Enclosing the Behavior of a Hybrid System up to and Beyond a Zeno Point

Michal Konečný*, Walid Taha†‡, Jan Duracz†, Adam Duracz† and Aaron Ames§
*School of Engineering and Applied Science, Aston University, Birmingham, UK, Email: M.Konecny@aston.ac.uk
†Halmstad University, Halmstad, Sweden, Email: Walid.Taha@hh.se, Jan.Duracz@hh.se, Adam.Duracz@hh.se
‡Computer Science Department, Rice University, Texas, USA
§Department of Electrical & Computer Engineering, Texas A&M University, USA, Email: aames@tamu.edu

***Abstract*—Even simple hybrid systems like the classic bouncing ball can exhibit Zeno behaviors. The existence of this type of behavior has so far forced 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 modeling. 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 independently of the occurrence of a given event. Such an event can then even occur an unbounded number of times, thus making it possible to handle certain types of Zeno behavior.**

I. Introduction

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

1) Number representation and implementation of arithmetic [1]. Real numbers are infinite values, 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 are 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 have standard, closed-form representations. This is not the case, in general, for non-linear differential equations, which arise naturally in a wide range of domains, especially in three-dimensional space. It is useful to note that this difficulty can be seen as a strict generalization of the first one.
3) Event detection and localization [4]. Event detection is also only semi-decidable, and localization is, in general, not computable.
4) Zeno behavior [5], [6], [4], [7]. 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 systems and, as we will review next, for simulation tools.

Of these four difficulties, Zeno behavior may have received the least attention. It was first studied in the context of hybrid systems over a decade ago [6], where it was observed to be an interesting pathology of hybrid systems. This seemingly singular behavior turns out to be an essential point of divergence between traditional dynamical systems and hybrid systems—its existence can result in 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.

This phenomenon arises naturally when modeling physical systems. It can occur, for example, in rigid body dynamics with impact constraints (such as those modeling bipedal robots with mechanical knee stops [8]). This can be illustrated with a simple bouncing ball modeled as follows: the position, velocity and acceleration of the ball are denoted as functions of t by $x(t)$ and $x'(t)$ and $x''(t)$, respectively. Fix the initial condition to be $x(0) = 10$ 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 let $x'(t^+) = -x'(t^-)/2$, 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 series. The time which this series converges to is the Zeno time, and the dynamics governing the system at this time is termed Zeno point [9], [10].

A. Problem

Simulation tools do not fare well when we try to use them to simulate the bouncing ball model presented above. For example, we consider Simulink (v7.9) [11], SystemModeler (v3.0.0) [12], OpenModelica (v1.9.0 beta 4) [13], Charon (v1.0) [14], and the FRP [15] implementation Yampa (v0.9.3) [16]. With Simulink, it is not clear how to express the equality condition directly. For the rest of the tools, none detect the condition $x(t) = 0$ that triggers the bounce. For most of the tools, this means that the ball falls through the floor. In other words, a behavior that is not admitted by the model is produced by the simulator. SystemModeler is the only tool that gives a warning to the user that the model tests equality on real numbers and reports that it considers this problematic. Running this example aborts with an error.

The problem can be made a bit easier for most tools by replacing 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 produces results past the Zeno point, because it does not attempt to detect all event occurrences. SystemModeler (and, on one formulation, Simulink) 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 [4] 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, the use of such heuristics does not come with any guarantees about resulting behavior.

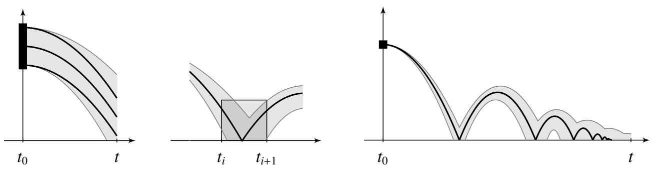
Can we go past a Zeno point *and* produce a correct result?

B. Related Work

There has been a concerted effort to detect and handle Zeno behavior a priori through analytic methods, with the end result being conditions on the existence of Zeno behavior [17], [9], [18]. In addition, with the understanding that the solutions should be extendable beyond the Zeno point to allow for valid post-Zeno behavior, a method for completing hybrid systems was purposed in the context of mechanical systems undergoing impacts [5], [19]. But the *correctness* of these methods relies on manual (human) analysis of individual models rather than a mechanized simulation method.

Most hybrid system verification tools do not handle Zeno systems. Many tools such as KeYMaera [20], [21] and Hydlogic [22], [23] assume that the modeled system is free of Zeno behavior. Others such as the reachability analysis tool Flow* [24] require that an explicit upper bound is provided on the number of transitions taken into account, effectively excluding Zeno systems. One reachability analysis tool, SpaceEx [25], computes an enclosure of reachable states using the LeGuernic-Girard (LGG) algorithm [26]. This computation terminates on some Zeno systems, but communications with the authors indicate that it is not clear whether or not these enclosures are valid past the Zeno point.

Formulations that are closer to traditional programming semantics, and for which software implementations exist, include the denotational semantics for Functional Reactive Programming (FRP) [15] and the operational semantics given to HyVisual [27]. The semantics of 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. This means that, by increasing the sampling rate, one can get arbitrarily close to the idealized answer. However, 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 of 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. As languages, neither FRP nor HyVisual provide any special support for handling Zeno behavior. On basic examples, both produce behavior



(a) Enclosing continuous behaviors (b) Event detection and handling (c) Enclosing sequences of events

Fig. 1: Basic simulator functions. This paper focuses on (b) and (c).

similar to what most of the tools described above produce. Implementations of both systems use standard floating-point representations for numbers.

Finally, there are methods based on Moreau's sweeping process [28] for simulating mechanical systems with impacts, that do handle some Zeno systems (see for example [29]). However, we are not aware of methods for computing enclosures in this domain.

C. Contributions

A simulator that can handle Zeno behaviors can be seen as consisting of three basic simulator functions: enclosing continuous behaviors (Figure 1a), event detection and handling (Figure 1b), and enclosing sequences of events (Figure 1c). The proposed method is independent of the first part. So, we assume that an enclosure-based ODE IVP solver (*cf.* [30], [31], [32]) is provided to compute continuous parts of system behavior (Figure 1a).

After introducing a formal notion of hybrid systems and their evolution (Section II), we present a semantics for event detection and localization (Section III). This semantics is algorithmic, and can therefore be used directly for simulation. Focusing our attention on enclosures suggests an elegant way for dealing with events (Definition III.8). It also suggests a natural improvement that allows us to both ensure soundness and avoid having to detect some inconsequential events. This enables enclosing Zeno behaviors. We show that this semantics, when defined, produces enclosures that provide upper and lower bounds on functions that satisfy the model being simulated (Theorem III.9).

There are two distinct challenges in simulating Zeno systems. The first challenge 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 challenge is to compute a tight enclosure. The proposed algorithm deals with the first challenge; the second challenge is addressed by refining the hybrid system models to ensure that they contain enough information to produce a tight enclosure. This is achieved by introducing additional semantically redundant constraints to the model. Figure 2 illustrates how the new method overcomes these two challenges on two classic examples of hybrid systems.

ODE IVP solvers can choose different representations for enclosures. The event handling algorithm presented in the paper places a minimal set of requirements on the enclosures produced by the ODE IVP solver. Similarly, different solver

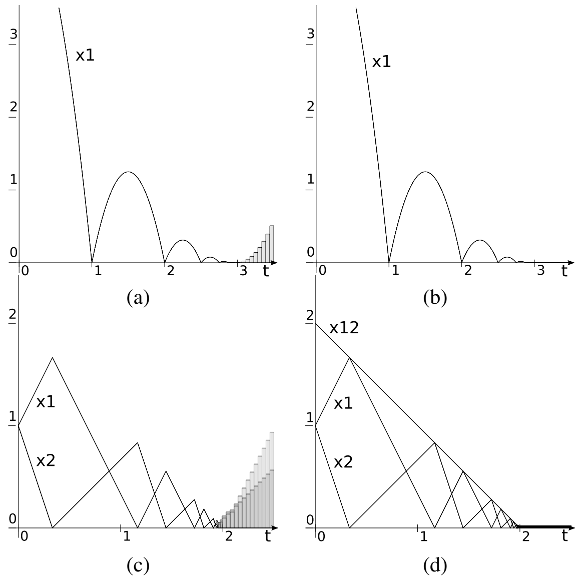


Fig. 2: Enclosures for systems with Zeno behavior. Figures (a) and (c) show the results for the bouncing ball and water tank systems of Lygeros [33, Figures 3.1 and 3.7]. However, we make these models more challenging to simulate by replacing some inequalities with equalities (such as the bouncing conditions). Figures (b) and (d) show the results for the examples when additional (redundant) constraints are added to achieve more precise enclosures.

strategies can be used to compose the basic simulator functions. The main task of such a strategy is to partition time into steps and apply the basic simulator functions over the segments. The strategy used to produce Figure 2 is adaptive, refining the partition nearer events, leading to an accurate event handling.

II. Hybrid Enclosures

This section defines a class of hybrid systems, their evolutions and enclosures of all potential evolutions from an interval initial state. Our hybrid system notation broadly follows [9], [18]. We first introduce notation related to intervals and interval functions.

Let $\mathbb{I}$ denote the set of closed real intervals. We use the notation $\underline{A}$ for the *left endpoint* and $\overline{A}$ for the *right endpoint* of $A \in \mathbb{I}$, and then write $A = [\underline{A}, \overline{A}]$. We use boldface letters to denote vectors, *e.g.* $\boldsymbol{A} \in \mathbb{I}^k$. We identify such vectors with the Cartesian product of the component intervals, called *boxes*. Thus we also have $\boldsymbol{A} \subseteq \mathbb{R}^k$. Let $\text{Hull}\{\boldsymbol{A}_i\}_{i\in I}$ be the smallest box containing all $\boldsymbol{A}_i$.

To represent enclosures of hybrid system trajectories, we will use *interval functions*, i. e. functions from time to vectors of intervals, denoted $\boldsymbol{X}, \boldsymbol{Y} \colon T \to \mathbb{I}^k$. The primary source of interval functions are the enclosures produced by the ODE IVP solver that parametrizes our semantics, as described in Section I-C. These enclosures are sometimes further combined using $\text{Hull}\{\boldsymbol{Y}_i\}_{i\in I}$ — the box hull taken pointwise over the shared domain of the functions $\boldsymbol{Y}_i$.

Definition II.1. *A* hybrid system *is a tuple*

$$\mathcal{H} = (Q,\ E,\ \sigma,\ \tau,\ \{\boldsymbol{D}_q\}_{q\in Q},\ \{\boldsymbol{f}_q\}_{q\in Q},\ \{C_e\}_{e\in E},\ \{\boldsymbol{r}_e\}_{e\in E})$$

consisting of:

- Q *— a finite set of* modes q
- E *— a finite set of* event types e
- $\sigma(e), \tau(e) \in Q$ *— the* source *and* target *of event e, respectively*
- $\boldsymbol{D}_q \subseteq \mathbb{R}^n$ *— the* domain *of mode q*
- $\boldsymbol{f}_q \colon \boldsymbol{D}_q \to \mathbb{R}^n$ *— the* vector field *of mode q*
- $\boldsymbol{C}_e \subseteq \mathbb{R}^n$ *— the* guard *of event e*
- $\boldsymbol{r}_e \colon \mathbb{R}^n \to \mathbb{R}^n$ *— the* reset map *of event e.*

In practice we use encodings of hybrid systems in which the vector fields $\boldsymbol{f}_q$ are given as expressions understood by the chosen ODE solver and the sets $\boldsymbol{D}_q$ and $\boldsymbol{C}_e$ are given as supports of predicates in a language that allows us to compute approximate intersections of interval vectors with these sets.

Definition II.2. *Let the hybrid system $\mathcal{H}$ be given as in Definition II.1. An* evolution *of $\mathcal{H}$ over time $T \in \mathbb{I}$ is a sequence*

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

either infinite or ending with $(T_k, q_k, \boldsymbol{x}_k)$ for some $k \in \mathbb{N}$, such that:

- *$T = \bigcup_i T_i$ with $\overline{T_i} = \underline{T_{i+1}}$ for all permissible i. We denote $t_{i-1} = \underline{T_i}$.*
- *For each i, $\boldsymbol{x}_i : T_i \to \mathbb{R}^n$ satisfies $\boldsymbol{x}_i' = \boldsymbol{f}_{q_i}(\boldsymbol{x}_i)$ and $\boldsymbol{f}_{q_i}(\boldsymbol{x}_i) \in \boldsymbol{D}_{q_i}$ on the interior of T_i.*
- *$q_i = \sigma(e_i)$, $q_{i+1} = \tau(e_i)$ for each event occurrence e_i.*
- *$\boldsymbol{x}_i(t_i) \in \boldsymbol{C}_{e_i}$ and $\boldsymbol{x}_{i+1}(t_i) = \boldsymbol{r}_{e_i}(\boldsymbol{x}_i(t_i))$.*

Definition II.3. *A* state *of hybrid system $\mathcal{H}$ is a pair $s \in Q\times\mathbb{R}^n$.*

Definition II.4 (Evolution trajectory)**.** *For an evolution ζ over T, using notation as in Definition II.2, the trajectory of ζ is defined as follows:*

$$\text{traj}(\zeta)(t) = \{\boldsymbol{x}_i(t) \mid \textit{for all } i \textit{ such that } t \in T_i\}$$

Definition II.5. *An* interval state *of $\mathcal{H}$ is a pair $S \in Q \times \mathbb{I}^n$.*

Definition II.6. *A* hybrid IVP *is a tuple $(\mathcal{H}, T, S)$ comprising $\mathcal{H}$, a hybrid system, $T \in \mathbb{I}$ and an interval state $S = (q_{\text{init}}, \boldsymbol{A})$. Its* solution *is an evolution ζ_{s} on T with $q_1 = q_{\text{init}}$, $\boldsymbol{x}_1(\underline{T}) \in \boldsymbol{A}$.*

In this paper we are concerned with computing an enclosure of all solutions of a hybrid IVP in the following sense:

Definition II.7. *An* enclosure *of a solution ζ_{s} over T of a hybrid IVP $(\mathcal{H}, T, S)$ with $S = (q_{\text{init}}, \boldsymbol{A})$ is an interval function $\boldsymbol{X} \colon T \to \mathbb{I}^n$ with* $\text{traj}(\zeta_{\text{s}})(t) \subseteq \boldsymbol{X}(t)$ *for all $t \in T$.*

III. Lazy Event Detection and Handling

In this section we begin by defining a basic operation for the detection of individual events, an important sub-task of event processing (Subsection III-A). We then proceed to present the core of our method, which is an algorithm for enclosing all possible sequences of events in a fixed time interval T without further localization (Subsection III-B). The details of the proof are omitted for space reasons.

A. Detecting the Next Event

We define an algorithm **detect-next-event** which will provide the main mechanism to gain information about potential event sequences that could occur in the given time interval being considered. It is invoked after having established the potential occurrence of an event sequence $v = e_1 \dots e_\ell$, and with the aim of determining what the next event $e_{\ell+1}$ within T could be. Its parameters do not explicitly include the sequence of previously detected events. Instead, they include an interval function $\boldsymbol{Y}$ that encloses any continuous evolution that follows the event sequence v. Thus the enclosure $\boldsymbol{Y}$ is assumed to be valid on a subset of T that starts some unspecified time $t_\ell \in T$, and that ends either at the time of the next event or, if no further event occurs, at $\overline{T}$.

Definition III.1. *For any $\boldsymbol{Y}\colon T \to \mathbb{I}^n$, $q \in Q$, let*

$$\textbf{detect-next-event}(\mathcal{H}, T, q, \boldsymbol{Y}) = \begin{cases} \mathsf{CertainlyOneOf}(E') & \text{if } \boldsymbol{Y}(\overline{T}) \cap \boldsymbol{D}_q = \emptyset \\ \mathsf{MaybeOneOf}(E') & \text{otherwise} \end{cases}$$

where CertainlyOneOf *and* MaybeOneOf *are symbolic constants and $E' = \{e \in E \mid \sigma(e) = q,\ \mathrm{Range}(\boldsymbol{Y}) \cap C_e \neq \emptyset\}$.*

The result of **detect-next-event** encodes some information about the potential event $e_{\ell+1}$, including distinguishing between the three alternatives that are specified in the following proposition.

Proposition III.2. *Assume an evolution ζ of $\mathcal{H}$ on T with at least ℓ events and an enclosure $\boldsymbol{Y}$ on T, with $\boldsymbol{x}_{\ell+1}(t) \subseteq \boldsymbol{Y}(t)$ for all $t \in T_{\ell+1}$, and let $R =$ **detect-next-event**$(\mathcal{H}, T, q_{\ell+1}, \boldsymbol{Y})$. Then*

(a) $R = \mathsf{CertainlyOneOf}(E')$ implies that ζ has at least $\ell + 1$ events and $e_{\ell+1} \in E'$.

(b) $R = \mathsf{MaybeOneOf}(\emptyset)$ implies that ζ has precisely ℓ events.

(c) $R = \mathsf{MaybeOneOf}(E')$ implies that if ζ has at least $\ell + 1$ events then $e_{\ell+1} \in E'$.

Establishing (a) ensures that the interval $\boldsymbol{Y}(\overline{T})$ is irrelevant for the solution enclosure at point $\overline{T}$. Establishing (b) means that one does not have to consider any further events. (c) is a fallback "no information" result, except for restricting the type of the potential event. These properties establish the safety of step (2) in Definition III.7.

Since all the checks used to make the decision and compute E' are inclusion isotone, **detect-next-event** is also inclusion isotone in the following sense:

Proposition III.3 (Inclusion Isotonicity of Event Detection)**.**

(1) **detect-next-event**$(\mathcal{H}, T, q, \boldsymbol{Y}_1) = \mathsf{CertainlyOneOf}(E'_1)$ *and $\boldsymbol{Y}_1 \supseteq \boldsymbol{Y}_2$ implies that* **detect-next-event**$(\mathcal{H}, T, q, \boldsymbol{Y}_2) = \mathsf{CertainlyOneOf}(E'_2)$ *for some $E'_2 \subseteq E'_1$.*

(2) **detect-next-event**$(\mathcal{H}, T, q, \boldsymbol{Y}_1) = \mathsf{MaybeOneOf}(E'_1)$ *and $\boldsymbol{Y}_1 \supseteq \boldsymbol{Y}_2$ implies that* **detect-next-event**$(\mathcal{H}, T, q, \boldsymbol{Y}_2)$ *returns some $E'_2 \subseteq E'_1$.*

Intuitively, Proposition III.3 states that improving the knowledge of the set of solutions can only preserve or improve the knowledge of the set of event occurrences extracted by **detect-next-event**. This property establishes the safety of step (1) in Definition III.7.

B. Enclosing Sequences of Events

We define an algorithm **enclose-events** which, given a hybrid IVP, identifies a set that includes all possible sequences of events on T. This is an over-approximation, so it will often include sequences strictly not possible. Nevertheless, we can use it to build an algorithm to produce an enclosure of all solutions of the hybrid IVP over T, as well as at time $\overline{T}$ (Definition III.8).

An essential feature of **enclose-events** is that it does not determine the timing of the events beyond knowing that they happen in T. The lack of event localization only leads to a loss of precision of magnitude comparable to the size of T. Since we place no requirements on the strategy that determines the time T, the strategy has the freedom to ensure that T is systematically reduced until the precision of the outgoing trajectory is acceptable.

To account for the general case where we cannot determine the order of different types of events, we consider a set of event sequences instead of just one sequence. The set of sequences is prefix-closed. In other words, it is equivalent to a tree whose branching is determined by possible "next events". We use the following notation for sequences:

Notation III.4. *Let E^* denote the set of finite sequences of elements of E where the empty sequence is denoted ϵ and vw denotes the concatenation of sequences $v, w \in E^*$.*

Definition III.5 (Event tree)**.** *Let $\mathbb{B}$ denote the set of Boolean values. An* event tree *with initial mode q is a tuple (V, μ) where $V \subseteq E^*$ and $\mu : V \to (T \to \mathbb{I}^n) \times \mathbb{B}$ assigns to each sequence $v \in V$ the following:*

$\boldsymbol{Y}(v)$ — *"a state enclosure following the events v"*

$\mathrm{MayBeLast}(v)$ — *whether $\boldsymbol{Y}(v)$ could apply up to time $\overline{T}$*

Let us set the following notation:

$\sigma(\epsilon) = q$ *and* $\sigma(ev) = \sigma(e)$ *for each $e \in E, v \in E^*$*

$\tau(\epsilon) = q$ *and* $\tau(ve) = \tau(e)$ *for each $e \in E, v \in E^*$*

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

$v \in V$, i. e. V is prefix-closed

$\sigma(w) = \tau(v)$

The following interpretation of a tree will be used to determine a valid enclosure of the state variables at time $\overline{T}$:

Definition III.6. *The* end-time states *of an event tree (V, μ) is the following set of interval states:*

$$(V, \mu)(\overline{T}) \overset{\text{def}}{=} \mathrm{M}\left\{\left(\tau(v), \boldsymbol{Y}(v)(\overline{T})\right) \middle| \, v \in V, \mathrm{MayBeLast}(v)\right\}$$

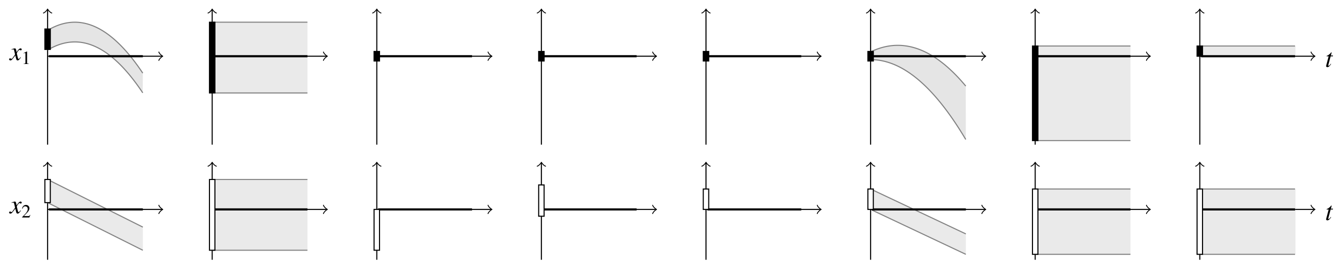
where

$$\mathrm{M}\,\{(q_i, \boldsymbol{B}_i)\}_{i \in I} \overset{\text{def}}{=} \left\{(q, \boldsymbol{B}_q) \,\middle|\, q \in \{q_i\}_{i \in I}, \boldsymbol{B}_q = \mathrm{Hull}\,\{\boldsymbol{B}_i \mid q_i = q\}\right\}$$

In Definition III.8 we also extract an enclosure over the whole of T. Note that over T we must drop the condition $\mathrm{MayBeLast}(v) = \text{true}$ and thus lose even more information.

In what follows we assume that **solve-ivp** is any given enclosure ODE IVP solver, such as [32]. In our experiments we used a solver based on the interval Picard operator [34].

The essence of the algorithm that we will present for event tree construction is in computing an enclosure for all behavior



(a) event detected (b) simplify (c) $\cap$ w. guards (d) reset (e) $\cap$ w. invariant (f) after event (g) time blur (h) $\cap$ w. invariant

Fig. 3: An event is irrelevant when (h) is included in (b).

stemming from the occurrence of a particular event, starting from a given enclosure. To understand how this is done, it is instructive to first focus on a case when there is only one type of event and, therefore, no branching. The extension to the branching case does not involve any surprises. Figure 3 illustrates the basic process of determining that an event is irrelevant for an instance of the bouncing ball system. First, **solve-ivp** produces an enclosure (a) that is valid on assumption of no events. Then **detect-next-event** returns CertainlyOneOf({bounce}) for this enclosure. An approximation of the range of the enclosure (b) is intersected with the guard for the bounce event (c) and the reset for the bounce event is applied (d). The resulting interval vector is intersected with the domain invariant for the target mode of the bounce event (e) yielding an initial condition for the system's evolution after the bounce in the target mode (f). As the event could occur at any point in the time interval, the range of the obtained enclosure (g) needs to be taken to obtain a safe approximation for the post-event behavior. This approximation is intersected with the domain invariant for the current mode, resulting in a tighter enclosure (h) for V(bounce). Establishing that one occurrence of an event is irrelevant to the soundness of the enclosure means that whether an event occurs zero or more times is also irrelevant to the soundness of the enclosure. This basic fixed-point property is key to being able to simulate Zeno behaviors.

Definition III.7 (Event tree construction algorithm**).** *For an interval initial state $S = (q, \boldsymbol{A})$ the event tree (V_S, μ_S) is constructed as the limit of the sequence of trees obtained by iteratively applying the tree-enlarging operator* **add-layer** *given below, starting with the initial tree (V_0, μ_0) with initial mode q defined as follows:*

$$V_0 = \{\epsilon\}$$
$$\boldsymbol{Y}(\epsilon) = \textbf{solve-ivp}(\boldsymbol{F}_q, T, \boldsymbol{A})$$
$$\text{MayBeLast}(\epsilon) = \text{false } \textit{(provisionally)}$$

The tree-enlarging operator **add-layer** *takes an event tree (V, μ) and, for each sequence $v \in V$ that is not a prefix of another sequence in V, does the following, in order to try and add further sequences to the tree and adjust* MayBeLast(v)*:*

(1) Check whether there is a prefix w of v such that $\tau(v) = \tau(w)$ and $\boldsymbol{Y}(v) \subseteq \boldsymbol{Y}(w)$. If so, stop processing for this v.

(2) If step (1) failed, apply **detect-next-event**$(\mathcal{H}, T, \tau(v), \boldsymbol{Y}(v))$ *and based on the computed set E' and the inferred decision (a)–(c) in the sense of Proposition III.2, then:*

(a) For each $e \in E'$ compute (as illustrated in Figure 3 where (a) is $\boldsymbol{Y}(v)$):

$$\boldsymbol{A} = \text{Hull}\Big(D_{\tau(e)} \cap \boldsymbol{r}_e\Big(\text{Range}(\boldsymbol{Y}(v)) \cap \boldsymbol{C}_e\Big)\Big)$$
(steps (b)–(e) in Figure 3)
$$\boldsymbol{N} = \text{Range}(\textbf{solve-ivp}(\boldsymbol{F}_{\tau(e)}, T, \boldsymbol{A}))$$
(steps (f)–(g) in Figure 3)

and if the sets $\boldsymbol{A}$ *and* $\boldsymbol{N}$ *are non-empty, add ve to V and set:*

$$\text{MayBeLast}(ve) = \text{false } \textit{(provisionally)}$$
$$\boldsymbol{Y}(ve)(t) = \text{Hull}(\boldsymbol{N} \cap \boldsymbol{D}_{\tau(e)}) \textit{ (step (h) in Figure 3)}$$

(b) Set MayBeLast(v) *to* true*.*

(c) Do both (a) and (b) above.

Step (1) makes it possible for the algorithm to terminate even when there may be an infinite number of events occurring in the given time. Note that, at any point in the algorithm, whenever we are considering a particular sequence of events, we have already computed an enclosure for all possible prefixes of that sequence. This step checks whether there is any prefix of the current event sequence for which a) the next mode is the same, and b) the enclosure for that prefix includes the enclosure for the current sequence. Whenever this is the case it means we have reached a fixed point in the analysis; the events that extend that prefix to the current sequence can occur zero or more times and it does not affect the validity of the enclosure that was computed for the prefix. This insight is what allows us to compute 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.

Also note that in (V_S, μ_S) all enclosures $\boldsymbol{Y}(v)$ are *constant* interval functions except the root enclosure $\boldsymbol{Y}(\epsilon)$. Simplifying (or over-approximating) to a constant enclosure is necessary at some stage in order to be able to deal with initial time uncertainty. Our choice of when to perform the simplification is motivated by the need to be able to easily intersect the enclosures with the relevant mode domain $\boldsymbol{D}_q$ and with the guard $\boldsymbol{C}_e$.

Definition III.8. *For any hybrid IVP $(\mathcal{H}, T, S)$, let*

$$\textbf{enclose-events}(\mathcal{H}, T, S) \overset{\text{def}}{=} ((V_S, \mu_S)(\overline{T}), \boldsymbol{Y})$$

where $\boldsymbol{Y} = \text{Hull}\{\boldsymbol{Y}(v) \mid v \in V_S\}$.

When implementing this function, we add another parameter K and abort the algorithm when an event tree exceeds size K to prevent non-termination.

Note that $\boldsymbol{B} = \text{Hull}\{\boldsymbol{B}_q\}$ (where $\boldsymbol{B}_q$ are the boxes calculated by enclose-events as specified in Definitions III.8 and III.6) is often a proper sub-interval of $\boldsymbol{Y}(\overline{T})$ because the definition of $\boldsymbol{B}$ excludes the enclosure of any event sequence for which it was established that at least one further event must occur before time $\overline{T}$. Getting a better enclosure at time $\overline{T}$ is significant because this enclosure serves as the initial value for the subsequent time interval.

Theorem III.9 (Main theorem)**.** *For any hybrid IVP* $(\mathcal{H}, T, S)$*, the enclosure* $(\{(q, \boldsymbol{B}_q)\}, \boldsymbol{Y})$ *computed by* **enclose-events** *will enclose every solution* ζ_s *of* $(\mathcal{H}, T, S)$ *in the sense that* $\forall t \in T.\, \text{traj}(\zeta_s)(t) \subseteq \boldsymbol{Y}(t)$ *and* $\text{traj}(\zeta_s)(\overline{T}) \subseteq \text{Hull}\{\boldsymbol{B}_q\}$.

Thus our semantics is sound in the sense that the enclosures contain trajectories that start in the given initial interval state.

IV. Conclusions

This paper showed how to use enclosures to simulate hybrid systems in such a way that certain types of Zeno systems can be accommodated. In contrast to previous proposals for dealing with Zeno behavior, the new method avoids the need for extending individual models specifically to enable a transition to a post-Zeno state. The method does require refining the model by adding additional constraints, but only to improve the quality of enclosures.

In future work, we plan a more detailed analysis of the performance characteristics of the implementation when simulating three dimensional rigid-body dynamics problems. We are interested in understanding the design space for solver strategies and the performance impacts of the various parameters to the semantics.

Acknowledgements

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

References

[1] E. Darulova and V. Kuncak, "Trustworthy numerical computation in scala," in *OOPSLA*, C. V. Lopes and K. Fisher, Eds. ACM, 2011.

[2] M. H. Escardó, "PCF extended with real numbers," *Theor. Comput. Sci.*, vol. 162, no. 1, pp. 79–115, 1996.

[3] F. E. Cellier and E. Kofman, *Continuous system simulation*. Springer-Verlag, 2006.

[4] F. Zhang, M. Yeddanapudi, and P. Mosterman, "Zero-crossing location and detection algorithms for hybrid system simulation," *IFAC World Congress*, 2008.

[5] A. D. Ames, H. Zheng, R. D. Gregg, and S. Sastry, "Is there life after Zeno? Taking executions past the breaking (Zeno) point," in *25th American Control Conference*, Minneapolis, MN, 2006.

[6] K. H. Johansson, J. Lygeros, S. Sastry, and M. Egerstedt, "Simulation of Zeno hybrid automata," in *Proceedings of the 38th IEEE Conference on Decision and Control*, Phoenix, AZ, 1999.

[7] J. Zhang, K. H. Johansson, J. Lygeros, and S. Sastry, "Zeno hybrid systems," *Int. J. Robust and Nonlinear Control*, vol. 11, no. 2, 2001.

[8] A. D. Ames, "Characterizing knee-bounce in bipedal robotic walking: A Zeno behavior approach," in *Hybrid Systems: Computation and Control*, Chicago, IL, 2011.

[9] A. Lamperski and A. D. Ames, "On the existence of Zeno behavior in hybrid systems with non-isolated Zeno equilibra," in *IEEE Conference on Decision and Control*, 2008.

[10] Y. Or and A. D. Ames, "Stability of Zeno equilibria in Lagrangian hybrid systems," in *IEEE Conference on Decision and Control*, 2008.

[11] L. P. Carloni, R. Passerone, A. Pinto, and A. L. Sangiovanni-Vincentelli, "Languages and tools for hybrid systems design," *Foundations and Trends in Design Automation*, vol. 1, no. 1/2, 2006.

[12] Wolfram, "SystemModeler," http://wolfram.com/system-modeler, 2012.

[13] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystrom, L. Saldamli, D. Broman, and A. Sandholm, "OpenModelica - a free open-source environment for system modeling, simulation, and teaching," in *Proceedings of the 2006 IEEE International Symposium on Intelligent Control*, 2006.

[14] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, "Modular specification of hybrid systems in CHARON," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science. 2000, vol. 1790.

[15] Z. Wan and P. Hudak, "Functional reactive programming from first principles," in *the Symposium on Programming Language Design and Implementation (PLDI '00)*. ACM, 2000.

[16] H. Nilsson, A. Courtney, and J. Peterson, "Functional reactive programming, continued," in *Haskell '02: Proceedings of the 2002 ACM SIGPLAN workshop on Haskell*, 2002.

[17] A. D. Ames, A. Abate, and S. Sastry, "Sufficient conditions for the existence of Zeno behavior in nonlinear hybrid systems via constant approximations," in *IEEE Conference on Decision and Control*, 2007.

[18] A. Lamperski and A. D. Ames, "Lyapunov theory for Zeno stability," *IEEE Transactions on Automatic Control (To Appear)*, 2013. [Online]. Available: http://ames.tamu.edu/Lyapunov_Zeno.pdf

[19] H. Zheng, E. A. Lee, and A. D. Ames, "Beyond Zeno: Get on with it!" in *Hybrid Systems: Compuation and Control*, 2006.

[20] A. Platzer and J.-D. Quesel, "KeYmaera: A hybrid theorem prover for hybrid systems." in *IJCAR*, ser. LNCS, A. Armando, P. Baumgartner, and G. Dowek, Eds., vol. 5195. Springer, 2008, pp. 171–178.

[21] A. Platzer, "Differential dynamic logic for hybrid systems," *J. Autom. Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

[22] D. Ishii, K. Ueda, and H. Hosobe, "An interval-based sat modulo ode solver for model checking nonlinear hybrid systems," *International Journal on Software Tools for Technology Transfer*, vol. 13, no. 5, 2011.

[23] D. Ishii, "Simulation and verification of hybrid systems based on interval analysis and constraint programming," Ph.D. dissertation, Waseda University, 2010.

[24] X. Chen, E. Abrahám, and 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.

[25] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV)*. Springer, 2011.

[26] C. L. Guernic and A. Girard, "Reachability analysis of linear systems using support functions," *Nonlinear Analysis: Hybrid Systems*, vol. 4, no. 2, pp. 250 – 262, 2010.

[27] E. A. Lee and H. Zheng, "Operational semantics of hybrid systems," in *HSCC*, ser. Lecture Notes in Computer Science, M. Morari and L. Thiele, Eds., vol. 3414. Springer, 2005, pp. 25–53.

[28] J. Moreau, "Numerical aspects of the sweeping process," *Computer Methods in Applied Mechanics and Engineering*, vol. 177, no. 3–4, pp. 329 – 349, 1999.

[29] V. Acary, B. Brogliato, and D. Goeleven, "Higher order Moreau's sweeping process: Mathematical formulation and numerical simulation." *Mathematical Programming*, vol. 113, no. 1, pp. 133–217, 2008.

[30] K. Makino and M. Berz, "New applications of Taylor model methods," in *Automatic Differentiation of Algorithms: From Simulation to Optimization*. Springer, 2002, ch. 43, pp. 359–364.

[31] N. Nedialkov and M. von Mohrenschildt, "Rigorous simulation of hybrid dynamic systems with symbolic and interval methods," in *American Control Conference 2002*, 2002.

[32] A. Rauh, E. P. Hofer, and E. Auer, "ValEncIA-IVP: A comparison with other initial value problem solvers," *Scientific Computing, Computer Arithmetic and Validated Numerics, International Symposium on*, vol. 0, p. 36, 2006.

[33] J. Lygeros, "Lecture notes on hybrid systems," in *Notes for an ENSIETA workshop*, 2004.

[34] A. Edalat and D. Pattinson, "A domain-theoretic account of Picard's theorem," *LMS Journal of Computation and Mathematics*, vol. 10, 2007.