数式処理ソフトウェア sage を用いた ハイブリットシステムのモデリング

早稲田大学　基幹理工学研究科
情報理工・情報通信専攻 上田研究室
5123F084　冨永 帆高

2023/5/23

研究概要・背景

HyLaGIの主要関数

sageMathでの実装

まとめ・今後の課題

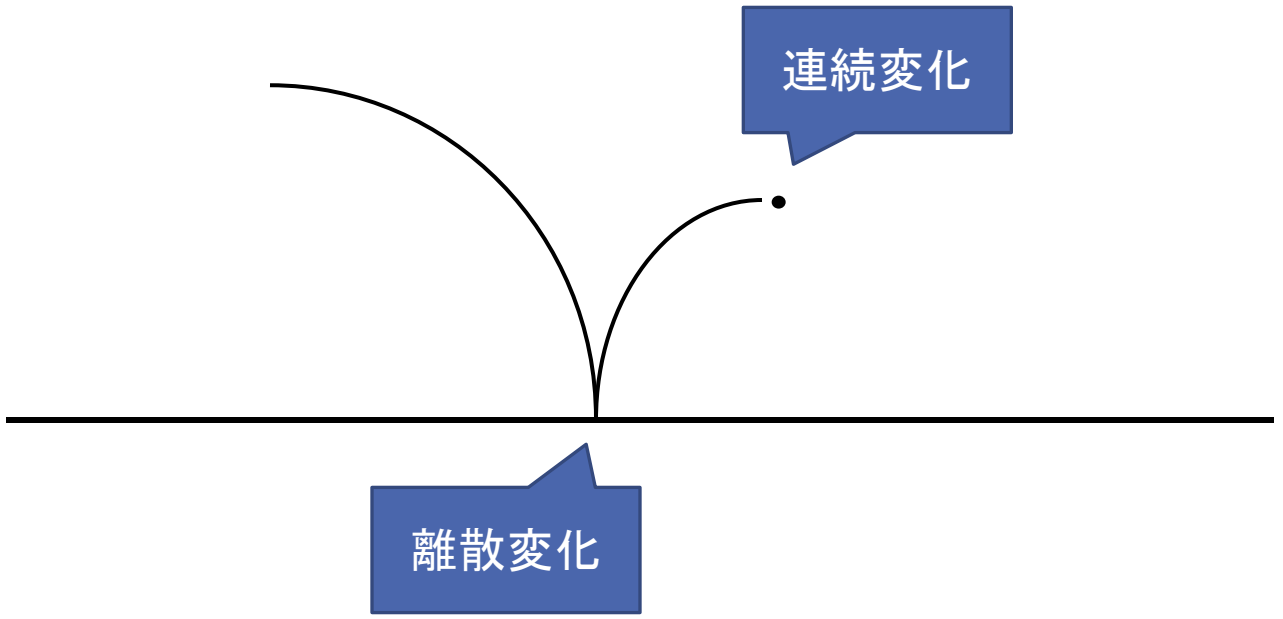
研究概要・背景

- 目的：
  ハイブリットシステムモデリング言語 HydLa のソルバとして
  mathematicaの代わりに数式処理ソフトウェアsageMathを
  使ってハイブリットシステムのモデリングを行う.

- その足がかりとしてHyLaGIの主要関数をsageMathを用いて記述する.

研究概要・背景

- ハイブリットシステムとは
  - 時間の経過に伴って状態が連続変化したり, 状態や方程式自体が離散変化したりするシステムのことを指す[1].
  - 例えば, 床を跳ねるボールは, ボールの位置は連続変化であるが, 床を跳ねる時の速度は離散変化である.



研究概要・背景

ハイブリットシステムモデリング言語 HydLa とは

- HydLa はハイブリットシステムを記述するための制約概念に基づいた宣言型言語である.
- HydLa プログラムの変数は時刻についての関数変数であり, システムが満たすべき制約を等式・不等式と微分方程式により記述することでモデリングを行う[2].
- 離散変化を扱うポイントフェーズ(PP), 連続変化を扱うインターバルフェーズ(IP)の計算を交互に行う.

例：高さ9~11から天井に向かって速度１０で鉛直投げ上げ

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

研究概要・背景

➢ハイブリットシステムモデリング言語 HydLa とは

- 用語
  - 制約: 等式・不等式, 微分方程式
    - ex) y’ = 10, 9 <= y <= 11, y- = 15
  - 制約モジュール: 制約同士を結合したり, 制約に時相演算子[]を加えたもの
    - ex) [](y’’ = -10), [](y- = 15 => y’ = -0.8y’-)
  - 解候補モジュール集合: 優先順位を満たす制約モジュールの集合
    - ex) {INIT, FALL, BOUNCE}, {INIT,BOUNCE}
  - 記号定数: 初期値に範囲が与えられ,
    値の定まっていないもの
    - ex) y

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

- HydLaプログラムは解候補モジュール集合の中から無矛盾で極大のものを選ぶ

研究概要・背景

- HyLaGI とは
  - HydLa言語の処理系
  - 記号計算による誤差がないことが保証されたシミュレーションを実現する
  - バックエンドソルバとして数式処理システムmathematicaを使用

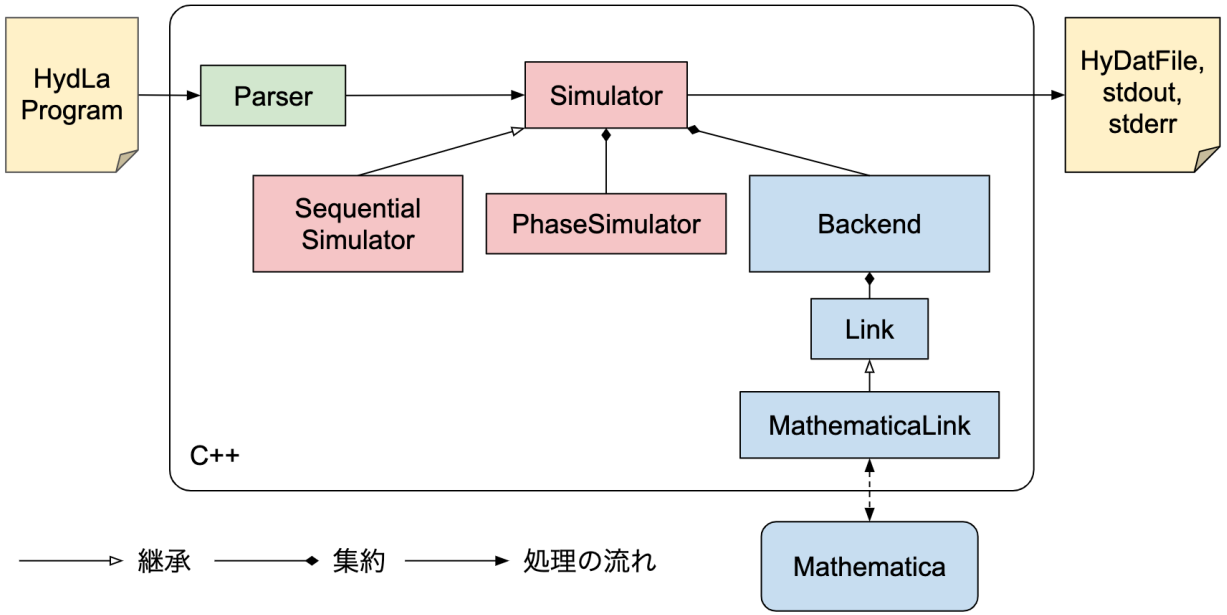


図:HyLaGI構成図

研究概要・背景

- 数式処理ソフトウェア sageMath について
  - 多くの数学ソフトウェアを統合したオープンソースの数学ソフトウェア
  - Pythonを使って実装されている
    - Python の文法でプログラムの記述が可能
  - 公式サイト
    - https://www.sagemath.org
  - チュートリアル
    - https://doc.sagemath.org/html/ja/tutorial/index.html

研究概要・背景

➢sageMathをソルバとするメリット

- オープンソースである.
  - エラーが起きた時に追うことができる.

- quantifier elimination(QE:限定記号消去)ができる
  - 限定記号付きの式を入力として
    入力と等価の限定記号なしの式を出力する

$$\exists x(x^2 - bx + c = 0) \Rightarrow b^2 - 4ac \geq 0$$

研究概要・背景

HyLaGIの主要関数

sageMathでの実装

まとめ・今後の課題

HyLaGIの主要関数

- ここで取り上げるHyLaGIの主要関数は以下の３つ
  - CheckConsistencyPP
    - PP(離散変化時)において与えた制約が成り立つのか判定
  - CheckConsistencyIP
    - 与えられた制約に含まれる微分方程式を解く
    - IP(連続変化時)において与えた制約が成り立つのか判定
  - FindMinTime・CompareMinTime
    - 次の離散変化の時刻を１つのモジュールのガード条件を解いて求める
    - 求めた離散変化の時刻を比べて最小の時刻を求める

HyLaGIの主要関数

➢CheckConsistencyPP

- 記号定数の条件CPと制約Sを同時に
  満たす制約S内の変数が存在するような
  記号定数の条件CP_tmpを求める
- CP_tmpがなければ (false, CP) を返す
  - 記号定数が範囲の中の任意の値で制約を満たさない.
- CP_tmpが与えた記号定数の条件と
  一致するならば (true, CP) をそのまま返す
  - 記号定数が範囲の中の任意の値で制約を満たす.
- CP_tmpが与えた記号定数の条件と
  一致しないならば (true, CP_tmp) か (false, CP∧¬CP_tmp)を非決定的に返す
  - 記号定数の範囲の中に制約を満たす値と制約を満たさない範囲がある.

Input: 制約ストア S, 記号定数の条件 CP
Output: 充足可能性, 記号定数の条件

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** $GetElement(\{(true, CP_{tmp}),$
9: $(false, CP \wedge \neg CP_{tmp})\})$
10: **end if**

HyLaGIの主要関数

- 例
  - 最初のPP
    - CP = 9 <= py <= 11 (py: yの記号定数)
    - S = {y = py, y' = 10, y'' = -10}
    - V = {y, y', y''}
    - CP_tmp = ∃V(S∧CP) = 9 <= py <= 11
    - CP_tmp = CP なので (true, 9<=py<=11)を返す.
  - ２回目のPP
    - CP = 10 <= py <= 11
    - S = {y = 15, y' = 2(-50+5py)^(1/2), y'' = -10, y' = -0.8y'-}
    - V = {y, y', y''}
    - CP_tmp = ∃V(S∧CP) = (py = 10)
    - (true, py=10)か(false, 10<py<=11)を返す.

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

Input: 制約ストア S, 記号定数の条件 CP
Output: 充足可能性, 記号定数の条件

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** $GetElement(\{(true, CP_{tmp}),$
9: $(false, CP \wedge \neg CP_{tmp})\})$
10: **end if**

HyLaGIの主要関数

➢CheckConsistencyIP

- CheckConsistencyPPとほとんど同じ処理を行なっている.

- 異なった処理を行なっている以下の２つ
  - はじめに制約Sに含まれる微分方程式を解いて時刻の式S_tを導出する
  - tが正の近傍のときに
    記号定数の条件CPとS_tを同時に満たす
    制約S内の変数が存在するような
    記号定数の条件CP_tmpを求める.

Input: 制約ストア S, 記号定数の条件 CP
Output: 充足可能性, 記号定数の条件
1: $S_t := SolveDifferentialEquation(S)$
2: $V := GetVariables(S_t)$
3: $CP_{tmp} := \exists V(Inf\{t \mid \exists_t(S_t \wedge t > 0)\} = 0) \wedge CP)$
4: **if** $CP_{tmp} = false$ **then**
5: **return** $(false, CP)$
6: **else if** $CP_{tmp} = CP$ **then**
7: **return** $(true, CP)$
8: **else**
9: **return** $GetElement(\{(true, CP_{tmp}),$
10: $(false, CP \wedge \neg CP_{tmp})\})$
11: **end if**

HyLaGIの主要関数

- 例
  - 最初のIP
    - S = {y = py, y’ = 10, y’’ = -10}
    - S内の微分方程式を解くと
      S_t = {y = py + 10t – 5t^2, y’ = 10 - 10t, y’’ = -10}
    - CP = 9 <= py <= 11
    - CP_tmp = 9 <= py <= 11
    - CP_tmp = CP なので
      (true, 9 <= py <= 11)

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

Input: 制約ストア S，記号定数の条件 CP
Output: 充足可能性，記号定数の条件

1: $S_t := SolveDifferentialEquation(S)$
2: $V := GetVariables(S_t)$
3: $CP_{tmp} := \exists V(Inf\{t \mid \exists_t(S_t \wedge t > 0)\} = 0) \wedge CP)$
4: **if** $CP_{tmp} = false$ **then**
5: **return** $(false, CP)$
6: **else if** $CP_{tmp} = CP$ **then**
7: **return** $(true, CP)$
8: **else**
9: **return** $GetElement(\{(true, CP_{tmp}),$
10: $(false, CP \wedge \neg CP_{tmp})\})$
11: **end if**

HyLaGIの主要関数

➢FindMinTime・CompareMinTime

- FindMinTime
  - CheckConsistencyIPで導出した時刻の式とガード条件から
    ガード条件を満たす最小時刻tを計算する.
  - 最小時間 t は必ず 0 より大きい.

- CompareMinTime
  - FindMinTimeで求めた最小時刻t同士を比較することで
    最小離散変化の時刻を導出する.
  - 記号定数の条件によって最小離散時間が変化する場合はそれぞれの場合の
    記号定数の条件を求める.

HyLaGIの主要関数

➢例

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

- CheckConsistencyIPで導出した時刻の式は以下のようになる.
  - y = py + 10t – 5t^2, y' = 10 - 10t, y'' = -10　(py: y の記号定数)
- 上の式とBOUNCEのガード条件を組み合わせて最小時間を計算(FindMinTime)
  - BOUNCE: (y- = 15) ∧ (y = py + 10t – 5t^2) => t = 1- (py/5 - 2)^(1/2)
- 記号定数の条件と最小離散時間を計算(CompareMinTime)
  - py >= 10 の場合は t = 1- (py/5 - 2)^(1/2) で天井に接触する.
  - py < 10 の場合は t = ∞ で離散変化が起こらない.

研究概要・背景

HylaGIの主要関数

sageMathでの実装

まとめ・今後の課題

sageMathでの実装

➢主要関数の実装

- 上田研究室のプロジェクトで過去に作られたpython２のプログラムを元に, **python３で動くように**, また, **2階微分方程式を解くことができるように** 改良をした.

- 約500行ほど

- gitlab branch python3debug
  - https://gitlab.ueda.info.waseda.ac.jp/hydla/wintercamp2019

sageMathでの実装

- CheckConsistencyPP
  - p5の例題を動かす
  - 最初のPP
    - CP = 9 <= py <= 11 (py: yの記号定数)
    - S = {y = py, y' = 10, y'' = -10}

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

```
== Check Consistency PP by QE ==========================================
==== input ======
constr store    :       [[y == p_y, dy == 10, ddy == -10]]
symbolic params :       [[p_y >= 9, p_y <= 11]]
# tmp_parameters is modified
==== output =====
consistency     :       [True, False]
updated params  :       [[py - 9 >= 0 /\ py - 11 <= 0], []]
==== measure ====
time            :       0.0954190000000147
```

sageMathでの実装

➢例

- 最初のPP
  - CP = 9 <= py <= 11 (py: yの記号定数)
  - S = {y = py, y' = 10, y'' = -10}
  - V = {y, y', y''}
  - CP_tmp = ∃V(S∧CP) = 9 <= py <= 11
  - CP_tmp = CP なので (true, 9<=py<=11)を返す.
- ２回目のPP
  - CP = 10 <= py <= 11
  - S = {y = 15, y' = 2(-50+5py)^(1/2),
    y'' = -10, y' = -0.8y'-}
  - V = {y, y', y''}
  - CP_tmp = ∃V(S∧CP) = (py = 10)
  - (true, py=10)か(false, 10<py<=11)を返す.

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

Input: 制約ストア S，記号定数の条件 CP
Output: 充足可能性，記号定数の条件

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** $GetElement(\{(true, CP_{tmp}),$
9: $(false, CP \wedge \neg CP_{tmp})\})$
10: **end if**

sageMathでの実装

- CheckConsistencyPP
  - p5の例題を動かす
  - ２回目のPP
    - CP = 10 <= py <= 11
    - S = {y = 15, y' = 2(-50+5py)^(1/2),
      y'' = -10, y' = -0.8y'-}

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

```
== Check Consistency PP by QE ==========================================
==== input ======
constr store    :       [[dy == 2*sqrt(5*p_y - 50), y == 15, ddy == -10, dy == -4/5*dy]]
symbolic params :       [[p_y >= 10, p_y <= 11]]
```

  - この入力だとQEを行うところで止まる.
    - y' = 2(-50+5py)^(1/2) を py について解いて式変形することで回避可能

sageMathでの実装

- CheckConsistencyPP
  - p5の例題を動かす
  - ２回目のPP
    - CP = 10 <= py <= 11
    - S = {y = 15, y' = 2(-50+5py)^(1/2),
      y'' = -10, y' = -0.8y'-}

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

```
== Check Consistency PP by QE ==========================================
==== input ======
constr store    :       [[p_y == 1/20*dy^2 + 10, y == 15, ddy == -10, dy == -4/5*dy]]
symbolic params :       [[p_y >= 10, p_y <= 11]]
# tmp_parameters is modified
==== output =====
consistency     :       [True, False]
updated params  :       [[py - 10 = 0], [py - 10 > 0 /\ py - 11 <= 0]]
==== measure ====
time            :       1.7195779999999985
```

sageMathでの実装

- 例
  - 最初のPP
    - CP = 9 <= py <= 11 (py: yの記号定数)
    - S = {y = py, y' = 10, y'' = -10}
    - V = {y, y', y''}
    - CP_tmp = ∃V(S∧CP) = 9 <= py <= 11
    - CP_tmp = CP なので (true, 9<=py<=11)を返す.
  - ２回目のPP
    - CP = 10 <= py <= 11
    - S = {y = 15, y' = 2(-50+5py)^(1/2), y'' = -10, y' = -0.8y'-}
    - V = {y, y', y''}
    - CP_tmp = ∃V(S∧CP) = (py = 10)
    - (true, py=10)か(false, 10<py<=11)を返す.

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

Input: 制約ストア S, 記号定数の条件 CP
Output: 充足可能性, 記号定数の条件

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** $GetElement(\{(true, CP_{tmp}),$
9: $(false, CP \wedge \neg CP_{tmp})\})$
10: **end if**

sageMathでの実装

- CheckConsistencyIP
  - 同様の例題を動かす
  - 1回目のIP
    - CP = 10 <= py <= 11
    - S = {y = py, y' = 10, y'' = -10}

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

- 入力
  - 微分方程式の集合 [-10 = diff(dy,1), dy= diff(y ,1)]
    - diff(y,1): yの1階微分　diff(dy,1): dyの1階微分
    - sageMathでは2階微分を含む微分方程式を解くことができないため, 中間変数を用意する.
  - 関数の初期値の集合 [py, 10]
  - 解きたい関数の集合 [y, dy]
- 出力として時刻の式の集合 [y(t) == -5t^2 + 10t + p_y, dy(t) = -10*t + 10]

sageMathでの実装

- 例
  - 最初のIP
    - S = {y = py, y' = 10, y'' = -10}
    - S内の微分方程式を解くと
      S_t = {y = py + 10t – 5t^2, y' = 10 - 10t, y'' = -10}
    - CP = 9 <= py <= 11
    - CP_tmp = 9 <= py <= 11
    - CP_tmp = CP なので
      (true, 9 <= py <= 11)

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

Input: 制約ストア S, 記号定数の条件 CP
Output: 充足可能性, 記号定数の条件

1: $S_t := SolveDifferentialEquation(S)$
2: $V := GetVariables(S_t)$
3: $CP_{tmp} := \exists V(Inf\{t \mid \exists_t(S_t \wedge t > 0)\} = 0) \wedge CP)$
4: **if** $CP_{tmp} = false$ **then**
5: **return** $(false, CP)$
6: **else if** $CP_{tmp} = CP$ **then**
7: **return** $(true, CP)$
8: **else**
9: **return** $GetElement(\{(true, CP_{tmp}),$
10: $(false, CP \wedge \neg CP_{tmp})\})$
11: **end if**

sageMathでの実装

- CheckConsistencyIP

```
INIT   <=> 9 <= y <= 11 & y' = 10.
FALL   <=> [](y'' = -10).
BOUNCE <=> [](y- = 15 => y' = -0.8y'-).

INIT, FALL << BOUNCE.
```

  - 入力
    - 時刻の式の集合にt=0を代入した集合 [y == p_y, dy = 10, ddy = -10]
      - sageMath で Inf を表現するために t = 0 を代入
      - mathematicaによる実装でも同様の処理を行なっている

```
== Check Consistency IP by QE ================================================
==== input ======
constr store    :       [[y == p_y, dy == 10, ddy == -10]]
symbolic params :       [[p_y >= 9, p_y <= 11]]
[ddy, dy, y] [y = p_y /\ dy = 10 /\ ddy = -10 /\ p_y >= 9 /\ p_y <= 11]
# tmp_parameters is modified
==== output =====
consistency     :       [True, False]
updated params  :       [[py - 9 >= 0 /\ py - 11 <= 0], []]
==== measure ====
time            :       4.315759999999997
```

研究概要・背景

プログラムについて

実行例

まとめ・今後の課題

まとめ・今後の課題

- まとめ
  - HyLaGIの主要関数のアルゴリズムの説明を行った.
  - HyLaGIの主要関数CheckConsistencyIP,PPをsageMathで実装した.

- 今後の課題
  - FindMinTime, CompareMinTimeをsageMathで実装する.

参考文献

1. 渋谷俊, 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 の実行アルゴリズム, コンピュータソフトウェア, 2011
2. 高田賢士郎, 渋谷俊, 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 の数式処理実行系, 情報処理学会 第73回全国大会 1B-5, 2011
3. 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, 2020
4. 山田 悠之介: 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, 2020
5. 松本翔太: ハイブリッド制約言語HydLaの記号実行シミュレータHyroseの実装と最適化, 早稲田大学大学院基幹理工学研究科, 修士論文, 2013.
6. 佐々木優友: ハイブリッド制約言語HydLaのREDUCEを用いた記号実行系, 早稲田大学大学院基幹理工学研究科, 修士論文, 2014.
7. SageMath - Open-Source Mathematical Software System, https://www.sagemath.org