An Interval-based Approximation Method for Discrete Changes in Hybrid cc

Daisuke Ishii[†] Kazunori Ueda[†,‡] Hiroshi Hosobe[‡]
† Waseda University ‡ National Institute of Informatics

Dec. 1, 2006

Abstract

Hybrid cc is a constraint-based programming language aimed for simple modeling and rigorous simulation of hybrid systems. The existing implementation of Hybrid cc uses intervals to handle partially specified models. Although the implementation solves nonlinear equations and ordinary differential equations (ODEs) based on interval arithmetic, discrete changes are not properly computed with intervals. We propose a method for enclosing every solution of a hybrid system with a set of boxes. Our method computes continuous state changes by solving ODEs with initial time and values expressed by intervals. Then the method detects and encloses every state that will cause a discrete change. We devise algorithms for pruning and splitting intervals to obtain tight enclosures. The experimental results show the efficiency of the algorithms.

1 Introduction

Hybrid systems are systems consisting of continuous and discrete changes. Figure 1 (a) illustrates a simple hybrid system in which a particle falls down by gravity and bounces off the ground. While the particle's falling is modeled by differential equations, the bouncing is governed by equations describing discrete changes.

Hybrid cc (*Hybrid concurrent constraint programming*) [5], a compositional and declarative language based on constraint programming, has been proposed as a high-level development tool of hybrid systems for simulation, animation and design. Below is a description of the hybrid system in Figure 1 (a) (See Section 2 for the syntax and implementation):

```
y = 10, y' = 0,              // initial conditions
hence {
    cont(y),                 // height is continuous
    if y > 0 then y'' = -10, // free fall
    if y = 0 then
        y' = -0.5 * prev(y') // bounce
}
```

In our experience of modeling a number of hybrid systems using Hybrid cc, however, we have encountered several difficulties. Figure 1 (b) shows a further execution result of the above example. As the distance between the particle and the ground gets closer, the particle moves in an unexpected way[*1]. The problem is due to the computation of discrete changes in the current implementation of Hybrid cc (Section 3). After the particle loses energy and its height stays within ϵ (to tolerate numerical errors), the implementation ignores discrete changes, and as the result the particle goes underground.

The contribution of this paper is to develop a method for obtaining (guaranteed) approximate solutions

[*1] This kind of phenomenon that causes an infinite number of discrete changes in a finite length of time is known as *Zeno behavior*.

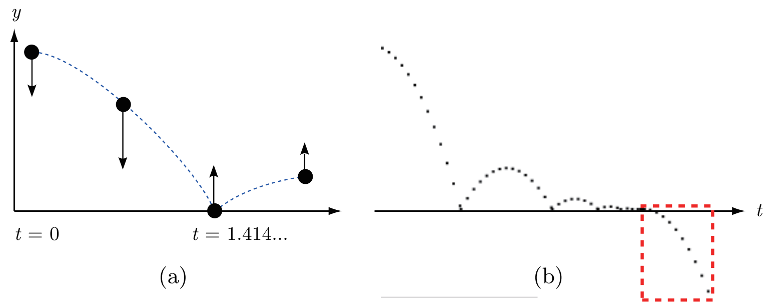

Fig. 1 (a) A bouncing particle and (b) an unexpected result

Table. 1 The basic syntax of Hybrid cc processes

C	Add the constraint C
if C **then** A	If C holds, reduce to A
A, A	Parallel composition
hence A	Execute A at every instant after now
cont(C)	Declare that C is continuous over time
prev(C)	The value of C before entering a phase

of discrete changes in hybrid systems. As we discuss in Section 3, our goal is to bring correct techniques to a number of issues such as reliable simulation, reachability analysis and backward computation in hybrid systems with Hybrid cc. Our approach is based on interval arithmetic (Section 4). The proposed method detailed in Section 5 aims to enclose trajectories of hybrid systems by tight intervals or boxes without losing any solutions. We experimented and evaluated the effect of the method (Section 6).

2 An Overview of Hybrid cc

2.1 The Hybrid cc Language

The formal operational semantics of Hybrid cc is described in [5]. The basic syntax for processes in Hybrid cc is shown in Table 1, where we denote processes by A and constraints by C. Computation in Hybrid cc is performed by processes interacting with the *constraint store*. A process **if** C **then** A triggers discrete changes when an *ask condition* C is entailed by the current constraint store. If C is entailed, the process is reduced to A. Otherwise, the process is ignored. If the entailment of C cannot be determined at the moment, the process is suspended.

2.2 Implementation of Hybrid cc

The implementation of the Hybrid cc interpreter is detailed in [1]. The interpreter handles a Hybrid cc process by alternating *point phases* and *interval phases* as follows:

Step 1 Computation starts from a point phase.

Step 2 In the point phase, the interpreter performs reductions of processes such as the propagation of arithmetic constraints. The computation of the point phase ends at a stable point, where all constraints have been propagated and all possible reductions have been completed.

Step 3 After the point phase, the computation proceeds to an interval phase. Each expression **hence** A, which is passed from Step 2, is reduced to A and processed. All the processes are reduced until a stable point is reached as in Step 2. The resulting constraint store consists of constraints on the derivatives of variables (i.e., ODEs), conditions for discrete changes and constraints to be reduced in the future.

Step 4 Integration of the arithmetic constraints that were told in the previous step is processed until one of the conditions changes its status.

Step 5 If there are no processes left in the constraint store, the computation terminates. Otherwise, go to Step 2.

The arithmetic constraint solvers in the implementation handle nonlinear equations (NLEs) and ordinary differential equations (ODEs). In the implementation, the following approaches are adopted:

- The computation by the solvers is based on interval arithmetic to model uncertainty in the parameters.
- The NLE solver takes an arithmetic constraint which can be rewritten in the implicit form $f(x) = 0$. Solving of the constraint is done by interval pruning of each variable in the constraint. In the implementation, the following four pruning operators are adopted: (1) use of indexicals, (2) splitting of intervals, (3) the Newton-Raphson method, (4) the Simplex method (optional).
- The ODE solver uses the Runge-Kutta-Fehlberg method which numerically integrates with adaptive step-sizes. The solver reduces equations of arbitrary form to an explicit form by propagating equations in each step of integration.
- The ODE solver stops the integration at the *breakpoint* at which any state of constraints in the constraint store is changed. To take care of an overshoot from the breakpoint, the solver backtracks until an exact solution within a constant ϵ is found.

3 The Objective of the Paper

We are developing systems for handling physical simulations and animations based on Hybrid cc. For example, we want to simulate a billiard table on which a number of balls roll and collide with other balls or table edges. The goal is to obtain trajectories of the balls from a description written in Hybrid cc that directly reflects the laws of elementary physics.

However, the current implementation of Hybrid cc has several limitations in the reliability aspect as follows:

- Although the implementation supports interval arithmetic, it does not use interval arithmetic to handle discrete changes such as **if** C **then** A, based on interval arithmetic.
- The ODE solver detects discrete changes during the integration with an ad hoc method. To work around computation errors, the detection allows for a tolerance ϵ, but this may result in qualitatively different trajectories.

The goal of this paper is to propose a reliable method for computing trajectories in Hybrid cc. Our method obtains valid and tight interval enclosures of discrete changes, corresponding to Step 4 in Section 2.2. By combining the proposed method and other known techniques for reliable computing, we will be able to guarantee the accuracy of trajectories, as well as to verify the reachable area of objects. We also expect to apply the method to problems such as backward computation of trajectories.

4 Background of Interval Arithmetic

To describe our method based on interval arithmetic [9], the following notions and definitions are used.

4.1 Basic Notions of Interval Arithmetic

$\mathcal{F}$ denotes the set of machine-representable floating-point numbers, $\mathcal{I}$ denotes the set of intervals over $\mathbb{R}$ whose bounds are in $\mathcal{F}$, and I denotes an interval in $\mathcal{I}$. $\mathrm{lb}(I)$ denotes the lower bound, $\mathrm{ub}(I)$ denotes the upper bound, and $\mathrm{w}(I)$ denotes the width of I. $\mathcal{D}$ denotes the set of boxes over $\mathbb{R}^n$ whose bounds are in $\mathcal{F}$, and D denotes a box in $\mathcal{D}$. Given a real r and a subset A of $\mathbb{R}^n$, $\bar{r}$ denotes the smallest interval in $\mathcal{I}$ containing r, and $\Box A$ the smallest box in $\mathcal{D}$ containing A. If g is a function, G denotes the *interval extension* of g.

4.2 ODE Solving based on Interval Arithmetic

There have been several techniques for solving ODEs based on interval arithmetic or constraint propagation; see [8, 11, 3, 2] for example. A *solution* of an ODE system $\mathcal{O}$ with the initial value $u(t_0) = u_0$ is a function $s^*(t) : \mathbb{R} \to \mathbb{R}^n$ satisfying $\mathcal{O}$ and the initial conditions $s^*(t_0) = u_0$. Also the solution of $\mathcal{O}$ is denoted as the function $s(t_0, u_0, t) : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ such that $s(t_0, u_0, t) = s^*(t)$. An *interval solution* of $\mathcal{O}$ is an interval extension S of s. A box B is an *a priori bound* of a solution s with respect to $\mathcal{O}$, $[t_0, t_1]$ and D, if for all t in $[t_0, t_1]$, $S(t_0, D, t) \subseteq B$.

Lemma 1 (Continuity of the solution of ODEs) Let f be continuous over an open set E over $\mathbb{R} \times \mathbb{R}^n$ with the property that for every (t_0, u_0) in E, the initial value problem $\{u' = f(t, u), u(t_0) = u_0\}$ has a unique solution $u(t) = s(t_0, u_0, t)$. Let $T(t_0, u_0)$ be the maximal interval in which $u(t) = s(t_0, u_0, t)$ exists. Then s is continuous over $\{(t_0, u_0, t) \mid (t_0, u_0) \in E, t \in T(t_0, u_0)\}$.

5 Proposed Method

Two algorithms Trace and PruneAndMerge compose our method. By processing Step 2 and Step 3 in Section 2.2 using interval arithmetic, we obtain an ODE system $\mathcal{O}$ which has an initial value D_0 and an initial time T_0. The main Trace algorithm computes trajectories with respect to $\mathcal{O}$. Trace simulates the continuous evolution of the system until an ask condition C is entailed. PruneAndMerge algorithm computes the precise enclosure of the point of a discrete change.

5.1 Assumptions on the Proposed Method

We consider problems that satisfy the following assumptions:

- Any ODE system $\mathcal{O}$ has the form $u' = f(t, u)$ (f is continuous) and a unique solution.
- Any ask condition C is an arithmetic constraint of the form $f_d = 0$, where f_d is a continuous function $f_d : \mathbb{R}^n \to \mathbb{R}$ and is invariant over time.

Also, in this paper, we investigate the method under the following additional assumptions that simplify the problems:

- We consider problems with a single variable and assume they cause only one discrete change at one time. It is a topic of future study to improve the method in this paper to handle more general cases.
- The solution $s(t_0, u_0, t)$ of $\mathcal{O}$ either increases or decreases monotonically over any interval T_d whose width is equal to $\mathrm{w}(T_0)$ and which T_d includes t_d satisfying $f_d(s(t_0, u_0, t_d))$.

5.2 Trace Algorithm

We give the Trace algorithm in Figure 2, and illustrate its basic idea in Figure 3. Trace computes a trajectory with respect to $\mathcal{O}$ and ensures a set of step solutions S. The main idea is to obtain the "lower bound" of the trajectory, which is computed from the initial value D_0 at $\mathrm{lb}(T_0)$, and to shift it by $\mathrm{w}(T_0)$ over time. At the beginning of the loop (line 6), Trace obtains the one-step solution of the trajectory consisting of an a priori bound $B_{t,h}$ and a tight solution D_{t+h}. Interval solutions of ODEs can be obtained using existing techniques, e.g. [11]. Trace preserves several a priori bounds of the width $\mathrm{w}(T_0)$ in a buffer Q (line 12), and obtains the step solution by taking the union of elements in Q (line 15). Figure 3 (a) illustrates the idea. Note that we assume the step size h is equal to $\mathrm{w}(T_0)$. Although boundary trajectories do not enclose an envelope created by shifting an extremal value in a trajectory (consider Figure 3 (b)), Trace can safely enclose the envelope since a priori bounds are preserved in Q.

Trace detects a time point of a discrete change by evaluating the ask condition C in each step of the

Require: an ODE system $\mathcal{O}$, initial time T_0, initial value D_0, step size h, an ask condition C
Ensure: a set S of step solutions, an area of a discrete change B_{result}

```
1: $t := \mathrm{lb}(T_0)$
2: $D := D_0$
3: $Q :=$ a buffer to preserve a trajectory for several steps
4: $B_l := \emptyset$
5: loop
6:     $(B_{t,h}, D_{t+h}) :=$ do a step computation w.r.t. $\mathcal{O}$, $t$, $D$, $h$
7:     if $C(D_{t+h})$ is entailed then
8:         $B_l := B_{t,h}$
9:     else if $C(D_{t+h})$ is overshot then
10:        break
11:    end if
12:    put $(B_{t,h}, D_{t+h})$ to the tail of $Q$
13:    remove a redundant element from the top of $Q$
14:    $D := D_{t+h}$
15:    $S := S \cup \{$the sum of solutions in Q for $\mathrm{w}(T_0)\}$
16:    $t := t + h$
17: end loop
18: $B_{result} :=$ PruneAndMerge$(Q, \mathcal{O}, C, \mathrm{w}(T_0))$
19: return $B_{result}$
```

Fig. 2 Trace algorithm

loop (lines 7 and 9). We can use the existing interval-based method like [15] to evaluate ask conditions. Trace preserves step solutions in the buffer Q, from B_l in which C is first entailed to a step where C is overshot (line 8). Figure 3 (c) illustrates the idea. As a result, Q encloses an area of a discrete change, and we obtain an output B_{result} by passing Q to the PruneAndMerge algorithm. Below is a soundness theorem of the method:

Theorem 1 Assume the assumptions in Section 5.1 hold. Assume there exists $t \in \mathbb{R}$ such that for all $u \in D_0$, $f_d(s(t_0, u, t)) > 0$ holds. Assume there exists $t' \in \mathbb{R}$ such that for all $u \in D_0$, $f_d(s(t_0, u, t')) < 0$ holds. Then, for all $u \in D_0$, there exists $t_d \in \mathbb{R}$ between t and t' such that the ask condition $f_d(s(t_0, u, t_d)) = 0$ is entailed.

Proof. For each u in D_0, $f_d(s(t_0, u, t)) > 0$ and $f_d(s(t_0, u, t')) < 0$ hold. Since f_d is continuous, there exists a solution $s_d \in \mathbb{R}^n$ such that $f_d(s_d) = 0$ holds between $s(t_0, u, t)$ and $s(t_0, u, t')$. Since s is continuous by Lemma in Section 4.2, and also since s increases or decreases monotonically, there exists a unique time t_d between t and t' such that $s(t_0, u, t_d) = s_d$ holds. ∎

5.3 PruneAndMerge Algorithm

We give the PruneAndMerge algorithm in Figure 4. Figure 5 illustrates an application of the algorithm to an expanded example in which a 2-dimensional variable over the x and y axes and a nonlinear condition C are used. Note that Figure 5 (a) corresponds to Figure 3 (c), and is overwritten by the computation result of PruneAndMerge. We apply a *branch and prune* algorithm to the time component of an area of a discrete change based on the following hull consistency (line 1):

Definition 1 (Hull consistency of a time interval of a discrete change) A time interval T_d of a discrete change with respect to an ODE system $\mathcal{O}$ and an ask condition $f_d = 0$ (F_d is an interval extension of f_d) is hull-consistent with respect to $T_d = [t_l, t_u]$, if the condition

$$(F_d(D_l) > \bar{0} \wedge F_d(D_u) < \bar{0}) \vee (F_d(D_l) < \bar{0} \wedge F_d(D_u) > \bar{0})$$

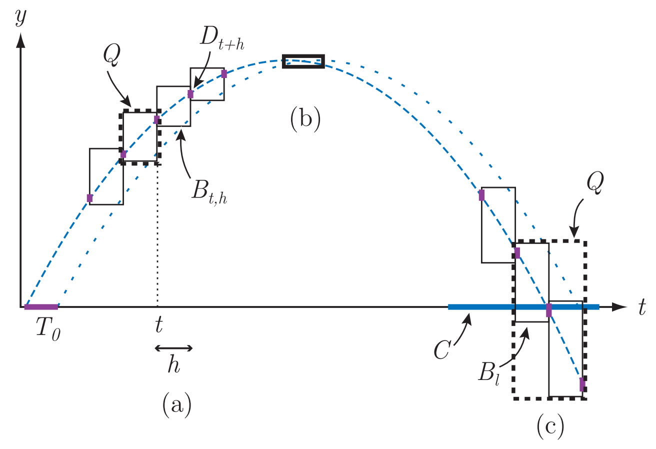

Fig. 3 Solving of an ODE system with an ask condition by TRACE

```
Require: a buffer of solutions $Q$, an ODE system $\mathcal{O}$, an ask condition $C$, width $w$
Ensure: a box $B_{result}$
1: $B := \square\{\text{branch and prune } Q \text{ w.r.t. } \mathcal{O}, C\}$
2: $B' :=$ enlarge $B$ for $w$
3: $B_{result} :=$ prune $B'_D$ w.r.t. $C$
4: return $B_{result}$
```

Fig. 4 PRUNEANDMERGE algorithm

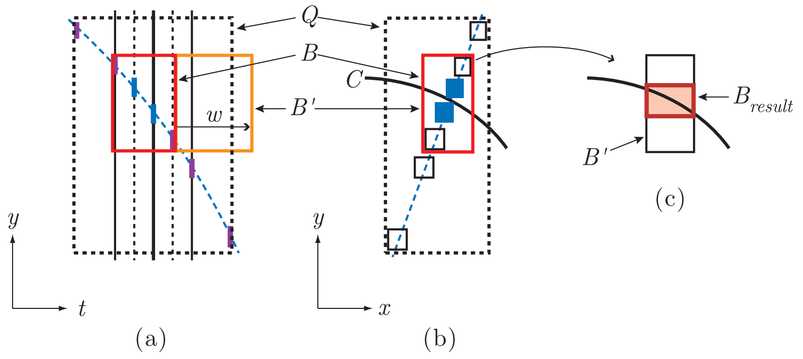

Fig. 5 Pruning of an area of a discrete change by PRUNEANDMERGE

holds, where $D_l = S(t_0, D_0, t_l)$ and $D_u = S(t_0, D_0, t_u)$.

Figure 5 (a) and (b) illustrates the result of the bisection method. The result is merged by obtaining the box enclosure (line 1). The algorithm enlarges B by w at line 2, as TRACE shifts trajectories for w(T_0) ($= w$). Moreover, we can prune B' by solving the condition C over the (x, y)-component (line 3, Figure 5 (c)).

6 Experimental Results

We evaluated the effect of the proposed method by simulating the example in Section 1. In the simulation, we used the existing interval-based solvers; VNODE-LP [10] for ODEs and Elisa [4] for NLEs. Table 2 shows the initial value, the solution at $t = 2$ after the first bounce ($t = \sqrt{2}$), and the solution at $t = 3$ after the second bounce ($t = 2\sqrt{2}$). We solved ODEs with the step size of 0.0001. We

Table. 2 Experimental results

t	(a) Direct computation	(b) Proposed method	(c) Theoretical solution
0	[10.00000, 10.00000]	[10.00000, 10.00000]	10
2	[2.424746, 2.427454]	[2.426262, 2.426676]	2.426406...
3	[0.4398578, 0.4739294]	[0.4559554, 0.4613689]	0.4594154...

can see an improvement in accuracy by comparing the results obtained by (a) direct interval computation without refinements and (b) our method. We can also make sure that the results enclose the theoretical solution (c).

7 Related Work

Techniques for modeling and verification of hybrid systems based on constraint logic programming have been proposed. Hickey and Wittenberg [7] presented an approach using the CLP(F) language, which can describe analytic relations between real variables and functions. CLP(F) supports interval arithmetic. However suppression of interval divergence in computing discrete changes is not considered. Urbina [14] proposed a method using the CLP($\mathcal{R}$) language. Computations in the method are not based on intervals, and solutions are not accurate. Moreover, only linear numerical constraints are supported.

Various safety verification techniques for hybrid systems, which compute reachable state space represented by intervals, have been developed. Some of the techniques use box representation of continuous parts of state space to check how a trajectory moves over their boundaries [13, 6]. Ratschan and She [12] proposed methods to refine the box representation using constraint propagation techniques. Refinement is done by adding constraints describing continuous and discrete changes. In these techniques, hybrid systems are modeled based on hybrid automata.

8 Conclusion

We proposed a method for obtaining interval enclosures of discrete changes in hybrid systems. After relaxing the assumptions in Section 5.1, we will next implement the proposed method in the Hybrid cc interpreter. The improved interpreter provides a reliable framework for modeling and simulating variety of hybrid systems that occur in real-life applications.

Acknowledgement.

The authors thank Mitsuhiro Nishimura, Ryota Nishizawa, Yoshiyuki Ohno, Yui Sasajima, Shinichi Tobita and Satoshi Usami for helping the evaluation of the proposed method.

References

[1] B. Carlson and V. Gupta. Hybrid cc with interval constraints. In *Proc. of HSCC'98*, *LNCS* 1386, pages 80–95. Springer, 1998.

[2] J. Cruz and P. Barahona. Constraint satisfaction differential problems. In *Proc. of CP'03*, *LNCS* 2833, pages 259–273. Springer, 2003.

[3] Y. Deville, M. Janssen, and P. Van Hentenryck. Consistency techniques in ordinary differential equations. In *Proc. of CP'98*, *LNCS* 1520, pages 162–176. Springer, 1998.

[4] L. Granvilliers and V. Sorin. Elisa. http://sourceforge.net/projects/elisa/, 2005.

[5] V. Gupta, R. Jagadeesan, V. A. Saraswat, and D. Bobrow. Programming in hybrid constraint languages. In *Hybrid Systems II*, *LNCS* 999, pages 226–251. Springer, 1995.

[6] T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond HyTech: Hybrid systems analysis using interval numerical methods. In *HSCC'00*, *LNCS* 1790, pages 130–144. Springer, 2000.

[7] T. J. Hickey and D. K. Wittenberg. Rigorous modeling of hybrid systems using interval arithmetic constraints. In *HSCC'04*, *LNCS* 2993, pages 402–416, 2004.

[8] R. J. Lohner. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In *Computational Ordinary Differential Equations*, pages 425–435. Oxford University Press, 1992.

[9] R. E. Moore. *Interval Analysis.* Prentice-Hall, 1966.

[10] N. S. Nedialkov. Vnode-lp: A validated solver for initial value problems in ordinary differential equations,. In *TR CAS-06-06-NN.* McMaster University, 2006.

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

[12] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. In *Proc. of HSCC'05*, *LNCS* 3414, pages 573–589. Springer, 2005.

[13] O. Stursberg, S. Kowalewski, I. Hoffmann, and J. Preusig. Comparing timed and hybrid automata as approximations of continuous systems. In *Hybrid Systems IV*, *LNCS* 1273, pages 361–377. Springer, 1997.

[14] L. Urbina. Analysis of hybrid systems in CLP(R). In *Proc. of CP'96*, *LNCS* 1118, pages 451–467. Springer, 1996.

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