Nonlinear Analysis: Hybrid Systems 20 (2016) 1–20

Contents lists available at ScienceDirect

Nonlinear Analysis: Hybrid Systems

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

Enclosing the behavior of a hybrid automaton up to and beyond a Zeno point☆

Michal Konečný [a], Walid Taha [b,c], Ferenc A. Bartha [c], Jan Duracz [b], Adam Duracz [b,*], Aaron D. Ames [d]

[a] Computer Science Group, Aston University, Birmingham, B4 7ET, UK

[b] School of Information Science, Computer and Electrical Engineering, Halmstad University, Box 823, S-301 18, Halmstad, Sweden

[c] Department of Computer Science, Rice University, PO Box 1892, MS-132, Houston TX 77251, USA

[d] Woodruff School of Mechanical Engineering, School of Electrical & Computer Engineering, Georgia Institute of Technology, Atlanta GA, USA

ARTICLE INFO

Article history:
Received 2 December 2013
Accepted 27 October 2015

Keywords:
Hybrid automata
Zeno behavior
Hybrid systems
Rigorous computations
Validated numerics

ABSTRACT

Even simple hybrid automata like the classic bouncing ball can exhibit Zeno behavior. The existence of this type of behavior has so far forced a large class of simulators to either ignore some events or risk looping indefinitely. This in turn forces modelers to either insert ad-hoc restrictions to circumvent Zeno behavior or to abandon hybrid automata. To address this problem, we take a fresh look at event detection and localization. A key insight that emerges from this investigation is that an *enclosure* for a given time interval can be valid independent of the occurrence of a given event. Such an event can then even occur an unbounded number of times. This insight makes it possible to handle some types of Zeno behavior. If the post-Zeno state is defined explicitly in the given model of the hybrid automaton, the computed enclosure covers the corresponding trajectory that starts from the Zeno point through a restarted evolution.

1. Introduction

Simulation is widely used to analyze the design of complex systems that comprise both physical and digital components. Such *Cyber-Physical Systems* include a wide range of novel products such as active prosthetics, advanced driver assistance systems, and elderly assistance robots. Unfortunately, many widely used simulation tools exhibit failure modes that limit the validity and utility of the results they produce. Four main sources of such failures can be identified:

1. Number representation and implementation of arithmetic [1]. There are uncountably many real numbers, and our machines can only compute using finite observations about them. Even when a lazy representation is used [2], it does not change the fact that equality or comparison is only semi-decidable.
2. Function representation and construction [3]. A continuous system is often modeled using differential equations, and simulation means finding a function that solves these equations. When the differential equations are linear, the solutions

☆ A preliminary version of this paper was published in the proceedings of CPSNA 2013 (Konečný et al., 2013) [10]. The present version adds: Sections 4 and 5 where the results stated in the preliminary version are recast in a stronger form and with complete proofs; a simplified version of event tree (Definition 4.8); and new examples, diagrams and explanations.

* Corresponding author. Tel.: +46 (0) 35 16 71 63.
E-mail address: adam.duracz@hh.se (A. Duracz).

http://dx.doi.org/10.1016/j.nahs.2015.10.004

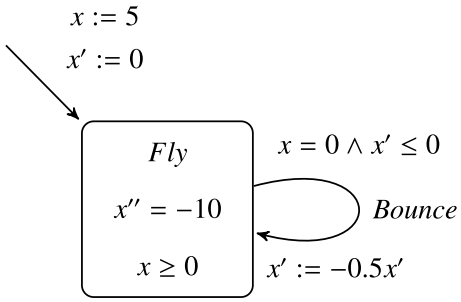


Fig. 1. A simple hybrid automaton model of a bouncing ball.

have standard, closed-form representations. In general, this is not the case for non-linear differential equations, which arise naturally in a wide range of domains, especially in three-dimensional physical space. Approximate solutions for such problems are usually obtained using an appropriate numerical method [4,5] chosen based on the characteristics of the model and the expectations on the solution (e.g. stiffness, symplecticity). The difficulty with function representation can be seen as a strict generalization of that with number representation.

3. Event detection and localization [6]. Event detection is only semi-decidable, and "perfect" localization is, in general, not computable in finite time.
4. Zeno behavior [6–10]. This difficulty arises as a result of the interaction between continuous and discrete components. It is a problem both for formalizing the semantics of hybrid automata and, as we review next, for simulation tools.

Of these four sources of failures, Zeno behavior may have received the least attention. It was first studied in the context of hybrid systems over a decade ago [8], where it was observed to be an interesting pathology of hybrid automata.

This seemingly singular behavior turns out to be an essential point of divergence between smooth (traditional) dynamical systems and hybrid automata—its existence can result in hybrid automata simulators producing incorrect solutions and/or entering infinite loops. As such, it is an essential consideration in determining whether the behavior of a hybrid systems simulator is acceptable.

Zeno behavior arises naturally when modeling physical systems. Indeed, Zeno behavior has long been studied in optimal control and nonsmooth mechanics, including mechanical systems with impacts [11–13], systems with friction [12] or electrical systems with switching, albeit using formalisms tailored for these domains. Numerous works study nonsmooth linear models and give sufficient condition on such systems for existence [14] or non-existence [15,16] of certain types of Zeno behavior. In mechanics, Zeno behavior arising from impacts at the transition from bouncing to sliding is commonly observed [17]. For example, it can occur in rigid body dynamics with impact constraints, such as those modeling bipedal robots with mechanical knee stops [18]. In this paper, we use examples of mechanical nature only due to their universal intuitive appeal. The results apply to hybrid systems that have no physical interpretation.

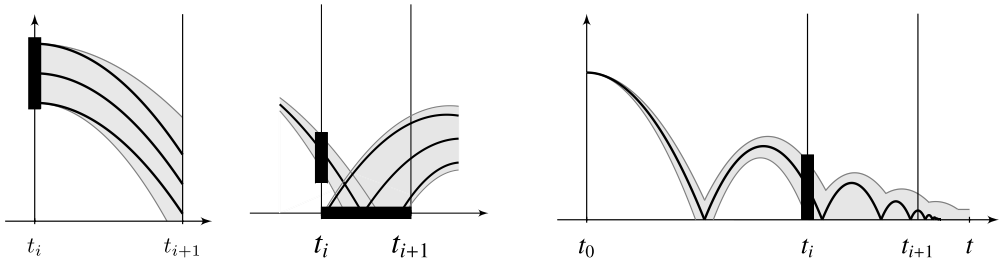
Zeno behavior can be illustrated with the simple bouncing ball modeled as in Fig. 1. The position, velocity and acceleration of the ball are denoted as functions of t by $x(t)$, $x'(t)$ and $x''(t)$, respectively. We fix the initial condition to be $x(0) = 5$ and $x'(0) = 0$, and let $x''(t) = -10$ as long as $x(t) \geq 0$ to model the effect of gravity. If $x(t^-) = 0$ and $x'(t^-) \leq 0$, then we let $x'(t^+) = -0.5x'(t^-)$, which models an impact with the ground where half of the speed is lost with every bounce. A simple calculation shows an interesting characteristic of this system: the time between each successive impact of the ball forms a convergent geometric progression. The time that this progression converges to is called the Zeno time, and the state of the system at this time is called Zeno point[1] [20,21]. While this model does not specify the post-Zeno behavior of the bouncing ball, the issue may be handled by extending the automaton [22].

1.1. Problem

Hybrid automata simulation tools struggle with the bouncing ball model presented above. For example, consider Simulink (v7.9) [23], SystemModeler (v3.0.0) [24], OpenModelica (v1.9.0 beta 4) [25], Charon (v1.0) [26], and the FRP [27] implementation Yampa (v0.9.3) [28]. With Simulink, it is not clear how to express the equality condition directly. None of the other tools detect the condition $x(t) = 0$ that should trigger the bounce. For most of the tools, this means that the ball falls through the floor. SystemModeler gives a warning to the user that the model tests equality on real numbers and reports that it considers this problematic. Running it on this example produces an error.

The model becomes a bit easier for most tools if we replace the $x = 0$ test by $x \leq 0$. This helps the tools hide the fact that they fail to detect some events. In OpenModelica, the ball still falls through the floor. Yampa and Charon continue past the Zeno point because they do not attempt to detect all event occurrences. SystemModeler (and, on one formulation, Simulink)

[1] Zeno behavior may be absent from the so-called set-valued bouncing ball, where the acceleration of the ball (still being negative) may take different values from a compact interval [19].



(a) Enclosing continuous segments.

(b) Detecting and handling single events.

(c) Handling segments with multiple events.

Fig. 2. Basic components of a Simulator for Zeno systems.

become increasingly slower as they get closer to the Zeno point, which suggests that they are attempting to correctly handle events when they are expressed as inequalities. Thus, even though they get stuck indefinitely, they are arguably better than the other tools because they at least try to detect all events expressed as inequalities. Simulink can exhibit another behavior when a slightly different model is used. It employs some heuristics [6] to try to deal with Zeno behavior by giving up after a preset number of events has occurred and if the changes between events are within a preset threshold. Of course, using such heuristics foregoes any guarantees about resulting behavior.

Are there alternative approaches to simulation that can at least handle some instances of Zeno behavior?

1.2. Contributions

We propose an interval-based method for enclosing hybrid automaton evolutions that is capable of going up to, including, and beyond Zeno point for certain types of Zeno systems. The key features of the method are:

- If the post-Zeno behavior is modeled explicitly in the automaton, the result encloses the post-Zeno trajectory. The benefit of the method here is that, when it works, it can simulate systems which essentially all existing traditional simulation methods are unable to simulate.
- If the post-Zeno behavior is not modeled explicitly, the result contains all trajectories starting close to the Zeno point. In this case, the resulting enclosure can be seen by the user as a suggestion for a natural completion of the automaton, that is, an extra state that models the post-Zeno behavior explicitly.

The method in its current form appears to work for Zeno systems with a stable finite orbit.

Any simulator can be seen as consisting of three basic simulator functions: enclosing continuous segments using an enclosure-based ordinary differential equation initial value problem (ODE IVP) solver (Fig. 2(a)), detecting and handling single events (Fig. 2(b)), and handling segments with multiple events (Fig. 2(c)). The method presented is parametric in the solver and places only minimal requirements on the representation of enclosures [29–31].

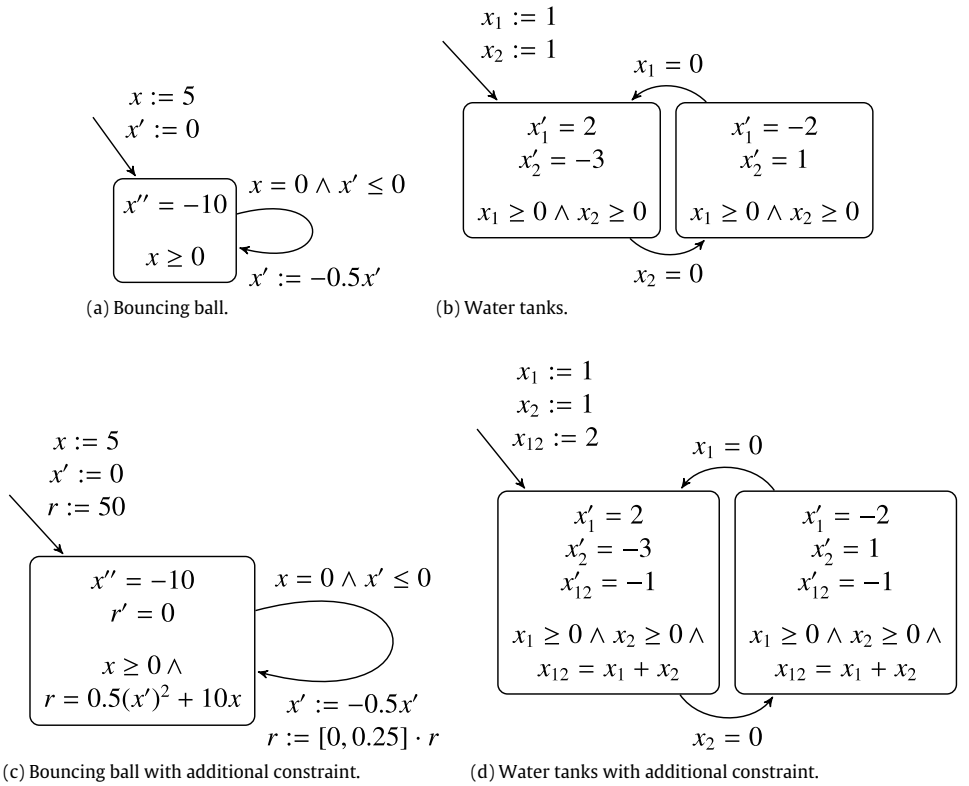
There are two distinct challenges in enclosing Zeno behaviors. The first is to compute *any* enclosure past the Zeno point. This cannot be done (in a finite number of steps) by an algorithm that attempts to explicitly handle every event. The second is to compute a tight enclosure. The proposed algorithm deals with the first challenge; but, as we illustrate using some simple examples, the second challenge can be addressed by the user, refining the hybrid system models to ensure that they contain enough information to produce a tight enclosure. This is achieved by allowing the user to introduce additional semantically redundant constraints to the model.[2] Adding these supplementary constraints, obtained by qualitative analysis, helps to reduce the over-approximation that arises from numerical methods. Fig. 4 illustrates how the new method overcomes these two challenges on two classic examples of hybrid systems, given in Fig. 3. The strategy used to produce the enclosures in Figs. 4, 11, 10 and 12 is adaptive, it refines the time segments near events which leads to accurate event handling.

The paper is structured as follows. After reviewing related work (Section 2) and introducing a formal notion of hybrid automata, evolutions, limit states and their enclosures (Section 3), we present a semantics for event detection and handling (Sections 4 and 5). The use of enclosures suggests an elegant way to deal with events (Definition 5.1) that makes it possible to enclose Zeno behaviors. The semantics is algorithmic, and can therefore be used directly for simulation. We show that it produces enclosures which, when defined, provide upper and lower bounds on functions that satisfy the model being simulated (Theorem 5.2). Finally, to emphasize that our method does not make any assumptions about the linearity of the underlying system, in Section 6 we present the computed enclosures for some non-linear variants of the model in Fig. 1.

2. Related work

Zeno behavior is an issue that appears in and affects several distinct areas of research. In this section, we address the ones that pertain most directly to explaining the need and value of the work presented in this paper.

[2] In future work we plan to address the second challenge by more advanced interval methods, reducing the need for refining models in this way.



(a) Bouncing ball. (b) Water tanks.

(c) Bouncing ball with additional constraint. (d) Water tanks with additional constraint.

Fig. 3. Hybrid automata for the Zeno systems whose solution enclosures are shown in Fig. 4. Figure (a) and (b) show the bouncing ball and water tank systems of Lygeros [32, Figures 3.1 and 3.7]. We make these models more challenging to simulate by replacing some inequalities with equalities (such as the bouncing conditions). In models (c) and (d), the additional constraint variables r and x_{12} represent the energy and the total amount of water, respectively.

2.1. Zeno in nonsmooth dynamical systems

Our work is conceptually analogous to realizing Filippov's approach, which models nonsmooth dynamical systems by differential inclusions [11–13,33]. For the resulting models, simulation techniques using Moreau's sweeping process [34,35], are able to handle some Zeno systems [36]. These techniques use time-stepping schemes for (interval) set-based simulation which are hard-coded in the simulator, and the user must modify the original model by explicitly augmenting the model with additional constraints, for example, using the linear complementary notation.

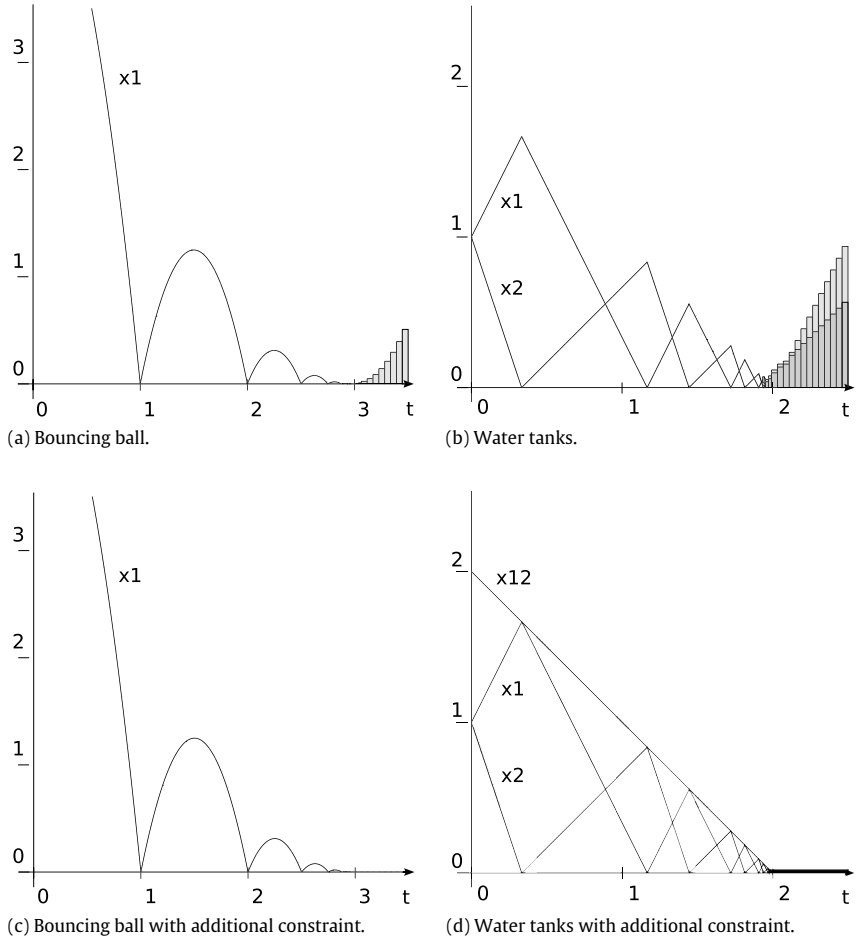
In contrast, our approach is aimed at a semantically justifiable formulation of hybrid systems with complete or incomplete specifications of post-Zeno behavior. We see the methodology for computing the approximation of the resulting systems as a first step towards showing that there is a sense in which they satisfy a formal notion of correctness defined entirely based on what is explicitly stated in the original (complete or incomplete post-Zeno specification) model, and as a result, is more directly applicable to a larger class of hybrid systems.

2.2. Detecting Zeno behavior and completing hybrid systems

There are methods (using a priori analysis) for statically checking that a system exhibits Zeno behavior [20,37,38]. For mechanical systems with impacts, there are techniques to extend the definition of systems exhibiting Zeno with a definition for behavior past the Zeno point [7,22]. But the *correctness* of these methods relies on manual (human) analysis of individual models rather than an automatic, self-contained simulation method for which proving key correctness properties is possible. Our method does allow human intervention, but only to improve the accuracy of the result, and not to achieve correctness.

2.3. Formal semantics for the verification of hybrid systems

In contrast to verification tools, simulation tools have not been traditionally built with an eye on formal semantics. But several possible starting points exist for such a formal basis. The seminal works of Alur [39] and Henzinger [40] on the theory of hybrid automata use *transition system semantics*. This type of semantics focuses on transitions between discrete states, and views the continuous behaviors within states as idealized dynamical (differential equation) systems without worrying about computer representation. More recent work by Platzer on *differential dynamical logic* [41] gives a semantics



(a) Bouncing ball. (b) Water tanks.

(c) Bouncing ball with additional constraint. (d) Water tanks with additional constraint.

Fig. 4. Enclosures for hybrid automaton evolutions with Zeno behavior. Figure (a) and (b) show the enclosures for the bouncing ball and water tank systems. Figure (c) and (d) show the results for the examples when additional (redundant) constraints are added to achieve more precise enclosures. Fig. 3 specifies these systems in detail using the automaton notation.

to a logical language that can express differential equations. This approach assumes that solutions to such equations are provided as idealized mathematical objects. Thus, neither approach explicitly addresses the questions of computerized representation or construction of real numbers or functions over reals. Discontinuities are assumed to happen only between continuous transitions, and any evolution describing a segment of system behavior consists of a finite number of transitions. This excludes Zeno behavior.

Most hybrid system verification tools also do not handle Zeno systems. Many tools, such as KeYMaera [41,42] (which combines deductive, real algebraic and computer algebra provers) and hydlogic [43,44] (an interval SAT Modulo ODE solver based on the validated ODE IVP solver VNODE-LP [45]) explicitly assume that the modeled system is free of Zeno behavior. Others, such as the reachability analysis tool Flow* [46], require the user to provide an explicit upper bound on the number of transitions to be taken into account, effectively excluding Zeno systems. One reachability analysis tool, SpaceEx [47], computes an enclosure of reachable states using the LeGuernic–Girard (LGG) algorithm [48].

This computation terminates on some Zeno systems, but communications with the authors indicate that it is not clear whether or not these enclosures are formally sound.

2.4. Formal semantics for hybrid systems simulation languages

Language definitions that have been implemented and that are closer to formal semantics include the denotational semantics for Functional Reactive Programming (FRP) [27] and the operational semantics of HyVisual [49]. The semantics for FRP takes an approach that is closely representative of that used in traditional numerical methods, namely, discretization of time into samples with non-zero time steps in between. In this approach, the fundamental correctness property is that the computed solution converges to the ideal solution when, in essence, the sampling rate goes to infinity. By increasing the sampling rate, one can get arbitrarily close to the idealized answer. No bound on the distance from the idealized answer is provided. As a result, the user has to reason independently to determine how close their simulation is to the

mathematical solution of the problem. While the semantics of HyVisual is described in a more operational manner, it is close in spirit to the semantics for FRP. In addition, HyVisual is an open framework for expressing actors and is not intended as a closed core language for simulating hybrid systems. For example, defining the processes for event detection and handling is something that actor definitions are expected to provide. Neither FRP nor HyVisual provides any special support for handling Zeno behavior. On basic examples, both produce behavior similar to what most of the tools described above produce. Implementations of both systems use standard floating-point representations for numbers.

In contrast to the widely used FRP and HyVisual implementations, more foundational efforts investigating the semantics of hybrid systems address issues of numerical precision and sampling by a semantics built on top of adequate representations of real numbers and functions of real numbers, such as those developed by Edalat and Pattinson [50] and by Bouissou and Martel [51]. The first effort uses exact real arithmetic to represent real numbers [2] and converging sequences of function enclosures to represent functions. It also introduces the notion of an *interval Picard iteration* to provide an elegant, high-level yet constructive semantics to differential equations. The second work is concerned with separating the semantics for the continuous behavior from the discrete behavior, with the purpose of developing a reference model for the integrated simulation of continuous and discrete systems. Because both semantics are an interpretation into an effective domain, they can in principle be used to provide a software implementation. It is not clear that either semantics has been implemented. More importantly, both assume that an evolution is a finite sequence, and can therefore not produce an answer that covers a Zeno point.

2.5. Other languages and systems

There has been recent work on hybrid data-flow languages and non-standard analysis (NSA). A large subset of the former may be translated into hybrid automata, though, in general, they have slightly higher expressive power [52]. The frameworks based on NSA unify discrete and continuous domains by time scales [53–55]. NSA has also been used to define rigorous semantics for hyperstreams that are able to represent some Zeno systems [56]. It is not obvious at this point whether the NSA semantics is directly executable. Investigating the relation between this approach and ours will be interesting future work.

3. Hybrid enclosures

We now formally introduce a largely standard notion of hybrid automata [20,38], their evolutions, and the enclosures of all potential evolutions (starting from a given interval initial state) that we will use in this paper. Because we base our computations on intervals and interval functions, we begin by introducing the basic concepts of interval analysis [57–60].

3.1. Interval arithmetic

Elements of the set of closed real intervals $\mathbb{I}$ are written as $A = [\underline{A}, \overline{A}]$, where $\underline{A}, \overline{A} \in \mathbb{R}$ are the *left endpoint* and *right endpoint* of A, respectively. Boldface letters denote vectors, *e.g.* $\boldsymbol{A} \in \mathbb{I}^k$. We identify vectors of intervals with the Cartesian product of the component intervals, called *boxes*. Thus we also have $\boldsymbol{A} \subseteq \mathbb{R}^k$. Hull $\{\boldsymbol{A}_i\}_{i\in I}$ denotes the smallest box containing all $\boldsymbol{A}_i$. The ODE IVP solver that parametrizes our semantics, as described in Section 1.2, provides *interval functions*, *i.e.*, functions from time to vectors of intervals, denoted $\boldsymbol{X}, \boldsymbol{Y} : T \to \mathbb{I}^k$ as enclosures of continuous segments.

The set of all intervals, a set of interval vectors of a certain dimension and a set of interval functions of a fixed type are all sets partially ordered by reverse inclusion. An operation on such sets is called *inclusion isotone* if it is monotonic with respect to this order. In other words, replacing parameter values for the operation with values that are subsets of the original values yields a subset of the original result. For example, interval vector addition is inclusion isotone as $\boldsymbol{A}_1 \subseteq \boldsymbol{B}_1, \boldsymbol{A}_2 \subseteq \boldsymbol{B}_2 \implies \boldsymbol{A}_1 + \boldsymbol{A}_2 \subseteq \boldsymbol{B}_1 + \boldsymbol{B}_2$. When we extend functions of reals to functions of intervals by replacing the constants and variables with their interval counterparts, the extensions are inclusion isotone [57–60].

3.2. Hybrid automata

Definition 3.1. A *hybrid automaton* is a tuple

$$\mathcal{H} = \Big(Q,\ \{\boldsymbol{D}_q, \boldsymbol{f}_q\}_{q\in Q},\ E,\ \{\sigma_e,\ \tau_e,\ \boldsymbol{C}_e,\ \boldsymbol{r}_e\}_{e\in E}\Big),$$

where

- Q is the finite set of *modes* q,
- $\boldsymbol{D}_q \subseteq \mathbb{R}^n$ is the *mode invariant* of q,
- $\boldsymbol{f}_q : \mathbb{R}^n \to \mathbb{R}^n$ is the *vector field* of mode q,
- E is the finite set of *event types* e,

- $\sigma_e \in Q$ is the *source* mode[3] of event e,
- $\tau_e \in Q$ is the *target* mode (see footnote 3) of event e,
- $\boldsymbol{C}_e \subseteq \mathbb{R}^n$ is the *guard* of event e,
- $\boldsymbol{r}_e : \mathbb{R}^n \to \mathbb{R}^n$ is the *reset map* of event e. □

The implementation of our hybrid automata simulator places further constraints on an automaton so that it can be executed. We require that the vector fields $\boldsymbol{f}_q$ are given as expressions understood by the chosen ODE IVP solver and the sets $\boldsymbol{C}_e$ and $\boldsymbol{D}_q$ are given as predicates in a formal language that allows us to compute (over-approximated) intersections of interval vectors with these sets. Moreover, we require that $\boldsymbol{C}_e : e \in E$ and $\boldsymbol{D}_q : q \in Q$ be closed sets. This requirement ensures the uniqueness of the time of the first crossing between a continuous trajectory and a set of active guards.

The vector fields $\boldsymbol{f}_q$ and reset maps $\boldsymbol{r}_e$ are assumed to be defined over $\mathbb{R}^n$ for the sake of simplicity. Restricting them to $\boldsymbol{f}_q : \mathcal{D}_q \supseteq \boldsymbol{D}_q \to \mathbb{R}^n$ and $\boldsymbol{r}_e : \mathcal{C}_e \supseteq \boldsymbol{C}_e \to \mathbb{R}^n$ with open domains $\mathcal{D}_q$ and $\mathcal{C}_e$, respectively, would impose the additional task to check if we stray out of their domains and abort the computation. Even though this would not impose additional difficulties in the implementation (interval arithmetic is able to handle such restrictions in a natural way), it would still complicate several formulas and thus render the rest of this paper more difficult to follow.

Definition 3.2 (*Hybrid Automaton Evolution*)**.** Let the hybrid automaton $\mathcal{H}$ be given as in Definition 3.1. An *evolution of* $\mathcal{H}$ *over time* $T_{\mathcal{E}} \in \mathbb{I}$ is a sequence

$$\mathcal{E} = (T_1, q_1, \boldsymbol{x}_1), e_1, (T_2, q_2, \boldsymbol{x}_2), e_2, \ldots$$

either ending with $(T_k, q_k, \boldsymbol{x}_k)$ for some $k \in \mathbb{N}$ or infinite. If $\mathcal{E}$ is finite, then let $\mathbb{N}_{\mathcal{E}} = \{1, \ldots, k\}$, otherwise let $\mathbb{N}_{\mathcal{E}} = \mathbb{N}$. We require that

- $T_i \in \mathbb{I}$ (possibly a singleton[4]), $q_i \in Q$, $\boldsymbol{x}_i : T_i \to \boldsymbol{D}_{q_i}$ for all $i \in \mathbb{N}_{\mathcal{E}}$,
- $\overline{T_i} = \underline{T}_{i+1}$ for all $i, i+1 \in \mathbb{N}_{\mathcal{E}}$, and
- $\overline{T}_{\mathcal{E}} = \overline{T_k}$ or $\overline{T}_{\mathcal{E}} = \lim_{i\to\infty} \overline{T_i}$.

Therefore if $\mathcal{E}$ is finite, then $T_1, \ldots, T_k$ is a partition of $T_{\mathcal{E}}$. If $\mathcal{E}$ is infinite, then $T_1, T_2, \ldots$ is a partition of either $[\underline{T}_{\mathcal{E}}, \overline{T}_{\mathcal{E}})$ or $T_{\mathcal{E}}$, corresponding, respectively, to a genuine Zeno or chattering Zeno evolution [61].

Moreover, introducing $t_0 \stackrel{\text{def}}{=} \underline{T}_1$ and $t_i \stackrel{\text{def}}{=} \overline{T}_i$, $i \in \mathbb{N}_{\mathcal{E}}$, the following conditions hold

- $\boldsymbol{x}_i$ is continuous on $[t_{i-1}, t_i] = T_i$ for $i \in \mathbb{N}_{\mathcal{E}}$,
- $\boldsymbol{x}_i$ is differentiable and $\boldsymbol{x}_i'(t) = \boldsymbol{f}_{q_i}(\boldsymbol{x}_i(t))$[5] on (t_{i-1}, t_i) for $i \in \mathbb{N}_{\mathcal{E}}$,
- $q_i = \sigma(e_i)$, $q_{i+1} = \tau(e_i)$ for each event occurrence e_i,
- $\boldsymbol{x}_i(t_i) \in \boldsymbol{C}_{e_i}$ and $\boldsymbol{x}_{i+1}(t_i) = \boldsymbol{r}_{e_i}(\boldsymbol{x}_i(t_i))$. □

Intuitively, the evolution of a hybrid automaton begins by solving an ODE IVP based on the vector field $\boldsymbol{f}_{q_1}$ of the initial mode $q_1 \in Q$, proceeding until the solution intersects the guard $\boldsymbol{C}_{e_1}$ of some event e_1 with $q_1 = \sigma(e_1)$. This may trigger an event of type e_1 and thus cause a discrete transition into the mode $q_2 = \tau(e_1)$, after which the system continues to evolve according to $\boldsymbol{f}_{q_2}$ until another event triggers, and so on.

While this definition does not specify the behavior *beyond* a Zeno point, it can still be used to describe the behavior at a Zeno time as a limit. It follows from the definition that $T_{\mathcal{E}}$ may contain at most one such point, namely its right endpoint. If $\overline{T}_{\mathcal{E}}$ is a genuine Zeno time, that is, infinitely many transitions are accumulating towards it from the left, then $\overline{T}_{\mathcal{E}} \notin T_i$ for all $i \in \mathbb{N}_{\mathcal{E}}$. On the other hand, if $\overline{T}_{\mathcal{E}}$ is a chattering Zeno time, then $T_i = [\overline{T}_{\mathcal{E}}, \overline{T}_{\mathcal{E}}]$ for all sufficiently large i.

Note that if the mode invariant is not violated, then the evolution is not forced to make a transition even when a guard becomes active, leading to non-determinism. Thus, to use this non-deterministic semantics for hybrid automata to produce results similar to where events are taken as soon as a guard is crossed, it would be necessary to add invariants that match guards (such as $x \geq 0$ in Fig. 3(a)). Nevertheless, non-determinism cannot be avoided in general. For example, as we shall see later on, perfect localization of every event is unattainable in most cases. Consequently we cannot determine what (possibly infinite) sequence of events takes place within a fixed time segment, even for deterministic models. Despite this complication, we will show that this can be efficiently handled by an enclosure semantics that takes into account every possible transition and produces an *enclosure of the evolution* and not an individual evolution.

---

3 The notations $\sigma(e)$ and $\tau(e)$ will also be used for σ_e and τ_e.

4 A singleton represents an instantaneous transition.

5 In the case of non-autonomous differential equations, we amend the space $\mathbb{R}^n$ with an additional dimension representing time, by which we obtain time-invariant (or autonomous) ODEs.

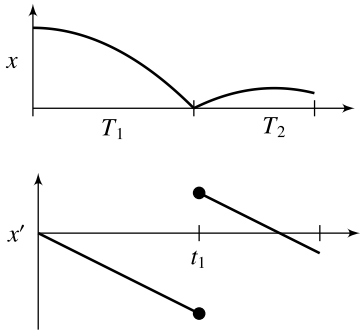


Fig. 5. Trajectory of an evolution of the bouncing ball system, as modeled in Fig. 1. The evolution is $\mathcal{E} = (T_1, \mathit{Fly}, \boldsymbol{x}_1), \mathit{Bounce}, (T_2, \mathit{Fly}, \boldsymbol{x}_2)$, with $T_1 = [0, 1]$ and $T_2 = [1, 1.75]$. Over the intervals $[0, 1)$ and $(1, 1.75]$, the trajectory is a function giving a single pair of values (x, x') to each time point. On $[0, 1)$ $\mathrm{traj}(\mathcal{E})(t) = \{\boldsymbol{x}_1(t)\}$, with $\boldsymbol{x}_1(t) = (-5t^2 + 5, -10t)$ and on $(1, 1.75]\mathrm{traj}(\mathcal{E})(t) = \{\boldsymbol{x}_2(t)\}$, with $\boldsymbol{x}_2(t) = (-5t^2 + 15t - 10, -10t + 15)$. At time $t_1 = \overline{T_1} = \underline{T_2} = 1$, the trajectory consists of both $\boldsymbol{x}_1(t)$ and $\boldsymbol{x}_2(t)$, that is $\mathrm{traj}(\mathcal{E})(t_1) = \{\boldsymbol{x}_1(t_1), \boldsymbol{x}_2(t_1)\}$.

3.3. Enclosures

Definition 3.3 (*Hybrid Automaton States*)**.** Let $\mathcal{H}$ be a hybrid automaton. The pairs $\mathsf{s} \in Q \times \mathbb{R}^n$ and $\mathsf{S} \in Q \times \mathbb{I}^n$ are called *real states* and *box states* of $\mathcal{H}$, respectively. □

Definition 3.4 (*Order of States*)**.** For two box states $(q, \boldsymbol{A})$ and $(q', \boldsymbol{A}')$ we define the order relation $(q, \boldsymbol{A}) \Subset (q', \boldsymbol{A}') \iff q = q'$ and $\boldsymbol{A} \subseteq \boldsymbol{A}'$.

For the real state $\mathsf{s} = (q, \boldsymbol{a}) \in Q \times \mathbb{R}^n$, a box state $\mathsf{S} \in Q \times \mathbb{I}^n$ and sets of box states $\mathcal{S}, \mathcal{S}' \in \mathcal{P}(Q \times \mathbb{I}^n)$ we extend this order as

$$\begin{aligned}
\mathsf{S} \Subset \mathcal{S}' &\iff \exists \mathsf{S}' \in \mathcal{S}'.\ \mathsf{S} \Subset \mathsf{S}',\\
\mathcal{S} \Subset \mathcal{S}' &\iff \forall \mathsf{S}_0 \in \mathcal{S}.\ \mathsf{S}_0 \Subset \mathcal{S}',\\
\mathsf{s} \Subset \mathcal{S}' &\iff (q, \{\boldsymbol{a}\}) \Subset \mathcal{S}'. \quad \square
\end{aligned}$$

Definition 3.5 (*States of an Evolution*)**.** For an evolution $\mathcal{E}$ of $\mathcal{H}$ over $T_\mathcal{E}$ we define the *states of* $\mathcal{E}$ *at* $t \in [\underline{T}_\mathcal{E}, \overline{T}_\mathcal{E})$ as the set $\mathcal{S}_\mathcal{E}(t) = \{(q_i, \boldsymbol{x}_i(t)) \mid t \in T_i, i \in \mathbb{N}_\mathcal{E}\}$.

The states of $\mathcal{E}$ at $\overline{T}_\mathcal{E}$, also called *limit states*, are given as follows.

- If $\mathbb{N}_\mathcal{E} = \{1, \ldots, k\}$ that is $\mathcal{E}$ is finite, then the set of *limit states of* $\mathcal{E}$ is the singleton $\mathcal{S}_\mathcal{E}(\overline{T}_\mathcal{E}) \stackrel{\text{def}}{=} \left\{\left(q_k, \{\boldsymbol{x}_k(\overline{T}_\mathcal{E})\}\right)\right\}$.
- If $\mathbb{N}_\mathcal{E} = \mathbb{N}$ that is $\mathcal{E}$ is infinite, the *limit states of* $\mathcal{E}$ comprise all states $(q, \boldsymbol{A})$ for which there exist infinitely many $i \in \mathbb{N}$ such that $q = q_i$, and $\boldsymbol{A}$ is hull of all ω-limits of the ranges of the corresponding $\boldsymbol{x}_i$-s that is

$$\begin{aligned}
\mathcal{S}_\mathcal{E}(\overline{T}_\mathcal{E}) \stackrel{\text{def}}{=} \Big\{ (q, \boldsymbol{A}) \Big| &\exists (i_k)_{k=1}^\infty : i_k < i_{k+1},\ q = q_{i_k} \text{ for all } k \in \mathbb{N},\\
&\boldsymbol{A} = \mathrm{Hull}\left\{\boldsymbol{A}' \;\middle|\; \exists (k_l)_{l=1}^\infty : k_l < k_{l+1}, \boldsymbol{A}' = \lim_{l\to\infty} \mathrm{Range}\left(\boldsymbol{x}_{i_{k_l}}\right)\right\}\Big\}. \quad \square
\end{aligned}$$

This set of limit states for an infinite evolution $\mathcal{E}$ contains (w.r. to the order $\Subset$) the Zeno set introduced in [9].

Definition 3.6 (*Trajectory of an Evolution*)**.** For an evolution $\mathcal{E}$ over $T_\mathcal{E}$, the trajectory $\mathrm{traj}(\mathcal{E})$ is the set-valued function given by:

$$\mathrm{traj}(\mathcal{E})(t) = \begin{cases} \left\{\boldsymbol{a} \;\middle|\; (q, \boldsymbol{a}) \in \mathcal{S}_\mathcal{E}(t)\right\} & \text{if } t < \overline{T_\mathcal{E}},\\ \cup \left\{\boldsymbol{A} \;\middle|\; (q, \boldsymbol{A}) \in \mathcal{S}_\mathcal{E}(\overline{T}_\mathcal{E})\right\} & \text{if } t = \overline{T_\mathcal{E}}. \quad \square \end{cases}$$

The trajectory of a concrete evolution is illustrated in Fig. 5. Note that the set $\mathrm{traj}(\mathcal{E})(t)$ typically contains two values when there is an event at time t. When there are multiple mode transitions and resets at time t, (corresponding to a sequence of segments $T_j = T_{j+1} = \cdots = [t, t]$), the set can contain more than two elements. The trajectory at the end time $\mathrm{traj}(\mathcal{E})(\overline{T_\mathcal{E}})$ may contain infinitely many elements.

The motivation of the following definition is that we will provide initial states for our simulation and look for evolutions that are consistent with these.

Definition 3.7 (*Hybrid IVP and its Solutions*)**.** A *hybrid IVP* is a tuple $(\mathcal{H}, T, \mathrm{S}_{\mathrm{init}})$ comprising a hybrid automaton $\mathcal{H}$, a time interval $T \in \mathbb{I}$ and a box state $\mathrm{S}_{\mathrm{init}} = (q_{\mathrm{init}}, \boldsymbol{A}_{\mathrm{init}})$.

A *solution* of the hybrid IVP $(\mathcal{H}, T, (q_{\mathrm{init}}, \boldsymbol{A}_{\mathrm{init}}))$, is an evolution $\mathcal{E}$ that satisfies $\underline{T} = \underline{T}_{\mathcal{E}} = \underline{T}_1, q_1 = q_{\mathrm{init}}, \boldsymbol{x}_1(\underline{T}) \in \boldsymbol{A}_{\mathrm{init}}$ and $\overline{T}_{\mathcal{E}} \leq \overline{T}$. □

It is important to note that a hybrid IVP usually has a chain of solutions even if the initial state is a real state due to the freedom to choose the length of $T_{\mathcal{E}}$. To gain uniqueness, one may require that a solution has to be maximal in its time-span $T_{\mathcal{E}}$.

Now we give an important definition that will be used to determine what kind of enclosures we are aiming to obtain for $\mathcal{H}$.

Definition 3.8 (*Enclosure of Solutions and Restarted Evolutions*)**.** Consider a hybrid IVP $\mathcal{M} = (\mathcal{H}, T, \mathrm{S}_{\mathrm{init}})$ and a function $\mathcal{Z} : T_{\mathcal{Z}} \to \mathcal{P}(Q \times \mathbb{I}^n)$ defined over the interval $T_{\mathcal{Z}} = [\underline{T}, \overline{T}_{\mathcal{Z}}] \subseteq T$ that maps to sets of box states and satisfies $\mathrm{S}_{\mathrm{init}} \Subset \mathcal{Z}(\underline{T})$.

If for all evolutions $\mathcal{E}$ over $T_{\mathcal{E}} \subseteq T_{\mathcal{Z}}$ such that $\mathcal{S}_{\mathcal{E}}(\underline{T}_{\mathcal{E}}) \Subset \mathcal{Z}(\underline{T}_{\mathcal{E}})$ and

- $\underline{T}_{\mathcal{E}} = \underline{T}$ and $\mathcal{S}_{\mathcal{E}}(\underline{T}_{\mathcal{E}}) \Subset \mathrm{S}_{\mathrm{init}}$ that is $\mathcal{E}$ is a solution of $\mathcal{M}$ or
  (*encloses solutions*)
- $T_1 = [t_0, t_0]$ that is T_1 is a singleton
  (*encloses restarted evolutions*)

it holds that $\mathcal{S}_{\mathcal{E}}(t) \Subset \mathcal{Z}(t)$ for all $t \in T_{\mathcal{E}}$, then we call $\mathcal{Z}$ an *enclosure of solutions and restarted evolutions of* $\mathcal{M}$ *over* $T_{\mathcal{Z}}$. □

This is a consistency statement between the function $\mathcal{Z}$ and the hybrid IVP $\mathcal{M}$. The *solutions* are allowed to have non-singleton initial segments in contrast to the *restarted evolutions* that immediately undergo a transition through an event as they are of the form $\mathcal{E} = ([t_0, t_0], q_1, \boldsymbol{x}_1), e_1, (T_2, q_2, \boldsymbol{x}_2), e_2, \ldots$

The role of restarted evolutions is to track what happens after a solution (or another, already enclosed, restarted evolution) reaches one of its limit states. The beginning of $\mathcal{E}$ may be considered as a restart from the initial state $(q_1, \boldsymbol{x}_1(t))$ at time t through the event e_1.

4. Event detection and handling

In this section we present a procedure to obtain an enclosure of solutions and restarted evolutions for a hybrid IVP. We begin by defining the basic operations for the detection of individual events (Section 4.1). Next, we consider the effect of a single event and how the associated change in dynamics may be captured (Section 4.2). We then proceed with enclosing multiple events in a given time interval T without addressing localization (Section 4.3). Note that this method is compatible with advanced localization techniques [62–66].

4.1. Detecting the next event

We now define the functions **possible-events** and **event-is-necessary** that will provide information (without further subdivision of the interval) about events occurring in a time interval T under consideration.

Definition 4.1. For an interval function $\boldsymbol{Y} : T \to \mathbb{I}^n$, and mode $q \in Q$, let

$$\textbf{possible-events}(\mathcal{H}, \boldsymbol{Y}, q) \stackrel{\text{def}}{=} \left\{ e \in E \;\middle|\; \sigma(e) = q, \mathrm{Range}(\boldsymbol{Y}) \cap C_e \neq \emptyset \right\},$$

$$\textbf{event-is-necessary}(\mathcal{H}, \boldsymbol{Y}, q) \stackrel{\text{def}}{=} \begin{cases} \text{true} & \text{if } \boldsymbol{Y}(\overline{T}) \cap \boldsymbol{D}_q = \emptyset, \\ \text{false} & \text{otherwise.} \end{cases}$$ □

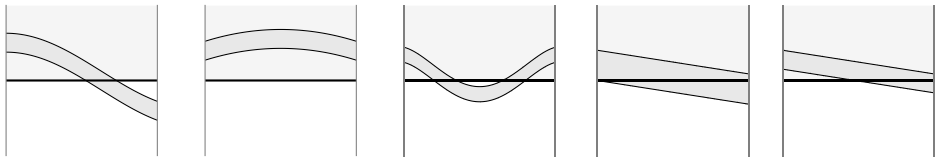
We shall use these functions upon establishing that $\boldsymbol{Y}$ encloses certain evolutions of $\mathcal{H}$. $\mathrm{Range}(\boldsymbol{Y}) \cap \boldsymbol{C}_e \neq \emptyset$ represents the possibility of event e as the range of the enclosure crosses its guard. $\boldsymbol{Y}(\overline{T}) \cap \boldsymbol{D}_q = \emptyset$ signals that $\boldsymbol{Y}$ is outside the mode invariant D_q at the right endpoint of T. Thus, any evolution that is in mode q at time $\underline{T}$ and enclosed by $\boldsymbol{Y}$, must have transitioned before $\overline{T}$.

Fig. 6 illustrates the three cases that we are able to distinguish using these two functions. **event-is-necessary** distinguishes case (a) from cases (b) and (c), while **possible-events** $\subseteq E$ returns a set that restricts the type of the (potential) event. Case (b) corresponds to the situation when this set is empty.

Proposition 4.2 (*Inclusion Isotonicity of Event Detection*)**.** *Assume* $\boldsymbol{Y}_1 \subseteq \boldsymbol{Y}_2$ *(i.e.,* $\boldsymbol{Y}_1$ *is more informative than* $\boldsymbol{Y}_2$*).*

(1) **possible-events**$(\mathcal{H}, \boldsymbol{Y}_2, q) \supseteq$ **possible-events**$(\mathcal{H}, \boldsymbol{Y}_1, q)$
(2) **event-is-necessary**$(\mathcal{H}, \boldsymbol{Y}_2, q) \Longrightarrow$ **event-is-necessary**$(\mathcal{H}, \boldsymbol{Y}_1, q)$.

Proof. (1) and (2) follow from the fact that all appearing operations are inclusion isotone, namely the intersection with a fixed set and the range of a function. □



(a) Certain event. (b) No event. (c) Maybe event.

Fig. 6. Three cases of event detection without further localization. In each plot, the upper half is the mode invariant and the horizontal line dividing the upper and lower half is the guard. While the possibility of an event is detected whenever the guard is crossed, the event is determined with certainty only when the whole enclosure lies outside the mode invariant at the endpoint of the given time interval.

Intuitively, Proposition 4.2 states that improving the enclosure of evolutions, thus improving the range and end point enclosure, can only preserve or improve the knowledge extracted by the two functions on event occurrences in T.

We now discuss how these functions can provide information on an evolution $\mathcal{E}$ after we have established the potential occurrence of its first $\ell \geq 0$ events, and the existence of an interval function $\boldsymbol{Y}$ enclosing $\mathcal{E}$ over the time interval $T_{\ell+1}$.

Proposition 4.3 (*Soundness of Event Detection*)**.** *Let $\mathcal{E}$ be an evolution of $\mathcal{H}$ over time $T_{\mathcal{E}} \subseteq T$ with ℓ or more events (that is $\ell \in \mathbb{N}_{\mathcal{E}}$) and assume that the interval function $\boldsymbol{Y} : T \to \mathbb{I}^n$ encloses $\mathcal{E}$ over $T_{\ell+1}$ that is $\boldsymbol{x}_{\ell+1}(t) \in \boldsymbol{Y}(t)$ for all $t \in T_{\ell+1}$.*

(1) *If* **event-is-necessary**$(\mathcal{H}, \boldsymbol{Y}, q_{\ell+1})$*, then either $\ell + 1 \in \mathbb{N}_{\mathcal{E}}$ (there are at least $\ell + 1$ events) or $\overline{T}_{\mathcal{E}} < \overline{T}$.*
(2) *If $\ell + 1 \in \mathbb{N}_{\mathcal{E}}$, then $e_{\ell+1} \in$* **possible-events**$(\mathcal{H}, \boldsymbol{Y}, q_{\ell+1})$.

Proof. Set $q = q_{\ell+1}$ and $\boldsymbol{x} = \boldsymbol{x}_{\ell+1}$.

(1) We have $\boldsymbol{x}(t) \in \boldsymbol{D}_q$ and $\boldsymbol{x}(t) \in \boldsymbol{Y}(t)$ for all $t \in T_{\ell+1}$. As $\boldsymbol{Y}(\overline{T}) \cap \boldsymbol{D}_q = \emptyset$ holds, we get $\overline{T} \notin T_{\ell+1}$. If $\mathcal{E}$ has exactly ℓ events, that is $\mathbb{N}_{\mathcal{E}} = \{1, \ldots, \ell\}$, then $\overline{T}_{\mathcal{E}} = \overline{T}_{\ell+1}$, thus $\overline{T}_{\mathcal{E}} < \overline{T}$.
(2) Let $e = e_{\ell+1}$ and $t = t_{\ell+1}$. By the last two items in Definition 3.2, we get $\sigma(e) = q$ and $\boldsymbol{x}(t) \in \boldsymbol{C}_e$. The latter implies that $\mathrm{Range}(\boldsymbol{Y}) \cap C_e \neq \emptyset$. Therefore $e \in$ **possible-events** in Definition 4.1. □

If **possible-events** is empty while **event-is-necessary** for the same parameters, it follows that any evolution enclosed by $\boldsymbol{Y}$ must terminate before $\overline{T}$.

4.2. Enclosing one event

In this section we consider how the behavior of an evolution after an event occurrence might be captured. The procedure will require a validated (that is, enclosure producing) ODE IVP solver. We assume that **solve-ivp** is inclusion isotone with respect to the initial condition, for example one of those reviewed in [31], including VNODE-LP [45] and COSY INFINITY [67]. The presented results are compatible with any of these well-established validated solvers.[6]

Definition 4.4 (*Enclose-One-Event*)**.** Given a box state $S = (q, \boldsymbol{A})$ and an event $e \in E$ with $\sigma(e) = q$, let

$$\textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{A}, e) \stackrel{\text{def}}{=} \mathrm{Hull}\big(\boldsymbol{R} \cap \boldsymbol{D}_{\tau(e)}\big),$$

where $\boldsymbol{R} = \mathrm{Range}\big(\textbf{solve-ivp}(\boldsymbol{f}_{\tau(e)}, T, \boldsymbol{A}')\big)$ with $\boldsymbol{A}' = \mathrm{Hull}\Big(\boldsymbol{D}_{\tau(e)} \cap \boldsymbol{r}_e\big(\boldsymbol{A} \cap \boldsymbol{C}_e\big)\Big)$. □

Fig. 7 illustrates obtaining an enclosure for the behavior during and after an event for the bouncing ball. Steps (c)–(h) are encapsulated in **enclose-one-event**.

We now investigate the isotonicity properties of the function introduced above.

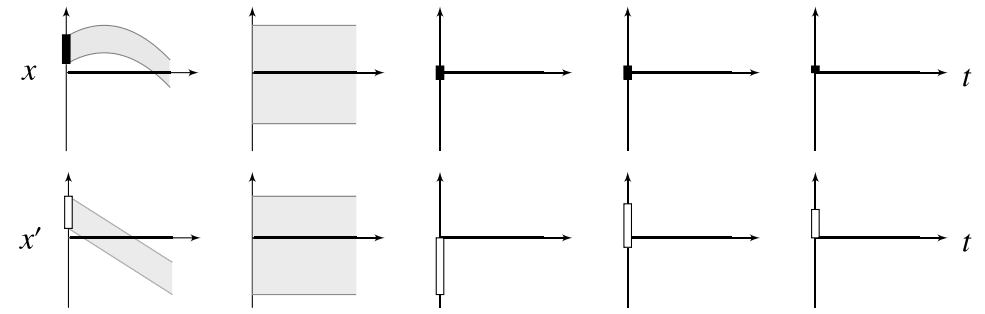
Proposition 4.5 (*Inclusion Isotonicity of Event Enclosures*)**.** *When $\boldsymbol{A}_1 \subseteq \boldsymbol{A}_2$, then*

$$\textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{A}_1, e) \subseteq \textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{A}_2, e).$$

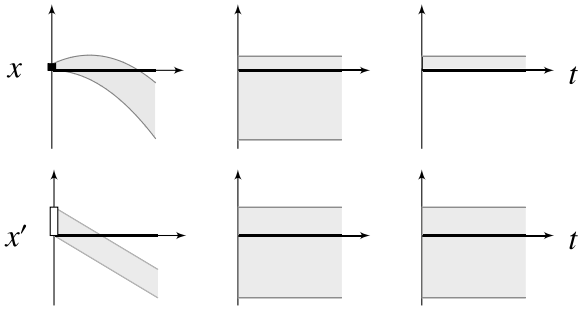
Proof. The inclusion isotonicity of **enclose-one-event** follows from inclusion isotonicity of all operations that it is composed of. Namely, intersection with a fixed set, the reset map applied to interval boxes, the interval hull, **solve-ivp** with respect to the initial condition and the range of a function. □

The following proposition establishes what information we gain about an evolution that undergoes a transition.

6 The enclosures plotted in Fig. 4 have been computed using a simple ODE IVP solver based on the interval Picard operator [68,69]. More advanced solvers (coupled with sophisticated data representations) will produce better results.



(a) Event detected. (b) Simplify. (c) ∩ w. guards. (d) Reset. (e) ∩ w. invar.



(f) After event. (g) Time blur. (h) ∩ w. invar.

Fig. 7. Enclosing behavior during and after an event. (a) **solve-ivp** $= \boldsymbol{Y} : T \to \mathbb{I}^n$ that encloses the trajectory up to the first event. **possible-events** $= \{e = \text{Bounce}\}$ as the guard $x = 0 \wedge x' \leq 0$ is intersected. **event-is-necessary** $=$ false as $\boldsymbol{Y}(\overline{T})$ intersects the mode invariant $x \geq 0$. (b) We simplify by taking the range $\boldsymbol{A}$ of the interval function. (c) $\boldsymbol{A} \cap \boldsymbol{C}_e$: the range is intersected with the guard $\boldsymbol{C}_e$ for the event. (d) $\boldsymbol{r}_e(\boldsymbol{A} \cap \boldsymbol{C}_e)$: the reset $\boldsymbol{r}_e$ for the event is applied on the intersection. This incurs a noticeable over-approximation. (e) $\boldsymbol{A}' = \text{Hull}(\boldsymbol{D}_{\tau(e)} \cap \boldsymbol{r}_e(\boldsymbol{A} \cap \boldsymbol{C}_e))$: the result is intersected with the mode invariant for the target mode and the hull is taken. (f) **solve-ivp**$(\boldsymbol{f}_{\tau(e)}, T, \boldsymbol{A}')$: the hull is used as an initial condition for the evolution in the target mode. (g) $\boldsymbol{R} = \text{Range}\big(\textbf{solve-ivp}(\boldsymbol{f}_{\tau(e)}, T, \boldsymbol{A}')\big)$: as the time of the event is uncertain, the range is taken to obtain a safe enclosure of the post-event behavior. (h) **enclose-one-event**$(\mathcal{H}, T, \boldsymbol{A}, e) = \text{Hull}\big(\boldsymbol{R} \cap \boldsymbol{D}_{\tau(e)}\big)$: the range is intersected with the target mode invariant, then the hull is taken.

Proposition 4.6 (*Soundness of **Enclose-One-Event***). *Let $\mathcal{E}$ be an evolution of the hybrid IVP $\mathcal{M}$ over $T_{\mathcal{E}} \subseteq T$. Assume that $\boldsymbol{x}_i(t_i) \in \boldsymbol{A}_i \in \mathbb{I}^n$ for all $i \in \mathbb{N}_{\mathcal{E}}$.*

Then, $\text{Range}(\boldsymbol{x}_{i+1}) \subseteq$ **enclose-one-event**$(\mathcal{H}, T, \boldsymbol{A}_i, e_i)$ *for $i \geq 1$ such that $i + 1 \in \mathbb{N}_{\mathcal{E}}$.*

Proof. Recall that **enclose-one-event**$(\mathcal{H}, T, \boldsymbol{A}_i, e_i) = \text{Hull}\big(\boldsymbol{R}_i \cap \boldsymbol{D}_{q_{i+1}}\big)$ and by definition $\text{Range}(\boldsymbol{x}_{i+1}) \subseteq \boldsymbol{D}_{q_{i+1}}$. Thus, we need to show that

$$\text{Range}(\boldsymbol{x}_{i+1}) \subseteq \boldsymbol{R}_i = \text{Range}\big(\textbf{solve-ivp}(\boldsymbol{f}_{q_{i+1}}, T, \boldsymbol{A}_i')\big) \quad \text{if } i, i+1 \in \mathbb{N}_{\mathcal{E}}.$$

The inclusion isotonicity in the initial condition of **solve-ivp** implies that establishing $\boldsymbol{x}_{i+1}(t_i) \in \boldsymbol{A}_i' = \text{Hull}\big(\boldsymbol{D}_{q_{i+1}} \cap \boldsymbol{r}_{e_i}(\boldsymbol{A}_i \cap \boldsymbol{C}_{e_i})\big)$ is sufficient. From Definition 3.2 we have that $x_i(t_i) \in \boldsymbol{C}_{e_i}$ and $\boldsymbol{r}_{e_i}(\boldsymbol{x}_i(t_i)) = x_{i+1}(t_i) \in \boldsymbol{D}_{q_{i+1}}$, therefore the claim follows from $\boldsymbol{x}_i(t_i) \in \boldsymbol{A}_i$. □

4.3. Enclosing multiple events

Now we discuss how multiple event occurrences within the time interval T can be handled using the results of Sections 4.1 and 4.2. Let us recall that no localization shall take place.[7]

The key idea is that, after handling the first event, repeating

1. **possible-events** detects the possible subsequent events still in T,
2. **enclose-one-event** handles each of these events.

We can repeat these two steps until 2 results in such states that are subsets of others handled in a previous step. This is illustrated in Fig. 8.

Now we introduce event sequences. They represent series of events taking place within the same time interval T.

[7] This only leads to an over-approximation proportional to the size of the interval T. A dynamic time stepping strategy may ensure that the step size is systematically reduced until the precision of the outgoing trajectory is optimal.

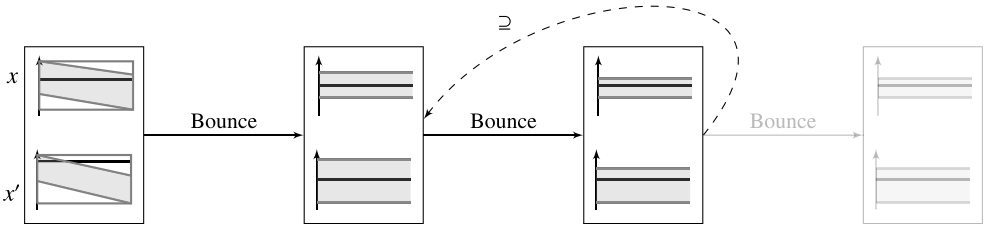


Fig. 8. Enclosing multiple events for the bouncing ball model given in Fig. 3(a). One and two bounces are explicitly handled. More than two bounces are represented through an implicit loop corresponding to the inclusion.

Definition 4.7 (*Finite Sequences of Events*)**.** Consider a hybrid IVP $\mathcal{M}$ and let E^* denote the set of *finite sequences of events* from E where ϵ is the empty sequence and vw is the concatenation of sequences $v, w \in E^*$, thus $\epsilon v = v\epsilon = v$ for all $v \in E^*$. The notation for source and target modes is extended to non-empty event sequences as $\sigma(ev) = \sigma(e)$ and $\tau(ve) = \tau(e)$ for each $e \in E$, $v \in E^*$. Furthermore, we set $\sigma(\epsilon) = \tau(\epsilon) = q_{\text{init}}$. □

To deal with the uncertainty about the order of events in T, we will work with a certain subset of E^* that is prefix-closed, thus, it is equivalent to a tree whose branching is determined by possible next events.

Definition 4.8 (*Event Tree*)**.** An *event tree* for the hybrid IVP $\mathcal{M}$ is a pair $(V, \boldsymbol{\mu})$ whose components are as follows:

- $V \subseteq E^*$ a set of event sequences
- $\boldsymbol{\mu} : V \to \mathbb{I}^n$ an interval box for each sequence.

An event tree is called *valid* if for every $vw \in V$:

- $\sigma(w) = \tau(v)$, *i.e.*, sequences in V respect modes, and
- $v \in V$, *i.e.*, V is prefix-closed. □

The box $\boldsymbol{\mu}(v)$ associated with the event sequence v of an event tree will be interpreted as an enclosure of evolution segments within the mode $\tau(v)$ by introducing the constant interval function $\boldsymbol{Y}_T^{\mu(v)} : t \mapsto \mu(v)$, where $T \in \mathbb{I}$ is a given interval and $t \in T$.

We now introduce the construction of a certain event tree that, as we will see, encloses all solutions of a hybrid IVP together with their limit states. Crucially, this tree may be finite even in the presence of Zeno behavior.

Definition 4.9 (*Event Tree Construction*)**.** For a hybrid IVP $\mathcal{M} = (\mathcal{H}, T, (q_{\text{init}}, \boldsymbol{A}_{\text{init}}))$ let

$$\textbf{construct-event-tree}(\mathcal{M}) \stackrel{\text{def}}{=} (V, \boldsymbol{\mu})$$

where the components of the tree are constructed using the following algorithm:

1. $L := [\epsilon]$, $V := \{\epsilon\}$, $\boldsymbol{\mu}(\epsilon) := \text{Range}\big(\textbf{solve-ivp}(\boldsymbol{f}_{q_{\text{init}}}, T, \boldsymbol{A}_{\text{init}})\big)$
2. While the list L is not empty, repeat the following:
   2.1. $v :=$ first element of L, remove v from L
   2.2. If $\boldsymbol{\mu}(v) \subseteq \boldsymbol{\mu}(w)$ for some $w \in V$ shorter than v with $\tau(v) = \tau(w)$, go to 2.
   2.3. For each $e \in \textbf{possible-events}(\mathcal{H}, \boldsymbol{Y}_T^{\mu(v)}, \tau(v))$, repeat:
      2.3.1. Add ve to V, set $\boldsymbol{\mu}(ve) := \textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{\mu}(v), e)$.
      2.3.2. Add ve to the end of L.

Note that in step 2.3, the set of possible events may be empty, in which case the node v will have no children. For example, when there are no events on T and the enclosure is tight enough so that their absence can be detected, the tree has only one node. Similarly, one can obtain finite trees in some cases when there are only a small number of events on T.

In case of Zeno behaviors, the termination condition for the algorithm amounts to a fixed point being reached during **construct-event-tree** as a result of contracting hybrid dynamics (step 2.2, illustrated in Fig. 8). Since such a fixed point may fail to exist, the implementation is instrumented with an additional termination heuristic to avoid divergence. The heuristic we currently use is a tree size threshold. Upon exceeding this, the computation is aborted.

We recall that $\boldsymbol{\mu}(v)$ and $\boldsymbol{\mu}(w)$ are boxes, thus the check for inclusion in step 2.2 is computationally feasible. At any point in the procedure, whenever we are considering a particular sequence of events, we have already dealt with all possible shorter sequences as the tree is constructed in a breadth-first manner. Step 2.2 checks whether there is any shorter sequence w for which (i) the target mode is the same, and (ii) the box enclosure for w is a superset of the box enclosure for the current sequence v.

In the linear, non-branching event tree depicted in Fig. 8, the two leftmost nodes constitute such a shorter sequence, as the interval in the second node contains the interval in the third node and the system has only one mode. Finding such a sequence w means that we have reached a fixed point in the analysis; any events that follow the current sequence v also

follow the shorter sequence w and thus are enclosed by some node in the tree. In other words, any events that follow v will not give rise to a trajectory outside the already computed interval functions. This insight is what allows us to compute solution enclosures over a given interval without the need to individually handle every event that occurs in that interval, and is the key to our method for handling some classes of Zeno systems.

In what follows, we consider the hybrid IVP $\mathcal{M} = (\mathcal{H}, T, (q_{\text{init}}, \boldsymbol{A}_{\text{init}}))$ and assume that the event tree $(V, \boldsymbol{\mu}) = \textbf{construct-event-tree}(\mathcal{M})$ is successfully constructed.

The following lemma establishes that even if an event sequence is skipped by step 2.2, the successive events are enclosed.

Lemma 4.10 (*Event Tree Folding*)**.** *Let $v \in V$ and assume that*

$$e \in \textbf{possible-events}(\mathcal{H}, \boldsymbol{Y}_T^{\mu(v)}, \tau(v)).$$

Then, there exists $v' \in V \setminus \{\epsilon\}$ such that $\tau(v') = \tau(e)$ and

$$\textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{\mu}(v), e) \subseteq \boldsymbol{\mu}(v').$$

Proof. We prove by induction based on the length of $v \in V$.

Note that $v = \epsilon$ was surely not skipped in step 2.2 of Definition 4.9 during the construction as it is the shortest element in V. Therefore step 2.3.1 was executed for each $e \in \textbf{possible-events}(\mathcal{H}, \boldsymbol{Y}_T^{\mu(\epsilon)}, \tau(\epsilon))$, thus the claim holds as equality with $v' = e$.

Consider now $v \in V$, $e \in E$ as above and assume the following:

(*IH*) The statement is satisfied for any u shorter than v.

If v was not skipped in step 2.2, then our claim holds with $v' = ve$ since step 2.3.1 is executed for e. If v was skipped due to the existence of a shorter w, then we get

$$\begin{aligned} e &\in \textbf{possible-events}(\mathcal{H}, \boldsymbol{Y}_T^{\mu(v)}, \tau(v)) \\ &\subseteq \textbf{possible-events}(\mathcal{H}, \boldsymbol{Y}_T^{\mu(w)}, \tau(w)) \end{aligned}$$

using $\tau(v) = \tau(w)$, $\boldsymbol{\mu}(v) \subseteq \boldsymbol{\mu}(w)$ and Proposition 4.2. Setting $v' = w'$, where $w' \in V \setminus \{\epsilon\}$ is given by *(IH)*, establishes our claim since

$$\begin{aligned} &\textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{\mu}(v), e) \\ &\subseteq \textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{\mu}(w), e) && (\boldsymbol{\mu}(v) \subseteq \boldsymbol{\mu}(w) \text{ and Proposition 4.5}) \\ &\subseteq \boldsymbol{\mu}(w') \quad \text{and} \quad \tau(w') = \tau(e) && (IH). \quad \square \end{aligned}$$

We now state in what manner the event tree is able to enclose a certain type of evolutions of $\mathcal{H}$.

Proposition 4.11 (*Event Tree Encloses Evolutions*)**.** *Assume that $v_1 \in V$ and*

$$\mathcal{E} = (T_1, q_1, \boldsymbol{x}_1), e_1, (T_2, q_2, \boldsymbol{x}_2), e_2, \ldots$$

is an evolution of $\mathcal{H}$ over $T_{\mathcal{E}} \subseteq T$ such that $\mathcal{S}_{\mathcal{E}}(t) \Subset (\tau(v_1), \boldsymbol{\mu}(v_1))$ for all $t \in T_1$.

Then, for all $i \in \mathbb{N}_{\mathcal{E}}$, there exists $v_i \in V$ such that $\mathcal{S}_{\mathcal{E}}(t) \Subset (\tau(v_i), \boldsymbol{\mu}(v_i))$ for all $t \in T_i$.

Proof. We proceed by induction. The statement is trivial for $i = 1$.

The induction case. $e_i \in \textbf{possible-events}(\mathcal{H}, \boldsymbol{Y}_T^{\mu(v_i)}, q_i)$ follows from Proposition 4.3 as the interval function $t \mapsto \boldsymbol{\mu}(v_i)$ encloses $\boldsymbol{x}_i(t)$ on T_i. Thus, by applying Lemma 4.10 with the event e_i, there exists $v_{i+1} \stackrel{\text{def}}{=} v_i' \in V$ such that

$$\textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{\mu}(v_i), e_i) \subseteq \boldsymbol{\mu}(v_{i+1}).$$

As $\boldsymbol{x}_i(\overline{T}_i) \in \boldsymbol{\mu}(v_i)$, we get that $\boldsymbol{x}_{i+1}(t) \in \textbf{enclose-one-event}(\mathcal{H}, T, \boldsymbol{\mu}(v_i), e_i)$ holds for every $t \in T_{i+1}$ using Proposition 4.6. Consequently, $\boldsymbol{x}_{i+1}(t) \in \boldsymbol{\mu}\big(v_{i+1}\big)$ for every $t \in T_{i+1}$ is satisfied. $\square$

Note that in order to establish the property for $i \geq 2$, it is sufficient to show that the state $(q_1, \boldsymbol{x}_1(\overline{T}_1))$ is enclosed by the event tree, that is, there exists $v_1 \in V$ such that $\tau(v_1) = q_1$ and $\boldsymbol{x}_1(\overline{T}_1) \in \boldsymbol{\mu}(v_1)$. This is analogous with the requirements for enclosing of restarted evolutions in Definition 3.8.

The event following the initial segment (that is enclosed by assumption) is essential as **solve-ivp** is always called over the whole time interval T in absence of further localization. This way, we can be sure that we do not consider any integration longer than T within one mode. This clearly reflects step 2.3 in the construction algorithm. There, the already computed $\boldsymbol{\mu}(v)$

encloses continuous behaviors of length T within mode $\tau(v)$. The construction continues by processing the set of possible events.

Turning to solutions, it is a straightforward consequence of Proposition 4.11 that they are guaranteed to be enclosed.

Corollary 4.12 (*Event Tree Encloses Solutions*)**.** *Consider a solution of $\mathcal{M}$:*

$$\mathcal{E} = (T_1, q_1, \boldsymbol{x}_1), e_1, (T_2, q_2, \boldsymbol{x}_2), e_2, \ldots$$

Then, for all $i \in \mathbb{N}_{\mathcal{E}}$, there exists $v_i \in V$ such that $\mathcal{S}_{\mathcal{E}}(t) \Subset (\tau(v_i), \boldsymbol{\mu}(v_i))$ for all $t \in T_i$.

Proof. Recall that $\boldsymbol{\mu}(\epsilon) = \mathrm{Range}\big(\textbf{solve-ivp}(\boldsymbol{f}_{q_{\mathrm{init}}}, T, \boldsymbol{A}_{\mathrm{init}})\big)$. Setting $v_1 \stackrel{\mathrm{def}}{=} \epsilon \in V$, we get $\tau(\epsilon) = q_{\mathrm{init}} = q_1$ and $\boldsymbol{x}_1(t) \in \boldsymbol{\mu}(v_1)$ for all $t \in T_1$.

For $i \geq 2$, apply Proposition 4.11. □

Another important feature of the event tree is that whenever it encloses an evolution, we gain information about the corresponding limit states.

Proposition 4.13 (*Event Tree Encloses Limit States*)**.** *Assume that*

$$\mathcal{E} = (T_1, q_1, \boldsymbol{x}_1), e_1, (T_2, q_2, \boldsymbol{x}_2), e_2, \ldots$$

is an evolution of $\mathcal{H}$ such that $T_{\mathcal{E}} \subseteq T$ and for every $i \in \mathbb{N}_{\mathcal{E}}$ there exists $v_i \in V$ such that $\mathcal{S}_{\mathcal{E}}(t) \Subset (\tau(v_i), \boldsymbol{\mu}(v_i))$ for all $t \in T_i$.

Then, for every limit state $S \in \mathcal{S}_{\mathcal{E}}(\overline{T}_{\mathcal{E}})$ there exists $v \in V$ such that $S \Subset (\tau(v), \boldsymbol{\mu}(v))$.

Proof. We distinguish two cases based on the finiteness of $\mathcal{E}$.

Case $\mathcal{E}$ is finite. Assume that $\mathbb{N}_{\mathcal{E}} = k$. Recall from Definition 3.5 that $\mathcal{S}_{\mathcal{E}}(\overline{T}_{\mathcal{E}})$ contains only $\Big(q_k, \{\boldsymbol{x}_k(\overline{T}_{\mathcal{E}})\}\Big)$, thus set $v \stackrel{\mathrm{def}}{=} v_k$.

Case $\mathcal{E}$ is infinite. Pick a limit state $(q, \boldsymbol{A}) \in \mathcal{S}_{\mathcal{E}}(\overline{T}_{\mathcal{E}})$. According to Definition 3.5, there exists a sequence $(i_k)_{k\in\mathbb{N}}$ such that $i_k \to \infty$, $q = q_{i_k}$ and $\boldsymbol{A}$ is the limit of the ranges of the corresponding $\boldsymbol{x}_{i_k}$-s.

We have $\boldsymbol{\mu}(v_{i_k}) \supseteq \mathrm{Range}(\boldsymbol{x}_{i_k}) \to \boldsymbol{A}$. As V is finite, the infinite sequence $\{v_{i_k}\}_{k\in\mathbb{N}} \subseteq V$ contains only a finite number of different event sequences. Since any $\boldsymbol{\mu}(v_{i_k}) \in \mathbb{I}^n$ is a compact set, we are able to choose a $v \in \{v_{i_k}\}_{k\in\mathbb{N}}$ (any suffices that occurs infinitely many times) such that $\boldsymbol{A} \subseteq \boldsymbol{\mu}(v)$ and $\tau(v) = q$ hold. □

Note that, given that a limit state $(q, \boldsymbol{A}) \in \mathcal{S}_{\mathcal{E}}(\overline{T}_{\mathcal{E}})$ is enclosed as above, and that there exists an event immediately connecting it with another evolution $\mathcal{E}'$ defined over $T_{\mathcal{E}'} = [\overline{T}_{\mathcal{E}}, \overline{T}_{\mathcal{E}'}] \subseteq T$, in the sense that $([\underline{T}_{\mathcal{E}'}, \underline{T}_{\mathcal{E}'}], q, a), e, \mathcal{E}'$ is an evolution of $\mathcal{H}$ for some $a \in \boldsymbol{A}$, then this new evolution is enclosed according to Proposition 4.11. This behavior is present in the *more-complete* Zeno models in [22] where, starting from Zeno states, successive evolutions arise due to an initial transition.

Moreover, since $(\tau(v), \boldsymbol{\mu}(v)) \in Q \times \mathbb{I}^n$ for every $v \in V$, the construction algorithm actually computes box states of $\mathcal{H}$. According to Proposition 4.11, if from any such state, by an initial event, an evolution exists over $T_{\mathcal{E}} \subseteq T$, then it is enclosed together with its limit states.

We may summarize all the important properties proven in Section 4.3 as follows.

Theorem 4.14. *Let $\mathcal{Z} : T \to \mathcal{P}(Q \times \mathbb{I}^n)$ be $\mathcal{Z} = t \mapsto \{(\tau(v), \boldsymbol{\mu}(v)) \mid v \in V\}$. Then, $\mathcal{Z}$ is an enclosure of solutions and restarted evolutions of $\mathcal{M}$.*

Note that if step 2.2 (event tree folding) was not executed during the construction of the enclosure $\mathcal{Z}$ for the hybrid IVP $\mathcal{M} = (\mathcal{H}, T, (q_{\mathrm{init}}, \boldsymbol{A}_{\mathrm{init}}))$, then Zeno behavior is absent from the time interval T. Note also that the execution of step 2.2 does *not* imply the existence of a Zeno point, only its possibility.

5. Enclosing in multiple steps

Having formally expressed these key properties of an event tree $(V, \boldsymbol{\mu})$, constructed for a given hybrid IVP over T, we present our algorithm for producing enclosures of solutions and restarted evolutions over multiple time segments.

First, we define an algorithm called **enclose-evolutions-step** that constructs the event tree for a given hybrid IVP over T. Thereafter, it computes an enclosure of solutions and restarted evolutions of the hybrid IVP and a set of box states that enclose some of their limit states. A limit state is enclosed whenever it belongs to an evolution $\mathcal{E}$ that satisfies $\overline{T}_{\mathcal{E}} = \overline{T}$, thus the state will serve as a new initial condition for the subsequent time segment.

Definition 5.1. For any hybrid IVP $\mathcal{M} = (\mathcal{H}, T, (q_{\text{init}}, \boldsymbol{A}_{\text{init}}))$ let

$$\begin{aligned}
&\textbf{enclose-evolutions-step}(\mathcal{M}) \stackrel{\text{def}}{=} (\mathcal{Z}, \mathcal{S}),\\
\text{with}\quad &\mathcal{Z}(t) = \mathcal{Z}_V(t) \cup \left\{\left(q_{\text{init}}, \boldsymbol{Y}_\epsilon(t)\right)\right\}\\
&\mathcal{S} = \text{MergeByMode}\left(\mathcal{Z}_V\left(\overline{T}\right) \cup \mathcal{S}_\epsilon\right)\\
\text{where}\quad &\boldsymbol{Y}_\epsilon = \textbf{solve-ivp}(\boldsymbol{f}_{q_{\text{init}}}, T, \boldsymbol{A}_{\text{init}}),\\
&(V, \boldsymbol{\mu}) = \textbf{construct-event-tree}(\mathcal{M}),\\
&\mathcal{Z}_V(t) = \left\{\left(\tau(v), \boldsymbol{\mu}(v)\right) \;\middle|\; v \in V, v \neq \epsilon\right\}\\
&\mathcal{S}_\epsilon = \begin{cases}\emptyset & \text{if } \textbf{event-is-necessary}(\mathcal{H}, \boldsymbol{Y}_\epsilon, q_{\text{init}})\\ \left\{\left(q_{\text{init}}, \boldsymbol{Y}_\epsilon\left(\overline{T}\right)\right)\right\} & \text{otherwise,}\end{cases}\\
&\text{MergeByMode}\left\{\left(q_i, \boldsymbol{A}_i\right)\right\}_{i\in I} \stackrel{\text{def}}{=} \left\{\left(q, \boldsymbol{A}_q\right) \;\middle|\; q \in \{q_i\}_{i\in I}, \boldsymbol{A}_q = \text{Hull}\{\boldsymbol{A}_i \mid q_i = q\}\right\}. \quad \square
\end{aligned}$$

Note that the extra information we incorporate into $\mathcal{Z}$ compared to Theorem 4.14 is the enclosure $\boldsymbol{Y}_\epsilon$ of the trajectories, as opposed to their range within the initial segment of the solutions. If, instead of using $\boldsymbol{Y}_\epsilon$ in the definition of $\mathcal{S}_\epsilon$, we were to use $\boldsymbol{Y}_T^{\mu(\epsilon)}$, then **event-is-necessary** would always return false given that $t \mapsto \boldsymbol{\mu}(\epsilon)$ is a constant enclosure, the range of $\boldsymbol{Y}_\epsilon$, thus the initial condition is enclosed always and the certainty of an event cannot be established.

When **event-is-necessary** returns true, the function $\boldsymbol{Y}_\epsilon$ is excluded from the definition of $\mathcal{S}$. This happens if it is established that at least one event must occur on T. Getting a better enclosure at time $\overline{T}$ is significant because this enclosure serves as the initial value for the subsequent time interval.

There could be several limit states passed on as the initial state for the following segment, all sharing the same mode. Using MergeByMode, we unify such states so that the number of intermediate states does not grow beyond control.

Theorem 5.2 (*Soundness of One Step*)**.** *Consider the hybrid IVP $\mathcal{M} = (\mathcal{H}, T, (q_{\text{init}}, \boldsymbol{A}_{\text{init}}))$ and assume that the computation of $(\mathcal{Z}, \mathcal{S}) = \textbf{enclose-evolutions-step}(\mathcal{M})$ has been successful.*

1. *The function $\mathcal{Z}$ is an enclosure of solutions and restarted evolutions of $\mathcal{M}$.*
2. *$\mathcal{S}_\mathcal{E}(\overline{T}_\mathcal{E}) \subseteq \mathcal{S}$ for any evolution $\mathcal{E}$ that is enclosed by $\mathcal{Z}$ and satisfies $\overline{T}_\mathcal{E} = \overline{T}$.*

Proof. (1) We know that $t \mapsto \mathcal{Z}_V(t) \cup \left\{\left(\tau(\epsilon), \boldsymbol{\mu}(\epsilon)\right)\right\}$ is an enclosure of solutions and restarted evolutions of $\mathcal{M}$ from Theorem 4.14. As $\tau(\epsilon) = q_{\text{init}}$, we only replace $\boldsymbol{\mu}(\epsilon) = \text{Range}(\boldsymbol{Y}_\epsilon)$ with $\boldsymbol{Y}_\epsilon(t)$. According to Definition 3.8 and due to that $v' \neq \epsilon$ in Lemma 4.10, we only need to assure that $\mathcal{Z}$ still encloses the first segment of any solution after this change. This is a consequence of the soundness of **solve-ivp**.

(2) From (1) we know that $\mathcal{Z}\left(\overline{T}\right)$ has this property. The box state $\left(q_{\text{init}}, \boldsymbol{Y}_\epsilon\left(\overline{T}\right)\right)$, when enclosing limit states, is only relevant for event-free solutions defined over T. Recall that if **event-is-necessary**$(\mathcal{H}, \boldsymbol{Y}_\epsilon, q_{\text{init}})$ returns true, then every solution must contain at least one event according to Proposition 4.3, thus there are no event-free solutions. Therefore, in this situation, the state may be safely ignored.

Merging the states by MergeByMode clearly does not affect these properties. $\square$

So far, we have focused on enclosing hybrid automaton evolutions on a single fixed time interval T. We now formalize some aspects of a higher-level enclosure semantics in which **enclose-evolutions-step** is applied repeatedly on a contiguous sequence of time segments T^1, T^2, etc., partitioning the simulation interval $\mathcal{T}$. In this paper we do not specify any particular strategy determining the length of the individual segments, we assume that this is done by an unspecified procedure. We focus on the mechanism of applying **enclose-evolutions-step** on a given arbitrary partition of $\mathcal{T}$. One example is given in Definition 5.3.

Definition 5.3 (*Step by Step Solving*)**.** Let $T^1, T^2, \ldots, T^k$ be a subdivision of the time interval $\mathcal{T}$ where all of the intervals T^i are closed and non-singleton. For any hybrid IVP $(\mathcal{H}, \mathcal{T}, \mathrm{S}_{\text{init}})$ let

$$\textbf{enclose-evolutions-multi-step}(\mathcal{H}, \{T^i\}_{i=1\ldots k}, \mathrm{S}_{\text{init}}) = \mathcal{S}_0, \mathcal{Z}_1, \mathcal{S}_1, \ldots \mathcal{Z}_k, \mathcal{S}_k,$$

where

$$\begin{aligned}
&\mathcal{S}_0 = \left\{\mathrm{S}_{\text{init}}\right\}\\
&\mathcal{Z}_i = \bigcup\left\{\mathcal{Z}_{i,S} \;\middle|\; S \in \mathcal{S}_{i-1}\right\} && \text{for all } i = 1, \ldots, k\\
&\mathcal{S}_i = \text{MergeByMode}\left(\left\{\mathcal{S}_{i,S} \;\middle|\; S \in \mathcal{S}_{i-1}\right\}\right) && \text{for all } i = 1, \ldots, k
\end{aligned}$$

and

$$(\mathcal{Z}_{i,S}, \mathcal{S}_{i,S}) = \textbf{enclose-evolutions-step}(\mathcal{H}, T^i, S). \quad \square$$

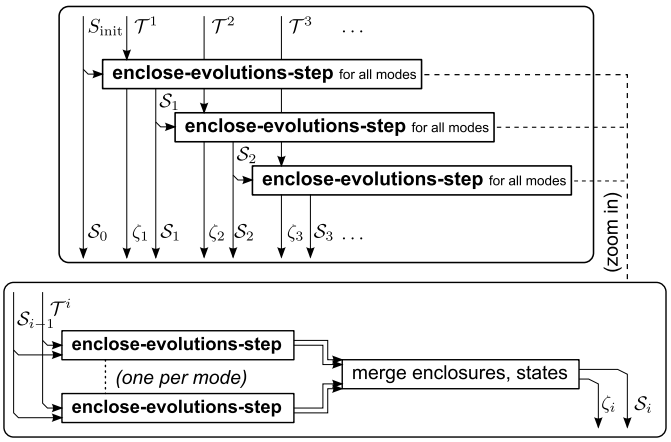


Fig. 9. Enclosing evolutions step by step.

Fig. 9 contains an informal data flow diagram illustrating the definition. Intuitively, **enclose-evolutions-multi-step** applies **enclose-evolutions-step** on each segment using every initial condition passed on from the enclosure computed for the previous segment. Due to the possible ambiguity of the system mode on the boundaries, there could be several box states passed on as the initial state for the following segment. Therefore **enclose-evolutions-step** is applied on each segment once for each possible initial mode. If there are multiple such modes, we end up with multiple enclosures for that segment. The final enclosure for the segment is the hull of all these enclosures. Similarly, the final states for that segment are computed by unifying the sets of box states returned by MergeByMode.

Theorem 5.4 (*Soundness of Step by Step Solving*)**.** *Assume that*

$$\textbf{enclose-evolutions-multi-step}(\mathcal{H}, \{T^i\}_{i=1\ldots k}, \mathrm{S}_{\text{init}}) = \mathcal{S}_0, \mathcal{Z}_1, \mathcal{S}_1, \ldots \mathcal{Z}_k, \mathcal{S}_k,$$

then

$$\mathcal{Z}(t) \stackrel{\text{def}}{=} \bigcup \left\{ \mathcal{Z}_i(t) \;\middle|\; t \in T^i \right\}$$

is an enclosure of solutions and restarted evolutions of the hybrid IVP $(\mathcal{H}, \mathcal{T}, \mathrm{S}_{\text{init}})$.

Theorem 5.4 is a direct consequence of Theorem 5.2 as when we stop processing an evolution $\mathcal{E}$ that potentially crosses the time barrier $\overline{T^i}$ by a continuous segment, then the remaining parts of the evolution will constitute a solution of the consequent hybrid IVP on T^{i+1}, thus $\mathcal{Z}_i$ will properly enclose it.

6. Additional examples

We have implemented the method presented in this paper and tested it on slightly more complicated (than in Fig. 3), but still basic non-linear systems exhibiting Zeno behavior [70]. The main purpose of these tests is to get an initial idea of how the method performs on some basic examples. Detailed analysis of what needs to be done to make this work in a realistic setting is a subject of future investigations.

The three examples are modifications of the classical bouncing ball (coupled with the output). First we consider Newtonian gravity, as seen in Fig. 10. Second, Fig. 11 gives a model of the bouncing ball when air resistance is present. Due to a limitation of the ODE solver, $|x'|$ is modeled with x' and $-x'$ in two separate modes (Rise and Fall). Finally, the result of combining both effects is shown in Fig. 12. In each case, the top pair of enclosures use steps T with $|T| \geq 2^{-12}$, while the bottom pair of enclosures use much larger steps ($|T| \geq 2^{-6}$) to make the effects of the approximation easier to see.

All event trees constructed during the six simulations were linear, because we only have one outgoing event from any state. For the enclosure in Fig. 10, all trees contained one or two nodes. For the enclosures in Figs. 11 and 12, a few trees contained up to 55 nodes, while most comprised only one or two nodes. In the case where 55 nodes were needed, the enclosure kept growing until the last step, at which it reached a fixed point. The enclosure kept growing because, for some reason, the approximation errors were larger than the contraction (of the system around the Zeno point). This behavior shows that the current algorithm is sensitive to the number of events in the minimal cycle in the automaton during Zeno behavior. We conjecture that this sensitivity can be reduced by techniques that over approximate the effect of multiple transitions at a time, as opposed to doing it one at a time as is currently the case.

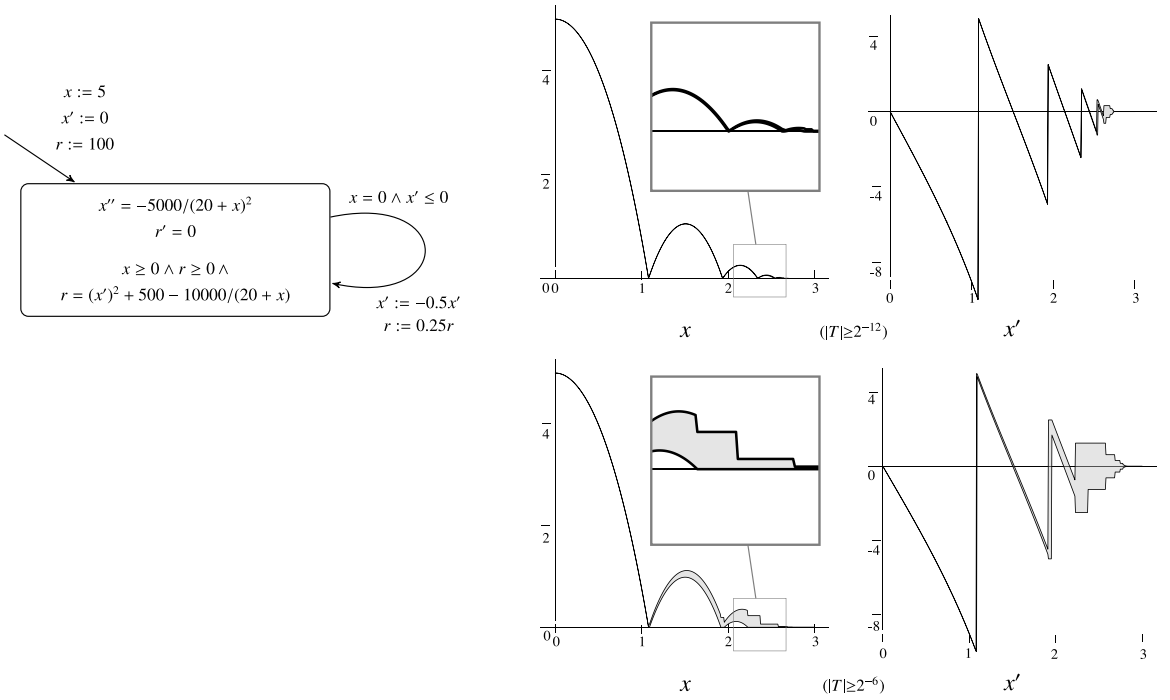


Fig. 10. Bouncing ball with Newtonian gravity and additional constraint. The Newtonian gravity is represented by the acceleration being inversely proportional to the square of the position. Enclosures at and beyond the Zeno point for x and x' are shown on the right.

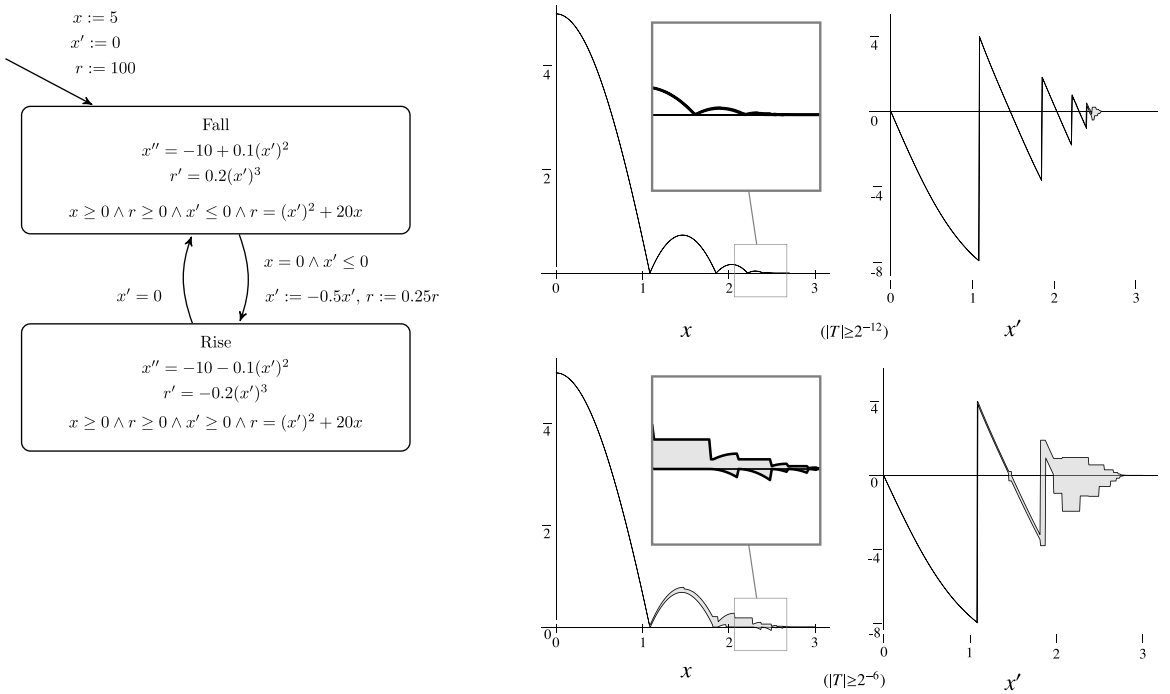


Fig. 11. Bouncing ball with air resistance and additional constraint. The air resistance is represented by the term $0.1(x')^2$ in the acceleration. Enclosures at and beyond the Zeno point for x and x' are shown on the right.

7. Conclusions

This paper presents an enclosure semantics that, akin to a simulator, is an algorithm that is able to enclose trajectories up to and beyond the Zeno point for a non-trivial class of problems (when the event tree construction terminates). We

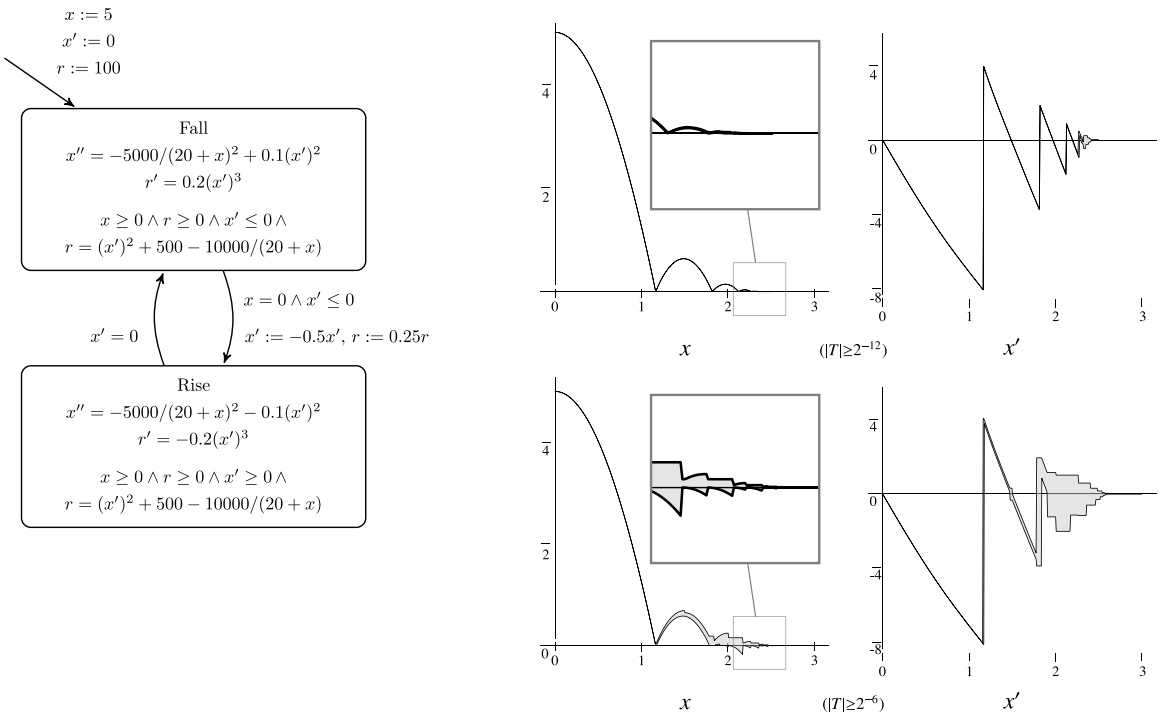


Fig. 12. Bouncing ball with Newtonian gravity, air resistance and additional constraint. Enclosures at and beyond the Zeno point for x and x' are shown on the right.

conjecture that Zeno trajectories converging towards a stable finite orbit are tractable. When this is not satisfied, *e.g.* for the bouncing ball on rising or dropping floor, the construction will not terminate. We expect that it will be possible to go beyond stable finite orbit problems by providing special handling for situations where certain inputs (such as the level of the floor) are not affected by events.

Currently, if the construction terminates and the post-Zeno state is modeled explicitly in the given hybrid automaton, the computed enclosure covers the corresponding trajectory as it starts from the Zeno point through a restarted evolution. On the other hand, if a post-Zeno state is not modeled explicitly, there is no generally accepted definition of the trajectories beyond Zeno. In this case, by Definition 3.8, the computed enclosure contains all trajectories starting close to the Zeno (state space) point. We believe that these enclosures suggest a sensible completion for Zeno systems given in hybrid automata formalism.

Natural completions exist for certain automata [7,18] given by unilateral constraints. Such *more-complete* hybrid systems have been used in practical examples, *e.g.* a bipedal robot walking with knee locking attains Zeno periodic orbits that relate to stable walking gaits [18,71]. Our future work includes investigating whether these completions are enclosed by our algorithm.

To achieve tight enclosures with the current method, we rely on the addition of supplementary constraints obtained by qualitative analysis (*e.g.* first integrals). The enrichment of the model with semantically redundant constraints in order to achieve accuracy is a standard technique used in reachability and verification procedures [72,73]. The need for such enrichments is partly due to the dependency and wrapping effects caused by box enclosure overestimations near events. In future work we plan to use more flexible enclosures, such as zonotopes, and expect that the modified algorithm will be substantially more applicable, *e.g.* directly for the bouncing ball without supplementary constraints.

We also plan a more detailed analysis of the performance characteristics of the implementation when simulating 3D rigid-body dynamics followed by similar analysis for even higher dimensional problems. In addition, we are interested in understanding the design space for simulator strategies and the performance impacts of the various parameters to the semantics.

Acknowledgments

The authors are grateful to the anonymous referees for their useful comments and suggestions that led to significant improvements in this paper.

This work was supported by the US National Science Foundation, awards NSF-CPS-1136099/1136104, the Swedish Knowledge Foundation (KK) and the Center for Research on Embedded Systems (CERES) grant number 20100314, and EPSRC grant number EP/C01037X/1.

Dedicated to the Memory of Paul Hudak, who introduced us to Hybrid Systems and inspired us to study their semantics and Zeno behavior.

References

[1] E. Darulova, V. Kuncak, Trustworthy numerical computation in scala, in: C.V. Lopes, K. Fisher (Eds.), OOPSLA, ACM, 2011, pp. 325–344.
[2] M.H. Escardó, PCF extended with real numbers, Theoret. Comput. Sci. 162 (1) (1996) 79–115.
[3] F.E. Cellier, E. Kofman, Continuous System Simulation, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[4] E. Hairer, S.P. Nørsett, G. Wanner, Solving Ordinary Differential Equations I (2Nd Revised. Ed.): Nonstiff Problems, Springer-Verlag New York, Inc., New York, NY, USA, 1993.
[5] E. Hairer, G. Wanner, Stiff and differential–algebraic problems, in: Solving Ordinary Differential Equations. II, second ed., in: Springer Series in Computational Mathematics, vol. 14, Springer-Verlag, Berlin, 1996.
[6] F. Zhang, M. Yeddanapudi, P. Mosterman, Zero-crossing location and detection algorithms for hybrid system simulation, in: IFAC World Congress, 2008, pp. 7967–7972.
[7] A. Ames, H. Zheng, R. Gregg, S. Sastry, Is there life after Zeno? Taking executions past the breaking (Zeno) point, in: 25th American Control Conference, 2006, Minneapolis, MN, 2006, pp. 2652–2657.
[8] K. Johansson, J. Lygeros, S. Sastry, M. Egerstedt, Simulation of zeno hybrid automata, in: Proceedings of the 38th IEEE Conference on Decision and Control, 1999. Vol. 4, 1999, pp. 3538–3543.
[9] J. Zhang, K.H. Johansson, J. Lygeros, S. Sastry, Zeno hybrid systems, Internat. J. Robust Nonlinear Control 11 (5) (2001) 435–451.
[10] M. Konečný, W. Taha, J. Duracz, A. Duracz, A. Ames, Enclosing the behavior of a hybrid system up to and beyond a zeno point, in: 2013 IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications, CPSNA, 2013, pp. 120–125.
[11] J. Moreau, Unilateral contact and dry friction in finite freedom dynamics, in: J. Moreau, P. Panagiotopoulos (Eds.), Nonsmooth Mechanics and Applications, in: International Centre for Mechanical Sciences, vol. 302, Springer Vienna, 1988, pp. 1–82.
[12] Manuel Duque Pereira Monteiro Marques, Differential Inclusions in Nonsmooth Mechanical Problems: Shocks and Dry Friction, Birkhäuser, Boston, MA, 1993.
[13] D.E. Stewart, A Dynamics With Inequalities: Impacts and Hard Constraints, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011.
[14] J.M. Schumacher, Time-scaling symmetry and Zeno solutions, Automatica J. IFAC 45 (5) (2009) 1237–1242.
[15] J. Shen, J.-S. Pang, Linear complementarity systems: Zeno states, SIAM J. Control Optim. 44 (3) (2005) 1040–1066. (electronic).
[16] L.Q. Thuan, M.K. Camlibel, Continuous piecewise affine dynamical systems do not exhibit Zeno behavior, IEEE Trans. Automat. Control 56 (8) (2011) 1932–1936.
[17] V. Acary, B. Brogliato, Numerical Methods for Nonsmooth Dynamical Systems: Applications in Mechanics and Electronics, Vol. 35, Springer Science & Business Media, 2008.
[18] A.D. Ames, Characterizing knee-bounce in bipedal robotic walking: A Zeno behavior approach, in: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control, HSCC'11, ACM, New York, NY, USA, 2011, pp. 163–172.
[19] Y. Or, A. Teel, Zeno stability of the set-valued bouncing ball, IEEE Trans. Automat. Control 56 (2) (2011) 447–452.
[20] A. Lamperski, A.D. Ames, On the existence of Zeno behavior in hybrid systems with non-isolated Zeno equilibria, in: 47th IEEE Conference on Decision and Control, 2008, IEEE, 2008, pp. 2776–2781.
[21] Y. Or, A. Ames, Stability and completion of Zeno equilibria in Lagrangian hybrid systems, IEEE Trans. Automat. Control 56 (6) (2011) 1322–1336.
[22] H. Zheng, E.A. Lee, A.D. Ames, Beyond zeno: Get on with it!, in: Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, 2006, pp. 568–582.
[23] L.P. Carloni, R. Passerone, A. Pinto, A.L. Angiovanni-Vincentelli, Languages and tools for hybrid systems design, Found. Trends Electron. Des. Autom. 1 (1–2) (2006) 1–193.
[24] Wolfram, SystemModeler, 2012. http://wolfram.com/system-modeler.
[25] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystrom, L. Saldamli, D. Broman, A. Sandholm, OpenModelica—A free open-source environment for system modeling, simulation, and teaching, in: Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006, pp. 1588–1595.
[26] R. Alur, R. Grosu, Y. Hur, V. Kumar, I. Lee, Modular specification of hybrid systems in CHARON, in: Hybrid Systems: Computation and Control, in: Lecture Notes in Computer Science, vol. 1790, Springer, 2000, pp. 6–19.
[27] Z. Wan, P. Hudak, Functional reactive programming from first principles, in: ACM SIGPLAN Conference on Programming Language Design and Implementation, 2000, pp. 242–252.
[28] H. Nilsson, A. Courtney, J. Peterson, Functional reactive programming, continued, in: Proceedings of the 2002 ACM SIGPLAN Workshop on Haskell, Haskell'02, ACM, New York, NY, USA, 2002, pp. 51–64.
[29] K. Makino, M. Berz, New applications of Taylor model methods, in: Automatic Differentiation of Algorithms: From Simulation to Optimization, Springer, 2002, pp. 359–364. (Chapter 43).
[30] N. Nedialkov, M. von Mohrenschildt, Rigorous simulation of hybrid dynamic systems with symbolic and interval methods, in: Proceedings of the 2002 American Control Conference, Vol. 1, 2002, pp. 140–147.
[31] A. Rauh, E. Hofer, E. Auer, ValEncIA-IVP: A comparison with other initial value problem solvers, in: Scientific Computing, Computer Arithmetic and Validated Numerics, 2006. SCAN 2006. 12th GAMM—IMACS International Symposium on, 2006, pp. 36–36.
[32] J. Lygeros, Lecture notes on hybrid systems, in: Notes for an ENSIETA Workshop, 2004.
[33] B. Brogliato, Nonsmooth Mechanics, Communications and Control Engineering, Springer Verlag, London, 1999.
[34] J.J. Moreau, Numerical aspects of the sweeping process, Comput. Methods Appl. Mech. Engrg. 177 (3) (1999) 329–349.
[35] V. Acary, F. Pérignon, Siconos: A software platform for modeling, simulation, analysis and control of nonsmooth dynamical systems, Simul. News Europe 17 (3–4) (2007) 19–26.
[36] V. Acary, B. Brogliato, D. Goeleven, Higher order Moreau's sweeping process: Mathematical formulation and numerical simulation, Math. Program. 113 (1) (2008) 133–217.
[37] A. Ames, A. Abate, S. Sastry, Sufficient conditions for the existence of Zeno behavior in a class of nonlinear hybrid systems via constant approximations, in: 46th IEEE Conference on Decision and Control, 2007, pp. 4033–4038.
[38] A. Lamperski, A. Ames, Lyapunov theory for Zeno stability, IEEE Trans. Automat. Control 58 (1) (2013) 100–112.
[39] R. Alur, D.L. Dill, A theory of timed automata, Theoret. Comput. Sci. 126 (2) (1994) 183–235.
[40] T.A. Henzinger, The theory of hybrid automata, in: LICS, IEEE Computer Society, 1996, pp. 278–292.
[41] A. Platzer, Differential dynamic logic for hybrid systems, J. Automat. Reason. 41 (2) (2008) 143–189.
[42] A. Platzer, J.-D. Quesel, KeYmaera: A hybrid theorem prover for hybrid systems, in: A. Armando, P. Baumgartner, G. Dowek (Eds.), IJCAR, in: LNCS, vol. 5195, Springer, 2008, pp. 171–178.
[43] D. Ishii, K. Ueda, H. Hosobe, An interval-based sat modulo ode solver for model checking nonlinear hybrid systems, Int. J. Softw. Tools Technol. Trans. 13 (5) (2011) 449–461.
[44] D. Ishii, Simulation and verification of hybrid systems based on interval analysis and constraint programming (Ph.D. thesis), Waseda University, 2010.
[45] N.S. Nedialkov, VNODE-LP—A validated solver for initial value problems in ordinary differential equations, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1, Technical Report CAS-06-06-NN, November 2006.
Retrieved from: http://www.cas.mcmaster.ca/~nedialk/vnodelp.

[46] X. Chen, E. Abrahám, S. Sankaranarayanan, Taylor model flowpipe construction for non-linear hybrid systems, in: Real-Time Systems Symposium (RTSS), 2012 IEEE 33rd, IEEE, 2012, pp. 183–192.
[47] 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: G. Gopalakrishnan, S. Qadeer (Eds.), Computer Aided Verification, in: Lecture Notes in Computer Science, vol. 6806, Springer, Berlin, Heidelberg, 2011, pp. 379–395.
[48] C.L. Guernic, A. Girard, Reachability analysis of linear systems using support functions, Nonlinear Anal. Hybrid Syst. 4 (2) (2010) 250–262.
[49] E. Lee, H. Zheng, Operational semantics of hybrid systems, in: M. Morari, L. Thiele (Eds.), Hybrid Systems: Computation and Control, in: Lecture Notes in Computer Science, vol. 3414, Springer, Berlin, Heidelberg, 2005, pp. 25–53.
[50] A. Edalat, D. Pattinson, Denotational semantics of hybrid automata, in: L. Aceto, A. Ingólfsdóttir (Eds.), Foundations of Software Science and Computation Structures, in: Lecture Notes in Computer Science, vol. 3921, Springer, Berlin, Heidelberg, 2006, pp. 231–245.
[51] O. Bouissou, M. Martel, A hybrid denotational semantics for hybrid systems, in: S. Drossopoulou (Ed.), Programming Languages and Systems, in: Lecture Notes in Computer Science, vol. 4960, Springer, Berlin, Heidelberg, 2008, pp. 63–77.
[52] P. Schrammel, B. Jeannet, From hybrid data-flow languages to hybrid automata: A complete translation, in: Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control, ACM, 2012, pp. 167–176.
[53] A. Benveniste, B. Caillaud, M. Pouzet, The fundamentals of hybrid systems modelers, in: 2010 49th IEEE Conference on Decision and Control (CDC), IEEE, 2010, pp. 4180–4185.
[54] S. Bliudze, S. Furic, An operational semantics for hybrid systems involving behavioral abstraction, in: Proceedings of the 10th International Modelica Conference, Linköping Electronic Conference Proceedings, Linköping University Electronic Press, Linköping, 2014, pp. 693–706.
[55] S. Bliudze, D. Krob, Modelling of complex systems: Systems as dataflow machines, Fund. Inform. 91 (2) (2009) 251–274.
[56] K. Suenaga, H. Sekine, I. Hasuo, Hyperstream processing systems: nonstandard modeling of continuous-time signals, in: ACM SIGPLAN Notices, Vol. 48, ACM, 2013, pp. 417–430.
[57] R.E. Moore, Interval Analysis, Vol. 4, Prentice-Hall, Englewood Cliffs, 1966.
[58] L. Jaulin, Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics, Vol. 1, Springer Science & Business Media, 2001.
[59] R.E. Moore, R.B. Kearfott, M.J. Cloud, Introduction to Interval Analysis, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.
[60] W. Tucker, Validated Numerics, A Short Introduction to Rigorous Computations, Princeton University Press, Princeton, NJ, 2011.
[61] A. Ames, S. Sastry, Characterization of Zeno behavior in hybrid systems using homological methods, in: Proceedings of the American Control Conference, Vol. 2, 2005, pp. 1160–1165.
[62] G. Frehse, C. Le Guernic, A. Donze, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, SpaceEx: scalable verification of hybrid systems, in: Computer Aided Verification, in: Lecture Notes in Comput. Sci., vol. 6806, Springer, Heidelberg, 2011, pp. 379–395.
[63] M. Althoff, B.H. Krogh, Avoiding geometric intersection operations in reachability analysis of hybrid systems, in: Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control, HSCC'12, ACM, New York, NY, USA, 2012, pp. 45–54.
[64] X. Chen, E. Abraham, S. Sankaranarayanan, Taylor model flowpipe construction for non-linear hybrid systems, in: Proceedings of the 2012 IEEE 33rd Real-Time Systems Symposium, RTSS'12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 183–192.
[65] N. Ramdani, N.S. Nedialkov, Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques, Nonlinear Anal. Hybrid Syst. 5 (2) (2011) 149–162.
[66] N. Ramdani, L. Trave-Massuyes, A fast method for solving guard set intersection in nonlinear hybrid reachability, Decision and Control, CDC, IEEE, 2013, pp. 508–513.
[67] K. Makino, M. Berz, Cosy infinity version 9, Nucl. Instrum. Methods Phys. Res. A 558 (1) (2006) 346–350.
[68] A. Edalat, D. Pattinson, A domain-theoretic account of Picard's theorem, LMS J. Comput. Math. 10 (2007) 83–118.
[69] M. Konečný, J. Duracz, A. Farjudian, W. Taha, Picard method for enclosing ODEs with uncertain initial values, in: 11th International Conference on Computability and Complexity in Analysis, Darmstadt, Germany, 2014, pp. 41–42.
[70] M. Konečný, A Library for Approximating Exact Real Numbers and Functions (AERN), A branch dedicated to experiments reported in this paper, https://github.com/michalkonecny/aern/tree/nahs2015-plots.
[71] Y. Or, A.D. Ames, Stability and completion of zeno equilibria in Lagrangian hybrid systems, IEEE Trans. Automat. Control 56 (2011) 1322–1336.
[72] E. Asarin, T. Dang, O. Maler, R. Testylier, Using redundant constraints for refinement, in: Automated Technology for Verification and Analysis, Springer, 2010, pp. 37–51.
[73] B. Carlson, V. Gupta, Hybrid CC and interval constraints, in: T.A. Henzinger, S. Sastry (Eds.), Hybrid Systems 98: Computation and Control, in: Lecture notes in computer science, vol. 1386, Springer Verlag, 1998, pp. 80–94.