, Jacques-Henri Jourdan, Sylvie Putot, and Sriram Sankaranarayanan. Finding non-polynomial positive invariants and lyapunov functions for polynomial systems through darboux polynomials. In *American Control Conference*, pages 3571–3578, 2014.

[18] Sumit Gulwani and Ashish Tiwari. Constraint-based approach for analysis of hybrid systems. In *Computer Aided Verification*, pages 190–203. Springer, 2008.

[19] Michael Huth. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[20] Luc Jaulin, Michel Kieffer, Olivier Didrit, and Éric Walter. *Applied Interval Analysis*. Springer, 2001.

[21] Luc Jaulin and Éric Walter. Guaranteed tuning, with application to robust control and motion planning. *Automatica*, 32(8):1217–1221, 1996.

[22] James Kapinski, Jyotirmoy V. Deshmukh, Sriram Sankaranarayanan, and Nikos Arechiga. Simulation-guided lyapunov analysis for hybrid dynamical systems. In *Conference on Hybrid Systems: Computation and Control*, pages 133–142. ACM, 2014.

[23] Antonis Papachristodoulou and Stephen Prajna. On the construction of Lyapunov functions using the sum of squares decomposition. In *Conference on Decision and Control*, volume 3, pages 3482–3487, 2002.

[24] Antonis Papachristodoulou and Stephen Prajna. Analysis of non-polynomial systems using the SoS decomposition. In *Positive Polynomials in Control*, pages 23–43. Springer, 2005.

[25] Pablo A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*, 96(2):293–320, 2003.

[26] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In *TABLEAUX*, volume 4548 of *LNCS*, pages 216–232. Springer, 2007.

[27] André Platzer. *Logic analysis of hybrid systems*. Springer-Verlag, 2010.

[28] Stephen Prajna. *Optimization-Based Methods for Nonlinear and Hybrid Systems Verification*. PhD thesis, California Institute of Technology, Pasadena, California, 2005.

[29] Stephen Prajna. Barrier certificates for nonlinear model validation. *Automatica*, 42(1):117–126, 2006.

[30] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.

[31] Stephen Prajna and Anders Rantzer. Primal–dual tests for safety and reachability. In *Hybrid Systems: Control and Computation*, volume 3414 of *LNCS*, pages 542–556. Springer-Verlag, 2005.

[32] Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *Transaction on Computational Logic*, 7(4):723–748, 2006.

[33] Stefan Ratschan. Rsolver user manuel. http://rsolver.sourceforge.net, 2007.

[34] Stefan Ratschan and Zhikun She. Providing a basin of attraction to a target region by computation of Lyapunov-like functions. In *Conference on Computational Cybernetics*, pages 1–5, 2006.

[35] Rachid Rebiha, Nadir Matringe, and Arnaldo Vieira Moura. Transcendental inductive invariants generation for non-linear differential and hybrid systems. In *Conference on Hybrid Systems: Computation and Control*, pages 25–34. ACM, 2012.

[36] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constructing invariants for hybrid systems. In *Hybrid Systems: Computation and Control*, pages 539–554. Springer, 2004.

[37] Christoffer Sloth, George J. Pappas, and Rafael Wisniewski. Compositional safety analysis using barrier certificates. In *Hybrid Systems: Computation and Control*, pages 15–24. ACM, 2012.

[38] Zhendong Sun, S. S. Ge, and T. H. Lee. Controllability and reachability criteria for switched linear systems. *Automatica*, 38(5):775–786, 2002.

[39] Ashish Tiwari. Approximate reachability for linear systems. In *Proc. Hybrid Systems: Computation and Control*, pages 514–525. Springer, 2003.

[40] Antonìn Vaněček and Sergej Čelikovskỳ. *Control systems: from linear analysis to synthesis of chaos*. Prentice Hall, 1996.

[41] Zhengfeng Yang, Wang Lin, and Min Wu. Exact safety verification of hybrid systems based on bilinear SoS representation. *Transaction on Embedded Computing Systems*, 14(1), 2015.