Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 317 (2015) 85–100

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

# Monitoring Bounded LTL Properties Using Interval Analysis

Daisuke Ishii[1] Naoki Yonezaki[2]

*Tokyo Institute of Technology, Tokyo, Japan*

Alexandre Goldsztejn[3]

*CNRS, IRCCyN, Nantes, France*

Abstract

Verification of temporal logic properties plays a crucial role in proving the desired behaviors of hybrid systems. In this paper, we propose an interval method for verifying the properties described by a bounded linear temporal logic. We relax the problem to allow outputting an inconclusive result when verification process cannot succeed with a prescribed precision, and present an efficient and rigorous monitoring algorithm that demonstrates that the problem is decidable. This algorithm performs a forward simulation of a hybrid automaton, detects a set of time intervals in which the atomic propositions hold, and validates the property by propagating the time intervals. A continuous state at a certain time computed in each step is enclosed by an interval vector that is proven to contain a unique solution. In the experiments, we show that the proposed method provides a useful tool for formal analysis of nonlinear and complex hybrid systems.

*Keywords:* Hybrid systems, interval analysis, linear temporal logic, bounded model checking.

## 1 Introduction

Reasoning of the temporal logic properties in a hybrid system is a challenging and important task that lies in the intersection among computer science, numerical analysis, and control theory. Various methods for falsification of hybrid systems with temporal properties have been developed, e.g., [21,20,6,27], and these methods enable verification of various properties (e.g., safety, stability, and robustness) of large and complex systems. The state-of-the-art tools are based on numerical simulations whose numerical errors often produce a qualitatively wrong result and become problematic even in a statistical evaluation.

[1] Email: dsksh@acm.org

[2] Email: yonezaki@cs.titech.ac.jp

[3] Email: alexandre.goldsztejn@gmail.com

http://dx.doi.org/10.1016/j.entcs.2015.10.009

A fundamental process in formal methods for hybrid systems is computation of rigorously approximated reachable sets. The techniques based on interval analysis (Section 3) have shown practicality in the reachability analysis of nonlinear and complex hybrid systems [8,5,24,15,3,11,12]. In these frameworks, computation is $\delta$-*complete* [10]: function values are allowed to be perturbed within a predefined $\delta \in \mathbb{R}_{>0}$, and, by setting bounds in a problem description, many generically undecidable problems become decidable. The $\delta$-complete verification of generic properties other than reachability is a challenging topic.

In this paper, we present an interval method for verifying bounded linear temporal logic (BLTL) properties (Section 5) for a class of hybrid automata (Section 4). Our method computes three values in a reliable manner: the algorithm assures the soundness using interval analysis when the result valid or unsat is output; otherwise, unknown is output when the verification process reaches a prescribed precision threshold. We present an algorithm (Section 6) based on the forward simulation of a system. It encloses a trajectory with a set of *boxes* (i.e., interval vectors) and also ensures the unique existence property (i.e., we ensure that a unique state is enclosed in a box corresponding to each initial value) for each step of the simulation. For each atomic proposition involved in a property $\varphi$ to verify, the algorithm obtains an inner and outer approximation of the time intervals in which the proposition holds. Next, the set of time intervals is modified according to the syntax of the property $\varphi$, and finally the algorithm checks whether $\varphi$ holds at the initial time. Using our implementation, we show that nonlinear models are verified and the numerical robustness of a trajectory is assured (Section 7). Although our method is simple, it enables reliable analysis of a set of trajectories and provides a foundation for validated model checking and controller synthesis.

## 2 Related Work

Many previous studies have applied interval methods to reachability analysis of hybrid systems [8,5,24,15,3,11,12]. The outcome of these methods is an over-approximation of a set of reachable states with a set of boxes. In interval analysis, a computation often provides a proof of unique existence of a solution within a resulting interval. This technique also applies in interval-based reachability analysis [15,14], but it is not considered in most of the methods for hybrid systems. Our method enforces the use of the proof to verify more generic temporal properties.

Reasoning of real-time temporal logic has been a research topic of interest [2,25]. Numerical method for *falsification* of a temporal property is straightforward [16]. It simulates a trajectory of a bounded length and checks the satisfiability of the negation of the property described by a bounded temporal logic. This paper presents an interval extension of this falsification method.

A tree-search method for searching witness trajectories [21], a falsification method based on a Monte-Carlo optimization technique [20], and statistical model checking methods [6,27] for hybrid systems have been proposed. These methods have been shown their practicality in the verification of realistic nonlinear models;

however, their implementations are based on numerical simulations and might suffer from numerical error. An application of our interval method includes an integration with these statistical methods to achieve both reliability and practicality. An integrated statistical and interval method was also proposed in [26] for reachability analysis.

Notions of *robustness* have been proposed to facilitate the simulation-based verification of temporal properties [9,7,20]. In these works, the degree of robustness is represented as a distance between a trajectory and a region where a proposition holds. A non-robust trajectory, which is computed numerically, is likely to be inconsistent with the considered model due to numerical errors. Our method ensures a robustness rigorously by verifying that a trajectory intersects with each boundary in the state space.

There exist a few methods for model checking of temporal logic properties [23,4]. [23] proposed a method specialized in stability properties, which is described as a specific form of temporal logic formula. [4] proposed a method that translates a verification problem into a reachability problem with the $k$-Liveness scheme, which is incomplete in general settings. Our method can be viewed as a bounded model checking method that validates a bounded temporal property by ensuring that all trajectories that emerge from an initial interval value satisfy the property.

## 3 Interval Analysis

This section introduces selected topics and techniques based on interval analysis [17,19]. The techniques are used in the proposed method in Section 6.

### 3.1 Basic Notions and Techniques

A (bounded) *interval* $\boldsymbol{a} = [\underline{a}, \overline{a}]$ is a connected set of real numbers $\{b \in \mathbb{R} \mid \underline{a} \leq b \leq \overline{a}\}$ and $\mathbb{I}$ denotes the set of intervals. For an interval $\boldsymbol{a}$, $\underline{a}$ and $\overline{a}$ denote the lower and upper bounds; the width is defined as $\overline{a} - \underline{a}$; and $\operatorname{int} \boldsymbol{a}$ denotes the interior $\{b \in \mathbb{R} \mid \underline{a} < b < \overline{a}\}$. $[a]$ denotes a point interval $[a, a]$. For intervals $\boldsymbol{a}$ and $\boldsymbol{b}$, $d(\boldsymbol{a}, \boldsymbol{b})$ denotes the hypermetric between the two, i.e., $\max(|\overline{a} - \overline{b}|, |\underline{a} - \underline{b}|)$. For a set $S \subset \mathbb{R}$, $\square S$ denotes the interval $[\inf S, \sup S]$. All of these definitions are naturally extended to interval vectors; an $n$-dimensional *box* (or interval vector) $\boldsymbol{a}$ is a tuple of $n$ intervals $(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$, and $\mathbb{I}^n$ denotes the set of $n$-dimensional boxes. For $a \in \mathbb{R}^n$ and $\boldsymbol{a} \in \mathbb{I}^n$, we use the notation $a \in \boldsymbol{a}$, which is interpreted as $\forall i \in \{1, \ldots, n\}\ a_i \in \boldsymbol{a}_i$. In an actual implementation, the bounds of intervals should be machine-representable floating-point numbers and other real values are rounded in the appropriate directions.

For a function $f : \mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{f} : \mathbb{I}^n \to \mathbb{I}$ is known as an *interval extension* of $f$ if and only if it satisfies the containment condition $\forall \boldsymbol{a} \in \mathbb{I}^n\ \forall a \in \boldsymbol{a}\ (f(a) \in \boldsymbol{f}(\boldsymbol{a}))$. This definition is generalized to function vectors $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^{n_f}$ where $n_f \in \mathbb{N}_{>1}$. Given intervals $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{I}$, interval extensions of four operators $\circ \in \{+, -, *, /\}$ can be computed as $\square\{\underline{a} \circ \underline{b}, \underline{a} \circ \overline{b}, \overline{a} \circ \underline{b}, \overline{a} \circ \overline{b}\}$ (we assume $0 \notin \boldsymbol{b}$ for division).

For arbitrary intervals $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d} \in \mathbb{I}$, the *extended division* $\square\{d \in \boldsymbol{d} \mid \exists a \in \boldsymbol{a}\ \exists b \in$

$\boldsymbol{b}\ a = bd\}$ can be implemented as follows (see Section 4.3 of [19]):

$$\mathsf{ExtDiv}(\boldsymbol{a},\boldsymbol{b},\boldsymbol{d}) := \begin{cases} \boldsymbol{a}/\boldsymbol{b} \cap \boldsymbol{d} & \text{if } 0 \notin \boldsymbol{b} \\ \square(\boldsymbol{d} \setminus (\underline{a}/\underline{b}, \underline{a}/\overline{b})) & \text{if } \boldsymbol{a} > 0 \in \boldsymbol{b} \\ \square(\boldsymbol{d} \setminus (\overline{a}/\overline{b}, \overline{a}/\underline{b})) & \text{if } \boldsymbol{a} < 0 \in \boldsymbol{b} \\ \boldsymbol{d} & \text{if } 0 \in \boldsymbol{a},\boldsymbol{b} \end{cases}$$

In the second and third cases, when $\underline{b} = 0$ (resp. $\overline{b} = 0$), we set $\underline{a}/\underline{b}$ and $\overline{a}/\underline{b}$ as $-\infty$ and $\infty$ (resp. $\underline{a}/\overline{b}$ and $\overline{a}/\overline{b}$ as $\infty$ and $-\infty$).

Given a differentiable function $f(a) : \mathbb{R} \to \mathbb{R}$ and a domain interval $\boldsymbol{a}$, a root $\tilde{a} \in \boldsymbol{a}$ of $f$ such that $f(\tilde{a}) = 0$ is included in the result of an *interval Newton operator*

$$\boldsymbol{a} \cap \left(\hat{a} - \boldsymbol{f}(\hat{a}) / \tfrac{d}{da}\boldsymbol{f}(\boldsymbol{a})\right),$$

where $\hat{a} \in \boldsymbol{a}$, and $\boldsymbol{f}$ and $\frac{d}{da}\boldsymbol{f}$ are interval extensions of $f$ and the derivative of $f$. Iterative applications of the operator will converge. Let $\boldsymbol{a}'$ be the result of applying the operator to $\boldsymbol{a}$. If $\boldsymbol{a}' \subseteq \operatorname{int} \boldsymbol{a}$ holds, a unique root exists in $\boldsymbol{a}'$.

## *3.2 ODE Integration*

An initial value problem (IVP) for an ordinary differential equation (ODE) is specified by a triple $(t_0, x_0, F)$ consisting of an initial value $x_0 \in \mathbb{R}^n$ at time $t_0 \in \mathbb{R}$ and a flow function $F : \mathbb{R}^n \to \mathbb{R}^n$ (assume Lipschitz continuity). Given a time interval $\boldsymbol{t} \in \mathbb{I}$ and a *continuous trajectory* $\tilde{x}(t) : \boldsymbol{t} \to \mathbb{R}^n$, the satisfaction for IVP-ODEs is defined as

$$\tilde{x}, \boldsymbol{t} \models (t_0, x_0, F) \ \text{ iff } \ \tilde{x}(t_0) = x_0 \wedge \forall \tilde{t} \in \boldsymbol{t}\ \tfrac{d}{dt}\tilde{x}(\tilde{t}) = F(\tilde{x}(\tilde{t})).$$

Given $\boldsymbol{t}_0 \in \mathbb{I}$ and $\boldsymbol{x}_0 \in \mathbb{I}^n$, we can consider a parametric IVP-ODE $(\boldsymbol{t}_0, \boldsymbol{x}_0, F)$, where the initial condition is parameterized, and its satisfaction relation is defined as

$$\tilde{x}, \boldsymbol{t} \models (\boldsymbol{t}_0, \boldsymbol{x}_0, F) \ \text{ iff } \ \exists t_0 \in \boldsymbol{t}_0\ \exists x_0 \in \boldsymbol{x}_0\ \tilde{x}, \boldsymbol{t} \models (t_0, x_0, F).$$

$TS_{\boldsymbol{t}}(\boldsymbol{t}_0, \boldsymbol{x}_0, F)$ denotes the set of satisfied trajectories on $\boldsymbol{t}$.

Using the tools based on the interval Taylor methods, e.g., CAPD [4] and VNODE [18], we can obtain an interval extension $\mathsf{X} : \mathbb{I} \to \mathbb{I}^n$ of solution trajectories in $TS_{\boldsymbol{t}}(\boldsymbol{t}_0, \boldsymbol{x}_0, F)$. Given $\boldsymbol{t}' \in \mathbb{I}$, such tools compute a value $\mathsf{X}(\boldsymbol{t}')$ by performing the stepwise integration of the flow function $F$ from the initial time $\boldsymbol{t}_0$ to time $\overline{t}'$. In the stepwise computation of the interval Taylor methods, the *unique existence* of a solution is verified for a box enclosure computed in each step based on the Picard-Lindelöf operator and Banach's fixpoint theorem. Accordingly, when an interval enclosure $\mathsf{X}(\boldsymbol{t}')$ (assume $\underline{t}' \geq \overline{t}_0$) is computed with an interval Taylor method, the following property holds:

$$\forall t_0 \in \boldsymbol{t}_0\ \forall x_0 \in \boldsymbol{x}_0\ \exists\text{unique}\ \tilde{x} \in (\boldsymbol{t}' \to \mathsf{X}(\boldsymbol{t}'))\ \tilde{x}, \boldsymbol{t}' \models (t_0, x_0, F).$$

[4] http://capd.ii.uj.edu.pl/

In principle, if $F$ is Lipschitz continuous and we can assume an arbitrary precision, we obtain an arbitrary narrow interval enclosure $\mathsf{X}([t])$ for $t \in \mathbb{R}$. However, since the implementations use machine-representable real numbers, it may fail to compute an enclosure in the process that verifies the unique existence property, even with the smallest step size.

## 4 Hybrid Automata

We model a hybrid system as a hybrid automaton [1]. For simplicity in this paper, we consider deterministic systems, i.e., the location invariant is the negation of guard conditions and two guards do not overlap in a location. The proposed method can be extended to handle non-deterministic systems, e.g., by enumerating possible paths and computing a trajectory enclosure for each path.

**Definition 4.1** A *hybrid automaton* is a septet

$$HA := \big(Q, x, X, \mathit{Init}, \{F_q\}_{q\in Q}, \{G_{q,q'}\}_{q\in Q, q'\in Q}, \{R_{q,q'}\}_{q\in Q, q'\in Q}\big),$$

that consists of the following components:

- A finite set of *locations* $Q = \{q_1, \ldots, q_{n_q}\}$.
- A vector of real-valued variables $x = (x_1, \ldots, x_n)$.
- A domain $X \subseteq \mathbb{I}^n$ for the valuation of the variables.
- A set of initial values $\mathit{Init} \subseteq \{q\} \times X$ where $q \in Q$.
- A set of *vector fields* $F_q : X \to X$ (assume Lipschitz continuity).
- A set of *guards* $G_{q,q'} \subseteq X$ described by a condition of the form $g(x) = 0 \wedge h(x) < 0$ where $g, h : X \to \mathbb{R}$.
- A set of *reset functions* $R_{q,q'} : X \to X$.

Behaviors of the states $\sigma \in Q \times X$ over the timeline are formalized as *trajectories*. In this work, we assume that there are no consecutive multiple discrete changes.

**Definition 4.2** Given an $HA$, an initial state $(q_0, s_0) \in \mathit{Init}$, and a time interval $\boldsymbol{t} = [0, t_{\max}]$ ($t_{\max} \in \mathbb{R}_{\geq 0}$), a state at each time $t \in \boldsymbol{t}$ is determined as a *trajectory* $(\mathbf{q}, \mathbf{s})$, which consists of a *location trajectory* $\mathbf{q} : \boldsymbol{t} \to Q$ and a *continuous state trajectory* $\mathbf{s} : \boldsymbol{t} \to X$. The value of the trajectory is defined recursively as follows:

$$\begin{aligned}&(\mathbf{q}(0), \mathbf{s}(0)) := (q_0, s_0),\\ &(\mathbf{q}(t), \mathbf{s}(t)) := \sigma \in Q \times X \ \text{ s.t. } \exists t' \in (0, t)\ \exists \sigma' \in Q \times X\ (\mathbf{q}(t'), \mathbf{s}(t')) \xrightarrow{t-t'} \sigma' \xrightarrow{0} \sigma,\end{aligned}$$

where the relation $\sigma_1 \xrightarrow{t} \sigma_2 \in (Q \times X) \times \boldsymbol{t} \times (Q \times X)$ is given by the following rules:

$$\frac{G_{q,q'}(s) \quad R_{q,q'}(s) = s'}{(q, s) \xrightarrow{0} (q', s')} \qquad \frac{\begin{array}{r} f(0) = s \quad \forall \tilde{t} \in [0, t]\ \frac{d}{dt} f(\tilde{t}) = F_q(f(\tilde{t})) \\ \forall \tilde{t} \in [0, t)\ \forall q' \in Q\ \neg G_{q,q'}(f(\tilde{t}))\end{array}}{(q, s) \xrightarrow{t} (q, f(t))}$$

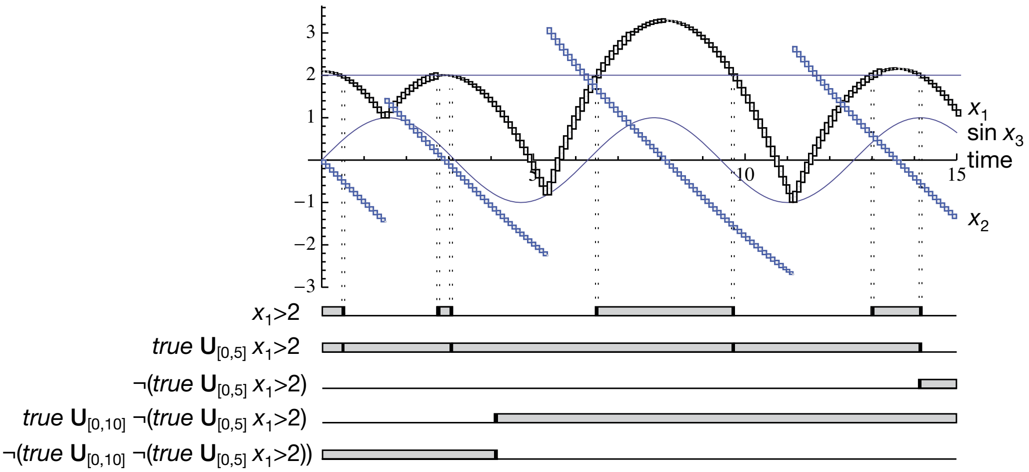


Figure 1. Verification of the bouncing ball example

Note that the second rule also applies for $t = 0$. The set of trajectories of length $t_{\max}$ is denoted by $TS_{t_{\max}}(HA)$.

When a discrete change $\overset{0}{\to}$ is applied at time $t$, the state $(\mathbf{q}(t), \mathbf{s}(t))$ overwrites the state $\sigma'$ before the discrete change. An $HA$ has a unique trajectory $(\mathbf{q}, \mathbf{s})$ starting from an initial state $(q_0, s_0) \in \mathit{Init}$ because we have assumed that two guards do not hold simultaneously.

**Example 4.3** We model a bouncing ball on a moving table as an $HA$:

$$\begin{aligned}
x &:= (x_1, x_2, x_3) \in X := ([-1, 10], [-10, 10], [0, 1000]) \\
L &:= \{q\} \\
\mathit{Init} &:= \{q\} \times ([2, 7], [0], [0]) \\
F_q &:= (x_2, -1 + 0.04x_2^2 \operatorname{sgn} x_2, 1) \\
G_{q,q} &:= x_1 - \sin x_3 = 0 \wedge x_2 - \cos x_3 < 0 \\
R_{q,q} &:= (x_1, -0.9x_2 + 1.9 \cos x_3, x_3)
\end{aligned}$$

Variables $x_1, x_2$, and $x_3$ represent the height and velocity of the ball, and the (global) time, respectively. Air resistance is considered in the dynamics. The height of the table sinusoidally oscillates within $[-1, 1]$ and is represented as $\sin x_3$. The second proposition of the guard is to forbid the guard to hold right after a discrete change. Possible trajectories of $x_1$ and $x_2$ are illustrated in Figure 1.

## 5 Bounded Linear Temporal Logic

We consider a fragment [16] of the real-time metric temporal logic [2] such that the temporal modalities are bounded by an interval $\boldsymbol{t} = [\underline{t}, \overline{t}]$ such that the bounds $\underline{t}, \overline{t}$ are in $\mathbb{Q}$. We refer to the logic *bounded linear temporal logic* (BLTL) as in [27].

**Definition 5.1** We consider constraints in the real domain as atomic propositions. The syntax of the BLTL formulae is defined by the grammar

$$\varphi ::= \mathsf{true} \mid p \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \, \mathsf{U}_{\boldsymbol{t}} \, \varphi \qquad p ::= f(x) < 0 \mid f(x) \leq 0$$

where $p$ belongs to a set of *atomic propositions* $AP_\varphi$, $\mathsf{U}_{\boldsymbol{t}}$ is the "until" operator bounded with a non-empty positive time interval $\boldsymbol{t} \in \mathbb{I}$, $x = (x_1, \dots, x_n)$ is a vector of variables, and $f : \mathbb{R}^n \to \mathbb{R}$. We use the standard abbreviations, e.g., $\varphi_1 \wedge \varphi_2 := \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\mathsf{F}_{\boldsymbol{t}}\varphi := \mathsf{true}\,\mathsf{U}_{\boldsymbol{t}}\,\varphi$ ("eventually"), and $\mathsf{G}_{\boldsymbol{t}}\varphi := \neg\mathsf{F}_{\boldsymbol{t}}\neg\varphi$ ("always"). An equation $f(x) = 0$ can be encoded as $f(x) \leq 0 \wedge -f(x) \leq 0$.

## 5.1 Semantics

The necessary length $\|\varphi\|$ of trajectories for checking a formula $\varphi$ is inductively defined by the structure of the formula:

$$\begin{aligned} \|p\| &:= 0 & \qquad \|\varphi_1 \vee \varphi_2\| &:= \max\left(\|\varphi_1\|, \|\varphi_2\|\right) \\ \|\neg\varphi\| &:= \|\varphi\| & \qquad \|\varphi_1 \, \mathsf{U}_{\boldsymbol{t}} \, \varphi_2\| &:= \max\left(\|\varphi_1\|, \|\varphi_2\|\right) + \overline{t} \end{aligned}$$

A map $\mathcal{O} : AP_\varphi \to 2^X$ corresponds each proposition $p \in AP_\varphi$ to a set $\mathcal{O}(p) = \{s \in X \mid p(s)\}$. Let $(\mathbf{q}, \mathbf{s})$ be a trajectory in $TS_{\|\varphi\|}(HA)$ and $\varphi$ be a BLTL property. We have a satisfaction relation defined as follows:

$$\begin{aligned} &\mathbf{s}, t \models \mathsf{true} \\ &\mathbf{s}, t \models p && \text{iff} \;\; \mathbf{s}(t) \in \mathcal{O}(p) \\ &\mathbf{s}, t \models \varphi_1 \vee \varphi_2 && \text{iff} \;\; \mathbf{s}, t \models \varphi_1 \;\vee\; \mathbf{s}, t \models \varphi_2 \\ &\mathbf{s}, t \models \neg\varphi && \text{iff} \;\; \mathbf{s}, t \not\models \varphi \\ &\mathbf{s}, t \models \varphi_1 \, \mathsf{U}_{\boldsymbol{t}} \, \varphi_2 && \text{iff} \;\; \exists t' \in (t + \boldsymbol{t}) \; \mathbf{s}, t' \models \varphi_2 \;\wedge\; (\forall t'' \in [t, t'] \; \mathbf{s}, t'' \models \varphi_1) \end{aligned}$$

$\varphi_1 \, \mathsf{U}_{\boldsymbol{t}} \, \varphi_2$ intuitively means that (assuming we are at time $t$) $\varphi_2$ will hold within the time interval $t + \boldsymbol{t}$ and $\varphi_1$ always hold until then. We also have a validation relation defined as:

$$HA \models \varphi \;\; \text{iff} \;\; \forall(\mathbf{q}, \mathbf{s}) \in TS_{\|\varphi\|}(HA) \; \mathbf{s}, 0 \models \varphi$$

## 5.2 Method for Monitoring BLTL Formulae

Our interval method is based on the method proposed in [16] that decides whether a trajectory satisfies a BLTL property. In this section, we explain this basic method. First, we introduce the notion of consistent time intervals against BLTL formulae.

**Definition 5.2** Let $(\mathbf{q}, \mathbf{s})$ be a trajectory of length $t_{\max}$ and $\varphi$ be a BLTL formula. We say that a left-closed and right-open interval $[\underline{t}, \overline{t}) \subset [0, t_{\max}]$ is *consistent* with $\varphi$ iff $\forall t \in [\underline{t}, \overline{t}) \; \mathbf{s}, t \models \varphi$.

The satisfiability of a property $\varphi$ by a trajectory is checked as follows:

(i) For each atomic proposition $p$ in $\varphi$, monitor the trajectory of length $\|\varphi\|$ and identify a non-overlapping set of consistent time intervals $T_p = \{\boldsymbol{t}_1, \dots, \boldsymbol{t}_{n_p}\}$.

(ii) Following the parse tree of $\varphi$ in a bottom-up fashion, compute a set of consistent time intervals of $\varphi$. For each construct of BLTL, compute as follows:

$$T_{\neg p} := [0, \|\varphi\|) \setminus T_p \qquad T_{p_1 \vee p_2} := T_{p_1} \cup T_{p_2}$$
$$T_{p_1 \mathsf{U}_{\boldsymbol{t}} p_2} := \{\mathsf{Shift}_{\boldsymbol{t}}(\boldsymbol{t}_1 \cap \boldsymbol{t}_2) \cap \boldsymbol{t}_1 \mid \boldsymbol{t}_1 \in T_{p_1}, \boldsymbol{t}_2 \in T_{p_2}\}$$

where $\mathsf{Shift}_{\boldsymbol{t}}(\boldsymbol{s}) := [\underline{s} - \overline{t}, \overline{s} - \underline{t}) \cap [0, \|\varphi\|)$.

(iii) Check whether $\boldsymbol{t}_1 \in T_\varphi$ contains time 0. If yes, $\varphi$ is satisfied; otherwise, it is not satisfied.

**Example 5.3** We verify the property

$$\mathsf{G}_{[0,10]} \mathsf{F}_{[0,5]}\, 2 - x_1 < 0 \;\equiv\; \neg(\mathsf{true}\, \mathsf{U}_{[0,10]}\, \neg(\mathsf{true}\, \mathsf{U}_{[0,5]}\, 2 - x_1 < 0))$$

for the model in Example 4.3. Computation with the monitoring method (which is extended to an interval method) is illustrated in Figure 1.

## 6 Interval-Based Simulation and Monitoring Method

In this section, we propose an interval extension of the monitoring method in Section 5.2.

In Step (i) of the method, given an *HA* and a BLTL property $\varphi$, we first simulate the *HA* for length $\|\varphi\|$. Our simulation method computes an over-approximation of a trajectory of *HA* in which the existence of a unique trajectory is verified, i.e., it verifies the property

$$\forall (q_0, x_0) \in \mathit{Init}\ \exists \text{unique}\ (\mathbf{q}, \mathbf{s}) \in \mathit{TS}_{\|\varphi\|}(\mathit{HA}) \ \ \mathbf{q}(0) = q_0 \wedge \mathbf{s}(0) = x_0$$

meaning that for each initial value, there exists a unique trajectory of length $\|\varphi\|$.

Next, our method identifies the time intervals that are consistent with an atomic proposition in $\varphi$ (Definition 5.2). A consistent time interval $[\underline{t}, \overline{t})$ is, in general, not representable in an actual implementation; therefore, we approximate it by a pair of intervals $\boldsymbol{u}$ and $\boldsymbol{u}'$ such that each of them encloses the boundaries $\underline{t}$ or $\overline{t}$. Given an atomic proposition $f(x) \circ 0$ ($\circ \in \{<, \leq\}$) and a trajectory $\mathbf{s}$, we search for a boundary enclosure $\boldsymbol{u}$ such that $\boldsymbol{f}(\mathbf{s}(\boldsymbol{u})) \ni 0$ where $\boldsymbol{f}$ is an interval extension of $f$.

In Step (ii), the set of consistent time intervals is updated to be consistent with $\varphi$. This computation requires $\boldsymbol{u}$ to contain a unique boundary point, and thus a naive over-approximation $\tilde{\boldsymbol{u}}$ does not suffice because an interval extension $\boldsymbol{f}(\tilde{\boldsymbol{u}})$ may contain 0 although $f(\tilde{\boldsymbol{u}})$ does not contain a boundary or contains several boundaries. Thanks to interval techniques, our method verifies the unique existence of a boundary in $\boldsymbol{u}$. Finally, as an extension of the above property, our method verifies

$$\forall (q_0, x_0) \in \mathit{Init}\ \exists \text{unique}\ (\mathbf{q}, \mathbf{s}) \in \mathit{TS}_{\|\varphi\|}(\mathit{HA}) \ \ \mathbf{q}(0) = q_0 \wedge \mathbf{s}(0) = x_0 \wedge \mathbf{s}, 0 \models \varphi.$$

The proposed method has some limitations. First, it is a semi-decision procedure that may output an inconclusive result ($\mathsf{unknown}$) because of a failure in

```
Input: $HA$, $\varphi$
Output: valid, unsat, or unknown
1: $\boldsymbol{t} := 0; \quad q := q_0; \quad \boldsymbol{x} := \boldsymbol{x}_0; \quad \boldsymbol{T} := \{\emptyset, \dots, \emptyset\}$
2: while $\underline{t} < \|\varphi\|$ do try
3:    $\boldsymbol{t}_c := \|\varphi\|$
4:    for $q' \in Q$ do                                              {Find zeros for each edge}
5:       $\boldsymbol{t}'_c :=$ SearchZero(X, $F_q, G_{q,q'}, [\underline{t}, \|\varphi\|]$)
6:       if $\boldsymbol{t}'_c \neq \emptyset \wedge \overline{t}'_c < \underline{t}_c$ then
7:          $q'' := q'; \ \boldsymbol{t}_c := \boldsymbol{t}'_c$
8:       else if $\overline{t}'_c \geq \underline{t_c}$ then
9:          return unknown
10:      end if
11:   end for
12:   for $p = f \circ 0 \in AP_\varphi$ do                            {Find boundaries of APs}
13:      $\boldsymbol{t}'_c := [\underline{t}, \overline{t}_c]$
14:      loop
15:         $\boldsymbol{t}'_c :=$ SearchZero(X, $F_q, f = 0, \boldsymbol{t}'_c$)
16:         if $\boldsymbol{t}'_c = \emptyset$ then break end if
17:         $\boldsymbol{t}'_c := [\overline{t}'_c, \overline{t}_c]; \quad \boldsymbol{T}_p := \boldsymbol{T}_p \cup \{\boldsymbol{t}'_c\}$
18:      end loop
19:   end for
20:   $\boldsymbol{x} :=$ Jump(X, $\boldsymbol{t}_c, R_{q,q''}, AP_\varphi, \boldsymbol{T}$)            {Discrete change}
21:   $q := q''; \quad \boldsymbol{t} := \boldsymbol{t}_c$
22: catch error then return unknown end try
23: end while
24: return AnalIntervals($\boldsymbol{T}$)
```

Figure 2. Monitor algorithm

the verification of unique existence; both the procedures for enclosing a continuous trajectory and enclosing a time where a discrete change occurs may cause errors. However, this mechanism is valuable in terms of reliability and complexity of the problem; a non-robust trajectory and a zeno $HA$ will be rejected as an error in the verification process. In practice, when addressing a nonlinear $HA$s, the method may only work successfully with a sufficiently small subset $Init' \subset Init$ of initial values. In this way, the method can be still used for sat/invalid checking. Second, the method is a bounded model-checking method in the sense that the domain $X$ of the variables is bounded, and it assumes a bounded length and a number of discrete changes in a trajectory.

### *6.1 Main Algorithm*

Given an $HA$ and a BLTL property $\varphi$, the proposed Monitor algorithm (Figure 2) outputs the following results: valid that implies $HA \models \varphi$; unsat that implies $HA \models$

$\neg\varphi$; or $\mathsf{unknown}$ when the computation is inconclusive.

An iteration of the outmost loop corresponds to a continuous phase of the trajectory and a discrete change. At Lines 4–11, each guard for a possible transition is evaluated. The $\mathsf{SearchZero}$ algorithm is described in Section 6.4 and will return a time interval $\boldsymbol{t}_c'$ within which the guard is satisfied or $\emptyset$ if no state satisfies the guard. Next, the algorithm attempts to decide the *earliest* time interval by checking whether $\boldsymbol{t}_c'$ is strongly less than $\boldsymbol{t}_c$. If two guard crossings are too close, so two crossing time intervals overlap, the algorithm returns $\mathsf{unknown}$. At Lines 12–19, for each atomic proposition of $\varphi$ of the form $f \circ 0$ ($\circ \in \{<, \leq\}$), the algorithm searches for a boundary where the sign of $f$ changes. Because several boundaries can exist in a continuous phase, the inner loop searches for all of them. The detected time intervals are saved in the set $\boldsymbol{T}$ associated with the atomic propositions. At Lines 20–21, the discrete change between the locations $q$ and $q''$ is computed by evaluating an interval extension of $R_{q,q''}(\mathsf{X}(\boldsymbol{t}_c))$. A jump of state might switch the state of an atomic proposition; if such a switch exists, the $\mathsf{Jump}$ procedure should detect and record it in $\boldsymbol{T}$. Finally, boundary points (which are enclosed by intervals) of the consistent time intervals saved in $\boldsymbol{T}$ are analyzed by the $\mathsf{AnalIntervals}$ procedure (Line 24, Section 6.2). The procedures $\mathsf{X}$ (Section 3.2) and $\mathsf{SearchZero}$ (Section 6.4) may results in errors. These errors are caught by the catch clause at Line 22.

## 6.2 Evaluation of BLTL Properties

The BLTL evaluation explained in Section 5.2 can be implemented as a rigorously approximated procedure $\mathsf{AnalIntervals}$.

First, we approximate a set of consistent time intervals $T_\varphi = \{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_{n_\varphi}\}$ by $\boldsymbol{T}_\varphi = \{\boldsymbol{u}_1, \boldsymbol{u}_1', \ldots, \boldsymbol{u}_{n_\varphi}, \boldsymbol{u}_{n_\varphi}'\}$ such that $\boldsymbol{u}_i, \boldsymbol{u}_i' \in \mathbb{I}$, $\underline{t_i} \in \boldsymbol{u}_i$, $\overline{t_i} \in \boldsymbol{u}_i'$, $\overline{u}_i \leq \underline{u}_i'$, and $\overline{u}_i' \leq \underline{u}_{i+1}$, for $i \in \{1, \ldots, n_\varphi\}$. $T = \emptyset$ and $T = \{[0, t_{\max})\}$ are approximated as $\boldsymbol{T} = \emptyset$ and $\boldsymbol{T} = \{[0], [t_{\max}]\}$. For $\boldsymbol{u}_i$, $\boldsymbol{u}_i'$ in $\boldsymbol{T}_\varphi$ and a continuous trajectory $\mathbf{s}$, $\forall t \in [\overline{u}_i, \underline{u}_i')\ \mathbf{s}, t \models \varphi$ holds.

Next, the evaluation on the set of time intervals in Step (ii) is extended to address the approximated sets: the set of the inverted time intervals for $\neg\varphi$ can be represented as $\boldsymbol{T}_{\neg\varphi} = \{[0], \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n_\varphi}', [t_{\max}]\}$; the union of two sets of time intervals for $\varphi_1 \vee \varphi_2$ can be implemented as a merge and sort process of the two approximated sets; and the $\mathsf{Shift}_{\boldsymbol{t}}$ procedure for $\varphi_1 \mathsf{U}_{\boldsymbol{t}} \varphi_2$ can be implemented as translations of the time intervals $\boldsymbol{u}_i$ and $\boldsymbol{u}_i'$ for $\overline{t}$ and $\underline{t}$, respectively. Some more case analyses should be applied, e.g., when the intervals in $\boldsymbol{T}_\varphi$ become redundant or when they overlap.

Finally, we obtain $\boldsymbol{T}_\varphi$ and conclude that $\varphi$ is $\mathsf{valid}$ if $\overline{u}_1 \leq 0 \leq \underline{u}_1'$; it is $\mathsf{unsat}$ if $\boldsymbol{T}_\varphi = \emptyset$ or $0 < \underline{u}_1$; or the satisfaction is $\mathsf{unknown}$ if $0 \in [\underline{u}_1, \overline{u}_1)$.

## 6.3 Computation of a Continuous Trajectory

If the system is in a location $q \in Q$ and the value of the state variables is $\boldsymbol{x} \in \mathbb{I}^n$ at time $\boldsymbol{t} \in \mathbb{I}$, then the subsequent continuous evolution is specified by an IVP-ODE $(\boldsymbol{t}, \boldsymbol{x}, F_q)$. Next, we can obtain an interval extension $\mathsf{X} : \mathbb{I} \to \mathbb{I}^n$ of the continuous

```
Input: $\mathsf{X} : \mathbb{I} \to \mathbb{I}^n$,   $F : X \to X$,   $g = 0 \wedge h < 0$,   $\boldsymbol{t}_{\mathrm{init}} \in \mathbb{I}$
Output: $\boldsymbol{t} \in \mathbb{I} \cup \{\emptyset\}$
Parameter: $\epsilon \in \mathbb{Q}_{>0}$, $\theta \in (0, 1)$
 1: $\boldsymbol{t} := \boldsymbol{t}_{\mathrm{init}}$
 2: repeat                                                    {Lower bound reduction}
 3:    $\boldsymbol{t}_{\mathrm{old}} := \boldsymbol{t}$
 4:    $\boldsymbol{d}_g := \mathsf{Dt}(g, \mathsf{X}, F, \boldsymbol{t})$;  $\boldsymbol{d}_h := \mathsf{Dt}(h, \mathsf{X}, F, \boldsymbol{t})$
 5:    $\boldsymbol{t} := \underline{t} + \mathsf{ExtDiv}(-\boldsymbol{g}(\mathsf{X}(\underline{t})),\ \boldsymbol{d}_g,\ \boldsymbol{t} - \underline{t})$
 6:    $\boldsymbol{t} := \underline{t} + \mathsf{ExtDiv}(-\boldsymbol{h}(\mathsf{X}(\underline{t})) - [0, \infty],\ \boldsymbol{d}_h, \boldsymbol{t} - \underline{t})$
 7: until $d(\boldsymbol{t}_{\mathrm{old}}, \boldsymbol{t}) \leq \epsilon$
 8: if $\boldsymbol{t} = \emptyset$ then return $\emptyset$ end if
 9: $\boldsymbol{t} := \underline{t}$
10: loop                                       {Unique solution existence verification}
11:    $\boldsymbol{d}_g := \mathsf{Dt}(g, \mathsf{X}, F, \boldsymbol{t})$
12:    if $\boldsymbol{d}_g \ni 0$ then error end if
13:    $\boldsymbol{t}' := \underline{t} - \boldsymbol{g}(\mathsf{X}(\underline{t}))/\boldsymbol{d}_g$
14:    if $\boldsymbol{t}' \subseteq \mathrm{int}\, \boldsymbol{t}$ then $\boldsymbol{t} := \boldsymbol{t}'$; break end if
15:    $\boldsymbol{t}_{\mathrm{bak}} := \boldsymbol{t}$
16:    $\boldsymbol{t} := \boldsymbol{t}_{\mathrm{init}} \cap \mathsf{Inflate}(\boldsymbol{t}', 1+\theta)$
17:    if $d(\boldsymbol{t}, \boldsymbol{t}') > (1-\theta)\, d(\boldsymbol{t}', \boldsymbol{t}_{\mathrm{bak}})$ then error end if
18: end loop
19: if $\sup \boldsymbol{h}(\mathsf{X}(\boldsymbol{t})) \geq 0$ then error end if
20: return $\boldsymbol{t}$
```

Figure 3. SearchZero algorithm

trajectories in $TS_{[\underline{t}, \|\varphi\|]}(\boldsymbol{t}, \boldsymbol{x}, F_q)$ as described in Section 3.2.

## 6.4 Evaluation of Boundary Conditions

Guards and atomic propositions can be treated as *boundary conditions* in the state space $X \subseteq \mathbb{R}^n$ of the form

$$B(x) \ := \ g(x) = 0 \wedge h(x) < 0,$$

where $g : X \to \mathbb{R}$ and $h : X \to \mathbb{R}$. We propose the SearchZero algorithm shown in Figure 3 for searching the intersection between a trajectory and a boundary condition. Inputs to the algorithm consist of an interval extension of the continuous trajectory $\mathsf{X} : \mathbb{I} \to \mathbb{I}^n$, a vector field of the current location $F : X \to X$, the boundary condition $B(x)$, and a time interval $\boldsymbol{t}_{\mathrm{init}} \in \mathbb{I}$ to be searched. SearchZero searches for the earliest time interval $\boldsymbol{t} \subseteq \boldsymbol{t}_{\mathrm{init}}$ such that the state $\mathsf{X}(\boldsymbol{t})$ encloses a unique solution of the boundary condition, i.e.,

$$\boldsymbol{t} = \square\big\{\min\{t \in \boldsymbol{t}_{\mathrm{init}} \mid B(\mathsf{s}(t))\} \mid \mathsf{s} \in TS_{\boldsymbol{t}_{\mathrm{init}}}(\boldsymbol{t}_0, \boldsymbol{x}_0, F_q)\big\}, \tag{1}$$

where $(\boldsymbol{t}_0, \boldsymbol{x}_0, F_q)$ denotes the IVP-ODE of the current location. Moreover, SearchZero verifies the following property:

$$\forall \mathbf{s} \in TS_{\boldsymbol{t}_{\text{init}}}(\boldsymbol{t}_0, \boldsymbol{x}_0, F_q) \ \exists \text{unique } t \in \boldsymbol{t} \ B(\mathbf{s}(t)) \tag{2}$$

Otherwise, SearchZero returns $\emptyset$ when the boundary condition is unsatisfiable, i.e.,

$$\forall \mathbf{s} \in TS_{\boldsymbol{t}_{\text{init}}}(\boldsymbol{t}_0, \boldsymbol{x}_0, F_q) \ \forall t \in \boldsymbol{t}_{\text{init}} \ \neg B(\mathbf{s}(t)). \tag{3}$$

**Lemma 6.1** *If SearchZero returns a non-empty interval* $\boldsymbol{t}$*, the properties* (1) *and* (2) *hold. If it returns* $\emptyset$*, the property* (3) *holds.*

To justify the soundness, we describe some details of the algorithm. At Lines 2–7, the time interval $\boldsymbol{t}$ is filtered repeatedly using an interval Newton operator. At Line 4 (and at Line 11), given a function $g$, the Dt procedure computes an interval enclosure of the derivative $\frac{d}{dt}g(\mathbf{s}(t))$ over the time interval $\boldsymbol{t}$ using the chain rule

$$\tfrac{d}{dt}g(\mathbf{s}(\boldsymbol{t})) = \tfrac{d}{dx}g(\mathbf{s}(\boldsymbol{t})) \cdot \tfrac{d}{dt}\mathbf{s}(\boldsymbol{t}) \ \subseteq \ \tfrac{d}{dx}\boldsymbol{g}(\mathsf{X}(\boldsymbol{t})) \cdot \boldsymbol{F}(\mathsf{X}(\boldsymbol{t})).$$

Next, at Lines 5 and 6, the interval Newton is applied. The extended division (Section 3) is used to implement the interval Newton to handle the numerator intervals $\boldsymbol{d}_g, \boldsymbol{d}_h$ containing zero. Because we expand the interval Newton on the lower bound $\underline{t}$ and the extended division encloses the values in the domain $\boldsymbol{t} - \underline{t}$, the resulting $\boldsymbol{t}$ is filtered its inconsistent portion, without losing the solutions or being expanded. When the interval Newton results in $\emptyset$, SearchZero also returns $\emptyset$ to signal the unsatisfiability. At Line 9, because $\boldsymbol{t}$ may contain several solutions, $\boldsymbol{t}$ is reset to the lower bound as a starting value to compute an enclosure of the earliest solution. At Lines 10–18, SearchZero applies the interval Newton method with the inclusion test to prove the unique existence of a solution within the contracted interval $\boldsymbol{t}'$. This interval Newton verification is repeated with an inflation process of the time interval (see [13] for a detailed implementation). When reaching Line 19 with no error, the time interval $\boldsymbol{t}$ is a sharp enclosure of the first zero of $g(\mathbf{s}(t)) = 0$. It remains to check that the inequality constraint $h(\mathbf{s}(t)) < 0$ is satisfied inside $\boldsymbol{t}$.

When SearchZero is implemented with machine representable real numbers, or when there is a tangency between the trajectory and the guard constraint, a computation may result in an error. Line 12 of SearchZero may give rise to error if the derivative on an (inflated) time interval contains zero. At Line 17, we limit the number of iterations according to whether the inflation ratio reaches the threshold as proposed in [13].

## 7 Experiments

We have implemented the proposed method and experimented on two examples to confirm the effectiveness of the method. Experiments were run using a 2.4GHz Intel Core i5 processor with 16GB of RAM.

## 7.1 Implementation

We have implemented the proposed algorithms in Figures 2 and 3 in OCaml and C/C++. The CAPD library was used for solving ODEs. Parameters $t_{\min}$, $\epsilon$, and $\theta$ should be configured. Each parameter corresponds to the smallest integration step size that CAPD can take, the threshold used in Figure 3, or a threshold used in Inflate. In the experiments, these parameters were set as $t_{\min} := 10^{-14}$, $\epsilon := 10^{-14}$, and $\theta := 0.01$.

For most of models, our implementation only accepts small interval values, as reported in the next section, because an interval enclosure of the state after a continuous evolution and a jump expands by quite a large amount, and thus, the verification process in SearchZero or the solving process of CAPD will fail.

## 7.2 Bouncing ball

Example 5.3 can be verified using the implementation by limiting the initial value of $x_2$ to a small interval of width at most 0.01. We verified the model with three configurations: 1) with setting a point initial value to $x_2$; 2) with an interval initial value of width 0.01; and 3) with a point initial value and the property in which the time bound for the G operator was set to $[0, 100]$. Each experiment was run 1000 times with the initial values of $x_2$ randomly picked within $[2, 7]$. The results are shown in Table 1. In the table, the second column represents the width of the

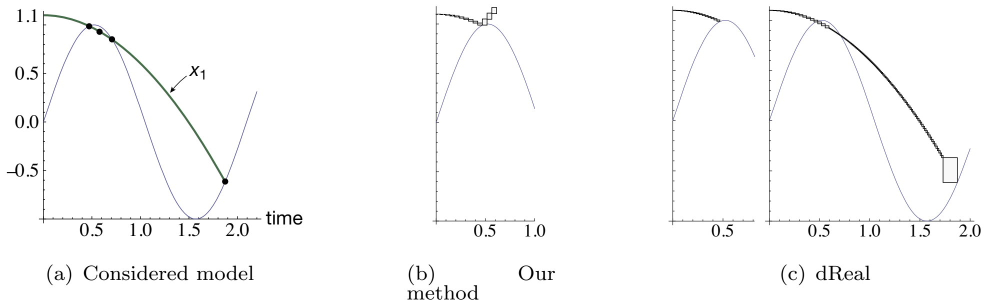
Table 1
Experimental results (bouncing ball)

| $\varphi$ | width | # valid | # unsat | # unknown | # errors | time |
|---|---|---|---|---|---|---|
| $\mathsf{G}_{[0,10]}\mathsf{F}_{[0,5]}2-x_1<0$ | 0 | 330 | 670 | 0 | 0 | 0.1s |
| $\mathsf{G}_{[0,10]}\mathsf{F}_{[0,5]}2-x_1<0$ | 0.01 | 87 | 10 | 903 | 903 | 0.1s |
| $\mathsf{G}_{[0,100]}\mathsf{F}_{[0,5]}2-x_1<0$ | 0 | 186 | 20 | 794 | 794 | 0.5s |

initial values. For each $r \in \{\mathsf{valid}, \mathsf{unsat}, \mathsf{unknown}\}$, the column "# $r$" represents the number of runs that resulted in $r$. The column "# errors" represents that how many of unknown results are caused by an error in the SearchZero procedure. The column "time" shows average timings.

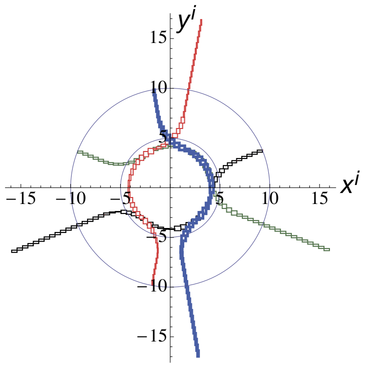
The computation of each experiment was quite efficient.

Regarding the rate of inconclusive runs in each experiment, all the verifications succeeded in the first experiment. Because we set the point initial values and the bounded simulation time, considered trajectories were always enclosed with tight intervals, and thus the verification process succeeded even in a situation that was close to singular. The result also implied that we did not meet a zeno behavior. Contrastingly, around 90% and 80% of the runs in the second and third experiments, respectively, resulted in errors. Our verification process with the interval Newton failed more often if a trajectory and a guard approached or they became close to tangent. Because the model was chaotic, a coarser enclosure of states or a longer simulation time increased the possibility to meet such situations.



(a) Considered model (b) Our method (c) dReal

Figure 4. Results of boundary detection of the bouncing ball example



$$\bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} \mathsf{G}_{[0,30]}(-(x^i - x^j)^2 - (y^i - y^j)^2 + \frac{64}{m^2} < 0)$$
$$\wedge \bigwedge_{i=1}^{m} \mathsf{F}_{[0,10]}\big(\mathsf{G}_{[0,10]}((x^i - c_x)^2 + (y^i - c_y)^2 - 25 < 0)$$
$$\wedge\, \mathsf{F}_{[10,20]}(-(x^i - c_x)^2 - (y^i - c_y)^2 + 100 < 0)\big)$$

Figure 5. A trajectory of ATM (left) and the BLTL property to verify (right)

In the second and third experiments, most of the successful runs resulted in valid. These runs became stable (i.e., there are less difference between the continuous trajectories in each step) in the later steps, and thus satisfied the property $\varphi$. We confirmed that most of the inconclusive runs fell into zeno behaviors.

It was quite rare that the result of AnalIntervals became unknown because there were always a few (or no) tight boundaries in $\boldsymbol{T}_\varphi$ and they rarely contained zero.

Next, we experimented with dReal (version 2.14.08), a solver for $\delta$-weakened SMT problems, for comparison. We consider a portion of the model of (another instance of) the bouncing ball such that the initial state is $(1.1, 0, 0)$ and the trajectories of the ball and the table become close to tangent (Figure 4 (a)). Next, we analyzed this model by simulating the underlying $HA$ with our method and by solving the SMT problem with dReal, respectively. Figure 4 (b) and (c) show the computed witness trajectories. Our implementation verified the occurrence of the first contact with the floor. dReal computed two enclosures for the possible trajectories of the model; they seemed corresponding to the first and last intersections with the guard. The second witness seemed wrong because the guard condition became $\delta$-sat around $t' = 0.5$ with $\delta = 0.001$. The computation of a boundary can be troublesome in this way, without the unique existence verification process.

### 7.3 *Air Traffic Maneuver*

We performed a verification of a simplified model of an air traffic maneuver (ATM) [22] in which the number of aircrafts was parameterized. A trajectory of $(x^i, y^i)$ when $m = 4$ is illustrated in Figure 5. We verified the following property shown in Figure 5. The first line describes that the distance between each pair of aircrafts is larger than the threshold $8/m$ during $\|\varphi\| = 30$ time units. The following of the property describes that all aircrafts reach within the circle with radius 5 within the time interval $[0, 10]$, stay there at least 10 time units, and reach outside the circle with radius 10 after another 10 time units. We verified 10 times for each instance with $m = 2, 4, 6, 8$. In each verification, we randomly picked a point initial value. All runs resulted in valid. The specification of the instances and the results are shown in Table 2. The columns "# vars" and "# APs" represent the numbers

Table 2
Experimental results (ATM)

| $m$ | # vars | # APs | time | $m$ | # vars | # APs | time |
|---|---|---|---|---|---|---|---|
| 2 | 10 | 5 | 0.5s | 6 | 26 | 27 | 28s |
| 4 | 18 | 14 | 5.5s | 8 | 34 | 44 | 98s |

of variables in $HA$ and APs in the property. The average timings rose exponentially as $m$ increased.

## 8 Conclusions

We have presented a sound BLTL validation method that assures that all initialized trajectories satisfy the property. The proposed method is able to detect a witness trajectory that is verified its unique existence with an interval-based ODE integration and an interval Newton method. We consider the experimental results are promising for the practical use.

In future work, our method and implementation should be improved to allow large and uncertain initial values. Examples in a realistic setting should be demonstrated with the implementation.

## Acknowledgment

This work was partially funded by JSPS (KAKENHI 25880008).

## References

[1] Alur, R., C. Courcoubetis, N. Halbwachsc, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis and S. Yovine, *The algorithmic analysis of hybrid systems*, Theoretical Computer Science **138** (1995), pp. 3–34.

[2] Alur, R., T. Feder and T. A. Henzinger, *The Benefits of Relaxing Punctuality*, Journal of the ACM **43** (1996), pp. 116–146.

[3] Chen, X., E. Abraham and S. Sankaranarayanan, *Taylor Model Flowpipe Construction for Non-linear Hybrid Systems*, in: *IEEE Real-Time Systems Symposium*, 2012.

[4] Cimatti, A., A. Griggio, S. Mover and S. Tonetta, *Verifying LTL Properties of Hybrid Systems with K-LIVENESS*, in: *CAV*, *LNCS* 8559, 2014, pp. 424–440.

[5] Collins, P. and A. Goldsztejn, *The Reach-and-Evolve Algorithm for Reachability Analysis of Nonlinear Dynamical Systems*, Electronic Notes in Theoretical Computer Science **223** (2008), pp. 87–102.

[6] David, A., D. Du, K. G. Larsen, A. Legay, M. Mikučionis, D. B. g. Poulsen and S. Sedwards, *Statistical Model Checking for Stochastic Hybrid Systems*, Electronic Proceedings in Theoretical Computer Science **92** (2012), pp. 122–136.

[7] Donzé, A. and O. Maler, *Robust Satisfaction of Temporal Logic over Real-Valued Signals*, in: *FORMATS*, *LNCS* 6246, 2010, pp. 92–106.

[8] Eggers, A., M. Franzle and C. Herde, *SAT Modulo ODE : A Direct SAT Approach to Hybrid Systems*, in: *ATVA*, *LNCS* 5311, 2008, pp. 171–185.

[9] Fainekos, G., A. Girard and G. Pappas, *Temporal logic verification using simulation*, in: *FORMATS*, *LNCS* 4202, 2006, pp. 171–186.

[10] Gao, S., J. Avigad and E. M. Clarke, *Delta-Decidability over the Reals*, in: *LICS*, 2012, pp. 305–314.

[11] Gao, S. and E. M. Clarke, *Satisfiability Modulo ODEs*, in: *FMCAD*, 2013, pp. 105–112.

[12] Gao, S., S. Kong, W. Chen and E. M. Clarke, *$\delta$ -Complete Analysis for Bounded Reachability of Hybrid Systems* (2014).

[13] Goldsztejn, A. and L. Jaulin, *Inner approximation of the range of vector-valued functions*, Reliable Computing **14** (2010), pp. 1–23.

[14] Goubault, E., O. Mullier and M. Kieffer, *Inner Approximated Reachability Analysis*, in: *HSCC*, 2014, pp. 163–172.

[15] Ishii, D., 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 (STTT) **13** (2011), pp. 449–461.

[16] Maler, O. and D. Nickovic, *Monitoring Temporal Properties of Continuous Signals*, in: *FORMATS*, *LNCS* 3253, 2003, pp. 152–166.

[17] Moore, R. E., “Interval Analysis,” Prentice-Hall, 1966.

[18] Nedialkov, N. S., *VNODE-LP — A Validated Solver for Initial Value Problems in Ordinary Differential Equations*, Technical report, McMaster University (2006).

[19] Neumaier, A., “Interval Methods for Systems of Equations,” Cambridge University Press, 1990.

[20] Nghiem, T., S. Sankaranarayanan, G. Fainekos, F. Ivancic, A. Gupta and G. J. Pappas, *Monte-Carlo Techniques for Falsification of Temporal Properties of Non-Linear Hybrid Systems*, in: *HSCC*, 2010, pp. 211–220.

[21] Plaku, E., L. E. Kavraki and M. Y. Vardi, *Falsification of LTL Safety Properties in Hybrid Systems*, in: *TACAS*, *LNCS* 5505, 2009, pp. 368–382.

[22] Platzer, A. and E. M. Clarke, *Formal Verification of Curved Flight Collision Avoidance Maneuvers : A Case Study*, in: *FM*, *LNCS* 5850, 2009, pp. 547–562.

[23] Podelski, A. and S. Wagner, *Model Checking of Hybrid Systems : From Reachability towards Stability*, in: *HSCC*, *LNCS* 3927, 2006, pp. 507–521.

[24] Ramdani, N. and N. S. Nedialkov, *Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques*, Nonlinear Analysis: Hybrid Systems **5** (2011), pp. 149–162.

[25] Shultz, B. and B. J. Kuipers, *Proving properties of continuous systems : qualitative simulation and temporal logic*, Artificial Intelligence **92** (1997), pp. 91–129.

[26] Wang, Q., P. Zuliani, S. Kong, S. Gao and E. Clarke, *SReach : Combining Statistical Tests and Bounded Model Checking for Nonlinear Hybrid Systems with Parametric Uncertainty*, Technical report, Carnegie Mellon University (2014).

[27] Zuliani, P., A. Platzer and E. M. Clarke, *Bayesian statistical model checking with application to Stateflow/Simulink verification*, Formal Methods in System Design **43** (2013), pp. 338–367.