The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

Dynamic Reduction of Guarded Constraints for the Hybrid Systems Modeling Language HydLa

Takafumi Horiuchi Kazunori Ueda

Department of Computer Science and Engineering, Waseda University

HydLa is a language for modeling hybrid systems—dynamical systems that intermix discrete and continuous behavior. Its adoption of a constraint-based framework benefits the language in various ways, such as allowing a concise representation of systems and performing error-free high precision simulations. In spite of all the advantages, the computations among sets of constraints become a bottleneck in simulation time when handling some large-scale models. The purpose of this research lies in providing a method of improving the computational efficiency and the scalability of the language and its simulator. This is achieved by considering the monotonic aspects in HydLa models to dynamically reduce the size of guarded constraints. Results show that this approach is effective for models that contain multiple objects represented by guard conditions. As for the model evaluated in an experiment in the research, the overall computational time has reduced to approximately half the original length.

1. Introduction

In recent years, there has been a rapid growth of interest in hybrid systems [Lunze] — systems that intermix discrete and continuous behavior. HydLa [Ueda et al.], which is the subject language of this research, is a modeling language for hybrid systems. A notable feature of HydLa is that it allows a concise representation of hybrid systems by directly handling high-level descriptions of mathematical and logical formulas as the source program. This feature is realized by the adoption of a constraint-based framework. In HydLa, constraints are the basic components that specify the behavior of models. These constraints are structured in the form of a constraint hierarchy [Borning et al.], among which the consistency is retained when processing the model.

The guaranteed-accuracy implementation of HydLa is called HyLaGI [Matsumoto], which takes a HydLa model as the input, simulates the model, and outputs the solution trajectory. HyLaGI has strong computational features such as performing error-free symbolic calculations and handling uncertain parameters. However, it is known that the computations on constraint sets become a bottleneck when the number of guarded constraints (i.e. constraints enabled when guard conditions are entailed) is large.

In this paper, we observe the relation between the size of guarded constraints and simulation time and propose a method for improving the efficiency of simulations by considering the monotonic behavior in HydLa models to dynamically reduce the number of guarded constraints.

2. Guarded Constraints in Simulations

In the current simulation algorithm of HyLaGI, the simulation time increases in relation to the size of guarded constraints that appears in the model. This becomes a severe bottleneck when processing large-scale models that contain a large number of objects represented by guarded constraints. The correlation between these two elements can be observed in a series of simulations of the model shown in Figure 1. This HydLa model represents a ball bouncing on a flat surface, where the surface is split into `N` pieces, each of which has the length of 1 unit. Constraints `INIT_X`, `INIT_Y`, and `FALL` define the initial and default behavior of the ball and `BOUNCE` describes the behavior of the ball when it collides with the surface at height 0. In the constraint declaration at line 9, `FALL` is assigned a weaker priority than `BOUNCE`, therefore `FALL` is temporarily unadopted when the two constraints conflict at the timing of the bounce. The pieces of the surface are generated in constraint `BOUNCES` with a list notation. For simplicity, the coefficient of restitution, initial horizontal velocity, and initial height of the ball are all set to 1. The model is simulated until time `N`, where the ball travels from the left end to the right end of the split surface.

```
1  // #define N 100
2
3  INIT_X(v)   <=> x=0 & [](x'=v).
4  INIT_Y(h)   <=> y=h & y'=0.
5  FALL        <=> [](y''= -9.8).
6  BOUNCE(l,r) <=> []((y- = 0) & (l <= x- < r)
7                  => y'= -1.0*y'-).
8  BOUNCES := {BOUNCE(i,i+1) | i in {0..N-1}}.
9  INIT_X(1), INIT_Y(1), (FALL << BOUNCES).
10
11 // #hylagi -t 100
```

Figure 1: Model of Bouncing Ball on Split Surface

As an experiment, the model is simulated multiple times, each time with different values of `N` ranging from 10 to 200. This setting will change the number of guarded constraints in the model, with the increase of one for each increment on the value of `N`. Other conditions are consistent throughout

Contact: Takafumi Horiuchi, Department of Computer Science and Engineering, Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555, Japan, Bldg.63, 5F-02, 03-5286-3340, horiuchi(at)ueda.info.waseda.ac.jp

the experiment. The result[*1] of the experiment is shown in Figure 4 (circle plots). These data indicate that the increase in the value of `N` leads to a longer simulation time. The regression line among the plotted data (original) is represented by the following equation:

$$SimulationTime = 1.1463\,N^2 + 1.5554\,N + 0.4949 \quad (1)$$

Thus, it can be inferred that, for this model, the simulation time increases in proportion to the square of the number of guarded constraints. In general, this behavior can be a bottleneck in the simulation of large-scale models that contain a large number of guarded constraints.

3. Location of the Bottleneck

To identify the location of the bottleneck in the current simulation algorithm, the distribution of time consumption among different computational units was measured. In a simulation of the split surface model where `N` was set to 100, the entire simulation time was 1.23×10^8 seconds. The most time consuming procedure was *FindMinTime*, a function that calculates the minimum satisfiable time of the next discrete change, which consumed 1.17×10^8 seconds (i.e. 96% of the entire simulation time). Taking this into account, the computation of *FindMinTime* is the location of the aforesaid bottleneck problem.

4. Reduction of Guarded Constraints

The computation of *FindMinTime* is performed by checking the satisfiability of conditions in antecedents of all the guarded constraints that appears in the model. This can be a drawback in terms of efficiency of simulations. For example, when considering a model of a ball bouncing down a staircase (Figure 2), the ball will never interact with the steps which it has already passed. In this case, it is unnecessary to check the satisfiability of guards that represent the steps behind the current position of the ball. The present simulation algorithm does not consider these situations, resulting in redundant computations when iterating through all of the guards during the computations of *FindMinTime*.

This inefficiency in the current simulation algorithm leads to the proposal of this research—dynamically reducing the number of guarded constraints. The aim of this proposal is to avoid redundant calculations when evaluating the entailment of guards and improving the computational efficiency of simulations.

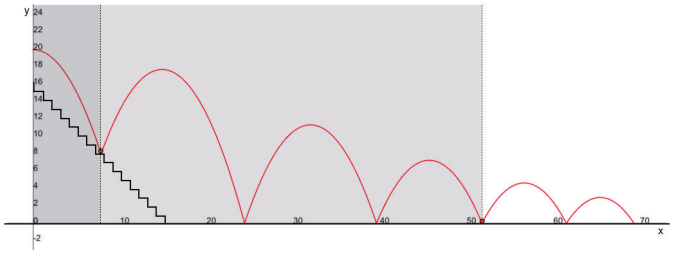

Figure 2: Concept of Omitting Verbose Guard Evaluations

The removal of guarded constraints is *safe only when it is certain that the guard condition will never be satisfied in the future*. Inadequate removal of guards will lead to wrong simulation results, which must be avoided. The decision of whether the guards are removable or not can be made by taking into account the concept of monotonicity, described by the following propositions:

$$\forall t \in (\alpha, \beta)\ \frac{d}{dt}x(t) \geq 0 \quad (2)$$

$$\forall t \in (\alpha, \beta)\ \frac{d}{dt}x(t) \leq 0 \quad (3)$$

Proposition (2) ensures the monotonic increase of variable x in the time interval (α, β) and proposition (3) ensures the monotonic decrease. The proposed method considers monotonicity of variables from two approaches, one dealing with uniform monotonicity and another with alternating monotonicity.

4.1 Approach to Uniform Monotonicity

The first approach considers the monotonicity of variables that are uniform throughout the simulation. This situation corresponds to the case where α and β are set to 0 and $MaxT$ (i.e. maximum simulation time), respectively, in propositions (2) and (3). For instance, this behavior is exhibited in the model of a ball bouncing on split surface, where variable `x` keeps increasing from the beginning to the end of simulation. The conceptual diagram of this approach is illustrated in Figure 3, representing the case where a variable is monotonically increasing throughout the simulation.

In order to apply this method for the dynamic reduction of guarded constraints, the truth/falsity of the uniform monotonicity must be evaluated, which can be achieved by using model checking techniques. After evaluating the uniformly monotonic behavior of variables, that information is used to determine the removable guarded constraints.

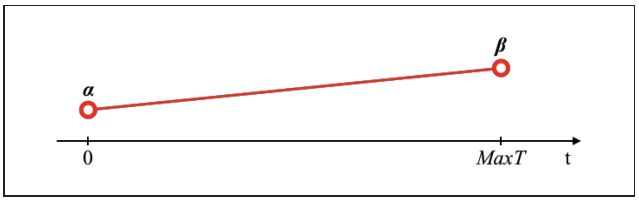

Figure 3: Concept of Approach to Uniform Monotonicity

4.2 Approach to Alternating Monotonicity

The types of model that the above-mentioned uniform approach can make use of is limited to cases where monotonic aspects are preserved from the beginning to the end of simulations. Variables that alternate its monotonic behavior during simulations would not be covered in this approach. To enhance the uniform approach and to enable the handling of alternating monotonicity of variables, the method can be enhanced by the integration with assertion

∗1 Execution environment: CentOS 7.4.1708, AMD Ryzen Threadripper 1950X 16-Core Processor, 64GB Memory, clang 6.0.0 compiler, Mathematica 11.3.0.

techniques. The basic concept of this method is equivalent to the uniform approach, where, in the beginning of the procedure, the uniformity of monotonicity is assumed to hold throughout the simulation. The notable feature of this method is that the uniformity is constantly asserted by statements `ASSERT(x'>=0)` and `ASSERT(x'<=0)`. When violations on these conditions are detected, the simulation is restarted from that time point with the assumptions on monotonicity inverted.

Illustrated in Figure 5 is the concept of the enhanced method. α and β in the diagram corresponds to those in propositions (2) and (3). In the beginning of the procedure, α is set to 0 and β is set to $MaxT$. The properties are *not* determined to be held at this point. This is different from the case in the method for uniform monotonicity, where either of the propositions (2) or (3) were determined to be held from the computations made prior to the simulation. On the occurrence of assertion violations, that time point is set as the new α and the procedure continues. This process is repeated until the simulation time reaches $MaxT$. The applicable guarded constraints are removed and reset dynamically during the simulation by using these information of monotonicity.

5. Experimental Results

The approach to uniform monotonicity was implemented and its effectiveness was evaluated with the model of a ball bouncing on split surface (Figure 1). The comparison between the simulation time of the original algorithm of HyLaGI and the algorithm with the proposed method is shown in Figure 4. The regression line for the result of the proposed method is represented by the following equation:

$$SimulationTime = 0.6083\,N^2 + 0.8763\,N + 1.691 \quad (4)$$

By comparing equations (1) and (4), we can infer that the simulation time with the proposed method is half the length of the original. Since the method aims to improve simulation efficiency by reducing the size of guarded constraints, this proposal is expected to be effective for models that contain large number of guarded constraints.

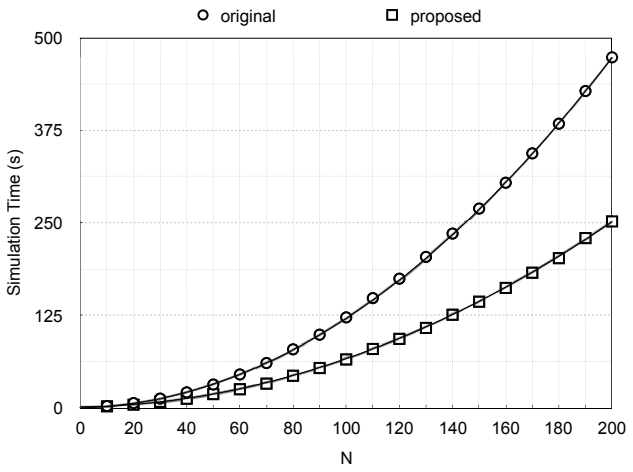

Figure 4: Simulation Time of Split Surface Model

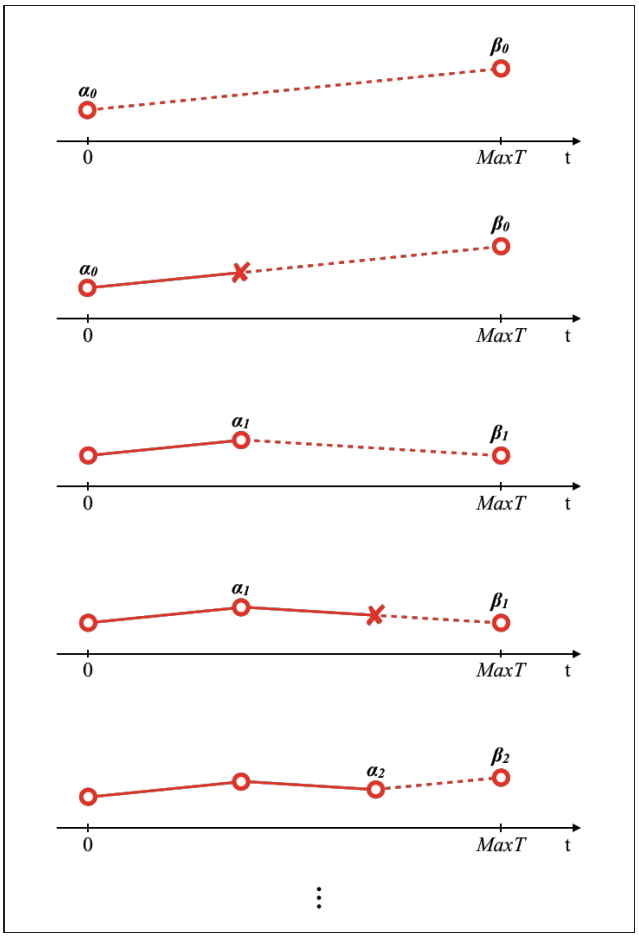

Figure 5: Concept of Approach to Uniform Monotonicity

6. Conclusion and Future Work

In this research, we proposed an optimization technique for the simulation algorithm of HyLaGI. This was achieved through dynamically reducing the size of guarded constraints that appears in the model, based on monotonicity.

Future work of this research includes evaluating the alternating monotonicity approach. Another work to put effort into is exploiting invariants other than monotonicity, for this would enlarge the scalability of this research.

References

[Lunze] J. Lunze, "Handbook of Hybrid Systems Control: Theory, Tools, Applications", Cambridge University Press, 2009.

[Ueda et al.] Ueda, K., Matsumoto, S., Takeguchi, A., Hosobe, H. and Ishii, D.: "HydLa: A High-Level Language for Hybrid Systems", In Proc. Second Workshop on Logics for System Analysis, 2012, pp.3-17.

[Borning et al.] Borning, A., Freeman-Benson, B. and Wilson, M., "Constraint Hierarchies", Lisp and Symbolic Computation, Vol.5, No.3, 1992, pp.223-270.

[Matsumoto] Shota Matsumoto: "Validated Simulation of Parametric Hybrid Systems Based on Constraints", Graduate School of Fundamental Science and Engineering, Waseda University, Doctoral Thesis, 2017.