2015 年度 修士論文

ハイブリッド制約言語 HydLa 処理系における
数式処理と区間演算を組み合わせたシミュレーション実行

提出日： 2016 年 1 月 23 日
指導： 上田 和紀 教授

早稲田大学大学院 基幹理工学研究科
情報理工・情報通信専攻

学籍番号：5114F099-9
和田 努

概要

現代社会においてコンピュータが担う役割は非常に大きく，様々なシステムがコンピュータによって制御されている．その中には原子力発電所の制御棒や自動車の自動ブレーキシステムのように人命に関わる重大なシステムも存在する．そのため，実世界とコンピュータとの間で相互作用するシステムであるサイバーフィジカルシステム（CPS）の研究は世界的にも重要な分野となっている．

CPS の多くはハイブリッドシステムとみなすことができる．ハイブリッドシステムとは時刻の経過に伴い状態や方程式が離散変化したり，状態が連続変化する動的システムのことを指す．

ハイブリッド制約言語 HydLa はハイブリッドシステムをモデリングすることを目的とした制約階層に基づく宣言型言語であり，制約を数学や論理学の記法を用いて表現することでモデルを簡潔に記述することができる．

HydLa の処理系 HyLaGI は数式処理を用いることで，計算誤差のないシミュレーションや，初期値に不定値を持つモデルの記号実行シミュレーションなどの特徴を有している．

ハイブリッドシステムの中には HyLaGI の数式処理シミュレーションでは扱うのが困難なものが存在する．困難な理由としては，モデルの挙動を示す微分方程式が解けないことや，離散変化を起こす時刻を解析的に求められないことが挙げられる．

本研究の目的は数式処理で扱うのが困難なモデルに対して厳密なシミュレーションが可能な手法を考案し，実装することである．本研究では精度保証数値計算手法の一つである区間演算を用いることで，離散変化時刻を解析的に求められないモデルに対するシミュレーションを行うことを考えた．区間演算として区間ニュートン法を採用したが，実用上の問題として，求めた区間解中に真の解が複数存在する可能性が否定できないことや，避けるべき 0/0 の計算を行ってしまうことがある．そこで，解の唯一性保証を行う手法や，0/0 計算を避けられる適切な値を探索する手法を考案することで，区間ニュートン法実用上の問題点を解決した．提案手法を HyLaGI に実装し，例題の評価を行ったところ，数式処理では離散変化時刻が求まらなかったモデルの中で，解軌道が一つに定まるモデルに対してシミュレーション可能になったことを確かめた．

Abstract

In modern society, computers hava a large role in controlling various systems. Some of them are safety-critical such as control lods in nuclear power plants and automatic braking system of automobiles. Therefore, the study of cyber-physical systems(CPSs), which combine cyber capabilities with physical capabilities, is an important field in the world.

Most CPSs are regarded as hybrid systems. Hybrid systems are dynamical systems with both continuous and discrete dynamic behavior.

The hybrid constraint language HydLa is a declarative programming language for modelling hybrid systems, which is based on constraint hierarchies. We can model hybrid systems easily by using constraints described in mathematical and logical notations.

HyLaGI, an implementation of HydLa, can simulate hybrid system models rigorously by formula manipulation and simulate models with indefinite initial values by symbolic execution.

Some of hybrid systems are difficult to simulate by the formula manipulation simulation of HyLaGI, because differential equations that describe the behavior of models are not always solvable and the time of discrete change cannnot always be solved analytically.

The purpose of this research is to design and implement a rigorous simulation method for models that are difficult to simulate by formula manipulation. In this research, we use interval arithmetic, a guaramnteed accuracy method of numerical calculation, to simulate models for which the time of discrete changes is hard to calculate with formula manipulation. We use the Interval Newton method to calculate the time of discrete changes, but it does not guarantee that there are exact solutions in interval solutions and may attempt to calculate 0/0.Therefore, We designed a method for guaranteeing that there is a unique exact solution in the interval solved by the Interval Newton method, and a method for avoiding the calculation of 0/0. We implemented thoese two methods in HyLaGI. As a result, HyLaGI is now able to simulate models that have only one solution trajectory and whose time of discrete changes is hard to calculate by formula manipulation.

目次

図目次

表目次

第 1 章

はじめに

1.1 研究の背景

現代社会においてコンピュータが担う役割は非常に大きく，様々なシステムがコンピュータによって制御されている．その中には原子力発電所の制御棒や自動車の自動ブレーキシステムのように人命に関わる重大なシステムも存在する．そのため，実世界とコンピュータとの間で相互作用するシステムであるサイバーフィジカルシステム（CPS）の研究は世界的にも重要な分野となっている．

CPS の多くはハイブリッドシステムとみなすことができる．ハイブリッドシステムとは時刻の経過に伴い状態や方程式が離散変化したり，状態が連続変化する動的システムのことを指す．ハイブリッドシステムは物理学や制御工学に適用可能な概念である．物理学では床を跳ねるボールの挙動やスイッチ付き電気回路がその例として挙げられる．制御工学では前述の原子力発電所の制御棒や自動車の自動ブレーキシステムが例として挙げられる．

ハイブリッドシステムをモデリングするツールとして，ハイブリッド制約言語 HydLa が存在する． HydLa はハイブリッドシステムをモデリングすることを目的とした制約階層に基づく宣言型言語である．システムの挙動を数学や論理学の記法で表現された制約として記述し，その優先度を設定することでモデルを簡潔に表現できることを特徴として有している．

上田研究室では HydLa の処理系である HyLaGI の開発が行われている．HyLaGI は数式処理を用いることで，計算誤差のないシミュレーションや，初期値に不定値を持つモデルの記号実行シミュレーションなどの特徴を有している．

ハイブリッドシステムの中には数式処理シミューレションで扱うのが困難なものが存在

する．例えば，万有引力のもとでの運動をシミュレーションすると，モデルの挙動を表す微分方程式の解析解が求まらずシミュレーションできなくなる．別の例として，近似した障害物のある単振り子の一周期の様子をシミュレーションすると，最初に障害物に引っかかる時刻を解析的に解くことができないためシミュレーションできなくなる．

1.2　研究の目的

本研究の目的は数式処理で扱うのが困難なモデルに対して厳密なシミュレーションが可能な手法を考案し，実装することである．本研究では精度保証数値計算手法の一つである区間演算を用いることで，離散変化時刻を解析的に求められないモデルに対するシミュレーションを行うことを考えた．区間演算として区間ニュートン法を採用したが，実用上の問題として，求めた区間解中に真の解が複数存在する可能性が否定できないことや，避けるべき 0/0 の計算を行ってしまうことがある．そこで，解の唯一性保証を行う手法や，0/0 計算を避けられる適切な値を探索する手法を考案することで，区間ニュートン法実用上の問題点の解決を図る．さらに提案手法を実装し，離散変化時刻を解析的に求められない例題に対して評価を行う．

本論文では，始めに背景としてハイブリッドシステム，HydLa，HyLaGI に関して説明したのち，区間演算を用いたシミュレーション手法に関して説明していく．

1.3　論文構成

本論文の構成を示す．2 章で，前提知識であるハイブリッドシステムの説明と，ハイブリッド制約言語 HydLa についての説明を行う．3 章で，HydLa の処理系である HyLaGI について，その特徴や実行アルゴリズムについて説明したのち，抱えている問題点について説明する．4 章では，3 章で説明した問題点を解決するための提案手法を，関連手法を交えて説明する．5 章では，4 章で説明した提案手法を実現するための実装について説明する．6 章では本論文における提案手法の評価を行った結果について説明する．7 章で関連研究について説明する．最後に，8 章で本論文のまとめと今後の課題について説明する．

第 2 章

ハイブリッド制約言語 HydLa

本章では，ハイブリッドシステムについて説明したのちに，HydLa について説明する．

2.1 ハイブリッドシステム

ハイブリッドシステム [4] とは時刻の経過に伴い状態や方程式が離散変化したり，状態が連続変化する動的システムのことを指し，物理学や制御工学に適用可能な概念である．ハイブリッドシステムの例としては以下に示すものが挙げられる．

- 床を跳ねるボールの動き（床との衝突が離散変化，自由落下が連続変化）．
- スイッチ付き回路（スイッチによる電流や電圧の変化を離散変化，それ以外は連続変化）．
- 自動車の自動ブレーキシステム（自動ブレーキが離散変化，それ以外は連続変化）．

2.2 HydLa の概要

HydLa[10] はハイブリッドシステムをモデリングするための宣言型言語である．HydLa ではモデルの挙動を数学や論理学の記法を用いて表現された制約として記述する．制約間に優先順位を設定することで，簡潔にモデルを表現できることを特徴としている．本論文では，HydLa によって記述されたモデルのことを HydLa プログラムと呼ぶ．

```
INIT <=> 9 < y < 11 & y' = 10.
FALL <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -(4/5) * y'-).

INIT, FALL << BOUNCE.
```

図 2.1　天井にボールが投げ上げられた時の様子を表した HydLa プログラム

2.3　HydLa の記法とモデリング例

本節では，HydLa の記法について具体例を用いて説明していく．HydLa で「天井にボールを投げ上げられた時の様子」をモデリングすることを考える．このモデルにおいて，ボールには常に重力加速度が加わっており，空気抵抗は無視できるほど小さいものとする．ボールの挙動としては次の 2 つが挙げられる．(1) 重力に従って落下していく．(2) 天井に衝突し，速度が反転する．前記 2 点の挙動は，ボールが天井に衝突する瞬間では同時に成り立たない挙動である．ボールが天井に衝突する瞬間では (2) の挙動が起きるため，(1) の挙動は瞬間的に無視されるべき挙動である．

以上を踏まえて，モデルを HydLa プログラムとして記述したものを図 2.1 に示す．HydLa プログラム中に出現する変数は，すべて時刻に関する関数である．本プログラムにおいて，プログラム中に出現する変数 y はボールの位置を表す変数として記述されている．記号 ' は時刻に関する微分を意味する記号であるため，プログラム中の y' や y'' はそれぞれボールの速度，加速度を表している．INIT，FALL，BOUNCE は制約名を表しており，同値記号 <=> によって定義されている．それぞれの制約について説明していく．

制約 INIT はボールの初期値に関する制約である．後述の記号 [] のついていない制約は「時刻 $t = 0$ の時のみ成り立つ」制約である．HydLa は，9 < y < 11 という記述があるように，不定値（以降，パラメータと呼ぶ）を記述することを許している．また，記号 & は論理積 $\wedge$ を表したものである．そのため，制約 INIT は，ボールの初期位置が 9 から 11 の間に存在し，初速度が 10 であることを意味する．

制約 FALL はボールが自由落下する挙動を示す制約である．記号 [] は時相論理における always を意味する記号で，「時刻 $t \geq 0$ において常に成り立つ」ということを意味する．したがって，制約 FALL は時刻 $t \geq 0$ において，ボールの加速度は常に -10 であるということを意味する．

制約 BOUNCE はボールが天井に衝突した時の挙動を示す制約である．記号 => が記述されている制約のことを条件付き制約と呼び，=> の前件のことをガード条件と呼ぶ．=> の後件に記述されている制約は，ガード条件が成立している場合のみ有効となる．記号 - は変数の左極限値を意味する記号である．したがって，制約 BOUNCE は時刻 $t \geq 0$ において，ボールの位置が 15 に達した時のみ，速度が反転（$-4/5$ 倍）することを意味する．

プログラム中の記述 INIT, FALL << BOUNCE は制約の優先度の設定を行っているものである．制約 FALL と BOUNCE は y-=15 において互いに矛盾を起こす制約となっている．そのため，矛盾が発生した場合に，どちらの制約を採用すべきかを記述する必要がある．記号 , や << は優先度の設定を行っている．, は同じ優先度であることを意味する記号で，本プログラムの場合，制約 INIT と BOUNCE が同じ優先度の制約であることを示している．<< は制約の優先順位を意味する記号で，本プログラムでは，制約 FALL と BOUNCE の間で優先順位が設定されており，BOUNCE の方が FALL より優先順位が高くなっている．そのため，FALL と BOUNCE の間で矛盾が発生した場合，BOUNCE が採用されることになる．この優先度の設定のため，天井にボールが達した (y-=15) 場合，ボールが跳ね返るという挙動が問題なく記述できている．

以上の記述によって，ボールが天井に投げ上げられた時の様子を HydLa プログラムとして正しく表現することができた．

2.4　HydLa プログラム例

本節では，HydLa プログラムの例題について説明する．なお，本節で説明する HydLa プログラムは 6 章での提案手法の評価に用いられる例題である．

2.4.1　Sin 波の形をした床の上で跳ねるボール

図 2.2 に本モデルの HydLa プログラムを示す．本モデルではボールに大きさはなく，質量は 1 で，重力加速度は -10 である．制約 INIT はボールの初期位置や初速度，反発係数に関する設定を行う制約である．制約 FALL はボールの自由落下に関する制約で，ボールの高さ方向 (y) の加速度を -10 と設定している．制約 CONSTX はボールの水平方向 (x) の加速度に関して設定している制約で，風など水平方向に関して影響を与える現象はないため，ボールの水平方向の加速度は 0 となっている．傾いた床を跳ねる際，衝突地点での床の接線と水平方向との関係が $\tan(\theta) = f(x_{prev})$ となる．（接戦と水平方向とのなす角を θ とする．）このことを考慮して衝突後のボールの水平方向と高さ方向の速度を

```
INIT <=> x = 0 & x' = 0 & y = 10 & y' = 0 & e = 1 & [](e' = 0).
FALL <=> [](cont = 1 => y'' = -10).
CONSTX <=> [](cont = 1 => x'' = 0).
CONTINUOUS <=> [](cont = 1).
CONST <=> [](s = cos(x-)/(1 + cos(x-)^2)^(1/2) &
c = 1      /(1 + cos(x-)^2)^(1/2)).
BOUNCE <=> []( y- = sin(x-) => cont = 0 &
x' = ((-e) * s^2 + c^2) * x'- + ((e+1) * s * c) * y'-
&
y' = ((e+1) * s * c) * x'- + (s^2 + (-e) * c^2) * y'-).

INIT, CONST, FALL, CONSTX, CONTINUOUS << BOUNCE.
```

図 2.2　Sin 波の形をした床で跳ねるボールの HydLa プログラム

考えると，式 2.1 のようになる．（式中の k は反発係数を意味する．）

$$\begin{cases} s = \dfrac{f'(x_{prev})}{\sqrt{1+(f'(x_{prev}))^2}} \\ c = \dfrac{1}{\sqrt{1+(f'(x_{prev}))^2}} \\ x'_{new} = (-k*s^2*c^2)*x'_{prev} + (k+1)*s*c*y'_{prev} \\ y'_{new} = (k+1)*s*c*x'_{prev} + (-k*s^2*c^2)*y'_{prev} \end{cases} \tag{2.1}$$

制約 CONST は式 2.1 中の s や，c の値を設定するための制約である．制約 BOUNCE は式 2.1 に従い，ボールが Sin 波の形をした床と衝突した際の挙動を示した制約である．なお，制約 CONTINUOUS は連続変化であることを示すための補助的な制約である．本モデルでは床との衝突時に速度の離散変化が発生するため，離散変化を表す制約である BOUNCE を連続変化であることを示す制約 CONTINUOUS より優先度を高く設定している．

2.4.2　自動車の自動ブレーキシステム

本モデルのイメージを図 2.3 に示す．本モデルは，自動車の自動ブレーキシステムを簡潔にモデル化したものである．自動車が壁から 100m 離れたところから，壁に向かって円軌道を描きながら進んでいく．ある程度壁から離れたところから自動ブレーキシステムが作動し，自動車は円軌道の接線方向に減速する．このとき，自動車が壁に衝突するかどうかを検証するのに用いるモデルである．

本モデルの HydLa プログラムを図 2.4 に示す．本プログラムでは，円の接線方向に関するブレーキを表現するために，極座標による表現を用いている．制約 INIT はモデル

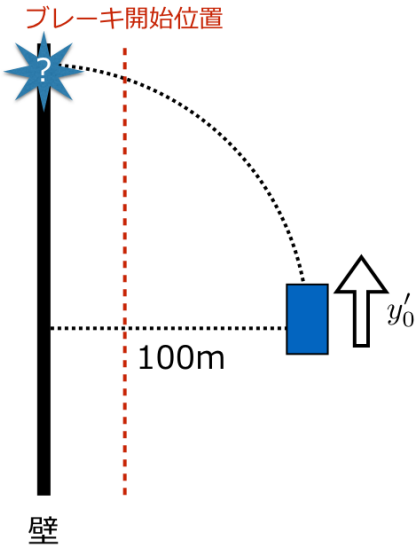

図 2.3　自動車の自動ブレーキシステム

の初期設定に関する制約である．変数 theta は自動車の位置を表す角度であり，半径は 100 で固定である．自動車の初速度は時速 40km と想定しており，INIT 中の thata' に関する式は，初速度に関して設定を行っている．変数 theta1 は theta の加速度を設定するための補助変数である．変数 collision は衝突したかどうかを表す変数であり，衝突時に 1 になるように設定する． 制約 X，Y は自動車の xy 平面上の座標を表現するための制約である．制約 CONST，CONST1 により，自動車の加速度が設定され，ブレーキが作動するまでは等速度運動であることを意味している．制約 CONST2 は変数 collision が衝突時まで変化しない変数であることを意味している．制約 BRAKE は自動ブレーキを表現した制約である．図 2.4 の場合，壁から x 座標換算で 20m 離れたところからブレーキが作動するようになっている．ブレーキの大きさに関しては，「時速 40km の自動車が 20m 進んだ地点で止まる」ほどの大きさであると想定して設定した．制約 COLL は衝突したことを示す制約である．壁に衝突した時点で，自動車は動かなくなることを意味している．制約 BRAKE と COLL は離散的な変化を表した制約であるため，関係する変数の連続的な変化に関する制約 CONST，CONST1，CONST2 よりも優先度を高く設定する．

```
INIT <=> theta1 = 0 & 100*theta' = (40000/3600) &
         theta = 0 & collision = 0.
X <=> [](x = 100*cos(theta)).
Y <=> [](y = 100*sin(theta)).
CONST <=> [](theta'' = theta1).
CONST1<=>[](theta1' = 0).
CONST2 <=> [](collision' = 0).
BRAKE <=> [](x- - 20 = 0 => 100*theta1 = - 1/40*(40000/3600)^2).
COLL <=> [](x- = 0  & collision- = 0 =>
              theta' = 0 & theta1 = 0 & collision = 1).

INIT, X, Y, (CONST, CONST1, CONST2) << (BRAKE, COLL).
```

図 2.4　自動ブレーキシステムの HydLa プログラム

第 3 章

HydLa 処理系 HyLaGI

HyLaGI[6] は HydLa の処理系であり, 数式処理を用いた記号実行シミュレータである. 本章では HyLaGI の概要や実行アルゴリズムについて説明したのち，従来の HyLaGI が抱えていた問題点について説明する.

3.1 HyLaGI の概要

HyLaGI は HydLa の処理系として，早稲田大学上田研究室で開発が進められている．HyLaGI は入力として HydLa プログラムを受け取り， HydLa プログラム内で記述された制約を満たす変数の軌道群を出力する．HyLaGI は数式処理ソルバとして Mathematica[?] を採用している．そのため，数式処理を用いた計算誤差のないシミュレーションや，初期値にパラメータを含むモデルの記号実行シミュレーション，非決定実行による複数の解軌道の出力が可能となっている．HyLaGI は C++ で実装されており，約 2 2,000 行の規模となっている.

3.2 実行例

図 2.1 の HydLa プログラムを HyLaGI で実行した結果について示していく．図 2.1 のモデルは初期値にパラメータが含まれているため，以下の 3 通りの場合分けが発生する.

a. ボールが天井に接する場合（図 3.1 の Case a）
b. ボールが天井に衝突する場合（図 3.1 の Case b）
c. ボールが天井に接しない場合（図 3.1 の Case c）

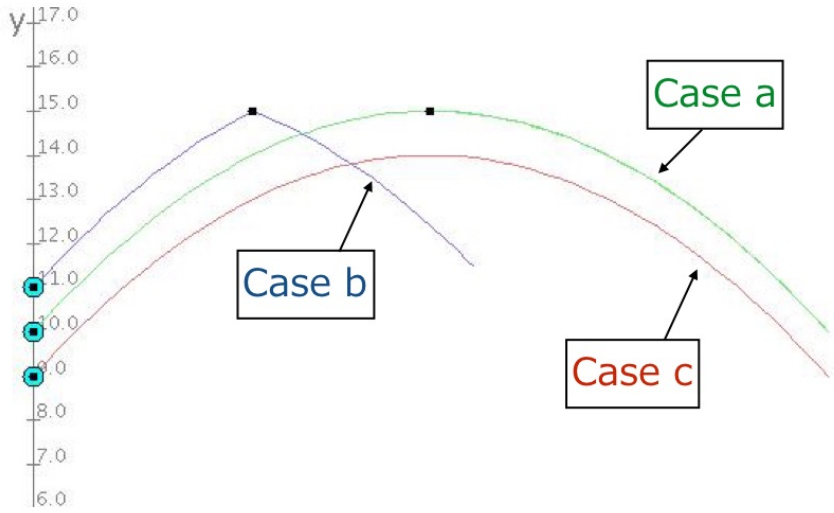

図 3.1　図 2.1 のシミュレーション結果

HyLaGI のシミュレーション結果を図示したものを図 3.1 に示す．さらに，シミュレーション結果のうち，a に対応する部分を図 3.2，b に対応する部分を図 3.3，c に対応する部分を図 3.4 に示す．これら実行結果より，HyLaGI のシミュレーションの特徴を具体的に説明する．HyLaGI の特徴の一つである計算誤差のないシミュレーションに関しては，それぞれの実行結果より読み取れる．例えば，図 3.2 より `PP3` における時刻 `t` の値 $1+(-1)*(-2+\frac{\texttt{p[y, 0, 1]}}{5})^{\frac{1}{2}}$ は数式処理によって実現できている計算誤差のない値である．HyLaGI では記号実行シミュレーションが可能であり，その特徴を示しているのは各実行結果中に出現する `p[y, 0, 1]` である．ボールの初期位置が HydLa プログラム中で $9 < \texttt{y} < 11$ と定義されているため，変数 `y` の初期値を `p[y, 0, 1]` という記号として扱いシミュレーションを行っている．さらに，記号 `p[y, 0, 1]` の値を数式処理を用いて適切に分割することによって，複数の解軌道が得られる非決定実行が行われている．Case a はボールが天井と接する場合であり，そのような場合のボールの初期位置は 10 の場合のみである．よって，実行結果である図 3.2 中では，記号 `p[y, 0, 1]` の値が 10 であると示されている．Case b はボールが天井と衝突する場合であり，この場合ボールの初期位置は $10 < \texttt{y} < 11$ である．実行結果を示している図 3.3 中では，記号 `p[y, 0, 1]` の値が $(10, 11)$ と開区間で表現されている．Case c はボールが天井と接しない場合であり，実行結果である図 3.4 中では，記号 `p[y, 0, 1]` の値が $(9, 10)$ となっている．

```
---------Case 1---------
---------1---------
---------PP 1---------
unadopted modules: {}
positive  :
negative  :
t : 0
y : p[y, 0, 1]
y': 10
y'': -10
---------IP 2---------
unadopted modules: {}
positive  :
negative  :
t : 0->1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)
y : 10*t+t^2*(-5)+p[y, 0, 1]
y': 10+t*(-10)
y'': -10
---------2---------
---------PP 3---------
unadopted modules: {}
positive  : y-=15=>y'=(-(4/5))*y'-
negative  :
t : 1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)
y : 15
y': 0
y'': -10
---------IP 7---------
unadopted modules: {}
positive  :
negative  : y-=15=>y'=(-(4/5))*y'-
t : 1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)->Infinity
y : 15+(-5)*(t+(-1)+(-2+p[y, 0, 1]*1/5)^(1/2))^2
y': (-10)*(t+(-1)+(-2+p[y, 0, 1]*1/5)^(1/2))
y'': -10
---------parameter condition(Case1)---------
p[y, 0, 1] : 10
# time reached limit
```

図 3.2　ボールが天井に接する場合の結果

```
---------Case 2---------
---------1---------
---------PP 1---------
unadopted modules: {}
positive  :
negative  :
t : 0
y : p[y, 0, 1]
y': 10
y'': -10
---------IP 2---------
unadopted modules: {}
positive  :
negative  :
t : 0->1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)
y : 10*t+t^2*(-5)+p[y, 0, 1]
y': 10+t*(-10)
y'': -10
---------2---------
---------PP 6---------
unadopted modules: {FALL}
unsat mod : {BOUNCE, FALL}
unsat cons : {y''=-10, y'=(-(4/5))*y'-}
positive  : y-=15=>y'=(-(4/5))*y'-
negative  :
t : 1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)
y : 15
y': 5^(-1/2)*(-8)*(-10+p[y, 0, 1])^(1/2)
---------IP 8---------
unadopted modules: {}
positive  :
negative  : y-=15=>y'=(-(4/5))*y'-
t : 1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)->Infinity
y : 15+5^(-1/2)*(-8)*(-10+p[y, 0, 1])^(1/2)*(t+(-1)+(-2+p[y, 0, 1]*1/5)^(1/2))+
    (-5)*(t+(-1)+(-2+p[y, 0, 1]*1/5)^(1/2))^2
y': 5^(-1/2)*(-8)*(-10+p[y, 0, 1])^(1/2)+(-10)*(t+(-1)+(-2+p[y, 0, 1]*1/5)^(1/2))
y'': -10
---------parameter condition(Case2)---------
p[y, 0, 1] : (10, 11)
# time reached limit
```

図 3.3　ボールが天井と衝突する場合の結果

```
---------Case 3---------
---------1---------
---------PP 1---------
unadopted modules: {}
positive  :
negative  :
t : 0
y : p[y, 0, 1]
y': 10
y'': -10
---------IP 4---------
unadopted modules: {}
positive  :
negative  :
t : 0->Infinity
y : 10*t+t^2*(-5)+p[y, 0, 1]
y': 10+t*(-10)
y'': -10
---------parameter condition(Case3)---------
p[y, 0, 1] : (9, 10)
# time reached limit
```

図 3.4　ボールが天井に接しない場合

3.3　実行アルゴリズム

HyLaGI の実行アルゴリズムを図 3.5 に示す．アルゴリズムの詳細に関しては文献 [6] にて与えられているため，ここでは本研究と深く関わりのある部分に関して説明する．

HyLaGI は離散変化についての計算を行うフェーズ Point Phase(PP) と連続変化について計算を行うフェーズ Interval Phase(IP) とを HyLaGI 実行時に指定した時刻 $MaxT$ になるまで交互に繰り返し計算することでシミュレーションを進めていく．図 3.5 の 9 行目から 12 行目までが PP について処理を行っている箇所で，14 行目から 19 行目までが IP について処理を行っている箇所である．8 行目で高階関数 *CalculateMSC* の引数として出現する関数 *CheckConsistencyPP* は PP において無矛盾性判定を行っている関数である．*CheckConsistencyPP* のアルゴリズムを図 3.6 に示す．*CheckConsistencyPP* は入力として与えられた制約ストア（制約の連言） S とパラメータに関する条件 CP から，S と CP を同時に満たすような変数値が存在するかを判定し，満たすかの真偽値とそのときのパラメータの条件の組を出力する．19 行目の処理では，IP において次の離

散変化が起こる時刻（以降，離散変化時刻と呼ぶ）を求めている．19 行目で現れる関数 *FindMinTime* のアルゴリズムを図 3.7 に示す．*FindMinTime* は入力として与えられた離散変化条件 *Trigger* とパラメータに関する条件 CP から，$(Trigger \wedge CP \wedge (t > 0))$ を満たす最小の時刻候補の式と，その式が最小となる条件の組のリストを出力する．

Input: HydLa プログラム *HydLaProgram*，シミュレーション終了時刻 *MaxT*

1: $MS := TopologicalSort(SolveCH(HydLaProgram))$
2: $M_{all} := MaxModuleSet(MS)$
3: $V := GetVariables(HydLaProgram)$
4: $T := 0;\quad S := true;\quad CP := true;\quad E := \emptyset$
5: **while** $T <_{CP} MaxT$ **do**
6: 　//PP
7: 　$S := SubstituteMinTime(S, T)$
8: 　$(S, CP, E, \_, \_, \_) := CalculateMSC(S, MS, E, CP, T, CheckConsistencyPP)$
9: 　**if** $S := true$ **then**
10: 　　*break*
11: 　**end if**
12: 　$(S, CP) := AddParameters(S, CP, V)$
13: 　//IP
14: 　$(S, CP, E, M, A_-, A_+) := CalculateMSC(S, MS, E, CP, T, CheckConsistencyIP)$
15: 　$S := SolveDifferentialEquations(S)$
16: 　**if** $S = false \ \vee \neg IsUnique(S, V)$ **then**
17: 　　*break*
18: 　**end if**
19: 　$(MinT, CP) := GetElement(CompareMinTime$
　　$(\{FindMinTime(S \wedge CP \Rightarrow g) \mid (g \Rightarrow c) \in A_-\})$
　　$\cup\{FindMinTime(S \wedge CP \Rightarrow \neg g) \mid (g \Rightarrow c) \in A_+\}$
　　$\cup\{FindMinTime(S \wedge CP \wedge M_-) \mid M_- \in (M_{all} \setminus M)\}$
　　$\cup\{FindMinTime(S \wedge CP \wedge \neg M_+) \mid M_+ \in M\}$
　　$\cup\{(MaxT - T, true)\}))$
20: 　$T := MinT + T$
21: **end while**

図 3.5　HyLaGI の実行アルゴリズム（文献 [6]）より引用

Input: 制約ストア S，パラメータに関する条件 CP
Output: 充足可能性，パラメータに関する条件
1: $V := GetVariables(S)$
2: $CP_{tmp} := \exists V\,(S \wedge CP)$
3: **if** $CP_{tmp} = false$ **then**
4:　　**return**　$(false, CP)$
5: **else if** $CP_{tmp} = CP$ **then**
6:　　**return**　$(true, CP)$
7: **else**
8:　　**return**　$GetElement(\{(true, CP_{tmp}), (false, CP \wedge \neg CP_{tmp})\})$
9: **end if**

図 3.6　*CheckConsistencyPP* のアルゴリズム（文献 [6]）より引用

Input: 離散変化条件 $Trigger$，パラメータに関する条件 CP
Output: 時刻と条件の組のリスト
1: **return**　$GetInf(Trigger \wedge CP \wedge (t > 0))$

図 3.7　*FindMinTime* のアルゴリズム（文献 [9] より引用）

3.4　実行時の問題点

HyLaGI は数式処理を用いることで，計算誤差のないシミュレーションや，初期値にパラメータを含んだモデルの記号実行などを実現している．しかし，ハイブリッドシステムの中には数式処理で扱うことが困難なものが存在し，HyLaGI ではそのようなモデルに対してのシミュレーションを行うことができない．本節では，HyLaGI の数式処理シミュレーションにおける問題点を具体例を用いて説明する．

数式処理で扱うことが困難なハイブリッドシステムの特徴として以下の 2 点が挙げられる．

- 変数の挙動を表した微分方程式が解析的に解くことができない．
- 離散変化時刻を求めるための時刻に関する条件式を解析的に解くことができない．

高信頼なハイブリッドシステムのシミュレータ開発のためには，前記 2 つの問題点は解決すべきものであるが，本研究ではそのうち「離散変化時刻を求めるための時刻に関する方程式を解析的に解くことができない」という問題に対して着目していく．

離散変化時刻を求められないモデルの例として，「近似した障害物のある単振り子」が

挙げられる．このモデルの様子を図示したものを図 3.8 に，HydLa プログラムとして記述されたものを図 3.9 に示す．本モデルでの初期値や振り子の運動についてまとめたものを表 3.1 に示す．なお，本モデルでは変数の挙動を表す微分方程式が数式処理で解析的に求まる．HyLaGI を用いて，本モデルにおいて振り子の 1 周期にかかる時刻や，速度，角速度を調べるためにシミュレーションを行うとする．すると，振り子が最初に障害物に引っかかる際の離散変化時刻を求めるための条件式は式 3.1 のようになる．HyLaGI の数式処理シミュレーションにおいては，式 3.1 を満たす最小の時刻を解析的に求めることができない．したがって，シミュレーションが中断されてしまい，単振り子の挙動を求められない．

次章では前記の問題点を解決するための手法について説明する．

$$\begin{cases} \dfrac{1}{7}\left(\sqrt{5}\sin\left(\dfrac{7t}{\sqrt{5}}\right)+\dfrac{7}{180}\pi\cos\left(\dfrac{7t}{\sqrt{5}}\right)\right)>0 \\ t>0 \end{cases} \tag{3.1}$$

表 3.1　近似した障害物のある単振り子の設定

変数及び条件	初期値及び運動方程式
振り子の角度 θ	$\dfrac{\pi}{180}$
振り子の角速度 θ'	1
紐の長さ l_1	1（固定）
$\theta <= 0$ における紐の長さ l_2	$\dfrac{1}{2}$（固定）
振り子の質量	1（固定）
重力加速度 g	$\dfrac{98}{10}$（固定）
$\theta > 0$ において	$\theta'' = -\dfrac{g}{l_1}\theta$
$\theta < 0$ において	$\theta'' = -\dfrac{g}{l_2}\theta$

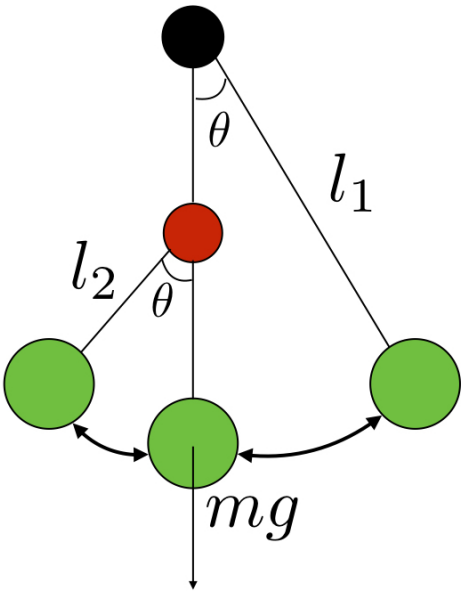

図 3.8　近似した障害物のある単振り子

```
INIT <=> theta = Pi/180 & theta' = 1.
CONS <=> [](l1 = 1 & l2 = 1/2 & g = 98/10).
F1 <=> [](theta- > 0 => theta'' = -g / l1 * theta).
F2 <=> [](theta'' = -g / l2 * theta).

INIT, CONS, F2 << F1.
```

図 3.9　近似した障害物のある単振り子の HydLa プログラム

第 4 章

区間演算を用いたシミュレーション手法

ハイブリッドシステムの中には，離散変化時刻を数式処理で求めることができないため HyLaGI でシミュレーションできないものが存在する．そこで本章では，離散変化時刻を区間演算を用いて求め，求めた時刻をシミュレーション上で利用するための手法について説明する．

4.1 区間演算および区間ニュートン法

本節では，提案手法で採用している区間演算について説明する．さらに，区間演算による求解手法の一つである区間ニュートン法及び拡張区間ニュートン法について説明する．

4.1.1 区間演算

区間演算 [7] とは精度保証数値計算の一つである．浮動小数点による誤差などを考慮し，真の解を必ず包含するような区間を求める手法である．区間は実数の閉区間として表現され，例えば $X = [\underline{X}, \overline{X}]$ のように表現する．なお，$\underline{X}$，$\overline{X}$ はそれぞれ区間 X の下限値，上限値を意味しており，$\underline{X}, \overline{X} \in \mathbb{R}$ である．区間同士の四則演算は式 4.1 で示すよう

に定義されている．

$$\begin{cases} X+Y = \left[\underline{X}+\underline{Y}, \overline{X}+\overline{Y}\right] \\ X-Y = \left[\underline{X}-\overline{Y}, \overline{X}-\underline{Y}\right] \\ X*Y = \left[min\left(\underline{X}*\underline{Y}, \underline{X}*\overline{Y}, \overline{X}*\underline{Y}, \overline{X}*\overline{Y}\right), \right. \\ \qquad\qquad \left. max\left(\underline{X}*\underline{Y}, \underline{X}*\overline{Y}, \overline{X}*\underline{Y}, \overline{X}*\overline{Y}\right)\right] \\ X/Y = X*\left[1/\overline{Y}, 1/\underline{Y}\right] (0 \notin Y) \end{cases} \tag{4.1}$$

また，区間同士の共通部分や和集合も定義されており，式 4.2 で示すような定義になっている．

$$\begin{cases} X \cap Y = \emptyset & (\overline{Y} < \underline{X} \ or \ \overline{X} < \underline{Y}) \\ X \cap Y = \left[max(\underline{X}, \underline{Y}), min\left(\overline{X}, \overline{Y}\right)\right] & (otherwise) \\ X \cup Y = \left[min(\underline{X}, \underline{Y}), max\left(\overline{X}, \overline{Y}\right)\right] \end{cases} \tag{4.2}$$

4.1.2　区間ニュートン法

区間ニュートン法 [7] は，近似求解手法であるニュートン法を区間演算に拡張させた手法である．近似解を導出するニュートン法と異なり，区間ニュートン法は真の解を含む区間を導出する．

実関数 f に対して区間ニュートン法を適用させるとする．ただし，関数 f は初期区間 $X^{(0)}$ において連続かつ微分可能で，f' の区間拡張 F' が $0 \notin F'\left(X^{(0)}\right)$ を満たすとする．このとき，式 4.3 で示した計算式を繰り返していくと，真の解へと収束していく区間を求めることができる．なお，式中の $X^{(k)}$ は区間ニュートン法の計算区間列を表し，$m(X)$ は区間 X 中の点であることを意味する．一般的に $m(X)$ は X の中点とすることが多い．

$$\begin{cases} X^{(k+1)} = X^{(k)} \cap N\left(X^{(k)}\right) & (k = 0, 1, 2, \dots) \\ N(X) = m(X) - F(m(X))/F'(X) \end{cases} \tag{4.3}$$

4.1.3　区間ニュートン法の性質

区間ニュートン法の性質として，区間ニュートン法によって得られた区間解中に真の解が唯一存在するというものがある．詳しい証明に関しては文献 [7] や文献 [2] で与えられているが，ここではその性質に関してのみ説明する．区間ニュートン法の適用対象の方程式を $f(x) = 0$ とする．区間ニュートン法の初期区間を $X^{(0)}$ とし，$f(x) = 0$ の $X^{(0)}$ 中の真の解を x_{true} とする．このとき，区間ニュートン法は以下に示す性質を持つ．

性質 1　解の唯一性
$0 \notin F'\left(X^{(0)}\right)$ かつ $N\left(X^{(k)}\right) \subset int\left(X^{(k)}\right)$ を満たす場合，計算区間列 $X^{(k)}$ 中に真の解 x_{true} が唯一存在する．

ここで，$int(X)$ は区間 X から端点を除いた区間のことを表す．

4.1.4　拡張区間ニュートン法

区間ニュートン法を用いることで，真の解を含む区間を求めることができるが，区間ニュートン法を適用できるのは $0 \notin F'\left(X^{(0)}\right)$ を満たす場合のみであり，実用上そのような区間 $X^{(0)}$ を見つけることは非常に難しい．そこで，文献 [7] や文献 [2] で与えられている，区間演算の除算の拡張を行うことで，$0 \in F'\left(X^{(0)}\right)$ を満たす場合でも計算できるようになった拡張区間ニュートン法 [7][2] を実用する．

通常の区間演算の除算では，分母に 0 を含むような区間が来た場合，計算することが不可能になる．しかし，文献 [7] や文献 [2] で与えられている式 4.4 に除算を拡張することで，分母に 0 を含むような区間が来た場合でも計算可能になる．

$$1/[c,d] = \begin{cases} [1/d, +\infty) & (c = 0 < d) \\ (-\infty, 1/c] \cup [1/d, +\infty) & (c < 0 < d) \\ (-\infty, 1/c] & (c < d = 0) \end{cases} \tag{4.4}$$

この拡張により，区間ニュートン法の計算結果は式 4.5 に示すように，二つの区間に分割される場合がある．

$$X^{(k+1)} = X_L \cup X_U \quad (X_L < X_U) \tag{4.5}$$

分割した区間それぞれに対して計算を続けていくことで，区間ニュートン法を用いて，初期区間 $X^{(0)}$ 中に存在するすべての解について求めることが可能となる．

4.2　提案手法

本節では，区間ニュートン法を用いた離散変化時刻を求める手法について説明する．

4.2.1　概要

従来の HyLaGI におけるシミュレーションでは，関数 *FindMinTime* において数式処理を用いて離散変化時刻を求めようとしていた．しかし，3.4 節で説明したように，シミュレーション対象のモデルによっては，離散変化時刻を解析的に求められないものが存

在する．そこで，*FindMinTime* 中の離散変化時刻を求める部分に関して数式処理を用いる代わりに区間ニュートン法を用いて求めることを考えた．

本手法では，*FindMinTime* を図 4.1 に示すアルゴリズム [5] に変更する．図 4.1 で示しているアルゴリズムでは，HyLaGI に区間ニュートン法を組み込むための様々な工夫が施されている．次節以降では図 4.1 のアルゴリズム中の核となる手法に関して説明していく．なお，本手法ではシミュレーション中に場合分けが起きるようなモデルに対しての考慮はしていない．

4.2.2　用語解説

提案手法の説明中に出現する用語について解説する．

原子ガード条件

原子ガード条件とは，ガード条件を構成する制約の最小単位のことを示す．例えば，ガード条件 $x > 0 \wedge y > 0 \wedge x^2 + y^2 = 1$ に対する原子ガード条件は，それぞれ $x > 0$，$y > 0$，$x^2 + y^2 = 1$ であると言える．

4.2.3　区間ニュートン法計算アルゴリズム

図 4.1 で示されているアルゴリズム *IntervalNewton* を図 4.2 に示す．本アルゴリズムは入力として離散変化時刻を求める対象となるガード条件 g，実行時における各パラメータが持つ条件 P，微分方程式の求解結果である S_t，区間ニュートン法の計算時に必要となる初期区間の下限値となる時刻 t_{start} を持つ．出力は区間ニュートン法によって求まった時刻のリスト $ITVs$ である．文献 [11] の時点では，条件を満たす最小時刻 1 つのみを求めていたが，他の制約との比較を考慮した結果，最小時刻候補のリストを出力するように変更した．基本的には式 4.3 や式 4.5 を実現しているアルゴリズムであるが，区間ニュートン法を実用する上で以下に挙げる 2 点に注意しなければならない．

1. 得られた解の中に真の解が唯一存在することを示す必要がある
2. $0/0$ を含む計算を考慮する必要がある

そこで，本アルゴリズムでは前記 2 点を考慮に入れた手法を導入している．それぞれについて詳しく説明していく．

Input: 微分方程式の求解結果 S_t，ガード条件 G，パラメータに関する条件 P，直前の時刻で成否が変化した原子ガード条件のリスト g_{border}

Output: ガードの成否が変化する最小の時刻 t_{min}，その時刻で成否の変化する原子ガード条件 g_{list}

1: $g_{list} := GetAtomicBorderConditions(G)$
2: $t_{list} := \emptyset$; $Map_g := \emptyset$
3: **for** $g \in g_{list}$ **do**
4: **if** $g \in g_{border}$ **then**
5: $t_{itv} := GetFirst(IntervalNewton(Derivative(g), P, S_t))$
6: $t_{start} := t_{itv}.lowerBound$
7: **else**
8: $t_{start} := 0$
9: **end if**
10: **if** $Solvable(g)$ **then**
11: $t_{list}.addEach(Solve(g, P, S_t, t_{start}), g)$
12: **else**
13: $itvs := IntervalNewton(g, P, S_t, t_{start})$
14: $t_{list}.addEach(ToParameters(itvs), g)$
15: **end if**
16: $Map_g.insert(g, TrueAtInitialTime(g, g_{border}, S_t))$
17: **end for**
18: **while** $t_{list} \neq \emptyset$ **do**
19: $(t_{min}, g_{list}, P) := PopMinimum(t_{list}, P)$
20: $(Map_g, satisfied) :=$
21: $CheckAndUpdateGuards(Map_g, g_{list}, t_{min}, G)$
22: **if** $satisfied = true$ **then**
23: $break$
24: **end if**
25: **end while**

図 4.1　区間ニュートン法を用いた *FindMinTime*（文献 [5] より引用）

解の唯一性保証

図 4.2 の 31 行目において，区間ニュートン法の計算停止条件として $Current = Prev$ を判定している．この条件は計算機の浮動小数点の限界を考慮して設定した．しかし，この条件は性質 1 で示した解の唯一性の条件を満たさない場合があり，得られた区間中に真の解が唯一存在することを示しきれていない．例えば，計算区間 $X^{(k)}$ の端点に真の解が存在する場合，解の唯一性の条件の一部である $N\left(X^{(k)}\right) \subset int\left(X^{(k)}\right)$ を満たさない．したがって，得られた区間中に真の解が唯一存在することを示す必要がある．解候補区間内 $Candidate$ に真の解が唯一存在することを示すためには，$Candidate$ を少しだけ広げた区間 T を用意し，T に対して区間ニュートン法を 1 ステップ進めた時に，T が計算結果の区間を強く包含していることを確認できれば良い．ただし，区間の広げ方に関しては以下の 2 点に注意しなければならない．

- 広げ幅が広すぎる場合，他の真の解を含む可能性がある．
- 広げ幅が狭すぎる場合，計算を行っても区間が縮まらない可能性がある．

そこで，適切な区間の広げ幅を探索によって求めることにした．

解の唯一性保証を行うアルゴリズムを図 4.3 に示す．このアルゴリズムは図 4.2 の 35 行目に現れる *ShowExistence* を表現しているものである．入力として解候補区間 $Candidate$ ，原子ガード条件 g，パラメータに関する条件 P，微分方程式の求解結果 S_t を持つ．出力は解候補区間内に真の解が唯一存在することを保証できたかの真偽値 $assurance$ である．

まず，与えられた解候補区間 $Candidate$ が縮退区間（上限と下限の値が等しい区間）であるかを判定し，縮退区間であった場合，0 以外の値で 1 つの値に収束したことになるので，$assurance$ を $true$ にする．次に，縮退区間でなかった場合の計算について説明する．始めに $Candidate$ に対する広げ幅の最大値 max_widen と最小値 min_widen をそれぞれ $2 * Width(Candidate)$ ，0 に設定する．ここで，$Width$ を与えられた区間の区間幅を返す関数である．図 4.3 の 6 行目から 27 行目は $assurance$ の値が定まるまで繰り返し行われる計算である．繰り返しの始めでは区間の広げ幅 $widen$ の値を $(max\_widen + min\_widen)/2$ とする．次に $Candidate$ を $widen$ だけ広げた区間として T を用意する．13 行目と 14 行目は区間ニュートン法の計算時に必要な値を計算しており, 分母部分にあたる $ResultD$ の中に 0 が含まれていた場合, 区間分割を起こし収束しないため, 広げ幅が大きすぎたと判断し, max_widen の値を $(max\_widen + min\_widen)/2$ とし，繰り返しの始めに戻る．19 行目では式 4.3 の計算を行っており，計算結果であ

る $ResultDivs$ が T に強く包含されていた場合，$Candidate$ の中に真の解が唯一存在することを示せたので，$assurance$ の値を $true$ にし，繰り返し計算を終了する．強く包含されていなかった場合，広げ幅が小さすぎたと判断し，min_widen の値を $(max\_widen + min\_widen)/2$ にし，繰り返しの始めに戻る．そして，7 行目の判定において，$max\_widen <= min\_widen$ を満たしてしまった場合，適切な広げ幅を見つけることができなかったと判断，つまり唯一性の保証に失敗したと判断し，$assurance$ を $false$ にして繰り返しを終了する．これらの処理により唯一性保証を行っている．

探索による 0/0 計算回避手法

節 4.1.4 において，区間演算の除算を分母に 0 を含むような場合でも行えるように拡張できることを説明した．しかし，実際に区間ニュートン法の計算を行っていると，分母だけではなく，分子にも 0 を含むような除算の計算を行ってしまう場合があり，その場合の計算結果に関しては一般的にも未定義になっているため計算を続行できなくなる．しかし，区間ニュートン法では式 4.3 に現れる x は対象区間 X に存在するどの値でもよく，必ずしも $Mid(X)$ でなくても良い．そこで，分母分子がともに 0 を含むような除算を行う恐れがある場合に，適切な x の値を探索する処理を加えることで，0/0 計算を除外することを考えた．

図 4.4 は 0/0 計算を除外する手法のアルゴリズムを示している．このアルゴリズムは図 4.2 の 16 行目に現れる $CalculateMidWithoutZero$ を表している．入力は区間ニュートン法の計算対象区間である $Current$ ，原子ガード条件 g，パラメータに関する条件 P，微分方程式の求解結果 S_t で，出力は $Current$ 中に存在する計算を行うのに適切な値 $valid$ である．

本アルゴリズムでは $valid$ を幅優先探索を用いて探索している．まずキュー $queue$ に数値の組 $(0, 1)$ を格納する．$queue$ に格納される数値の組は，区間中の 1 点を示すために必要な内分比 $rate$ を計算するのに用いられる．2 行目から 20 行目までの繰り返しは，$queue$ が空になるか，$valid$ の値が定まるまで行われる．始めに $rate$ の計算に必要な数値の組を $queue$ から取り出し，値の小さい方を $rate\_lower$，値の大きい方を $rate\_upper$ に代入する．これら 2 つの値から 4 行目で $rate$ を計算し，5 行目で内分点 mid を計算している．6 行目では計算した mid が適切な値かを判定している．区間ニュートン法計算時の除算の分子部分に該当する $CalculateIntervalValue(g, P, S_t, mid)$ に 0 が含まれているかどうかを判定しており，含まれていない場合，0/0 計算は起こらないので mid は適切な値であったと判断され，出力である $valid$ に代入され，繰り返しを終了する．$CalculateIntervalValue(g, P, S_t, mid)$ に 0 が含まれていた場合，$Current$ 中の別の点

を探索するための処理を行う．まず，$(rate\_upper - rate\_lower)$ を分割数 $STEP$ で割った値がしきい値 $THRESHOLD$ より小さいかどうかを判定し，小さければこれ以降の処理を行わず，別の値を探索するために繰り返しの始めに戻る．別の内分点を探索するために $queue$ に候補となる点に関する数値の組を格納していく．まず，13 行目で格納する数値の組の小さい方の値 $child\_rate\_lower$ に $rate\_lower$ の値を代入する．14 行目から 18 行目までの For ループによって，$rate\_lower$ から $rate\_upper$ までの値を $STEP$ の数だけ分割して，それぞれの数値の組を $queue$ に格納していく．21 行目から 23 行目までは $queue$ が空になるまたは $valid$ に値が代入された段階で行われる処理で，$valid$ の値が定まっていない場合，適切な値を見つけることができなかったと判断され，$valid$ に適切な値を見つけられなかったことを示す値 $InvalidMid$ を代入する．以上の処理により，0/0 計算を除外するのに適切な値を探索している．

4.2.4　数値計算によって発生する誤差への対応 [5]

4.2.3 節で示したアルゴリズムにより，離散変化時刻を真の解を唯一含んでいる数値区間として得られるようになった．しかし，得られた解区間を数式シミュレーションで使用すると，「求めた区間解がガード条件の成否の境界線上にあるかどうか」に関する場合分け（以降，「冗長な場合分け」と呼ぶ）が発生する．このような場合分けでは，シミュレーション結果として定性的に誤った解軌道を示してしまう恐れがある．そこで，このような冗長な場合分けを排除するための手法が必要となる．文献 [5] によって，冗長な場合分けを排除する手法は与えられている．本節ではその手法に関して説明する．

解区間の導入により冗長な場合分けが発生するのは，以下の 2 つの計算時である．

1. PP におけるガード条件判定時
2. $FindMinTime$ における各原子ガード条件の求解時

以降，それぞれに対応する手法に関して説明する．

PP におけるガード条件判定時

PP におけるガード条件判定は図 3.5 において $CheckConsistencyPP$ という関数で行っている．このガード条件判定時に，離散変化時刻の計算において成否が変化すると判明した原子ガード条件に関して，「各変数の左極限値は必ず境界線上に存在する」と前提しておくことによって，冗長な場合分けを除く．例えば，離散変化時に成否が変化する原子ガード条件が $x- = 0$ であると判明していた場合に，次の PP において $x- = 0$ を前

提として制約求解を行う．こうすることで，区間解がガード条件の境界をまたぐようなものだった場合でも，冗長な場合分けを避けることができる．

FindMinTime における各原子ガード条件の求解時

この問題に対する対策は，図 4.1 の 4 行目から 9 行目で行われている．直前の離散変化において成否が変化した原子ガード条件のリスト g_{border} に着目する．扱っている原子ガード条件 g が g_{border} 中に存在していた場合，g から得られる時刻に関する関数 $g(t) := lhs(g) - rhs(g)$ は，$g(0) = 0$ を満たす．$g(t)$ は連続かつ微分可能な関数なので，時刻 0 以降で $g(t) = 0$ を満たす時刻は，0 以降で最初に極値をとる時刻，つまり，$g'(t) = 0$ を満たす最初の時刻 t_{start} 以降に存在する．したがって，t_{start} の下限値を離散変化時刻の下限とすることで，冗長な場合分けを避けることができる．図 4.1 の 5 行目では，区間ニュートン法を用いて時刻の下限 t_{start} を求めている．

4.2.5　補助関数

各アルゴリズム中で出現する補助関数について説明する．なお，関数 *GetAtomicBorderConditions* ，*PopMinimum*，*CheckAndUpdateGuards* に関しては文献 [5] にて与えられている．

CalculateIntervalValue

入力として与えられた原子ガード条件 g と，パラメータに関する条件 P，微分方程式の求解結果 S_t から時刻に関する関数 $F(T)$ を生成し，入力として与えられている時刻 $Current$ を代入した値 $F(Current)$ を出力する関数.

DivideInterval

区間演算の拡張除算を実現している関数である．入力として，第 1 引数に分子，第 2 引数に分母が与えられる．出力は除算の結果である区間値を示す値 *ResultDivs* と区間分割が生じたかを示す真偽値 $two\_intervals$ との組を返す．$two\_intervals$ が $true$ の場合，$ResultDivs$ には 2 つの区間が格納されており，大きい値の区間は $larger$ に，小さい値の区間は $smaller$ に格納されている．$two\_intervals$ が $false$ の場合，$ResultDivs$ は 1 つの区間値のみを持ち，$unique$ に格納されている．

IsUnique

入力として与えられた変数が, なんらかの値を定義されているかどうかを判定する関数.

4.2.6　全体の流れ

前述までの説明を踏まえた上で，改めて提案手法全体のアルゴリズムである図 4.1 の説明をする.

まず，入力として与えられたガード条件 G から，G を構成する原子ガード条件のリスト g_{list} を関数 $GetAtomicBorderConditions$ によって取得する．t_{list} は各原子ガード条件の成否が変化する時刻のリストで，Map_g は各原子ガード条件の成否表である．3 行目から 17 行目までの処理を g_{list} に含まれているすべての原子ガード条件に対して行う．4 行目から 9 行目に関しては前述のように，離散変化時刻の下限を設定している．10 行目では対象となっている原子ガード条件が解析的に解けるかを判定しており，解ける場合は数式処理で，そうでない場合は区間ニュートン法を用いて離散変化時刻を求め，求めた時刻を t_{list} に追加していく．さらに，16 行目において，初期時刻における原子ガード条件の成否を Map_g に登録する．18 行目から 25 行目までの処理で，t_{list} の中から G の成否が変化する最小の時刻 t_{min} を求める．このようにして，ガード条件 G の成否が変化する最小の時刻を求める.

Input: 原子ガード条件 g，パラメータに関する条件 P，微分方程式の求解結果 S_t，区間ニュートン法計算時の初期時刻 t_{start}
Output: 時刻を表した区間のリスト $ITVs$

if $IsDefined\,(t_{start}) = true$ **then**
  $T_{init} := [t_{start}, t_{end}]$
else
  $T_{init} := [0, t_{end}]$
end if
$stack.push\,(T_{init})$
while $stack \neq \emptyset$ **do**
  $Current := stack.pop$
  for $i = 0$ to $MaxIteration$ **do**
    if $Current = InvalidInterval$ **then**
      break
    end if
    $Prev := Current$
    $ResultD := CalculateIntervalValue\,(Derivative\,(g)\,, P, S_t, Current)$
    if $0 \in ResultD$ **then**
      $mid := CalculateMidWithoutZero\,(g, P, S_t, Current)$
    else
      $mid := Mid\,(Current)$
    end if
    $ResultF := CalculateIntervalValue\,(g, P, S_t, mid)$
    $(ResultDivs, two\_intervals) := mid - DivideInterval\,(ResultF, ResultD)$
    if $two\_intervals = true$ **then**
      $Larger := ResultDivs.larger \cap Prev$
      if $Lager \neq InvalidInterval$ **then**
        $stack.push\,(Larger)$
      end if
      $Current := ResultDivs.smaller \cap Prev$
    else
      $Current := ResultDivs.unique \cap Prev$
    end if
    if $Current = Prev$ **then**
      break
    end if
  end for
  if $ShowExistence\,(g, P, S_t, Current) = true$ **then**
    $ITVs.push\,(Current)$
  end if
end while

図 4.2　区間ニュートン法計算アルゴリズム *IntervalNewton*

Input: 原子ガード条件 g，パラメータに関する条件 P，微分方程式の求解結果 S_t，解候補区間 $Candidate$

Output: 保証の成否 $assurance$

if $Candidate = DegenerateInterval$ **then**
  $assurance := true$
else
  $max\_widen := 2 * Width\left(Candidate\right)$
  $min\_widen := 0$
  while $true$ **do**
    if $max\_widen <= min\_widen$ **then**
      $assurance = false$
      break
    end if
    $widen = \left(max\_widen + min\_widen\right)/2$
    $T := [Candidate.lower - widen/2, Candidate.upper + widen/2]$
    $ResultF := CalculateIntervalValue\left(g, P, S_t, Mid\left(T\right)\right)$
    $ResultD := CalculateIntervalValue\left(Derivative\left(g\right), P, S_t, T\right)$
    if $0 \in ResultD$ **then**
      $max\_widen := \left(max\_widen + min\_widen\right)/2$
      continue
    end if
    $\left(ResultDivs, two\_intervals\right) := Mid\left(T\right) - DivideInterval\left(ResultF, ResultD\right)$
    if $ResultDivs \subsetneq T$ **then**
      $assurance = true$
      break
    else
      $min\_widen = \left(max\_widen + min\_widen\right)/2$
      continue
    end if
  end while
end if

図 4.3　解の唯一性保証

Input: 原子ガード条件 g，パラメータに関する条件 P，微分方程式の球界結果 S_t，区間ニュートン法の計算対象区間 $Current$

Output: 適切な $Current$ 中の値 $valid$

```
1:  queue.push((0, 1))
2:  while queue ≠ ∅ do
3:      (rate_lower, rate_upper) := queue.pop()
4:      rate := (rate_lower + rate_upper) /2
5:      mid := Current.lower * (1 − rate) + Current.upper * rate
6:      if 0 ∉ CalculateIntervalValue(g, P, S_t, mid) then
7:          valid := mid
8:          break
9:      else
10:         if (rate_upper − rate_lower)/STEP < THRESHOLD then
11:             continue
12:         end if
13:         child_rate_lower := rate_lower
14:         for i = 0 To STEP do
15:             child_rate_upper := rate_lower + (rate_upper − rate_lower) * (i + 1) /STEP
16:             queue.push((child_rate_lower, child_rate_upper))
17:             child_rate_lower := child_rate_upper
18:         end for
19:     end if
20: end while
21: if IsDefined(valid) = false then
22:     valid := InvalidMid
23: end if
```

図 4.4　探索による 0/0 計算回避手法

第 5 章

HyLaGI への実装

本章では，4 章で説明した提案手法を HyLaGI に実装する際に必要となる区間評価について説明する．

5.1 実装の概要

4 章で説明した提案手法では，数式の区間評価を行う必要がある．しかし，従来の HyLaGI では数式の区間評価を行うことができなかった．そこで，HyLaGI において区間評価を行うためのクラスを実装することで，提案手法を HyLaGI に実装する．次節以降では，HyLaGI で区間評価を行うために必要となったライブラリに関する説明と，実際に作成した区間評価を行うクラスに関して説明していく．

5.2 kv ライブラリ [3]

区間演算を実現しているライブラリの一つとして，kv ライブラリ [3] が存在する．kv ライブラリは，早稲田大学基幹理工学部応用数理学科の柏木雅英教授が開発を行っているライブラリで，C++ で区間演算を行えるライブラリとなっている．区間演算の四則演算はもちろんのこと，log や sin などの数学関数もサポートされている．さらに，式 4.4 で示した拡張除算もサポートしている．HyLaGI の実装も C++ で行われているため，kv ライブラリとの親和性が高く，本研究における区間演算の実現として採用した．

5.3　区間評価を行うクラス

従来の HyLaGI では，時刻や変数に関するパラメータは数式として扱う形で保持されていた．しかし，本研究によって区間演算を導入することとなったため，数式として扱われていたパラメータを区間として扱うための機能が必要となった．そこで，HyLaGI に新しく数式を区間値へと評価するクラス `IntervalTreeVisitor` と整数値の保持を目的としたクラス `IntervalOrInterger` クラスを作成した．それぞれのクラスについて説明していく．

5.3.1　IntervalTreeVisitor

`IntervalTreeVisitor` クラスは，数式を区間値に変換することを目的として作成されたクラスである．数式を表したデータ構造を走査し，最終的に区間値を返すメソッド *get_interval_value* を持つ．区間値の四則演算や三角関数，対数に関しては，5.2 節で説明した kv ライブラリを用いて実装している．しかし，冪演算に関しては，後述の `IntervalOrInterger` クラスを用いて，なるべく指数部に整数値が入るように工夫して使用している．

5.3.2　IntervalOrInterger

kv ライブラリの提供している冪演算を行う関数 *pow* は第 1 引数に底を，第 2 引数に指数部を適用させることで，冪演算の結果を返す．内部的な計算の際，第 2 引数の型によって *pow* 関数は異なる動作を行う．第 2 引数が整数型であった場合，*pow* 関数は内部関数である *pow_point* という関数を呼び出し，単純な乗算を繰り返すことで冪演算の結果を返している．第 2 引数が区間型であった場合，内部的には $\exp(y \log(x))$ という計算を行って冪演算の結果を返している．（ x は第 1 引数，y は第 2 引数を表している．）内部的に対数関数である *log* を用いているため，第 2 引数の区間中に負の値が存在していた場合，*pow* 関数は計算不能というエラーを返す．しかし，区間の中には上下限値が同じ値の区間である退廃区間というものも存在しており，*pow* 関数は第 2 引数に退廃区間が適用されている場合も同様に区間として扱ってしまう．そのため，例えば第 2 引数に $[-1, -1]$ という実質的に -1 として扱えるような値が適用されている場合，結果として x^{-1} を期待するが，区間中に負の値が存在すると判断され，エラーを返してしまう．このような予期しない *pow* 関数の挙動を避けるために，`IntervalTreeVisitor` 中で

整数値をなるべく長く保持するためのクラス `IntervalOrInterger` クラスを作成した．`IntervalOrInterger` クラスでは，メンバ $is\_integer$ によって，現在整数値を扱っているかどうかを判定しており，除算などにより整数値で保持出来なくなった場合に $false$ となる．`IntervalTreeVisitor` クラスでは，`IntervalOrInterger` クラスのメンバを持つことで，整数値の保持を行っている．これにより，前述のような予期しない pow 関数の挙動を避けられるようになった．

第 6 章

例題を用いた評価

本章では，5 章で実装した提案手法の評価を行い，その結果と考察について説明していく．なお，本章中では数値区間の表記に略記法を用いており，数値区間の上下限値の差異のみを区間として表記している．例えば，$[1.23456, 1.23465]$ という数値区間を $1.234[56, 65]$ と表記する．

6.1 障害物のある近似した単振り子

本モデルを用いて，単振り子の 1 周期にかかる時刻を求める．HyLaGI のシミュレーションにおいて，単振り子の 1 周期は 8 フェーズ分実行したことに相当する．

6.1.1 実行結果

図 3.9 の HydLa プログラムを，表 6.1 の設定で実行した結果を時刻についてまとめたものを表 6.2 に示す．

表 6.1 障害物のある近似した単振り子の設定 1

シミュレーション設定	設定値など
実行フェーズ数	8 フェーズ
離散変化ごとの区間評価	なし

表 6.2　障害物のある近似した単振り子の実行結果

障害物に引っかかった回数	前回との遷移時刻
1 回目	0.98610900550035[39, 7]
2 回目	0.70961344755686[37, 46]
3 回目	1.00354496157724[5, 8]
4 回目	0.70961344755686[37, 46]

6.1.2　考察

本モデルは，数式処理シミュレーションにおいては，最初に障害物に引っかかる時刻すら求められずに実行が終了してしまうモデルであった．区間ニュートン法を用いたアルゴリズムで実行したところ，表 6.2 に示すような結果が得られるようになった．また，この表より単振り子の 1 周期にかかる時刻を計算することができる．実際に 1 周期は 3 回目と 4 回目の障害物に引っかかった時の時刻の合計なので，1 周期の時刻は 1.7131584091341[08, 13] となる．

6.2　Sin 波の形をした床で跳ねるボール

本節では，例題として使用した図 2.2 のモデルに関する説明をし，例題を提案手法でシミュレーションした結果とその考察について説明していく．

6.2.1　例題説明

本モデルは Sin 波の形をした床にボールを落とした場合にどのような挙動になるかを調べるためのモデルとなっている．従来の HyLaGI では，最初に床を跳ねる時刻は解析的に求められるが，2 回目以降の床を跳ねる時刻を求めることが困難になるためシミュレーションが中断されてしまっていた．そこで，本研究の提案手法を用いることで，ボールが 2 回目に跳ねた以降の挙動について求める．

```
Root[{-Sin[10*Sqrt[2]*#1] + 5*#1 & ,
      0.20689248725348538525`19.315744720711198}]
```

図 6.1　数式処理で求まる 2 回目の衝突時の離散変化時刻

表 6.3　Sin 床の上で跳ねるボールのシミュレーション設定

シミュレーション設定	設定値など
実行フェーズ数	24
離散変化時の区間評価	毎回

6.2.2　問題箇所

本モデルを数式処理シミュレーションで実行した場合，ボールが 2 回目に床と衝突する時刻に関して図 6.1 に示すような式が求まる．この式は数式処理ソルバの Mathematica の関数 `Root` によって表現された値であり，$-\sin\left(10\sqrt{2}t\right)+5t^2=0$ の解であり，数値的には `0.20689248725348538525` の近傍に存在することを意味している．この式を扱って数式処理シミュレーションを継続させるのは困難なため，シミュレーションが中断されてしまう．

6.2.3　実行結果

図 2.2 のプログラムを表 6.3 の設定でシミュレーションを行った．前回の衝突からかかった時刻に関してまとめたものを表 6.4 に示す．衝突時の x 座標と y 座標に関してまとめたものをそれぞれ表 6.5 と表 6.6 に示す．また，シミュレーション結果から得られたボールの軌道を描いたものを図 6.2 に示す．

6.2.4　考察

実行結果より，提案手法によって Sin 床の上で跳ねるボールをシミュレーションした場合，10 回目の床との衝突まで実行できることが確認できた．しかし，ボールの位置を示す値である x や y の区間幅は衝突するたびにおよそ 10 倍になるなど指数的に増加しており，11 回目の離散変化時刻計算時には区間幅が広すぎてしまい区間ニュートン法で収束

表 6.4　Sin 床の上で跳ねるボールの時刻に関するシミュレーション結果

衝突回数	前回衝突からの時刻
0 回目から 1 回目	[1.414213562373095, 1.414213562373095]
1 回目から 2 回目	0.206892487253485[2, 6]
2 回目から 3 回目	2.7520159154525[18, 42]
3 回目から 4 回目	2.12183897665[237, 3837]
4 回目から 5 回目	2.40422971[5824132, 6146107]
5 回目から 6 回目	1.32085944[5359835, 8733315]
6 回目から 7 回目	2.57645[0707067337, 1363534535]
7 回目から 8 回目	0.2778[372488828096, 455919120477]
8 回目から 9 回目	2.655[036822670285, 218201912244]
9 回目から 10 回目	[2.281129188774213, 2.305392493084465]

表 6.5　Sin 床の上で跳ねるボールの x 座標に関するシミュレーション結果

衝突回数	x 座標	x 座標の区間幅
1 回目	0	0
2 回目	-2.9259016142698[21, 14]	6.66134e-15
3 回目	1.853991510811[773, 92]	1.47882e-13
4 回目	19.6710974257[01, 3891]	3.79075e-11
5 回目	3.08931623[2080733, 8230502]	6.14977e-09
6 回目	19.328649[52883397, 60527823]	7.64443e-08
7 回目	6.99092[2678922335, 9601693447]	6.92277e-06
8 回目	3.931[505538619339, 96570228105]	0.00020634
9 回目	13.64[391957092958, 606083915295]	0.00214127
10 回目	[-3.776719240361091, -3.26532302813464]	0.511396

表 6.6　Sin 床の上で跳ねるボールの x 座標に関するシミュレーション結果

衝突回数	y 座標	y 座標の区間幅
1 回目	[-2.220446049250313e-15, 2.220446049250313e-15]	4.44089e-15
2 回目	-0.2140225064096[707, 654]	5.32907e-15
3 回目	0.96016752686[14756, 27261]	1.25056e-12
4 回目	0.7321966074[403043, 934849]	5.31806e-11
5 回目	0.0522526[0442501423, 1820087468]	1.37759e-08
6 回目	0.460975[0067105445, 1016022194]	9.48917e-08
7 回目	0.6501[045595477351, 331265028654]	2.8567e-05
8 回目	-0.064[13765595406912, 05064374495731]	8.70122e-05
9 回目	0.8[76659643455568, 859525521598073]	0.00929291
10 回目	[-0.1127299230517131, 0.8388191963173419]	0.951549

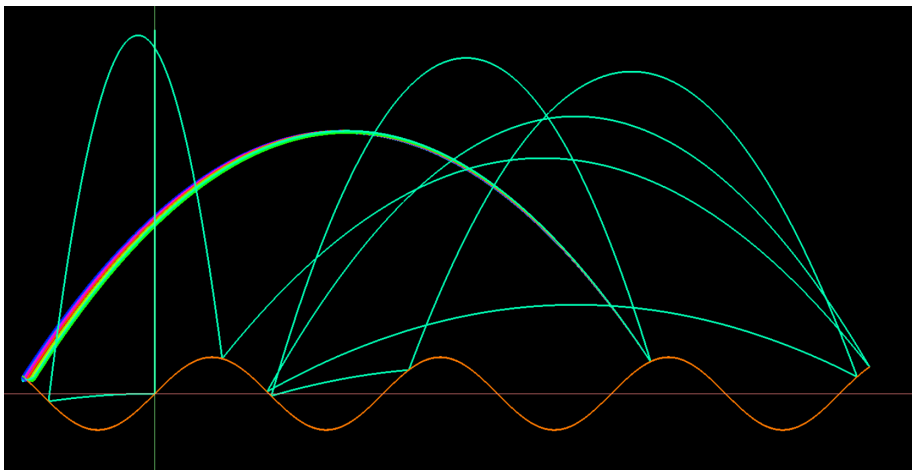

図 6.2　Sin 床の上で跳ねるボールの軌道

できなくなってしまった.

6.3　自動車の自動ブレーキシステム

6.3.1　問題説明

本例題のイメージおよびプログラムはそれぞれ図 2.3 と図 2.4 に示されている．本例題ではブレーキの作動する距離に関して 2 つの場合に分けてシミュレーションを行う.

- 壁から x 座標換算で 19m の地点からブレーキが作動する．
- 壁から x 座標換算で 20m の地点からブレーキが作動する．

なお，本例題に関しては 5 フェーズ実行すれば自動車が衝突したかどうかを判断できる．

6.3.2　実行結果 1：19m 地点でのブレーキ作動

19m 地点からブレーキを作動させた際のシミュレーション結果の 5 フェーズ目を図 6.3 に示す．この結果は制約 `COLL` が採用されていることを示しているため，この場合は自動車が壁に衝突してしまっていることを表している．

6.3.3　実行結果 2：20m 地点でのブレーキ作動

20m 地点からブレーキを作動させた際のシミュレーション結果の 5 フェーズ目を図 6.4 に示す．この結果は，自動車が 5 フェーズ目において制約 `BRAKE` が 2 回目に採用されていることを示している．したがって，壁との衝突は起こらなかったことを意味している．

6.3.4　考察

自動ブレーキの実行結果より，ブレーキが壁から 19m の位置で作動する場合，自動車は壁と衝突してしまう．20m の位置で作動する場合は，壁とは衝突を起こさないことがわかる．したがって，本モデルの設定の場合，19m と 20m の間で衝突を起こすか起こさないかの境界点が存在することが分かった．HyLaGI ではそのような境界点を探す際に，記号実行による非決定実行により，境界点を探し出せる特徴を持つが，本提案手法ではそのような非決定実行に対応できていないため，現状では境界点を正確に求めることはできていない．

```
---------PP 5---------
unadopted modules: {CONST, CONST1, CONST2}
unsat mod : {COLL, CONST2}
{COLL, CONST, CONST1, X, Y}
{COLL, CONST1}
{COLL, CONST, X, Y}
unsat cons : {collision'=0, collision=1}
{x=100*cos[theta], y=100*sin[theta], theta''=theta1, theta1'=0, theta'=0, theta1=0}
{theta1'=0, theta1=0}
{x=100*cos[theta], y=100*sin[theta], theta''=theta1, theta'=0, theta1=0}
positive  : x-=0&collision-=0=>theta'=0&theta1=0&collision=1
negative  :
t : p[t, -1, 1]+p[t, -1, 4]
collision : 1
width(collision): 0
theta : (36*p[t, -1, 1]+36*p[t, -1, 4]+(-5)*p[t, -1, 4]^2)*1/324
width(theta): 2.13163e-14
theta1 : 0
width(theta1): 0
x : 100*cos[(36*p[t, -1, 1]+36*p[t, -1, 4]+(-5)*p[t, -1, 4]^2)*1/324]
width(x): 2.15383e-12
y : 100*sin[(36*p[t, -1, 1]+36*p[t, -1, 4]+(-5)*p[t, -1, 4]^2)*1/324]
width(y): 1.42109e-14
theta': 0
width(theta'): 0
---------parameter condition(Case1)---------
p[t, -1, 1] : [12.41670762237453, 12.41670762237454]
p[t, -1, 2] : [14.13716694115407, 14.13716694115407]
p[t, -1, 3] : [7.199999999999988, 7.200000000000011]
p[t, -1, 4] : [2.843235238143711, 2.843235238143806]
```

図 6.3　19m 地点からブレーキを作動させた際のシミュレーション結果（5 フェーズ目）

```
---------PP 5---------
unadopted modules: {}
positive  : x--20=0=>100*theta1=(-1)/40*(40000/3600)^2
negative  :
t : p[t, -1, 1]+p[t, -1, 3]
collision : 0
width(collision): 0
theta : (36*p[t, -1, 1]+36*p[t, -1, 3]+(-5)*p[t, -1, 3]^2)*1/324
width(theta): 8.21565e-15
theta1 : (-5)/162
width(theta1): 3.46945e-18
x : 100*cos[(36*p[t, -1, 1]+36*p[t, -1, 3]+(-5)*p[t, -1, 3]^2)*1/324]
width(x): 8.49099e-13
y : 100*sin[(36*p[t, -1, 1]+36*p[t, -1, 3]+(-5)*p[t, -1, 3]^2)*1/324]
width(y): 2.55795e-13
collision': 0
width(collision'): 0
theta': (18+(-5)*p[t, -1, 3])*1/162
width(theta'): 6.52256e-16
theta1': 0
width(theta1'): 0
theta'': (-5)/162
width(theta''): 3.46945e-18
---------parameter condition(Case1)---------
p[t, -1, 1] : [12.32494565404109, 12.3249456540411]
p[t, -1, 2] : [14.13716694115407, 14.13716694115407]
p[t, -1, 3] : [7.19999999999999, 7.200000000000009]
p[t, -1, 4] : [17.8648943453993, 17.86489434539931]
```

図 6.4　20m 地点からブレーキを作動させた際のシミュレーション結果（5 フェーズ目）

第 7 章

関連研究

7.1 Acumen

Acumen[8] は CPS をシミュレーションし，それを 3D 画面上で描画するためのツールである．HydLa と同様にハイブリッドシステムのモデリングも可能である．

Acumen ではモデルの記述を独自の手続き型言語を用いて行っており，宣言型である HydLa との相違がある．また，制度保証は区間演算により行っており，数式処理も用いて制度保証している HyLaGI と異なっている．

7.2 Flow *

Flow *[1] はテイラーモデルベースの解析器で，非線形常微分方程式で表された連続システムやハイブリッドシステムを計算できるツールである．

Flow * におけるモデリング手法はハイブリッドオートマトンであり，HyLaGI とは異なる．また，制度保証を区間演算のみで行っている点も，HyLaGI とは異なる．

第 8 章

まとめと今後の課題

8.1 まとめ

本論文では，区間ニュートン法を用いた実行アルゴリズムを考案することで，離散変化時刻が解析的に求められないモデルをシミュレーション可能にすることを図った．区間ニュートン法を HyLaGI に導入するために，求められた区間解の中に真の解が唯一存在することを示す手法と探索による 0/0 計算回避手法を考案した．

考案した手法を用いて，離散変化時刻が解析的に求められなかった例題に対してシミュレーションを行ったところ，離散変化時刻が求まることを確認した．

8.2 今後の課題

8.2.1 区間ニュートン法適用の自動化

現状の実装では，区間ニュートン法を適用させる制約を手動で選択する必要がある．しかし，モデルによっては数式処理で解くべき制約と区間ニュートン法で解くべき制約が混じっているようなモデルも存在する．そのようなモデルに対してシミュレーションを行う際，実行前にどの制約が区間ニュートン法で解くべきかを判断することが難しく，場合によっては複数回実行することで把握することもある．そこで，区間ニュートン法で解くべき制約の判断を自動で行う手法を考案する必要がある．区間ニュートン法で解くべきかの判断は 2 通り考えられる．1 つ目は本論文においても扱っているような離散変化時刻が解析的に求まらないような場合である．この場合，数式処理ソルバより解析的に求められなかったことを示すエラーが出力されるので，このエラーを検出することで区間ニュートン法で解くべき制約であると判断する．2 つ目は理論的には解析的に求められるはずだが，

数式の複雑化の影響で現実的な時間で実行が終わらない場合である．この場合には，数式処理による求解に制限時間を設けることで，その制限時間を超えた場合，区間ニュートン法で解くようにするという手法が考えられる．

8.2.2　計算誤差を抑える手法の考案

区間ニュートン法の導入により，離散変化時刻が解析的に求められないモデルに対するシミュレーションが可能になったが，シミュレーションが進んでいくうちに区間幅が拡大していき，区間ニュートン法でうまく収束できなくなってしまうほどになってしまい，結果的にシミュレーションできなくなってしまう場合がある．そこで，現状よりも実行できるフェーズ数を増やすために，区間ニュートン法による計算誤差の拡大を抑える手法を考案する必要がある．

8.2.3　場合分けが生じるモデルへの対応

HyLaGI の数式処理シミュレーションの特徴の一つとして，非決定実行による異なる性質の解軌道をすべて導出できるという点がある．しかし，本論文におけるアルゴリズムは，非決定実行に関して考慮しておらず，場合分けの生じるモデルに対して対応できていない．そこで，場合分けが生じるモデルに対する区間演算を用いたシミュレーションアルゴリズムを考案する必要がある．

謝辞

3 年間にわたってご指導下さった上田和紀教授に心より感謝いたします．またご指導の他にも，たまにネパールレストランにご一緒させていただいたことにも感謝しております．それをきっかけに，インド・ネパールレストランへ積極的に行けるようになり，私の昼のレパートリーが広がりました．

私の研究のきっかけとなる区間ニュートン法について教えてくださった石井大輔さん，日々の研究の相談に乗ってくださった HydLa 班の皆様に感謝いたします．特に，研究について多大なるご協力をいただいた松本翔太さんに深く感謝いたします．頻繁によく分からないことを言って困らせてしまっていましたが，いつもその逞しい腕で様々な問題を解決していく姿に尊敬していました．これからもその腕を鍛えていって，いつかハイブリッドシステムを完全に制圧できる日が来るのを楽しみにしています．

日々の生活をともに過ごしていった同期と後輩の皆様にもそこそこ感謝しております．研究室の至る所にイタズラ装置を設置してみんなを楽しませていた伊藤剛史くんに感謝いたします．伊藤小商事にはお世話になりました！突然狂いだす様子の面白かった奈良耕太くんに感謝いたします．決闘はいつでも受けるぜ！修羅の国からやってきた三原色を使いこなす男である若槻祐彰くんに感謝いたします．卒業してからもハンティングしようぜ！また，自分の目標に向かって研究室から離れていった同期の赤池和喜くんにも感謝いたします．いつか本当に格闘やろうね！気分が乗らないときは Max なものでも飲んでテンションあげてこうね！吉元祐介くん．バイトやら研究室のことやら，あとは色々任せるよ小山峻平くん．遊びと研究のバランスに気をつけてね坂爪裕也くん．生活態度にもう少し気を配ろうね恒川雄太郎くん．HydLa 班は君に任せるよ別納健市くん．自分のように後輩を色んなお店に連れてってあげてね松澤望くん．

最後に，自分のことに盛大にちゃらんぽらんな私をいつでも見守ってくださった家族のみんなに心より深く感謝いたします．本当にありがとうございました．

2016 年 1 月　和田努

参考文献

[1] Chen, Xin; brahm, Erika; Sankaranarayanan, Sriram. Flow*: An analyzer for non-linear hybrid systems. In: Computer Aided Verification. Springer Berlin Heidelberg, 2013. p. 258-263.

[2] Chin-Yun Chen: Extended interval Newton method based on the precise quotient set, Computing, August 2011, Volume 92, issue 4, pp. 297–315.

[3] 柏木雅英：kv ライブラリ，早稲田大学基幹理工学部応用数理学科，2016, http://verifiedby.me/kv/.

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

[5] 松本翔太，上田和紀：ハイブリッドシステムのシミュレーションにおける制度保証数値計算と数式処理との連携，電子情報通信学会技術報告，2016.

[6] 松本翔太, 上田和紀: ハイブリッド制約言語 HydLa の記号実行シミュレータ Hyrose, コンピュータソフトウェア，Vol.30，No.4(2013)，pp.18–35.

[7] Ramon E.Moore, R.Baker Kearfott, Michael J.Cloud: Introduction to INTERVAL ANALYSIS, Society for Industrial and Applied Mathematics, Philadelphia, 2009.

[8] Taha, Walid, et al. A core language for executable models of cyber-physical systems (preliminary report). In: Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on. IEEE, 2012. p. 303-308.

[9] 松本翔太，櫻庭翔，高田賢士郎，細部博史，上田和紀：ハイブリッドシステムモデリング言語 HydLa の実装，日本ソフトウェア科学会第 28 回大会，2011.

[10] 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，SSV2008（第 5 回システム検証の科学技術シンポジウム），2008.

[11] 和田努，松本翔太，上田和紀：ハイブリッド制約言語 HydLa 処理系における数式処

理と区間計算を組み合わせたシミュレーション実行，人工知能学会第 29 回全国大会，1E3-3，2015.

[12] Wolfram Research, Inc., Mathematica, http://www.wolfram.com/mathematica/index.html, 2016.

発表文献および受賞

[1] 和田努，松本翔太，上田和紀：ハイブリッド制約言語 HydLa 処理系における数式処理と区間計算を組み合わせたシミュレーション実行，人工知能学会第 29 回全国大会，1E3-3，2015，登壇発表（筆頭著者，4 pages，学生奨励賞受賞）．

[2] 若槻祐彰，松本翔太，伊藤剛史，和田努，上田和紀：ハイブリッド制約処理系 HyLaGI による微笑誤差を用いたモデル解析，日本ソフトウェア科学会 第 32 回大会，2015，登壇発表（共著，9 pages）．