2020 年度 修士論文

ハイブリッドシステムモデリング言語 HydLa の宣言的意味論の精密化とその形式的検証

Refinement and Formal Verification of the Declarative Semantics of Hydla: A Hybrid System Modeling Language

提出日： 2021 年 1 月 25 日
指導： 上田 和紀 教授
研究指導名： 並列知識情報処理研究

早稲田大学 基幹理工学研究科
情報理工・情報通信専攻

学籍番号：5119F105-5
山田 悠之介

概要

ハイブリッドシステムは離散変化と連続変化を繰り返すシステムであり，力学系，回路，制御系など幅広い分野のシステムを表現することができる．HydLa はハイブリッドシステムモデリング言語であり，制約の宣言と階層によって簡潔な記述が可能である．HyLaGI は HydLa の処理系で，記号計算による誤差のない計算が特徴である．HydLa と HyLaGI において解はそれぞれ宣言的意味論と操作的意味論により与えられている．

意味論の応用として意味論より得られる解集合を比較することで異なるプログラムの等価性を判定することができる．またプログラムに対して特定の変換を考えると，変換の前後で解集合が変わらなければその変換は安全であることがわかる．一方でこれまでに HydLa において意味論を用いた研究はなかったため，まずは意味論自体の再検討も必要である．また HydLa の宣言的意味論は複雑であるため，紙とペンでの証明ではなく定理証明支援系を用いた形式的手法によって解計算を検証することも必要である．

本研究ではまず既存の意味論を例題で検討した．検討の結果，軌道の再定義および暗黙の連続性の扱い方の改良が必要であることがわかった．これを受け，値域を拡張した軌道およびその極限，連続性，微分などを定義した．特に「ある点での繋がり方」は宣言的意味論で重要である．そして，暗黙の連続性を動的に扱う宣言的意味論を提案した．また提案した意味論によって例題に一意に解が定まること，およびその系としてプログラムの等価性を証明した．

また再定義した軌道や意味論について，定理証明支援系 Coq を用いた実装と検証を行った．軌道については大きく二つのことを示した．一つは，値が実数値をとる区間では極限，連続性，微分を通常の実数上関数と同様に考えて良いということである．二つ目として，値が $\bot$ になる点では左連続性，右連続性，連続性は成り立たず，微分も値が $\bot$ になることを示した．意味論については，パラメータを含む例題に対して同じパラメータを含むある軌道が解となること，およびその系としてパラメータに具体値を代入した任意のプログラムでの解計算を証明した．

Abstract

Hybrid systems are systems that include both continuous and discrete changes. Hybrid systems can represent a wide range of systems, including dynamical systems, circuits, and control systems. HydLa is a hybrid system modeling language that enables concise descriptions through constraint declarations and constraint hierarchies. HyLaGI, an implementation of HydLa, featured error-free computation with symbolic parameters. In HydLa and HyLaGI, solutions are obtained by declarative semantics and operational semantics, respectively.

As an application of declarative semantics, we can prove the equivalence of different programs by comparing solution sets obtained from the semantics. In addition, when we consider a particular program transformation, we can prove that the transformation is safe if the solution set does not change after the transformation. On the other hand, since there has been no theoretical study using the declarative semantics of HydLa, it is necessary to re-examine the semantics itself. Because of the complexity of HydLa's declarative semantics, it is also necessary to verify the calculation of solutions formally using proof assistants instead of pen-and-paper proofs.

In this research, we first examined the existing declarative semantics with example programs. As a result, we found that the redefinition of trajectory and improving the way implicit continuity is handled are needed. Based on this result, we defined trajectories with an extended codomain and their limits, continuity, differentiation, and so on. Then, we proposed a declarative semantics that deals implicit continuity dynamically. Also, we proved that the proposed semantics uniquely determines a solution to a deterministic example program, and as a corollary, the programs are equivalent.

We also implemented and verified the redefined trajectories and the declarative semantics using Coq, a proof assistant. We proved two kinds of properties about trajectories. First, in intervals not containing $\bot$, limits, continuity, and differentiation are the same as those of ordinary functions on real numbers. Second, at the point where the value is $\bot$, we proved that left continuity, right continuity, and continuity do not hold, and the derivative also has the value $\bot$. For the declarative semantics, we proved that certain parameterized trajectories are solutions to an example program containing a parameter, and as a corollary, one of the trajectories is a solution to any program in which the parameter is instantiated to concrete values.

目次

図目次

第 1 章

はじめに

1.1 背景と目的

自動運転や IoT のように物理的な現実空間とコンピュータ上の仮想空間の間の相互作用で成り立つシステムへの関心が高まっている．このようなシステムは典型的には，時間進行に伴う現実空間の滑らかな状態変化（連続変化）と，条件が成立したときに発生する内部状態の変化（離散変化）を繰り返す．この離散変化と連続変化を繰り返すシステムをハイブリッドシステム [1] といい，力学系・回路・制御系など幅広い分野のシステムを表現することができる．自動運転のようなシステムのおいて，システムの振る舞いをシミュレーションすることや，その安全性に関する性質を検証することは重要である．ハイブリッドシステムを形式的にモデリングする手法としては，ハイブリッドオートマトン [2] によるものが一般的である．ハイブリッドシステムモデリング言語には様々なものがある [3] ものの，ハイブリッドオートマトンを基にした記述は煩雑である．ハイブリッドシステムモデリング言語 HydLa[4][5] は，制約の宣言と制約階層 [6] によって簡潔な記述が可能である．ハイブリッドシステムの記述に制約を用いた言語として他には Hybrid CC[7] がある．HyLaGI[8][9] は HydLa の記号シミュレータで，記号計算によって誤差のない計算をはじめとした様々な機能を実装している．ハイブリッドシステムに対し厳密なアプローチを行うツールとしては他に KeYmaera X[10] や dReach[11] があるが，これらはシミュレータではなく検証ツールである．また区間計算に基づくシミュレータとして Acumen[12] や Flow*[13] がある．HydLa と HyLaGI において解はそれぞれ宣言的意味論と操作的意味論により与えられている．宣言的意味論とは宣言型プログラミングにおいてプログラムの解とは何であるかを形式的に記述したものであり，操作的意味論とはプログラムの実行手順を形式化したものである．

意味論の応用として意味論より得られる解集合を比較することで異なるプログラムの等価性を判定することができる．またプログラムに対して特定の変換を考えると，変換の前後で解集合が変わらなければその変換は安全であることがわかる．一方でこれまでに HydLa において意味論を用いた研究はなかったため，まずは意味論自体の再検討も必要である．また HydLa の宣言的意味論は複雑であるため，紙とペンでの証明ではなく定理証明支援系を用いた形式的手法によって解計算を検証することも必要である．

1.2　本論文の構成

本論文の構成を以下に示す．第 2 章ではハイブリッドシステム及び制約に基づくハイブリッドシステムモデリング言語 HydLa の構文と意味論の概要について説明する．第 3 章では HydLa の記号処理系 HyLaGI と記号処理によって実現されている機能を紹介する．第 4 章では第 6 章で HydLa の宣言的意味論形式化に用いる定理証明支援系 Coq について説明する．第 5 章では HydLa の既存の宣言的意味論について述べた後，その問題点及びそれを解決する宣言的意味論の提案について述べる．第 6 章では第 5 章で提案する意味論について Coq での形式化及び検証を述べる．第 7 章では今後の課題と本論文のまとめについて述べる．また第 2 章，第 3 章は文献 [9]，第 5 章は文献 [14] が基になっている．

第 2 章

制約に基づく言語 HydLa

HydLa は制約の宣言と制約階層によってハイブリッドシステムをモデリングする言語である．この章ではハイブリッドシステムについて述べた後に，HydLa の構文と意味論の概要を説明する．本章は文献 [9] が基になっている．

2.1 ハイブリッドシステム

ハイブリッドシステムは時間の経過にともない状態が離散変化，もしくは連続変化するシステムである．ハイブリッドシステムは離散系・連続系を包含し，力学・回路・制御工学などの様々な分野への応用がある．例えば力学では，物体が滑らかに移動する間は連続変化であり，衝突によって速度などが不連続に変化する瞬間が離散変化である．回路では，過渡状態・定常状態が連続変化であり，スイッチ切り替えの瞬間が離散変化である．一般に制御工学では内部状態が切り替わる瞬間が離散変化で，それ以外の時間は連続変化である．

ハイブリッドシステムを形式的にモデリングする手法として最もよく知られているのはハイブリッドオートマトン [2] である．ハイブリッドオートマトンの構文を以下に述べる．

定義 1. ハイブリッドシステム H は次の要素からなる．

変数

変数の有限集合 $X = \{ x_1, \cdots, x_n \}$．$n$ は次元数と呼ばれる．微分を表すドット付き変数の集合 $\{ \dot{x_1}, \cdots, \dot{x_n} \}$ を $\dot{X}$ で，離散変化後を表すダッシュ付き変数の集合 $\{ x'_1, \cdots, x'_n \}$ を X' で書く．

制御グラフ

有限有向多重グラフ (V, E)．V 中の頂点はコントロールモードと呼ばれ，E 中の辺をコントロールスイッチと呼ぶ．

初期条件，不変条件，フロー条件

各 $v \in V$ に対し述語を割り当てる関数 $init, inv, flow$．各 $init(v), inv(v)$ 中の自由変数は X に属し，各 $flow(v)$ 中の自由変数は $X \cup \dot{X}$ に属する．

ジャンプ条件

各 $e \in E$ に対し述語を割り当てる関数 $jump$．各 $jump(e)$ 中の自由変数は $X \cup X'$ に属する．

イベント

イベントの有限集合 Σ と，イベントの割り当て $event : E \to \Sigma$.

ハイブリッドオートマトンは数学的なオブジェクトなので制約の宣言である．一方でハイブリッドシステムをシミュレーションするためのツールでは命令的な構成を含む Modelica [15] や，同期プログラミングをベースにした Zélus [16] などがある．これに対し HydLa は，制約ベースの言語はハイブリッドシステムの高レベルモデリングに適切なのかという課題の下で開発された言語である．

2.2　HydLa の構文

HydLa の詳細な文法について述べる前に，簡単な例を用いてその概要を述べる．

図 2.1 に床を跳ねるボールの HydLa プログラムを示す．HydLa では全ての変数は時間の関数である．例えば変数 y は関数 $\mathrm{y}(t)$ $(t \geq 0)$ の略記である．図 2.1 のプログラムでは，変数 y, y', y'' はそれぞれボールの高さ，速度，加速度を表す時間の関数である．はじめの 3 行は制約モジュール，もしくは単にモジュールと呼ばれる名前付き制約の定義である．制約には微分方程式と論理演算を使うことができる．INIT はパラメータ化されたボールの初期位置と初速度を定義している．FALL は自由落下を，BOUNCE はボールが床で跳ねる様子を表現している．“always” と呼ばれる時相論理演算子 [] は制約が生成された瞬間以降常に成り立ち続けることを表す．後置演算子-はその変数の左極限値を表す．例えば $\mathrm{y}\text{-}(t)$ は関数 $\lim_{t' \to t-0} \mathrm{y}(t')$ である．=>は含意を表す論理演算子である．5 行目ではこれらの 3 つのモジュールがどのように影響し合うかを，モジュール間の相対的な強さとともに宣言している．このプログラムでは FALL は BOUNCE より弱く宣言されていて，BOUNCE と矛盾したときには無視される．

HydLa の詳細な構文を図 2.2 に示す．ここで，*dname*, *cname*, *vname* はそれぞれ定

```
1 INIT <=> 7 < y < 12 & y' = 0.
2 FALL <=> [](y'' = -10).
3 BOUNCE <=> [](y- = 0 => y' = -4/5 * y'-).
4
5 INIT, (FALL << BOUNCE).
6 //#hylagi -p10
```

図 2.1　初期位置がパラメータ化された床を跳ねるボールの HydLa プログラム

(HydLa program)	P	::=	$(Def \mid Decl)^*$
(definition)	Def	::=	$dname(\vec{X})\{Decl\} \mid cname(\vec{X}) \Leftrightarrow C$
(constraint)	C	::=	$A \mid C \wedge C \mid G \Rightarrow C \mid \exists vname.C$
			$\mid \Box C \mid cname(\vec{E})$
(guard)	G	::=	$A \mid G \wedge G \mid G \vee G \mid \neg G$
(atomic constraint)	A	::=	$E\ Rop\ E$
(relational operator)	Rop	::=	$= \mid \neq \mid > \mid \geq \mid < \mid \leq$
(expression)	E	::=	$E\ Aop\ E \mid Prev \mid constant$
(arithmetic operator)	Aop	::=	$+ \mid - \mid \times \mid \div \mid \hat{}$
(previous)	$Prev$	::=	$D \mid D-$
(derivative)	D	::=	$vname \mid D'$
(declaration)	$Decl$	::=	$M \mid Decl, Decl \mid Decl \ll Decl$
(module)	M	::=	$C \mid dname(\vec{E})$

図 2.2　HydLa の構文

義，制約，変数の名前を表す．定義 *Def* では名前付き制約と同様に制約階層を含むような名前付きの宣言も定義することができる．また引数のある制約を定義することもできる．制約 C の構文ではガード付き制約の後件に *always* 制約が現れることを許している．宣言 *Decl* では演算子 “$\ll$” の結合度は “,” よりも高い．そのため例えば $A \ll B, C$ は $(A \ll B), C$ に等しい．

表 2.1 に図 2.2 の抽象構文と例題で用いられる具体構文の対応を示す．

表 2.1　抽象構文と具体構文の対応

Abstract	Concrete	Abstract	Concrete	Abstract	Concrete
$\ll$	<<	$\neq$	!=	$\Box$	[]
$\Leftrightarrow$	<=>	$\neg$	!	$\exists$	\
$\geq$	>=	$\vee$	\/ or |		
$\leq$	<=	$\wedge$	/\ or &		

```
1  INIT <=> switch = 0 & timer = 0.
2  CONST <=> [](switch' = 0).
3  TIMER <=> [](timer' = 1).
4  ON <=> [](timer- = 1 => switch = 1 & timer = 0).
5  STAY <=> [](timer- = 1 => switch = 0 & timer = 0).
6  TRUE <=> [](1 = 1).
7
8  INIT, (CONST, TIMER) << (ON, STAY) << TRUE.
9  //#hylagi --fnd -p6
```

図 2.3　非決定的なスイッチのモデル

2.3　HydLa の意味論の概要

HydLa プログラムの宣言的意味はプログラム中で与えられる制約を満たす軌道の集合である．ただし HydLa は各時刻でモジュールの極大無矛盾集合を採用する．ここでは意味論の重要な側面のみに触れる．

第一の側面として HydLa は非決定性を許している．HydLa において非決定性は 2 つのレベルで生じる．1 つは初期値が区間で与えられている時のように制約集合の解が複数ある場合，もう 1 つは極大無矛盾集合が一意に定まらない場合である．複数解はプログラム中にパラメータが現なくても生じる．例えば図 2.3 は，timer の値が 1 になるたびにスイッチの値が非決定的に切り替わる例である．

第二の側面としてプログラム中で明示的に与えられる制約は解軌道を決定するのに十分ではない．例えば図 2.1 の床を跳ねるボールのモデルでは，ボールの位置について，明示

的に与えられている制約から不連続性が導かれない限り連続であるというフレーム公理が暗黙に仮定される．そうでなければ，ボールが跳ねた直後に床から移動を再開すると推論することはできない．これは暗黙の連続性と呼ばれる．暗黙の連続性やその他の意味論については 5 章で詳細に述べる．

第 3 章

HydLa の記号処理系 HyLaGI

HyLaGI は HydLa の処理系であり，ハイブリッドシステムの非決定的かつ厳密なシミュレーションを行う．そのための主な手法として記号計算を用いている．ただし HyLaGI は最小離散変化時刻の導出の最適化として部分的に区間計算も採用している．HyLaGI によるシミュレーションは終了条件を満たすまでポイントフェーズ (PP) とインターバルフェーズ (IP) を交互に繰り返す．終了条件にはシミュレーション内の時刻やフェーズ数を与えることができる．IP の計算ではパラメータを含む微分方程式の求解と次の離散変化時刻の最小化問題の求解を行う．記号パラメータによる非決定性として定性的に異なる軌道が解となることがある．その場合 HyLaGI は自動的に場合分けを行い，各ケース毎の新たなパラメータの範囲を記号的に求める．この記号的な場合分けは制約ソルバによる量化子除去によって実装されている．HyLaGI の詳細なアルゴリズムについては文献 [8] を参照せよ．この章の残りでは記号的なアプローチによって実現される HyLaGI の機能について述べる．本章は文献 [9] が基になっている．

3.1 アサーション

HyLaGI は到達可能性の有界モデル検査を行えるよう，制約を用いた `ASSERT` を提供している．検査したい性質は G をガード条件として，`ASSERT`(G) と書く．`ASSERT`(G) の宣言的な意味は `[]G`，もしくは `[](!G => false)` と同じであるが，検証条件とモデルの記述を分離するために別の機能として提供されている．`ASSERT`(G) は G が偽になったときに，非決定的なシミュレーションのうちの該当する分岐のみを中断する．アサーションは検証のみでなく，逆問題の求解にも用いられる．逆問題の求解については 3.4 節で述べる．

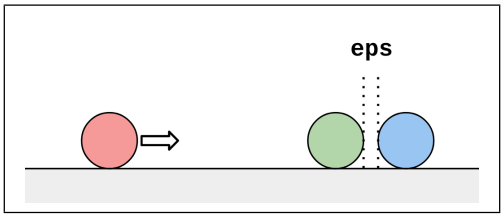

図 3.1　三体衝突

3.2　イプシロンモード

ハイブリッドシステムのモデリングではモデルがしばしば特異的な状況に陥ることがある．特異的な状況とは例えば，次のような状況である．

1. 箱の中をボールが跳ねまわっている時に壁と床に同時にぶつかる．
2. 2 つの接しているボールの片方に 3 つ目のボールがぶつかる（図 3.1）．
3. 2 つの接している物体の片方に力を加え，両方を動かす．

1 つ目の例は現実に起きる可能性は 0 であるが，ボールが壁か床にわずかに早く当たる時の極限として考えることはできる．2 つ目と 3 つ目の例は共に，2 つの物体がわずかに離れているという状況の極限と捉えられる．HyLaGI は上の例のようなモデルを通常のシミュレーションの極限として計算するイプシロンモードを提供している．

イプシロンモードでは図 3.2 のように無限小を表すパラメータとして変数 eps を使う．変数 eps は，各フェーズで極大無矛盾集合の導出とそれによって求まる変数の値を計算した後で，高次の項が除去される．そして全てのフェーズが終了した後で 0 への極限を取る．図 3.2 のプログラムでは直径 1 の 3 つのボールが一直線に並び，x2 と x3 は eps だけ離れ，x1 が x2 に向かって速度 1 で転がってゆく．図 3.2 のプログラムの eps が残ったシミュレーション結果を図 3.3 に示す．

三体衝突には様々なバリエーションが考えられる．ここでは両側から同時に真ん中のボールに衝突するモデル [17] を考える．図 3.4 の 4 行目のように，ボールは異なる質量を持っているとする．この問題では図 3.5 に示すように，結果は eps の値の正負に依存し，左極限と右極限は一致しない．

ディラックのデルタ関数もイプシロンモードを使うことで表現できる．デルタ関数は図 3.6 のように，幅が eps で値が 1/eps であるような関数で，eps を正方向から 0 に近づけた極限と考えられる．デルタ関数は力学における瞬間的な衝撃や電子回路におけるインパ

```
1  INIT <=> x1 = 0 & x2 = 5 & x3 = 6+eps
2          & x1' = 1 & x2' = 0 & x3' = 0.
3  EPS <=> 0 < eps < 0.1 & [](eps' = 0).
4  CONST(x) <=> [](x'' = 0).
5  COLLISION(xa, xb) <=>
6     [](xa- = xb- - 1 => xa' = xb'- & xb' = xa'-).
7
8  INIT, EPS.
9  (CONST(x1),CONST(x2),CONST(x3))
10    << (COLLISION(x1,x2), COLLISION(x2,x3)).
11 //#hylagi --fnd -p6 -e1
```

図 3.2　三体衝突の HydLa プログラム

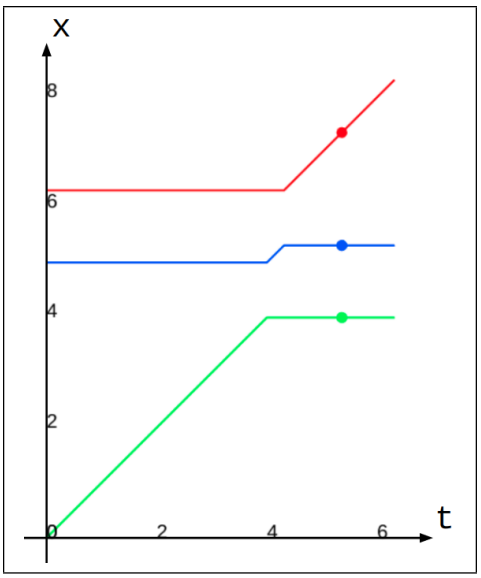

図 3.3　図 3.2 のシミュレーション結果

ルス応答に用いることができる.

```
1  INIT <=> x1 = 0 & x2 = 5 & x3 = 10+eps
2           & x1' = 1 & x2' = 0 & x3' = -1.
3  EPS <=> -0.1 < eps < 0.1 & [](eps' = 0).
4  MASS <=> [](m1 = 0.2 & m2 = 1 & m3 = 5).
5  CONST(x) <=> [](x'' = 0).
6  COLLISION(xa,ma,xb,mb) <=>
7      [](xa- = xb- - 1 =>
8         xa' = (xa'- *(ma-mb) + 2*mb*xb'-)/(ma+mb)
9       & xb' = (xb'- *(mb-ma) + 2*ma*xa'-)/(ma+mb)).
10
11 INIT. EPS. MASS.
12 (CONST(x1),CONST(x2),CONST(x3))
13    << (COLLISION(x1,m1,x2,m2), COLLISION(x2,m2,x3,m3)).
14 //#hylagi --fnd -p12 -e1
```

図 3.4　質量のパラメータ化された 3 体衝突

3.3　ハイブリッドオートマトンモード

HyLaGI は通常，与えられたフェーズ数，もしくはシミュレーション時刻の計算を行う．変数への値の割り当てを制約だと考えることで，あるフェーズでのシステムの状態はそれ以前のフェーズでの状態によって包含されているかどうかを調べることができる．ここで状態とは，変数の値（IP なら軌道）と採用されたモジュール集合である．採用されたガード付き制約のガードの成否はこの 2 つから導くことができる．この状態の包含によって有限のフェーズによって無限のフェーズの遷移を表現できることがあり，HyLaGI ではハイブリッドオートマトンモードとして実装されている．ハイブリッドオートマトン構築アルゴリズムの詳細については [18] を参照せよ．ハイブリッドオートマトンを構築するためには，変数の初期値を適切にパラメータ化しなければならない．例えば床を跳ねるボールの例では，図 3.7 のように高さの初期値を完全にパラメータ化する必要がある．図 3.7 のプログラムから得られた状態遷移グラフを図 3.8 に示す．

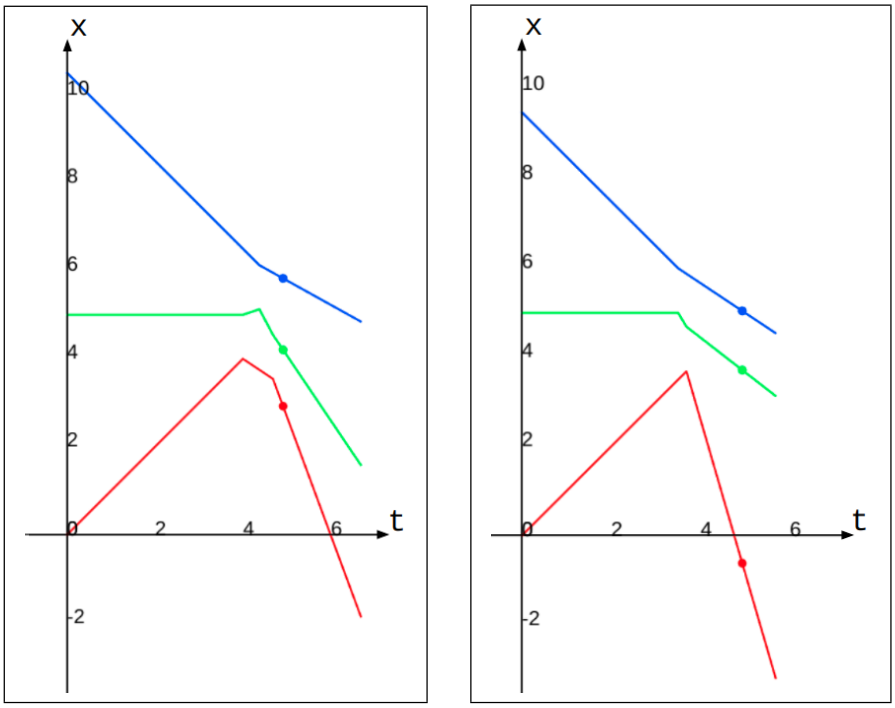

図 3.5　図 3.4 のシミュレーション結果

```
1  TIMER <=> timer = 0 & [](timer' = 1).
2  EPS <=> 0 < eps < 0.1 & [](eps' = 0).
3  OFF <=> [](v = 0).
4  ON <=> []((1 < timer < 1+eps) => v = 1/eps).
5
6  TIMER, EPS, (OFF << ON).
7  //#hylagi -e1
```

図 3.6　デルタ関数のモデル

3.4　ハイブリッドシステムの逆問題の求解

逆問題とは与えられたゴールを満たすような初期条件を求める問題である．HyLaGI は，アサーションとパラメータの記号的な場合分けを組み合わせることでハイブリッドシステムの逆問題を解くことができる，このアプローチはゴールからの逆実行ではなく，順方向の記号的なシミュレーションに基づいている．

```
1 INIT <=> y > 0.
2 FALL <=> [](y'' = -10).
3 BOUNCE <=> [](y- = 0 => y' = -4/5 * y'-).
4
5 INIT, FALL << BOUNCE.
6 //#hylagi --fha
```

図 3.7　初期位置がパラメータ化された床を跳ねるボールのモデル

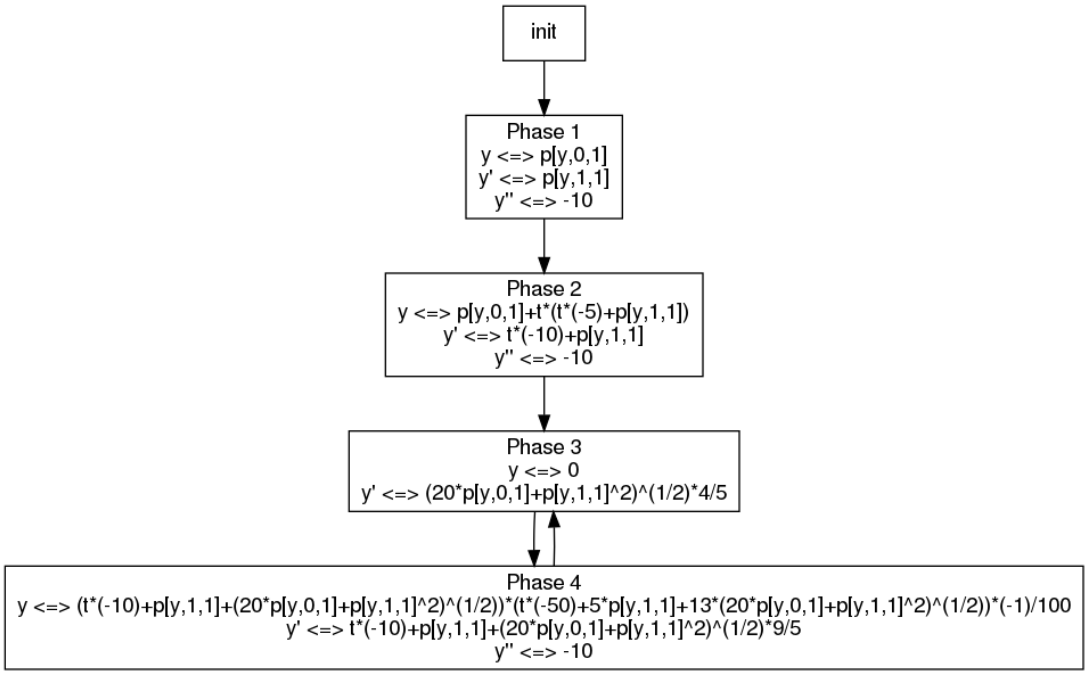

図 3.8　図 3.7 のモデルから HyLaGI が計算した状態遷移グラフ

まずはシンプルな例としてホールインワンを狙うためのゴルフボールの打ち方を考える．プログラムを図 3.9 に示す．x 方向の初速度はパラメータ化し，y 方向の初速度は速度の大きさが 10 になるように定める．ボールは x 方向に等速で進み，y 方向には床を跳ねるボールのように振舞う．カップが 9.5 と 10 の間にあるとして，ASSERT はホールインワンとなるパラメータを求めるために使う．ASSERT の中身に入る制約は，HyLaGI が反例を求めるため，求めたいゴールの否定となる．この場合は!(y = 0 & 9.5 <= x <= 10) である．表 3.1 に図 3.9 から得られるボールの振る舞いとパラメータの範囲の対応を示す．ただし，“bounce” はボールが弾むことを，“cup-in” はボールがカップに入ること

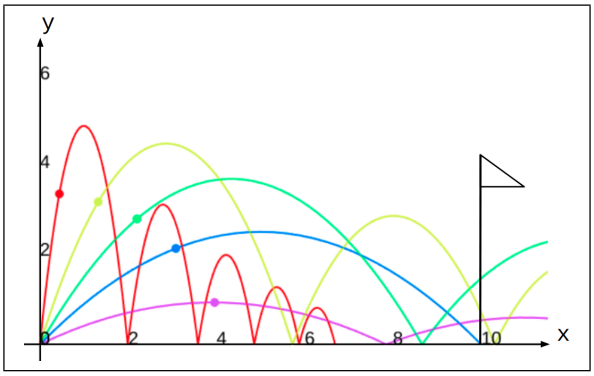

図 3.9　ホールインワン問題

```
1  INIT <=> x = 0 & y = 0 & 1 < x' < 9 & y' = (100 - x'^2)^0.5.
2  AXCONST <=> [](x'' = 0).
3  FALL <=> [](y'' = -10).
4  BOUNCE <=> [](y- = 0 => y' = -0.8*y'-).
5
6  ASSERT(!(y = 0 & 9.5 <= x <= 10)).
7  INIT, AXCONST, FALL << BOUNCE.
8  //#hylagi --fnd -p10
```

図 3.10　ホールインワンのためのパラメータを求める HydLa プログラム

を表す.

ここからは永続的な後件を用いた例を紹介する．HydLa の文法は always 制約 $\Box C$ が含意の後件に現れることを許している．そのような制約は永続的な後件 (persistent conseqent) と呼ばれる．永続的な後件 $\Box(G \Rightarrow \Box C)$ は通常のガード付き制約と異なり，一度前件 G が成り立つとその後，後件 C は成り立ち続ける．後件 C はもとの制約と同じ優先度の制約として展開される．一度展開された制約は除去されないため，永続的な後件はシステムの不可逆な変化を表現することができる．

図 3.11 に床を跳ねるボールが床にダメージを与えそのうちに壊れるモデルを示す．4 行目の BOUNCE で，速度の 2 乗に比例するダメージが床に衝突するたびに蓄積される様子を表している．蓄積されたダメージがある閾値に達すると，床は落下し続けるようになる．これによって床が壊れたことを表す．どのような条件で床が割れるのか計算するためには，ボールの高さが非負であるという制約をアサーションにかける．表 3.2 に図

表 3.1　図 3.10 の実行結果

ボールの振る舞い	パラメータの範囲
cup-in	$\left[\sqrt{\frac{5(20-\sqrt{39})}{2}}, \sqrt{\frac{5(20+\sqrt{39})}{2}}\right]$
bounce, cup-in	$\left[\frac{5\sqrt{\frac{36-\sqrt{935}}{2}}}{3}, \frac{5\sqrt{14}-10}{3}\right]$
bounce, bounce, cup-in	$\left[5\sqrt{\frac{244-\sqrt{50511}}{122}}, 5\sqrt{\frac{2(61-6\sqrt{86})}{61}}\right]$
bounce, bounce, bounce, cup-in	$\left[\frac{5\sqrt{\frac{1476-\sqrt{1952951}}{82}}}{3}, \frac{5\sqrt{\frac{2(369-2\sqrt{30134})}{41}}}{3}\right]$
bounce, bounce, bounce, bounce	others

表 3.2　図 3.11 の実行結果

ボールの振る舞い	パラメータの範囲
bounce, bounce, bounce, bounce, bounce	(5, 7812500/968561)
bounce, bounce, bounce, bounce, break	[7812500/968561, 312500/36121)
bounce, bounce, bounce, break, through	[312500/36121, 12500/1281)
bounce, bounce, break, through	[12500/1281, 10)

3.11 から HyLaGI が計算したシステムの振る舞いとパラメータ範囲の対応を示す．ここで “bounce” はボールが床で跳ねる様子を，“break” はボールが床で跳ねて床が壊れる様子を，“through” はボールが壊れた床を通り過ぎる様子をそれぞれ表す．

```
1  INIT <=> 5 < y < 10 & y' = 0 & d = 0.
2  FALL <=> [](y'' = -10).
3  CONST <=> [](d' = 0).
4  BOUNCE <=> [](y- = 0 => y' = -4/5 * y'-
5                          & d = d- + y'-^2 / 100).
6  BREAK <=> [](d >= 4 => [](y'' = -10 & d' = 0)).
7
8  INIT, (FALL, CONST) << BOUNCE << BREAK.
9  ASSERT(y >= 0).
10 //#hylagi -p12 --fnd
```

図 3.11　床を跳ねるボールが床にダメージを与えるモデル

第 4 章

定理証明支援系 Coq

4.1 定理証明支援系と Coq

定理証明支援系とはコンピュータを用いて形式的な証明の支援を行うツールのことであり，既知の定理のライブラリ化や証明の各ステップの正しさの保証，自動証明などの特徴がある．コンピュータを用いた証明と定理証明支援系を用いた証明ははほぼ同義であるが，定理証明支援系が発展する以前ではそうでなかった．例えばコンピュータを用いた証明として有名な四色定理の証明 [19] があるが，この証明では定理証明支援系は使われていない．

Coq は定理証明支援系の 1 つである．Coq は以下のような利点を持っている．

- 豊富なライブラリ（特に実数ライブラリがある）
- 純粋数学的（例えば極限が実数上ではなく距離空間上で定義されている）
- いくつかの自動証明
- 日本を含めユーザが多く，学習しやすい

主な Coq を用いた証明として，数学においては四色定理 [20] や奇数位数定理 [21] が，ソフトウェアにおいては C コンパイラ CompCert の検証 [22] がある．

Coq 以外の定理証明支援系としてはここでは Isabelle/HOL と TLA+ を挙げる．Isabelle/HOL を用いた証明として，数学においてはケプラー予想 [23] が，ソフトウェアにおいてはマイクロカーネル seL4 の検証 [24] がある．TLA+ は主に分散システムの検証で用いられており，Amazon DynamoDB と Amazon S3[25]，Azure Cosmos DB[26]，TiDB[27] などでの利用がある．

次節以降では第 6 章で必要となる Coq の基本的な文法と実数ライブラリについての説

明を行う.

4.2　Coq の構文

Coq では大きく Vernacular[28]，Gallina[29] およびタクティック [30] の 3 つによって命題と証明を記述していく.

Vernacular は宣言や定義，命題や証明の開始と終了など，Coq プログラムのうち証明本体以外の部分を記述する言語である．主な Vernacular コマンドとしては以下がある

`Require`
: Coq モジュールのインポートおよびエクスポートのために用いる.

`Parameter, Axiom, Variable, Hypothesis`
: 変数および証明なしで認める命題の宣言を行う．`Parameter` と `Axiom` はグローバルな宣言である.

`Definition, Example, Let`
: 定義を行う．`Definition` と `Example` はグローバルな定義である.

`Inductive, Fixpoint`
: これらも定義を行う．`Inductive` は帰納的な定義を，`Fixpoint` は不動点を用いた定義を行う.

`Theorem, Lemma, Proposition`
: 命題を記述する．`Definition` と等価である.

`Proof, Qed`
: 証明モードの開始と終了を表す.

Gallina およびタクティックはそれぞれ証明を直接的，間接的に記述するための言語である．Gallina は項の記述のみに用いて，証明はタクティックを用いるのが通常である．項には以下のような要素がある.

全称量化子
: 例：`forall n, 0 < n`

関数
: 例：`fun n => n + n`

マッチング
: 例：`match n with O => true | S m => false end`

if then else

例：`if eq_nat_dec a b then a else b`

束縛

例：`let x := 1 in x + x`

再帰関数

例：`fix func n := match n with O => 1 | S n' => n * (func n') end`

Coq のタクティックは膨大な数があり，以下では全ての証明でほぼ確実に使われるようなもののみを紹介する．

`intros`

量化された変数や含意の前件を仮定に追加する．

`unfold`

定義を展開する．

`apply`

仮定や命題を適用する．

`rewrite`

等式変形を行う．

`destruct`

場合分けを行う．

Coq には証明を自動化するタクティックがある．主なものを以下に紹介する，

`auto`

`intros`, `apply`, `assumption`, `reflexivity` のみを用いて証明を試みる．

`omega`

不等式を含むプレスバーガー算術の論理式を証明する．

`ring`

環における等式を証明する．特に実数は環であるので，実数における等式を証明できる．

`congruence`

等式変形のみで自動証明を試みる．

4.3　Coq の実数ライブラリ

実数の多くは計算不可能であるため，定理証明支援系では実数全体の集合を構成することはできない．そこで Coq では実数の連続性などを単に公理として認めている．ここでは Coq の実数ライブラリのうち `Coq.Reals.Reals` がエクスポートしているものを紹介する．

`Coq.Reals.Rbase`
: 実数の公理，自然数や整数から実数への埋め込み，等式と不等式に関する定理など．

`Coq.Reals.Rfunctions`
: 実数上の基本的な関数．特に距離関数としての絶対値や冪乗を含む．

`Coq.Reals.SeqSeries`
: 数列や級数に関する定理．

`Coq.Reals.Rtrigo`
: 三角関数に関する定理．

`Coq.Reals.Ranalysis`
: 実解析における定理．特に実数における極限，連続性や微分を含む．実数における極限は実際には `Coq.Reals.Rlimit` で距離空間上で定義されている．日本でよく知られた定義と異なり，`limit_in` では極限を取る点に近い点として同じ点をとっても良いことに注意せよ（`continue_in`，`D_in` では異なる点を要求している）．

`Coq.Reals.Integration`
: 積分についての定理．

第 5 章

HydLa の宣言的意味論

本章では HydLa プログラムの等価性判定を考える．はじめに HydLa における既存の宣言的意味論について検討した後，HyLaGI の操作的意味論を取り入れた新たな宣言的意味論を提案し，例題においてその妥当性を見る．プログラムの等価性は妥当性検討の結果としてえられる．以降，単に意味論と言った場合には宣言的意味論を指すものとする．本章は文献 [14] が基になっている．

5.1 既存の宣言的意味論の検討

既存の宣言的意味論として上田らによる意味論 [5] と松本による意味論 [8] がある．松本による意味論は上田らによる意味論に対して修正と詳細化を行ったもので，特に hybrid trajectory を導入しているため，本論文のベースとする．以降，既存の意味論と言った場合には松本による意味論を指す．

5.1.1 暗黙の連続性

既存の宣言的意味論は変数について，次の暗黙の連続性と呼ばれるフレーム公理を導入している．「HydLa のオリジナルの意味論では，与えられた HydLa プログラム中の制約 C がある変数の微分に言及するなら，その変数は暗黙的に連続である．そのような連続性は C 自身より高い優先度を持ち，C より高い優先度を持つ他の制約より弱い優先度を持つ.」（文献 [8] より著者訳）このように既存の宣言的意味論の暗黙の連続性は HydLa プログラムに対して静的に仮定しているという特徴がある．

検討の始めとして既存の意味論，特に暗黙の連続性がある程度妥当であることを確認

```
1 INIT <=> y = 10 & y' = 0.
2 FALL <=> [](y'' = -10).
3 BOUNCE <=> [](y- = 0 => y' = -4/5 * y'-).
4 INIT, FALL << BOUNCE.
```

図 5.1　優先度を用いた床を跳ねるボールのプログラム P_1

する．例えば図 5.1 のような床を跳ねるボールの HydLa プログラム P_1 を考える．このプログラムは図 2.1, 3.7 とほぼ同じかつ，より簡単な例であるが，意味の計算を簡単化するために新たに紹介する．HydLa の宣言的意味論は，各時刻で優先度関係を満たす最も大きなモジュールの集合（以下，極大無矛盾集合）を採用する軌道を解とする．このプログラムでは極大無矛盾集合は {INIT，FALL，BOUNCE} か {INIT，BOUNCE} である．図 5.1 のプログラムで暗黙の連続性を考える．モジュール FALL が y'' に言及しているため，y' についての左右の連続性がそれぞれ FALL，BOUNCE の間の優先度となる．そのため，FALL が採用されない瞬間においても y' をできるだけ連続にしようとする軌道（正しく床を跳ねるボールの軌道）が解となる．またモジュール INIT，BOUNCE が y' に言及しているため，y についての左右の連続性がそれぞれ INIT，BOUNCE より高い優先度となる．文献 [8] では触れられていないが，INIT より導かれる暗黙の連続性は時刻 0 でのみのものだと考えるのが自然である．すなわち，図 5.1 のプログラムで制約階層は実際には次のようになる．

```
INIT << (y は左連続，y は右連続),
FALL << ([](y' は左連続), [](y' は右連続))
  << BOUNCE << ([](y は左連続), [](y は右連続)).
```

したがって厳密には極大無矛盾集合となりうるモジュール集合は 2 つではない．そして，ボールが宙にあるうちは全てのモジュールが採用され，ボールが床に衝突した瞬間では FALL，[](y' は左連続) 以外が採用されることで，ボールは放物運動とバウンドを繰り返す．この例では静的な暗黙の連続性を考えることによりプログラムは簡潔になり，かつプログラムに対し自然な解が定まっている．

静的な暗黙の連続性は，宣言的意味論も簡潔になるというメリットがある一方で問題点もある．例えば図 5.2 のような HydLa プログラムを考える．図 5.2 のプログラムは図 5.1 のプログラムに対し，制約階層を平坦化するというプログラム変換を適用した結果と言える．図 5.2 のプログラムはガード条件を if 文のように用いて，y の値で挙動を切り替える

```
1 INIT <=> y = 10 & y' = 0.
2 FALL <=> [](y != 0 => y'' = -10).
3 BOUNCE <=> [](y = 0 => y' = -0.8*y'-).
4 INIT, FALL, BOUNCE.
```

図 5.2　平坦な制約改装の床を跳ねるボールのプログラム P_2

```
1 [](0 = 1 => x' = 0).
2 [](x = 0) << [](t = 1 => [](x = 1)).
```

図 5.3　ステップ関数の HydLa プログラム P_3

ことを意図している．しかし静的な暗黙の連続性を考えると，FALL が y'' に，BOUNCE が y' に言及している．y，y' は連続でなくてはならないためボールは弾むことができず，意図した軌道を表せていないことが分かる．

より極端な例としては図 5.3 のような HydLa プログラムが考えられる．図 5.3 では 1 行目の絶対に成り立たない制約が x' に言及しているため x は連続になってしまい，既存の宣言的意味論ではステップ関数とならない．

この問題を解決するには暗黙の連続性の扱い方を改善する必要があり，ここでは二つの方針を紹介する．一つは暗黙の連続性を静的なまま改良する方法である．具体的にはガード付き制約によって追加される暗黙の連続性には，その制約と同じガードを連続性制約の前件とする．これによって上で述べたような暗黙の連続性が成立しない制約から導かれるような状況は解消される．もう一つは暗黙の連続性は静的にプログラムに追加されるのではなく，採用されかつ有効な制約中の変数のみを動的に考える方法である．実際に HydLa の実装である HyLaGI では連続性についての制約を動的に追加するため，プログラム P_2 に対しては床を跳ねるボールの軌道のみを，プログラム P_3 に対してはステップ関数のみを出力する．本研究では将来的な宣言的意味論と操作的意味論の比較を念頭に，二つ目の暗黙の連続性を動的に扱う意味論を提案する．5.2 節以降では，暗黙の連続性を動的に扱う宣言的意味論を厳密化し，上記のような制約階層のプログラムに対しても自然に解集合が定まることを確認していく．

5.1.2　軌道

次に hybrid trajectory について検討する．以降，hybrid trajectory を単に軌道と呼ぶ．HydLa における軌道の微分は直感的には一般的な微分と同様であるが，そうでないと考えるのが自然な場合もある．たとえば床を跳ねるボールの例では，ボールが床に当たる瞬間はボールの位置は通常の意味では微分不可能だが，変数 y' にはその時刻で右微分値が割り振られていると考えるのが自然である．また軌道の微分も軌道であると考えるのが自然である．しかし，例えば床を跳ねるボールの例でボールが床に当たる瞬間に変数 y'' は未定義であり，これは軌道の値域が $\mathbb{R}$ であることに反する．

軌道は宣言的意味論のベースとなる概念であるので，5.2 節ではまず軌道やその微分を再定義する必要がある．

5.2　新たな宣言意味論の提案

5.1 節で見たように，軌道やその微分の再定義をした後，暗黙の連続性を動的に扱う宣言的意味論を提案する．

5.2.1　軌道

軌道とその性質について以下で定義していく．

定義 2. ある変数 x が軌道であるとは，値 $x_i^D \in \mathbb{R} \cup \{\bot\}$ $(1 \leq i \leq n)$ と滑らかな関数 $x_i^C : (\tau_{i-1}, \tau_i) \to \mathbb{R}$ $(1 \leq i \leq n, \tau_0 = 0, \tau_{i-1} < \tau_i, \tau_i \in \mathbb{R} \cup \{\infty\})$ に対し，$x = \langle x_1^D, \langle x_1^C, \tau_1 \rangle, \ldots, x_n^D, \langle x_n^C, \tau_n \rangle \rangle$ となっていることである．このとき x を次のような関数 $[0, \tau_n) \to \mathbb{R} \cup \{\bot\}$ と同一視する．

$$x(t) = \begin{cases} x_{i-1}^D & (t = \tau_{i-1}) \\ x_i^C(t) & (\tau_{i-1} < t < \tau_i) \end{cases}$$

以下で値域が $\mathbb{R} \cup \{\bot\}$ である関数の極限を考えるが，そのためには $\mathbb{R} \cup \{\bot\}$ 上での演算を定義する必要がある．$\mathbb{R} \cup \{\bot\}$ 上の四則演算については，2 元が $\mathbb{R}$ の要素なら $\mathbb{R}$ と同様に，どちらかが $\bot$ なら $\bot$ になるとする．$\mathbb{R} \cup \{\bot\}$ 上の比較演算については，全順序集合 $\mathbb{R}$ に $\bot$ を比較不能な元として追加したと考える．各比較演算の詳細については表 5.1 に示す．表 5.1 において実数同士の演算は空欄としている．$\mathbb{R} \cup \{\bot\}$ 上の絶対値についても，$\mathbb{R}$ の要素なら $\mathbb{R}$ と同様に，$\bot$ なら $\bot$ になるとする．

表 5.1　$\mathbb{R} \cup \{\bot\}$ 上の比較演算

$>, <$	実数	$\bot$
実数		偽
$\bot$	偽	偽

$\geq, \leq, =$	実数	$\bot$
実数		偽
$\bot$	偽	真

$\neq$	実数	$\bot$
実数		真
$\bot$	真	偽

次に暗黙の連続性を動的に扱う意味論に必要となる「ある点における繋がり方」の概念を以下のように導入する．

定義 3. $f, g : [0, t) \to \mathbb{R} \cup \{\bot\}$ を軌道，$a \in [0, t)$ とする．「f の a での左極値が L である」とは

$$\lim_{x \to a-0} f(x) = L \stackrel{def}{\equiv} \forall \varepsilon > 0 \, \exists \delta > 0 \, \forall x \in [0, t) \\ (0 < a - x < \delta \Rightarrow |f(x) - L| < \varepsilon)$$

「f が a で左連続である」とは

$$LeftCont(f, a) \stackrel{def}{\equiv} \lim_{x \to a-0} f(x) = f(a)$$

右極限値と右連続性についても同様に定める．「a で f と g の繋がり方が同じである」とは

$$f \cong_a g \stackrel{def}{\equiv} LeftCont(f, a) \Leftrightarrow LeftCont(g, a) \\ \wedge RightCont(f, a) \Leftrightarrow RightCont(g, a)$$

極限値と連続性については一般的な定義と同じ書き方であるが演算は $\mathbb{R} \cup \{\bot\}$ のものである．また両側連続であることを $Cont(f, a)$ と書き，時刻 $t = 0$ のみ特別に $RightCont(f, a)$ なら $Cont(f, 0)$ とする．軌道の微分を次のように定める．

定義 4. 軌道 x に対し次を満たす軌道 $x'(\mathrm{dom}(x) = \mathrm{dom}(x'))$ を x の微分という．

$$(t \in (\tau_{i-1}, \tau_i) \Rightarrow x'(t) = x_i'^C(t)) \\ \vee (t = \tau_{i-1} \wedge Cont(x, t) \Rightarrow x'(t) \in \mathbb{R}) \\ \vee (t = \tau_{i-1} \wedge \neg Cont(x, t) \Rightarrow x'(t) = \bot)$$

また全ての τ_i $(0 \leq i \leq n)$ が等しい複数の軌道 $x_1, \ldots, x_m$ の組 $\langle x_1, \ldots, x_m \rangle$ を次のよ

うに定め，$\overline{x}$ のように書く．

$$\begin{aligned}&\langle x_1,\ldots,x_m\rangle\\ &\stackrel{def}{\equiv}\langle\langle(x_1)_1^C,\ldots,(x_m)_1^C\rangle,\langle\langle(x_1)_1^D,\ldots,(x_m)_1^D\rangle,\tau_1\rangle,\\ &\qquad\ldots,\langle(x_1)_n^C,\ldots,(x_m)_n^C\rangle,\langle\langle(x_1)_n^D,\ldots,(x_m)_n^D\rangle,\tau_1\rangle\rangle\end{aligned}$$

ある軌道の微分は $t=\tau_{i-1}\wedge Cont(x,t)$ なら任意の実数値を取って良いため，一意に定まらない．軌道 y が連続なら，定義 2 より微分 y' が一意に定まらないのは有限個の継ぎ目の点のみであるので，y' の原始関数は y のみである．しかし y が不連続点を含むなら y' は $\bot$ を取る点があるため，y は広義積分で考えても原始関数にならない．そのため解軌道を考える際には導関数も明示的に与える必要がある．

例 1. 軌道の例として時刻 $[0,13\sqrt{2}/5)$ で床を跳ねるボールの位置を挙げる．

$$y=\langle 10,\langle 10-5t^2,\sqrt{2}\rangle,0,\langle -5t^2+18\sqrt{2}t-26,13\sqrt{2}/5\rangle\rangle$$

解軌道を考える際には微分を明示的に与えなくてはいけないので，例えば次のような速度と加速度の軌道を考え組にする．

$$\begin{aligned}y'&=\langle 0,\langle -10t,\sqrt{2}\rangle,8\sqrt{2},\langle -10t+18\sqrt{2},13\sqrt{2}/5\rangle\rangle]\\ y''&=\langle -10,\langle -10,\sqrt{2}\rangle,\bot,\langle -10,13\sqrt{2}/5\rangle\rangle\\ \overline{y}&=\langle y,y',y''\rangle\\ &=\langle\langle 10,0,-10\rangle,\langle\langle 10-5t^2,-10t,-10\rangle,\sqrt{2}\rangle,\langle 0,8\sqrt{2},\bot\rangle,\\ &\qquad\langle\langle -5t^2+18\sqrt{2}t-26,-10t+18\sqrt{2},-10\rangle,13\sqrt{2}/5\rangle\rangle\end{aligned}$$

5.2.2　宣言的意味論

意味論の提案の前にいくつか補助的な概念を導入する．HydLa プログラム中の制約 C を，関数 $C(0)=\{C\},C(t)=\varnothing(t>0)$ と同一視する．また制約の集合と制約の論理積を同一視する．

定義 5. 時間から制約集合への関数 C に対し，$\Box$ 閉包 C^* を次のように定める．全ての実数 $t>0$ に対し，

- $C(t)\subseteq C^*(t)$
- $\Box a\in C^*(t)\Rightarrow\forall t'\geq t(a\subseteq C^*(t'))$

- $C^*(t)$ は上を満たす最小の集合

$C = C^*$ であるような C は $\Box$-*closured* であると言われる．制約の集合 X に対し，X から $\Box C$ の形の論理式を取り除いた集合を $X_{\setminus\Box}$ と書き，$\Box$ 除去と呼ぶ．制約の集合 X に対し，X から $C_1 \Rightarrow C_2$ の形の論理式を取り除いた集合を $X_{\setminus\Rightarrow}$ と書き，$\Rightarrow$ 除去と呼ぶ．

$\Box$ 閉包は，直感的にはある時刻で成り立った always 制約の中身がそれ以降の時刻で成り立つことを意図している．以上を踏まえ，HydLa の宣言的意味論を次のように定める．ここで Q はモジュール M と時刻 t を取り，制約の集合を返す関数である．

定義 6. $\langle \overline{x}, Q\rangle \models P \Leftrightarrow (i) \wedge (ii) \wedge (iii) \wedge (iv)$ ただし

(i) $\forall M \in P(Q(M) = Q(M)^*)$

(ii) $\forall M \in P(M^* \subseteq Q(M))$

(iii) $\forall t\, \exists E \in MS$ (s0)

$(\overline{x} \Rightarrow \{Q(M)(t)_{\setminus\Box} | M \in E\})$ (s1)

$\wedge \neg\, \exists E' \in MS(E \subsetneq E' \wedge \overline{x} \Rightarrow \{Q(M)(t)_{\setminus\Box} | M \in E'\})$ (s2-1)

$\wedge \neg\, \exists \overline{x}'\, \exists E' \in MS($ (s2-2)

$\forall t' < t(\overline{x}'(t') = \overline{x}(t')) \wedge \overline{x} \not\cong_t \overline{x}'$ (s2-2)

$\wedge E \subseteq E' \wedge (\overline{x}' \Rightarrow \{Q(M)(t)_{\setminus\Box} | M \in E'\}))$ (s2-2)

$\wedge \forall d \forall e \forall M \in E((\overline{x} \Rightarrow d) \wedge ((d \Rightarrow e) \in Q(M)(t)) \Rightarrow e \subseteq Q(M)(t))$ (s3)

$\wedge \forall M \in E\, \forall var \in Q(M)(t)_{\setminus\Box\setminus\Rightarrow}$ (s4)

$((\overline{x} \Rightarrow \mathit{LeftCont}(var, t)) \Rightarrow \mathit{LeftCont}(var, t) \in Q(M)(t)$ (s4)

$\wedge(\overline{x} \Rightarrow \mathit{RightCont}(var, t)) \Rightarrow \mathit{RightCont}(var, t) \in Q(M)(t))$ (s4)

(iv) $Q(M)(t)$ は (i)-(iii) を満たす最小の集合

$\langle \overline{x}, Q\rangle \models P$ を満たすとき，$\langle \overline{x}, Q\rangle$ を解，$\overline{x}$ を解軌道，Q を成立履歴もしくは制約ストアと呼ぶ．制約ストアという呼び方は HyLaGI の操作的意味論から来ている．定義 6 のうち (i) は Q が $\Box$-closured であること意味する．(ii) より $Q(M)$ は各モジュールの閉包 M^* を基にしており，(s3), (s4) などで制約が追加されるが，(iv) で余分な制約は入っていないことを要求する．(iii) について詳しく見る．(s0) は各時刻で極大無矛盾集合 E があることを，(s1) は軌道による値の割り当てが制約を満たすことを，(s3) は含意制約が成り立った時にはその後件が有効になることを表す．暗黙の連続性を動的に扱う意味論の本質的な変更は (s4) で，変数についての片側連続性を動的に，すなわち極大無矛盾集合中

のモジュールの有効な制約に現れる変数についてのみ制約ストアに追加することを表す．(s2-1), (s2-2) は (s4) の追加に合わせた従来の (s2) の修正で，(s2-1) が $\overline{x}$ に対して E が極大無矛盾であることを表し，(s2-2) が $\overline{x}$ とは異なる振る舞いをする $\overline{x}'$ で E より大きな E' を満たすものがないことを要求する．立ち位置としては (s2-2) が従来の (s2) であり，(s2-1) が新たに追加されたものである．(s2-2) で $\overline{x} \not\simeq_t \overline{x}'$ を要求しているのは，パラメータを含むプログラムについてパラメータの値は異なるが定性的に等しい軌道を解にするためであり，E' として E が取れるのは MS に連続性制約が含まれなくなったためである．(s2-2) で異なる繋がり方を要求した結果として，(s2-1) を追加しなければ比較可能な極大無矛盾集合 E' があっても良くなってしまうことに注意せよ．

5.3　例題

例 2. まず図 *5.1* のプログラム P_1 に対し，床を跳ねる軌道のみが解となることを見る．解となるのは例 *1* の $\overline{y}$ と次の Q_1 である．ただし $Q_1(*) = \bigcup_M Q_1(M)$ とし，各制約がどのモジュールより来るかは省略する．また $t = \sqrt{2}$ では *FALL* は採用されないが，Q_1 には入ることに注意せよ．

$$
Q_1(*)(t) = \begin{cases}
\{y = 10, y' = 0, \Box(y'' = -10), y'' = -10, & \\
\ \Box(y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ (y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ \mathit{RightCont}(y,t), \mathit{RightCont}(y',t), \mathit{RightCont}(y'',t)\} & (t = 0) \\
\{y'' = -10, (y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ \mathit{Cont}(y,t), \mathit{Cont}(y',t), \mathit{Cont}(y'',t)\} & (0 < t < \sqrt{2}) \\
\{y'' = -10, (y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ y' = -0.8 * y'-, \mathit{Cont}(y,t), \mathit{RightCont}(y',t)\} & (t = \sqrt{2}) \\
\{y'' = -10, (y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ \mathit{Cont}(y,t), \mathit{Cont}(y',t), \mathit{Cont}(y'',t)\} & (\sqrt{2} < t < \dfrac{13\sqrt{2}}{5})
\end{cases}
$$

まず $\langle \overline{y}, Q_1 \rangle$ が解であるかを考える．時刻 $t = \sqrt{2}$ 以外では全ての制約を採用し変数が連続であるため，時刻 $t = \sqrt{2}$ でより多くの制約を採用する軌道やより多くの連続性を満たす軌道がなければ良い．軌道 $\overline{y_1}$ が $t = \sqrt{2}$ で極大無矛盾集合 {*INIT, FALL, BOUNCE*} を採用するとする．$y- = 0$ より $y' = -0.8 * y'-$ と $y'' = -10$ が矛盾するためこれはありえない．次に軌道 $\overline{y_2}$ が $t = \sqrt{2}$ で {*INIT, BOUNCE*} を極大無矛盾集合とし，$\overline{y}$ より多

くの連続性を満たすとする．$y- = 0$ より $y' = -0.8 * y'-, y'' =\bot$ であるのでこれもありえない．よって $\overline{y}, Q_1$ は解の *1* つである．

次に $\langle \overline{y}, Q_1 \rangle$ が唯一の解であることを見る．他に解 $\overline{y_3}$ があるとする．$\overline{y_3}$ の極大無矛盾集合は $\overline{y}$ と同じで，$t = \sqrt{2}$ 以外では $\overline{y} = \overline{y_3}$ かつ，$t = \sqrt{2}$ では軌道が満たす連続性制約の集合が $\overline{y}$ と比較不能である．また軌道が満たす連続性制約の集合が $\overline{y}$ と比較可能であってはならないので，時刻 $t = \sqrt{2}$ では $LeftCont(y', t), LeftCont(y'', t), RightCont(y'', t)$ のいずれかが成り立つ必要がある．$LeftCont(y', t)$ が成り立つ場合，y' が離散変化しないことから $y \neq 0, y'' = -10$ が導かれる．よっていずれにしても $y'' \neq \bot$ で $Cont(y, t), Cont(y', t)$ が導かれるが，$t = \sqrt{2}$ で $Cont(y, t) \wedge Cont(y', t)$ は矛盾する．よって $\langle \overline{y}, Q_1 \rangle$ はプログラム P_1 の唯一の解である．これによって，プログラム P_1 のような既存の意味論において自然に意味が定まるプログラムに対して，暗黙の連続性を動的に扱う意味論で自然に意味が定まることが確認できた．

例 3. 図 *5.2* のプログラム P_2 に対し，床を跳ねる軌道のみが解となることを見る．解となるのは例 *1* の $\overline{y}$ と次の Q_2 である．

$$
\begin{aligned}
&Q_2(*)(t)\\
&= \begin{cases}
\{y = 10, y' = 0, \Box(y \neq 0 \Rightarrow y'' = -10), & \\
\ (y \neq 0 \Rightarrow y'' = -10), y'' = -10, & \\
\ \Box(y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ (y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ RightCont(y, t), RightCont(y', t), RightCont(y'', t)\} & (t = 0)\\
\{(y \neq 0 \Rightarrow y'' = -10), y'' = -10, & \\
\ (y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ Cont(y, t), Cont(y', t), Cont(y'', t)\} & (0 < t < \sqrt{2})\\
\{(y \neq 0 \Rightarrow y'' = -10), & \\
\ (y = 0 \Rightarrow y' = -0.8 * y'-), y' = -0.8 * y'-, & \\
\ Cont(y, t), RightCont(y', t)\} & (t = \sqrt{2})\\
\{(y \neq 0 \Rightarrow y'' = -10), y'' = -10, & \\
\ (y = 0 \Rightarrow y' = -0.8 * y'-), & \\
\ Cont(y, t), Cont(y', t), Cont(y'', t)\} & (\sqrt{2} < t < \dfrac{13\sqrt{2}}{5})
\end{cases}
\end{aligned}
$$

まず $\langle \overline{y}, Q_2 \rangle$ が解であるかを考える．極大無矛盾集合は常に $\{$`INIT, FALL, BOUNCE`$\}$ である．また時刻 $t = \sqrt{2}$ 以外では全ての変数が連続であるため，時刻 $t = \sqrt{2}$ でより多くの連続性を満たす軌道がなければ良い．そのような軌道 $\overline{y_4}$ があったとする．$\overline{y_4}$ は y が

連続であるので，$y = 0$ かつ y' が離散変化するため，$\overline{y} = \overline{y_4}$ となり矛盾する．よって $\langle \overline{y}, Q_2 \rangle$ は解の *1* つである．

次に $\langle \overline{y}, Q_2 \rangle$ が唯一の解であることを見る．他に解 $\overline{y_5}$ があるとする．$\overline{y_5}$ が時刻 $t = \sqrt{2}$ 以外では $\overline{y}$ と等しくなくてはならないことが採用する制約や連続性から分かる．また軌道が満たす連続性制約の集合が $\overline{y}$ と比較可能であってはならないので，時刻 $t = \sqrt{2}$ では $LeftCont(y', t), LeftCont(y'', t), RightCont(y'', t)$ のいずれかが成り立つ必要がある．いずれの場合でも $y- = 0, y' = -0.8 * y'-$ より矛盾する．以上の議論より $\langle \overline{y}, Q_2 \rangle$ がプログラム P_2 の唯一の解である．またプログラム P_2 に対して暗黙の連続性を動的に扱う意味論では自然に意味が定まることが確認できた．

また以上を通して，2 つのプログラムに対して解軌道となりうる軌道が同一であることが示せたため，2 つのプルグラムの等価性を示すことができた．

第 6 章

意味論の Coq による実装と検証

本章では HydLa の宣言的意味論の Coq 実装および例題での解の検証を行う．ただし本論文では Coq プログラムのうちの要点のみを掲載し，プログラム全体は [31] にて公開した．

6.1 諸概念の実装

はじめに，軌道や HydLa における制約の Coq 実装について述べる．

6.1.1 軌道

軌道の定義を紹介する前に $R \cup \{ \bot \}$ での演算について述べる．$R \cup \{ \bot \}$ は Coq においては `option R` 型で表す．図 6.1 に $R \cup \{ \bot \}$ での演算の実装のうち，今後極限の定義などで必要な部分を示す．5.2 節で定義したように，算術演算と絶対値は $\bot$ が関係すれば $\bot$ になり，比較演算は $\mathbb{R}$ に比較不能な一点を追加したものとなる．

続いて軌道とその極限，連続性，および微分について，図 6.2 に示す．5.2 節でも述べたように，連続性の定義は一般的に知られた $\varepsilon - \delta$ 論法と同じ論理式であるが，演算が図 6.1 のものである．

6.1.2 HydLa の制約

本小節では HydLa における制約の Coq 実装を紹介する．Coq には標準で論理式が定義されているが，HydLa では時相論理演算子が使われているため新たに定義する必要がある．図 6.3 に制約とその略記の実装を示す．制約の実装は Coq 標準の論理式である

Prop を基にしている．

6.2　軌道の性質

本節では 6.1 節で定義した軌道に関する性質を見る．

まずは実数上の関数を軌道に埋め込むことを考える．実数上の関数 $\mathbb{R} \to \mathbb{R}$ は軌道 $\mathbb{R} \to \mathbb{R} \cup \{ \bot \}$ のサブタイプであり，自然に軌道と考えることができる．ここではこの自然な埋め込みに対して，極限と連続性は同値であり，微分可能性は一方向で保たれることを紹介する．定理を図 6.4 に示す．図 6.4 で，limit1_in, D_x, no_cond, continue_in, D_in は Coq 標準の関数である．定理 preserve_limit は実数上の関数 f について，f の点 a での極限が l であるかどうかが，f を実数上の関数と考えた時と軌道と考えた時で同値であることを言っている．preserve_continuity は実数上の関数 f について，f が点 a で連続かどうかが，f を実数上の関数と考えた時と軌道と考えた時で同値であることを言っている．preserve_deriv は実数上の関数 f について，関数 d が点 a のまわりで f の微分ならば，軌道として見たときも同様であることを言っている．preserve_deriv のみ同値でないのは，軌道の微分が本来の意味で微分可能でなくてもよくなっているためである．図 6.4 より得られる重要な知見として，軌道について新たに極限，連続性，微分を定義したものの，実数値をとる領域では実数上の関数の時と同様に考えて良いということが挙げられる．

次は反対に，軌道が値として $\bot$ を取る点での連続性や微分について見る．定理を図 6.5 に示す．定理 not_left_cont_at_bottom，not_right_cont_at_bottom, not_cont_at_bottom はそれぞれ，値が $\bot$ になる点では左連続性，右連続性，連続性は成り立たないことを言っている．また定理 deriv_bottom_at_bottom は，値が $\bot$ になる点ではその微分も値が値が $\bot$ になることを言っている．

6.3　例題

本節では図 6.6 のプログラムに対して，床を跳ねる軌道が解となることの Coq による証明について述べる．図 6.6 では初期値をパラメータ化し，簡単のため反発係数は 1 としている．

また証明にあたり以下の簡略化をした．

- 全称量化されている変数については，該当するものを列挙した．ただし時間につい

ては列挙できないため，フェーズ毎にまとめて証明した．

- 実数の計算や連続性に関する証明は省略した．特に連続性は図 6.4 の定理 `preserve_continuity` より明らかである．

まずはプログラムおよび解について図 6.7 に示す．ただし，`list (set formula)` 型を `constraint` 型とおいて，“`~~>`” は `string` 型から `constraint` 型への対応，“`|-->`” は `option R` 上の対応を表す．“`f|_(a,b)`” は軌道 `f` の開区間 `(a,b)` への制限を表す．

次に解であることを確認するための命題を見ていく．まず時刻 t に関係のない $(i), (ii)$ について，図 6.8 に示す．図 6.8 で `unfold_some` は `Q` が `string -> option constraint` 型であり，`Some` を剥がすためだけに使われている．`c_eq`, `c_subset` はそれぞれ `constraint` 型の等価性と包含の判定である．`Is_true` は `bool` 型を `Prop` 型に変換している．

最後に時刻に関係のある (iii) について見る．すでに触れたように時刻で全称量化された部分はフェーズ毎に証明した．ここでは PP1 のみを図 6.9 に示す．図 6.9 で，`f2p` は `formula` 型から `Prop` 型への変換，`andl` はリストの要素の連言を取る関数，`ssubset` は `set string` 型の包含判定，`set_mem Feq_dec x X` は `x` が `X` の要素かの判定である．IP では，PP では `let` 句で指定していた時刻を，“`0 < t < sqrt(p/5) ->`” のように仮定で指定する．

```
1  Definition minus (x : option R) (y : option R) : option R :=
2    match (x, y) with
3    | (Some x', Some y') => Some (x' - y')
4    | (_, _) => None
5    end.
6
7  Definition mul (x : option R) (y : option R) : option R :=
8    match (x, y) with
9    | (Some x', Some y') => Some (x' * y')
10   | (_, _) => None
11   end.
12
13 Definition div (x : option R) (y : R) : option R :=
14   match x with
15   | Some x' => Some (x' / y)
16   | _ => None
17   end.
18
19 Definition abs (x : option R) : option R :=
20   match x with
21   | Some x => Some (Rabs x)
22   | None => None
23   end.
24
25 Definition lt (r1 r2 : option R) : Prop :=
26   match (r1, r2) with
27   | (Some x', Some y') => x' < y'
28   | (_, _) => False
29   end.
```

図 6.1　$R \cup \{ \perp \}$ での演算の実装

```
1  Definition partial := R -> option R.
2
3  Definition leftLim (f : partial) (a : R) (l : option R) : Prop
4    := forall eps : R, eps > 0 ->
5      exists del : R, del > 0 /\
6        forall x : R, 0 < a - x < del
7          -> lt (abs (minus (f x) l)) (Some eps).
8  Definition rightLim (f : partial) (a : R) (l : option R) : Prop
9    := forall eps : R, eps > 0 ->
10     exists del : R, del > 0 /\
11       forall x : R, 0 < x - a < del
12         -> lt (abs (minus (f x) l)) (Some eps).
13 Definition lim (f : partial) (a : R) (l : option R) : Prop
14   := forall eps : R, eps > 0 ->
15     exists del : R, del > 0 /\
16       forall x : R, 0 < Rabs(x - a) < del
17         -> lt (abs (minus (f x) l)) (Some eps).
18
19 Definition leftCont (f : partial) (x : R) : Prop
20   := leftLim f x (f x).
21 Definition rightCont (f : partial) (x : R) : Prop
22   := rightLim f x (f x).
23 Definition cont (f : partial) (x : R) : Prop
24   := leftCont f x /\ rightCont f x.
25
26 Definition sim (f g : partial) (x : R)
27   := (leftCont f x <-> leftCont g x)
28     /\ (rightCont f x <-> rightCont g x).
29
30 Definition sec (f : partial) (a : R)
31   := fun x => div (minus (f x) (f a)) (x - a).
32 Definition D (f d : partial) (a : R) : Prop :=
33   lim (sec f a) a (d a)
34   \/ (~ lim (sec f a) a (d a) /\ cont f a /\ d a <> None)
35   \/ (~ cont f a /\ d a = None).
```

図 6.2　軌道の実装

```
1  Inductive formula := I (p : Prop) | A (f g : formula)
2    | T (p : Prop) (f : formula) | S (f : formula).
3
4  Notation "'$' p" := (I p) (at level 75).
5  Notation "f '&' g"
6    := (A f g) (at level 80, right associativity).
7  Notation "p '~>' f"
8    := (T p f) (at level 85, right associativity).
9  Notation "'[]' f" := (S f) (at level 100).
```

図 6.3　HydLa の制約の実装

```
1  Theorem preserve_limit:
2    forall (f : R -> R) (a l : R),
3      limit1_in f (D_x no_cond a) l a
4        <-> lim (fun x => Some (f x)) a (Some l).
5
6  Theorem preserve_continuity:
7    forall (f : R -> R) (a l : R),
8      continue_in f no_cond a <-> cont (fun x => Some (f x)) a.
9
10 Theorem preserve_deriv:
11   forall (f d: R -> R) (a : R),
12     D_in f d no_cond a
13       -> D (fun x => Some (f x)) (fun x => Some (d x)) a.
```

図 6.4　実数上の関数を軌道に埋め込んだときの性質

```
1  Theorem not_left_cont_at_bottom:
2    forall (f : partial) (a : R),
3      f a = None -> not (leftCont f a).
4  Theorem not_right_cont_at_bottom:
5    forall (f : partial) (a : R),
6      f a = None -> not (rightCont f a).
7  Theorem not_cont_at_bottom:
8    forall (f : partial) (a : R), f a = None -> not (cont f a).
9
10 Theorem deriv_bottom_at_bottom:
11   forall (f d : partial) (a : R),
12     D f d a /\ f a = None -> d a = None.
```

図 6.5　値が ⊥ となる点での軌道の性質

```
1  INIT <=> y > 0 & y' = 0.
2  FALL <=> [](y'' = -10).
3  BOUNCE <=> [](y- = 0 => y' = -y'-).
4  INIT, FALL << BOUNCE.
```

図 6.6　床を跳ねるボールのプログラム

```
Definition INIT := [[$ gt (x t) (Some 0); $ x'(t) = Some 0]].
Definition FALL := [[[]$ x''(t) = Some(-10)]].
Definition BOUNCE
  := [[[]prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t))]].

Definition Q : history :=
  "INIT" ~~> [[$ gt (x t) (Some 0); $ x'(t) = Some 0]; nil; nil; nil];
  "FALL" ~~> [[[]$ x''(t) = Some(-10); $ x''(t) = Some(-10)];
              [$ x''(t) = Some(-10)]; [$ x''(t) = Some(-10)];
              [$ x''(t) = Some(-10)]];
  "BOUNCE" ~~> [[[]prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
                 prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t))];
                [prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t))];
                [prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
                 $ x'(t) = mul (Some(-1)) (prevx'(t))];
                [prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t))]];
  "continuity" ~~> [[$rightCont x t; $rightCont x' t; $rightCont x'' t];
                    [$cont x t; $cont x' t; $cont x'' t];
                    [$cont x t; $rightCont x' t];
                    [$cont x t; $cont x' t; $cont x'' t]].

Variable p: R.
Definition y := Some 0 |--> Some p;
                (fun t => Some(p-5*t^2))|_(Some 0, Some(sqrt (p/5)));
                Some(sqrt (p/5)) |--> Some 0;
                (fun t => Some(4*sqrt(5*p)*t-5*t^2-3*p))
                  |_(Some(sqrt (p/5)), Some(3*sqrt(p/5))).
Definition y' := Some 0 |--> Some 0;
                 (fun t => Some(-10*t))|_(Some 0, Some(sqrt (p/5)));
                 Some(sqrt (p/5)) |--> Some(2*sqrt(5*p));
                 (fun t => Some(4*sqrt(5*p)-10*t))
                  |_(Some(sqrt (p/5)), Some(3*sqrt(p/5))).
Definition y'' := Some 0 |--> Some(-10);
                  (fun t => Some(-10))|_(Some 0, Some(sqrt (p/5)));
                  (fun t => Some(-10))|_(Some(sqrt (p/5)), Some(3*sqrt(p/5))).
Definition prevy := (fun t => Some(p-5*t^2))|_(Some 0, Some(sqrt (p/5)));
                    Some(sqrt (p/5)) |--> Some 0;
                    (fun t => Some(4*sqrt(5*p)*t-5*t^2-3*p))
                      |_(Some(sqrt (p/5)), Some(3*sqrt(p/5))).
Definition prevy' := (fun t => Some(-10*t))|_(Some 0, Some(sqrt (p/5)));
                     Some(sqrt (p/5)) |--> Some(-2*sqrt(5*p));
                     (fun t => Some(4*sqrt(5*p)-10*t))
                      |_(Some(sqrt (p/5)), Some(3*sqrt(p/5))).
Definition prevy'' := (fun t => Some(-10))|_(Some 0, Some(3*sqrt(p/5))).
```

図 6.7　プログラムと解の Coq での表現

```
1  Theorem semantics_i :
2    let init := unfold_some (Q "INIT"%string) in
3    let bounce := unfold_some (Q "BOUNCE"%string) in
4    let fall := unfold_some (Q "FALL"%string) in
5      Is_true (c_eq init (closure init)) /\
6      Is_true (c_eq fall (closure fall)) /\
7      Is_true (c_eq bounce (closure bounce)).
8
9  Theorem semantics_ii :
10   let init := unfold_some (Q "INIT"%string) in
11   let bounce := unfold_some (Q "BOUNCE"%string) in
12   let fall := unfold_some (Q "FALL"%string) in
13     Is_true (c_subset (closure INIT) init) /\
14     Is_true (c_subset (closure FALL) fall) /\
15     Is_true (c_subset (closure BOUNCE) bounce).
```

図 6.8　意味論 $(i), (ii)$ の Coq 実装

```
Theorem semantics_iii_s1_pp1 :
  let E := ["INIT"%string; "FALL"%string; "BOUNCE"%string] in
  let t := 0 in
  let x := y in let x' := y' in let x'' := y'' in
  let prevx := prevy in let prevx' := prevy' in let prevx'' := prevy'' in
  let QEt := [$ gt (x t) (Some 0); $ x'(t) = Some 0;
              []$ x''(t) = Some(-10); $ x''(t) = Some(-10);
              []prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              $rightCont x t; $rightCont x' t; $rightCont x'' t] in
  p > 0 -> andl (map f2p QEt).

Theorem semantics_iii_s2_1_pp1 :
  let E := ["INIT"%string; "FALL"%string; "BOUNCE"%string] in
  let t := 0 in
  let x := y in let x' := y' in let x'' := y'' in
  let prevx := prevy in let prevx' := prevy' in let prevx'' := prevy'' in
  let QEt := [$ gt (x t) (Some 0); $ x'(t) = Some 0;
              []$ x''(t) = Some(-10); $ x''(t) = Some(-10);
              []prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              $rightCont x t; $rightCont x' t; $rightCont x'' t] in
  p > 0 -> ~ exists E', ssubset E' MS = true /\ ssubset E E' = true
                     /\ E <> E' /\ andl (map f2p QEt).

Theorem semantics_iii_s2_2_pp1:
  let E := ["INIT"%string; "FALL"%string; "BOUNCE"%string] in
  let t := 0 in
  let QEt := fun (x x' x'' prevx prevx' prevx'' : partial) =>
              [$ gt (x t) (Some 0); $ x'(t) = Some 0;
              []$ x''(t) = Some(-10); $ x''(t) = Some(-10);
              []prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              $rightCont x t; $rightCont x' t; $rightCont x'' t] in
 p > 0 ->
  ~ exists (z z' z'' prevz prevz' prevz'' : partial), exists E',
  let x := z in let x' := z' in let x'' := z'' in
  let prevx := prevz in let prevx' := prevz' in let prevx'' := prevz'' in
    ssubset E' MS = true
    /\ (forall t', t' < t /\ z(t') = y(t') /\ z'(t') = y'(t') /\ z''(t') = y''(t'))
    /\ (~ sim y z t \/ ~ sim y' z' t \/ ~ sim y'' z'' t)
    /\ ssubset E E' = true
    /\ andl (map f2p (QEt x x' x'' prevx prevx' prevx'')).

Theorem semantics_iii_s3_pp1:
  let E := ["INIT"%string; "FALL"%string; "BOUNCE"%string] in
  let t := 0 in
  let x := y in let x' := y' in let x'' := y'' in
  let prevx := prevy in let prevx' := prevy' in let prevx'' := prevy'' in
  let QEt := [$ gt (x t) (Some 0); $ x'(t) = Some 0;
              []$ x''(t) = Some(-10); $ x''(t) = Some(-10);
              []prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              $rightCont x t; $rightCont x' t; $rightCont x'' t] in
  p > 0 -> prevx(t) = Some 0
    -> set_mem Feq_dec ($ x'(t) = mul (Some(-1)) (prevx'(t))) QEt = true.

Theorem semantics_iii_s4_pp1:
  let E := ["INIT"%string; "FALL"%string; "BOUNCE"%string] in
  let t := 0 in
  let x := y in let x' := y' in let x'' := y'' in
  let prevx := prevy in let prevx' := prevy' in let prevx'' := prevy'' in
  let QEt := [$ gt (x t) (Some 0); $ x'(t) = Some 0;
              []$ x''(t) = Some(-10); $ x''(t) = Some(-10);
              []prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              prevx(t) = Some 0 ~> $ x'(t) = mul (Some(-1)) (prevx'(t));
              $rightCont x t; $rightCont x' t; $rightCont x'' t] in
  p > 0 ->
  leftCont x t -> set_mem Feq_dec ($leftCont x t) QEt = true
  /\ leftCont x' t -> set_mem Feq_dec ($leftCont x' t) QEt = true
  /\ leftCont x'' t -> set_mem Feq_dec ($leftCont x'' t) QEt = true
  /\ rightCont x t -> set_mem Feq_dec ($rightCont x t) QEt = true
  /\ rightCont x' t -> set_mem Feq_dec ($rightCont x' t) QEt = true
  /\ rightCont x'' t -> set_mem Feq_dec ($rightCont x'' t) QEt = true.
```

図 6.9　意味論 (*iii*) の Coq 実装

第 7 章

まとめと今後の課題

7.1 まとめ

本研究では HydLa プログラムの等価性判定を目的とし，まず例題を用いた宣言的意味論の再検討を行った．検討の結果，軌道および暗黙の連続性について精密化が必要であることがわかった．特に暗黙の連続性については二つの修正方針があったが，将来的な操作的意味論との比較を見据え，動的な暗黙の連続性による解決を選択した．そして検討の結果に基づき，軌道および暗黙の連続性を動的に扱う宣言的意味論を提案し，提案した意味論においてプログラムの解集合の計算と等価性証明を行った．また等価性証明の系として，具体例においてではあるが HydLa におけるプログラム変換の安全性を確認することができた．

また本研究では複雑な宣言的意味論による解計算をより安全に行うため，Coq を用いた意味論の形式化と検証を行った．検証では新たに定義した軌道に関する性質の証明や，パラメータを含む HydLa プログラムの解の証明を行った．特に軌道については直感的には，本研究で定義した極限，連続性，微分について，実数値をとる領域では実数上の関数の通常の極限，連続性，微分と同様に考えて良いという性質を示した．またパラメータを含む HydLa プログラムの解の証明については，初期位置が具体値である任意の床を跳ねるボールのプログラムについても示せたことになる．

7.2 今後の課題

本研究では Coq を用いて意味論の形式化と解計算の証明を行ったが，一方で非形式的証明では示した解の一意性証明は形式的証明では行っていない．また非形式的証明では具

体的な 2 つのプログラムの等価性を証明したが，一般的なプログラムに対するプログラム変換の正当性は行っていない．この 2 つは比較的難しくない課題である．一方で，本研究で提案した暗黙の連続性を動的に扱う意味論によって HydLa の実装である HyLaGI の操作的意味論とのギャップは減少したものの，宣言的意味論と操作的意味論の間のギャップは大きい．その等価性証明もしくはどのように違うのかという議論のためには，操作的意味論においても検証や精密化などの更なるステップが必要と考えられる．

謝辞

本研究にあたって熱心なご指導をいただいた上田先生，議論をしてくださった先輩方，発表などで質問をくださった同期と後輩の皆様に感謝いたします．特に上田先生には研究以外でも，配属前から頻繁に出入りさせていただいたことや，山登りや温泉などの多くの影響をいただいたことなども感謝いたします．また長年に渡って多くの勉強会で共に勉強させていただいた皆様もありがとうございました．最後にこれまでの学生生活を支えてくださった家族に深く感謝いたします．

2021 年 1 月 山田悠之介

参考文献

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

[2] Thomas A. Henzinger. *The Theory of Hybrid Automata*, pp. 265–292. Springer Berlin Heidelberg, 2000.

[3] Luca P. Carloni, Roberto Passerone, Alessandro Pinto, and Alberto L. Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Foundations and Trends in Electronic Design Automation*, Vol. 1, No. 1/2, pp. 1–193, 2006.

[4] 上田和紀, 石井大輔, 細部博史. 制約概念に基づくハイブリッドシステムモデリング言語 HydLa. 第五回システム検証の科学技術シンポジウム, 2008.

[5] 上田和紀, 細部博史, 石井大輔. ハイブリッド制約言語 HydLa の宣言的意味論. コンピュータ ソフトウェア, Vol. 28, No. 1, pp. 306–311, 2011.

[6] A. Borning, B. Freeman-Benson, and M. Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, Vol. 5, No. 3, pp. 233–270, 1992.

[7] Vineet Gupta, Radha Jagadeesan, Vijay Saraswat, and Daniel G. Bobrow. Programming in hybrid constraint languages. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II*, pp. 226–251. Springer, 1995.

[8] 松本翔太. *Validated Simulation of Parametric Hybrid Systems Based on Constraints*. PhD thesis, 早稲田大学大学院, 2017.

[9] Yunosuke Yamada, Masashi Sato, and Kazunori Ueda. Constraint-based modeling and symbolic simulation of hybrid systems with HydLa and HyLaGI. In Roger Chamberlain, Martin Edin Grimheden, and Walid Taha, editors, *Cyber Physical Systems. Model-Based Design*, LNCS 11971, pp. 153–178. Springer International

Publishing, 2020.

[10] Nthan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In Amy P. Felty and Aart Middeldorp, editors, *CADE'15*, LNCS 9195, pp. 527–538. Springer, 2015.

[11] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dreach: δ-reachability analysis for hybrid systems. In Christel Baier and Cesare Tinelli, editors, *TACAS 2015*, LNCS 9035, pp. 200–205. Springer, 2015.

[12] Walid Taha, et al. Acumen: An open-source testbed for cyber-physical systems research. In Benny Mandler, et al., editors, *Internet of Things. IoT Infrastructures*, pp. 118–130. Springer, 2016.

[13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In N. Sharygina and H. Veith, editors, *CAV'13*, LNCS 8044, pp. 258–263. Springer, 2013.

[14] 山田悠之介, 上田和紀. 制約に基づくハイブリッドシステムモデリング言語 HydLa の宣言的意味論の拡張. 人工知能学会全国大会論文集, Vol. JSAI2020, pp. 2N5OS17b04–2N5OS17b04, 2020.

[15] Modelica Association. *Modelica – Unified Object-Oriented Language for Systems Modeling: Language Specification (Version 3.4)*, 2007.

[16] Timothy Bourke and Marc Pouzet. Zélus: A synchronous language with ODEs. In *HSCC'13*, pp. 113–118. ACM, 2013.

[17] E. A. Lee. Constructive models of discrete and continuous physical phenomena. *IEEE Access*, Vol. 2, pp. 797–821, 2014.

[18] 竹口輝, 和田亮, 松本翔太, 細部博史, 上田和紀. ハイブリッド制約言語プログラムのハイブリッドオートマトンへの変換アルゴリズム. In *The 29nd JSSST Annual Conference, 2A-3*, 2012.

[19] K. Appel and W. Haken. Every planar map is four colorable. part i: Discharging. *Illinois J. Math.*, Vol. 21, No. 3, pp. 429–490, 09 1977.

[20] Georges Gonthier. Formal proof—the four-color theorem. *Notices of the American Mathematical Society*, Vol. 55, No. 11, pp. 1382–1393, 2008.

[21] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and

Laurent Théry. A machine-checked proof of the odd order theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pp. 163–179. Springer Berlin Heidelberg, 2013.

[22] Sandrine Blazy and Xavier Leroy. Mechanized semantics for the clight subset of the c language. *Journal of Automated Reasoning*, Vol. 43, No. 3, p. 263–288, Jul 2009.

[23] Thomas C. Hales. Formal proof. *Notices of the American Mathematical Society*, Vol. 55, No. 11, pp. 1370–1380, 2008.

[24] Gerwin Klein. From a verified kernel towards verified systems. In *Proceedings of the 8th Asian Conference on Programming Languages and Systems*, APLAS'10, p. 21–33. Springer-Verlag, 2010.

[25] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How amazon web services uses formal methods. *Commun. ACM*, Vol. 58, No. 4, p. 66–73, March 2015.

[26] High-level tla+ specifications for the five consistency levels offered by azure cosmos db. `https://github.com/Azure/azure-cosmos-tla`.

[27] Tla+ in tidb. `https://github.com/pingcap/tla-plus`.

[28] Vernacular commands. `https://coq.github.io/doc/v8.11/refman/proof-engine/vernacular-commands.html`.

[29] The gallina specification language. `https://coq.github.io/doc/v8.11/refman/language/gallina-specification-language.html`.

[30] Tactics. `https://coq.github.io/doc/v8.11/refman/proof-engine/tactics.html`.

[31] Coq implementation of denotational semantics. `https://github.com/HydLa/Coq-Impl-of-Denotational-Semantics`.

発表論文

[1] Yunosuke Yamada, Masashi Sato, and Kazunori Ueda. Constraint-based modeling and symbolic simulation of hybrid systems with HydLa and HyLaGI. In Roger Chamberlain, Martin Edin Grimheden, and Walid Taha, editors, Cyber Physical Systems. Model-Based Design, LNCS 11971, pp. 153–178. Springer International Publishing, 2020.

[2] 山田 悠之介, 上田 和紀. 制約に基づくハイブリッドシステムモデリング言語 HydLa の宣言的意味論の拡張. 人工知能学会全国大会論文集, Vol. JSAI2020, pp. 2N5OS17b04–2N5OS17b04, 2020.

[3] 山田 悠之介, 上田 和紀. ハイブリッド制約処理系 HyLaGI への共通部分式除去を用いた式の簡約の導入. 第 21 回プログラミングおよびプログラミング言語ワークショップ, ポスター発表, 2019.