Interval-based Solving of Hybrid Constraint Systems ⋆

Daisuke Ishii * **Kazunori Ueda** *,** **Hiroshi Hosobe** **
Alexandre Goldsztejn ***

* *Dept. of Computer Science, Waseda University, 3-4-1, Okubo, Shinjuku-ku, Tokyo 169-8555, Japan (e-mail: {ishii, ueda}@ueda.info.waseda.ac.jp).*
** *National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan (e-mail: hosobe@nii.ac.jp)*
*** *LINA, Université de Nantes, 2, Rue de la Houssinnière, BP 92208, F-44322 Nantes Cedex 3, France (e-mail: alexandre.goldsztejn@univ-nantes.fr)*

Abstract: An approach to reliable modeling, simulation and verification of hybrid systems is interval arithmetic, which guarantees that a set of intervals narrower than specified size encloses the solution. Interval-based computation of hybrid systems is often difficult, especially when the systems are described by nonlinear ordinary differential equations (ODEs) and nonlinear algebraic equations. We formulate the problem of detecting a discrete change in hybrid systems as a *hybrid constraint system* (HCS), consisting of a *flow constraint* on trajectories (i.e. continuous functions over time) and a *guard constraint* on states causing discrete changes. We also propose a technique for solving HCSs by coordinating (i) interval-based solving of nonlinear ODEs, and (ii) a constraint programming technique for reducing interval enclosures of solutions. The proposed technique reliably solves HCSs with nonlinear constraints. Our technique employs the interval Newton method to accelerate the reduction of interval enclosures, while guaranteeing that the enclosure contains a solution.

Keywords: Hybrid systems, interval arithmetic, constraint programming

1. INTRODUCTION

Detection of states causing discrete changes in continuously evolving hybrid systems plays a significant role in simulation and verification. The problem is described by the conjunction of an ordinary differential equation (ODE) and a condition for a discrete change (guard condition). Many techniques for the problem (e.g. Esposito and Kumar (2007)) perform numerical computation. Since the techniques may compute unexpected results due to rounding errors, various workarounds have been investigated; for example, Park and Barton (1996) handled the discontinuity sticking problem, that is, the problem of detecting the same discrete event right after a discrete change.

Interval arithmetic (Moore et al. (2009)) provides rigor in numerical computation. Computation in interval arithmetic produces *over-approximation* of continuous states which is a set of intervals or boxes enclosing the theoretical solutions and the accumulation of round-off errors. Interval arithmetic has been applied to the simulation (Nedialkov and von Mohrenschildt (2002)) and verification (Henzinger et al. (2000); Ratschan and She (2007)) of hybrid systems.

Hybrid systems are modeled by *constraints*, i.e. equations on real numbers and functions, and intervals (inequalities) for uncertain parameters such as initial values (Hickey and Wittenberg (2004)). The reliable simulation and verification are done by integrating continuous dynamics and discrete changes, together with handling uncertainties and computation errors. It is not obvious to reliably compute hybrid systems described by nonlinear ODEs and nonlinear conditions for discrete changes. This paper proposes a framework for such nonlinear hybrid systems.

- We propose *hybrid constraint systems* (HCSs) to describe the problem of detecting discrete changes by constraints (Section 4). An HCS consists of a *flow constraint* on trajectories and a *guard constraint* on continuous states. Solving the HCS is computation of *box-consistent* domains, which means enclosing states with intervals that satisfy every constraint. We see how an HCS serves as a key component in the simulation of hybrid systems (Section 6.1).
- We develop a technique for solving HCSs by integrating an interval-based method for nonlinear ODEs by Nedialkov et al. (1999), and an interval-based constraint programming framework by van Hentenryck et al. (1997)(Section 5). The technique generates a set of boxes smaller than a specified size that encloses the theoretical solution.
- The proposed technique employs the interval Newton method for the quadratic convergence of the reduction of boxes, and to guarantee that the boxes contain a solution (Section 5.3). The method uses an interval Newton operator derived from flow and guard con-

⋆ This research is partially supported by JSPS, Grant-in-Aid for Young Scientists (B) 20700033.

straints. Experimental results show the efficiency of the method in simulating nonlinear hybrid systems (Section 6).

2. RELATED WORK

This work extends a technique by Ishii et al. (2008). This earlier work was limited in formalizing the problem of detecting discrete changes as a constraint system. The efficiency of the method was also limited since pruning on a time domain was based on a binary search technique.

Park and Barton (1996) introduced the interval Newton method to the detection process to guarantee the existence of an event within a time interval. However, the guarantee is based on interpolation polynomials converted from the original problems. Nedialkov and von Mohrenschildt (2002) proposed an interval-based method for simulating hybrid systems. The method is limited in efficiency and handling of nonlinear guard conditions.

Cruz (2005), Lin and Stadtherr (2007), and others are applying constraint programming techniques to interval-based solving of ODEs. Their frameworks allow the use of parameters in ODEs as well as the addition of various constraints such as value restriction constraints. Although the frameworks do not handle constraints equivalent to the guard constraints in this paper, the frameworks might be integrated with ours.

3. PRELIMINARIES

This section introduces concepts for describing the technique we propose.

3.1 Interval Arithmetic

A set of machine-representable floating-point numbers is denoted by $\boldsymbol{F}$. A (bounded) *interval* $I = [l, u]$ $(l, u \in \boldsymbol{F})$ is a set of real numbers, where

$$I = \{r \in \boldsymbol{R} \mid l \leq r \leq u\}.$$

$\boldsymbol{I}$ denotes a set of intervals. A *box* B is a tuple of n intervals $(I_1, \ldots, I_n)$. $\boldsymbol{I}^n$ denotes a set of boxes. For an interval I, $\mathrm{lb}(I)$ denotes the lower bound, $\mathrm{ub}(I)$ denotes the upper bound, $\mathrm{w}(I)$ denotes the width, $\mathrm{int}(I)$ denotes the internal of I, and $\mathrm{m}(I)$ denotes the center of I. For $r \in \boldsymbol{R}$, $[r]$ denotes the narrowest interval containing r.

For $f : \boldsymbol{R}^m \to \boldsymbol{R}^n$, $F : \boldsymbol{I}^m \to \boldsymbol{I}^n$ is called an f's *interval extension* iff it satisfies the following condition (F_i denotes the i-th component of the value of F)

$$\forall I_1 \in \boldsymbol{I} \cdots \forall I_m \in \boldsymbol{I}\, \forall r_1 \in I_1 \cdots \forall r_m \in I_m\, \forall i \in \{1, \ldots, n\}$$
$$(f_i(r_1, \ldots, r_m) \in F_i(I_1, \ldots, I_m)).$$

For $I_1, \ldots, I_m \in \boldsymbol{I}$ and an interval extension F of f, a box $F(I_1, \ldots, I_m)$ is called an *interval enclosure* of possible values of f over $I_1, \ldots, I_m$. For a bounded set $R \subset \boldsymbol{R}$, $\square R$ denotes the smallest interval $I \in \boldsymbol{I}$ that encloses R. For a constraint $c \subseteq \boldsymbol{R}^n$, $C \subseteq \boldsymbol{I}^n$ is an interval extension of c iff it satisfies the following condition

$$\forall I_1 \in \boldsymbol{I} \cdots \forall I_n \in \boldsymbol{I}$$
$$(\forall r_1 \in I_1 \cdots \forall r_n \in I_n\, ((r_1, \ldots, r_n) \in c) \Rightarrow (I_1, \ldots, I_n) \in C).$$

3.2 Interval Newton Method

Given an equation $h(t) = 0$, where $h : \boldsymbol{R} \to \boldsymbol{R}$ is a continuously differentiable function, a solution of the equation in an interval T is also included in an interval obtained by the following interval operator

$$N_{H,\dot{H}}(T) = T \cap \left([\mathrm{m}(T)] - \frac{H([\mathrm{m}(T)])}{\dot{H}(T)} \right),$$

where H and $\dot{H}$ are interval extensions of h and its derivative. $N_{H,\dot{H}}(T)$ is defined iff $0 \notin \dot{H}(T)$ holds. The (uni-variate) *interval Newton method* iteratively refines an interval enclosure by the operator above. By taking a sufficiently small enclosure T of a solution, iterated applications of $N_{H,\dot{H}}(T)$ will converge. The fixpoint is denoted by $N^*_{H,\dot{H}}(T)$. If the condition $N_{H,\dot{H}}(T) \subseteq \mathrm{int}(T)$ holds, a unique solution $t^* \in N_{H,\dot{H}}(T)$ exists (see Theorem 8.4 in Moore et al. (2009)).

3.3 Interval-based Solving of ODEs

Let y denotes a vector-valued continuous function over time $\boldsymbol{R} \to \boldsymbol{R}^n$ called *trajectory*. An *initial value problem for an ODE* (IVP-ODE) is formed by the conjunction of equations

$$\dot{y}(t) = f(t, y(t)) \wedge y(t_0) = y_0,$$

where $t_0 \in \boldsymbol{R}, y_0 \in \boldsymbol{R}^n$ and $f : \boldsymbol{R}^{n+1} \to \boldsymbol{R}^n$ (assuming Lipschitz continuity). Given an IVP-ODE, a *solution* denoted by y_{t_0,y_0} is a trajectory that satisfies the equations.

Given an initial value set $(Y_0, T_0) \in \boldsymbol{I}^{n+1}$, an interval extension of the solution y_{t_0,y_0}, denoted by $Y_{T_0,Y_0} : \boldsymbol{I} \to \boldsymbol{I}^n$, satisfies the following condition

$$\forall t_0 \in T_0\, \forall y_0 \in Y_0\, \forall t \in T\, (y_{t_0,y_0}(t) \in Y_{T_0,Y_0}(T)),$$

where T is a time interval such that $\mathrm{lb}(T) \geq \mathrm{ub}(T_0)$.

We employ an existing method VNODE proposed in Nedialkov et al. (1999) and Nedialkov (2006) for solving IVP-ODEs based on interval arithmetic. Consider an IVP-ODE, an initial value set (T_0, Y_0) and a time interval $T_1 \in \boldsymbol{I}$. We obtain a box $Y_1 = Y_{T_0,Y_0}(T_1)$ using VNODE. As a by-product of the computation, VNODE computes an enclosure $Y_{T_0,Y_0}([\mathrm{lb}(T_0), \mathrm{ub}(T_1)])$, because the computation is done iteratively from the initial value.

In our method, we also need an interval extension of the derivative of the solution $\dot{Y}_{T_0,Y_0} : \boldsymbol{I} \to \boldsymbol{I}^n$. Let $T_1 \in \boldsymbol{I}$ be a time interval. Then $\dot{Y}_{T_0,Y_0}(T_1)$ will be implemented as computation of a function $F(T_1, Y_{T_0,Y_0}(T_1))$, that is, an interval extension of $f(t, y(t))$ in the ODE. Note that $Y_{T_0,Y_0}(T_1)$ should be computed by VNODE beforehand.

3.4 Consistency Technique for Continuous Constraints

Van Hentenryck et al.(1997) proposed a method for solving a set of continuous constraints $\mathcal{C}$ with a tuple of n variables $\mathcal{X} = (x_1, \ldots, x_n)$ and a domain $\mathcal{D} = (D_1, \ldots, D_n)$. See Benhamou and Granvilliers (2006) for an introduction. $\mathcal{C}$ is a conjunction of constraints in the form of $g(\mathcal{X}) \bullet 0$, where $\bullet$ is a relation e.g. $\leq$ and $=$, and g is a continuous function $\boldsymbol{R}^n \to \boldsymbol{R}$ built with elementary arithmetic operations. The method refines a given domain $\mathcal{D}$ by the BranchAndPrune algorithm and computes a set of boxes smaller

Input: set of constraints $\mathcal{C}$, initial domain $\mathcal{D}$, error tolerance ϵ
Output: set of consistent domains $\{\mathcal{D}_i\}_{i\in\{1,\ldots,n\}}$

```
1:  D' := Prune(C, D)
2:  if D' ≠ ∅ then
3:    if the precision of D' is under ε then
4:      return {D'}
5:    else
6:      i := select a component
7:      (D1, D2) := Branch(D', i)
8:      return BranchAndPrune(C, D1) ∪
          BranchAndPrune(C, D2)
9:    end if
10: else
11:   return ∅
12: end if
```

Fig. 1. BranchAndPrune algorithm.

than a given error tolerance ϵ, so that the union of boxes is a *box-consistent* domain (see Section 4.1) for the constraint system. Figure 1 shows the BranchAndPrune algorithm. The algorithm recursively alternates reducing and branching of a domain in a problem. In the algorithm, two procedures are left open as Prune and Branch. Algorithms such as HC4, BC5 and 3B are known as implementations for Prune. Branch is usually implemented as partitioning of a domain into several boxes along a dimension of the domain.

4. HYBRID CONSTRAINT SYSTEMS

In this section, we define HCSs. We consider continuous states as $(n+1)$ dimensional real vectors over time and space. An HCS describes a crossing point of a trajectory over time and a time-invariant boundary in the state space. Figure 2 illustrates an HCS.

Consider a tuple of variables $\mathcal{X} = (x_0, x_1, \ldots, x_n)$ consisting of a variable x_0 representing the time at a crossing point and n variables $x_1, \ldots, x_n$ representing a state at the time x_0. We then define the following two kinds of constraints.

- A *flow constraint* $\mathcal{C}_f$ is a constraint described by the conjunction of the following equations
$$(\dot{y}(t) = f(t, y(t)) \wedge y(t_0) = y_0) \\ \wedge\ y(x_0) = (x_1, \ldots, x_n) \wedge x_0 > t_0,$$
where the first part is an IVP-ODE representing an n-dimensional trajectory y_{t_0,y_0} ($t_0 \in \boldsymbol{R}, y_0 \in \boldsymbol{R}^n$, and $f : \boldsymbol{R}^{n+1} \to \boldsymbol{R}^n$ is a Lipschitz continuous function).
- A *guard constraint* $\mathcal{C}_g$ is a constraint described by
$$g(x_1, \ldots, x_n) = 0,$$
where g is a differentiable function $\boldsymbol{R}^n \to \boldsymbol{R}$.

Possible values for the variables belong to a domain $\mathcal{D} = (D_0, D_1, \ldots, D_n) \in \boldsymbol{I}^{n+1}$, where each component is associated with a variable in $\mathcal{X}$. Values expressing parameters t_0 and y_0 in the initial condition in $\mathcal{C}_f$ are given as an initial value set $\mathcal{D}_0 = (D_{0,0}, \ldots, D_{n,0}) \in \boldsymbol{I}^{n+1}$. A *hybrid constraint system* is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{D}_0, \mathcal{C}_f, \mathcal{C}_g)$ with a tuple $\mathcal{X}$ of variables, a domain $\mathcal{D}$, an initial value set $\mathcal{D}_0$, a flow constraint $\mathcal{C}_f$ and a guard constraint $\mathcal{C}_g$.

For example, a ball that bounces off a sinusoidal surface is modeled as an HCS $(\mathcal{X}, \mathcal{D}, \mathcal{D}_0, \mathcal{C}_f, \mathcal{C}_g)$, where
$$\mathcal{X} = (x_0, x_1, x_2, x_3, x_4),$$
$$\mathcal{D} = (\boldsymbol{R}^{\geq 0}, \boldsymbol{R}, \boldsymbol{R}, \boldsymbol{R}, \boldsymbol{R}),\ \mathcal{D}_0 = ([0], [1], [3], [6], [-2]),$$
$$\mathcal{C}_f \equiv (\dot{y}(t) = (\dot{y}_1(t), \dot{y}_2(t), \dot{y}_3(t), \dot{y}_4(t)) \\ = (y_3(t), y_4(t), 0, -g - k \cdot y_4(t)) \wedge y(t_0) = y_0) \\ \wedge\ y(x_0) = (x_1, x_2, x_3, x_4) \wedge x_0 > t_0,$$
$$\mathcal{C}_g \equiv \sin(2 \cdot x_1) - x_2 = 0.$$
x_0 is a variable representing time, x_1, x_2, x_3 and x_4 are variables representing the position (x_1, x_2) and velocity (x_3, x_4) of the ball. Movement of the ball is described by $\mathcal{C}_f$ and a contact with the surface is detected by checking whether $\mathcal{C}_g$ is entailed. The initial condition in $\mathcal{C}_f$ is interpreted as $\forall t_0 \in [0]\, \forall y_0 \in [1] \times [3] \times [6] \times [-2]\, (y(t_0) = y_0)$. Figure 2 (a) illustrates the trajectory of the ball with parameters set to $g = 9.8$ and $k = 0.3$.

Let $(\mathcal{X}, \mathcal{D}, \mathcal{D}_0, \mathcal{C}_f, \mathcal{C}_g)$ is an HCS, a *valuation* is a map $\mathcal{X} \to \mathcal{D}$ from every variable $x_i \in \mathcal{X}$ to a value $d_i \in D_i$ ($i \in \{0, \ldots, n\}$). A *solution* of the HCS is a valuation satisfying constraints $\mathcal{C}_f$ and $\mathcal{C}_g$. In general, an HCS may have multiple solutions. When applying HCSs to the simulation of hybrid systems, the one we are interested in is the *earliest* solution. In Figure 2 (a), the HCS has three solutions inside each of the boxes $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{D}_3$. The ball bounces at the earliest solution in $\mathcal{D}_1$.

4.1 Box Consistency for HCSs

In this paper, interval-based solving of an HCS means refining an initial domain to a *box-consistent* (van Hentenryck et al. (1997)) domain.

We modify the definition of box consistency for HCSs as follows. Let y be a solution trajectory of a flow constraint and g be a function describing a guard constraint. Consider interval extensions G of g and Y_{T_0,Y_0} of y_{t_0,y_0}, where T_0 is $D_{0,0}$ and Y_0 is $(D_{1,0}, \ldots, D_{n,0})$. For an index $i \in \{0, \ldots, n\}$, the i-th component D_i of a domain $\mathcal{D}$ is *box-consistent* with respect to the other components iff
$$D_0 = \square\{d_0 \in D_0 \mid 0 \in G(Y_{T_0,Y_0}([d_0] \pm [0, h_{min}]))\}$$
(for $i = 0$), where $h_{min} \in \boldsymbol{R}^+$ and $h_{min} \geq \mathrm{w}(T_0)$, and
$$D_i = \square\{d_i \in D_i \mid d_i \in Y_{T_0,Y_0,i}(D_0) \wedge \\ 0 \in G(D_1, \ldots, D_{i-1}, [d_i], D_{i+1}, \ldots, D_n)\}$$
(for $i \in \{1, \ldots, n\}$). An HCS is box-consistent iff all the components are box-consistent. h_{min} in the condition above is used because it is difficult to compute $Y_{T_0,Y_0}([d_0])$ by VNODE.

Consider a set of floating-point numbers, $\boldsymbol{F} = \{n \cdot 10^{-2} \mid n \in \boldsymbol{Z}\}$. For the bouncing ball example, a box $\mathcal{D}_1 = ([0.44, 0.54], [3.86, 3.96], [0.95, 1.05], [5.95, 6.05], [-6.1, -6.2])$ is appropriate for the box-consistent domain.

5. TECHNIQUE FOR SOLVING AN HCS

In this section, a technique for computing the box-consistent domain of an HCS is described. The technique applies the BranchAndPrune algorithm in Section 3.4 to HCSs with an implementation of Prune called $\textsc{Prune}_{\mathrm{HCS}}$. The proposed method computes a set of boxes that encloses all the solutions in the initial domain. The

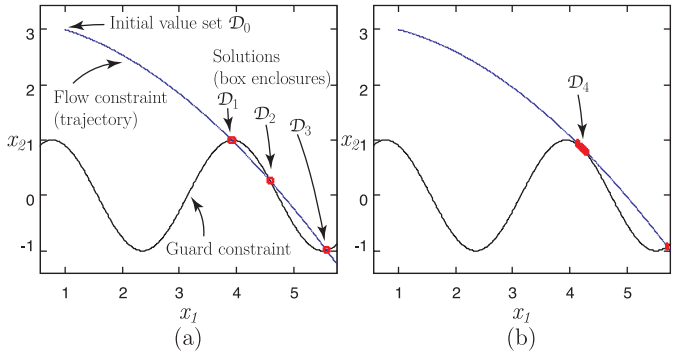

Fig. 2. Examples of an HCS.

Input: flow constraint $\mathcal{C}_f \equiv (\dot{y}(t) = f(t, y(t)) \wedge y(t_0) = y_0) \wedge (x_1, \ldots, x_n) = y(x_0) \wedge x_0 > t_0$,
guard constraint $\mathcal{C}_g \equiv g(x_1, \ldots, x_n) = 0$,
initial domain $\mathcal{D} = (D_0, D_1, \ldots, D_n) \in \boldsymbol{I}^{n+1}$,
initial value set $\mathcal{D}_0$
Output: box-consistent domain $\mathcal{D}' = (D'_0, \ldots, D'_n) \in \boldsymbol{I}^{n+1}$

1: **repeat**
2: $\mathcal{D}' := \mathcal{D}$
3: obtain H and $\dot{H}$ from $\mathcal{C}_g$, $\mathcal{C}_f$ and $\mathcal{D}_0$
4: $D_0 := \textsc{NarrowL}(H, \dot{H}, D_0)$
5: $D_0 := \textsc{NarrowU}(H, \dot{H}, D_0)$
6: $(D_1, \ldots, D_n) := (D_1, \ldots, D_n) \cap Y_{T_0,Y_0}(D_0)$
7: $(D_1, \ldots, D_n) := \textsc{Narrow}_{\textsc{CCS}}(\mathcal{C}_g, (D_1, \ldots, D_n))$
8: **until** $\mathcal{D} = \mathcal{D}'$
9: return $\mathcal{D}'$

Fig. 3. $\textsc{Prune}_{\textsc{HCS}}$ algorithm.

accuracy of results are specified by the real-valued parameters ϵ and h_{min}. ϵ used in the BranchAndPrune algorithm determines the maximum width of intervals in a result. A solution exists within h_{min} from each bound of the time domain.

Figure 3 shows the $\textsc{Prune}_{\textsc{HCS}}$ algorithm. $\textsc{Prune}_{\textsc{HCS}}$ reduces a domain of an HCS by enforcing the box consistency. $\textsc{Prune}_{\textsc{HCS}}$ takes a flow constraint $\mathcal{C}_f$, a guard constraint $\mathcal{C}_g$, a domain $\mathcal{D}$, and an initial value set $\mathcal{D}_0$ as input. $\textsc{Prune}_{\textsc{HCS}}$ iteratively prunes each component of a domain until the fixpoint is reached (lines 1–8).

At lines 3–5, the domain D_0 of the time variable x_0 is reduced, and then at lines 6–7, the domain $(D_1, \ldots, D_n)$ is reduced. Details on each of the reduction methods are supplied in Sections 5.1 and 5.2. The algorithm returns a pruned domain $\mathcal{D}'$ at line 9.

5.1 Reduction of the Time Domain

$\textsc{Prune}_{\textsc{HCS}}$ reduces a time domain efficiently by applying the interval Newton method that solves flow and guard constraints simultaneously. At line 3 in Figure 3, the algorithm constructs functions H and $\dot{H}$ for the interval Newton method. The essential idea is that the numerical solution of the IVP-ODE, denoted hereafter as Y_{T_0,Y_0}, is used to construct an interval Newton operator. Y_{T_0,Y_0} is obtained by iterative calculation with respect to a flow constraint $\mathcal{C}_f$ and an initial value set $\mathcal{D}_0$, as described in Section 3.3. Accordingly, H and $\dot{H}$ are given by

$$H(T) = G(Y_{T_0,Y_0}(T)),\ \dot{H}(T) = \sum_{i=1}^{n} \left(\frac{\partial G}{\partial X_i} \cdot \dot{Y}_{T_0,Y_0,i}(T) \right),$$

where $Y_{T_0,Y_0}, G, \dot{Y}_{T_0,Y_0}$ and $\partial G/\partial X_i$ are interval extensions of a trajectory y_{t_0,y_0}, a function g in a guard constraint $\mathcal{C}_g$, and their derivatives, respectively. X_i is a variable of G ($i \in \{1, \ldots, n\}$). To compute $\partial G/\partial X_i$, we apply automatic differentiation to G.

At lines 4 and 5 of $\textsc{Prune}_{\textsc{HCS}}$, procedures NarrowL and NarrowU reduce the lower and upper edges of the domain. This procedure is an adaptation to HCS of the algorithm shown in van Hentenryck et al. (1997). The NarrowL procedure is as follows (NarrowU is similar except that it operates on the upper edge instead of the lower edge).

(1) First, the guard constraint is checked for the current time domain D_0. If $0 \notin H(D_0)$ holds, return $\emptyset$ and terminate.
(2) Calculate the fixpoint of the interval Newton method $D'_0 = N^*_{H,\dot{H}}(D_0)$. To obtain $N^*_{H,\dot{H}}(D_0)$, $D'_0 = N_{H,\dot{H}}(D_0)$ is repeatedly computed until the ratio of D'_0 to D_0 is under a threshold.
(3) Let h_{min} is a minimal step width for ODE solving, and consider the box $L = [\mathrm{lb}(D'_0), \mathrm{lb}(D'_0) + h_{min}]$. If the interval $H(L)$ contains 0, return an interval $[\mathrm{lb}(D'_0), \mathrm{ub}(D_0)]$, and terminate.
(4) Split D_0 in two and apply NarrowL recursively for each interval. Find the smallest bound l in the results, return an interval $[l, \mathrm{ub}(D_0)]$, and terminate.

5.2 Reduction of the Continuous State Domain

Once the narrowing operators reduce a time interval D_0, VNODE computes a box $Y_{T_0,Y_0}(D_0)$ enclosing the trajectories over D_0 as described in Section 3.3 (line 6 of Figure 3). Then we can build a continuous constraint system described in Section 3.4 consisting of the guard constraint $\mathcal{C}_g \equiv g(x_1, \ldots, x_n) = 0$ and the initial domain $(D_1, \ldots, D_n) = Y_{T_0,Y_0}(D_0)$. At line 7 of $\textsc{Prune}_{\textsc{HCS}}$, the $\textsc{Narrow}_{\textsc{CCS}}$ procedure reduces the domain $(D_1, \ldots, D_n)$ by applying one of the narrowing operators shown in Section 3.4.

5.3 Testing the Unique Existence of a Solution

In general, the number of solutions in boxes computed by BranchAndPrune is unknown. Using the interval Newton method, the uniqueness and existence of a solution within a box is determined under a certain condition.

Theorem 1. Consider an HCS $(\mathcal{X}, \mathcal{D}, \mathcal{D}_0, \mathcal{C}_f, \mathcal{C}_g)$, where $\mathcal{D} = (D_0, \ldots, D_n)$. Suppose a call to the procedure $\textsc{Prune}_{\textsc{HCS}}(\mathcal{C}_f, \mathcal{C}_g, \mathcal{D}, \mathcal{D}_0)$ returns $\mathcal{D}' = (D'_0, \ldots, D'_n)$. In the computation, suppose the procedure NarrowL or NarrowU reduces a time interval D''_0 to $N_{H,\dot{H}}(D''_0)$, and the following conditions hold. Then, a solution of the HCS *uniquely* exists in $\mathcal{D}'$.

- A unique trajectory y_{t_0,y_0} exists over D_0 with respect to $\mathcal{C}_f$ and $\mathcal{D}_0$,
- g in $\mathcal{C}_g$ is continuously differentiable over $(D_1, \ldots, D_n)$,
- $N_{H,\dot{H}}(D''_0) \subseteq \mathrm{int}(D''_0)$,
- $D'_0 \subseteq D''_0$.

Proof. From the first and second conditions, a function $g \circ y_{t_0,y_0}$ exists, and is continuously differentiable over D_0. Since $G \circ Y_{T_0,Y_0}$ composed by PRUNE$_{\text{HCS}}$ is an interval extension of $g \circ y_{t_0,y_0}$, the interval Newton method proves that a root of $g \circ y_{t_0,y_0}$ uniquely exists in D_0''. The third condition is for the interval Newton method. The last condition proves the unique existence of the solution in $\mathcal{D}'$ because the computation by PRUNE$_{\text{HCS}}$ completely encloses the solution. □

VNODE validates the first condition for the computed enclosure. The third condition will be tested in the narrowing procedures.

For example, the box-consistent domain for the bouncing ball shown in the previous section encloses a unique solution. Consider a bouncing ball with an initial value set $\mathcal{D}_0 = ([0], [1], [3], [6], [-1.7, -1.6])$ (see Figure 2 (b)). The box-consistent domain $\mathcal{D}_4$ indicated in the figure is not guaranteed to have a solution.

5.4 Computing an Enclosure for the Earliest Solution

The union of boxes computed by the proposed technique may enclose multiple solutions. Boxes enclosing the earliest solution are selected as follows.

(1) Compute the clusters of boxes by concatenating two boxes if they are adjacent.
(2) Find a cluster containing the earliest time.
(3) If the cluster intersects with the initial value set, then discard this and search for the next earliest cluster.

Note that we assume each of the clusters computed in (1) and the initial value set in (3) enclose a unique solution.

6. EXAMPLES AND EXPERIMENTATION

We implemented the proposed method on top of the `Elisa` system (Granvilliers and Sorin (2005)) which is an implementation of the BRANCHANDPRUNE algorithm. We implemented data structures for representing flow constraints. PRUNE$_{\text{HCS}}$ was implemented by extending the reduction procedure in `Elisa`. `VNODE-LP` (Nedialkov (2006)) was used to solve IVP-ODEs. The implementation caches the results by `VNODE-LP` for reusing. Our implementation consists of about 2000 lines of C++ code.

Section 6.1 shows a simulation of a bouncing ball by modeling each bounce of the ball as an HCS. Section 6.2 reports the results of comparisons with existing methods. The parameters in the proposed method were set as $\epsilon = 10^{-2}$ and $h_{min} = 10^{-11}$. BC5 was used as NARROW$_{\text{CCS}}$. In the experimentation, we modified the implementation to terminate the computation after an enclosure for the earliest solution was obtained. We experimented on a 2.4GHz Intel Core 2 Duo processor with 2GB of RAM.

6.1 Computation of a Bouncing Ball

Consider the bouncing ball described in Section 4. Here, we change the initial value set to $\mathcal{D}_0 = ([0], [2], [5], [0], [-5])$, and the guard constraint to $\mathcal{C}_g \equiv \sin(x_1) - x_2 = 0$. Figure 4 illustrates the boxes enclosing a trajectory of the ball bouncing off the surface three times. Those boxes

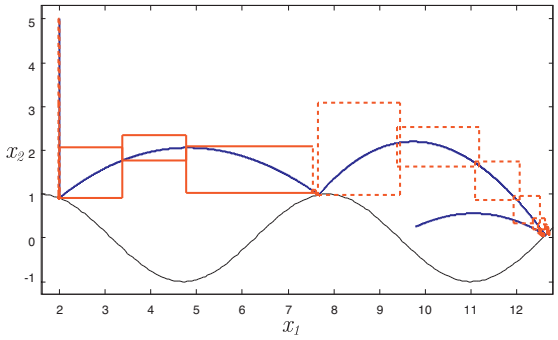

Fig. 4. A trajectory of a bouncing ball on a sinusoidal surface and an interval enclosure of the trajectory.

were computed while solving three HCSs, each of which corresponded to a parabolic motion of the ball. From the result of solving an HCS, a bounce of the ball with a coefficient of restitution of 0.8 was computed by interval arithmetic to set up the initial values for the next phase.

We confirmed that the interval D_0 of the domain was quadratically reduced by the interval Newton operator. For example, in the solving of the first bounce, we had

$$D_0^0 = [0.5366024006905468, 0.6462371408659776],$$
$$D_0^1 = [0.5655469230670141, 0.5675738062165443],$$
$$D_0^2 = [0.5663629650559695, 0.5663632653947441],$$
$$D_0^3 = [0.5663631007048800, 0.5663631007048839].$$

The reductions provide the guarantee of the existence and uniqueness of a solution within the domain.

The left half of Table 1 shows computation results. Each row corresponds to a result of solving an HCS. Each column shows the number of bounces, the time interval, the number of calling PRUNE$_{\text{HCS}}$, the number of calling NARROWL and NARROWU, and the execution time, in milliseconds, of a solving. The width of the first result was 10^{-11}, according to the value of h_{min}. The following results widened as the initial value set widened.

6.2 Comparison with Existing Methods

To evaluate the computational efficiency and to confirm the accuracy of the results, we solved several HCS problems with several solvers including the proposed method.

The following problems were solved: (1, 2) the first and second bounces of the bouncing ball; (3) the problem of detecting the intersection of a trajectory following the Van der Pol equation

$$\dot{y}_1 = y_2,\ \dot{y}_2 = 10 \cdot (1 - y_1^2) \cdot y_2 - y_1,\ y_1(0) = 1,\ y_2(0) = 0,$$

and the ellipse $\mathcal{C}_g \equiv x_1^2/9 + x_2^2/255 = 1$ (we abbreviate $y_i(t)$ with y_i); (4) detection of the intersection of a trajectory following the Lorenz equation

$$\dot{y}_1 = 10 \cdot (y_2 - y_1),\ \dot{y}_2 = y_1 \cdot (28 - y_3) - y_2,$$
$$\dot{y}_3 = y_1 \cdot y_2 - 8/3 \cdot y_3,\ y_1(0) = 15,\ y_2(0) = 15,\ y_3(0) = 36,$$

and the sphere $\mathcal{C}_g \equiv x_1^2 + x_2^2 + (x_3 - 28)^2 = 700$. We solved the above problems with the following methods: (a) the proposed method; (b) the method proposed by Ishii et al. (2008) that employs a binary search technique; (c) the symbolic `DSolve` solver with the `Minimize` function in Mathematica 6.0 (Wolfram Research (2007)); and (d, d')

Table 1. Computation results for the bouncing ball model.

	(a) proposed method				(b) existing method			
n	result (D_0)	branch	reduce	time	result (D_0)	branch	reduce	time
1	0.5663631007[04, 14]	1	2	61	0.566363100[662, 748]	3	132	515
2	1.5193134214[00, 25]	5	27	171	1.51931342[1229, 1508]	5	210	734
3	2.688336307[167, 706]	5	19	125	2.6883363[04386, 10239]	5	200	718

Table 2. Comparison results.

	(a) proposed method				(b) existing method			
problem	result (D_0)	branch	reduce	time	result (D_0)	branch	reduce	time
(3)	10.412056185[3994, 4956]	5	167	525	10.412056185[3614, 4187]	4	370	2022
(4)	10.097265[364782, 415188]	1	408	2146	10.097265[359764, 420194]	3	643	5982

	(c) Mathematica (symbolic)		(d) Mathematica (numeric)		(d') Mathematica (numeric)	
problem	result (t)	time	result (t)	time	result (t)	time
(1)	0.56636310070	78	0.56636309967	4	0.56636310070	15
(2)	unsolvable	–	1.51931341936	3	1.51931342141	12
(3)	unsolvable	–	10.41204598059	5	10.41205618538	266
(4)	unsolvable	–	10.09936465531	14	10.09726539120	1109

the numerical `NDSolve` solver with the `EventLocator` option in Mathematica 6.0. We solved the problem with the default settings in (d), and by setting `WorkingPrecision` to 28 and `MaxSteps` to `Infinity` in (d') [1] .

Table 1 and Table 2 report the computed (interval) values for the time variable x_0 (represented by D_0 and t), profiling results, and execution time in milliseconds. As shown in the results (a) and (b), the proposed technique decreases the number of reductions and outperforms the method (b) in efficiency. In (c), `DSolve` of Mathematica computed a rigorous solution but treated only the problem (1). The results (d) and (d') show that `NDSolve` solved HCSs more efficiently than our method. However, `NDSolve` uses approximation algorithms and cannot ensure the achieved accuracy of a result, whereas our method guarantees the accuracy of a result. Another advantage of our method is that we can give intervals to the initial values and coefficients in constraints. `DSolve` and `NDSolve` do not handle ODEs with uncertain parameters.

7. CONCLUSION AND FUTURE WORK

We have presented HCSs and proposed a technique for solving them. As described in Section 5.1, the technique guarantees the existence and uniqueness of a solution in a domain. This method also helps to find the earliest solutions reliably. We need more experimentation in realistic settings and various optimization of the implementation. HCSs will be extended to have multiple guard constraints either in conjunctive and disjunctive way. We plan to solve such problems by interacting with SAT solvers to compute every combination of constraints.

REFERENCES

Benhamou, F. and Granvilliers, L. (2006). Continuous and interval constraints. *Handbook of Constraint Programming*, 571–604. Elsevier.

Cruz, J. (2005). *Constraint Reasoning for Differential Models*. IOS Press.

Esposito, J. and Kumar, V. (2007). A state event detection algorithm for numerically simulating hybrid systems with model singularities. In *ACM TOMACS*, 17(1), 1–22.

Granvilliers, L. and Sorin, V. (2005). Elisa 1.0.4. http://sourceforge.net/projects/elisa.

Henzinger, T.A., Horowitz, B., Majumdar, R., and Wong-Toi, H. (2000). Beyond HyTech: hybrid systems analysis using interval numerical methods. In *Proc. HSCC*, *LNCS* 1790, 130–144.

Hickey, T.J. and Wittenberg, D.K. (2004). Rigorous modeling of hybrid systems using interval arithmetic constraints. In *Proc. HSCC*, *LNCS* 2993, 402–416.

Ishii, D., Ueda, K., and Hosobe, H. (2008). An interval-based consistency technique for reliable simulation of hybrid systems. In *IPSJ Trans. on Mathematical Modeling and its Applications*, 1(1), 149–159. (in Japanese)

Lin, Y. and Stadtherr, M. (2007). Fault detection in continuous-time systems with uncertain parameters. In *Proc. ACC*, 3216–3221.

Moore, R.E., Kearfott, R.B., and Cloud, M.J. (2009). *Introduction to Interval Analysis*. SIAM.

Nedialkov, N.S. (2006). VNODE-LP: a validated solver for initial value problems in ordinary differential equations. TR CAS-06-06-NN, McMaster University.

Nedialkov, N.S., Jackson, K.R., and Corliss, G.F. (1999). Validated solutions of initial value problems for ordinary differential equations. In *Applied Mathematics and Computation*, 105(1), 21–68.

Nedialkov, N.S. and von Mohrenschildt, M. (2002). Rigorous simulation of hybrid dynamic systems with symbolic and interval methods. In *Proc. ACC*, vol. 1, 140–147.

Park, T. and Barton, P. (1996). State event location in differential-algebraic models. In *ACM TOMACS*, 6(2), 137–165.

Ratschan, S. and She, Z. (2007). Safety verification of hybrid systems by constraint propagation-based abstraction refinement. In *ACM TECS*, 6(1).

van Hentenryck, P., McAllester, D., and Kapur, D. (1997). Solving polynomial systems using a branch and prune approach. In *SIAM Journal on Numerical Analysis*, 34(2), 797–827.

Wolfram Research (2007). Mathematica 6.0. http://www.wolfram.com/products/mathematica.

[1] The corresponding Mathematica notebooks are available at `http://www.ueda.info.waseda.ac.jp/~ishii/pub/adhs2009/`.