

# Symbolic Analysis of Hybrid Systems Involving Numerous Discrete Changes Using Loop Detection

Kenichi Betsuno<sup>(✉)</sup>, Shota Matsumoto, and Kazunori Ueda

Department of Computer Science and Engineering, Waseda University,  
3-4-1, Okubo, Shinjuku-ku, Tokyo 169-8555, Japan  
{betsuno,matsusho,ueda}@ueda.info.waseda.ac.jp

**Abstract.** Hybrid systems are dynamical systems that include both continuous and discrete changes. Some hybrid systems involve a large or infinite number of discrete changes within an infinitesimal-width region of phase space. Systems with sliding mode are typical examples of such hybrid systems. It is difficult to analyze such hybrid systems through ordinary numerical simulation, since the time required for simulation increases in proportion to the number of discrete changes. In this paper, we propose a method to symbolically analyze such models involving numerous discrete changes by detecting loops and checking loop invariants of the model's behavior. The method handles parameterized hybrid systems and checks inclusion of parameterized states focusing on the values of a switching function that dominate the dynamics of sliding mode. We implemented the main part of the method in our symbolic hybrid system simulator HyLaGI, and conducted analysis of example models.

**Keywords:** Hybrid systems · Sliding mode · Loop invariants · Verification · Symbolic analysis

## 1 Introduction

Hybrid systems [6] are dynamical systems which include both continuous and discrete changes. This feature enables a number of problems in a variety of fields to be modeled as hybrid systems, e.g., physics, control engineering, and biology. Examples of hybrid systems include physical systems with impact and electronic circuits with switching. In these systems, impact and switching are expressed as discrete changes. Many of cyber-physical systems are also regarded as hybrid systems, which include discrete decisions by computational parts and hybrid behavior of physical parts. Some hybrid systems include numerous discrete changes such as infinitely frequent switching. Such systems are difficult to simulate due to the explosion in the number of discrete changes, as the processing of discrete changes is the most costly part of simulation except for systems with

---

This paper is an extended version of our earlier article [2].

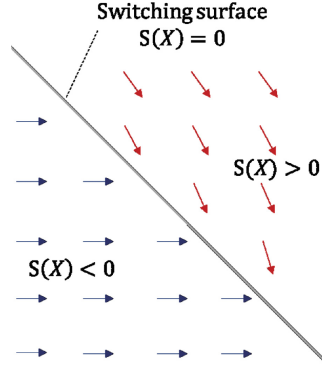


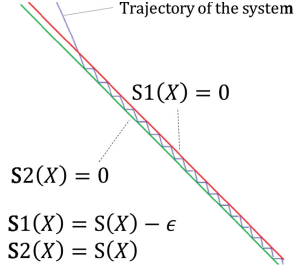
Fig. 1. Gradient vectors in the sliding mode

complex continuous dynamics. In this paper, we propose a method to analyze such models involving numerous discrete changes by detecting looping behavior of the models.

### 1.1 Sliding Mode

Sliding mode control is one of the control methods which switches the continuous dynamics of the systems when the system comes across a predefined surface, which is called a switching surface. In the sliding mode, the directions of the gradient vectors of the system (Fig. 1) are towards the switching surface  $S(X) = 0$  on the both sides of the surface. As a result, the behavior of the system is bound to the surface as if it is sliding on the surface, which is the reason why it is called a sliding mode. Systems with sliding modes can be regarded as hybrid systems involving infinitely frequent discrete changes of its control input. Such control cannot be performed in real systems as the switching of control inputs cannot be done within zero time. To reflect this, we regard the switching surface not as a simple boundary but as an infinitesimal band. This band is called a sliding region, in which the trajectory of the sliding mode is confined. Each switching function is defined as  $S1(X) = S(X)$  and  $S2(X) = S(X) + \epsilon$ , where  $S(X)$  is the ideal switching function and  $\epsilon > 0$  is infinitesimal (see Fig. 2). In this paper, we define a sliding mode as a behavior along a *sliding region*, a region enclosed by  $S1(X)$  and  $S2(X)$ .

**Example: Brake Control with Sliding Mode Control.** In this paper, we use a brake control problem as a running example, whose objective is to stop a vehicle at a target position. In this model, two control inputs are available: applying the brake or not. The position of the vehicle is denoted by variable  $x$ . The switching function is  $S(x, x') = x + x' - 100$ , where  $x'$  denotes the time derivative  $dx/dt$ . Introducing a switching function  $S(x, x')$ , the function of the current position  $x$  and the velocity  $x'$ , these two control inputs are switched



**Fig. 2.** Behavior in the sliding mode considering the switching time

according to whether the value of the switching function is positive or negative. The example model is given in the form of a HydLa [10] program. HydLa is a constraint-based language for modeling hybrid systems, and its detailed syntax and semantics are found in [10].

Figure 3 shows the example model. The top nine lines are definitions of constraints. In HydLa, all values of variables are functions of time, and each constraint is enabled at the initial time of the model. The constraint **INIT** describes the initial state of the model. The value of  $\mathbf{x}'$  is given by the inequality  $0 < \mathbf{x}' < 80$ , which means the initial velocity is uncertain but has a fixed range. The variable **sw** dominates the continuous behavior of the vehicle (see **ACC**). **EPS** describes the width of the sliding region denoted by **eps**. The temporal operator  $[\ ]$  means that the constraint always holds from the time point at which the constraint is enabled. **S** defines the switching surfaces denoted by **s1** and **s2** that enclose the sliding region. In this paper, we use variables **s1** and **s2** to represent the values of  $S1(X)$  and  $S2(X)$  at each time point, which are called *switching variables*. **SW\_CONST**, **SW\_ON** and **SW\_OFF** describe the behavior of **sw**, which is constant by default and switches only when the vehicle comes across the switching surface. The postfix minus sign in **sw-** denotes the left-hand limit  $\lim_{t_i \uparrow t} \mathbf{sw}(t_i)$ . **ACC** describes the acceleration of the vehicle and depends on the value of **sw**. **STOP** stops the vehicle if the speed equals zero and prevents the vehicle from moving back. The bottom line describes the priorities between the constraints using  $\ll$  in the form of a partial order. **SW\_ON** and **SW\_OFF** are stronger than **SW\_CONST**, and **STOP** is stronger than **ACC**, both because they describe an exceptional behavior of the vehicle.

Figure 4 shows one of the trajectories of the model assuming that the value of **eps** is sufficiently small so that  $S1(X)$ ,  $S2(X)$  and  $S(X)$  can be regarded as the same. The description of the states indicated as (1)–(7) in Fig. 4 are as follows:

1. The system is in its initial state.
2. The vehicle doesn't apply the brake and moves at a constant speed because  $S < 0$ .
3. The system reaches the switching surface and switches the control input; the vehicle begins to apply the brake.

---

```

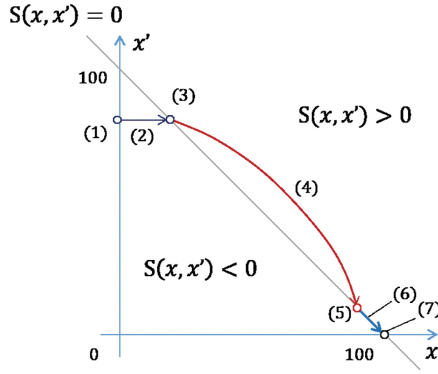
INIT <=> x = 0 /\ 0 < x' < 80 /\ sw = 0.
EPS <=> 0 < eps < 0.1 /\ [] (eps' = 0).
S <=> [] (s1 = x' + x - 100 - eps /\
          s2 = x' + x - 100).
SW_CONST <=> [] (sw' = 0).
SW_ON <=> [] (s1- = 0 /\ s1'- >= 0 => sw = 1 - sw-).
SW_OFF <=> [] (s2- = 0 /\ s2'- <= 0 => sw = 1 - sw-).
ACC <=> [] (x'' = -50 * sw).
STOP <=> [] (x' = 0 => x'' = 0).

INIT, SW_CONST << (SW_ON, SW_OFF), ACC << STOP, S, EPS.

```

---

**Fig. 3.** HydLa program of the vehicle with sliding mode control



**Fig. 4.** Trajectory of the vehicle with sliding mode control

4. The vehicle moves at a constant acceleration because of the brake.
5. The system reaches the switching surface again and switches the control input.
6. The system enters the sliding mode, that is, moves along the sliding surface.
7. The vehicle stops at the destination  $x = 100$ .

## 2 Proposed Method

The goal of our method is rigorous simulation and reasoning of hybrid systems which include sliding modes and other chattering behaviors. In this paper, we focus on the central part of our method, namely how to establish conditions that should hold upon exit of sliding mode, which will then enable us to continue simulation beyond the sliding mode if desired. This involves the recognition and analysis of a looping behavior with an arbitrary large number of iterations. Since fully automatic analysis of loops is hard in general, we allow some auxiliary

information to be provided for the analysis. Specifically, the input and output of our method, up until (but not beyond) loop analysis, are as follows:

**Input:**

1. Model description given as a HydLa program that expresses a target system,
2. Model specification given as a proposition that should be satisfied by the system upon exit of a loop,
3. Termination condition under which the system definitely exits from the loop,
4. Switching function which the switching of the dynamics depends on.

**Output:** Whether the system will satisfy the model specification or not upon exit of the loop.

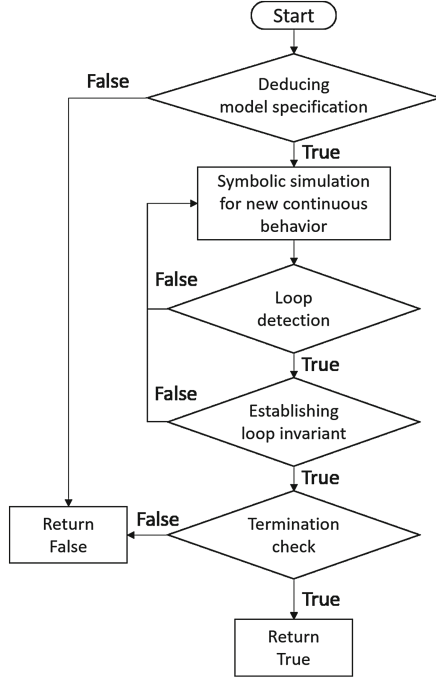
The key idea of the method is to regard the sliding mode as a loop, and the occurrence of the sliding mode will be detected through loop detection. The overall flow of the method is shown in Fig. 5. The proposed method consists of the following four steps:

1. **loop detection:** detecting a loop which may be sliding mode
2. **establishing loop invariant:** verifying that the system stays in the sliding region during the loop
3. **termination check:** verifying that the system will exit from the loop
4. **deducing model specification:** confirming that the system satisfies the goal conditions if it was within the sliding region and satisfies the loop termination condition.

## 2.1 Loop Detection

In this step, we judge whether newly computed continuous behaviours have already been computed in the past. A set of continuous behaviors is expressed by parameterized functions of time that are computed by symbolic simulation. Continuous behaviors contain information about initial values of state variables and constraints imposed on state variables. If the new behaviors are found to be already computed, it means that the behaviors of the system enter a loop and all the values of variables stay in an already computed range. In our previous work [12], we proposed a method to detect such loops focusing on inclusion relation between continuous behaviors. However, the method required the behaviors of *all* variables in the system to be included, which is not the case with many practical systems including sliding mode control. To solve this problem, we propose a new inclusion relation focusing only on switching variables as follows:

**Definition 1.** *Continuous behavior  $\beta_1$  includes continuous behavior  $\beta_2$  about switching functions if*



**Fig. 5.** Flowchart of the proposed method with a symbolic simulator

1. The constraints imposed on the state variables are the same.
2. The parameterized values of  $S1(X)$  and  $S2(X)$  at the beginning of  $\beta_2$  are included in those of  $\beta_1$ .
3. The parameterized values of the  $n$ -th derivatives of  $S1(X)$  and  $S2(X)$  at the beginning of  $\beta_2$  are included in those of  $\beta_1$ , where  $n$  is any positive integer.

The inclusion relation between two parameterized values means that a domain of one parameterized value includes that of the other. For example, provided that a parameter  $p$  satisfies  $p \in [0, 1]$  where  $[a, b]$  denotes a closed interval,  $p \times 0.8 \subseteq p$  holds. If the above conditions are satisfied, we can conclude that the system enters a loop about switching variables and that the value of each switching variable stays in a computed range.

We chose HydLa as a modeling language that does not come with an explicit notion of loops, while hybrid automata [4] represent explicit state transition. However, even if we choose hybrid automata, loop detection described here is still necessary because loops expressed by hybrid automata just represent those of control flow, while we are concerned with loops defined as state inclusion.

## 2.2 Establishing Loop Invariant

In this step, we verify that the loop detected in the previous step satisfies the loop invariant, which means that the systems stay in the sliding region. To verify that, we must confirm that the condition  $S1(X) \leq 0 \leq S2(X)$  holds throughout the first iteration of the loop. If it does, we can conclude that the condition holds until the loop terminates, because the values of switching variables  $S1(X)$  and  $S2(X)$  stay within the range of the first iteration as described in Sect. 2.1. Assuming that we can obtain information about the trajectories of switching variables from the symbolic simulator, this step is conducted by checking those trajectories for one iteration.

## 2.3 Termination Check

In this step, we verify that the termination condition eventually holds. We can verify it by finding some  $F(X)$  and a target value  $g$  that satisfies the conditions below:

- The value of  $F(X)$  increases at least by some  $k \in \mathbb{R}_+$  in each iteration.
- $F(X) \geq g \wedge \text{LoopInvariant}(X) \Rightarrow \text{LoopTerminationCondition}(X)$  is valid.

Note that similar conditions are found in the proof of loop termination in programming languages, and  $F(X)$  acts as a loop variant. In the second condition, we can also adopt an equality  $F(X) = g$  instead of  $F(X) \geq g$  if  $F(X)$  is known to be continuous. The choice about which one should be used depends on the form of  $\text{LoopTerminationCondition}(X)$ .

*Example 1.* In the brake control example, we define  $F(X) := -x'$  and  $g := 0$ . It should be verified that  $\exists k \in \mathbb{R}_+ (\forall i \in \mathbb{N} (F_{i+1} - F_i \geq k))$ . In the loop, the vehicle repeats braking and moving at a constant speed. Provided that the system is in the first quadrant of the phase space, braking causes the system to move in the positive direction of the  $x$  axis and the negative direction of the  $x'$  axis (see Fig. 1). Thus,  $x'$  decreases at least by  $\epsilon(\text{eps})$  in each braking, which means  $\forall i \in \mathbb{N} (F_{i+1} - F_i \geq \epsilon)$ .

## 2.4 Deducing Model Specification

In this step, it is confirmed that the system meets the target specification if it stays inside of the sliding region and satisfies the loop termination condition. Thus, the validity of the following formula should be verified:

$$\text{LoopInvariant}(X) \wedge \text{LoopTerminationCondition}(X) \Rightarrow \text{TargetSpecification}(X) \quad (1)$$

Note that the concrete form of  $\text{LoopInvariant}(X)$  here is the inequality  $S1(X) \leq 0 \leq S2(X)$ . The verification of Formula (1) can be performed statically, and this is why it is performed at the beginning rather than the end of the analysis in Fig. 5.

**Table 1.** Case branching in the brake control model with SMC

Initial speed $x'$	Existence of a loop
$(0, 50 - 35\sqrt{2})$	No
$50 - 35\sqrt{2}$	No
$(50 - 35\sqrt{2}, 80)$	Yes

*Example 2.* In the brake control example,  $LoopInvariant(X)$  is  $x + x' - 100 - \epsilon \leq 0 \leq x + x' - 100$ ,  $LoopTerminationCondition(X)$  is  $x' = 0$ , and  $TargetSpecification(X)$  is  $100 - \epsilon \leq x \leq 100 \wedge x' = 0$ . We can transform  $LoopInvariant(X)$  into  $100 - \epsilon \leq x \leq 100$  by letting  $x' = 0$ . Here,  $100 - \epsilon \leq x \leq 100 \wedge x' = 0$  is exactly the same as  $TargetSpecification(X)$ , therefore Formula (1) is verified.

### 3 Experiment

We have implemented the loop detection step in HyLaGI [7], which is a simulator of HydLa programs that we have developed. HyLaGI has several key features below:

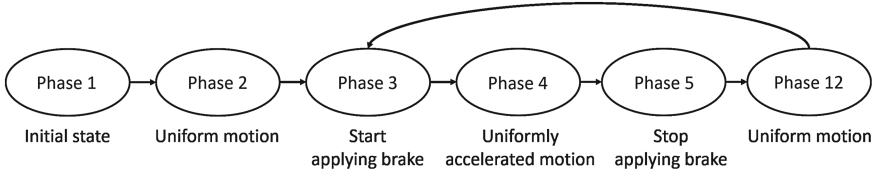
- HyLaGI performs all computations symbolically and the result is free from computation errors caused by floating-point arithmetic.
- Uncertain values in systems are expressed by symbolic parameters. This enables parameter analysis of models.
- Case analysis is conducted on demand if the uncertainty of the systems leads to qualitative branching of behavior, e.g., whether two balls collide or not.

With this implementation, we conducted experiments of loop detection for several models. In this section, we also discuss how the remaining properties can be derived from the output of HyLaGI.

**Brake Control Model with Sliding Mode Control.** The first example is the brake control model with sliding mode control, which is described in Sect. 1.1. In order to detect a loop of the model, the initial speed of the vehicle should be a parameterized value such as  $p \in (0, 80)$ . As for **eps**, we use a small fixed value 0.5. If we use an uncertain value for **eps**, the current symbolic computation engine fails to judge an inclusion relation of continuous behaviors, while it can still simulate up to the second braking. Even if we adopt a fixed value as **eps**, the proposed method has an advantage in reducing computational costs of simulation. With this initial condition, the computation with our implementation branched into three different cases depending on the initial speed, as shown in Table 1.

When  $0 < x' \leq 50 - 35\sqrt{2}$ , the vehicle stops on the first braking and the system doesn't enter a loop. Otherwise, the implementation detected a loop because the state of the first braking includes the state of the second braking. The





**Fig. 6.** State transition of the brake control model

parameterized values of  $s_1$  and its derivatives at the first and second brakings are shown in Table 2, where  $p_{s_1} \in ((-70\sqrt{2} - 101)/2, -41/2)$ . Figure 6 shows a state transition diagram computed by HyLaGI.

**Table 2.** The parameterized values of  $s_1$  and  $s_1'$  at the first and second brakings

Variable	First braking	Second braking
$s_1$	0	0
$s_1'$	$p_{s_1} + 201/2$	$50 - \sqrt{10401 + 4p_{s_1}(101 + p_{s_1})}/2$

In Fig. 6, Phase 1 corresponds to the initial state of the model. After that, the vehicle moves at a constant speed (Phase 2) and applies the brake when  $s_1 = 0$  (Phase 3). The speed decreases by the brake (Phase 4) and the braking is stopped when  $s_2 = 0$  (Phase 5). The vehicle moves in uniform motion again (Phase 12) and the state transits back to Phase 3 because the next state of Phase 12 is included by Phase 3. Note that this inclusion can be detected by focusing on switching variables, which is an important difference from the previous work.

In this model, the initial speed is abstracted to an interval value  $(0, 80)$ . If we adopted some fixed value as the initial speed, the first derivatives of the switching variables would decrease at each braking and no inclusion between states could have been detected. For example, Table 3 shows the values of variables at each time point of braking where the initial speed equals 80. In Table 3,  $s_1$  and  $s_2$  and their derivatives are fixed values, and the values of the derivatives at the first braking do not include those at the second braking, which means that the loop detection fails. This is why we need to abstract the initial speed in this model.

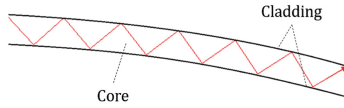
For the steps after loop detection, the readers are referred to Sect. 2 for this example.

If we conducted symbolic simulation without loop detection, the size and complexity of the symbolic formulas of state variables would increase quickly, which would make the simulation finally come to a halt.

**Optical Fiber Model.** The next model is an optical fiber model; a (slow) ray proceeds inside an optical fiber. When the ray hits a boundary, total internal reflection occurs and the ray is confined in the core. In this model, we suppose that the fiber forms a circle and the attenuation of the light can be ignored (Fig. 7). The HydLa program is shown in Fig. 8. The variables  $x$  and  $y$  denote

**Table 3.** The values of variables at each braking with  $x'(0) = 80$ 

Variable	First braking	Second braking
$s1$	0	0
$s2$	$1/2$	$1/2$
$s1'$	80	$50 - 5\sqrt{38}$
$s2'$	80	$50 - 5\sqrt{38}$
$x$	$41/2$	$(101 + 10\sqrt{38})/2$
$x'$	80	$50 - 5\sqrt{38}$

**Fig. 7.** Trajectory of an optical fiber model

---

```

INIT <=> x = 0 /\ y = r + eps / 2 /\ x' = 1 /\ y' = -1.
CONSTANTS <=> [](r = 5) /\ eps = 0.1 /\ [](eps' = 0).
S <=> [](s1 = x^2 + y^2 - (r+eps)^2 /\ s2 = x^2 + y^2 - r^2).
XCONST <=> [](x'' = 0).
YCONST <=> [](y'' = 0).
REFLECT_1 <=> [](s1- = 0 =>
  x' = x'- - 2*(x'- * x- + y'- * y-) / (r+eps)^2 * x- /\
  y' = y'- - 2*(x'- * x- + y'- * y-) / (r+eps)^2 * y-).
REFLECT_2 <=> [](s2- = 0 =>
  x' = x'- - 2*(x'- * x- + y'- * y-) / r^2 * x- /\
  y' = y'- - 2*(x'- * x- + y'- * y-) / r^2 * y-).

INIT, (XCONST, YCONST) << (REFLECT_1, REFLECT_2) << S, CONSTANTS.

```

---

**Fig. 8.** HydLa program of the optical fiber model

the position of the ray, while  $r$  denotes the radius of the circle and equals five. The width of the fiber is denoted by  $\text{eps}$  and is set to 0.1.

In the loop detection of this model, HyLaGI detected a loop between the first reflection on the inner edge and the second one. Table 4 shows the values of state variables at each inner reflection. Here, the symbolic expressions about  $x$ ,  $y$  and their derivatives at the second inner reflection are different from those at the first inner reflection and are more complex. However,  $s1$ ,  $s2$  and their derivatives are the same and that is the reason why we can conclude that the behavior of the system forms a loop. It follows from this result that the time of each cycle of the loop is constant because the trajectories of  $s1$  and  $s2$  are both constant in the loop.

**Table 4.** Result of the loop detection experiment of the optical fiber model

Variable	First inner reflection	Second inner reflection
$s1$	$-101/100$	$-101/100$
$s2$	$0$	$0$
$s1'$	$-9799^{1/2} \times 1/10$	$-9799^{1/2} \times 1/10$
$s2'$	$-9799^{1/2} \times 1/10$	$-9799^{1/2} \times 1/10$
$x$	$(101 - 9799^{1/2}) \times 1/40$	$(103937993^{1/2} \times 525301 + 1055803 \times 9799^{1/2} + 98393897 - 10607^{1/2} \times 48954599) \times 1/2080800000$
$x'$	$(201 + 101 \times 9799^{1/2}) \times 1/10000$	$(10141798597 + 103937993^{1/2} \times 2050401 + 10607^{1/2} \times 5045919499 - 9799^{1/2} \times 4121103) \times 1/520200000000$
$y$	$(101 + 9799^{1/2}) \times 1/40$	$(10607^{1/2} \times 48954599 + 103937993^{1/2} \times 525301 + 98393897 - 9799^{1/2} \times 1055803) \times 1/2080800000$
$y'$	$(101 \times 9799^{1/2} - 201) \times 1/10000$	$(10607^{1/2} \times 5045919499 - 10141798597 - 103937993^{1/2} \times 2050401 - 9799^{1/2} \times 4121103) \times 1/520200000000$

**Pushing Two Balls.** We cite this model from [5]. In the initial state of the model, two balls are touching each other and are not moving. After one second, we push one of the balls toward the other until 3.5s elapse. While pushing, infinitely frequent collisions between the two balls occur; therefore direct simulation is impossible. To handle such infiniteness, we use our loop detection method. Figure 9 shows a HyLa program of this model. We use an infinitesimal parameter  $\epsilon ps$  as the initial distance between the two balls. The variables  $x1$  and  $x2$  denote the positions of the balls. A timer variable  $tt$  is used to trigger events about pushing. The diameter of each ball and the coefficient of restitution are set to one.

The implemented system detected a loop between the first and the second collisions. Table 5 shows the values of variables at each collision. As in the optical fiber model, the values of  $s1$ ,  $s2$  and their derivatives are the same between two collisions, while  $x1$  and  $x2$  are different.

If we want to confirm that the loop invariant is satisfied, we need to check the validity of the following proposition derived from the symbolic simulation:

$$\begin{aligned} \forall t \in [0, \sqrt{2\epsilon ps}] \ (\forall \epsilon ps \in [0, 0.01] \ (-0.5t(t - \sqrt{2\epsilon ps}) - \epsilon ps \leq 0 \\ \wedge 0 \leq -0.5t(t - \sqrt{2\epsilon ps}))) \end{aligned} \quad (2)$$

We can confirm the validity of Formula 2 by the *Reduce* function of Mathematica currently used as the symbolic engine of HyLaGI.

Next, let us consider how to prove loop termination caused by the inequality  $tt \geq 3.5$ . As shown in Table 5, the values of  $s1$ ,  $s2$  and their derivatives are

---

```

INIT <=> x1 = 0 /\ x2 = 1 + eps /\ x1' = 0 /\ x2' = 0.
TIMER <=> tt = 0 /\ [] (tt' = 1).
S <=> [] (s1 = x2 - x1 - 1 - eps /\ s2 = x2 - x1 - 1).
FORCE(x) <=> [] (tt- < 1 => x'' = 0)
    /\ [] (1 <= tt- < 3.5 => x'' = 1)
    /\ [] (tt- >= 3.5 => x'' = 0).
CONSTV(x) <=> [] (x'' = 0).
COLLISION <=> [] (s2- = 0 => x1' = x2'- /\ x2' = x1'-).
EPS <=> 0 < eps < 0.01 /\ [] (eps' = 0).

INIT, S, EPS, TIMER,
(FORCE(x1), CONSTV(x2)) << COLLISION.

```

---

**Fig. 9.** HydLa program of the pushing of two balls**Table 5.** The values of variables at each collision

Variable	First collision	Second collision
$s1$	$-eps$	$-eps$
$s2$	0	0
$s1'$	$-\sqrt{2eps}$	$-\sqrt{2eps}$
$s2'$	$-\sqrt{2eps}$	$-\sqrt{2eps}$
$x1$	$eps$	$5eps$
$x2$	$1+eps$	$1+5eps$
$x1'$	0	$\sqrt{2eps}$
$x2'$	$\sqrt{2eps}$	$2\sqrt{2eps}$
$tt$	$1+\sqrt{2eps}$	$1+3\sqrt{2eps}$

the same after each collision. In addition, their second derivatives are  $s2'' = s1'' = x2'' - x1'' = 1$  throughout the loop. Therefore, the time interval of each iteration is also constant, and from Table 5 we can derive that  $tt$  increases by  $2\sqrt{2eps}$  at each iteration. Adopting  $F(X) := tt$  and  $g := 3.5$ , it is obvious that  $F(X)$  eventually exceeds  $g$ , and the proposition  $F(X) \geq g \wedge LoopInvariant(X) \Rightarrow tt- \geq 3.5$  also holds obviously.

Finally, consider  $x2 - x1 - 1 \leq eps$  as a target specification, which means that the relative position of the second ball from the first one is less than  $eps$  when we stop pushing. This specification is implied from the invariant about  $s1$ , that is,  $s1 = x2 - x1 - 1 - eps \leq 0$ . Thus, the specification is verified.

## 4 Related Work

In previous work, the sliding mode dynamics involving chattering was approximated by non-chattering dynamics on a sliding surface [1, 3, 8, 11]. Most of these

approaches adopt numerical simulation methods and aim at simulating such models with approximation. Our approach is based on symbolic simulation and focuses on validated simulation of parameterized hybrid systems rather than simulation with approximations. Compared to those numerical methods, our approach can deal with hybrid models with parameters and need not approximate the behavior of the models. Owing to its symbolic approach, the proposed method is currently applicable only to linear and simple nonlinear models. Challenges for the future include extending this approach to nonlinear models without analytic solutions, which involves integration of symbolic computation and rigorous numerical computation.

A hybrid theorem prover KeYmaera [9] supports verification of parametric hybrid systems. However, KeYmaera aims at verification, while our goal is to perform validated simulation of parametric hybrid systems.

## 5 Conclusion

In this paper, we proposed a method for the symbolic simulation of hybrid systems with a large or an infinite number of discrete changes, most typically due to sliding mode. We also implemented the algorithm for detecting loops caused by sliding mode by extending HyLaGI, a symbolic simulator of parameterized hybrid systems. Although the automatic discovery of loop invariants and loop variants is our future work, we found that once a loop is detected, the output of HyLaGI provides us with useful information for the rest of the four steps discussed in Sect. 2. Extending our algorithm and implementation with differential invariants and differential variants and continuing symbolic simulation until and beyond the exit of a loop is our future work. This part of verification is strongly dependent on the hardness of each model, since it is necessary to verify that the system will eventually satisfy exit conditions of a loop rather than simulating the system step by step. Automatic discovery of loop invariants and loop variants may be possible for some systems, but in general, interacting with users for auxiliary information as in proof assistant systems seems to be a key idea for the verification of complex models.

We have found that some models need to be abstracted with respect to the initial states. At present, such abstraction is given manually, but it is of course desirable to automate this step using a kind of abstraction refinement procedure that performs both refinement and generalization. Handling models with multiple switching surfaces is another direction of our future work.

## References

1. Aljarbouh, A., Caillaud, B.: Chattering-free simulation of hybrid dynamical systems with the Functional Mock-Up Interface 2.0. In: Proceedings of 1st Japanese Modelica Conference, Linköping Electronic Conference Proceedings, vol. 124, no. 013, pp. 95–105 (2016)

2. Betsuno, K., Matsumoto, S., Wakatsuki, Y., Ueda, K.: Analysis of hybrid systems involving numerous discrete changes using loop detection. In: Proceedings of 30th Annual Conference of the Japanese Society for Artificial Intelligence, 1F3-4 (2016). (in Japanese)
3. Filippov, A.F.: Differential Equations with Discontinuous Right-Hand Sides. Mathematics and its Applications. Kluwer Academic, Boston (1988)
4. Henzinger, T.: The theory of hybrid automata. In: Proceedings of LICS 1996, pp. 278–292. IEEE Computer Society Press (1996)
5. Lee, E.A.: Constructive models of discrete and continuous physical phenomena. IEEE Access **2**, 797–821 (2014)
6. Lunze, J.: Handbook of Hybrid Systems Control: Theory, Tools, Applications. Cambridge University Press, Cambridge (2009)
7. Matsumoto, S., Kono, F., Kobayashi, T., Ueda, K.: HyLaGI: symbolic implementation of a hybrid constraint language HydLa. Electron. Notes Theoret. Comput. Sci. **317**, 109–115 (2015)
8. Mosterman, P.J., Zhao, F., Biswas, G.: Sliding mode model semantics and simulation for hybrid systems. In: Antsaklis, P., Lemmon, M., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1997. LNCS, vol. 1567, pp. 218–237. Springer, Heidelberg (1999). doi:[10.1007/3-540-49163-5\\_12](https://doi.org/10.1007/3-540-49163-5_12)
9. Platzer, A., Quesel, J.-D.: KeYmaera: a hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 171–178. Springer, Heidelberg (2008). doi:[10.1007/978-3-540-71070-7\\_15](https://doi.org/10.1007/978-3-540-71070-7_15)
10. Ueda, K., Matsumoto, S., Takeguchi, A., Hosobe, H., Ishii, D.: HydLa: a high-level language for hybrid systems. In: Proceedings of 2nd Workshop on Logics for System Analysis (LfSA 2012, affiliated with CAV 2012), pp. 3–17 (2012)
11. Utkin, V.I.: Sliding Modes in Control and Optimization. Springer, Berlin (1992)
12. Wakatsuki, Y., Matsumoto, S., Ueda, K.: Introduction of LTL model checking to a hybrid constraint system HyLaGI. In: Proceedings of 30th Annual Conference of the Japanese Society for Artificial Intelligence, 1F3-1 (2016). (in Japanese)

Cyber Physical Systems. Design, Modeling, and  
Evaluation

6th International Workshop, CyPhy 2016, Pittsburgh,  
PA, USA, October 6, 2016, Revised Selected Papers

Berger, C.; Mousavi, M.R.; Wisniewski, R. (Eds.)

2017, XI, 127 p. 47 illus., Softcover

ISBN: 978-3-319-51737-7