HydLa: A High-Level Language for Hybrid Systems

Kazunori Ueda[1], Shota Matsumoto[1], Akira Takeguchi[1], Hiroshi Hosobe[2], and Daisuke Ishii[2]

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

[2] National Institute for Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
hosobe@nii.ac.jp, dsksh@acm.org

1 HydLa, A Hybrid Constraint Language

We have been working on the design and implementation of HydLa, a modeling language for hybrid systems [5][3]. The principal feature of HydLa is that it employs constraint-based formalisms both in the modeling and reliable simulation of hybrid systems. We take this approach for two reasons: one is that a constraint-based formalism is non-procedural but yet provides the language with control structures including synchronization and conditionals that are expressive enough to model hybrid systems, and the other is it allows us to handle uncertainties or partial information in a smooth way. Rather few tools for hybrid systems fully exploit constraint-based formalisms. The closest previous work was Hybrid cc [2][3], but HydLa differs in that its implementation ensures the correctness of simulation results. Another constraint-based approach was CLP(F), constraint logic programming over real-valued functions [4]. Both CLP(F) and HydLa aim at rigorous simulation and handle intervals, but they have very different control structures.

HydLa programs are sets of constraint modules that describe static and/or dynamic properties of systems using (among others) ordinary differential equations, implication, and a temporal operator. Constraint modules form constraint hierarchies [1] that define priorities between constraints. In determining the set of trajectories by constraint satisfaction, a maximal consistent subset of the set of constraint modules is taken that satisfies the requirements of HydLa's declarative semantics [5]. Implication and constraint hierarchy govern the change of the set of effective constraints over time. Constraint-based modeling allows high-level description but can easily cause over- and under-constrainedness, but constraint hierarchy provides us with a concise mechanism that makes trajectories well-defined.

[3] The English version of [5] appears in Appendix of this paper.

```
INIT   <=> h=10 /\ h'=0 /\ timer=0.
PARAMS <=> exT=3 /\ volume>3 /\ volume<10 /\ [](exT'=0 /\ volume'=0).

TIME   <=> [](timer'=1).
RESET  <=> [](timer- >=volume+exT => timer=0).

BURN   <=> [](timer- <volume => h''=1).
FALL   <=> [](timer- >=volume => h''=-2).

ASSERT(h>=0).

INIT, PARAMS, BURN, FALL, TIME<<RESET.
```

Fig. 1. Hot-air balloon model in HydLa

2 An Example Model

Figure 1 describes a model of a hot-air balloon going up by using multiple fuel tanks. Each fuel tank lasts `volume` time units and changing it takes `exT` time units. Uppercase names stand for constraint modules, x' stands for the time derivative of x, `[]` stands for the *always* temporal operator, and the postfix minus sign of x- stands for the left limit of x, where each variable is interpreted as a function of time. The first six lines are module definitions: `INIT` defines initial values of `h` and `timer`; `PARAMS` defines the values of the two parameters `exT` and `volume`; `TIME` and `RESET` define the continuous and discrete changes of the variable `timer`, respectively; `BURN` and `FALL` define the two modes of operations. `TIME<<RESET` means `TIME` is superseded by `RESET` when they contradict. Other modules are not superseded by any other modules and are always in effect. Note that the initial value of `volume` is given as an interval constraint. Figure 2 shows possible trajectories of the height `h`, where `volume` $= 3.0, 3.1, \ldots, 10.0$. The actual output from HydLa represents an infinite number of trajectories by using the symbolic parameter `pvolume` (see Section 4), and the trajectories of Fig. 2 were sampled for the purpose of drawing.

Although HydLa is a language for reliable simulation, it comes with an assertion construct as shown in Fig. 1 that can be used for checking simple global properties.

3 Nondeterministic Simulation Algorithm

We have been developing Hyrose, an implementation of HydLa's nondeterministic simulation algorithm given in [6]. The principles of Hyrose are (i) to guarantee the accuracy of answers and (ii) to be able to compute all possible trajectories so that it can be used for reasoning about hybrid systems. Simulation proceeds by successive constraint satisfaction of alternating *point phases* (PP, a.k.a. *jump*) and *interval phases* (IP, a.k.a. *flow*), where phase change is triggered either by the discharging of constraints from implicational constraints or the change of

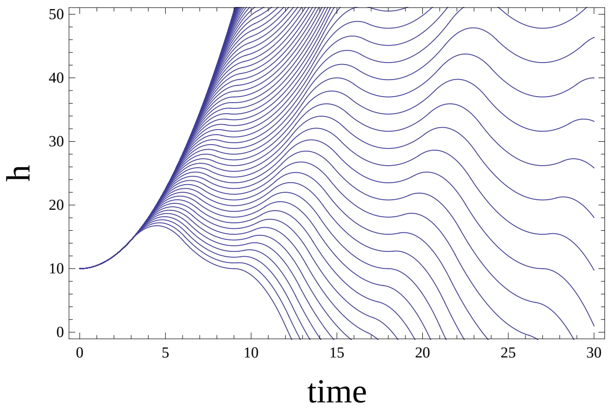

Fig. 2. Trajectories of a hot-air balloon

maximal consistent set of modules. An important feature of the HydLa's simulation algorithm is that it allows models containing symbolic parameters whose values are possibly specified as interval constraints. Uncertainty expressed this way may cause nondeterminism in the truth/falsity of the antecedent of an implicational constraint, in which case the simulation algorithm splits the interval into subintervals that make the antecedent uniformly true and those that make the antecedent uniformly false, and subsequent simulation may pursue all those alternatives. In this way, the algorithm automatically performs case analysis and classifies possible trajectories into qualitatively equivalent groups.

4 Simulating the Hot-Air Balloon Model

Hyrose is currently based on symbolic computation, though it also employs interval computation to be able to compare two concrete or parametric values rigorously. Figure 3 shows a fragment of the execution result (for 30 time units) of the hot-air balloon model without the `ASSERT` check. It shows the third point phase at time $3 + \texttt{pvolume}$ and the third interval phase of time $(3 + \texttt{pvolume}, 3 + 2 \cdot \texttt{pvolume})$ of the case $\texttt{pvolume} \in [9/2, 21/4)$, where `t` is the current time and `pvolume` is a symbolic parameter introduced by Hyrose to represent the initial values of `volume`. For this model, Hyrose returned a total of six cases which differed only in the number of phases within the simulation time. Hyrose's automatic case analysis can handle multiple symbolic parameters for this example, while the power of automatic case analysis depends on the underlying constraint solver (which can be chosen from Mathematica and REDUCE currently). Figure 4 shows five qualitatively different cases that may happen in 5 time units of simulation with $\texttt{volume} \in (1, 3)$ and $\texttt{exT} \in (2, 4)$. Zones marked as "assertion failed" violate the constraint $h \geq 0$. "PPn" means that n point phases have been encountered in the simulation.

```
#---------3---------
---------PP---------
time     : 3+pvolume
exT      : 3
h        : 1/2*(2+6*pvolume+pvolume^2)
timer    : 0
volume   : pvolume
exT'     : 0
h'       : -6+pvolume
timer'   : UNDEF
volume'  : 0
h''      : -2

---------IP---------
time     : 3+pvolume -> 3+2*pvolume
exT      : 3
h        : 1/2*(47+18*pvolume+(-18)*t+t^2)
timer    : -3+(-1)*pvolume+t
volume   : pvolume
exT'     : 0
h'       : -9+t
timer'   : 1
volume'  : 0
h''      : 1

#---------parameter condition---------
pvolume : [9/2, 21/4)
```

Fig. 3. Output from Hyrose (fragment)

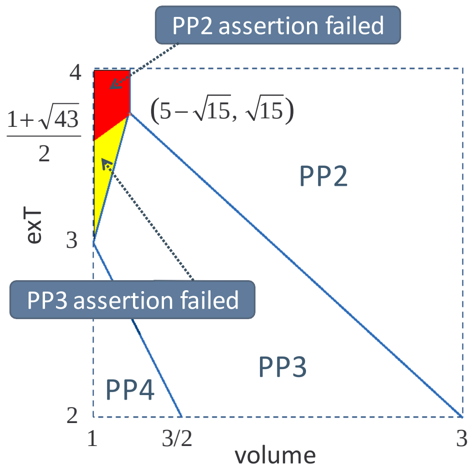

Fig. 4. Classifying two-dimensional parameter space

Other applications of hybrid systems with parameters or uncertainties include analysis of systems with singular points and sensitivity analysis. Hyrose is still in its initial stage and the size of the systems it can handle is limited by the underlying constraint solver, but it is beginning to show the viability of HydLa's constraint-based simulation algorithm and is gaining a role complementary to other tools aiming at simulating and analyzing large hybrid systems.

Acknowledgments We are indebted to the other members of the HydLa group for their contribution to the implementation and daily discussions. This research is partially supported by Grant-In-Aid for Scientific Research ((B) 23300011), JSPS, Japan.

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, C. and Gupta, V., Hybrid cc with Interval Constraints, in *Proc. HSCC'98*, LNCS 1386, Springer-Verlag, 1998, pp. 80–94.
3. Gupta, V., Jagadeesan, R., Saraswat, V., and Bobrow, D., Programming in Hybrid Constraint Languages, in *Hybrid Systems II*, LNCS 999, Springer-Verlag, 1995, pp. 226–251.
4. 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.
5. Ueda, K., Hosobe, H. and Ishii, D., Declarative Semantics of the Hybrid Constraint Language HydLa, *Computer Software*, Vol. 28, No. 1, 2011, pp. 306–311 (in Japanese). The English version appears in Appendix of this paper.
6. Shibuya, S., Takata, K., Hosobe, H. and Ueda, K., An Execution Algorithm for the Hybrid System Modeling Language HydLa, *Computer Software*, Vol. 28, No. 3, 2011, pp. 167–172 (in Japanese), available online at `https://www.jstage.jst.go.jp/article/jssst/28/3/28_3_3_167/_article/`.

Appendix:
Declarative Semantics of the Hybrid Constraint Language HydLa *

Kazunori Ueda[1], Hiroshi Hosobe[2], and Daisuke Ishii[3]

[1] Department of Computer Science and Engineering, Waseda University
[2] National Institute of Informatics
[3] University of Nantes

Abstract. Hybrid systems are dynamical systems with continuous evolution of states and discrete evolution of states and governing equations. We have been working on the design and implementation of HydLa, a constraint-based modeling language for hybrid systems, with a view to the proper handling of uncertainties and the integration of simulation and verification. HydLa's constraint hierarchies facilitate the description of constraints with adequate strength, but its semantical foundations are not obvious due to the interaction of various language constructs. This paper gives the declarative semantics of HydLa and discusses its properties and consequences by means of examples.

A.1 Introduction

Hybrid systems are dynamical systems with continuous evolution of states and discrete evolution of states and governing equations. We have been developing a modeling framework of hybrid systems based on the notion of Constraint Programming. Our goal is to establish a constraint-based paradigm in which (i) to describe diverse phenomena found in physical, cyber-physical, and biological systems using logical formulae involving equations and inequations and (ii) to solve or verify them using search techniques represented by constraint propagation.

Our motivation has been to establish, in the field of hybrid systems, a declarative programming paradigm that directly handles as source programs high-level description of problems in mathematical and logical formulas, as opposed to traditional formalisms based on automata and Petri Nets [4][1]. A similar approach was first taken by Hybrid CC [3], and we have made a lot of experiments on Hybrid CC programming. However, we found that it was not necessarily straightforward to specify constraints that a system consisting of alternate discrete and continuous phases should satisfy, and this lead us to design a new language that enables a concise description of hybrid systems.

* This is an English translation of the paper that appeared in *Computer Software*, Vol. 28, No. 3 (2011), pp.167–172, available online at
`https://www.jstage.jst.go.jp/article/jssst/28/1/28_1_1_306/_article/`.

```
   INIT ⇔ ht=10 ∧ ht'=0.
 PARAMS ⇔ □(g=9.8 ∧ c=0.5).
   FALL ⇔ □(ht''=-g).
 BOUNCE ⇔ □(ht- =0 ⇒ ht'= -c*(ht'-)).
 INIT, PARAMS, (FALL << BOUNCE).
```

Fig. A.1. A bouncing ball

Since the basic design of HydLa was established in 2008 [9], we studied the details of the language through the description of a number of examples [5], developed a simulation algorithm [8] and a prototype implementation, and explored technologies for implementing discrete changes with guaranteed accuracy [6]. All those studies contributed to the clarification of the essence and subtle points of the HydLa language specification. Based on those experiences, this paper formulates the declarative semantics of the core of HydLa, and discusses its descriptive power and properties by means of examples.

A.2 Overview of HydLa

HydLa is a declarative language for hybrid systems. Its objective is to allow one to provide the mathematical formulation of a given problem with minimal modification and to simulate or analyze them. For the design principles and related work of HydLa, the readers are referred to [9].

Dynamical systems that HydLa aims to handle are in general represented as a countable number of real-valued functions $x_1(t)$, $x_2(t)$, $\ldots (t \geq 0)$ that include integer-valued functions as a special case. A HydLa program imposes constraints on the behavior of those functions (hereafter called *trajectories*) that may cause continuous or discrete changes over time. The declarative semantics of a HydLa program P is defined as a satisfaction relation between trajectories $\overline{x}(t) = \{x_i(t)\}_{i\geq 1}$ and P, or equivalently, the set of all $\overline{x}(t)$'s that satisfy P.

In order to describe hybrid systems in a concise manner, the use of hierarchies to represent *defaults* and *exceptions* will play an important role exactly as in knowledge representation and object-oriented design. Consider a ball bouncing on a floor. The change of the velocity of the ball is determined by the gravity most of the time (default), while it is determined by the collision equation when the ball hits the floor (exception). A mathematically concise way to describe solution trajectories of such systems in a well-defined matter would be to introduce partial order between candidate sets of equations that the system should satisfy and to take a maximally consistent element of the partially ordered set (poset) of sets of constraints. HydLa's design principle is exactly based on this idea.

Figure A.1 shows the description of a bouncing ball in HydLa. The first four lines are the definition of constraint modules. Constraint modules are program

(program)	P	$::= (MS, DS)$
(module set)	MS	$::=$ poset of sets of M
(definitions)	DS	$::=$ set of D whose elements have different left-hand sides
(definition)	D	$::= M \Leftrightarrow C$
(constraint)	C	$::= A \mid C \wedge C \mid G \Rightarrow C \mid \Box C \mid \exists x.C$
(guard)	G	$::= A \mid G \wedge G$
(atomic constraint)	A	$::= E\ \mathit{relop}\ E$
(expression)	E	$::=$ ordinary expressions $\mid E' \mid E-$

Fig. A.2. Syntax of the Basic HydLa.

units which are combined to form a set of constraints and to which priorities may be given. In the right-hand side constraints, `'` stands for a time derivative, the postfix minus sign stands for the left-side limit of a trajectory, and $\Box$ stands for an *always* temporal operator. All the constraints stand for constraints at time 0. However, since the constraints other than `INIT` start with $\Box$, they hold at all time points on and after time 0. A constraint with an implication (such as `BOUNCE`) is called a *conditional constraint*. A conditional constraint prefixed by an $\Box$ imposes its consequent exactly when its antecedent (guard) holds.

The final line combines the four constraint modules. A comma stands for composition without priorities, while `<<` gives a higher priority to `BOUNCE` than to `FALL`. In this example, all the four constraints are taken when the ball is in the air, while {`INIT`, `PARAMS`, `BOUNCE`} will be taken as the maximally consistent set when the ball hits the floor because `FALL` and `BOUNCE` become inconsistent.

This example is known to exhibit a Zeno behavior, an infinite number of discrete changes within a finite amount of time, beyond which the simulation normally does not proceed.

A.3 Basic HydLa

We consider the semantics of the *Basic HydLa* whose syntax is shown in Fig. A.2. Basic HydLa simplifies HydLa [9] as follows:

1. For each time point, HydLa chooses a consistent set of constraint modules that satisfies the priority constraint and that is maximal with respect to the set inclusion relation between constraint modules. More specifically, from a relative priority relation between constraint modules, HydLa first derives a poset whose elements are admissible (with regard to constraint priorities) sets of all the subsets of constraint modules [5], and then chooses a maximal consistent element. Basic HydLa does not handle this derivation but assumes that the "(irreflexive) poset of sets of constraint modules" is directly given in a program together with the definitions of constraint modules. *Default* constraints such as the continuity of trajectories (frame axioms, see Section A.6.1) are to be explicitly specified within this poset. The constraints

at the top of a constraint hierarchy should often be treated as *required* constraints that *must* be adopted, and whether to do so can be expressed explicitly within the poset.

2. Basic HydLa does not support the time shift (i.e. delay) operator ˆ. We can use the feature explained in the next item instead.
3. To enable dynamic creation of trajectories, Basic HydLa introduces an existential quantifier $\exists$ for local variable creation. This enables us to dynamically create a timer with which to represent a delay between the detection of some condition and the issue of a new constraint.
4. Basic HydLa does not support program definitions since they can be simply inlined.
5. For the same reason, Basic HydLa does not support the operator $\forall$ to generate a family of trajectories.

We assume that a Basic HydLa program (MS, DS) satisfies $\bigcup MS \subseteq \mathrm{dom}(DS)$, where $\bigcup MS$ is the set of modules appearing in MS and $\mathrm{dom}(DS)$ is the set of left-hand sides of DS. In the following, we consider a set DS of constraint module definitions as a function from module names to constraints.

As shown in Fig. A.2, we restrict the guard constraints to atomic constraints and their conjunctions. HydLa does not specify the class of constraints that can be described in a program. In this sense, HydLa is a *language scheme* that parameterizes constraint systems. The reason why we allow only $\Box$ as a temporal operator is that our syntax is targeted at the *modeling* of systems. Other temporal operators such as $\Diamond$ will be included in the specification language when we construct a verification system that use HydLa as a modeling language.

A.4 Declarative Semantics of Basic HydLa

As shown in Section A.2, the declarative semantics of HydLa is defined as a relation meaning that a given trajectory (or *interpretation*) satisfies a program (or *specification*). The information to be maintained by the declarative semantics depends on design criteria such as what class of programs it deals with and what degree of compositionality (i.e., the ability to compose the overall semantics from the semantics of components) it aims at. The semantics in [9] dealt with programs containing no $\Box$ operators in the consequents of conditional constraints. Parameters and behaviors of systems with a finite number of components and no delays can be described by constraints with $\Box$'s only in their prenex positions. When those programs contain conditional constraints, their consequents hold exactly when the antecedents hold, which means that a maximal consistent set of constraints can be chosen at that time.

However, a constraint whose consequent includes an $\Box$ leaves the consequent as a candidate for choice even after the corresponding antecedent ceases to hold. If we have to judge which consequents of constraints should be chosen in the future when the corresponding antecedents held, it would be a lookahead of the future. Thus the choice of a maximal consistent set must be performed not when constraints are discharged but when the constraints are actually applied. Therefore we further refine our semantics in the following way.

First, we identify a conjunction of constraints with a set of constraints; i.e., we view the syntax of a constraint in Fig. A.2 as

$$C ::= \{A\} \mid C \cup C \mid \{G \Rightarrow C\} \mid \{\Box C\} \mid \{\exists x.C\},$$

and also allow an empty set. By Skolemization, we recursively eliminate existential quantifiers $\exists$ except for those occurring in the consequents of conditional constraints.

Next, we consider constraint sets as functions of time. For example, a constraint C that occurs in a program is regarded as a function $C(0) = C$, $C(t) = \{\}$ $(t > 0)$.

For a constraint $C(t)$ that is a function of time, the $\Box$-closure $C^*(t)$ is defined as a function that satisfies the following properties:

- (Extension) $\forall t(C(t) \subseteq C^*(t))$;
- ($\Box$-closure) $\forall t(\Box a \in C^*(t) \Rightarrow \forall t' \geq t\,(a \subseteq C^*(t')))$;
- (Minimality) For each t, $C^*(t)$ is the minimum set that satisfies the above two conditions.

For $C = \{\texttt{f=0}, \Box\{\texttt{f'=1}\}\}$ for example, we have $C^*(0) = \{\texttt{f=0}, \texttt{f'=1}, \Box\{\texttt{f'=1}\}\}$, $C^*(t) = \{\texttt{f'=1}\}$ $(t > 0)$.

The constraint set that determines a solution trajectory of a HydLa program may change over time for two reasons: one is that a maximal consistent set may change; the other is that the consequent of a conditional constraint is newly added when its antecedent holds. The choice of a maximal consistent set in the former case is performed independently at each time point. By contrast, when the program has a constraint whose consequent begins with $\Box$, whether the constraint is active or not depends on whether its antecedent has been activated *in the past*; hence the state of a system should maintain the activation history of the antecedents. Therefore it is appropriate to consider a satisfaction relation stating that a program $P = (MS, DS)$ is satisfied by a pair $\langle \overline{x}, Q\rangle$ of a solution trajectory $\overline{x} = \overline{x}(t)$ and the constraint module definition $Q = Q(M)(t)$ $(M \in \mathrm{dom}(DS))$ recording the activation of antecedents. We define this relation as shown in Fig. A.3.

The principle of the declarative semantics in Fig. A.3 is the consistency-based adoption of constraints. It requires that, at each time point, a consistent set of constraint modules with a maximal preference must be adopted and satisfied.

Condition (i) requires $Q(M)$ to satisfy the $\Box$-closure property, and Condition (ii) requires $Q(M) = Q(M)^*$ to be an extension of $DS(M)^*$. Now we look into Condition (iii) closely. The order of the quantifiers at Line (s0) allows $\overline{x}$ to choose, at each time point, a different set of candidate modules from the constraint hierarchy. Line (s1) means that, at time t, $\overline{x}$ satisfies some set of candidate modules in the constraint hierarchy. Lines (s2) mean that there is no trajectory $\overline{x}'$ that behaves exactly as $\overline{x}$ before t and satisfies a better candidate module set than $\overline{x}$ at t. Lines (s3) mean that, when the antecedent of a chosen conditional

$$
\begin{array}{ll}
\langle \overline{x}, Q \rangle \models (MS, DS) \Leftrightarrow \text{(i)} \wedge \text{(ii)} \wedge \text{(iii)} \wedge \text{(iv), where} & \\
\text{(i) } \forall M\, (Q(M) = Q(M)^*); & \\
\text{(ii) } \forall M\, (DS(M)^* \subseteq Q(M)); & \\
\text{(iii) } \forall t\, \exists E \in MS\, (& \text{(s0)} \\
\qquad (\overline{x}(t) \Rightarrow \{Q(M)(t) \mid M \in E\}) & \text{(s1)} \\
\quad \wedge\ \neg \exists \overline{x}'\, \exists E' \in MS\, (& \text{(s2)} \\
\qquad\quad \forall t' < t\, (\overline{x}'(t') = \overline{x}(t')) & \text{(s2)} \\
\qquad\quad \wedge\ E \prec E' & \text{(s2)} \\
\qquad\quad \wedge\ \overline{x}'(t) \Rightarrow \{Q(M)(t) \mid M \in E'\}) & \text{(s2)} \\
\quad \wedge\, \forall d\, \forall e\, \forall M \in E\, (& \text{(s3)} \\
\qquad (\overline{x}(t) \Rightarrow d) \wedge ((d \Rightarrow e) \in Q(M)(t)) \Rightarrow e \subseteq Q(M)(t))); & \text{(s3)} \\
\text{(iv) For each } M \text{ and } t,\ Q(M)(t) \text{ is the minimum set} & \\
\qquad \text{that satisfies (i)–(iii).} &
\end{array}
$$

Fig. A.3. Definition of $\langle \overline{x}, Q \rangle \models P$

constraint holds, Q is extended by expanding its consequent into the definition of the corresponding module M in Q. If a member of the consequent (regarded as a set of constraints) begins with $\Box$, it is further expanded by the $\Box$-closure condition (i). Also, if it begins with $\exists$, the quantifier is eliminated by using a Skolem function. Condition (iv) requires the minimality.

A.5 Examples

Using simple examples, we explain how the declarative semantics actually defines solution trajectories and the constraint sets used to determine them.

Example 1: The first example shows how the arrival of a monotonically increasing function at a certain threshold is reflected to another function with a delay.

$$
\begin{aligned}
P_1 &= (MS_1, DS_1) \\
MS_1 &= (\{\{\mathtt{A,C}\}, \{\mathtt{A,B,C}\}\}, \{\{\mathtt{A,C}\} \prec \{\mathtt{A,B,C}\}\}) \\
DS_1 &= \{\ \mathtt{A} \Leftrightarrow \mathtt{f=0} \wedge \Box(\mathtt{f'=1}), \\
&\qquad \mathtt{B} \Leftrightarrow \Box(\mathtt{g=0}), \\
&\qquad \mathtt{C} \Leftrightarrow \Box(\mathtt{f=5} \Rightarrow \exists \mathtt{a}\,.\,(\mathtt{a=0} \wedge \Box(\mathtt{a'=1}) \wedge \Box(\mathtt{a=2} \Rightarrow \mathtt{g=1})))\}
\end{aligned}
$$

Here, `f` is a function that expresses the current time, `a` is a timer invoked by `f=5` as the trigger, and `g` is a pulse function that is usually 0 but momentarily becomes 1 two seconds after the invocation of the timer. The solution trajectory $\overline{x}$ expresses those behaviors of `f`, `a` (whose Skolem function is also called `a` here), and

g. Now we see all the constraints $Q(*)(t) = \bigcup\{Q(M)(t) \mid M \in \mathrm{dom}(DS_1)\}$ that are stored in Q. At $0 < t < 5$, $Q(*)(t)$ consists of f'=1, g=0, and the constraint C with the leftmost $\Box$ removed. At $t = 5$, a=0, $\Box$(a'=1), a'=1, $\Box$(a=2 $\Rightarrow$ g=1), and a=2 $\Rightarrow$ g=1 are added to them. At $5 < t < 7$, a=0, $\Box$(a'=1), and $\Box$(a=2 $\Rightarrow$ g=1) are removed. At $t = 7$, g=1 replaces g=0. At $t > 7$, g=1 is replaced by g=0 again; the other constraints that remain are f'=1, a'=1 and the two conditional constraints.

Example 2: The declarative semantics presented in the previous section disallows the propagation of constraints to the past. This may be obvious from the construction of the semantics, but we confirm it by using an example since it is an important property.

$$\begin{aligned} P_2 &= ((\mathcal{P}(\{\mathtt{D},\mathtt{E},\mathtt{F}\}), \subsetneq), DS_2\,) \\ DS_2 &= \{\, \mathtt{D} \;\Leftrightarrow\; \mathtt{y{=}0}, \\ &\quad\;\; \mathtt{E} \;\Leftrightarrow\; \Box(\mathtt{y'{=}1} \wedge \mathtt{x'{=}0}), \\ &\quad\;\; \mathtt{F} \;\Leftrightarrow\; \Box(\mathtt{y{=}5} \Rightarrow \mathtt{x{=}1})\} \end{aligned}$$

P_2 leaves the initial value of x undefined. We check whether the constraint x=1 imposed by F at $t = 5$ propagates to the past by the effect of x'=0 in E. We consider the following three cases as candidates for solution trajectories.

1. y$(t) = t$ $(t \geq 0)$ and x$(t) = 1$ $(t \geq 0)$ satisfy all the constraints D, E, and F at all times.
2. y$(t) = t$ $(t \geq 0)$ and x$(t) = 2$ $(t \geq 0)$ satisfy all the constraints except at $t = 5$ and satisfy D and E at $t = 5$.
3. y$(t) = t$ $(t \geq 0)$, x$(t) = 2$ $(t < 5)$, and x$(t) = 1$ $(t \geq 5)$ satisfy all the constraints except at $t = 5$ and satisfy D and F at $t = 5$.

Case 1 is a solution since it obviously satisfies the maximality. Cases 2 and 3 obviously satisfy the maximality except at $t = 5$, and there are no better solutions than these. Neither of them is worse than the other at $t = 5$, and there are no other solutions that satisfy all the constraints; hence both of them are maximal. In other words, any of Cases 1 to 3 is a result of "extending a solution along the time axis so the maximality is satisfied," and is therefore a solution to P_2.

A.6 Discussions on the Specification and the Semantics of the Language

A.6.1 Differential Constraints

The basic principle of HydLa to utilize existing mathematical and logical notations as much as possible suggests that the precise meaning of the notations should also conform to mathematical conventions. For example, at the time point where a piecewise continuous trajectory causes a discrete change, we do not consider the trajectory differentiable even if it is differentiable both from the left and the right, and we do not deactivate the differential constraints at that time point. We also assume only the right continuity and right differentiability at the initial time.

For the reasons above, the priority of the differential constraints of a piecewise continuous function should in general be lower than that of the constraints describing discrete changes. On the other hand, for a continuous trajectory after a discrete change to be well-defined as an initial value problem of an ordinary differential equation, we need to assume the right continuity at the time of the discrete change. Since the differential constraints are deactivated when a discrete change occurs, we also require left continuity to be able to decide the value of a trajectory. Accordingly, HydLa assumes both the right and the left continuity of trajectories described by differential constraints. Since these two continuity constraints are automatically entailed whenever a trajectory is differentiable, we assume them separately with a priority higher than differential constraints.

A.6.2 Expressive Power of HydLa

Although the primary purpose of HydLa is to describe piecewise continuous trajectories, we can define various trajectories or sets of trajectories using HydLa's constraints and constraint hierarchies.

Trajectories defined without using differential equations. HydLa allows us to describe trajectories without using differential constraints. For example, a drifting parameter can be described by a constraint $\Box(\texttt{0.9<a} \wedge \texttt{a<1.1})$, which represents the set of all trajectories whose range is $(0.9, 1.1)$.

Note that a trajectory defined by the above constraint may not be continuous. Hence, a trajectory defined by $\texttt{f=0} \wedge \Box(\texttt{f'=1})$ is not guaranteed to satisfy `f=a` between time 0.9 and 1.1. By adding a constraint $\Box(\texttt{a'=b})$ (we do not add any constraint for `b`), `a` stands for a set of all continuous and differentiable trajectories whose range is $(0.9, 1.1)$, and is guaranteed to intersect with `f`.

A pulse function is another example defined without differential constraints. An example of a pulse function is `g` of Example 1 (Section A.5). Pulse functions play a significant role in representing the occurrences of events. Since pulse functions are not right-continuous at the time of discrete changes, we conjecture that a trajectory after the discrete change cannot be defined directly by a differential equation. The following example shows that our attempt to define a pulse function `b` fails:

$$
\begin{aligned}
P_3 &= (MS_3, DS_3\,) \\
MS_3 &= (\{\{\texttt{G,J}\}, \{\texttt{G,H,J}\}\}, \{\{\texttt{G,J}\} \prec \{\texttt{G,H,J}\}\}) \\
DS_3 &= \{\, \texttt{G} \;\Leftrightarrow\; \texttt{a=0} \wedge \texttt{b=0} \wedge \Box(\texttt{a'=1}), \\
&\qquad \texttt{H} \;\Leftrightarrow\; \Box(\texttt{b'=0}), \\
&\qquad \texttt{J} \;\Leftrightarrow\; \Box(\texttt{a-=1} \Rightarrow \texttt{b=1}) \wedge \Box(\texttt{b-=1} \Rightarrow \texttt{b=0})\}
\end{aligned}
$$

Based on the discussion in Section A.6.1, between two sets of constraint modules in MS_3, there exist several sets with additional constraints on the continuity including the right continuity of `b`. At $t = 1$, the set $\{\mathtt{G,H,J}\}$ is not satisfiable but $\{\mathtt{G,J}\}$ with the right continuity of `b` is satisfiable, and $\mathtt{b}(1) = 1$ holds from the first constraint of `J`. However, then, the greatest lower bound of the time when the guard of the second constraint of `J` holds is $t = 1$. The consequent of the constraint $\mathtt{b}(t) = 0$ is thus activated at $t > 1$ and contradicts the right continuity. Now suppose we drop the assumption of the right continuity. Then it turns that $\mathtt{b}(t) = c\ (t > 1)$ is consistent for all $c \neq 1$. Therefore, although there exists a solution trajectory, HydLa fails to guarantee its uniqueness.

Zeno behaviors Let us reinvestigate the bouncing ball example in Section A.2 based on the declarative semantics of Section A.4. Although the program in Fig. A.1 specifies a unique solution trajectory until the Zeno time, after that, it allows a trajectory that falls through the floor. We need some additional rules to specify the behavior after the Zeno time [10]. In HydLa, we can define it as $\Box(\mathtt{ht-=0} \wedge \mathtt{ht'-=0} \Rightarrow \Box(\mathtt{ht=0}))$, though checking the guard condition would need a special simulation method, e.g., in [7].

The following program shows another method for detecting the Zeno time. It checks the convergence of a function `vmax` that holds the velocity at the last bounce.

$$\Box(\mathtt{vmax'=0}) \ll \quad \Box(\mathtt{ht'-!=ht'} \Rightarrow \mathtt{vmax=ht'}) \wedge \Box(\mathtt{vmax-=0} \Rightarrow \Box(\mathtt{ht=0}))$$

This example shows that the left limit operator `-` is also useful for a function that only causes discrete changes.

A.7 Conclusions and Future Work

This paper gave the declarative semantics of HydLa, a hybrid constraint language with hierarchical structure, described its mechanisms and consequences by means of examples, and discussed the language features and expressive power.

The semantics given in this paper regards trajectories as functions over time. On the other hand, the theory of hybrid systems often adopts hybrid time that allows more than one discrete change at a single time point [2]. One of the motivations of hybrid time is to model computation involving multiple steps at the time of a single discrete change. However, because HydLa is constraint-based, such evolution can be represented as constraint propagation rather than state changes. Another motivation of hybrid time is to deal with the stability and convergence of trajectories in a declarative framework. This would require the extension of our semantics with hybrid time, which is a topic of future work.

We are currently working on the formulation and its implementation of a simulation algorithm corresponding to our declarative semantics. The resulting system is planned to exploit the flexibility of constraint programming and the affinity to interval computation.

Acknowledgments The design of HydLa has received constant feedback from the theoretical and implementation work and daily discussions of the past and present members of the HydLa group. This research is partially supported by Grant-In-Aid for Scientific Research ((B) 20300013), JSPS, Japan.

References

1. David, R. and Alla, H.: On Hybrid Petri Nets, *Discrete Event Dynamic Systems*, Vol. 11, No. 1–2 (2001), pp. 9–40.
2. Goebel, R., Sanfelice, R. G., Teel, A. R.: Hybrid Dynamical Systems, *IEEE Control Systems Magazine*, Vol. 29, No. 2 (2009), pp. 28–93.
3. Gupta, V., Jagadeesan, R., Saraswat, V. and Bobrow, D.: Programming in Hybrid Constraint Languages, in *Hybrid Systems II*, LNCS 999, Springer-Verlag, 1995, pp. 226–251.
4. Henzinger, T. A.: The Theory of Hybrid Automata, in *Proc. LICS'96*, 1996, pp. 278–292.
5. Hirose, K., Otani, J., Ishii, D., Hosobe, H. and Ueda, K.: Modeling techniques of Hybrid Systems using Constraint Hierarchies, in *Proc. 26th Annual Conference of Japan Society for Software Science and Technology*, 2D-2, 2010 (in Japanese).
6. Ishii, D., Ueda, K., Hosobe, H., Goldsztejn, A: Interval-based Solving of Hybrid Constraint Systems, in *Proc. ADHS'09*, pp. 144–149, 2009.
7. Ohno, Y., Ishii, D. and Ueda, K.: A Method of Deriving Zeno States in Hybrid Systems using Formula Manipulation and Quantifier Elimination, in *Proc. 22nd Annual Conference of the Japanese Society for Artificial Intelligence*, 1D1-3, 2008 (in Japanese).
8. Shibuya, S., Takata, K., Hosobe, H. and Ueda, K.: An Execution Algorithm for the Hybrid System Modeling Language HydLa, *Computer Software*, Vol. 28, No. 3 (2011), pp. 167–172 (in Japanese), available online at `https://www.jstage.jst.go.jp/article/jssst/28/3/28_3_3_167/_article/`.
9. Ueda, K., Ishii, D. and Hosobe, H.: Constraint-based Hybrid System Modeling Language HydLa, in Proc. 5th Symposium on System Verification, Research Center for Verification and Semantics, AIST, 2008, pp. 1–6 (in Japanese).
10. Zheng, H., Lee, E. A., Ames, A. D.: Beyond Zeno: Get on with It!, in *Proc. HSCC 2006*, LNCS 3927, Springer-Verlag, pp. 568–582, 2006.