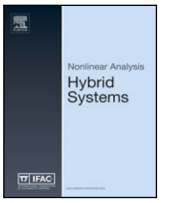
Nonlinear Analysis: Hybrid Systems 54 (2024) 101532

Contents lists available at ScienceDirect

Nonlinear Analysis: Hybrid Systems

journal homepage: www.elsevier.com/locate/nahs



Constraint-driven nonlinear reachability analysis with automated tuning of tool properties

Luca Geretti [a,*], Pieter Collins [b], Pierluigi Nuzzo [c,d], Tiziano Villa [a]

[a] University of Verona, Verona, Italy
[b] Maastricht University, Maastricht, The Netherlands
[c] University of Southern California, Los Angeles (CA), USA
[d] University of California at Berkeley, Berkeley (CA), USA

ARTICLE INFO

Keywords:
Reachability analysis
Tool automation
Safety verification
Optimization
Rigorous numerics

ABSTRACT

The effectiveness of reachability analysis often depends on choosing appropriate values for a set of tool-specific properties which need to be manually tailored to the specific system involved and the reachable set to be evolved. Such *property tuning* is a time-consuming task, especially when dealing with nonlinear systems. In this paper, we propose, instead, a methodology to automatically and dynamically choose property values for reachability analysis along the system evolution, based on the actual verification objective, i.e., the verification or falsification of a set of constraints. By leveraging an initial solution to the reachable set, we estimate bounds on the numerical accuracy required from each integration step to provide a definite answer to the satisfaction of the constraints. Based on these accuracy bounds, we design a cost function which we use, after mapping the property space to an integer space, to search for locally optimal property values that yield the desired accuracy. Results from the application of our methodology to the nonlinear reachability analysis tool ARIADNE show that the frequency of correct answers to constraint satisfaction problems increases significantly with respect to a manual approach.

1. Introduction

Reachability analysis is concerned with the computation of the *reachable set*, i.e., the set of points reached by an initial set that evolves under a system's dynamics. The reachable set of a dynamical system allows reasoning about its behaviors, and determining whether the system satisfies a specification, represented as geometric constraints on the reached points.

For linear systems, tools like SpaceEx [1] and HyPro [2] allow an efficient representation of the system evolution. Computing the reachable set becomes particularly challenging for nonlinear systems. Different approaches are used in the literature; see, for example, the tools KeYmaera X [3], HSolver [4], CORA [5], ARIADNE [6], JuliaReach [7], and Flow* [8]. In this paper, we focus on a numerical approach based on computing over-approximations of the reachable set.

Regardless of the tool, automation can play a critical role toward improving the quality of the representation of the approximate reachable set. Typically, the user needs to provide sensible values for a set of tool *properties*, such as the integration step size or the polynomial order of a set representation. These properties usually affect the quality of the numerical approximation, or toggle specific features, ultimately controlling the over-approximation error. Unfortunately, however, optimal values for these properties are difficult to find until the system under analysis is understood. The user ends up iteratively refining the property values until

* Corresponding author.
E-mail addresses: luca.geretti@univr.it (L. Geretti), pieter.collins@maastrichtuniversity.nl (P. Collins), nuzzo@usc.edu (P. Nuzzo), tiziano.villa@univr.it (T. Villa).

https://doi.org/10.1016/j.nahs.2024.101532
Received 19 September 2023; Received in revised form 20 May 2024; Accepted 21 July 2024
Available online 6 August 2024

an acceptable result is obtained, a task that becomes time-consuming, when done manually with a trial-and-error approach. The situation is exacerbated for systems with nonlinear dynamics, since symbolic approaches are more difficult to pursue, and evaluating the reachable set can be computationally intensive. The overhead due to user interactions with the tool becomes non-negligible, in the worst case requiring to spend hours while observing the behavior of the system, also due to the lack of intuition on the complex interactions among the tool properties. Finally, user interactions may not be feasible in the case of online (dynamic) verification [9] or model predictive monitoring [10] and control [11], since real-time requirements make efficiency more critical than in the offline (static) case.

In this paper, we propose a methodology for the automated choice of the values of a set of tool properties for computing constrained reachability problems in nonlinear systems. Differently from approaches that compute a sequence of converging approximations to the exact result [12], the methodology aims to solve the problem

1. ideally within a single run of execution,
2. with minimal sufficient accuracy, and
3. with no manual tuning by the user.

We use the term *property* in place of the common term *parameter* [13,14] to avoid confusion with the case of parametric systems, where the dynamical system model (not the analysis tool) is partially unspecified and includes time-invariant quantities defined in a range [15]. In our methodology, the goal is to provide a definite answer (positive or negative) to the satisfaction of a set of constraints over a time interval. This objective allows identifying the required accuracy of the reachable set. We then search the property space at each integration step and choose the values that yield the minimum accuracy necessary to prove that each constraint is either satisfied or not. Our approach is *adaptive*, since optimal property values are selected at each integration step and allowed to change, across different reachability problems and different integration steps of the same problem, to accommodate different dynamical responses.

To the best of our knowledge, this is the first approach addressing tool property tuning in a general way for nonlinear systems. Approaches to bound the computation error by a single user-defined value were developed [16] for linear systems, where sets can be represented very efficiently. Recent efforts in CORA [17–19] have been able to identify analytical expressions to control the values of multiple internal numerical properties. For close-to-linear systems, accuracy can still be sufficiently controlled due to the partial ability to maintain a symbolic representation of the error [20]. For systems exhibiting strong nonlinearity, the error has a complex dependency on the dynamics, and set representations are less efficient in space and time, making automated tuning highly desirable. Unfortunately, however, the current tools in this category tend to mostly focus on the automated refinement of the integration step size [21] or support the tuning of multiple properties [22], but not the systematic search of optimal properties guided by cost criteria.

Recently, a methodology to identify multiple properties automatically has been proposed for nonlinear systems [14,23], leveraging analytical formulae for the numerical over-approximation error. However, the approach is tailored to a specific set of properties of interest. Moreover, while aiming to guarantee rigorous bounds on the error, worst-case analytical formulae may result in overly conservative property values and excessive accuracy, as is the case for a lower integration step than is actually necessary. While we use an analytical formula for estimating the error, we propose to numerically validate the conservatism across a range of property values and choose the best ones based on the numerical validation. In doing so, we adopt a heuristic approach, rather than aiming to generate invariants from a formal analysis of the dynamics [24]. Still, our methodology is *generic*, in that it is agnostic of the impact of a given property on the error. Different properties can be appropriately tuned based on the tool or the system. Our approach is therefore *extensible*: it can accommodate new tool features and can be used to assess their impact on system analysis.

We support both positive and negative answers to constraint satisfaction, by leveraging state-of-the-art methods to compute inner approximations [25]. Conversely, an abstraction-based approach using a constraint solver [4] would avoid reachability calculation, but at the same time it would only be able to return a positive answer. We remain numerically rigorous with respect to constraint satisfaction: heuristics are only applied to automatically choose property values, as an alternative to manual selection. To summarize, the main contribution of the paper is a practical methodology for automated reachability calculation for constraint checking. Cornerstones are the identification of an approximate error bound function and a strategy to select results from different valuations of tool properties.

The rest of the paper is organized as follows. We start in Section 2 with the preliminary concepts and definitions. The problem formulation is presented in Section 3, followed by an overview of the algorithm in Section 4. Sections 5 and 6 provide the bulk of the methodology, with Section 7 dedicated to the resulting algorithm. Section 8 discusses property space exploration. Section 9 analyzes some nonlinear systems to illustrate the benefits of the approach. Conclusions are drawn in Section 10.

2. Preliminaries

Let us consider a nonlinear time-variant system

$$\dot{\mathbf{x}} = f(\mathbf{x}, t), \quad f : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N \tag{1}$$

with f (locally) Lipschitz.

We define $R(t)$ as the *evolved set* of the system, i.e., the set of points reached, starting from an initial set I and an initial time T_I, at a specific time $t \geq T_I$ under differential dynamics given by f. Instead we call *reached set* $R(t_1, t_2)$ or *flow tube* the set of points

reached between t_1 and t_2. Therefore $R(t)$ can also be referred to as the *section of the flow tube*. If we identify a finite time interval $\Delta T = [T_I, T_F]$, then we use R with no arguments to refer to the *reachable set* (also called *reachability*) of the system in ΔT implicitly.

Since the exact value of $R(t)$ is not computable in general for nonlinear systems, we are interested in computing approximations to it. Let us define the three approximations of interest:

- $\overline{R}(t)$: an outer-approximation of $R(t)$, i.e., $R(t) \subset \overline{R}(t)$;
- $\underline{R}(t)$: an inner-approximation of $R(t)$, i.e., $\underline{R}(t) \subset R(t)$;
- $\tilde{R}(t)$: an approximation of $R(t)$, such that $\tilde{R}(t) \subset \overline{R}(t)$.

In the following we will also refer to $\tilde{R}$ as the *approximate reachability* and $\overline{R}$ as the *rigorous (over-approximated) reachability*. Moreover, an *uncontrolled reachability* is a reachability obtained using a fixed valuation of the properties, while a *controlled reachability* is the result of our methodology, able to control the over-approximation error along the evolution time in ΔT.

Such approximations will be used to check the *satisfiability* of a set of M nonlinear time-varying constraints

$$C = \{c_m(\mathbf{x}, t) \geq 0,\ m = 0, \dots, M-1\}. \tag{2}$$

The outcome of satisfiability checking of the mth constraint for a given $\hat{\mathbf{x}}, \hat{t}$ pair is a logical value in the $\{\top, \bot, ?\}$ set, with a *positive* answer $\top$ when the $c_m(\hat{\mathbf{x}}, \hat{t}) \geq 0$ predicate is satisfied, a *negative* answer $\bot$ when it is not satisfied, and an *indeterminate* answer "?" when it is not possible to obtain either $\top$ or $\bot$. We say that we provide a *definite answer* when it is not indeterminate.

Moving to sets, with some abuse of notation, we introduce the *constraint evaluation* c_m at a given t using a generic set $S(t)$ as the following interval in the reals:

$$c_m(S(t), t) = c_m([S(t)], [t]) = \left[\underline{c}_m(S(t), t), \overline{c}_m(S(t), t)\right]$$

where we use square brackets to refer to the *bounding box* over-approximation of a set, i.e., a tight coordinate-aligned box enclosing the set. Here we assume that time may also be represented as an interval in rigorous calculations.

The evaluation of constraints during reachability computation represents a constrained Initial Value Problem (IVP). In particular, one where we compute approximations to the reachable set that are the least accurate (i.e., fastest to compute) while providing a definite answer to the satisfiability of the constraints. In the following Section we will provide a formal definition of such a problem.

3. Problem formulation

Our assumptions for the computation of a solution to the constrained IVP can be stated as follows:

- We rely on a reachability tool in which we iteratively perform an *integration step* from a time t_k to a time t_{k+1}, $k = 0, \dots, K$ across the whole ΔT interval;
- Rigorous evolution is performed to compute $\overline{R}(t)$, while $\underline{R}(t)$ can be derived from $\overline{R}(t)$ [25] if necessary.

We also identify a set of Q independent tool properties $\mathcal{P} = \{P_q\}$ that a designer can tune when computing reachable sets. Conventionally, one would adopt a user-defined valuation $\hat{\mathbf{p}} \in \mathrm{dom}(\mathcal{P})$ valid for all $t \in \Delta T$. Our goal is, instead, to automatically tune the property values across integration steps based on the constraints.

Constraint evaluation can be performed using interval arithmetic, which returns an over-approximation of the result. In particular, interval arithmetic can be directly used for $[\overline{R}(t)]$, since the input is already an over-approximation. For $\underline{R}(t)$, instead, we need to guarantee that a negative evaluation is not the result of a spurious point caused by over-approximation. Therefore, $c(\underline{R}(t), t)$ is evaluated point-wise, by progressively splitting the domain of $\underline{R}(t)$ down to a user-defined maximum splitting depth, and evaluating the midpoint of the set resulting from the midpoint of each subdomain. While an evaluation $\left[\underline{c}_m(\underline{R}(t), t), \overline{c}_m(\underline{R}(t), t)\right]$ can be built using this procedure, we are typically interested only in checking that $\underline{c}_m(\underline{R}(t), t) < 0$ holds. This can be performed via splitting until a subdomain is found that satisfies the constraint, the constraint is found infeasible for all points of all the subdomains, or the maximum splitting depth is reached. Remarkably, computing $\underline{R}(t)$ is generally expensive, with an exponential cost in the typical implementation, and the result may still be a very small or empty inner approximation, which would not be effective for our evaluation purposes. Due to these limits in state-of-the-art inner approximation methods, instead of treating the maximum splitting depth as another optimizable property, we rather chose a fixed value of 1. We define the *satisfaction* of c_m in ΔT with

$$\overline{\sigma}_m(\Delta T) = \begin{cases} \top, & \forall t \in \Delta T : \underline{c}_m(\overline{R}(t), t) \geq 0 \\ \bot, & \exists t \in \Delta T : \underline{c}_m(\underline{R}(t), t) < 0 \\ ?, & \text{otherwise.} \end{cases} \tag{3}$$

We use bars above and below σ_m to denote that satisfaction is assessed on rigorous approximations of $R(t)$. While (3) is sound, based on the above considerations on inner approximations, the condition $\underline{c}_m(\underline{R}(t), t) < 0$ may not be effective for falsification, and is often inefficient to compute. Hence, we expand our options for falsification as follows:

$$\overline{\sigma}_m(\Delta T) = \begin{cases} \top, & \forall t \in \Delta T : \underline{c}_m(\overline{R}(t), t) \geq 0 \\ \bot_A, & \exists t \in \Delta T : \overline{c}_m(\overline{R}(t), t) < 0 \\ \bot_S, & \exists t \in \Delta T : \underline{c}_m(\underline{R}(t), t) < 0 \\ ?, & \text{otherwise,} \end{cases} \tag{4}$$

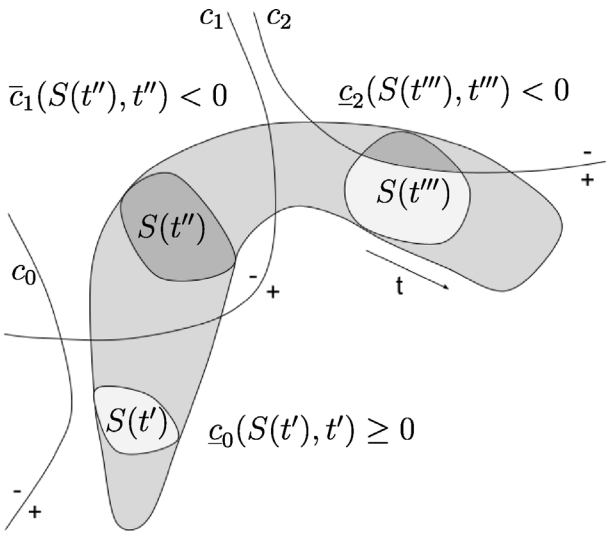


Fig. 1. Representation of different constraint satisfaction scenarios for sets, corresponding to the $\top$, $\bot_A$, and $\bot_S$ conditions.

i.e., we can falsify a constraint using $\overline{R}(t)$ if all points do not satisfy the constraint. $\bot_A$ stands for *false for all points* while $\bot_S$ stands for *false for some points*. For effectiveness and efficiency, we will give priority to $\bot_A$ over $\bot_S$ if the condition for the former is satisfied. In the following, we will use $\bot$ with no subscript when referring to both cases. As an example, Fig. 1 shows a generic reachable flow tube $S(t)$ and three constraints c_0, c_1, and c_2. The three outcomes for $\overline{\sigma}_m(\Delta T)$ are shown, where sections at times t', t'', and t''', respectively, represent cases for $\top$ on c_0, $\bot_A$ on c_1, and $\bot_S$ on c_2.

We measure the size of the sets and relate them to computation errors by relying on the normalization of the *volume of the bounding box* of a set

$$\beta(t) = \sqrt[N]{\prod_{i=0}^{N-1} |[R(t)]_i|}, \tag{5}$$

i.e., the normalized product of the widths over each dimension i of the box. We correspondingly use the notation $\tilde{\beta}(t)$ for $\tilde{R}(t)$, $\overline{\beta}(t)$ for $\overline{R}(t)$, and $\underline{\beta}(t)$ for $\underline{R}(t)$.

We further define $\overline{\chi}_m(t)$ as the *expansion* or *contraction* factor from $\overline{R}(t)$ or $\underline{R}(t)$, respectively, to the boundary of the mth constraint that still gives the same constraint satisfaction condition:

$$\overline{\chi}_m(t) = \max_{\gamma\geq 1}\gamma \ \text{ s.t.} \begin{cases} \underline{c}_m(\gamma[\overline{R}(t)],t)\geq 0, & \underline{c}_m(\overline{R}(t),t)\geq 0 \\ \overline{c}_m(\gamma[\overline{R}(t)],t)<0, & \overline{c}_m(\overline{R}(t),t)<0 \\ \underline{c}_m(\frac{1}{\gamma}[\underline{R}(t)],t)<0, & \underline{c}_m(\underline{R}(t),t)<0 \end{cases} \tag{6}$$

with $\gamma[S(t)]$ denoting the bounding box, where each width is scaled by a factor of γ, i.e., the actual volume becomes $\overline{\chi}_m^N(t)\overline{\beta}^N(t)$ or $\underline{\beta}(t)^N/\underline{\chi}_m^N(t)$. Then, $\chi_m(t)$ would be the expansion or contraction factor from $R(t)$: in practice it represents how much a set can be enlarged or shrinked while still be useful for constraint satisfaction.

We can finally introduce the *local robustness* for constraint satisfaction as the *available* normalized volume with respect to any additional error in the evolved set:

$$\overline{\rho}_m(t) = \begin{cases} \overline{\chi}_m(t)\overline{\beta}(t) - \overline{\beta}(t), & \overline{\sigma}_m(\Delta T) = \{\top, \bot_A\} \\ \underline{\beta}(t) - \underline{\beta}(t)/\overline{\chi}_m(t), & \overline{\sigma}_m(\Delta T) = \bot_S \end{cases} \tag{7}$$

where the condition for $\overline{\sigma}_m(\Delta T)$ is fixed for all t; we set $\overline{\rho}_m(t) = 0$ whenever the corresponding condition is not satisfied. Then, $\rho_m(t)$ represents the robustness with respect to $R(t)$, as a function of $\beta(t)$ and $\chi_m(t)$. The robustness can be related to the distance (maximum for $\bot$, minimum for $\top$) between the constraint boundary and the set. The *global robustness* becomes

$$\overline{\rho}_m(\Delta T) = \begin{cases} \min_{t\in\Delta T}\overline{\rho}_m(t), & \overline{\sigma}_m(\Delta T) = \top \\ \max_{t\in\Delta T}\overline{\rho}_m(t), & \overline{\sigma}_m(\Delta T) = \bot \end{cases} \tag{8}$$

i.e., we pick the worst-case margin for $\top$. For $\bot$ it is sufficient to find one t for which a negative value is identified, hence we can focus on controlling the error with respect to the largest margin available. Our problem is finally formalized as follows:

Constraint-Driven Tool Property Tuning. *Given the system in (1), the constraints in (2), and a set of Q tool properties $\mathcal{P} = \{P_q\}$, $q = 1, \dots, Q$, we want to solve the constrained IVP in a time interval ΔT by looking for*

$$\underset{\mathbf{p}(t) \in dom(\mathcal{P})}{\operatorname{argmin}} \left\{ \overline{\rho}_m(\Delta T) \right\} \quad \text{s.t. } \underline{\sigma}_m(\Delta T) \neq ? \ \forall\, m = 0, \dots, M-1 \tag{9}$$

The rationale is that the *fastest solution* to the IVP is obtained by minimizing the robustness that achieves a definite answer to all constraints, since that translates into selecting property values that result in larger errors, hence less computationally demanding operations.

4. Overview of the algorithm

We summarize the steps of the algorithm resulting from our methodology, which is fully described in Section 7. For each step, we provide the Section in which the step is discussed in square brackets.

1. Identify a random set of property valuations Π [Section 8];
2. Pre-analyze the system on one element of Π by computing the approximate and rigorous reachabilities and using them to:
   (a) remove constraints for which an answer is already identified during the rigorous case [Section 6];
   (b) identify the expected answer for each remaining constraint [Section 5];
   (c) construct expressions for the bounds on the growth of the computation error [Section 5];
3. From T_I to T_F, perform the kth step of controlled reachability:
   (a) $\forall\, \mathbf{p} \in \Pi$ concurrently:
      i. Compute one rigorous integration step [Section 9.3];
      ii. Evaluate the satisfaction of the bounds on the error growth [Section 5];
   (b) Adopt the values providing the minimum accuracy that satisfies the bounds [Section 5];
   (c) Check answers to constraint satisfiability [Section 6];
   (d) Update the Π population as a function of the points that satisfy the bounds with minimal margin [Section 8];
4. If new answers are found, remove the corresponding constraints and repeat from (2), otherwise
5. Return the reached set and the answers for the satisfaction of each constraint.

Computing reachabilities depends on the reachability tool, and consequently, is not the focus of this paper. We still provide details about the integration step in Section 9.3 to understand the role of the chosen properties. In the following, we detail the proposed solution strategy for problem (9).

5. Derivation of the criteria for robustness minimization

Solving (9) requires us to predict the answer to constraint satisfiability in advance, in order to be able to minimize the robustness margin with respect to constraint satisfaction. To make the prediction, we can compute $\tilde{R}(\hat{\mathbf{p}}, t)$ and $\overline{R}(\hat{\mathbf{p}}, t)$ and perform a difference analysis, where $\hat{\mathbf{p}} \in \mathrm{dom}(\mathcal{P})$ is a random point fixed for all times. We privilege a random point over a "default" valuation of the tool properties (or even the midpoint of $\mathrm{dom}(\mathcal{P})$) to make no assumptions on the initial property valuations. When using a Taylor model approximation to the flow tube, $\tilde{R}(t)$ can be obtained by discarding error terms at all integration steps. We have that $\tilde{R}(t) \not\subset R(t)$ and $\tilde{R}(t) \not\supset R(t)$, while $\tilde{R}(t) \subset \overline{R}(t)$ is guaranteed by construction. By comparing $\tilde{R}(t)$ and $\overline{R}(t)$, we can estimate the growth of the over-approximation error along evolution and control it to yield the minimum robustness allowed by property valuations.

If the approximate reachability $\tilde{R}(\hat{\mathbf{p}}, t)$ is used, computing the satisfaction as in (4) translates into a *prescription* of a given satisfaction outcome as follows:

$$\tilde{\sigma}_m(\Delta T) = \begin{cases} \top, & \forall t \in \Delta T : \underline{c}_m(\tilde{R}(\hat{\mathbf{p}}, t), t) \geq 0 \\ \bot_A, & \exists t \in \Delta T : \overline{c}_m(\tilde{R}(\hat{\mathbf{p}}, t), t) < 0 \\ \bot_S, & \exists t \in \Delta T : \underline{c}_m(\tilde{R}(\hat{\mathbf{p}}, t), t) < 0 \end{cases} \tag{10}$$

Notably, when prescribing $\bot_A$ for the mth constraint, we can avoid the computation of $\underline{R}(t')$ for that constraint whenever $\underline{c}_m(\underline{R}(t'), t') < 0$ is expected to hold for any $t' \in \Delta T$. Due to the cost of inner approximations, this is a decisive performance advantage brought by pre-analysis. Equivalent expressions using $\tilde{R}(\hat{\mathbf{p}}, t)$ are obtained for $\tilde{\chi}_m(t)$ from (6) and for $\tilde{\rho}_m(t)$ from (7).

In the following Subsections, we progressively refine the approximate formulation for controlling the error from a basic version to the complete one.

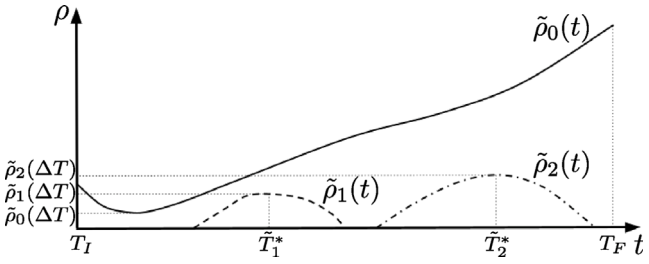


Fig. 2. Robustness of the trajectory in Fig. 1 with respect to each of the three constraints.

5.1. Bounding the over-approximation error

First, we introduce the notion of approximate upper bound on the maximum time for checking a constraint:

$$\tilde{T}_m^* = \begin{cases} T_F, & \tilde{\sigma}_m(\Delta T) = \top \\ \underset{t \in \Delta T}{\operatorname{argmax}}\, \tilde{\rho}_m(t), & \tilde{\sigma}_m(\Delta T) = \bot \end{cases} \tag{11}$$

i.e., while we need to check $\top$ for all times, for $\bot$ we assume that a success after the time of the maximum robustness is unlikely if not already obtained earlier. Fig. 2 shows $\tilde{T}^*$ as well as the local and global $\tilde{\rho}$ from the trajectory and constraints of Fig. 1, where $S(t) = \tilde{R}(t)$.

Since we compute the evolution using outer-approximations to $R(t)$, let us consider the accumulated over-approximation error at time t as the quantity

$$e(t) = \overline{\beta}(t) - \beta(t) \tag{12}$$

where $\beta(t)$ is the unknown exact normalized volume. Turning to evolution by discrete integration steps, and making explicit the fact that we choose a specific point $\mathbf{p} \in \mathrm{dom}(\mathcal{P})$ at each t_k, we want to guarantee that

$$e(\mathbf{p}(t_k), t_k) \leq \rho_m(t_k) \begin{cases} \forall t_k \in \Delta T, & \tilde{\sigma}_m(\Delta T) = \top \\ t_k = \tilde{T}_m^*, & \tilde{\sigma}_m(\Delta T) = \bot \end{cases} \tag{13}$$

However, since the error tends to increase over time, we also want to control its growth to ultimately satisfy (13). To predict the growth with respect to time, we perform a difference analysis using an uncontrolled rigorous reachability $\overline{R}(\hat{\mathbf{p}}, t)$ under the same $\hat{\mathbf{p}}$ used for $\tilde{R}(\hat{\mathbf{p}}, t)$. This result is different from the controlled rigorous reachability $\overline{R}(t)$, where the property values may change with time. By using approximate results instead of exact ones, we find

$$\tilde{e}(\hat{\mathbf{p}}, t_a) = \overline{\beta}(\hat{\mathbf{p}}, \tilde{t}_a) - \tilde{\beta}(\hat{\mathbf{p}}, t_a) \tag{14}$$

where we use a in place of k since time points between controlled rigorous reachability and uncontrolled approximate reachability may differ. For similar reasons, we introduce $\tilde{t}_a$ as the time closest to t_a in the uncontrolled rigorous reachability, since even with the same tool properties, time points between uncontrolled approximate and rigorous reachabilities may differ, i.e., the integration step size may be adaptive as a function of other properties.

By dividing both sides of (13) by $\tilde{e}(\hat{\mathbf{p}}, \tilde{t}_k)$, with $\tilde{t}_k$ the closest time point to t_k in the t_a series, and approximating $\rho_m(t_k)$ with $\tilde{\rho}_m(\hat{\mathbf{p}}, \tilde{t}_k)$, we obtain

$$\frac{e(\mathbf{p}(t_k), t_k)}{\tilde{e}(\hat{\mathbf{p}}, \tilde{t}_k)} \leq \frac{\tilde{\rho}_m(\hat{\mathbf{p}}, \tilde{t}_k)}{\tilde{e}(\hat{\mathbf{p}}, \tilde{t}_k)} \tag{15}$$

From the right hand side of (15) we can identify a strictly positive upper bound on the error ratio:

$$\alpha_m = \min \frac{\tilde{\rho}_m(\hat{\mathbf{p}}, t_a)}{\tilde{e}(\hat{\mathbf{p}}, t_a)} \begin{cases} \forall t_a \in \Delta T, & \tilde{\sigma}_m(\Delta T) = \top \\ t_a = \tilde{T}_m^*, & \tilde{\sigma}_m(\Delta T) = \bot \end{cases} \tag{16}$$

Using α_m in place of the right hand side of (15), substituting (14) and (12) with $\tilde{\beta}(\hat{\mathbf{p}}, \tilde{t}_k)$ in place of the exact normalized volume, we obtain

$$\overline{\beta}(\mathbf{p}(t_k), t_k) - \tilde{\beta}(\hat{\mathbf{p}}, \tilde{t}_k) \leq \alpha_m \left(\overline{\beta}(\hat{\mathbf{p}}, \tilde{t}_k) - \tilde{\beta}(\hat{\mathbf{p}}, \tilde{t}_k) \right) \tag{17}$$

and consequently, the following constraint for optimization:

$$\left(\alpha_m + 1\right) \tilde{\beta}(\hat{\mathbf{p}}, \tilde{t}_k) - \alpha_m \overline{\beta}(\hat{\mathbf{p}}, \tilde{t}_k) - \overline{\beta}(\mathbf{p}(t_k), t_k) \geq 0 \tag{18}$$

which for compactness we refer to as $s'_m(\mathbf{p}, t_k) \geq 0$, leaving the dependency of $\mathbf{p}$ from t_k implicit from now on.

Fig. 3, related to Fig. 2, visually represents the error under a given $\overline{R}(t)$, assuming that $S(t) = \tilde{R}(t)$ in Fig. 1. For visualization purposes, the y axis is magnified with respect to Fig. 2. The figure shows the α_m scaling of $\tilde{e}(t)$ which bounds the error of controlled

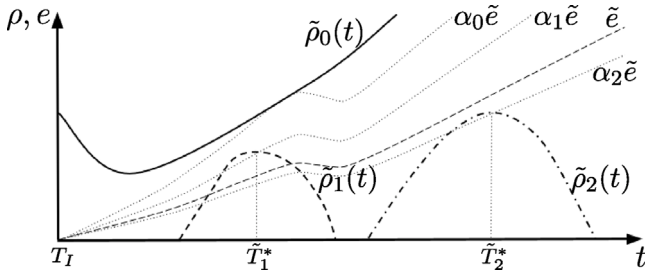


Fig. 3. Example of error $\tilde{e}$ and its modulation with α_m for all constraints in Fig. 1.

reachability by $\tilde{\rho}_m(t)$. We observe that c_2 is the constraint that gives the stricter bound on the error growth. Moreover, α_0 is not determined by $\tilde{\rho}_0(\Delta T)$ according to the example curve derived from $\tilde{e}$.

Given a selection of points $\Pi \subset \mathrm{dom}(\mathcal{P})$ that represent the explored valuations of tool properties, the minimization in problem (9) can be performed approximately by finding at each step k

$$\underset{\mathbf{p}\in\Pi}{\operatorname{argmin}} \sum_{m=0}^{M-1} s'_m(\mathbf{p}, t_k) \quad \text{s.t.} \quad s'_m(\mathbf{p}, t_k) \geq 0 \ \ \forall\ m = 0, \dots, M-1 \tag{19}$$

5.2. Relaxing the error bounds

Since Π is finite, there may not be a solution to (19) at any given step k. However, a failure for step k may still be compensated on the following steps by a success with higher than necessary accuracy. Therefore, we do not want our search to fail due to any violated constraints. We then accept some *soft failures*, i.e., $s'_m(\mathbf{p}, t_k) < 0$ for some m. To do so, we relax the error constraints in (19) and modify the cost function so that the minimum is taken over a pair of objectives. First, we aim to minimize the number of soft failures in (19). Then, we minimize our robustness objective. More formally, we rewrite (19) as an unconstrained multi-objective optimization problem

$$\underset{\mathbf{p}\in\Pi}{\operatorname{argmin}} \left(\#\{s'_m(\mathbf{p}, t_k) < 0\}, \sum_{m=0}^{M-1} |s'_m(\mathbf{p}, t_k)| \right) \tag{20}$$

where priority is given on the left component of the cost function. We use # to denote the *cardinality* of a set. Moreover, we use absolute values in the summation on the right, since some of the addends may be negative.

5.3. Controlling offsets in the robustness estimates

An offset $s'_m(\mathbf{p}, t_k) \neq 0$ in any of the estimates in (20), which is usually observed in practice, may produce a drift in the cost function. A negative drift leads to failure in providing a definite answer to constraint satisfaction. A positive drift leads to overly accurate computation of rigorous approximations. We therefore aim to compensate the offsets along the remaining time to T_m^*, by redefining the *soft constraint* function as follows

$$s_m(\mathbf{p}, t_k) = s'_m(\mathbf{p}, t_k) + \left(t_k - t_{k-1}\right) \sum_{j=0}^{k-1} \frac{s_m(\mathbf{p}, t_j)}{T_m^* - t_j} \tag{21}$$

with $s_m(\mathbf{p}, t_0) = 0$, which distributes the kth offset across the remaining time, proportionally to the time step performed.

In (21) we shift the minimum for the objective value, favoring solutions that perform in the opposite direction to the offset. While the compensation converges to a negative offset from below, preventing it from reaching positive values, this control strategy is still acceptable numerically. In fact, relaxing robustness constraints into soft constraints tends to produce an overall positive drift. Additionally, the β measure, being obtained from the bounding box, is an over-approximation that encourages operating with more accurate sets than necessary. Based on these considerations, we rewrite our robustness minimization problem as follows

$$\underset{\mathbf{p}\in\Pi}{\operatorname{argmin}} \left(\#\{s_m(\mathbf{p}, t_k) < 0\}, \sum_{m=0}^{M-1} |s_m(\mathbf{p}, t_k)| \right) \tag{22}$$

5.4. Pruning constraints that are not satisfiable

While we prescribe a definite satisfaction result, over time some c_m may not be satisfiable anymore, leading to an indeterminate result. We define the disjoint subsets $\tilde{C}_k^{\top}$, $\tilde{C}_k^{\perp_A}$ and $\tilde{C}_k^{\perp_S}$ of constraints based on the expected satisfaction outcome. These sets shrink as k increases. Equivalently, if $\mathcal{M}_k$ is the union of the constraint indexes, then $\mathcal{M}_k$ also shrinks. We instead denote by $C^{\top}$ and $C^{\perp}$ the

(increasing) sets of constraints for which an answer to the satisfaction problem is available. For $\top$, an answer can only be obtained after T_F.

We define a *hard constraint* function related to the mth constraint for point $\mathbf{p}$ at step k as

$$h_m(\mathbf{p}, t_k) = \begin{cases} \underline{c}_m(\overline{R}(t_{k-1}, t_k), t_k), & \tilde{\sigma}_m(\Delta T) = \top \\ T_m^* - t_k, & \tilde{\sigma}_m(\Delta T) = \bot \end{cases} \tag{23}$$

where if $h_m(\mathbf{p}, t_k) < 0$, then the constraint is removed from $\mathcal{M}_k$. Note that the reached set in the whole integration time interval $\overline{R}(t_{k-1}, t_k)$ must be used for rigorousness. Due to the approximations adopted in formulating our robustness objective, some property assignments may give a better objective value (and lower failed soft constraints) while failing more hard constraints. We account for this effect by reformulating the optimization problem as follows

$$\underset{\mathbf{p}\in\Pi}{\operatorname{argmin}} \begin{pmatrix} \#_{m\in\mathcal{M}_k}\{h_m(\mathbf{p}, t_k) < 0\}, \\ \#_{m\in\mathcal{M}_k}\{s_m(\mathbf{p}, t_k) < 0\}, \\ \sum_{m\in\mathcal{M}_k} |s_m(\mathbf{p}, t_k)| \end{pmatrix} \tag{24}$$

where we prioritize, from top to bottom, the number of constraint violations (*hard failures*) versus the number of violation of the error bounds (soft failures), and the minimization of the overall robustness objective. The three components of the new cost function represent the *satisfiability* (i.e, not having to discard the constraint from now on), *effectiveness* (i.e., having respected the error bound), and *efficiency* (i.e., having respected the error bound with minimum accuracy) criteria, respectively. Clearly, a natural priority exists in satisfiability against effectiveness on one side, and in effectiveness against efficiency on the other.

In our problem formulation, we continue reachability even if all points in Π lead to one (or more) hard failures. Only when $\mathcal{M}_k = \emptyset$ we terminate reachability early since the cost function is no longer defined and there are no constraints on the accuracy of the result.

6. Rigorous constraint checking

While (24) provides an approximate rule to exclude constraints that are no longer satisfiable, to solve problem (9) we still need to rigorously verify or falsify the constraints. Success in satisfying the prescription given by $\tilde{\sigma}_m(\Delta T)$ is determined by the following checks:

$$\begin{cases} t_k = T_F: & c_m \in \tilde{C}_k^{\top}, & \tilde{\sigma}_m(\Delta T) = \top \\ \forall t_k: & \overline{c}_m(\overline{R}(t_k), t_k) < 0, & \tilde{\sigma}_m(\Delta T) = \bot_A \\ \forall t_k: & \underline{c}_m(\overline{R}(t_k), t_k) < 0 \wedge & \tilde{\sigma}_m(\Delta T) = \bot_S \\ & \underline{c}_m(\underline{R}(t_k), t_k) < 0, & \end{cases} \tag{25}$$

Success for $\tilde{\sigma}_m(\Delta T) = \top$ is already a result of h_m not failing at all times, and can be deduced trivially after T_F if the constraint is still in $\tilde{C}_k^{\top}$. For $\tilde{\sigma}_m(\Delta T) = \bot_A$, the condition for falsification is checked at each time, since it is inexpensive. For $\tilde{\sigma}_m(\Delta T) = \bot_S$, instead, we first perform a fast check on $\overline{R}$ to identify whether an inner approximation with negative constraint satisfaction is possible. If the preliminary check passes, then we perform the expensive construction of $\underline{R}$ and check against it.

Finally, by recognizing that uncontrolled rigorous reachability can already give answers to the satisfaction of some constraints, we introduce simplified checks to possibly remove those constraints. During pre-analysis, as we generate the constraint prescription, we do not rely on $\underline{R}(t_k)$ for efficiency. Still, we can already identify $\top$ and $\bot_A$ answers:

$$\begin{cases} \forall t_k: & \underline{c}_m(\overline{R}(t_k), t_k) > 0 \;\rightarrow\; c_m \in C^{\top} \\ \exists t_k: & \overline{c}_m(\overline{R}(t_k), t_k) < 0 \;\rightarrow\; c_m \in C^{\bot} \end{cases} \tag{26}$$

consequently reducing $\mathcal{M}_0$ for controlled reachability.

7. Algorithm

The complete constraint-driven reachability analysis flow is given in Alg. 1. It is a loop that terminates when a determinate answer to all the constraints is found or the number of indeterminates could not be reduced further during the last iteration. While Sections 3 to 6 dealt with only one round of iteration, we recognize that under a finite exploration of the search space, and given the approximate nature of the pre-analysis, multiple rounds are usually beneficial. We start by initializing the satisfaction results to all indeterminates (line 1). Each iteration computes uncontrolled reachabilities, both approximate $\tilde{R}_u$ and rigorous $\overline{R}_u$ (lines 7 and 9). The latter in particular is terminated early if the radius of the set (i.e., half of the infinite norm of the bounding box of the set) is larger than half of the radius of the bounding box B of $\tilde{R}$. This check is necessary since we have no guarantee that the property valuations chosen are sufficient to avoid blowup of the error. In line 9, we obtain S_u using the simplified checks of (26). The two reachability analyses are run using a random but common property assignment p. The results are analyzed (line 10) according to the formulae in Section 5 to obtain the vectors of values for σ and α (one per constraint) and the one for β (one per integration step with respect to $\tilde{R}_u$).

Controlled reachability is performed in line 19 using the h and s constraints, where an early termination of $\overline{R}_u$ (rigorous uncontrolled reachability), identified by $T_E < T_F$, prevents the $\top$ case from being used both for error control and constraint checking.

For $\overline{R}$ the ideal result from all rounds would be the intersection of all reachable sets found. Unfortunately, the intersection of nonconvex nonlinear sets, depending on the representation, may be expensive to compute. Instead we focus on the most accurate reachable set computed, by replacing in line 20 the original reachable set if the ending time is higher or if the ending time is the same but the number of steps is higher (under the assumption that smaller steps yield better results). Satisfaction results are merged by setting any definite answer over the previously indeterminate ones (line 21). Finally, the constraints C are reduced each time we update the satisfaction answers, maintaining only those with an indeterminate answer (line 22).

Algorithm 1 Constraint-Driven Reachability Analysis

Given the system sys, initial set I, initial time T_I, final time T_F and constraints C:

1: $S = \text{initialize\_satisfaction}(C)$
2: $\overline{R} = \emptyset$
3: $M_i = |C| + 1$
4: **while** $\text{num\_indeterminates}(S) < M_i$ **do**
5: $\quad M_i = \text{num\_indeterminates}(S)$
6: $\quad p = \text{random\_search\_point}()$
7: $\quad \tilde{R}_u = \text{approximate\_reach}(sys, I, T_I, T_F, p)$
8: $\quad B = \text{bounding\_box}(\tilde{R}_u)$
9: $\quad [S_u, \overline{R}_u] = \text{uncontrolled\_reach}(sys, I, T_I, T_F, C, B, p)$
10: $\quad [\sigma, \beta, \alpha] = \text{preanalyze}(\tilde{R}_u, \overline{R}_u, C)$
11: $\quad \overline{R} = \text{choose}(\overline{R}, \overline{R}_u)$
12: $\quad S = \text{merge}(S, S_u)$
13: $\quad C = \text{reduce\_from}(C, S)$
14: $\quad$ **if** $\text{num\_indeterminates}(S) = 0$ **then break**
15: $\quad$ **end if**
16: $\quad M_i = \text{num\_indeterminates}(S)$
17: $\quad T_E = \text{ending\_time}(\overline{R}_u)$
18: $\quad [h, s] = \text{build\_control\_constraints}(C, \sigma, \beta, \alpha, T_I, T_F, T_E)$
19: $\quad [S_c, \overline{R}_c] = \text{controlled\_reach}(sys, I, T_I, T_F, C, B, p, h, s)$
20: $\quad \overline{R} = \text{choose}(\overline{R}, \overline{R}_c)$
21: $\quad S = \text{merge}(S, S_c)$
22: $\quad C = \text{reduce\_from}(C, S)$
23: $\quad$ **if** $\text{num\_indeterminates}(S) = 0$ **then break**
24: $\quad$ **end if**
25: **end while**
26: **return** $[S, \overline{R}]$

In Alg. 2 we describe the controlled_reach procedure. Starting from a search point p, we generate an initial family of points Π (line 3). For each step, we perform a concurrent evaluation of all points in Π. For each point we set the algorithm to use the corresponding property values (line 9), compute in line 10 the evolved set $\overline{E}_j$ (previously referred to as $\overline{R}(t_j)$) and the reached set $\overline{R}_j$ (previously referred to as $\overline{R}(t_{j-1}, t_j)$). Hard and soft constraints are evaluated on $\overline{E}_j$ in line 11 and the tuple of the results is adjoined to an ep vector (line 12). The Π set is updated in line 14 according to Section 8. However, no particular update strategy is essential to the methodology. The best point for the vector is used to update the satisfaction answer for the original constraints using (25) in line 16, to adjoin to $\overline{R}$ and to set the next $\overline{E}$ and t (lines 17 to 19). If the radius of $\overline{E}$ is larger than the maximum allowed, then we terminate early (line 20). On this matter, we chose half the radius of B as a maximum value since under a higher value an evolved set would cover most of the domain and therefore be useless for constraint satisfaction. Conversely, too small a value may cause termination too early, preventing a potential definite answer to satisfaction.

8. Exploring the property space

In this Section, we discuss how to specify the tool properties for Alg. 1, followed by how to generate a population of points for searching the optimum and how to evolve it across integration steps. While this heuristic is an important part of the methodology, the specific algorithm can be replaced with any other search algorithm. This is the reason why we present this Section after the main algorithm of our methodology. It also gives us the opportunity to provide a concluding discussion on convergence aspects.

To perform a search in $\mathbb{Z}^Q$ across step, we rely on temporal locality of the optima, i.e., the optimal valuation of properties does not change significantly between steps k and $k+1$. Temporal locality holds for IVPs as long as the time step is reasonably small with respect to the dynamics. Under this assumption, we can direct the search at step $k+1$ by using the results obtained at step k.

Each property in our framework is defined by the tuple

$$\{\text{label}, \text{type}, \text{is\_metric}, \Gamma^{\rightarrow}, \Gamma^{\leftarrow}\}$$

- label: a name, unique for the specific object, required for lookup within the property tree;
- type: the basic type used (e.g., Boolean, double, integer) or an enumerated type of symbols or objects;
- is_metric: whether there is a notion of distance between points, typically true for all non-enumerated types;
- $\Gamma^{\rightarrow}$: only for non-enumerated types, a function that converts the basic type to $\mathbb{Z}$; defaults to the identity rounded to the closest integer value;

Algorithm 2 Controlled Reach

Given the system sys, initial set I, initial time T_I, final time T_F, constraints C, bounding box B, search point p, hard constraint functions h and soft constraint functions s:

$S = \text{initialize\_satisfaction}(C)$
$\overline{R} = \emptyset$
$\Pi = \text{generate\_points\_from}(p)$
$t = T_I$
$\overline{E} = I$
while $t < T_F$ **do**
  $ep = [\,]$
  for p_j **in** Π **concurrently do**
    $\text{use\_property\_point}(p_j)$
    $[\overline{E}_j, \overline{R}_j, t_j] = \text{integration\_step}(sys, \overline{E}, t)$
    $ev_j = \text{evaluate\_constraints}(\overline{E}_j, h, s)$
    $ep = \text{adjoin}(ep, [p_j, \overline{E}_j, \overline{R}_j, t_j, ev_j])$
  end for
  $\Pi = \text{update\_points}(ep)$
  $bp = \text{best}(ep)$
  $S = \text{update\_satisfaction}(S, bp, C)$
  $\overline{R} = \text{adjoin\_best\_result}(\overline{R}, bp)$
  $\overline{E} = \text{next\_set}(bp)$
  $t = \text{next\_time}(bp)$
  if $\text{radius}(\overline{E}) \geq \text{radius}(B)/2$ **then break**
  end if
end while
return $[S, \overline{R}]$

- $\Gamma^{\leftarrow}$: only for non-enumerated types, a function that converts from $\mathbb{Z}$ to the basic type; defaults to the identity.

The choice of Γ defines the discretization of the search space. This is determined when the property is defined for the tool, to be consistent with the conventions followed by the users when manually tuning the property. For example, the value of the step size is typically explored following a $\log_2$ progression.

Since the domain is bounded, we can generate Z initial points in the search space randomly with uniform probability across all properties. Each generated point can be evaluated and ranked according to (24). In our approach, exploration is based on *adjacency*. Two points $\mathbf{z}_q$ and $\mathbf{z}'_q$ are adjacent if they differ in one and only one property value, with a value distance of 1. Values for non-metric properties are assumed to have distance 1 between each other. At the kth step of integration we discard the worst $\lfloor Z/2 \rfloor$ points and, for each of the remaining $\lceil Z/2 \rceil$ points $\mathbf{z}_q$ we generate one new *adjacent* point $\mathbf{z}'_q$. The list of points for the $(k+1)$-th step is given by the union of the best $\lceil Z/2 \rceil$ points with the adjacent $\lfloor Z/2 \rfloor$ points. In this way, we progressively explore the search space along k without deviating from the best points too quickly.

The value of Z should reflect the number of performance cores in the CPU and consequently the number of concurrent runners available. Increasing Z obviously implies that we evaluate more points, although that does not necessarily cause a speedup in the overall procedure. In fact, while we are able to look for more diverse points, the deviation of the execution times for the parallelized algorithm under such collection of points also tends to increase. Since we need to wait for all runs to finish before we can rank the corresponding points, completion is bound by the slowest case. This inefficiency can be addressed by reducing the radius of exploration over time, or by designing a scheduler that dynamically estimates the execution time and makes better allocations of points to runners, so that more points can be searched within the same maximum execution time per step.

In general, the property valuation space is non-convex and our method only explores a bounded discretization of this space. Moreover, there are no guarantees that there exists a point in such space that satisfies the accuracy requirements for constraint satisfaction. Therefore, the proposed heuristic to explore the property valuation space cannot provide convergence guarantees by itself. In our problem setting, guaranteeing the convergence of the nonlinear ordinary differential equation solver to the desired tolerance, even with optimal properties, is also non-trivial. While numerical convergence to a given accuracy is easily achievable across a single integration step, it is difficult to guarantee a sufficiently low over-approximation error for the solver over multiple steps due to the growing complexity of the representation of the evolved set. Yet, if the properties of the reachability algorithm allow controlling the over-approximation error, our optimization method will succeed in finding the minimum accuracy that addresses the underlying constraint satisfaction problem.

9. Evaluation

We use a C++ implementation of our methodology to validate its effectiveness. In particular, the library for property definition and the corresponding runtime for space exploration are written as an open-source project [26] with no external dependencies. The reachability tool Ariadne is also open source [27].

Table 1
Summary information on systems tested.

Name	Alias	Ref	N	T_F	Initial set bounds
Brusselator	BRU	[21]	2	1	[0.95, 1.05][0.93, 1.07]
Jet engine	JET	[21]	2	5	[0.9, 1.1][0.86, 1.14]
Higgins–Sel'kov	HIG	[20]	2	5	[1.99, 2.01][0.99, 1.01]
Lorenz attractor	LOR	[28]	3	1	[0.99, 1.01][0.99, 1.01][0.99, 1.01]
Rössler attractor	ROS	[28]	3	12	[−9.01, −8.99][−0.01, 0.01][0, 0.02]
Chemical reactor	CHE	[29]	4	5	[0, 1e−3][0, 1e−3][0, 1e−3][0, 1e−3]

For reproducibility purposes, the evaluation code is made available in the tool distribution. All the results were obtained using a MacBook Air M2 laptop, with 4 performance cores, consequently processing 4 search points per integration step. No other parallel processing is involved in this evaluation.

9.1. Benchmarking setup

Table 1 provides the benchmark suite, including a reference to the literature for details on the nonlinear dynamics. Testing was limited to 4 state variables due to the substantial cost of computing inner approximations despite using the most efficient method currently available in the literature [25]. However, since our methodology mitigates the need for the expensive $\perp_S$ check, as N increases, we expect to scale favorably with respect to avoiding the pre-analysis step.

For all the original systems with time-varying inputs, we transform those inputs into constants at their midpoint value. Further, we focus on time independent differential systems, with $T_I = 0$, and time-invariant constraints, which are more common in the literature. However, our methodology supports time dependence in the dynamics and the constraints, with no practical impact on the algorithm and its performance, and therefore, no impact on our evaluation plan.

Based on these considerations, we design a simple and practical constraint generator as follows:

$$\begin{aligned} & \lambda_N + 4 \sum_{i=0}^{N-1} \lambda_i \left(\frac{x_i - \hat{B}_i}{\omega_i |B_i|} \right)^2 \geq 0 \\ \text{s.t.} \quad & \forall i = 0, \dots, N,\ \lambda_i \in \{-1, 1\} \\ & \sum_{i=0}^{N} \lambda_i \neq \{-N-1, N+1\} \\ & \forall i = 0, \dots, N-1,\ \omega_i \in [0.5, 1] \vee \\ & \forall i = 0, \dots, N-1,\ \omega_i \in [1, 1.5] \end{aligned} \tag{27}$$

This generator produces axes-aligned ellipsoids and hyperboloids, centered in $\hat{B}$ with reference semi-widths given by $|B_i|/2$ values. If B is a bounding box of the reachable set, calculated using approximate reachability, then the reference ellipsoid and hyperboloid is internally or externally tangent to B, respectively. The λ_i coefficients, uniformly chosen, shuffle through all possible combinations, from which we exclude all positive or all negative coefficients since the resulting constraint would be trivial. The ω_i coefficient values instead are chosen uniformly from the same range (uniformly between $[0.5, 1]$ or $[1, 1.5]$) for all i, to produce small or large distances, and consequently, produce ellipsoids that may be enclosed by a cyclic trajectory or my enclose such a trajectory, and hyperboloids that may graze or cross a non-cyclic trajectory.

9.2. Benchmarking algorithm

The benchmarking algorithm compares the proposed constraint-driven methodology with a random unconstrained one, as shown in Alg. 3. We evaluate the number of constraints with indeterminate satisfiability answer under an equal real-time budget T_x. The time budget is set by the constraint-driven method, where we terminate as soon as no improvement over the previous internal iteration is detected. In the unconstrained case, we perform multiple runs of rigorous evolution by picking a random property assignment in the search space each time and tracking, across the runs, the constraints whose satisfiability has been decided. Once the time limit is passed, we report the comparison over the strict $[0, T_x]$ window. Similarly to the constraint-driven approach, we compute and use a bounding box also for the unconstrained methodology. If this were not the case, the unconstrained methodology would get stuck for a significant time whenever a blowup of the error is present, or it may not even terminate. All the partial constraint satisfiability results are retained as usual.

For comparison between the two methodologies we use two metrics. The first one is the final percentage of constraints with indeterminate satisfaction, called $\%^?$. However, this does not capture the speed of convergence, which can be different under the same final value. The second metric is then the integral of the percentage across the iterations, called $\int^?$.

Constraints are generated by first evaluating B from the approximate evolution of the system, by constructing 500 instances, and finally by running rigorous evolutions where $\top$ and $\perp_A$ are checked. All the constraints whose satisfaction is trivially identified by this run are removed; since assessing $\perp_S$ is not trivial, in general, and inefficient to check, we do not consider it. We observed a typical $60 - 70\%$ of constraints being discarded in this way. From the remaining constraints, 100 candidates are randomly selected. We run approximate evolutions once again on this set to provide a summary estimate of the frequency of the given prescriptions

Algorithm 3 Single run of benchmarking

Given the system sys, initial set I and final time T_F:

1: C = generate_constraints(sys, I, T_F)
2: $\tilde{\sigma}$ = identify_prescriptions(sys, I, T_F, C)
3: $[\%_c^?, \int_c^?, S_c, T_x]$ = constrained_check(sys, I, T_F, C)
4: $[\%_u^?, \int_u^?, S_u]$ = unconstrained_check(sys, I, T_F, C, T_x)
5: **return** $[\tilde{\sigma}, T_x, \%_c^?, \int_c^?, S_c, \%_u^?, \int_u^?, S_u]$

$\tilde{\sigma}$. The constrained_check method then runs Alg. 1 against the constraints and collects the required metrics, including a vector S_c with the frequencies of satisfaction for each $\top$, $\bot_A$, and $\bot_S$ possible outcome. The corresponding unconstrained_check mirrors the constraint-driven version, with no prescription of outcome and no control of the error. Only one approximate evolution is initially performed to identify a bounding box for early termination. Regular termination happens when T_x is hit.

9.3. Integration step

Before we introduce the properties to be chosen, we summarize how we perform the integration step. We use a *Picard integration* scheme [21], under which the reached set R_k between steps $k-1$ and k and the evolved set E_k at step k are computed from E_{k-1} as follows:

1. Given the bounds of E_{k-1}, hereby referred to as $[E_{k-1}]$, evaluate an approximation of the upper bound on the integration step size to use, hereby called δ_{lip};
2. Find an over-approximation of the bounds of the reach set $[R_k]$ by starting with an approximation computed using Euler's method and checking if it is a contraction according to Picard–Lindelöf theorem [30]; in this case, use the contracted box as $[R_k]$, otherwise halve the step size δ_{bnd} and repeat the procedure; hence it holds that $\delta_{bnd} \leq \delta_{lip}$;
3. Using Picard's method, starting from $[R_k]$ find a contraction of the flow function Φ_k with a given maximum error, progressively increasing the temporal order; if the error is too large, then the step size is halved and the procedure repeated, hence it holds that $\delta_k \leq \delta_{bnd}$;
4. $R_k = \Phi_k(E_{k-1}, [0, \delta_k])$ and $E_k = \Phi_k(E_{k-1}, \delta_k)$.

No global fixed value of the integration step size is suggested: the first approximate estimate δ_{lip} (called "Lipschitz step") is proportional to the inverse of the Lipschitz constant:

$$\delta_{lip} = \frac{K_{lip}}{\left| J_f([R_{k-1}] \times [0, \delta_{k-1}]) \right|} \tag{28}$$

where the Lipschitz constant is defined as the norm of the Jacobian of the vector field f of the dynamics, applied to the product of the previous evolve bounds and the time interval identified by the previous integration step size (zero, initially). The *tolerance constant* $K_{lip} < 1$ is a parameter used to obtain a contraction on the first attempt.

9.4. Properties under tuning

The properties that generate the search space can be organized in a property hierarchy tree, where their (alphabetical) position is identified by a path, in a similar way as with a file system, where the evolver object represents the root. For each property we specify the basic type for the value domain, whether this is a metric property (Y/N), and the optional conversion rule $\Gamma^{\leftarrow}(z)$ of a value z from the search domain back to the value domain. When $\Gamma^{\leftarrow}(z)$ is the identity, which is always true for an enumeration type, we do not specify it. The conversion forward is also implicit from $\Gamma^{\leftarrow}(z)$ in our cases and it is not mentioned.

- ENABLE_RECONDITIONING [Boolean, N]: states whether reconditioning of the set is allowed, pending the error being lower than the maximum spacial error. Reconditioning transforms the uniform error terms into additional parameters of the set, thus allowing to control the error better at the cost of additional complexity of the set.
- INTEGRATOR/BOUNDER/LIPSCHITZ_TOLERANCE [double, Y, $\Gamma^{\leftarrow} = 2^z$]: the K_{lip} constant in the previous Subsection.
- INTEGRATOR/MAXIMUM_TEMPORAL_ORDER [unsigned integer, Y]: the maximum order in the time variable for the polynomial expansion.
- INTEGRATOR/MINIMUM_TEMPORAL_ORDER [unsigned integer, Y]: the minimum order in the time variable for the polynomial expansion.
- INTEGRATOR/STEP_MAXIMUM_ERROR [double, Y, $\Gamma^{\leftarrow} = 10^z$]: the maximum error (i.e., remainder) of the flow tube polynomial function; the lower the value, the higher the temporal order used;
- INTEGRATOR/SWEEPER/THRESHOLD [double, Y, $\Gamma^{\leftarrow} = 10^z$]: the minimum value that a coefficient of a term in the polynomial representation of the flow function is allowed to have; terms with smaller coefficients are "swept" as addends into the uniform error term, in order to reduce the complexity of the polynomial representation without affecting the bounds of the corresponding set;

Table 2
Range of property values chosen for all experiments, with corresponding integer search space values.

Property	D		$\mathbb{Z}$	
	[	]	[	]
ENABLE_RECONDITIONING	False	True	0	1
INTEGRATOR/BOUNDER/LIPSCHITZ_TOLERANCE	0.0675	0.5	−4	−1
INTEGRATOR/MAXIMUM_TEMPORAL_ORDER	5	10	5	10
INTEGRATOR/MINIMUM_TEMPORAL_ORDER	0	5	0	5
INTEGRATOR/STEP_MAXIMUM_ERROR	1e−6	1e−4	−6	−4
INTEGRATOR/SWEEPER/THRESHOLD	1e−8	1e−6	−8	−6
MAXIMUM_SPACIAL_ERROR	1e−8	1e−5	−8	−5

Table 3
Comparison of unresolved constraints between the constrained and unconstrained methodologies, showing average and standard deviation of their final number $\%^?$ and integral number along optimization $\int^?$.

Sys			Constrained				Unconstrained			
	T_x (s)		$\%^?$		$\int^?$		$\%^?$		$\int^?$	
	avg	std	avg	std	avg	std	avg	std	avg	std
BRU	11	7	8	4	5	4	39	15	6	3
JET	162	120	9	3	52	31	39	12	96	55
HIG	359	375	6	2	108	161	44	13	180	132
LOR	155	153	3	2	42	54	62	27	117	72
ROS	59	59	12	21	22	36	48	26	56	38
CHE	2117	1648	12	5	836	729	28	13	757	583

- MAXIMUM_SPACIAL_ERROR [double, Y, $\Gamma^{\leftarrow} = 10^z$]: the maximum error in the set, over which reconditioning is performed if allowed.

The seven properties described are only a subset of the properties available to the user for continuous evolution (or at other layers of the tool), although arguably the most important ones. From their role it is already apparent that the interaction between them is non-trivial in terms of the resulting quality of the set. Aside from ENABLE_RECONDITIONING, which has 2 values, all properties are metric, hence the search moves by adjacency across all dimensions. Some other properties, however, would be inherently non-metric, such as enumerations of method objects conforming to an interface (e.g., different implementations of the inner approximator).

Table 2 provides the ranges chosen for each property, along with the corresponding ranges in the integer search space according to $\Gamma^{\rightarrow}$, resulting from our general experience with the tool. The resulting search space size is given by $2\times4\times6\times6\times3\times3\times4 = 10368$ points, more than enough to call for a directed search rather than sampling, even when high concurrency is available. Notably, all the combinations of these values are reasonable, i.e., from prior knowledge about the tool there are no search points for which an integration failure is expected. Even if a step failed for a property assignment, a secondary advantage of our multiple-point approach is that it is possible to progress using the successful ones and restore the population size at the following step. Overall, the domain chosen is general enough that we may manually choose any point. However, we will show how poorly a random fixed choice (representative of manual tuning) performs in terms of constraint satisfaction resolution.

9.5. Results

Tables 3 and 4 provide the results related to unresolved constraints and the success rate based on the expected outcome, respectively. Results are obtained from 20 independent runs, where both average and standard deviation are provided.

We can see from Table 3 that there is a definite efficiency advantage in the constraint-driven case: the $\perp_A$ case can be detected and acted upon, avoiding inner approximations altogether for all times. On the other hand, for the unconstrained case, if the conditions for $\perp_A$ are not satisfied but those for $\perp_S$ are, then the inner approximation needs to computed. Once it is computed for one constraint, the inner approximation can be used, whenever necessary, for any other constraints on that specific integration step.

Being driven toward higher accuracy, constraint-driven instances are slower, in general. Conversely, unconstrained instances rarely improve in terms of number of correct satisfaction answers over the first iterations. For example, on the CHE instance that took $T_x = 4387$ s, the constraint-driven methodology needed only 4 iterations before settling to a value of $\%^? = 6$. Using the same time budget, the unconstrained methodology was able to perform 1198 iterations but remained stuck at 14% of constraints unresolved from iteration 58 onward. The integral value $\int^?$ turns out to be favorable for the unconstrained methodology because it takes more time for the constraint-driven approach to perform its first iterations and resolve the majority of the constraints. We remark that constraint evaluation takes a negligible time as opposed to integration, therefore the slowdown in controlled reachability is largely independent of the number of constraints used.

Table 4 shows the success rates for each prescription. The $\tilde{\sigma}$ results provide the average and standard deviation of the distribution of the *expected* prescription across all three values, which is obtained during constraints construction. The actual prescription changes

Table 4
Expected prescription distribution and success rate for all systems, where values are expressed as percentages.

Sys	Expected $\tilde{\sigma}$						Success rate					
	$\top$		$\bot_A$		$\bot_S$		$\top$		$\bot_A$		$\bot_S$	
	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std
BRU	9	6	66	6	25	5	70	33	98	2	80	10
JET	7	12	65	9	28	6	42	49	98	2	76	8
HIG	8	9	83	8	9	2	56	38	99	1	61	19
LOR	18	20	81	20	2	1	72	34	95	22	9	27
ROS	30	27	70	27	1	1	55	49	100	0	29	49
CHE	20	10	52	9	28	3	71	24	100	1	70	14

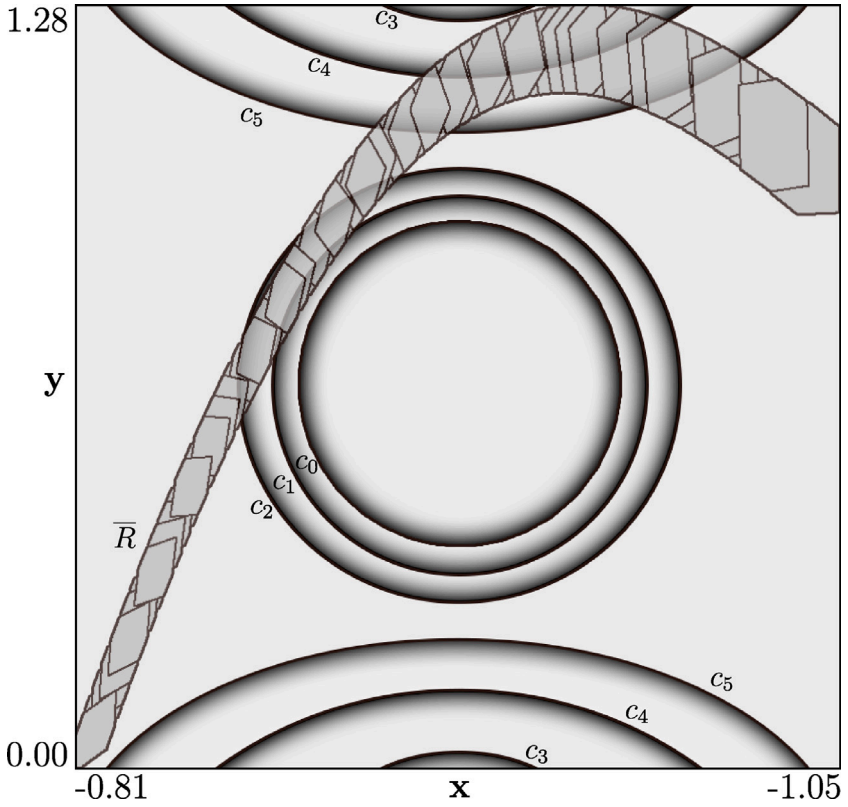


Fig. 4. Reachable set $\overline{R}$ of BRU along with elliptical (c_0, c_1 and c_2) and hyperbolic (c_3, c_4 and c_5) constraints each yielding the $\top$, $\bot_S$ and $\bot_A$ results respectively.

at each iteration of each run based on the point used for pre-analysis, so it can be slightly different, although we see from the Table that the deviation across runs is often small. The success rate columns give the percentage of providing the prescribed answer with respect to the actual prescription. Scenarios leading to $\bot_A$ are the easiest to generate and verify, while $\top$ and $\bot_S$ are less numerous. While both $\top$ and $\bot_S$ are more difficult to be identified, $\top$ also has a high variance, i.e., it is more sensitive to the instance and consequently to the over-approximation error induced by property values. This can be mainly ascribed to the fact that an excessive growth of the over-approximation error during pre-analysis causes early termination, which prevents from checking $\top$ during controlled reachability.

Finally, let us focus on a concrete case using BRU, whose reachable set as obtained by Ariadne is shown in Fig. 4. Here $\overline{R}$ is given by a sequence of nonlinear sets converted into affine sets, starting at T_I from the top right corner and ending at T_F at the bottom left corner. In addition, the boundaries of the six tailored constraints used to run this instance of BRU are displayed. Due to rendering limitations in Ariadne when displaying multiple constraints, those have all been drawn manually as accurately as possible using ellipses; the side of the boundary where $\bot$ is obtained is shaded.

Table 5 supplies all the information on the constraint definition, according to the formula described in (27), along with analysis data on their impact on optimization: the satisfaction result $\underline{\overline{\sigma}}$, the maximum time for checking the constraint $\tilde{T}^*$ and the maximum growth rate α. The values $|B| = (1.86, 1.28)$ and $\hat{B} = (0.12, 0.64)$ hold for the widths and center of the reachable set obtained. The values of $\underline{\overline{\sigma}}$ as obtained by our verification flow show that we purposely chose elliptical and hyperbolic constraints that stress the

Table 5
Summary information on constraints for the BRU run.

Name	$\lambda_{0,1,2}$	ω_0	ω_1	$\overline{\underline{\sigma}}$	$\tilde{T}^*$	α
c_0	1, 1, −1	0.50	0.54	$\top$	1.00	73
c_1	1, 1, −1	0.65	0.70	$\bot_S$	0.74	62
c_2	1, 1, −1	0.83	0.89	$\bot_A$	0.78	21
c_3	−1, 1, 1	0.86	0.89	$\top$	1.00	33
c_4	−1, 1, 1	0.74	0.75	$\bot_S$	0.25	1553
c_5	−1, 1, 1	0.59	0.62	$\bot_A$	0.27	2758

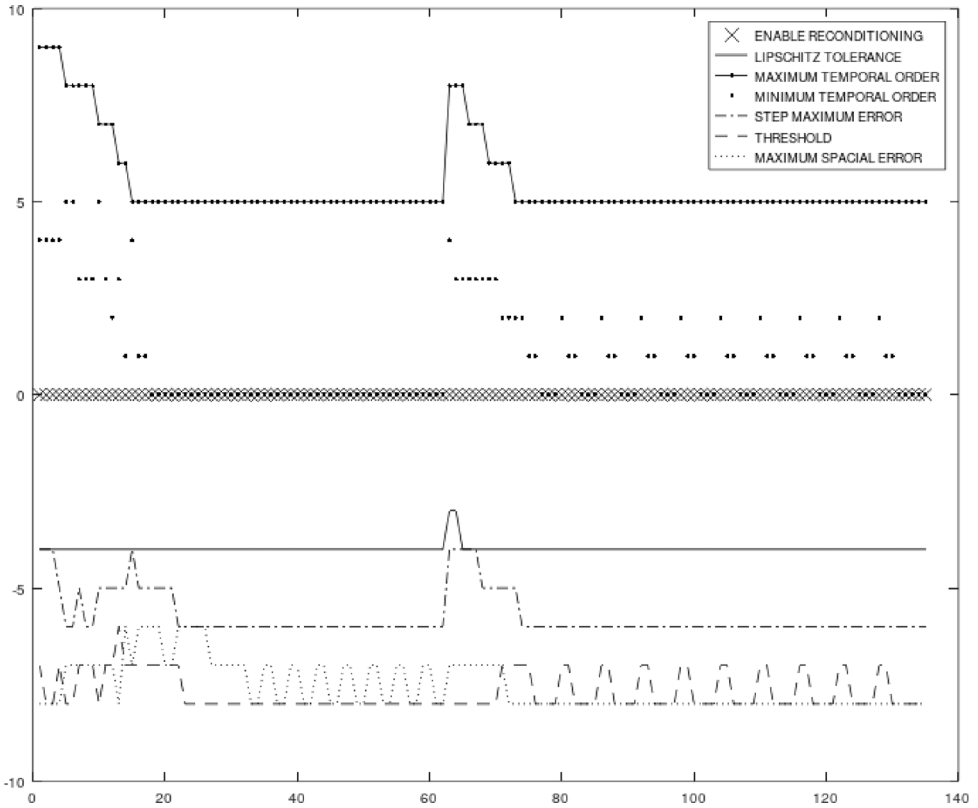


Fig. 5. Values of the properties along integration steps for the BRU run.

methodology by being relatively close to tangency. For the BRU system, this happens for $0.5 \leq \omega \leq 1.0$ in both elliptical ($\lambda_0\lambda_1 = 1$) and hyperbolic ($\lambda_0\lambda_1 = -1$) constraints. We can identify such stress for the single constraint by looking at the value of α, which according to (16) is an upper bound on the error growth. Consequently, the lower the value of α, the more the satisfaction of the constraint requires a higher accuracy: c_2 effectively is the most critical constraint since its boundary is close to the trajectory and that situation occurs at later times, since $\tilde{T}_2^* = 0.78$. A similar tangency with c_0 (to yield $\top$) is less critical since it occurs earlier, where accumulated error is lower. Therefore the least demanding constraints are c_4 and c_5, since there is ample margin for error and the $\bot$ condition needs to be checked for up to around 0.27 s only. Please note that while $\tilde{T}^*$ for constraints yielding $\bot$ matches the critical time, for constraints yielding $\top$ it is always the final time since the whole reachable set is required.

Fig. 5 instead shows the evolution of the optimal $\mathbb{Z}$ values of the properties listed in Table 2 across the evolution time. For completeness, the verification task took 11 s and terminated after 137 integration steps within a single run of the loop in Alg. 1. The first optimized point is $(0, -4, 9, 4, -4, -7, -8)$ and the final point is $(0, -4, 5, 0, -6, -8, -8)$. We can see that for most properties the value stabilizes within the first 20 steps. For others, such as the minimum temporal order, an oscillation in $[0, 2]$ is present, against the $[0, 5]$ range allowed. Around step 70 there is a significant variation in the values, due to the critical region where a satisfaction of c_1 and c_2 is obtained: since those constraints are not needed anymore, the conditions for optimization change.

10. Conclusions

We addressed the problem of calculating the reachable set of a generic nonlinear system in the most efficient way that can return a definite answer to the satisfaction of a set of constraints. We introduced a method to identify bounds for the growth of

the computation error along the system evolution. We then proposed to automatically search for appropriate choices of values of the reachability tool properties that can achieve the desired bounds. In our prototype implementation, the property value space is discretized and the search is performed by evolving a population of candidate property assignments based on adjacency relations. Results from our benchmark suite show that a random choice of property values cannot provide definite answers to a significant percentage of the constraints, even after a large number of trials. Conversely, our method can resolve almost all the constraints and it converges generally faster. By avoiding manual tuning of tool properties, it can make a significant leap in the effectiveness and automation of verification using reachability analysis.

Future plans include extending the theory to support time-varying inputs and hybrid systems. Moreover, we plan to extend the property definition and exploration framework to support unbounded spaces with constraints between properties, and implement more sophisticated search algorithms. Last but not least, we intend to interface the pExplore runtime to finite-time reachability tools other than Ariadne, in order to assess the efficiency of our methodology in different scenarios.

CRediT authorship contribution statement

Luca Geretti: Formal analysis, Investigation, Methodology, Writing – original draft, Writing – review & editing. **Pieter Collins:** Formal analysis, Methodology, Validation. **Pierluigi Nuzzo:** Supervision, Methodology, Writing – review & editing. **Tiziano Villa:** Methodology, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the US National Science Foundation under awards 1846524 and 2139982 and by the US Office of Naval Research (Science of AI and Science of Autonomy Programs) under award N00014-20-1-2258.

References

[1] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, SpaceEx: Scalable verification of hybrid systems, in: Proc. 23rd International Conference on Computer Aided Verification, CAV 2011, in: LNCS, vol. 6806, Springer, Snowbird, UT, USA, 2011, pp. 379–395.

[2] S. Schupp, E. Ábrahám, I. Makhlouf, S. Kowalewski, HyPro: A C++ library of state set representations for hybrid systems reachability analysis, in: C. Barrett, M. Davies, T. Kahsai (Eds.), NASA Formal Methods, Springer International Publishing, Cham, 2017, pp. 288–294.

[3] N. Fulton, S. Mitsch, J. Quesel, M. Völp, A. Platzer, KeYmaera X: An axiomatic tactical theorem prover for hybrid systems, in: A.P. Felty, A. Middeldorp (Eds.), CADE, in: LNCS, vol. 9195, Springer, 2015, pp. 527–538.

[4] S. Ratschan, Z. She, Safety verification of hybrid systems by constraint propagation based abstraction refinement, ACM Trans. Embed. Comput. Syst. 6 (1) (2007).

[5] M. Althoff, An introduction to CORA 2015, in: G. Frehse, M. Althoff (Eds.), ARCH14-15. 1st and 2nd International Workshop on Applied Verification for Continuous and Hybrid Systems, in: EPiC Series in Computing, vol. 34, EasyChair, 2015, pp. 120–151, http://dx.doi.org/10.29007/zbkv.

[6] D. Bresolin, P. Collins, L. Geretti, R. Segala, T. Villa, S. Zivanovic, A computable and compositional semantics for hybrid automata, in: Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control, 2020.

[7] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, C. Schilling, JuliaReach: A toolbox for set-based reachability, in: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 39–44, http://dx.doi.org/10.1145/3302504.3311804.

[8] X. Chen, E. Ábrahám, E. Sankaranarayanan, Flow*: An analyzer for non-linear hybrid systems, in: Proc. of the 25th International Conference on Computer Aided Verification, CAV 2013, in: LNCS, vol. 8044, Springer, Saint Petersburg, Russia, 2013, pp. 258–263.

[9] M. Althoff, J.M. Dolan, Online verification of automated road vehicles using reachability analysis, IEEE Trans. Robot. 30 (4) (2014) 903–918, http://dx.doi.org/10.1109/TRO.2014.2312453.

[10] Y. Chou, H. Yoon, S. Sankaranarayanan, Predictive runtime monitoring of vehicle models using bayesian estimation and reachability analysis, in: IEEE International Conference on Intelligent Robots and Systems, 2020, pp. 2111–2118.

[11] A. Alanwar, Y. Stürz, K.H. Johansson, Robust data-driven predictive control using reachability analysis, Eur. J. Control (2022).

[12] L. Benvenuti, D. Bresolin, P. Collins, A. Ferrari, L. Geretti, T. Villa, Assume-guarantee verification of nonlinear hybrid systems with Ariadne, Int. J. Robust Nonlinear Control 24 (4) (2014) 699–724, http://dx.doi.org/10.1002/rnc.2914.

[13] M. Wetzlinger, N. Kochdumper, M. Althoff, Adaptive parameter tuning for reachability analysis of linear systems, in: Proceedings of the IEEE Conference on Decision and Control, Vol. December, 2020, pp. 5145–5152.

[14] M. Wetzlinger, A. Kulmburg, M. Althoff, Adaptive parameter tuning for reachability analysis of nonlinear systems, in: HSCC 2021 - Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control, Part of CPS-IoT Week, 2021.

[15] L. Geretti, R. Muradore, D. Bresolin, P. Fiorini, T. Villa, Parametric formal verification: the robotic paint spraying case study, in: Proceedings of the 20th IFAC World Congress, 2017, pp. 9248–9253.

[16] G. Frehse, R. Kateja, C. Le Guernic, Flowpipe approximation and clustering in space-time, in: HSCC 2013 - Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, Part of CPSWeek 2013, Vol. 1, 2012, pp. 203–212.

[17] M. Wetzlinger, N. Kochdumper, S. Bak, M. Althoff, Fully automated verification of linear systems using inner and outer approximations of reachable sets, IEEE Trans. Autom. Control 68 (12) (2023) 7771–7786, http://dx.doi.org/10.1109/TAC.2023.3292008.
[18] M. Wetzlinger, N. Kochdumper, S. Bak, M. Althoff, Fully-automated verification of linear systems using reachability analysis with support functions, in: Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control, HSCC '23, 2023, http://dx.doi.org/10.1145/3575870.3587121.
[19] N. Kochdumper, S. Bak, Fully automated verification of linear time-invariant systems against signal temporal logic specifications via reachability analysis, 2024, arXiv:2306.04089.
[20] X. Chen, S. Sankaranarayanan, Decomposed reachability analysis for nonlinear systems, in: 2016 IEEE Real-Time Systems Symposium, RTSS, 2016, pp. 13–24.
[21] X. Chen, Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models (Ph.D. thesis), Aachen University, 2015.
[22] S. Bak, S. Bogomolov, C. Schilling, High-level hybrid systems analysis with Hypy, in: Proceedings of the Workshop on Applied Verification of Continuous and Hybrid Systems, Vol. April, 2016, pp. 80–90.
[23] M. Wetzlinger, A. Kulmburg, A. Le Penven, M. Althoff, Adaptive reachability algorithms for nonlinear systems using abstraction error analysis, Nonlinear Anal. Hybrid Syst. 46 (2022).
[24] S. Sankaranarayanan, Automatic invariant generation for hybrid systems using ideal fixed points, in: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC '10, Association for Computing Machinery, New York, NY, USA, 2010, pp. 221–230, http://dx.doi.org/10.1145/1755952.1755984.
[25] N. Kochdumper, M. Althoff, Computing non-convex inner-approximations of reachable sets for nonlinear continuous systems, in: 2020 59th IEEE Conference on Decision and Control, CDC, 2020, pp. 2130–2137, http://dx.doi.org/10.1109/CDC42340.2020.9304022.
[26] Pexplore: parallel exploration of properties of an iterative procedure, 2023, https://github.com/ariadne-cps/pexplore.
[27] Ariadne: an open library for formal verification of cyber-physicalsystems, 2020, http://www.ariadne-cps.org.
[28] S.H. Strogatz, Nonlinear Dynamics and Chaos (Second Edition), in: Studies in Nonlinearity, CRC Press, 2014.
[29] S. Harwood, P. Barton, Efficient polyhedral enclosures for the reachable set of nonlinear control systems, Math. Control Signals Systems 28 (8) (2016).
[30] V.I. Arnold, Ordinary Differential Equations, Springer Berlin, 1992.