Constraint-Based Framework for Reasoning with Differential Equations

Julien Alexandre dit Sandretto, Alexandre Chapoutot, and Olivier Mullier

Abstract An extension of constraint satisfaction problems with differential equations is proposed. Reasoning with differential equations is mandatory to analyze or verify dynamical systems, such as cyber-physical ones. A constraint-based framework is presented to model a wider class of problems based on logical combination of high-level properties. In addition, the complete correctness is verified using a set-membership approach in this framework. Finally, examples are given to demonstrate the benefits of the presented framework.

1 Introduction

In various domains, such as robotic, control theory, or biology, mathematical models based on differential equations are used to represent the temporal behavior of a particular system. Among the many classes of differential equations, this chapter is interested in *ordinary differential equations* (ODEs) and *differential algebraic equations* (DAEs) which are widely used in these domains. These models are then used for different purposes such as parameter identification, control synthesis, or safety verification. For example, interesting problems involving differential equations are:

- *Control synthesis problem*: A motion planning algorithm for a mobile robot R aims at finding a trajectory of R going from point $\mathbf{a}$ to a point $\mathbf{b}$ of the state space while avoiding an obstacle $\mathbf{o}$. Note that as the movement of R depends on actuators (e.g., engine speed), finding a trajectory is translated into finding control inputs such that R can reach $\mathbf{b}$.
- *Parameters identification problem*: Mathematical models are an approximation of the real word. To make them more faithful, usually an identification step is necessary. Starting from a list of n measures $\mathbf{m}_{i\in\{1,\dots,n\}}$ of the behavior of the real

J. Alexandre dit Sandretto (✉) · A. Chapoutot · O. Mullier
ENSTA ParisTech, Université Paris-Saclay, Palaiseau Cedex, France
e-mail: alexandre@ensta.fr; chapoutot@ensta.fr; mullier@ensta.fr

Ç. K. Koç (ed.), *Cyber-Physical Systems Security*,
https://doi.org/10.1007/978-3-319-98935-8_2

system S and a parametric mathematical model $M(\mathbf{p})$ of the temporal behaviors of S, the goal of the identification step is to find values $\hat{\mathbf{p}}$ of $\mathbf{p}$ such that $M(\hat{\mathbf{p}})$ fits the list of measures $\mathbf{m}_{i\in\{1,\dots,n\}}$.
- *Design problems*: It is closed to the parametric identification problem. The input data are a parametric model $M(\mathbf{p})$ of a system S and a list of n design specifications $P_{i\in\{1,\dots,n\}}$, for example, a car shall reach a speed of 100km/h from 0km/h in less than 6 s on flat dry road. The goal of the design problem is to find values $\hat{\mathbf{p}}$ of $\mathbf{p}$ such that $M(\hat{\mathbf{p}})$ respects all the specifications $P_{i\in\{1,\dots,n\}}$.

Defining automatic methods to solve such kind of problems is challenging. In this context, many attempts have been started based on constraint verification of differential systems [6, 13, 18, 22]. Indeed, the framework of constraint satisfaction problems is an appealing one to express such kinds of problems associated with efficient solving methods producing rigorous results.

We mainly focus on critical problems, coming from aeronautics, robotics, or medical fields. Handling problems in these fields implies to consider the uncertainties in presence using validated methods like interval analysis [24, 28]. We also impose that constraints to solve have to be properly verified. In this context, we use inner approximation to ensure the constraint satisfaction because an enclosure (outer approximation) approach would result in some points that are not solution [24, 28]. The classical approach for these requirements is the use of a Branch-and-Prune algorithm which is a dedicated solver for constraint satisfaction problems (CSP) and the most used in the case of numerical or continuous CSP [31, 32].

To handle differential equations, abstraction based on validated simulation or reachability is a common approach [10, 28–30]. This abstraction needs to be deeply studied to preserve the correctness of a constraint-based problem-solving.

In this chapter, we expand a framework for Constraint Satisfaction Differential Problems (CSDP) with the requirement of preserving the guarantee of the result while dealing with differential constraints. The main contributions are:

- A clearer definition of a CSP framework based on set-membership operations including differential equations ODEs or DAEs compared to previous work [6, 13, 18], namely, SCSDP for *Set-Based Constraint Satisfaction Differential Problems*. In particular, a better handling of quantified constraints is presented, and a better separation between mathematical model and solving algorithm is defined.
- A sound solving algorithm of SCSDP based on interval analysis and guaranteed integration methods is presented as well as a complete study of the impact of the representation of sets by boxes on the guarantee of the solution of SCSDP.
- A set of contractor operators on the solution of differential equations is defined to make Branch-and-Contract solver more efficient for SCSDP.

The chapter is organized as follows. In Sect. 2, the basics of numerical constraint satisfaction problem are provided. The mathematical formulation of CSCP is defined in Sect. 3, while solving algorithm is presented in Sect. 4. Some examples are given in Sect. 5 before concluding in Sect. 6.

Notations
Small italic letters x represent real variables, while real vectors $\mathbf{x}$ are in bold. The set of real numbers is denoted by $\mathbb{R}$. Intervals $[x]$ and interval vectors (boxes) $[\mathbf{x}]$ are represented between brackets. We denote by $\mathbb{IR}$ the set of closed intervals over $\mathbb{R}$. Sets $\mathcal{S}$ are in uppercase calligraphic. The powerset of a set $\mathcal{X}$ is denoted by $\wp(\mathcal{X})$. The derivative of a function x with respect to time t is denoted by $\dot{x}$. Uppercase typewriter letters stand for algorithmic data structures such as a stack `S` or a queue `Q`.

2 Preliminary Notions

A brief presentation of the *constraint satisfaction problem* framework is given in Sect. 2.1 to better understand how it is then extended in the rest of the chapter. A generic solving method based on Branch-and-Prune algorithm is presented in Sect. 2.2

2.1 Numerical Constraint Satisfaction Problems

In this section, we recall the numerical constraint satisfaction problem (NCSP) formalism, following the description given in [32], and present some basics on constraint programming. The approach of NSCP is both powerful to address complex problems (NP-hard problem with numerical issues, even in critical applications) and simple in the definition of a solving framework [1, 26].

A NCSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ is defined as follows:

- $\mathcal{V} := \{v_1, \ldots, v_n\}$ is a finite set of variables which can also be represented by the vector $\mathbf{v}$.
- $\mathcal{D} := \{[v_1], \ldots, [v_n]\}$ is a set of intervals such that $[v_i]$ contains all possible values of v_i. It can be represented by a box $[\mathbf{v}]$ gathering all $[v_i]$.
- $\mathcal{C} := \{c_1, \ldots, c_m\}$ is a set of constraints of the form $c_i(\mathbf{v}) \equiv \mathbf{g}_i(\mathbf{v}) = 0$ or $c_i(\mathbf{v}) \equiv \mathbf{g}_i(\mathbf{v}) \leqslant 0$, with nonlinear $\mathbf{g}_i : \mathbb{R}^n \to \mathbb{R}$ for $1 \leqslant i \leqslant m$. Constraints $\mathcal{C}$ are interpreted as a conjunction of equalities and inequalities, i.e., $\mathcal{C} \equiv c_1 \wedge c_2 \wedge \cdots \wedge c_m$.

The solution of a NCSP is a valuation of $\mathbf{v}$ ranging in $[\mathbf{v}]$ and satisfying the constraints $\mathcal{C}$.

2.2 Branch-and-Contract Solving Method

The classical algorithm to solve a NCSP, as previously defined, is the *Branch-and-Prune* method which needs only an interval evaluation of the constraints and an

initial domain for variables. More precisely, an interval $[\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leqslant x \leqslant \overline{x}\} \in \mathbb{IR}$ is defined by its lower and upper bounds $\underline{x}$ and $\overline{x}$ as a compact set of $\mathbb{R}$. An interval vector (or *box*) $[\mathbf{x}]$ of dimension n is a Cartesian product of intervals $[\underline{x}_0, \overline{x}_0] \times \cdots \times [\underline{x}_n, \overline{x}_n]$. An inclusion function $[f] : \mathbb{IR}^n \to \mathbb{IR}$ for $f : \mathbb{R}^n \to \mathbb{R}$ satisfies

$$\forall [\mathbf{x}] \in \mathbb{IR}\{f(\mathbf{x}) | \mathbf{x} \in [\mathbf{x}]\} \subseteq [f]([\mathbf{x}]). \tag{1}$$

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 leads in general to some overapproximation; see [24] for more details.

A more elaborated solving method named *Branch-and-Contract* is usually applied to accelerate the solving process of an NCSP. A generic version, using interval analysis, of this algorithm is given in Algorithm 1.

Algorithm 1 A generic Branch-and-Contract

Require: $\mathbf{v}$, $[\mathbf{v}]$, $\mathscr{C}_{\text{acc}}$, $\mathscr{C}_{\text{rej}}$
Ensure: $\mathbb{S}_{\text{acc}}$, $\mathbb{S}_{\text{rej}}$, $\mathbb{S}_{\text{unc}}$
1: $\mathbb{S} = \{[\mathbf{v}]\}$
2: $\mathbb{S}_{\text{acc}} = \emptyset$, $\mathbb{S}_{\text{rej}} = \emptyset$, $\mathbb{S}_{\text{unc}} = \emptyset$
3: **while** $\mathbb{S} \neq \emptyset$ **do**
4: Pop a $[\mathbf{v}]_{\text{current}}$ from $\mathbb{S}$
5: $[\mathbf{v}]_{\text{current}} = \text{Contract}\ (\mathscr{C}_{\text{acc}}, ([\mathbf{v}]_{\text{current}}))$ ▷ May be omitted to get branch-and-prune method
6: **if** Check($\mathbf{v}$,$[\mathbf{v}]_{\text{current}}$,$\mathscr{C}_{\text{acc}}$) **then** ▷ Satisfiability check
7: Push $[\mathbf{v}]_{\text{current}}$ in $\mathbb{S}_{\text{acc}}$
8: **else if** Check($\mathbf{v}$,$[\mathbf{v}]_{\text{current}}$,$\mathscr{C}_{\text{rej}}$) **then** ▷ Unsatisfiability check
9: Push $[\mathbf{v}]_{\text{current}}$ in $\mathbb{S}_{\text{rej}}$
10: **else if** Width($[\mathbf{v}]_{\text{current}}$) $> \varepsilon$ **then**
11: $([\mathbf{v}]_{\text{left}}, [\mathbf{v}]_{\text{right}}) = \text{Bisect}([\mathbf{v}]_{\text{current}})$ ▷ Spliting method
12: Push $[\mathbf{v}]_{\text{left}}$ in $\mathbb{S}$
13: Push $[\mathbf{v}]_{\text{right}}$ in $\mathbb{S}$
14: **else**
15: Push $[\mathbf{v}]_{\text{current}}$ in $\mathbb{S}_{\text{unc}}$
16: **end if**
17: **end while**

The key feature of Algorithm 1 is the function Check(),

$$\text{Check}() : \mathscr{V} \times \mathscr{D} \times \mathscr{C} \to \mathbb{B}$$

with $\mathbb{B} = \{\text{True}, \text{False}\}$. Check() is then a *decision procedure* which is able to verify if constraints $\mathscr{C}$ are satisfied by all the values of the domain $\mathscr{D}$. Note that due to pessimism of interval analysis approach, it may be not possible to decide if $\mathscr{C}$ is satisfied or not. In this case, False has to be interpreted to "undecidable."

Two kinds of constraints are considered in Algorithm 1, one with constraints $\mathscr{C}_{\text{acc}}$ and one with constraints $\mathscr{C}_{\text{rej}}$. The first set of constraints is used to accept the solutions, while the second is used to reject the nonsolutions, i.e., the points guaranteed to be outside the solution domain. It is optional but useful to speed up the algorithm by quickly eliminating unfeasible domain. $\mathscr{C}_{\text{rej}}$ is in a certain point of view the negation of $\mathscr{C}_{\text{acc}}$, but in general, $\mathscr{C}_{\text{acc}} \neq \neg\mathscr{C}_{\text{rej}}$, due to the abstraction of continuous domains (and the issue of disjunction). Note that this approach follows classical method in SIVIA (Set Inversion Via Interval Analysis) method [23].

The second key feature in *Branch-and-Contract* algorithm is the Contract() procedure which simultaneously reduces the domain studied ($[\mathbf{v}]_{\text{current}}$ in the algorithm) by the help of contractors [3]. A contractor associated to a constraint $c \equiv g(\mathbf{x}) \diamond 0$ with $\diamond \in \{=, \leqslant\}$ is a function C_c taking a box $[\mathbf{x}]$ as parameter and returns a box such that

$$C_c([\mathbf{x}]) \subseteq [\mathbf{x}] \qquad \text{(Reduction)} \qquad (2a)$$

$$g([\mathbf{x}]) \cap [z] = g(C_c([\mathbf{x}])) \cap [z] \qquad \text{(Correction)} \qquad (2b)$$

where $[z] = [0, 0]$ if $\diamond \equiv =$ and $[z] = [-\infty, 0]$ if $\diamond \equiv \leqslant$. The main strength of contractors is that they can reduce the domain $[\mathbf{x}]$ while preserving solution without using bisection, and so they can reduce, in practice, the algorithmic complexity of Algorithm 1. For more details on contractors, see [3].

The result of Algorithm 1, also known as *a paving*, is made of three lists of boxes $\mathbb{S}_{\text{acc}}$, $\mathbb{S}_{\text{rej}}$, and $\mathbb{S}_{\text{unc}}$ such that

- There is no solution of the NCSP in $\mathbb{S}_{\text{rej}}$.
- All the solutions of the NCSP, included in the initial domain, are in $\mathbb{S}_{\text{acc}} \cup \mathbb{S}_{\text{unc}}$.
- All the values in $\mathbb{S}_{\text{acc}}$ are solution of the NCSP.

Example 1 The result of a Branch-and-Contract method on the constraints $1 \leqslant x^2 + y^2 \leqslant 2$ with $(x, y) \in [-2, 2] \times [-2, 2]$ is given in Fig. 1. In this figure, blue boxes are elements of $\mathbb{S}_{\text{acc}}$, red boxes are in $\mathbb{S}_{\text{rej}}$, and white boxes (between red and blue boxes) are in $\mathbb{S}_{\text{unc}}$. ■

2.3 Some Limitations on NCSP

2.3.1 Equality Constraints

One of the main difficulties in NCSP approach is the handling of equality constraint, i.e., $\mathbf{g}(\mathbf{v}) = 0$. Indeed, a classical solving approach for NCSP is based on interval analysis which considers computations over boxes instead of points. So proving equality constraints usually involves some relaxation techniques such as proving a simpler constraint of the form $\mathbf{g}(\mathbf{v}) \in [-\varepsilon, \varepsilon]$ for a small positive value ε. Moreover specific algorithms have to be used to prove the existence and uniqueness of the

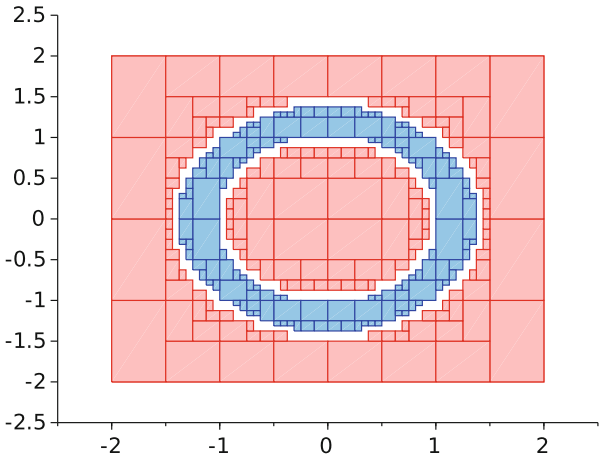

Fig. 1 Paving of a circle

solution of $\mathbf{g}(\mathbf{v}) = 0$ such as the interval Newton operator [24]. In consequence, solving equality constraints is often dependent on the solving algorithm as the relaxation is generally done internally in the solver. The aim of our work is to avoid this implementation trick and push out the relaxation choice to the designer by only allowing set-based constraints as inclusion constraint; see Sect. 3.

2.3.2 Differential Constraints

The framework of NCSP lacks expressiveness when dealing with differential equation. In [6], a first approach was given by introducing *Constraint Satisfaction Differential Problems* (CSDP). Basically, new variables are added to the set of variables of NCSP to represent time derivative, and a new type of constraints is added too to represent the dynamic of the differential system. The time variable being handled separately from the other variables, temporal properties, cannot be encoded with CSDP. For example, if a trajectory described by a differential system has to avoid an obstacle in a given time interval, modeling this using CSDP cannot be done in an obvious manner.

Another work in [18] also dealt with this problem. The dynamical system is abstracted with a solution operator ϕ representing the solution of the system. Differential equations are then naturally embedded in the NCSP framework. Its limitation is also this abstraction because constraints on the dynamical system cannot be easily expressed.

In these previous works, another drawback is the lack of quantification on variables. Bringing a solution to this is one of the motivations of this work.

3 Set-Based Constraint Satisfaction Differential Problems

The proposed extension of NSCP is based on set-based constraints and the embedding of differential constraints. These extensions allow to increase the class of problems which can be modeled and solved.

3.1 Dynamical Systems

In the rest of this article, a general class of differential equations is considered which can represent ordinary differential equations (ODEs), differential algebraic equations (DAEs) of index 1, and a mix of these equations with additional constraints, e.g., to model energy preservation. More precisely, differential systems of the form

$$\begin{cases} \dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}(t), \mathbf{x}(t), \mathbf{p}), \\ \quad 0 = \mathbf{g}(t, \mathbf{y}(t), \mathbf{x}(t)) \\ \quad 0 = \mathbf{h}(\mathbf{y}(t), \mathbf{x}(t)) \end{cases} . \tag{3}$$

with nonlinear functions $\mathbf{f} : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}^n$, $\mathbf{g} : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$, $\mathbf{h} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $t \in [0, t_{\text{end}}]$, $\mathbf{y}(0) \in \mathscr{Y}_0$, and $\mathbf{p} \in \mathscr{P}$ are considered. More precisely, initial value problems (IVP) for parametrized differential equations are considered over a finite time horizon $[0, t_{\text{end}}]$. Note that a bounded set of initial values and a bounded set of parameters are considered in this framework. This implies to deal with set of trajectories solution of Eq. (3). We assume classical hypothesis on **f**, **q**, and **h** to ensure the existence and uniqueness of the solution of Eq. (3).

In the rest of this section, we denote by $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ the set

$$\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P}) = \{\mathbf{y}(t; \mathbf{y}_0, \mathbf{p}) : t \in \mathscr{T}, \mathbf{y}_0 \in \mathscr{Y}_0, \mathbf{p} \in \mathscr{P}\} . \tag{4}$$

Intuitively, $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ gathers all the points reached by the solution $\mathbf{y}(t; \mathbf{y}_0, \mathbf{p})$ of Eq. (3) starting from all scalar initial values $\mathbf{y}_0$ and all scalar parameters **p**. Note that $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ is hardly computable in general, and the implementation issue is addressed in Sect. 4. Note also that the difference of the proposed approach comparing to [18] is that we consider a set-based solution operator which offers a convenient way to deal with quantification over variables.

The purpose of the proposed framework is to check if $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ fulfills some specification defined in terms of set-based constraints.

3.2 Set-Based Constraints

To avoid problematic issue due to equality constraints (see Sect. 2.3), set-based constraints are considered. More precisely, *inclusion* and *disjunction* operators are considered. The proposed framework will consider constraints of the form

$$\mathbf{g}(\mathscr{A}) \subseteq \mathscr{B} \tag{5a}$$

$$\mathbf{g}(\mathscr{A}) \supseteq \mathscr{B} \tag{5b}$$

$$\mathbf{g}(\mathscr{A}) \cap \mathscr{B} = \emptyset \tag{5c}$$

$$\mathbf{g}(\mathscr{A}) \cap \mathscr{B} \neq \emptyset \tag{5d}$$

where $\mathscr{A}$ and $\mathscr{B}$ are real compact sets and $\mathbf{g}$ is a nonlinear function. The lifting of $\mathbf{g}$ over sets is defined as usual by $\mathbf{g}(\mathscr{X}) = \{\mathbf{g}(x) : x \in \mathscr{X}\}$.

Note that these constraints can be seen as Boolean functions, but while, from a mathematical formulation, the truth value can always be obtained, it may not be the case when they have to be solved on a computer. The safe computer resolution of these kinds of constraints is one of the main contributions of this article, detailed in Sect. 4.

3.3 Set-Based Differential Constraint Satisfaction Problems

The handling of differential constraints here follows the approach given in [18] in the exception of the solution operator of Eq. (3), which is here represented as a set of solution $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ in order to unify the objects manipulated into constraints which are also sets.

Set-Based Constraint Satisfaction Differential Problems (SCSDP) based on a set-membership constraints and embedding differential constraints can now be defined.

Definition 1 (SCSDP) A SCSDP is a NCSP made of

- A finite set $\mathscr{S}$ of differential systems S_i as defined in Eq. (3)
- A finite set of variables $\mathscr{V}$ including the parameters of the differential systems S_i, i.e., $(\mathbf{y}_0, \mathbf{p})$, a time variable t and some other algebraic variables $\mathbf{q}$
- A domain $\mathscr{D}$ made of the domain of parameters $\mathbf{p} : \mathscr{D}_p$, of initial values $\mathbf{y}_0 : \mathscr{D}_{y_0}$, of the time horizon $t : \mathscr{D}_t$, and the domains of algebraic variables $\mathscr{D}_q$
- A set of constraints $\mathscr{C}$ which may be defined by inclusion or disjunction constraints (see Sect. 3.2) over variables of $\mathscr{V}$ and special variables $\mathscr{Y}_i(\mathscr{D}_t, \mathscr{D}_{y_0}, \mathscr{D}_p)$ representing the set of the solution of S_i in $\mathscr{S}$

Example 2 A cruise control system based on PI controller for a nonlinear dynamic of a car is considered [12]. The dynamic of the car is defined by

$$S \equiv \begin{cases} \dot{v} = \dfrac{k_p(v_{\text{set}} - v) + k_i s - 50.0v - 0.4v^2}{m} \\ \dot{s} = v_{\text{set}} - v \end{cases} . \tag{6}$$

with v the speed of the vehicle and s the integral part of the PI controller, k_p and k_i the parameters of the PI controller, $m \in [990, 1010]$ the mass of the vehicle, and $v_{\text{set}} = 10$ the target speed of the car from initial conditions $v(0) = 0$ and $s(0) = 0$. The term $k_p(v_{\text{set}} - v) + k_i s$ is the PI controller, $-50.0v$ is the resisting force due to the road, and $-0.4v^2$ is the aerodynamic friction.

The specification of the PI controller is such that it should stabilize in 10 s with a tolerance of 2%, and its overshoot should not be more than 5% of the target speed. These are translated into constraints such that

$$v(10) \subseteq [9.8, 10.2] \qquad \text{(At t=10, } v_{\text{set}} \pm 2\%\text{)}$$

$$\dot{v}(10) \subseteq [-\varepsilon, \varepsilon] \qquad \text{(At t =10, acceleration is around zero, with a small } \varepsilon > 0\text{)}$$

$$v([0, 10]) \subseteq [0, 10.5] \qquad \text{(For } t \in [0, 10], v \text{ should not be above } v_{\text{set}} + 5\%\text{)}$$

Note that in Eq. (6), the mass m is uncertain, so the solution of S is a thick function so the use of inclusion constraints. In summary, a SCSDP is defined by

- $\mathscr{S} = \{S \text{ defined in Eq. (6)}\}$
- $\mathscr{V} = \{k_p, k_i\}$
- $\mathscr{D} = \{[1, 4000], [1, 120]\}$
- $\mathscr{C} = \{v(10) \subseteq [9.8, 10.2], \dot{v}(10) \subseteq [-\varepsilon, \varepsilon], v([0, 10]) \subseteq [0, 10.5]\}$

■

Note that following NCSP and its solving algorithm, variables in $\mathscr{V}$ are quantified existentially, and other variables (not in $\mathscr{V}$) are quantified universally, e.g., the mass m in Eq. (6). Hence, there is no need to introduce quantifier in constraints. The proposed SCSDP framework is hence simpler than previous work [6, 13, 18] in embedding quantification constraints while taking into account differential equations. Nevertheless, one important challenge is to solve SCSDP in a guaranteed way, and for this purpose a computable representation of sets has to be defined. As interval analysis [28] brings very efficient techniques over boxes, it is a natural mean to solve SCSDP on computers.

4 Solving SCSDP

To solve a SCSDP as defined in Sect. 3.3, a proper representation of sets has to be defined. Interval analysis provides a simple representation of compact sets by the mean of interval values or boxes. This representation is simple enough to represent complex sets while being associated with fast computational methods. To solve SCSDP on a computer, set-based constraints defined in Sect. 3.2 have to be properly translated into boxes, and dynamical systems as defined in Sect. 3.1 have to be solved.

4.1 Interval-Based Constraints

In order to solve set-based constraints appearing in SCSDP, as defined in Sect. 3.2, an interval-based abstraction is given in this section. As shown in Sect. 2.1, complex compact sets can be represented by paving and so can be represented either by inner approximation (boxes in $\mathbb{S}_{\text{acc}}$) or outer approximation (boxes in $\mathbb{S}_{\text{acc}} \cup \mathbb{S}_{\text{unc}}$). In consequence, translating constraints defined in Eq. (5) to interval-based constraints, a proper representation of sets has to be defined. In particular, the validity of the translation is important to guarantee the result of solving a SCSDP on a computer.

In the sequel, $\text{Int}\mathscr{X}$ will stand for the interior of the compact set $\mathscr{X}$, while $\text{Hull}\mathscr{X}$ will stand for the outer approximation of $\mathscr{X}$. In each case, the inner approximation or the outer approximation can be defined by a box or a list of boxes. Note that the outer approximation of a nonlinear function g in interval arithmetic is given by inclusion function as defined in Eq. (1), and more information can be found in [24], while inner approximation of g requires special treatments as defined in [16, 17, 19, 21]. Note also that excepting a complete computation of inner approximation or outer approximation, there is no meaning to consider outer approximation of g with inner approximation of its parameter and reciprocally.

Table 1 Set-based constraint evaluation in interval analysis framework

			$\mathscr{A}$	
			$\text{Int}\,\mathscr{A}$	$\text{Hull}\,\mathscr{A}$
$\mathbf{g}(\mathscr{X})$	$\text{Hull}\,\mathbf{g}(\text{Hull}\,\mathscr{X})$	$\subseteq$	`true`	?
		$\supseteq$	`false`	?
		$\cap_{=\emptyset}$	?	`true`
		$\cap_{\neq\emptyset}$	?	`false`
	$\text{Int}\,\mathbf{g}(\text{Int}\,\mathscr{X})$	$\subseteq$	?	`false`
		$\supseteq$	?	`true`
		$\cap_{=\emptyset}$	`false`	?
		$\cap_{\neq\emptyset}$	`true`	?

In Table 1, a summary of the translation of constraints defined in Eq. (5) into an interval counterpart is given. For each case, inner approximation or outer approximation, the truth value of the constraints is inspected. When a value is `true`, that is, the constraint can be proved to be true, and when a value is `false`, then the constraint can be proved to be false. Otherwise, no conclusion can be made on the constraint and it is denoted by "?." For example, a proof of a constraint of the form $g(\mathscr{X}) \subseteq \mathscr{A}$ can be obtained only by considering an outer approximation of g and its parameter $\mathscr{X}$ and an inner approximation of $\mathscr{A}$. As a second example, a proof of unsatisfiability of the constraints $g(\mathscr{X}) \subseteq \mathscr{A}$ can be obtained considering an inner approximation of g and its parameter $\mathscr{X}$ and an outer approximation of $\mathscr{A}$.

As it is clear from Table 1, an interval counterpart of constraints defined in Eq. (5) is not so obvious in order to guarantee the result of a SCSDP.

4.2 *Interval-Based Differential Constraints*

As for the interval representation of compact sets which can be from an inner approximation or an outer approximation, dynamical systems can be solved to produce interval-based representation containing all the trajectories or a subset of the set of trajectories. In other terms, $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ can be inner-approximated or outer-approximated. A short review of these methods is given in the rest of this section as $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ can appear in constraints of SCSDP.

4.2.1 Outer Approximation of Differential Constraints

The computation of $\mathscr{Y}(\mathscr{T}, \mathscr{Y}_0, \mathscr{P})$ solution of a system S described in Eq. (4) has been studied for a long time with interval analysis methods. A complete approach named *guaranteed numerical integration* has been defined from the seminal work of Moore [28]. More precisely, initial value problems of ordinary differential equations have been solved with interval analysis mainly based on Taylor series [4, 27–29] and more recently with Runge–Kutta-based methods [2, 10, 15]. Initial value problems for algebraic differential equations have been studied in [11, 30]. All the works aim at producing an outer approximation of the solution of the dynamical systems using interval analysis tools.

The goal of a guaranteed numerical integration method to solve Eq. (3) is to compute a sequence of time instants $0 = t_0 < t_1 < \cdots < t_n = t_{end}$ and a sequence of boxes $[\mathbf{y}_0], \ldots, [\mathbf{y}_n]$ such that $\forall j \in [0, n], [\mathbf{y}_{j+1}] \supseteq \mathbf{y}(t_j; [\mathbf{y}_j], [p])$. In this article, we focus on single-step methods that only use $[\mathbf{y}_j]$ and approximations of $\dot{\mathbf{y}}(t)$ to compute $[\mathbf{y}_{j+1}]$.

The main approach in a guaranteed numerical integration method, as presented in [29], is that each step of a validated integration scheme consists of two phases

Phase 1 One computes an a priori enclosure $[\tilde{\mathbf{y}}_j]$ of the solution such that

- $\mathbf{y}(t;[\mathbf{y}_j])$ is guaranteed to exist for all $t \in [t_j, t_{j+1}]$, i.e. along the current step, and for all $\mathbf{y}_j \in [\mathbf{y}_j]$.
- $\mathbf{y}(t;[\mathbf{y}_j]) \subseteq [\tilde{\mathbf{y}}_j]$ for all $t \in [t_j, t_{j+1}]$.
- the step-size $h_j = t_{j+1} - t_j > 0$ is as large as possible in terms of accuracy and existence proof for the *IVP* solution.

Phase 2 One computes a tighter enclosure of $[\mathbf{y}_{j+1}]$ at time t_{j+1} such that $\mathbf{y}(t_{j+1}, [\mathbf{y}_j]) \subseteq [\mathbf{y}_{j+1}]$.

A guaranteed numerical integration for a system S, as defined in Eq. (3), starts with an outer approximation of initial condition $\text{Hull}\mathscr{U}_0 = [\mathbf{y}_0]$, the parameters $\text{Hull}\mathscr{P} = [\mathbf{p}]$, and an integration step size h (or a finite horizon). It applies the two-step approach until the end of the simulation time is reached. This process builds two lists of boxes:

- The list of discretization time steps: $\{[\mathbf{y}_0], \ldots, [\mathbf{y}_{\text{end}}]\}$
- The list of a priori enclosures: $\{[\tilde{\mathbf{y}}_0], \ldots, [\tilde{\mathbf{y}}_{\text{end}}]\}$

Based on these lists, two functions depending on time can be defined

$$R : \begin{cases} \mathbb{R} \mapsto \mathbb{IR}^n \\ t \to [\mathbf{y}] \end{cases} \tag{7}$$

with $\{\mathbf{y}(t;\mathbf{y}_0) : \forall \mathbf{y}_0 \subseteq [\mathbf{y}_0]\} \subseteq [\mathbf{y}]$ and

$$\widetilde{R} : \begin{cases} \mathbb{IR} \mapsto \mathbb{IR}^n \\ [\underline{t}, \overline{t}] \to [\tilde{\mathbf{y}}] \end{cases} \tag{8}$$

with $\{\mathbf{y}(t;\mathbf{y}_0) : \forall \mathbf{y}_0 \in [\mathbf{y}_0] \wedge \forall t \in [\underline{t}, \overline{t}]\} \subseteq [\tilde{\mathbf{y}}]$.

Function R, defined in (7), is obtained by new applications of validated integration method starting from $[\mathbf{y}_k]$ at t_k and finishing at t with $t_k < t < t_{k+1}$. Function $\widetilde{R}$, defined in (8), is obtained with the union of $[\tilde{\mathbf{y}}_k]$ with $k = a, \ldots, b$ and $t_a < \underline{t} < \overline{t} < t_b$. These functions are then strictly conservative.

More abstractly, the functions R and $\widetilde{R}$ define two interval enclosures of the solution function of differential equations defined in Eq. (3).

4.2.2 Inner Approximation of Differential Constraints

More recent work deals with the inner approximation of the reachable sets of ordinary differential equations such as [5, 20]. But a lot of work remains to be done to elevate this technique to a maturity level of guaranteed numerical integration as defined in Sect. 4.2.1. A short review of [20] is made in this section as it closely follows the two-step approach of guaranteed numerical integration.

In [20] a new approach for computing inner approximations of reachable sets of dynamical systems defined by nonlinear, uncertain, ordinary differential equations is defined. It extends [19, 21] which focused on discrete-time dynamical systems. It consists in using generalized affine forms combining modal interval analysis [25] (an extension of interval arithmetic dealing with quantifiers) with affine arithmetic [8] (an extension of interval arithmetic which can take into account some correlation between variables) to produce both inner and outer approximations of the flow of an uncertain ODE with a Taylor series approach.

The given algorithm consists of three steps: (1) computing rough enclosures over a time interval $[t_i, t_{i+1}]$ of the solution and its Jacobian over the initial conditions (which is the solution of the variational equation), (2) building the Taylor models of the solution and its Jacobian, and (3) computing the inner approximations of the flow pipe using generalized affine forms.

4.3 Revisiting Branch-and-Contract Solving Method

After defining a correct interval representation of compact sets in Sect. 4, a focus on the application of Branch-and-Contract algorithm to solve SCSDP is given. More precisely, as differential constraints imply set of trajectories, an extension of contractors to deal with this new object has to be defined.

In our approach based on interval analysis and contractor programming [3], an application of constraints at some given instants in the set of trajectories and a propagation can be performed on the interval representation of the trajectories. In the rest of this section, only outer approximation of dynamical systems is considered. This section defines the two methods, contraction and propagation, on set of trajectories.

4.3.1 Contraction

The considered approach allows one to contract a specific value $[\mathbf{y}_*] = R(t^*)$, an outer approximation of the solution of the IVP at time t^* such that $\mathbf{y}(t^*) \in [\mathbf{y}_*]$ with respect to a constraint g.

The simplest example is as follows: considering a system defined by $\dot{y} = f(y)$ and $y(0) = y_0$, if a set of measures $\{\mathbf{y}_1^*, \ldots, \mathbf{y}_m^*\}$ are taken at some specific instants $t_1^*, \ldots, t_m^*$, then a contraction can be applied following the rule $[\mathbf{y}(t_i^*)] = [\mathbf{y}(t_i^*)] \cap [y_i^*], \forall i = 1, \ldots, m$. In a more complex example, if the states of the system are constrained by the help of a function g, then a contractor such as HC4-Revise or interval Newton [24] can be used. For example, a constraint such as $y(t^*)^2 - 3\cos(y(t^*)) \subset [-\infty, 0]$ can be also considered.

Remark 1 If t^* is not in the already computed time steps, then the contraction procedure adds a kth integration step to the time discretization: $\{[y_0], \ldots, [y_i], [y_k], [y_{i+1}], \ldots, [y_N]\}$ such that $t_k = t^*$.

Remark 2 The contraction at a time t^* can be easily generalized to a contraction along an interval of instants $[\underline{t}^*, \overline{t}^*]$, by the help of the $\tilde{R}$ function and a priori enclosures $[\mathbf{y}_*]$.

4.3.2 Propagation

If a contraction has been obtained, then a Picard contractor [11] on $[\tilde{\mathbf{y}}_i]$ and a validated Runge–Kutta contractor [11] on $[\mathbf{y}_i]$ can be applied on each integration step i, in order to propagate this information on the whole simulation, i.e., on all the boxes in the lists:

- Forward for $t > t^*$ with the considered differential equation
- Backward for $t < t^*$ with the inverse of the considered differential equation

A fixed-point algorithm (a loop calling alternatively the forward and the backward steps till not enough improvement is obtained with respect to a given threshold) can be also used.

Example 3 Van der Pol system is considered and it is defined by

$$\dot{x} = y$$
$$\dot{y} = 2.0(1.0 - x^2)y - x$$

with the initial conditions $x(0) \in [2.0, 2.2]$ and $y(0) \in [0.0, 0.1]$, simulated from $t = 0$ to $t = 2.0$ (see Fig. 2). A measure is obtained at $t = 1.0$, such that $y(1.0) \in [1.58, 1.62]$ and $x(1.0) \in [-0.74, -0.69]$. A contraction and a forward propagation are applied; then a backward and finally a fixed point are applied (see Fig. 2). ■

5 Numerical Example

DynIBEX library [9] implements the outer-approximation version of the proposed CSDP framework. This library allows to solve complex constraint satisfaction problems mixing bounded uncertainties, variable quantification, and differential constraints.

Kinetic parameter estimation of an enzymatic reaction example has already been considered in [18]. It aims at illustrating the SCSDP framework described in this chapter. The goal is to obtain the kinetic parameters of an enzymatic reaction as

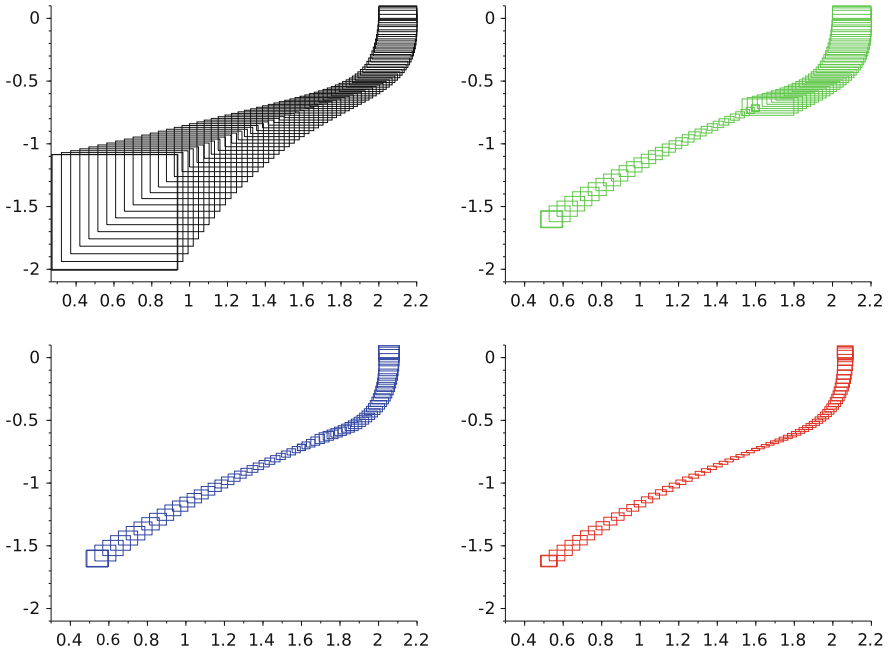

Fig. 2 Van Der Pol problem: initial (up, left), after contraction and forward propagation (up, right), after backward propagation (down, left), and after fixed-point mixing forward and backward propagation (down, right)

described in [14]. The differential equation is as follows:

$$(\mathscr{S}) \begin{cases} \dot{s}(t) = -\frac{V_{\max} s(t)}{k_s + s(t)} \\ \dot{p}(t) = \frac{V_{\max} s(t)}{k_s + s(t)} \end{cases} \tag{9}$$

with $p(t)$ and $s(t)$ the two concentrations and $V_{\max}$ and k_s the two parameters to infer from a series of measures on the concentration p during time. These measurements are shown in Table 2.

The corresponding SCSDP is as follows:

- $\mathscr{S}$ from Eq. (9)
- $\mathscr{V} = \{p_0, s_0, V_{\max}, k_s, t\}$
- $\mathscr{D} = \{[25], [0], [90, 110], [0, 10], [0, 1.0]\}$

Table 2 Measurements for the enzymatic reaction (± 0.1)

t_i	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
p_i	8.01	15.32	21.01	23.92	24.74	24.96	24.97	25.02	24.95	24.91

- $\mathcal{C} = \begin{cases} \text{Proj}_p(\mathcal{Y}(t_i, p_0, s_0, V_{\max}, k_s)) \subset p_i & \text{(Measurements)} \\ \mathcal{Y}(t, p_0, s_0, V_{\max}, k_s) \subset [0, +\infty]^2 & \text{(Nonnegative concentrations)} \end{cases}$

with the operator $\text{Proj}_x(\mathcal{Y})$ the projection over component x of the set $\mathcal{Y}$. This problem can be treated in two ways, whether we choose to consider contractors or not as depicted in Algorithm 1 with the addition of line 5 or not. A first resolution scheme is to directly apply a Branch-and-Prune algorithm on the parameters p. The function $Check()$ used to verify constraints $\mathcal{C}$ is as defined in Algorithm 2.

Algorithm 2 Check for the kinetic parameter estimation

Require: $(t_i, p_i)_{i=1,\ldots,10}$, $p_0, s_0, [V_{\max}], [k_s]$)
  bool IsUndecidable = false
  for all j = 1 to 10 **do**
    $[p]_{\text{current}} \leftarrow \text{Proj}_p(\mathcal{Y}(t_j, p_0, s_0, [V_{\max}], [k_s]))$
    $[y]_i \leftarrow \mathcal{Y}([t_{i-1}, t_i], p_0, s_0, V_{\max}, k_s)$
    if $[p]_{\text{current}} \cap p_j = \emptyset$ or $[y]_i \subseteq [0, +\infty] = \emptyset$ **then**
      return false
    end if
    if $[p]_{\text{current}} \not\subseteq p_j$ or $[y]_i \not\subseteq [0, +\infty]$ **then**
      IsUndecidable = true
    end if
  end for
  if IsUndecidable **then**
    return "undecidable"
  else
    return true
  end if

Another way is to consider the contractor described in Sect. 4.3.1. We recall that these contractors only apply to the state space of the solution of $(\mathcal{S})$, so the parameters have to be embedded into the state space. This is done in a classical way as follows:

$$(\mathcal{S}') \begin{cases} \dot{s}(t) = -\frac{V_{\max} s(t)}{k_s + s(t)} \\ \dot{p}(t) = \frac{V_{\max} s(t)}{k_s + s(t)} \\ \dot{V}_{\max} = 0 \\ \dot{k_s}(t) = 0 \end{cases} \tag{10}$$

A contractor can then be defined for the resolution of our problem by contracting and propagating each measurement. The solution of this example is depicted in Fig. 3.

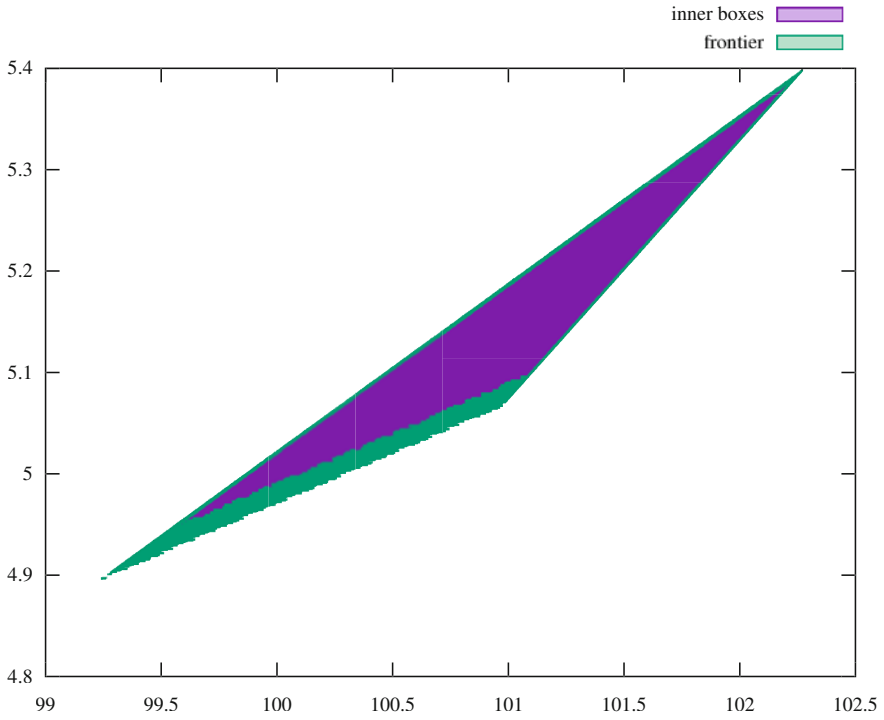

Fig. 3 Solution for the parameter estimation problem for enzymatic reaction

6 Conclusion

A constraint satisfaction problem framework has been extended to deal with differential constraints and quantification on variables. It extends other approaches [7, 18] by dealing more naturally with uncertainties and variable quantification. A discussion on the correctness of the interval representation of compact sets has been given. It emphasizes the problem of preserving the correctness of the approach when computing with tools coming from interval analysis.

A future work and extension of DynIBEX with inner approximation will be useful to address a more important class of problems. From a theoretical point of view, an extension of SCSDP with techniques coming from SMT approaches could be beneficial to increase the expressiveness of the framework, as for example, dealing with disjunctive constraints.

Acknowledgements This research benefited from the support of the "Chair Complex Systems Engineering – Ecole Polytechnique, THALES, DGA, FX, Dassault Aviation, DCNS Research, ENSTA ParisTech, Télécom ParisTech, Fondation ParisTech, and FDO ENSTA," and it is also partially funded by DGA MRIS "Safety for Complex Robotic Systems."

References

1. F. Benhamou, D. McAllester, P. Van Hentenryck, CLP (intervals) revisited. Technical Report, Providence (1994)
2. O. Bouissou, A. Chapoutot, S. Mimram, HySon: precise simulation of hybrid systems with imprecise inputs, in *IEEE Rapid System Prototyping* (2012)
3. G. Chabert, L. Jaulin, Contractor programming. Artif. Intell. **173**(11), 1079–1100 (2009)
4. X. Chen, E. Ábrahám, S. Sankaranarayanan, Flow*: an analyzer for non-linear hybrid systems, in *Proceedings of the International Conference on Computer Aided Verification* (Springer, Berlin, 2013), pp. 258–263
5. X. Chen, S. Sankaranarayanan, E. Ábrahám, Under-approximate flowpipes for non-linear continuous systems, in *Proceedings of the Conference on Formal Methods in Computer-Aided Design* (FMCAD Inc., Austin, 2014), pp. 59–66
6. J. Cruz, P. Barahona, Constraint satisfaction differential problems, in *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 2833 (Springer, Berlin, 2003), pp. 259–273
7. J. Cruz, P. Barahona, Constraint reasoning over differential equations. Appl. Numer. Anal. Comput. Math. **1**(1), 140–154 (2004)
8. L.H. de Figueiredo, J. Stolfi, Self-validated numerical methods and applications. Brazilian Mathematics Colloquium Monographs. IMPA/CNPq, Rio de Janeiro (1997)
9. J.A. dit Sandretto, A. Chapoutot, DynIBEX: a differential constraint library for studying dynamical systems, in *Conference on Hybrid Systems: Computation and Control* (2016). Poster
10. J.A. dit Sandretto, A. Chapoutot, Validated explicit and implicit Runge-Kutta methods. Reliab. Comput. **22**, 56–77 (2016)
11. J.A. dit Sandretto, A. Chapoutot, Validated simulation of differential algebraic equations with Runge-Kutta methods. Reliab. Comput. **22**, 56–77 (2016)
12. J.A. dit Sandretto, A. Chapoutot, O. Mullier, Tuning PI controller in non-linear uncertain closed-loop systems with interval analysis, in *2nd International Workshop on Synthesis of Complex Parameters*, *OpenAccess Series in Informatics*, vol. 44, pp. 91–102. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2015)
13. J.A. dit Sandretto, A. Chapoutot, O. Mullier, Formal verification of robotic behaviors in presence of bounded uncertainties. J. Softw. Eng. Robot. **8**(1), 78–88 (2017)
14. D. Eveillard, D. Ropers, H. De Jong, C. Branlant, A. Bockmayr, A multi-scale constraint programming model of alternative splicing regulation. Theor. Comput. Sci. **325**(1), 3–24 (2004)
15. K. Gajda, M. Jankowska, A. Marciniak, B. Szyszka, A survey of interval Runge–Kutta and multistep methods for solving the IVP, in *Parallel Processing and Applied Mathematics*. Lecture Notes in Computer Science, vol. 4967 (Springer, Berlin, 2008), pp. 1361–1371
16. A. Goldsztejn, W. Hayes, Rigorous inner approximation of the range of functions, in *12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics* (2006), p. 19
17. A. Goldsztejn, L. Jaulin, Inner approximation of the range of vector-valued functions. Reliab. Comput. **14**, 1–23 (2010)
18. A. Goldsztejn, O. Mullier, D. Eveillard, H. Hosobe, Including ODE based constraints in the standard CP framework, in *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 6308 (Springer, Berlin, 2010), pp. 221–235
19. E. Goubault, S. Putot, Under-approximations of computations in real numbers based on generalized affine arithmetic, in *Proceedings of the Static Analysis Symposium*, vol. 4634. Lecture Notes in Computer Science (Springer, Berlin, 2007), pp. 137–152
20. E. Goubault, S. Putot, Forward inner-approximated reachability of non-linear continuous systems, in *Proceedings of the International Conference on Hybrid Systems: Computation and Control* (ACM, New York, 2017), pp. 1–10

21. E. Goubault, O. Mullier, S. Putot, M. Kieffer, Inner approximated reachability analysis, in *Proceedings of the International Conference on Hybrid Systems: Computation and Control* (ACM, New York, 2014), pp. 163–172
22. M. Janssen, Y. Deville, P. Van Hentenryck, Multistep filtering operators for ordinary differential equations, in *Principles and Practice of Constraint Programming*. Lecture Notes in Computer Science, vol. 1713 (Springer, Berlin, 1999), pp. 246–260
23. L. Jaulin, E. Walter, Set inversion via interval analysis for nonlinear bounded-error estimation. Automatica **29**(4), 1053–1064 (1993)
24. L. Jaulin, M. Kieffer, O. Didrit, E. Walter, *Applied Interval Analysis* (Springer, Berlin, 2001)
25. E. Kaucher, *Interval Analysis in the Extended Interval Space IR* (Springer, Berlin, 1980), pp. 33–49
26. Y. Lebbah, O. Lhomme, Accelerating filtering techniques for numeric CSPs. J. Artif. Intell. **139**(1), 109–132 (2002)
27. R.J. Lohner, Enclosing the solutions of ordinary initial and boundary value problems, in *Computer Arithmetic* (B.G. Teubner, Stuttgart, 1987), pp. 255–286
28. R.E. Moore, *Interval Analysis*. Series in Automatic Computation (Prentice Hall, Englewood Cliffs, 1966)
29. N.S. Nedialkov, K. Jackson, G. Corliss, Validated solutions of initial value problems for ordinary differential equations. Appl. Math. Comput. **105**(1), 21–68 (1999)
30. A. Rauh, M. Brill, C. GüNther, A novel interval arithmetic approach for solving differential-algebraic equations with ValEncIA-IVP. Int. J. Appl. Math. Comput. Sci. **19**(3), 381–397 (2009)
31. F. Rossi, P. Van Beek, T. Walsh, *Handbook of Constraint Programming* (Elsevier, Amsterdam, 2006)
32. M. Rueher, Solving continuous constraint systems, in *International Conference on Computer Graphics and Artificial Intelligence* (2005)