Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 317 (2015) 109–115

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

HyLaGI: Symbolic Implementation of a Hybrid Constraint Language HydLa

Shota Matsumoto, Fumihiko Kono, Teruya Kobayashi and Kazunori Ueda [1]

Department of Computer Science and Engineering
Waseda University
Tokyo, Japan

Abstract

A modeling language for hybrid systems HydLa and its implementation HyLaGI are described. HydLa is a constraint-based language that can handle uncertainties of models smoothly. HyLaGI calculates trajectories by symbolic formula manipulation to exclude errors resulting from floating-point arithmetic. HyLaGI features a nondeterministic simulation algorithm so it can calculate all possible qualitative different trajectories of models with uncertainties.

Keywords: hybrid systems, rigorous simulation, constraints, symbolic computation

1 A Constraint-based Language HydLa

Reliability is a key issue in the computation of hybrid systems in which quantitative errors may easily result in qualitatively wrong results. With this in mind, we have developed HydLa [12], a constraint-based modeling language for hybrid systems [8], and its implementation HyLaGI based on symbolic computation. The key feature of HydLa is that constraints are used as the *sole* mechanism for defining both the continuous and discrete behavior of systems. Constraints include ODEs (ordinary differential equations) and temporal logic formulas over reals, as well as logical implication that expresses conditionals and synchronization.

A precursor of HydLa was Hybrid cc [2,5], but its implementation did not completely guarantee the correctness of results. Another approach was CLP(F) [7], constraint logic programming over real-valued functions. It aimed at rigorous simulation and handled interval constraints, but its control structure was very different from that of HydLa.

[1] Email: `{matsusho,kouno,teruya,ueda}@ueda.info.waseda.ac.jp`

http://dx.doi.org/10.1016/j.entcs.2015.10.011
1571-0661/© 2015 The Authors. Published by Elsevier B.V.

```
INIT    <=> y=10 /\ y'=0 /\ x=0
            /\ 0<x'<=20.
FALL    <=> [](y''=-10).
BOUNCE  <=> [](y- =-7
            \/ ((x- <=7 \/ x- >=10) /\ y- =0)
            => y' =-4/5*y'-).
XCONST  <=> [](x''=0).
XBOUNCE <=> []((x- =7 \/ x- =10) /\ y- <0
            => x'=-x'-).

INIT, FALL << BOUNCE, XCONST << XBOUNCE.
```

Fig. 1. A bouncing particle around a hole in HydLa

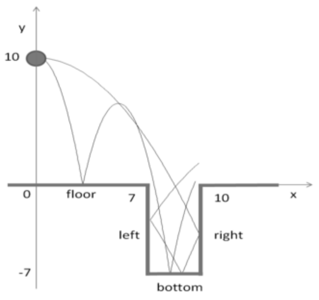

Fig. 2. A bouncing particle around a hole

HydLa allows one to assign priorities to individual constraints in the form of constraint hierarchy [1]. Basic units in a constraint hierarchy are called *constraint modules*. If some of the constraint modules in a HydLa program are inconsistent, constraints with high priority are adopted. In other words, the meaning of a constraint hierarchy is candidate sets of constraint modules which satisfy the priorities. With this feature, programmers don't have to enumerate all possible states manually, while it is necessary in other modeling methods such as hybrid automata [6]. The declarative semantics [12] of a HydLa program is a set of trajectories which satisfy one of maximal consistent candidate sets at each time.

2 Example Programs in HydLa

Figure 1 is a HydLa program that models a particle bouncing around a hole (Fig. 2). The variables `x` and `y` denote the position of the particle. Every variable in HydLa is implicitly a function of time, e.g., `x` means $x(t)$. This program includes five constraint modules `INIT`, `FALL`, `BOUNCE`, `XCONST` and `XBOUNCE`. `INIT` represents the initial state of the model. The operator "`'`" denotes the time derivative of a variable. The range of `x'` is described using two inequalities, which brings uncertainty into this model. `FALL` represents falling by gravity. The operator `[]` is the "always" operator in temporal logic meaning that the constraint is effective on and after the current time. `BOUNCE` represents vertical bounce of the particle. The operator "`=>`" forms a constraint with a guard condition, and a postfix minus sign means the left-hand limit of the variable. The right-hand side of `BOUNCE` says that if the particle reaches the bottom of the hole (`y- =-7`) or the ground around the hole (`(x- <=7 \/ x- >=10) /\ y- =0`), the vertical velocity is multiplied by `-4/5`. `XCONST` says that the horizontal velocity is constant. `XBOUNCE` represents horizontal bounce on the walls in the hole. The bottom line declares the constraint hierarchy of this program: `FALL` is weaker than `BOUNCE` and `XCONST` is weaker than `XBOUNCE`. When the ball reaches the ground or the hole, `FALL` or `XCONST` conflicts with `BOUNCE` or `XBOUNCE`, causing the former to be temporarily removed from the module sets.

HydLa recently featured list comprehensions to concisely describe models with multiple objects. Figure 3 is an example program with list comprehensions. This program includes three constraint modules `INIT`, `COL`, `CONST`, which stand for the initial state, collision between balls and uniform motion. `X` is a list of variables for

```
INIT(b, b0, vb0) <=> b = b0 /\ b' = vb0.
COL(b1, b2 )     <=> [](b1- = b2- => b1' = b2'- & b2' = b1'-).
CONST(b)         <=> [](b'' = 0).

X := {x0..x9}.
INITS := { INIT(X[i], 2*i-2, 0) | i in {2..|X|} }.
COL_HIERARCHY := { (CONST(X[i]), CONST(X[j]) ) << COL(X[i], X[j]) |
                                i in {1..|X|-1}, j in {i+1..|X|} }.

INIT(X[1], 0, 1), INITS, COL_HIERARCHY.
```

Fig. 3. A model of one-dimensional billiard in HydLa

the positions of ten balls. `INITS` is a list of `INIT`, which describes the initial state of the second and the subsequent balls. In the definition of `INITS`, `i` works like a loop variable in procedural languages. `COL_HIERARCHY` is a definition of constraint hierarchy for collision between balls, which states that each `CONST` is weaker than `COL` for the same ball. The bottom line composes constraint hierarchies defined as lists and `INIT` for the first ball. If a HydLa programmer wants to add an additional ball, he or she only has to modify the definition of `X` and specify the initial state of the new ball.

3 HyLaGI: a Symbolic Implementation of HydLa

We have been developing an implementation of HydLa named HyLaGI (HydLa Guaranteed Implementation) [9] [2]. HyLaGI computes trajectories of given HydLa programs. HyLaGI is written in C++ and currently uses Mathematica as its back-end solver. The C++ part consists of the parser and the high-level part of the simulation algorithm including case analysis. Mathematica is used for checking the consistency of conjunctions of constraints, solving ODEs, solving minimization problems and transforming both arithmetic expressions and logical formulas. We have also developed a frontend system for HyLaGI, which can visualize simulation results of HyLaGI by two dimensional plots and animations.

3.1 Symbolic Simulation

HyLaGI performs symbolic simulation, which has two important features.

Firstly, the results are free from errors caused by floating point arithmetic because HyLaGI calculates trajectories by formula manipulation. Outputs are symbolic representation of state variables at each time point and each time interval. Since errors in the simulation of hybrid systems can lead to qualitatively different results, this feature is more important than in ordinary dynamical systems. There are other rigorous tools for hybrid systems, but most of them are based on numerical methods [3,4,13] or theorem proving [10].

Secondly, HyLaGI calculates all possible trajectories for uncertain HydLa programs. This feature is important to handle real-world systems that may deteriorate over time or cannot be modeled precisely. To calculate all trajectories, HyLaGI adopts a nondeterministic simulation algorithm. Figure 4 is an overview of the

[2] available at http://www.ueda.info.waseda.ac.jp/hydla/ with HydLa IDE.

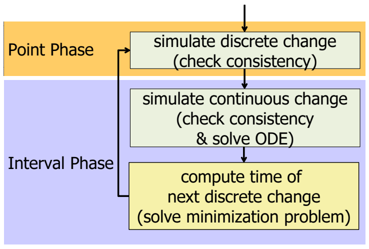
Point Phase
simulate discrete change (check consistency)
Interval Phase
simulate continuous change (check consistency & solve ODE)
compute time of next discrete change (solve minimization problem)

Fig. 4. Algorithm of Symbolic Simulation

(PP1 is Omitted)

```
---------IP 2---------
t : 0->2^(1/2)
x : t*p[x, 1, 1]
y : (t^2+(-2))*(-5)
x': p[x, 1, 1]
y': t*(-10)
x'': 0
y'': -10
```

(The following phases are Omitted)

```
---------parameter condition---------
p[x, 1, 1] : (0, 2^(-1/2)*7)
```

Fig. 5. Output from HyLaGI (Fragment)

algorithm. The simulation repeats *Point Phases* and *Interval Phases* until a termination condition (time limit or the number of phases) is satisfied. At each phase, maximal consistent sets of modules are calculated by checking consistency and guard conditions. A Point Phase is concerned with discrete change at a time point, while an Interval Phase is concerned with continuous change for a time interval. Calculation of Interval Phases is more complex than that of Point Phases because it includes solving ODEs and minimization problems.

To simulate uncertain models symbolically, HyLaGI uses symbolic parameters that denote the uncertain values of variables at certain time points. Figure 5 is a sample output from HyLaGI for the program of Fig. 1. `IP 2` stands for the second Interval Phase and the symbolic parameter `p[x,1,1]` stands for the initial horizontal velocity. This output represents infinitely many continuous trajectories that can be obtained by instantiating `p[x,1,1]`.

Note that the uncertainties represented by symbolic parameters may result in qualitative difference of trajectories. When such a difference arises, HyLaGI automatically performs case analysis, narrowing the range of parameters into each qualitative different case. Case analysis may occur at all boxes in Fig. 4, but in most models it occurs at the computation of the next discrete change because parameter conditions are strongly related to the order and number of discrete changes.

3.2 An Example of Case Analysis

As an example, we show the simulation result of the model of Fig. 1 with an assertion `ASSERT(!(y>=0 /\ x>=10))`, which states that the particle never goes beyond the hole. The time limit is set to 20 units and the number of phases is limited to 13. We obtain counterexamples in the form of parameter conditions of the initial horizontal velocity, which are shown in Table 1. Each row of the table corresponds to each qualitative different case. The condition of each case is represented by symbolic and numeric intervals which the initial horizontal velocity must belong to. Notice that HyLaGI's symbolic simulation distinguishes between open and closed boundaries.

4 Scalability of HyLaGI

One of the concerns with a symbolic technique is its scalability. A naïve implementation of HydLa would lead to the calculation of all constraints at each phase, but

Table 1
parameter conditions where the ball goes beyond the hole

order of bounces	symbolic representation	numeric representation
floor, floor, bottom	$[1250/(405\sqrt{2}+\sqrt{317}+9\sqrt{1387}),$ $1250/(405\sqrt{2}-\sqrt{317}+9\sqrt{1387})]$	$[1.36027, 1.40428]$
floor, floor	$[125\sqrt{2}/97, 35/(13\sqrt{2})]$	$[1.82244, 1.90375]$
floor, bottom	$(35/(13\sqrt{2}),$ $(1125\sqrt{2}+225\sqrt{67}+25\sqrt{197})/(928+81\sqrt{134})]$	$(1.90375, 2.02803]$
floor, right, bottom, left	$[-40\sqrt{197}/(928+81\sqrt{134})+360(5\sqrt{2}+\sqrt{67})$ $/(928+81\sqrt{134}), 25\sqrt{2}/13)$	$[2.64300, 2.71964)$
floor	$[25\sqrt{2}/13, 7/\sqrt{2}]$	$[2.71964, 4.94975]$
bottom, right, left	$((117\sqrt{85}+13\sqrt{485})/256, 10\sqrt{5/17})$	$(5.33196, 5.42326)$
bottom+right, left	$[10\sqrt{5/17}, 10\sqrt{5/17}]$	$[5.42326, 5.42326]$
right, bottom, left	$(10\sqrt{5/17}, (9\sqrt{85}+\sqrt{485})/16]$	$(5.42326, 6.56241]$
no bounce	$[5\sqrt{2}, 20]$	$[7.07107, 20]$

HyLaGI only calculates constraints related to each discrete change. This improvement is based on the following three ideas.

The first idea is to analyze the dependency of constraints. The dependency can be represented by a bipartite graph consisting of variable nodes and constraint nodes. Graph edges correspond to the references of variables from constraints. The dependency between constraints changes dynamically in the simulation of HyLaGI because (i) a guarded constraint may be switched on and off, (ii) constraint modules not chosen for a particular phase have no effect, and (iii) the left-hand limit of a variable in a Point Phase is regarded constant and has no relation with the variable itself. HyLaGI manages the effectivities of nodes and edges of a dependency graph dynamically and calculates minimal sets of related constraints.

The second idea is to exploit the continuity of the values of variables and its derivatives. Variables whose values jump must be referenced in constraints that triggered discrete changes, and HyLaGI keeps other variables change continuously without recalculation.

The third idea is to calculate candidate subsets of constraint modules dynamically on demand. In HydLa, the number of candidate subsets can increase exponentially with respect to the number of objects in the model. For example, it is 2^n for the program of Fig. 3 with n balls. However, the number of subsets to be checked for its maximality is usually small. HyLaGI calculates such subsets on demand by using the information of inconsistent subsets obtained in the process of consistency checking. When a subset is known to be inconsistent, at least one module must be removed from the subset to make it consistent, and HyLaGI removes a low-priority module M and those below M in the constraint hierarchy.

These improvements reduced the time complexity of the calculation of each discrete change. For example, for the program of Fig. 3, it is reduced from exponential (without the third idea) or $O(n^3)$ (with the third idea) to $O(n)$.

5 Current and Future Work

Our experiences has shown that the symbolic approach of HyLaGI shows great affinity with the constraint language HydLa, works well in principle, and brings several interesting advantages summarized as follows:

- rigorous simulation that may be extended to unbounded length and to verification,
- automatic, on-demand case analysis of parameter conditions, and
- calculation and maintenance of the relation between multiple parameters (cf. wrapping effect [11]).

On the other hand, we have found that the current limiting factor of HyLaGI is the power and the efficiency of the minimization problem solver (the third box of Fig. 4) which presumably uses quantifier elimination inside Mathematica: the formulas can become complex as simulation goes on and finally becomes unsolvable. To handle these cases, we started to integrate interval arithmetic into the symbolic simulation of HyLaGI. Intervals are a special form of constraints and show perfect affinity with the symbolic approach. Even when the states can be rigorously represented, it is sometimes useful to approximate them using intervals; it is a matter of tradeoff. We are implementing interval Newton method to reliably solve minimization problems and interval approximation of complex symbolic formulas. Integration of interval arithmetic into symbolic simulation is expected to be more smooth than the opposite direction.

Another limitation of the symbolic approach is that many if not all models with nonlinear ODEs cannot be solved symbolically. Here again, we can over-approximate the solution by a numerical method and regard the approximated solution as formulas with parameters, which is our future work. We hope that HyLaGI will find its place in applications exhibiting subtle behaviors and requiring careful analysis.

References

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

[2] Carlson, B. and Gupta, V. : Hybrid cc with Interval Constraints, In Proc. HSCC, LNCS 1386, Springer, 1998, pp. 80–94.

[3] Chen, X., Abraham, E. and Sankaranarayanan, S. : Flow* : An Analyzer for Non-Linear Hybrid Systems. In Proc. Computer Aided Verification, 2013, pp. 258–263

[4] Frehse, G., Guernic, C. L., Donzé, A., Cotton, S., Ray, R., Lebeltel, Olivier., Ripado, R., Girard, A., Dang, T., Maler, O. : SpaceEx : Scalable Verification of Hybrid Systems, In Proc. Computer Aided Verification, 2011, pp. 379–395.

[5] Gupta, V., Jagadeesan, R., Saraswat, V. and Bobrow, D. : Programming in Hybrid Constraint Languages, in Hybrid Systems II, LNCS 999, Springer, 1995, pp. 226–251.

[6] Henzinger, T. : The Theory of Hybrid Automata, In Proc. LICS'96, IEEE Computer Society Press, 1996, pp. 278–292.

[7] Hickey, T. J. and Wittenberg, D. K. : Rigorous Modeling of Hybrid Systems Using Interval Arithmetic Constraints, In Proc. HSCC 2004, LNCS 2993, Springer-Verlag, 2004, pp. 402–416.

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

[9] Matsumoto, S. and Ueda, K. : Hyrose : A Symbolic Simulator of Hybrid Constraint Language HydLa, Computer Software, Vol.30, No.4, 2013, pp.18–35 (in Japanese).

[10] Platzer, A., Quesel, J.D. : KeYmaera : A Hybrid Theorem Prover for Hybrid Systems. In IJCAR 2008, LNCS 5195, Springer-Verlag, 2008, pp.171–178.

[11] Moore, R. E., Kearfott, R. B., Cloud. M. J. : Introduction to Interval Analysis, Society for Industrial and Applied Mathematics, 2009.

[12] 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.

[13] Zeng, Y., Rose, C., Brauner, P., Taha, W., Masood, J., Philippsen, R.., O'Malley, M. and Cartwright, R. : Modeling Basic Aspects of Cyber-Physical Systems, Part II. In Proc. of The 11th IEEE International Conference on Embedded Software and Systems, 2014.