Software and Systems Modeling (2021) 20:711–724
https://doi.org/10.1007/s10270-020-00858-7

THEME SECTION PAPER

Guaranteed master for interval-based cosimulation

Adrien Le Coënt[1] · **Julien Alexandre dit Sandretto**[1] · **Alexandre Chapoutot**[1]

Received: 28 February 2020 / Revised: 15 December 2020 / Accepted: 24 December 2020 / Published online: 19 January 2021

Abstract
In this paper, we tackle the problem of guaranteed simulation of cyber-physical systems, an important model for current engineering systems. Their is always increasing complexity which leads to models of higher and higher dimensions, yet typically involving multiple subsystems or even multiple physics. Given this modularity, we more precisely explore cosimulation of such dynamical systems, with the aim of reaching higher dimensions of the simulated systems. In this paper, we present a guaranteed interval-based approach for cosimulation of continuous time systems. We propose an algorithm which first proves the existence and returns an enclosure of global solutions, using only local computations. This mitigates the curse of dimensionality faced by global (guaranteed) integration methods. Local computations are then realized with a safe estimate of the other sub-systems until the next macro-step. We increase the accuracy of the approach by using an interval extrapolation of the state of the other sub-systems. We finally propose some possible further improvements including adaptive macro-step size. Our method is fully guaranteed, taking into account all possible sources of error. It is implemented in a C++ prototype relying on the DynIbex library, and we illustrate our approach on multiple examples of the literature.

Keywords Cosimulation · Guaranteed simulation · Integration methods

1 Introduction

Context Modern system design involves more and more model-based design [35,49]. In a few words, model-based design requires modeling a plant, analyzing and synthesizing a controller for the plant, simulating the plant and controller, and finally integrating all these phases by deploying the controller. Such design process requires strong safety guarantees in each part of the design process, particularly when the final system in safety critical. Many dynamical systems are modeled with differential equations, and current controller synthesis methods are performed and applied with computers. Systems involving physical and software components belong to the class of cyber-physical systems (CPS) [41]. In this paper, we present some tools with strong safety guarantees for the simulation of the plant and controller. The main issue faced with strong guarantees and formal methods in general is usually the scalability [2,30,31], meaning that formal methods can only be applied to systems of dimension much smaller than industrial scale models. In order to overcome this issue, we propose to apply cosimulation principles in a guaranteed way, so that our methods get closer to applicability on industrial scale models.

In a cosimulation setting, the global system is divided in (or is composed of different) sub-systems, for which different simulation units (and possibly schemes) are used. This type of approaches is particularly appropriate for two different types of systems:

(i) Systems presenting different types of dynamics, such as stiff [29] and non-stiff [28] dynamics, or multi-physics dynamics such as fluid–structure interaction [12], but one could add linear and nonlinear, symplectic or not, etc. In this case, they are particularly appropriate because they

Communicated by Eugene Syriani and Manuel Wimmer.

This work was supported by the "Chair Complex Systems Engineering - École polytechnique, THALES, DGA, FX, Dassault Aviation, Naval Group Research, ENSTA Paris, Télécom Paris, and Fondation ParisTech".

✉ Adrien Le Coënt
adrien.le-coent@ens-cachan.fr

Julien Alexandre dit Sandretto
julien.alexandre-dit-sandretto@ensta-paris.fr

Alexandre Chapoutot
alexandre.chapoutot@ensta-paris.fr

[1] ENSTA Paris, 828 Boulevard des Maréchaux, 91762 Palaiseau Cedex, France

allow to use, *e.g.,* implicit and explicit schemes simultaneously for the different parts of the system, allowing to spend less time and energy on the easier parts of the simulation. The fluid–structure interaction [32] is one such interesting example since a fluid in usually modeled using and Eulerian description, while structures use Lagrangian descriptions. The numerical methods used to simulate both systems are thus inherently different .

(ii) Systems modeled by large-scale ordinary differential equations (ODEs), such as discretized partial differential equation (PDE) models for example used in structural mechanics computations [5,52]. Often, these models can be decomposed in sub-problems, for example using domain decomposition methods [50].

Cosimulation consists in enabling simulation of a coupled system through the composition of simulators, or simulation units (SUs) [25,36]. SUs are given an initial state and an input and produce an output and a simulation trace. SUs advance their simulation without exchange of information with the other SUs for given amounts of time that, in this paper, we call macro-steps. They exchange values of their outputs only at the end of these (macro) steps, usually called communication times. In order to develop a guaranteed procedure in this setting, the exchange of information done at communication times is crucial, and must contain all the information needed to safely simulate over the next macro step. Note that the SUs used here should be considered as white box SUs (as opposed to black box SUs [24]) since the dynamics of the sub-systems is written explicitly.

Given an ODE of the form $\dot{x}(t) = f(t, x(t))$ with f continuous in t and globally Lipschitz in x, and a *set* of initial values X_0, a symbolic (or "set-valued" since the symbols used here are sets) integration method consists in computing a sequence of approximations $(t_n, \tilde{x}_n)$ of the solution $x(t; x_0)$ of the ODE with $x_0 \in X_0$ such that $\tilde{x}_n \approx x(t_n; x_0)$. Note that the Lipschitz property guarantees the existence and unicity of solutions and is almost universally used in the domain of guaranteed integration, but smoother functions (of class $\mathcal{C}^k$ for some $k > 1$) can be required if high-order methods are used. Symbolic integration methods extend classical *numerical* integration methods which correspond to the case where X_0 is just a singleton $\{x_0\}$. The simplest numerical method is Euler's method in which $t_{n+1} = t_n + h$ for some step-size h and $\tilde{x}_{n+1} = \tilde{x}_n + hf(t_n, \tilde{x}_n)$; so the derivative of x at time t_n, $f(t_n, x_n)$, is used as an approximation of the derivative on the whole time interval. This method is very simple and fast, but requires small step-sizes h. More advanced methods coming from the Runge–Kutta family use a few intermediate computations to improve the approximation of the derivative. The general form of an explicit s-stage Runge–Kutta formula of the form $\tilde{x}_{n+1} = \tilde{x}_n + h\Sigma_{i=1}^{s} b_i k_i$ where $k_i = f(t_n + c_i h, \tilde{x}_n + h\Sigma_{j=1}^{i-1} a_{ij} k_j)$ for $i = 2, 3, \ldots, s$. A challenging question is then to compute a bound on the distance between the true solution and the numerical solution, i.e., $\|x(t_n; x_{n-1}) - x_n\|$. This distance is associated to the *local truncation error* (LTE) of the numerical method; its interval formulation can be bounded if f is of class $\mathcal{C}^{p+1}$ for an s-stage Runge–Kutta method of order p.

Contribution In this paper, we suppose that the system is provided with a suitable decomposition, and we propose to use SUs that rely on symbolic (set-valued) Runge–Kutta-based integration methods. In this case, the computation of the LTE is the most time-consuming task. For each integration time step, it requires the computation of the Picard–Lindelöf operator and its evaluation on the truncation error. The computation times quickly blow up with the dimension of the ODE and increase exponentially with the order of the scheme, limiting in practice the dimensions of the ODE to a few dozens or even less if the order of the scheme exceeds 4. The Picard–Lindelöf operator merely over-approximates (bounds) the state of the system over a given time step. This operator cannot be computed on a full composed (industrial scale) system. We thus propose to distribute its computation using local computations in an iterative way and call this procedure the cross-Picard operator, which is the main ingredient of guaranteed cosimulation. The cross-Picard operator is used at communication times to yield over-approximations of the global state of the system over the next macro-step (using only local computations). Local Picard–Lindelöf operators can then be used with safe approximations of the global state as parameters. Once the cross-Picard operator is established, further improvements are proposed, such as the use of extrapolation of inputs based on interpolation polynomials in order to improve the accuracy of the cosimulation. This allows to use past macro-steps information in order to improve the input bounding of the next macro-step. We also discuss some practical issues regarding macro-step size choice, as well as the initialization of the cross-Picard computation. Our implementation is available at [1].

Related work Computing the solution at discrete times of a linear ODE when the initial condition is given as a box can be easily done using *zonotopes* [4,23,37], and this, because we know exactly the solution of the ODE, can be written as an affine transformation. Yet, generally, the exact solution of nonlinear differential equations cannot be obtained, and a numerical integration scheme is used to approximate the state of the system.

Most of the recent work on the symbolic (or set-valued) integration of nonlinear ODEs is based on the upper bounding of the Lagrange remainders either in the framework of Taylor series or Runge–Kutta schemes [2,3,8,10,13,14,18,45].

Sets of states are generally represented as vectors of intervals (or "rectangles") and are manipulated through interval arithmetic [46] or affine arithmetic [17]. Taylor expansions with Lagrange remainders are also used in the work of [3], which uses "polynomial zonotopes" for representing sets of states in addition to interval vectors. Affine arithmetic and its geometrical representation through zonotopes help to counteract two of the main limitations coming with interval analysis: (i) the wrapping effect is reduced during integration [2]; (ii) a convex set is more tightly enclosed by a zonotope than a box [37].

The *guaranteed* or *validated* solution of ODEs using interval arithmetic is studied in the framework of Taylor series in [19,42,46,48] and Runge–Kutta schemes in [2,8,9,22]. The former is the oldest method used in interval analysis community because the expression of the remainder of Taylor series is simple to obtain. Nevertheless, the family of Runge–Kutta methods is very important in the field of numerical analysis. Indeed, Runge–Kutta methods have several interesting stability properties which make them suitable for an important class of problems. The recent work [1] implements Runge–Kutta-based methods which prove their efficiency at low orders and for short simulations (fixed by the sampling period of the controller). Runge–Kutta methods, however, present some inherent limitations. In a guaranteed context, the accumulation of error is taken into account and often leads to sets quickly growing within time. An experienced user would mitigate this effect by choosing the right order of method, using more or less tolerance on local errors and thus using finer or larger time steps, or using an appropriate number of zonotope *generators* [37].

In the methods of symbolic analysis and control of hybrid systems, the way of representing sets of state values and computing reachable sets for systems defined by autonomous ordinary differential equations (ODEs) is fundamental (see for example [3,23]). Many tools using, among other techniques, linearization or hybridization of these dynamics are now available (e.g., SpaceEx [21], Flow* [14], iSAT-ODE [20]). An interesting approach appeared recently, based on the propagation of reachable sets using guaranteed Runge–Kutta methods with adaptive step-size control (see [8,33]). An originality of our work is to use such guaranteed integration methods in a cosimulation framework. This notion of guarantee of the results is very interesting, because it allows applications in critical domains, such as aeronautical, military and medical ones.

Cosimulation has been extensively studied in the past years [24,25] and has been reported in a number of industrial applications (see [24] for an extensive list domain applications and associated publications). However, most of the uses and tools developed rely on the FMI/FMU standard [6,11,51], which do not allow guaranteed simulation. To our knowledge, guaranteed cosimulation of systems has never been studied, and, as pointed out in [26], remains an important challenge. The main original contribution of this paper is to provide a first approach to guaranteed cosimulation.

However, compositional principles are close to the ideas we use here. A recent work [16] proposes to safely simulate nonlinear systems by using hybrid automata abstractions that can be computed in a decomposed (compositional) way. Similarly in [15], compositional abstractions are computed, but the abstractions are performed using relations expressed in linear arithmetic. In [7], numerical integration is performed locally by using a splitting of the vector field that can also be performed in a compositional way. Nevertheless, none of these works perform actual simultaneous simulations, but rather rely on pre-computations and abstractions. The work closest to guaranteed cosimulation is the error analysis carried out in [6]; it provides strong stability results as well as some error estimates, but does not allow to formally bound the global state of the system.

Organization of the paper In Sect. 2, we present some notations and preliminaries before introducing the mathematical setting classically used for (guaranteed) numerical simulation and for cosimulation. In Sect. 3, we give the main ideas that are used in guaranteed Runge–Kutta-based integration, as well as its limits. The main contribution is presented in Sect. 4, in which we present the computation of the cross-Picard operator, the cosimulation orchestration, as well as the practical improvements that can be used. We present some numerical applications issued from the literature in Sect. 5, and we conclude in Sect. 6.

2 Problem setting

2.1 Notations and preliminaries

The simplest and most common way to represent and manipulate sets of values is *interval arithmetic* (see [46]). An interval $[x] = [\underline{x}, \overline{x}]$ defines the set of reals x such that $\underline{x} \leq x \leq \overline{x}$. $\mathbb{IR}$ denotes the set of all intervals over reals. The diameter or the width of $[x]$ is denoted by $w([x]) = \overline{x} - \underline{x}$.

Interval arithmetic extends to $\mathbb{IR}$ elementary operators over $\mathbb{R}$. For instance, the interval sum, i.e., $[x_1] + [x_2] = [\underline{x_1} + \underline{x_2}, \overline{x_1} + \overline{x_2}]$, encloses the image of the sum function over its arguments.

Considering a generic operator $\oplus$ on $\mathbb{R}$, its interval extension is obtained as follows:

$$[x_1] \oplus [x_2] = [\min\{\underline{x_1} \oplus \underline{x_2}, \underline{x_1} \oplus \overline{x_2}, \overline{x_1} \oplus \underline{x_2}, \overline{x_1} \oplus \overline{x_2}\}, \max\{\underline{x_1} \oplus \underline{x_2}, \underline{x_1} \oplus \overline{x_2}, \overline{x_1} \oplus \underline{x_2}, \overline{x_1} \oplus \overline{x_2}\}]$$

Nowadays, interval arithmetic libraries follow a standard, and thus, implement special intervals such as $\emptyset$ and

$[-\infty, +\infty]$ and associated set operations to handle forbidden operations without exception like a division by zero for example.

An interval vector or a *box* $[\mathbf{x}] \in \mathbb{IR}^n$ is a Cartesian product of n intervals. The enclosing property basically defines what is called an *interval extension* or an *inclusion function*.

Definition 1 (*Inclusion function*) Consider a function $f : \mathbb{R}^n \to \mathbb{R}^m$, then $[f] : \mathbb{IR}^n \to \mathbb{IR}^m$ is said to be an extension of f to intervals if

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \quad [f]([\mathbf{x}]) \supseteq \{f(\mathbf{x}), \mathbf{x} \in [\mathbf{x}]\} \ .$$

It is possible to define inclusion functions for all elementary functions such as $\times$, $\div$, sin, cos, exp, etc. 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 [34] for more details).

Finally, combining the inclusion function and the rectangle rule, an integral can be bounded as follows:

$$\int_a^b f(x)\,\mathrm{d}x \in (b-a) \cdot [f]([a,b])$$

2.2 Guaranteed simulation objective

We introduce the Initial Value Problem, which is the main problem we want to solve.

Definition 2 (*Initial value problem (IVP)*) Consider an ODE with a given initial condition

$$\dot{x} \in f(t, x, p) \quad \text{with} \quad x(0) \in [x^0], \ p \in [p], \tag{1}$$

with $f : \mathbb{R}^+ \times \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d$ assumed to be continuous in t and p and globally Lipschitz in x. We assume that parameters p are bounded in $[p]$ (used to represent a perturbation, a modeling error, an uncertainty on measurement, ...). Solving an *IVP* consists in finding a function $x(t)$ described by Eq. (1) for all perturbation p lying in $[p]$ and for all the initial conditions in $[x^0] \subseteq \mathbb{R}^d$.

Since this problem cannot be solved exactly, numerical schemes are used. In our case, the Runge–Kutta schemes we use return sets of boxes $\{[x^n]\}_n$ that cover the possible trajectories for a given time interval $[0, H]$: for all $t \in [0, H]$, $x(t) \in [x^n]$ for some n. (We leave the number of covering boxes arbitrary for now, see an illustration Fig. 1.)

We now suppose that the dynamics can be decomposed as follows:

$$\begin{aligned}
&\dot{x}_1 \in f_1(t, x_1, u_1) \quad \text{with} \quad x_1(0) \in [x_1^0], \ u_1 \in [u_1], \\
&\dots \\
&\dot{x}_2 \in f_2(t, x_2, u_2) \quad \text{with} \quad x_2(0) \in [x_2^0], \ u_2 \in [u_2], \\
&\dots \\
&\dot{x}_m \in f_m(t, x_m, u_m) \quad \text{with} \quad x_m(0) \in [x_m^0], \ u_m \in [u_m], \\
&L(x_1, \dots, x_m, u_1, \dots, u_m) = 0,
\end{aligned}$$

where the state x is decomposed in m components $x = (x_1, \dots, x_m)$, for all $i \in \{1, \dots, m\}$, $x_i \in X_i$, $X_1 \times \cdots \times X_m = \mathbb{R}^d$, and L is a coupling function between the components. The objective is now to compute, for each component $i \in \{1, \dots, m\}$, sets of boxes $\{[x_i^k]\}_k$ that cover the possible trajectories of the state x_i. For the remainder of the paper, index i is used to denote the state of a component.

A standard formalism introduced in [25] defines the behavior of a continuous time simulation unit S_i as:

$$\begin{aligned}
&S_i = \langle X_i, U_i, Y_i, \delta_i, \lambda_i, x_i(0), \Phi_{U_I} \rangle, \\
&\delta_i : \mathbb{R} \times X_i \times U_i \to X_i, \\
&\lambda_i : \mathbb{R} \times X_i \times U_i \to Y_i, \ \text{or } \mathbb{R} \times X_i \to Y_i, \\
&x_i(0) \in X_i, \\
&\Phi_{U_i} : \mathbb{R} \times U_i \times \cdots \times U_I \to U_i,
\end{aligned} \tag{2}$$

where

- X_i is the state vector space,
- U_i is the input vector space,
- Y_i is the output vector space,
- $\delta_i(t, x_i(t), u_i(t)) = x_i(t + H)$ or $\delta_i(t, x_i(t), u_i(t + H)) = x_i(t + H)$ is the function that instructs the SU to compute a behavior trace from t to $t + H$, making use of the input extrapolation (or interpolation) function Φ_{U_i}
- $\lambda_i(t, x_i(t), u_i(t)) = y_i(t)$ or $\lambda_i(t, x_i(t)) = y_i(t)$ is the output function; and
- $x_i(0)$ is the initial state.

We consider that the entire state of each sub-system is returned by each sub-system, so that we can omit the use of functions λ_i. Furthermore, we need interval-based simulations. For this purpose, we simply modify the δ_i functions as follows. The notation is now :

$$([x_i]', \{[x_i^k]'\}_k) := \delta_i([t, t'], [x_i], [u_i]).$$

They take as inputs:

- An interval of $\mathbb{R}$: $[t, t']$.
- A box of X_i: $[x_i]$.
- A box of U_i: $[u_i]$.

And they return:

- A box of X_i: $[x_i]'$. This box over-approximates the state $x_i(t)$ over the time interval $[t, t']$ for any input varying

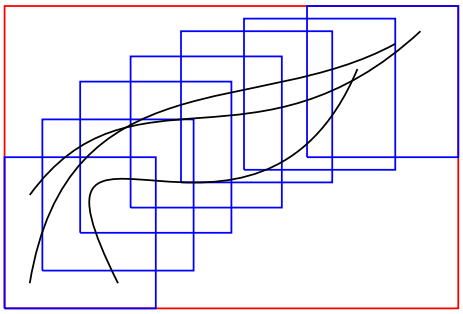

Fig. 1 Illustration of the outputs of functions δ_i. They return boxes covering the trajectories of their component over the next macro-step. The three black lines are exact trajectories over the time interval $[t, t']$. The red box is $[x_i]'$, over-approximating the state over the whole time interval. The blues boxes are the $\{[x_i^k]'\}_k$, covering the trajectories starting in $[x_i]$

in $[u_i]$. It is thus a box that over-approximates $x_i(t)$ over the next entire macro-step.

– A set of boxes of X_i: $\{[x_i^k]'\}_k$. This set of boxes covers trajectories starting in $[x_i]$ over the time interval $[t, t']$ and any input varying in $[u_i]$. The union of these boxes forms a tube that gives a tight over-approximation of the trajectories $x_i(t)$ over the time interval $[t, t']$.

An illustration given in Fig. 1 shows these different boxes.

A continuous time cosimulation scenario with reference cs includes at least the following information:

$$S = \langle U_{cs}, Y_{cs}, D, \{S_i : i \in D\}, L\rangle,$$
$$L : \left(\prod_{i\in D} Y_i\right) \times Y_{cs} \times \left(\prod_{i\in D} U_i\right) \times U_{U_{cs}} \to \mathbb{R}^m, \tag{3}$$

where D is an ordered set of SU references, each S_i is defined as in Eq. (2), $m \in \mathbb{N}$, U_{cs} is the vector space of input external to the scenario, Y_{cs} is the vector space of outputs of the scenario, and L induces the SU coupling constraints, that is, if $D = \{1, \ldots, n\}$, then the coupling is the solution to $L(y_1, \ldots, y_n, y_{cs}, u_1, \ldots, u_n, u_{cs}) = \bar{0}$, where $\bar{0}$ denotes the null vector. Note that, compared to [25], we do not consider approximation functions for inputs since we provide guaranteed results which cannot be established with such approximations.

In the following, we suppose that coupling constraints L are explicit, i.e., inputs can be written as $u_i = K_i(y_1, \ldots, y_n, y_{cs})$ for all i. This means that we do not currently consider algebraic loops. In future work, we plan on generalizing the coupling to arbitrary (algebraic) couplings, using differential algebraic equation formulations, already available with guaranteed integration methods such as in [18]. Note that the system definition used here implies that the global system is fully time-continuous. However, if the system presents time-dependent piecewise behavior (as would be found in, e.g., switching systems with time-dependent switchings [39]), then the system can be simulated using the present method as long as there is no switching during a macro-step. Indeed, using minor modifications of the procedure, communications can be scheduled to happen at the switching times and cosimulation can be performed safely until the next switching. State-dependent switchings (i.e., guards in the state-space) are still very hard to handle in guaranteed reachability analysis methods and are thus not possible in the present method.

3 Guaranteed Runge–Kutta schemes

In this section, we describe our approach for validated simulation based on Runge–Kutta methods [2,8], the goal being to obtain a set valued solution for a single system, described like in IVP in Eq. (1) for a given time interval $[T, T + H]$. The approach is used inside simulation units, and index i is omitted (only for this section). Note that the notations of this section differ from the others in order to keep the number of indexes as low as possible.

A numerical integration method computes a sequence of values (t_n, x_n) with $0 \le n \le N, t_n = T + n\frac{H}{N}$, approximating the solution $x(t; x_0)$ of the IVP defined in Eq. (1) such that $x_n \approx x(t_n; x_{n-1})$. The simplest method is Euler's method in which $t_{n+1} = t_n + h$ for some step size h and $x_{n+1} = x_n + h \times f(t_n, x_n, p)$; so the derivative of x at time t_n, $f(t_n, x_n, p)$, is used as an approximation of the derivative on the whole time interval to perform a linear interpolation. This method is very simple and fast, but requires small step sizes. More advanced methods, coming from the Runge–Kutta family, use a few intermediate computations to improve the approximation of the derivative. The general form of an explicit s-stage Runge–Kutta formula, that is using s evaluations of f, is

$$\begin{aligned} x_{n+1} &= x_n + h\sum_{i=1}^{s} b_i k_i \ , \\ k_1 &= f\big(t_n,\ x_n, p\big) \ , \\ k_i &= f\Big(t_n + c_i h,\ x_n + h\sum_{j=1}^{i-1} a_{ij} k_j,\ p\Big),\ i = 2, 3, \ldots, s \ . \end{aligned} \tag{4}$$

The coefficients c_i, a_{ij} and b_i fully characterize the method. To make Runge–Kutta validated, the challenging question is how to compute a guaranteed bound of the distance between the true solution and the numerical solution, defined by $x(t_n; x_{n-1}) - x_n$. This distance is associated to the *local truncation error* (LTE) of the numerical method.

To bound the LTE, we rely on *order condition* [28] respected by all Runge–Kutta methods. This condition states

that a method of this family is of order p iff the $p+1$ first coefficients of the Taylor expansion of the solution and the Taylor expansion of the numerical methods are equal. In consequence, LTE is proportional to the Lagrange remainders of Taylor expansions. Formally, LTE is defined by (see [8]):

$$\begin{aligned} & x(t_n; x_{n-1}) - x_n \\ & = \frac{h^{p+1}}{(p+1)!}\left(f^{(p)}\left(\xi, x(\xi; x_{n-1}), p\right) - \frac{d^{p+1}\phi}{dt^{p+1}}(\eta)\right) \\ & \quad \xi \in]t_n, t_{n+1}[\text{ and } \eta \in]t_n, t_{n+1}[\ . \end{aligned} \tag{5}$$

The function $f^{(n)}$ stands for the n-th derivative of function f w.r.t. time t that is $\frac{d^n f}{dt^n}$ and $h = t_{n+1} - t_n$ is the step size. The function $\phi : \mathbb{R} \to \mathbb{R}^n$ is defined by $\phi(t) = x_n + h\sum_{i=1}^{s} b_i k_i$ where k_i are defined as Eq. (4).

The challenge to make Runge–Kutta integration schemes safe w.r.t. the true solution of IVP is then to compute a bound of the result of Eq. (5). In other words, we do have to bound the value of $f^{(p)}\left(\xi, x(\xi; x_{n-1}), p\right)$ and the value of $\frac{d^{p+1}\phi}{dt^{p+1}}(\eta)$ with numerical guarantee. The latter expression is straightforward to bound because the function ϕ only depends on the value of the step size h, and so does its $(p+1)$-th derivative. The bound is then obtained using the affine arithmetic [17,18].

However, the expression $f^{(p)}\left(\xi, x(\xi; x_{n-1}), p\right)$ is not so easy to bound as it requires to evaluate f for a particular value of the IVP solution $x(\xi; x_{n-1})$ at an unknown time $\xi \in]t_n, t_{n+1}[$. The solution used is the same as the one found in [9,48], and it requires to bound the solution of IVP on the interval $[t_n, t_{n+1}]$. This bound is usually computed using the Banach's fixpoint theorem applied with the Picard–Lindelöf operator, see [48]. This operator is used to compute an enclosure of the solution $[\tilde{x}]$ of IVP over a time interval $[t_n, t_{n+1}]$, that is for all $t \in [t_n, t_{n+1}]$, $x(t; x_{n-1}) \in [\tilde{x}]$. We can hence bound $f^{(p)}$ substituting $x(\xi; x_{n-1})$ by $[\tilde{x}]$. This general approach used to solve IVPs in a validated way is called Lohner two step approach [43].

Complexity of LTE computation The validated computation of the LTE, given in Eq. (5), of Runge–Kutta methods can be performed using two different methods: symbolic differentiation or automatic differentiation (AD). The first method is based on Frechet derivatives and rooted trees [18], while the second exploits automatic differentiation and a weighted directed acyclic graph [47]. For a Runge–Kutta method of order k and an ODE of dimension d, the complexities are $\mathcal{O}(d^k)$ for symbolic method and $\mathcal{O}(d3^k)$ for AD [47]. A gain in term of dimension, for example by splitting the problem and using cosimulation, directly impacts the time of LTE computations and then the time of simulation.

4 Guaranteed cosimulation algorithm

In this section, the main contribution of the paper is presented. We first present in details the computation of the Picard–Lindelöf operator. The operator being too time-consuming to compute on industrial case studies, we then present the cross-Picard operator, which contains a procedure computing an enclosure of the global state of the system using local Picard–Lindelöf operators. Its computation is realized at communication times in order to over-approximate the state over the next macro-step. A given SU can then perform safe simulations until the end of the macro-step, by considering the inputs from the other sub-systems as bounded perturbations, the bounded set in which they lie being known from the cross-Picard operator.

4.1 The Picard–Lindelöf operator

Let us consider Eq. (1) with an initial condition $[x^T]$ and a perturbation set $[p]$; the guaranteed integration of such a system on a time interval $[T, T+H]$ is made possible by over-approximating the state over $[T, T+H]$ using the Picard–Lindelöf operator. Its construction is detailed in the following. We first recall the following theorem.

Theorem 1 (Banach fixed-point theorem) *Let (K, d) be a complete metric space, given by a set K and a distance function $d : K \times K \mapsto \mathbb{R}$, and let $g : K \to K$ be a contraction; that is for all x, y in K there exists $c \in (0, 1)$ such that*

$$d\left(g(x), g(y)\right) \leq c \cdot d(x, y)$$

Then g has a unique fixed-point in K.

In the context of IVPs and schemes of order p, we consider the space of continuously differentiable functions $C^{p+1}([T, T+H], \mathbb{R}^n)$ and the Picard–Lindelöf operator

$$\mathcal{P}_f(x) = t \mapsto x + \int_T^t f(s, x(s), [p])ds \ . \tag{6}$$

The Picard–Lindelöf operator is used to check the contraction of the solution on an integration step in order to prove the existence and the uniqueness of the solution of Eq. (1) as stated by the Banach's fixed-point theorem. Furthermore, this operator is used to compute an enclosure of the solution of IVP over a time interval $[T, T+H]$.

This operator, based on the Theorem 1 and defined in Eq. (6), allows one to compute the a priori enclosure $[\tilde{x}]$ such that

$$\begin{aligned} & \forall t \in [T, T+H], \\ & \quad \{x(t; x(T)) : x(T) \in [x^T], \ p \in [p]\} \subseteq [\tilde{x}] \ . \end{aligned}$$

In its simplest (rectangle) form, the operator is computed as:

$$\mathcal{P}_f([x^T],[p],[r],H) = [x^T] + f([r],[p])[0,H]\ . \tag{7}$$

The Picard–Lindelöf operator with Taylor expansion is given by:

$$\mathcal{P}_f([x^T],[p],[r],h) = [x^T] + \sum_{k=0}^{N} f^{[k]}([x^T],[p])[0,H^k] + f^{[N+1]}([r],[p])[0,H^{N+1}]\ . \tag{8}$$

If $\mathcal{P}_f([x^T],[p],[r],H) \subset \text{Int}([r])$, $\text{Int}(\cdot)$ denoting the interior of a set, then f is integrable and $\{x(t;x(T)) : x(T) \in [x^T],\ p \in [p]\} \subset [r]$ for any $t \in [T, T+H]$.

In order to ease the reading, in the remainder of the paper, the box $[r]$ that verifies $\mathcal{P}_f([x^T],[p],[r],H) \subset \text{Int}([r])$ is referred to as the Picard box on time interval $[T, T+H]$, and its computation is denoted by the operator $\mathcal{P}^H_{X,D}$ for an initial set X at time t, a disturbance set D, and a time step H.

Once the Picard box is computed, we can safely simulate the system on time interval $[T, T+H]$ by computing the LTE, and the result is validated for any disturbance $p \in [p]$ on the same time interval.

4.2 Cross-Picard operator

The purpose of the cross-Picard operator is to over-approximate the solutions of all the sub-systems over the next macro-step, using only local computations. The principle is that we compute local Picard operators, by considering the inputs coming from the other sub-systems as disturbances, the main issue being to compute the sets in which these disturbances evolve. To compute these sets, we start by guessing a rough over-approximation of the solutions over the next macro-step. From there, the idea is that we consider the inputs $u_i(t)$ of the sub-systems as bounded disturbances, the set in which they are bounded being constructed from functions K_i and the initial guesses. We then apply local Picard operators iteratively, until the proof of validity of the approximations is obtained for all sub-systems.

More precisely, let us consider a cosimulation scenario $S = \langle \emptyset, Y_{cs}, D, \{S_i : i \in D\}, L\rangle$ with simulation units $S_i = \langle X_i, U_i, Y_i, \delta_i, \lambda_i, x_i(0), \Phi_{U_I}\rangle$. Let us suppose that the sets of states are non-overlapping, i.e., simulation unit S_i does not share any state variables with simulation unit S_j for $i \neq j$. Note that, in order to ease the reading, we suppose an empty input set $U_{cs} = \emptyset$, but minor modifications would allow to take bounded inputs into account (as long as they can be bounded using intervals or boxes).

Let us denote by $[x_{i,n}]$ the initial state set $x_i(T_n)$. Let us denote by $[x^H_{i,n}]$ the over-approximation of $x_i(t)$ for $t \in [T_n, T_n+H]$. Let us denote by $[u^H_{i,n}]$ the over-approximation of $u_i(t)$ for $t \in [T_n, T_n+H]$. A local Picard operator can be computed as $\mathcal{P}^H_{[x_{i,n}],[u^H_{i,n}]}$. In order to prove that $[x^H_{i,n}]$ are indeed over-approximating $x_i(t)$ for all i over the next macro-step, the condition to verify is: $\mathcal{P}^H_{[x_{i,n}],[u^H_{i,n}]} \subset \text{Int}([x^H_{i,n}])$, for all i.

The global Picard box is then approximated by

$$\mathcal{P}^H_{[x_{1,n}]\times\cdots\times[x_{m,n}]} := [x^H_{1,n}] \times \cdots \times [x^H_{m,n}]$$

which ensures a safe over-approximation of the states over the next macro-time step.

In order to compute such safe approximations of the states over a macro-step, using only local computations, we perform the following procedure:

- For each i, compute rough guesses $[r^H_{i,n}]$ of the sets $[x^H_{i,n}]$
- From $\{[r^H_{i,n}]\}_{i=1,\ldots,m}$, deduce input box guesses $[k^H_{i,n}]$ over-approximating the inputs $u_i \in U_i$ on $[T_n, T_n+H]$: $[k^H_{i,n}] := K_i([r^H_{1,n}],\ldots,[r^H_{m,n}])$
- For each i, compute a Picard box $P^H_{[x_{i,n}],[k^H_{i,n}]}$
- While $[r^H_{i,n}] \not\subseteq \mathcal{P}^H_{[x_{i,n}],[p^H_{i,n}]}$, for all i, computes $[r^H_{i,n}] := \mathcal{P}^H_{[x_{i,n}],K_i([r^H_{1,n}],\ldots,[r^H_{m,n}])}$

The computation of the initial guesses is discussed in Sect. 4.5. The exact algorithm is detailed in Algorithm 1. We abbreviate the computation of such safe approximations of the states over a macro-step by the following operator, that we denote the *cross-Picard* operator:

$$\left([x^H_{1,n}],\ldots,[x^H_{m,n}]\right) = \mathcal{P}^H([x_{1,n}],\ldots,[x_{m,n}]) \tag{9}$$

Please note that this operator involves an iterative procedure for its practical computation.

Remark 1 We would like to point out that the computation of the cross-Picard operator can fail if the local computations $[r^H_{i,n}] := \mathcal{P}^H_{[x_{i,n}],K_i([r^H_{1,n}],\ldots,[r^H_{m,n}])}$ lead to overly large sets. This can be the case if:

- The system is hard to simulate due to, e.g., nonlinearities
- There is too much interaction between the sub-systems, i.e., the sub-systems share too many variables.
- The macro-step is too large.

All these problems can be mitigated in practice by using smaller macro-steps, which is why we provide an adaptive macro-step version of the procedure in Sect. 4.5. Smaller

macro-steps mean a faster convergence of the cross-Picard computations, and a generally more accurate simulation, but they also need to be performed more often. In practice, the accuracy required seems to be more limiting than the possible failure of the cross-Picard computation (see Sect. 5).

Algorithm 1 Computation of the cross-Picard operator

Data: $cs = \langle \emptyset, Y_{cs}, D = \{1, \dots, m\}, \{S_i\}_{i \in D}, L, \emptyset \rangle$, a time interval $[t, t + H]$, initial intervals $[x_{i,n}]$ and initial guesses $[r^H_{i,n}]$
Result: $\{[X^H_i]\}_{i=1,\dots,m}$, a set of boxes over-approximating the global state on $[T_n, T_n + H]$

for $i = 1, \dots, m$ (in parallel) **do**
  $[\tilde{X}^H_i] := [r^H_{i,n}]$
  $[U^H_i] := K_i([\tilde{X}^H_{1,n}], \dots, [\tilde{X}^H_{1,n}])$
  $[X^H_i] := \mathcal{P}^H_{[x_{i,n}],[U^H_i]}$
while $[X^H_i] \not\subseteq [\tilde{X}^H_i]$ for all i **do**
  for $i = 1, \dots, m$ (in parallel) **do**
    $[\tilde{X}^H_i] := [X^H_i]$
    $[U^H_i] := K_i([\tilde{X}^H_{1,n}], \dots, [\tilde{X}^H_{1,n}])$
    $[X^H_i] := \mathcal{P}^H_{[x_{i,n}],[U^H_i]}$
return $[X^H_i]$

4.3 Orchestration of simulation units

Once a valid over-approximation of the states is computed, cosimulations can be performed. The principle of cosimulation orchestration is illustrated in Fig. 2. The main idea is to compute (in a distributed way) safe and accurate simulations of each sub-system, by considering the other sub-systems as disturbances. Sub-systems exchange information every H time units. This exchange of information is used to update the disturbance set to consider in the next time step. This orchestration scheme is very close to a Jacobi orchestration scheme [27], which asks all units to simulate in parallel, exchanging their input values at the end of the macro-step. The main difference is that we rely on sets instead of point values. One could also argue that our method differs from a Jacobi scheme in the sense that we exchange values at the beginning of macro-step, in order to predict the global state of the system, and not at the end in order to readjust the input values.

The detailed orchestration procedure is given in Algorithm 2. We would like to point out that the cross-Picard operator used in Algorithm 2, as abbreviated in Eq. (9), actually includes the iterative computations detailed in the previous subsection and Algorithm 1.

Complexity discussion Let us recall that, for a Runge–Kutta method of order k and an ODE of dimension d, the

Algorithm 2 Cosimulation orchestrator for autonomous systems

Data: $cs = \langle \emptyset, Y_{cs}, D = \{1, \dots, m\}, \{S_i\}_{i \in D}, L, \emptyset \rangle$, a macro-step H
Result: A cosimulation trace given as a set of boxes
$n := 0$
$T_n := 0$
$[x_{i,n}] := x_i(0)$ for $i = 1, \dots, m$
while `True` **do**
  Compute $([x^H_{1,n}], \dots, [x^H_{m,n}]) := \mathcal{P}^H([x_{1,n}], \dots, [x_{m,n}])$
  for $i = 1, \dots, m$ (in parallel) **do**
    $[u^H_{i,n}] := K_i([x^H_{1,n}], \dots, [x^H_{1,n}])$
    Advance simulation $([x_{i,n+1}], \{[x^k_{i,n+1}]\}_k) := \delta_i([t_n, t_{n+1}], [x_{i,n}], [u^H_{i,n}])$
  $t_{n+1} := t_n + H$
  $n := n + 1$
return $\{[x_{i,n}], \{[x^k_{i,n}]\}_k\}_n$

complexities for computing the LTE are $\mathcal{O}(d^k)$ for symbolic differentiation and $\mathcal{O}(d3^k)$ for automatic differentiation [47]. When the system is decomposed in two components of dimension d_1 and d_2 with $d = d_1 + d_2$ and $d_1 = \mathcal{O}(d/2)$ and $d_2 = \mathcal{O}(d/2)$, the complexities become, for each SU, respectively $\mathcal{O}(\frac{d^k}{2^k})$ and $\mathcal{O}(\frac{d}{2}3^k)$. The first one is divided by two with a simple Euler scheme. We, however, need to compute the cross-Picard operator, which adds an iterative computation before each macro-step.

4.4 Guaranteed interval extrapolation

The guaranteed extrapolation relies on an interpolation of the previous time steps. As stated in Sect. 3, function Φ_{U_i} is the input function, it is given as an input to system i. In other words, inputs $u_i(t)$ is replaced by a function $\Phi_{U_i}(t)$. The simplest approach is to consider $\Phi_{U_i}(t)$ constant on the next macro-step $[T_n, T_{n+1}]$. In order to yield more accurate results, a classical approach (see [6]) is to build a extrapolation function based on interpolation polynomials:

$$\begin{aligned}\Phi_{U_i,n}(t) &= \sum_{l=0}^{k} u_i(T_{n-l}) \prod_{\substack{p=0 \\ p \neq l}}^{k} \frac{t - T_{n-p}}{T_{n-l} - T_{n-p}} \\ &= u_i(t) + O(H^{k+1}) \end{aligned} \tag{10}$$

In order to yield guaranteed results using such an approach, the formula has to be extended to interval values, and the remainder in $O(H^{k+1})$ has to be bounded. This remainder is given by $\frac{1}{(k+1)!} u_i^{(k+1)}(\xi_t) \prod_{i=0}^{k}(t - T_{n-k})$ for some $\xi_t \in [T_n, T_{n+1}]$. The exact interpolation is then given by:

$$\Phi_{U_i,n}(t) = \sum_{l=0}^{k} u_i(T_{n-l}) \prod_{\substack{p=0 \\ p \neq l}}^{k} \frac{t - T_{n-p}}{T_{n-l} - T_{n-p}}$$

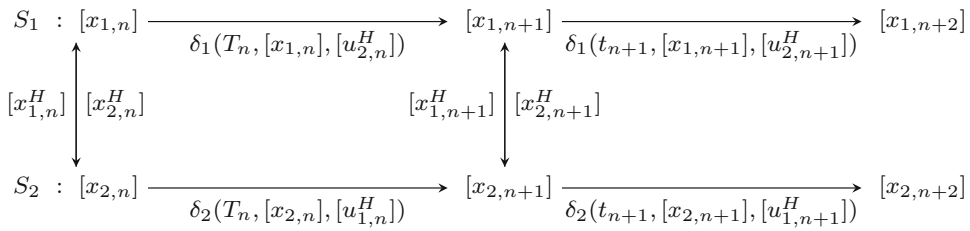

Fig. 2 Orchestration of two guaranteed simulation units between times T_n and $T_n + 2H$

$$+\frac{1}{(k+1)!}u_i^{(k+1)}(\xi_t)\prod_{i=0}^{k}(t-T_{n-k})$$

Inputs $u_i(t)$ being given by functions K_i, an interval bounding the derivatives $u_i^{(k+1)}$, can be evaluated exactly from the global Picard box. Recall that for all i,

$$u_i(t) = K_i(x_1(t), \ldots, x_m(t))$$

The k-th derivative of u_i can be evaluated exactly, either by hand if K_i is simple enough or using a higher chain formula [44] of the form:

$$u_i^k(t) = k!\frac{\partial^{r_1+\cdots+r_m}K_i}{\partial x_1^{r_1}\ldots\partial x_m^{r_m}}\prod_{j=1}^{s}\prod_{l=1}^{m}\frac{1}{m_{jl}!}\left[\frac{1}{p_j!}x_i^{(p_j)}\right]^{m_{jl}}$$

where multi-indexes r, m, and p are given in [44]. In any case, the derivative in the remainder only depends, numerically, on derivatives $x_i^{(p_j)}(\xi_t)$, which can fortunately be evaluated (symbolically) in DynIbex [1,18]. In the end, we have a safe over-approximation of $u_i^{(k)}(t)$ for any $t \in [T_n, T_{n+1}]$, that we denote by $[u_{i,n}^{(k),H}]$. We thus have the following guaranteed interval formula for extrapolating the inputs over the next macro-step:

$$[\Phi_{U_i,n}](t) = \sum_{l=0}^{k}[u_{i,n-l}]\prod_{\substack{p=0\\p\neq l}}^{k}\frac{t-T_{n-p}}{T_{n-l}-T_{n-p}} + \frac{1}{(k+1)!}[u_{i,n}^{(k),H}]\prod_{i=0}^{k}(t-T_{n-k}) \quad (11)$$

The orchestration of simulation units using such an extrapolation is given in Algorithm 3.

4.5 Practical improvements

Adaptive time step Guaranteed simulation can sometimes be overly conservative; it leads to simulations in the shape of trumpets (such as Fig. 4b); one of the main disadvantages is that it can sometimes fail to compute a Picard box. In this case, a smaller time step makes it easier to compute the Picard box. An adaptive time step is already used for local computations [18]. Algorithm 4 implements an adaptive macro-step

Algorithm 3 Cosimulation orchestrator for autonomous systems with extrapolation

Data: $cs = \langle \emptyset, Y_{cs}, D = \{1, \ldots, m\}, \{S_i\}_{i\in D}, L, \emptyset\rangle$, a macro-step H, an order of interpolation k
Result: A cosimulation trace given as a set of boxes
$n := 0$
$T_n := 0$
$[x_{i,n}] := x_i(0)$ for $i = 1, \ldots, m$
while `True` **do**
  Compute $([x^H_{1,n}], \ldots, [x^H_{m,n}]) := \mathcal{P}^H([x_{1,n}], \ldots, [x_{m,n}])$
  for $i = 1, \ldots, m$ (in parallel) **do**
    for $j = 1, \ldots, k$ **do**
      Evaluate $[x_{i,n}^{(j),H}]$
  for $i = 1, \ldots, m$ (in parallel) **do**
    $[u_{i,n}] := K_i([x_{1,n}], \ldots, [x_{1,n}])$
    Compute $[\Phi_{U_i,n}]$
    Advance simulation $([x_{i,n+1}], \{[x^k_{i,n+1}]\}_k) := \delta_i([t_n, t_{n+1}], [x_{i,n}], [\Phi_{U_i,n}])$
  $t_{n+1} := t_n + H$
  $n := n + 1$
return $\{[x_{i,n}], \{[x^k_{i,n}]\}_k\}_n$

guaranteeing that the simulation always succeeds. In this implementation, the computation

$$([x^H_{1,n}], \ldots, [x^H_{m,n}]) := \mathcal{P}^H([x_{1,n}], \ldots, [x_{m,n}])$$

is limited to a given number of iterations, after which a Boolean marker SUCCESS is set to 1 or 0 depending of the computation of a valid Picard box or not.

Computation of the initial guess We discuss here the computation of the initial guesses of Algorithm 1. More precisely: for each i, compute rough guesses $[r^H_{i,n}]$ of the sets $[x^H_{i,n}]$.

Several heuristics are possible for this. The first and simplest one is to take the previous Picard box $[x^H_{i,n-1}]$ and inflate it of some given percentage ε :

$$[r^H_{i,n}] := [\underline{x^H_{i,n-1}} - \varepsilon\%, \overline{x^H_{i,n-1}} + \varepsilon\%],$$

and hope that it is inflated enough to obtain $\mathcal{P}^H_{[x_{i,n}],[u^H_{i,n}]} \subset \text{Int}([x^H_{i,n}])$, for all i.

A more conservative possibility is to compute it as an inflation of the union of the previous Picard box and the

Algorithm 4 Cosimulation orchestrator for autonomous systems with extrapolation and adaptive macro-step

Data: $cs = \langle \emptyset, Y_{cs}, D = \{1, \ldots, m\}, \{S_i\}_{i \in D}, L, \emptyset \rangle$, a macro-step H, an order of interpolation k
Result: A cosimulation trace given as a set of boxes
$n := 0$
$T_n := 0$
$t_{n+1} := H$
$[x_{i,n}] := x_i(0)$ for $i = 1, \ldots, m$
while `True` **do**
  Compute $([x^H_{1,n}], \ldots, [x^H_{m,n}]) := \mathcal{P}^H([x_{1,n}], \ldots, [x_{m,n}])$
  if SUCESS **then**
    for $i = 1, \ldots, m$ (in parallel) **do**
      for $j = 1, \ldots, k$ **do**
        Evaluate $[x^{(j),H}_{i,n}]$
    for $i = 1, \ldots, m$ (in parallel) **do**
      $[u_{i,n}] := K_i([x_{1,n}], \ldots, [x_{1,n}])$
      Compute $[\Phi_{U_i,n}]$
    Advance simulation $([x_{i,n+1}], \{[x^k_{i,n+1}]\}_k) := \delta_i([t_n, t_{n+1}], [x_{i,n}], [\Phi_{U_i,n}])$
    $t_{n+1} := t_n + H$
    $n := n + 1$
  else
    $H := H/2$
return $\{[x_{i,n}], \{[x^k_{i,n}]\}_k\}_n$

current one:

$$[r^H_{i,n}]^{\text{temp}} := [x^H_{i,n-1}] \cup \mathcal{P}^H_{[x_{i,n}], K_i([r^H_{1,n}], \ldots, [r^H_{1,n}])},$$

and

$$[r^H_{i,n}] = [\underline{r^{H\ \text{temp}}_{i,n}} - \varepsilon\%, \overline{r^{H\ \text{temp}}_{i,n}} + \varepsilon\%].$$

Finally, the most conservative way to compute it is to ensure that an over-approximation of $[x^H_{i,n}]$ is obtained. It can be done in an iterative way as follows:

– Initialize

$$[r^H_{i,n}]^{\text{temp}} := [x^H_{i,n-1}] \cup \mathcal{P}^H_{[x_{i,n}], K_i([r^H_{1,n}], \ldots, [r^H_{1,n}])},$$

– For m iterations, compute:

$$[r^H_{i,n}]^{\text{temp}} := \mathcal{P}^H_{[x_{i,n}], K_i([r^H_{1,n}]^{\text{temp}}, \ldots, [r^H_{m,n}]^{\text{temp}})},$$

– Return

$$[r^H_{i,n}] = [r^H_{i,n}]^{\text{temp}}.$$

Just as in [2], m iterations are used to ensure that the growth of the input sets have propagated to all the dimensions.

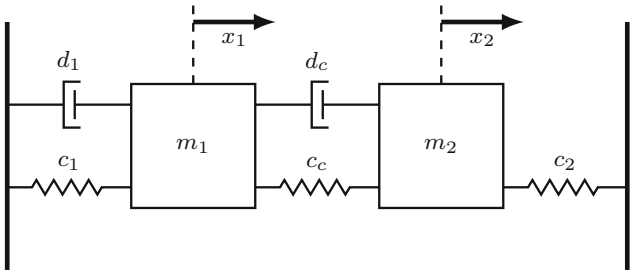

Fig. 3 Illustration of the two mass-spring-damper system

5 Numerical examples

The algorithms presented here are implemented in a C++ prototype relying on the DynIbex library [1]. Note that, in this prototype, the cosimulations are not performed in a parallel manner, but in a sequential one. On a given macro-step, the different simulation units compute their simulation step one after the other. Parallel executions of the DynIbex library is one of our future plans. The computation times given in the following are performed on a Intel Core i5-4430 associated to 8GB of RAM, running on Ubuntu 18.04 LTS.

5.1 Double mass-spring-damper oscillator

We consider a double mass-spring-damper oscillator considered in [25]. A figure of the system is given in Fig. 3.

The dynamics of the system is given by the following system of equations:

$$\begin{cases} \dot{x}_1 = v_1 \\ m_1 \dot{v}_1 = -c_1 x_1 - d_1 v_1 + c_c(x_2 - x_1) + d_c(v_2 - v_1) \\ \dot{x}_2 = v_2 \\ m_2 \dot{v}_2 = -c_c(x_2 - x_1) - c_2 x_2 - d_c(v_2 - v_1) \end{cases} \tag{12}$$

with the initial conditions $x_1(0) = x_2(0) = v_1(0) = v_2(0) = [1, 1]$ (a point interval).

The system is divided in two sub-systems of state (x_1, v_1) and (x_2, v_2), respectively. The coupling is realized with a displacement-displacement approach as follows:

$$\begin{aligned} K_1(x_1, v_1) &= (x_1, v_1), \\ K_2(x_2, v_2) &= (x_2, v_2). \end{aligned} \tag{13}$$

Simulations of the system are depicted in Figs. 4 and 5. In the figures, x_1 is plotted in red, and x_2 in blue; both are plotted within time with a time horizon of 3. These simulations are performed with a simple (guaranteed) Heun scheme in Fig. 4 for illustration purposes. The macro-step is set to $H = 0.05$

in order to amplify the accuracy gains obtained with extrapolation. In Fig. 5, an 4th order Runge–Kutta scheme is used, with a macro-step $H = 0.01$. For both cosimulations, the cross-Picard operator took between 13 and 14 iterations to compute when using macro-steps of size $H = 0.05$, and 8 to 9 iterations for macro-steps of size $H = 0.01$. The cosimulation with extrapolation is performed with an interpolation of order 3 in both macro-step cases.

5.2 Industrial 11-room house heating case study

This case study, proposed by the Danish company Seluxit, aims at controlling the temperature of an eleven rooms house, heated by geothermal energy. The *continuous* dynamics of the system is the following:

$$\frac{\mathrm{d}}{\mathrm{d}t}T_i(t) = \sum_{j=1}^{n} A^d_{i,j}(T_j(t) - T_i(t)) + B_i(T_{\mathrm{env}}(t) - T_i(t)) + H^v_{i,j}.v_j \quad (14)$$

The temperatures of the rooms are the T_i. The matrix A^d contains the heat transfer coefficients between the rooms, matrix B contains the heat transfer coefficients between the rooms and the external temperature, set to $T_{\mathrm{env}} = 10\,^{\circ}\mathrm{C}$ for the computations. The control matrix H^v contains the effects of the control on the room temperatures, and the control variable is here denoted by v_j. We have $v_j = 1$ (resp. $v_j = 0$) if the heater in room j is turned on (resp. turned off). We thus have $n = 11$ and $N = 2^{11} = 2048$ switching modes.

Note that the matrix A^d is parameterized by the open of closed state of the doors in the house. In our case, the average between closed and open matrices was taken for the computations. The controller has to select which heater to turn on in the eleven rooms. Due to a limitation of the capacity supplied by the geothermal device, the 11 heaters cannot be turned on at the same time. In our case, we limit to 4 the number of heaters that can be on at the same time.

We choose to simulate the system on a given sequence of switched modes, for a time horizon $T = 150$ minutes and initial conditions $T_i(0) = [20, 21]$ for all $i = 1, \ldots, 11$. We compare, for different cosimulation methods, the computation time and final area covered at final time in Table 1. The final area is representative of the accuracy of the method. More precisely, at the end of the simulation, we obtain a set of intervals $T_i(150) = [T_i^{f,\min}, T_i^{f,\max}]$. The final area is $\mathcal{A}(150) = (T_1^{f,\max} - T_1^{f,\min}) \times (T_2^{f,\max} - T_2^{f,\min}) \times \cdots \times (T_{11}^{f,\max} - T_{11}^{f,\min})$. Using different simulation methods, we obtain different values for the area. A smaller value means that the simulation is more accurate. The system of dimension 11 is naturally divided in two subsystems of dimension 5 and 6 (as in [38,40]) that can be cosimulated using the above procedure. We perform standard simulations, cosimulations without interpolation, and cosimulation with interpolation (of order 3), for two different numerical schemes (Heun and Runge–Kutta 4).

Table 1 Simulation results for the 11-room case study

Scheme	Computation time (s)	Final area (m^2)
HEUN	7.96	0.2165
Co-HEUN	5.95	0.2407
Co-HEUN-interp	27.05	0.2335
RK4	27.60	0.1821
Co-RK4	17.87	0.1932
Co-RK4-interp	122.17	0.1854

5.3 Discussion

In the first case study, the system has to be expressed as a system of dimension 5 (taking the time as a fifth variable, which derivative is equal to 1). The subsystems are expressed as systems of dimension 3. Thus, in terms of computation time, given the low dimension of the example, no gains are made, since the cross-Picard operator iterations take most of the simulation time. We, however, notice a substantial accuracy improvement using the extrapolation of inputs, which is, unsurprisingly, consistent with the results of [6].

In the second case study, given the sequential implementation of this prototype, the computation times show encouraging results, and the accuracy of the methods is comparable. The dynamics of this example being contractive, we observe a good accuracy for all the different methods since they all manage to capture the contractive behavior of the system. However, the interpolation does not show interesting time gains in this case. Higher dimensions and more complex dynamics seem to be more appropriate for using this approach.

In conclusion, given the complexity of the LTE computation, our recommendations would be to use cosimulation for systems of dimensions exceeding 6. The interpolation should be used only if the time horizon to consider is long enough to observe significant growth of the box areas.

6 Conclusion and future works

In this paper, a guaranteed cosimulation method is proposed. The main ingredient of our approach is the cross-Picard operator, which allows to compute, using only local computations, a (safe) over-approximation of the global state of the system. The cross-Picard operator itself relies on the possibility of considering bounded perturbations. Given a sub-system, its safety is verified by considering

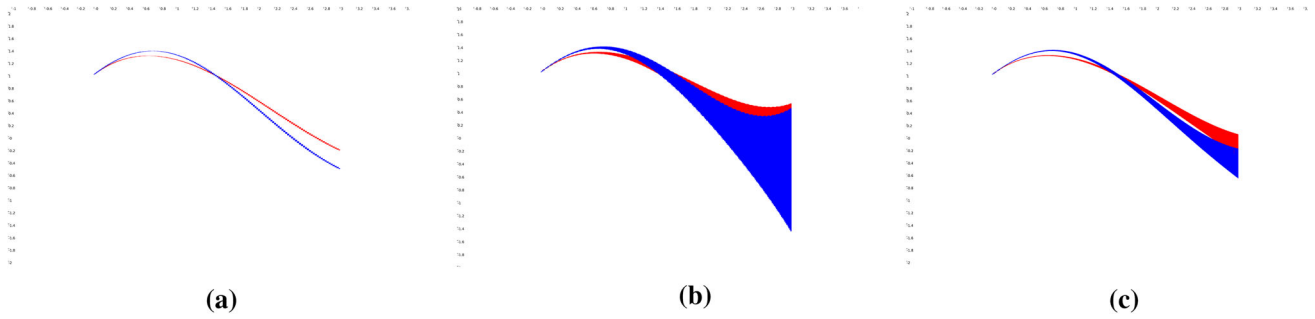

Fig. 4 Guaranteed simulations of the spring case study using a Heun scheme with tolerance 10^{-6} and macro-step $H = 0.05$: **a** global simulation; **b** cosimulation with constant extrapolation; **c** cosimulation with guaranteed extrapolation based on interpolation

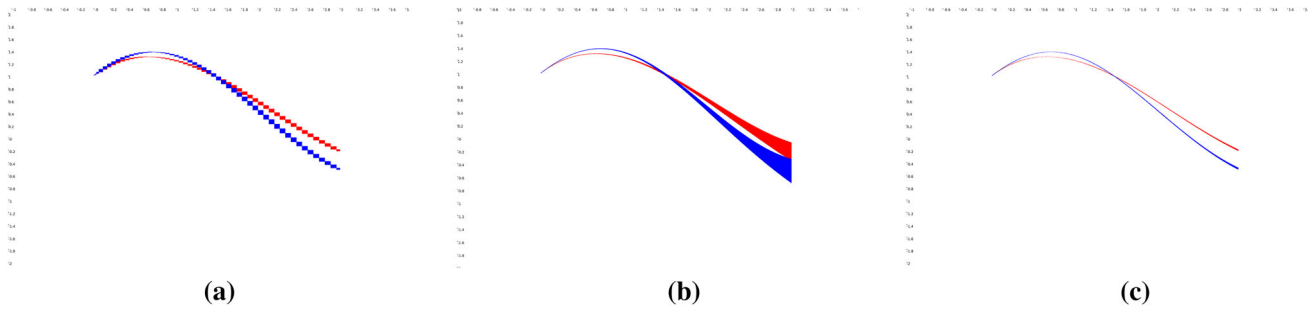

Fig. 5 Guaranteed simulations of the spring case study using a RK4 scheme with tolerance 10^{-8} and macro-step $H = 0.01$: **a** global simulation; **b** cosimulation with constant extrapolation; **c** cosimulation with guaranteed extrapolation based on interpolation

the other sub-systems as bounded perturbations, so that an over-approximation is determined, by iterating over the sub-systems, verifying that all the sub-systems do stay in their bounded (perturbation) set. Cosimulation then allows to update to perturbation sets to consider over the macro-steps. These sets can furthermore be replaced by a guaranteed extrapolation function, allowing to improve the accuracy of the method, substantially in some cases, marginally in others. The cosimulation algorithm thus has to be properly chosen in accordance to the case-study. Some practical details are presented, such as the adaptive macro-step which ensures the success of the procedure, as well as some pre-computations fastening the cross-Picard operator computation. Numerical applications are presented, showing the applicability of the method on an industrial case study.

In a broader setting, the decomposition of the system can have a crucial role in the success of the method and should be performed carefully. Industrial case studies can be designed in a component-based way, providing natural decompositions in sub-systems, but if the system is written as a large system of equations, the decomposition should be chosen so as to minimize the number of overlapping states, thus facilitating the computation of the cross-Picard operator. Indeed, the main limit of our procedure is the iterative computations involved in the cross-Picard operator. Less interactions mean faster convergence of the cross-Picard computations. Furthermore, the macro-step size should be chosen so that time-step adaptation is avoided in order to avoid useless cross-Picard computations.

Our future work will be devoted to the parallel implementation and distribution of a tool containing the presented methods, as well as applications to more case studies. We would also like to apply these methods to other domains, such as control synthesis. Since guaranteed simulation (or reachability analysis) is required in several symbolic and guaranteed control synthesis methods, we would like to implement our method in one of these tools, possibly with compositional principles as well, in order to get closer to industrial scale applications with such methods.

References

1. Alexandre dit Sandretto, J., Chapoutot, A.: DynIbex. https://perso.ensta-paris.fr/~chapoutot/dynibex/
2. Alexandre dit Sandretto, J., Chapoutot, A.: Validated explicit and implicit Runge–Kutta methods. Reliab. Comput. **22**, 79 (2016)
3. Althoff, M.: Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In: Hybrid Systems: Computation and Control, pp. 173–182 (2013)
4. Althoff, M., Stursberg, O., Buss, M.: Verification of uncertain embedded systems by computing reachable sets based on zonotopes. In: Proceedings of the 17th IFAC World Congress, vol. 41(2), pp. 5125–5130 (2008)
5. Ames, W.F.: Numerical Methods for Partial Differential Equations. Academic Press, Cambridge (2014)

6. Arnold, M., Clauß, C., Schierz, T.: Error analysis and error estimates for co-simulation in FMI for model exchange and co-simulation v2.0. In: Schöps, S., Bartel, A., Günther, M., ter Maten, E., Müller, P. (eds.) Progress in Differential-Algebraic Equations, pp. 107–125. Springer, Berlin (2014)
7. Blanes, S., Casas, F., Murua, A.: Splitting and composition methods in the numerical integration of differential equations. Boletin de la Sociedad Espanola de Matematica Aplicada **45**, 89–145 (2008)
8. Bouissou, O., Chapoutot, A., Djoudi, A.: Enclosing temporal evolution of dynamical systems using numerical methods. In: Brat, G., Rungta, N., Venet, A. (eds.) NASA Formal Methods, LNCS, vol. 7871, pp. 108–123. Springer, Berlin (2013)
9. Bouissou, O., Martel, M.: GRKLib: a guaranteed Runge Kutta Library. In: Scientific Computing, Computer Arithmetic and Validated Numerics (2006)
10. Bouissou, O., Mimram, S., Chapoutot, A.: HySon: set-based simulation of hybrid systems. In: Rapid System Prototyping. IEEE (2012)
11. Broman, D., Brooks, C., Greenberg, L., Lee, E.A.., Masin, M., Tripakis, S., Wetter, M.: Determinate composition of FMUs for co-simulation. In: 2013 Proceedings of the International Conference on Embedded Software (EMSOFT), pp. 1–12. IEEE (2013)
12. Bungartz, H.-J., Schäfer, M.: Fluid–Structure Interaction: Modelling, Simulation, Optimisation, vol. 53. Springer, Berlin (2006)
13. Chen, X., Abraham, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: IEEE 33rd Real-Time Systems Symposium, pp. 183–192. IEEE Computer Society (2012)
14. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification, pp. 258–263. Springer, Berlin (2013)
15. Chen, X., Mover, S., Sankaranarayanan, S.: Compositional relational abstraction for nonlinear hybrid systems. ACM Trans. Embed. Comput. Syst. **16**(5s), 1–19 (2017)
16. Chen, X., Sankaranarayanan, S.: Decomposed reachability analysis for nonlinear systems. In: 2016 IEEE Real-Time Systems Symposium (RTSS), pp. 13–24. IEEE (2016)
17. de Figueiredo, L.H., Stolfi, J.: Self-Validated Numerical Methods and Applications. Brazilian Mathematics Colloquium Monographs. IMPA/CNPq, Rio de Janeiro (1997)
18. dit Sandretto, J.A., Chapoutot, A.: Validated simulation of differential algebraic equations with Runge–Kutta methods. Reliab. Comput. **22**, 57 (2016)
19. Dzetkulič, T.: Rigorous integration of non-linear ordinary differential equations in Chebyshev basis. Numer. Algorithms **69**(1), 183–205 (2015)
20. Eggers, A., Fränzle, M., Herde, C.: SAT modulo ODE: a direct SAT approach to hybrid systems. In: Cha, S., Choi, J.Y., Kim, M., Lee, I., Viswanathan, M. (eds.) Automated Technology for Verification and Analysis. LNCS, vol. 5311, pp. 171–185. Springer, Berlin (2008)
21. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification. LNCS, vol. 6806, pp. 379–395. Springer, Berlin (2011)
22. Gajda, K., Jankowska, M., Marciniak, A., Szyszka, B.: A survey of interval Runge–Kutta and multistep methods for solving the initial value problem. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) Parallel Processing and Applied Mathematics. LNCS, vol. 4967, pp. 1361–1371. Springer, Berlin (2008)
23. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland, March 9–11, 2005, Proceedings, pp. 291–305 (2005)
24. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: state of the art (2017). arXiv preprint arXiv:1702.00686
25. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: a survey. ACM Comput. Surv. **51**(3), 1–33 (2018)
26. Gomes, C., Thule, C., Deantoni, J., Larsen, P.G., Vangheluwe, H.: Co-simulation: the past, future, and open challenges. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems, pp. 504–520. Springer, Cham (2018)
27. Gomes, C., Thule, C., Larsen, P.G., Denil, J., Vangheluwe, H.: Co-simulation of continuous systems: a tutorial (2018). arXiv preprint arXiv:1809.08463
28. Hairer, E., Norsett, S.P., Wanner, G.: Solving Ordinary Differential Equations I: Nonstiff Problems, 2nd edn. Springer, Berlin (2009)
29. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations II: Stiff and Differential- Algebraic Problems, 1st edn. Springer, Berlin (1996)
30. Heitmeyer, C.: On the need for practical formal methods. In: International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, pp. 18–26. Springer, Berlin (1998)
31. Heitmeyer, C., Kirby, J., Labaw, B.: Tools for formal specification, verification, and validation of requirements. In: Proceedings of COMPASS'97: 12th Annual Conference on Computer Assurance, pp. 35–47. IEEE (1997)
32. Hou, G., Wang, J., Layton, A.: Numerical methods for fluid-structure interaction: a review. Commun. Comput. Phys. **12**(2), 337–377 (2012)
33. Immler, F.: Verified reachability analysis of continuous systems. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. LNCS, vol. 9035, pp. 37–51. Springer, Berlin (2015)
34. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis. Springer, Berlin (2001)
35. Jensen, J.C., Chang, D.H., Lee, E.A.: A model-based design methodology for cyber-physical systems. In: 2011 7th International Wireless Communications and Mobile Computing Conference, pp. 1666–1671. IEEE (2011)
36. Kübler, R., Schiehlen, W.: Modular simulation in multibody system dynamics. Multibody Syst. Dyn. **4**(2–3), 107–127 (2000)
37. Kühn, W.: Zonotope dynamics in numerical quality control. In: Hege, H.C., Polthier, K. (eds.) Mathematical Visualization, pp. 125–134. Springer, Berlin (1998)
38. Larsen, K.G., Mikučionis, M., Muniz, M., Srba, J., Taankvist, J.H.: Online and compositional learning of controllers with application to floor heating. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 244–259. Springer, Berlin (2016)
39. Le Coënt, A., Sandretto, J.A., Chapoutot, A., Fribourg, L.: An improved algorithm for the control synthesis of nonlinear sampled switched systems. Formal Methods Syst. Des. **53**(3), 363–383 (2018)
40. Le Coënt, A., Fribourg, L., Markey, N., De Vuyst, F., Chamoin, L.: Compositional synthesis of state-dependent switching control. Theor. Comput. Sci. **750**, 53–68 (2018)
41. Lee, E.A.: Cyber physical systems: Design challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 363–369. IEEE (2008)
42. Lin, Y., Stadtherr, M.A.: Validated solutions of initial value problems for parametric odes. Appl. Numer. Math. **57**(10), 1145–1162 (2007)
43. Lohner, R.J.: Enclosing the solutions of ordinary initial and boundary value problems. In: Kaucher, E., Kulisch, U., Ullrich, Ch. (eds.) Computer Arithmetic, pp. 255–286. Teubner, Stuttgart (1987)

44. Ma, T.-W.: Higher chain formula proved by combinatorics. Electron. J. Comb. **16**(1), N21 (2009)
45. Makino, K., Berz, M.: Rigorous integration of flows and odes using Taylor models. In: Proceedings of the 2009 Conference on Symbolic Numeric Computation, SNC '09, pp. 79–84. ACM, New York (2009)
46. Moore, R.E.: Interval Analysis. Series in Automatic Computation. Prentice Hall, Upper Saddle River (1966)
47. Mullier, O., Chapoutot, A., Sandretto, J.A.D.: Validated computation of the local truncation error of Runge–Kutta methods with automatic differentiation. Optim. Methods Softw. **33**(4–6), 718–728 (2018)
48. Nedialkov, N.S., Jackson, K.R., Corliss, G.F.: Validated solutions of initial value problems for ordinary differential equations. Appl. Math. Comp. **105**(1), 21–68 (1999)
49. Nielsen, C.B., Larsen, P.G., Fitzgerald, J., Woodcock, J., Peleska, J.: Systems of systems engineering: basic concepts, model-based techniques, and research directions. ACM Comput. Surv. **48**(2), 1–41 (2015)
50. Quarteroni, A., Valli, A.: Domain Decomposition Methods for Partial Differential Equations. Oxford University Press, Oxford (1999)
51. Schierz, T., Arnold, M., Clauß, C.: Co-simulation with communication step size control in an fmi compatible master algorithm. In: Proceedings of the 9th International MODELICA Conference; September 3–5; 2012; Munich; Germany, number 076, pp. 205–214. Linköping University Electronic Press (2012)
52. Zienkiewicz, O.C., Taylor, R.L., Nithiarasu, P., Zhu, J.Z.: The Finite Element Method, vol. 3. McGraw-hill, London (1977)

Adrien Le Coënt has been an associate professor at Université Paris-Est Créteil since 2020. Before that, he was a postdoctoral researcher in the Semantics of Hybrid Systems team at ENSTA Paris. From 2017 to 2019, he was a postdoctoral researcher in the Department of Computer Science of Aalborg University. Adrien Le Coënt obtained a Ph.D. in applied mathematics in 2017 from CMLA, ENS Paris-Saclay. Between 2010 and 2014, he was a student at ENS Paris-Saclay, where he obtained a master's degree in Structural Mechanics. His main research interests include control of switched systems, formal verification, and guaranteed simulation.

Julien Alexandre dit Sandretto is associate professor in the Semantics of Hybrid System team at ENSTA Paris, Palaiseau, France. He is also associate member of the Cosynus team, part of the LIX laboratory at Ecole Polytechnique. He received the bachelors degree in Applied Mathematics and Computer Science from the University Joseph Fourier, Grenoble, France, in 2002 and the engineering degree in Industrial Computing and Instrumentation from Polytechnic school of Grenoble, France, in 2006. In 2013, he obtained his Ph.D. degree in Computer Science from University of Nice-Sophia Antipolis, France. His research interests deal with verification methods in presence of uncertainties. He currently focuses on parameter identification, simulation, modeling, and control problems for cyber-physical systems.

Alexandre Chapoutot is an associate professor at ENSTA Paris. He received his master degree in Computer Science from University Pierre et Marie Curie (UPMC) in 2005 and his Ph.D. in Computer Science from Ecole polytechnique in 2008. His research activities are focused on static analysis by abstract interpretation and interval-based methods for the verification of cyber-physical systems and for the analysis of floating-point accuracy in programs.