日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

ハイブリッドシステムモデリング言語 HydLa 処理系における実行アルゴリズム

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，状態や方程式が離散変化したりする系を指す．HydLa は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であり，精度保証されたシミュレーションを行うことでシステム検証に役立てることを目標としている．HydLa プログラムにおいて同時刻に複数の条件が成り立つ場合や連鎖的に複数の条件が成り立つ場合のシミュレーション方法が明らかでなかった．本稿では HydLa プログラムにおける離散変化と連続変化を正しく実行するアルゴリズムを述べる．

1　はじめに

ハイブリッドシステム [1] とは時間の経過に伴って状態が連続変化したり，状態や方程式系自体が離散変化したりするシステムを指す．ハイブリッドシステムのモデリングツールは物理学や制御工学，生命工学などの分野で重要な役割を果たす．

HydLa [2] は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である．HydLa には次のような特徴がある．

- 制約を宣言することでプログラムを記述する．
- 状態や条件を論理式と数式を組み合わせて表現する．
- 制約階層を用いることで，適用する制約間の優先順位を表現する．

HydLa で記述したプログラムのシミュレーションを行う HydLa 処理系が現在開発中である．

ハイブリッドシステムのモデリングツールにはハイブリッドオートマトンを用いてハイブリッドシステムを表現するものが多い [3]．制約を用いてハイブリッドシステムを扱う言語は少なく，Hybrid cc [4] [5] が数少ない例外である．簡潔な記述ができるが計算の精度が保証されていないため厳密なシミュレーションを行うことができない．

```
INIT   <=> ht=10 /\ v=0.
FALL   <=> [](ht' = v /\ v' = -10).
BOUNCE <=> [](ht- = 0 => v = -(4/5)*v-).
INIT, (FALL << BOUNCE).
```

図 1　床で弾むボールの挙動

2　HydLa によるモデリング例

床で弾むボールの挙動について HydLa を使ってモデリングしたプログラムを図 1 に示す．

このプログラムでは 1 行目から 3 行目で制約を定義し，4 行目で使用する制約を呼び出している．<=>の左辺は制約定義名を，右辺は論理記号や方程式によってその内容を表す．変数 ht はボールの高さ，v はボールの落下速度を表す．

制約 INIT は ht と v の時刻 0 での値を記述している．HydLa ではすべての変数は時刻の関数であり，後述する時相演算子 [](always) が付いていないものは時刻 0 での性質を表し，付いているものは時刻 0

An Execution Algorithm for the Hybrid System Modeling Language HydLa.
Shun Shibuya, Kenshiro Takata, 早稲田大学大学院基幹理工学研究科情報理工学専攻, Waseda University.
Hiroshi Hosobe, 国立情報学研究所, National Institute of Informatics.
Kazunori Ueda, 早稲田大学理工学術院情報理工学科, Waseda University.

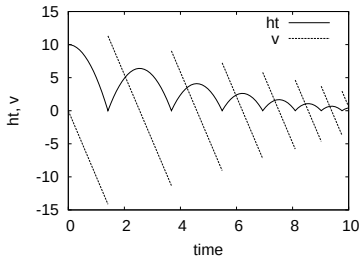

図 2　床で弾むボールの実行

以降の性質を表す.

制約 FALL は落下運動を示す．右辺全体が []( ) で囲まれている．これは always を表し，括弧内が時刻 0 以降の ht' と v' の性質を表す．ht' と v' はそれぞれ ht と v の時刻に対する一次微分である.

制約 BOUNCE は跳ね返りを表す．この制約は=>があるため条件付き制約である．条件付き制約は=>の左辺のガード条件が満たされた場合に，右辺の式が成立することを表す．ガード条件 ht-=0 の ht に付いている - はガード条件の判定時刻における ht の左極限の値をとることを表す.

4 行目で制約の優先順位に従って半順序集合を構成しており，半順序集合の各要素を制約モジュールと呼ぶ．=>で関係付けられた制約は，右辺が左辺より優先されて適用される.「,」で区切られた制約モジュール間には階層関係はない．この書き方により各時刻で FALL より BOUNCE が優先して適用される.

このプログラムの実行結果を図 2 に示す． 落下開始から ht=0 となるまでが連続変化，ht=0 で離散変化となる．その後再び跳ね返るまで連続変化，跳ね返りで離散変化となる．以降この連続変化と離散変化を繰り返すことがわかる.

3 HydLa 処理系のシミュレーション実行アルゴリズム

HydLa 処理系は HydLa プログラムを宣言的意味論に従って正しく実行することを目的としたものであり，開発改良が進められている [8] [9]．数式処理を主として用いて，数式処理では解けない計算について区間計算を組み合わせることで，精度保証されたシミュレーションを行うことを目標としている．精度保証されたシミュレーションはシステム検証においても重要な要素技術となる.

論文 [9] のアルゴリズムでは数式処理によるシミュレーションに目的を限定し，条件付きでない制約に不等式や不定な値を記述することを許さないものとしていた．本論文では条件付きでない制約にも不等式や

Input: $HydLaProgram$, シミュレーション終了時刻 $MaxT$
  $MS := TopologicalSort(SolveCH(HydLaProgram))$
  $T{:=}\,0$; $S := \emptyset$
  while $T <$ $MaxT$ **do**
    for $M \in MS$ **do**
      $(SS,MS_{tmp}) := PP(S, M, MS, T)$
      while $|SS| > 1$ **do**
        $S{:=}GetElement(SS)$
        $(SS,MS_{tmp}) := PP(S, M, MS, T)$
      end while
      if $|SS| = 1$ **then**
        $S := GetElement(SS)$; $MS := MS_{tmp}$
        goto IP
      end if
    end for
    break
    IP:
    for $M \in MS$ **do**
      $(SS, T_{tmp}, MS_{tmp}) := IP(S, M, MS, T, MaxT)$
      while $|SS| > 1$ **do**
        $S{:=}GetElement(SS)$
        $(SS, T_{tmp}, MS_{tmp}) := IP(S, M, MS, T, MaxT)$
      end while
      if $|SS| = 1$ **then**
        $S := GetElement(SS)$; $MS := MS_{tmp}$
        $T{:=}\,T_{tmp}$
        goto PP
      end if
    end for
    break
    PP:
  end while

図 3　HydLa 処理系のシミュレーションの非決定実行アルゴリズム

Input: 制約ストア S，制約モジュール集合 M，解候補モジュール集合のリスト MS， 現在のシミュレーション時刻 T
Output: 制約ストアの集合 SS，新しい解候補モジュール集合のリスト MS
  if $T > 0$ **then**
    $M := EliminateNotAlways(M)$
  end if
  $(SS, \_, \_, MS) := CalculateClosure(S, M, MS,$
    $CheckConsistencyPP, CheckEntailmentPP)$
  return (SS, MS)

図 4　PP のアルゴリズム

Input: 制約ストア S，現在の制約モジュール集合 M，解候補モジュール集合のリスト MS，現在のシミュレーション時刻 T，最大シミュレーション時刻 $MaxT$
Output: 制約ストアの集合 SS，IP 終了時刻 $EndT$，新しい解候補モジュール集合のリスト MS

$M := EliminateNotAlways(M)$
$M_{all} := MaxModule(MS)$
$(SS, A_-, A_+, MS) := CalculateClosure(S, M, MS, CheckConsistencyIP, CheckEntailmentIP)$
if $|SS| \neq 1$ **then**
  return $(SS, MaxT, MS)$
end if
$S_t := SolveDifferentialEquation(GetElement(SS))$
$(MinT, SS) := CompareMinTime($
  $\{FindMinTime(S_t \Rightarrow g) | (g \Rightarrow c) \in A_-\}$
  $\cup \{FindMinTime(S_t \Rightarrow \neg g) | (g \Rightarrow c) \in A_+\}$
  $\cup \{FindMinTime(S_t \wedge M_-) | M_- \in (M_{all} \setminus M)\}$
  $\cup \{FindMinTime(S_t \wedge \neg M_+) | M_+ \in M\}$
  $\cup \{(MaxT - T, true)\})$
if $|SS| > 1$ **then**
  return $(SS, MaxT, MS)$
end if
$S := SubstituteMinTime(S_t, MinT)$
return $(\{S\}, MinT + T, MS)$

図 5　IP のアルゴリズム

不定な値を記述できるように対応したアルゴリズムとなっている．

3.1　HydLa 処理系の非決定実行アルゴリズム

HydLa 処理系の非決定実行アルゴリズムを図 3 に示す．まず *SolveCH* によりプログラムから制約階層の処理を行い [7]，解候補となる制約モジュール集合のリスト MS を求める．MS 内の要素は集合の包含関係に従ってトポロジカルソートされて並んでいる．

HydLa 処理系はシミュレーション時刻が終了するまで，離散変化を扱うポイントフェーズ (PP) と連続変化を扱うインターバルフェーズ (IP) を交互にシミュレーション実行する．それぞれのフェーズでは無矛盾かつ極大な制約モジュール集合に基づいてシミュレーションを行う [2]．PP と IP のアルゴリズムを図 4，5 に示す．アルゴリズム内に記述しないが，各フェーズでの計算結果を出力として表示するものとする．以降では時刻について，シミュレーション開始時刻からの経過を T，ある IP において直前の PP の時刻からの経過を t を用いて表す．

S はそのフェーズで成立する制約を集めたもので制約ストアと呼ぶ．制約ストアには前フェーズ終了時の変数の値に関する制約が入り，PP では左極限値に関する制約，IP ではフェーズ開始時点で成立する制約が含まれる．S にはさらに各フェーズにおいて成立する制約が入る．

Input: 制約ストア S，現在の制約モジュール集合 M，解候補モジュール集合のリスト MS，無矛盾性判定関数 $CheckConsistency(S)$，ガード条件判定関数 $CheckEntailment(S, g)$
Output: 制約ストアの集合 SS，成立しない条件付き制約の集合 A_-，成立する条件付き制約の集合 A_+，新しい解候補モジュール集合のリスト MS

$A_- := \emptyset$;　$A_+ := \emptyset$
repeat
  $S := CheckConsistency(CollectTell(M, S))$
  if $S = \bot$ **then**
    return $(\emptyset, \emptyset, \emptyset, MS)$
  end if
  $A_- := A_- \cup CollectAsk(M)$
  $Expanded := false$
  for $(g \Rightarrow c) \in A_-$ **do**
    $(S_{true}, S_{false}) := CheckEntailment(S, g)$
    if $(S_{true} \neq \emptyset) \wedge (S_{false} \neq \emptyset)$ **then**
      return $(\{S_{true}, S_{false}\}, A_-, A_+, MS)$
    end if
    if $S_{true} \neq \emptyset$ **then**
      $M := DeleteGuard(M, (g \Rightarrow c))$
      if $CheckAlways(c)$ **then**
        $MS := \{DeleteGuard(M_i, (g \Rightarrow c)) | M_i \in MS\}$
      end if
      $A_- := A_- \setminus \{g \Rightarrow c\}$
      $A_+ := A_+ \cup \{g \Rightarrow c\}$
      $Expanded := true$
    end if
  end for
until $\neg Expanded$
return $(\{S\}, A_-, A_+, MS)$

図 6　CalculateClosure のアルゴリズム

制約モジュール集合 M には，always が付いていないシミュレーション開始時刻 ($T = 0$) でのみ成立する制約が含まれる．この制約は $T = 0$ より後のフェーズを実行する際には不要であるので *EliminateNotAlways* で取り除く．M_{all} は *HydLaProgram* から制約階層を除いたもので MS の先頭要素である．

PP や IP をシミュレーションした結果，制約ストアの集合 SS が得られ，SS の要素数によりフェーズでの処理が異なる．

SS が空集合の場合，採用した制約モジュール集合の制約間に矛盾があったことを意味する．現在のフェーズで実行中の制約モジュール集合 M は失敗とし，M については何もせず，MS のリストから次に優先度の高い制約モジュール集合を用いてフェーズの

```
CheckConsistencyPP(S){
  return ∃(S)
}

CheckConsistencyIP(S){
  S_t := SolveDifferentialEquation(S)
  return (t = 0 の正の近傍で ∃(S_t) を満たす )
}

CheckEntailmentPP(S, g){
  return ((S ⇒ g) となる S, (S ⇒ ¬g) となる S)
}

CheckEntailmentIP(S, g){
  S_t := SolveDifferentialEquation(S)
  return ((Inf{t | (S_t ⇒ g) ∧ (t > 0)} = 0) となる S,
(Inf{t | (S_t ⇒ g) ∧ (t > 0)} ≠ 0) となる S)
}
```

図 7　CheckConsistency, CheckEntailment

実行をやり直す.

SS の要素が複数の場合，条件付きでない制約に不等式や不定な値の記述があり，不等式や不定な値の範囲によって解が分岐したことを意味する．*SS* の要素である制約ストアには分岐先で満たすべき制約がそれぞれ入っている．*GetElement* により任意の制約ストアを非決定的に 1 つ選び，このフェーズをやり直すことで分岐先を実行する.

SS の要素が 1 つの場合，その制約ストアを次のフェーズに引き継ぎ，シミュレーション実行を進める.

3.2　CalculateClosure

PP や IP の *CalculateClosure*(図 6) では，条件付き制約が成立する場合に右辺の制約を追加する閉包計算を繰り返し，そのフェーズ開始時点で成立しない条件付き制約の集合 A_- と成立する条件付き制約の集合 A_+ を求める．このとき，*CollectAsk* は条件付き制約がどの制約モジュールに含まれていたものか区別する．これにより同じ形の条件付き制約が A_- の要素に含まれたとしても，後で条件付き制約を取り出す際に互いを区別できるようになる.

S に不等式や不定な値が含まれると，条件付き制約が成立する場合と成立しない場合が同時に存在することがあり，このとき解が分岐する．解を正しく求めるために *CalculateClosure* は条件付き制約が成立する範囲と成立しない範囲に絞りこんだ制約ストアをそれぞれ返す．成立する条件付き制約のガード条件部分は *DeleteGuard* で取り除く．成立する条件付き制約の $\Rightarrow$ の右辺に always が含まれるとき，*MS* の要素である各モジュール集合からも，制約モジュールに含まれているガード条件部分を取り除く.

3.3　CheckConsistency, CheckEntailment

CalculateClosure では制約ストア S 内の無矛盾性判定を PP 用の *CheckConsistencyPP*，IP 用の *CheckConsistencyIP*(図 7) により判定する．$\exists(S)$ は S の存在閉包を表す．*CheckConsistencyIP* では制約ストア S から各変数の，t に関する式 S_t を *SolveDifferentialEquation* で求め，さらに IP 開始直後に S_t が成立するかについて判定している．矛盾の場合，制約が過剰に含まれるため $\bot$ を返す.

条件付き制約のガード条件の判定は PP と IP で異なり，それぞれ *CheckEntailmentPP*，*CheckEntailmentIP*(図 7) で判定し，条件付き制約が成立する範囲と成立しない範囲に絞りこんだ制約ストアをそれぞれ返す．PP での判定は制約ストア S がガード条件 g を含意するか判定する ($\forall$ は全称閉包)．IP での判定は $t = 0$ の直後で制約ストア S がガード条件 g を含意するか判定する．*Inf* は最大下界 (greatest lower bound) となる値を求める．判定の結果，該当する S が無い場合には $\emptyset$ を返す.

3.4　FindMinTime と CompareMinTime

IP では次に離散変化となる時刻を *FindMinTime* と *CompareMinTime* で求める．*SS* の要素である制約ストア S から *SolveDifferentialEquation* で求める t に関する式 S_t を求める．S_t, A_-, A_+，IP 開始時点で採用していない制約モジュール，IP 開始時点で採用している制約モジュールから，次に離散変化となる時刻を，直前の PP の時刻を $t = 0$ として *FindMinTime* で求める．その際，引数として渡す論理式が満たされない場合は離散変化となる時刻として ∞ を返す.

制約に不等式や不定な値が含まれていると，次に離散変化となる時刻が一意に定まらず、解が分岐する場合がある．解が分岐する時，分岐ごとに離散変化となる時刻は異なる．*FindMinTime* では離散変化となる時刻だけでなく，離散変化となる制約の範囲も返す.

```
A <=> x=0.
B <=> [](y=1).
C <=> [](x'=1 /\ (x>3 => y=2)).
A, (B << C).
```

図 8　ある時刻の直後からガード条件が成立する例

```
INIT  <=> x=0 /\ y=0.
CON   <=> [](y'=0).
CROSS <=> [](x=5 /\ y=2).
INC   <=> [](x'=1).
INIT, (CON << (CROSS << INC)).
```

図 9　極大が変化することによる離散変化の例

CompareMinTime では *FindMinTime* の結果から，次に離散変化となる時刻を場合分けし，分岐先で満たすべき S をそれぞれ用意し，S の集合である SS を返す．

4　例題

本論文で提案したアルゴリズムの実行が HydLa の宣言的意味論 [2] に従うことを例題によって示す．3 節と同様に時刻について，シミュレーション開始時刻からの経過を T，ある IP において直前の PP の時刻からの経過を t を用いて表す．

図 8 について考える．制約 C のガード条件が成立し，y=2 となるのは T=3 の直後からである．今回のアルゴリズムでは T=3 の PP で x=3 となり，次の IP でガード条件 x>3 が成立するか判定する．もしこのとき PP の判定のように前回のフェーズ終了時の値 x=3 とガード条件 x>3 を連立すると *false* となる．そこで IP 開始時点で制約ストアから求めた，t に関する x の式 x=t+3 とガード条件 x>3 を連立したものが t=0 の直後で成り立つことを検査している．

図 9 について考える．HydLa の宣言的意味論では制約モジュール集合の中から無矛盾かつ極大なものを時々刻々と満たすことを求めている [2]．初めの IP では CROSS と CON，INC は矛盾するため，制約階層の優先順位に従って CON と INC が採用される [7]．このことから x'=1 により x は増加し，やがて T=5 で x=5 となると，CROSS が採用できるため，y=2 となる．つまり極大な制約モジュール集合が変化することから，離散変化が起こる．今回のアルゴリズムでは *FindMinTime* により現在採用されていない制約が採用される時刻を求めることができ，PP では CROSS が採用され，y=2 となる．

```
INIT   <=> 9<ht /\ ht<11 /\ v=10.
FALL   <=> [](ht'=v /\ v'=-10).
ROOF   <=> [](ht-=15 => v=-(4/5)*v-).
BOUNCE <=> [](ht-=0  => v=-(4/5)*v-).
INIT, (FALL << (ROOF, BOUNCE)).
```

図 10　不等式を含むことで分岐する例

```
INIT(q,initq,v,initv)
          <=> q=initq /\ v=initv.
CONT(q,v) <=> [](q'=v /\ v'=0).
COLLISION(q1,v1,m1,q2,v2,m2,e)<=>
[](q1-=q2- =>
   v1=-((e*m2*v1-)-(e*m2*v2-)
         -(m1*v1-)-(m2*v2-))
      /(m1+m2) /\
   v2=-((-e*m1*v1-)+(e*m1*v2-)
         -(m1*v1-)-(m2*v2-))
      /(m1+m2)
).
EPS <=> eps>0.

EPS,INIT(qa,-4,va,5),INIT(qb,1,vb,0),
                INIT(qc,1+eps,vc,0).
(CONT(qa,va),CONT(qb,vb),CONT(qc,vc))
<<(COLLISION(qa,va,1,qb,vb,1,1) /\
   COLLISION(qb,vb,1,qc,vc,1,1)).
```

図 11　隣接したボールの衝突の例

初期値に幅を持つと，実行結果が分岐する場合があり，ここでは図 10 を基に考える．図 1 と同じく ht が高さ，v が落下速度であるが，v=10 により鉛直方向上向きに投げ上げ，ROOF により床だけでなく ht=15 にある天井で衝突した時も跳ね返るモデルとなっている．この INIT は不等式を含むために，天井に衝突する場合と衝突せずに落下する場合の両方が考えられる．*FindMinTime* では ROOF のガード条件より，10<=ht,ht<11(天井に衝突する) と 9<ht,ht<10(天井に衝突しない) で分ける．*CompareMinTime* では他のガード条件や制約，シミュレーション終了時刻と比較し，分岐先ごとにそれぞれ次の離散変化となる時刻を求める．

図 11 について考える．ニュートンのゆりかごとして知られる複数の隣接した球がきわめて短い時間の間

に連続して衝突を繰り返すモデルの一部である．球は大きさを持たない質点とする．隣接して配置するために内側の球は動かずに最も外側の球だけが飛び出すように見える．物体の衝突後の速度は運動量保存の法則と反発係数の定義を連立したものから求めている．球 A,B,C の座標を qa,qb,qc，速度を va,vb,vc とし，球 A を静止した球 B,C に衝突させる．INIT は位置と速度の初期値，CONT は速度と加速度，COLLISION は衝突した場合の速度変化を表す．このモデルをシミュレーションするには静止した球を隣接して配置することが必要であるが，隣接してしまうと球 B と C の座標が重なる．すると T=1 の PP で球 A と B，B と C の COLLISION の条件付き制約がどちらも成り立ち，vb に代入するため矛盾し，シミュレーションを行うことができない．よって少しだけ間隔を空けることになる．

この間隔を具体的な数値で決めるのではなく，不定な値 eps で決めるとする．これによりシミュレーション実行過程の式に eps が含まれ，eps が 0 より大きい場合の分岐をすると実行時間や途中式に eps を含んだままシミュレーションを行うことができる．T=1 で球 A と B が衝突し，次に T=1+2eps で球 B と C が衝突する．T>1+2eps の IP で座標の t に関する式は qa=1,qb=1+eps,qc=5t となり球 A,B は動かないが，球 C は動き続けることがわかる．eps を 0 に近づける極限を考えると，球 A,B が隣接し動かず，端の球 C だけが飛び出すことになる．

5 まとめと今後の課題

HydLa の実行アルゴリズムを定式化し処理の難しい例題を基に検討を行った．今後も処理系の機能を拡充するとともに区間計算と組み合わせることやシステム検証を視野に入れた処理系を目指す．

謝辞 本研究を行うにあたり貴重な意見を頂いた上田研究室 HydLa 班の皆様に感謝します．

参 考 文 献

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

[2] 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予稿集, pp. 1-6, 2008.

[3] L. Carloni, R. Passerone, A. Pinto, A. L. Sangiovanni-Vincentelli : Languages and Tools for Hybrid Systems Design, Foundations and Trends in Design Automation 1(1), pp. 1-204, 2006.

[4] B. Carlson, V. Gupta : Hybrid cc with Interval Constraints, in Proc. HSCC'98, LNCS 1386, Springer, pp. 80-94, 1998.

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

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

[7] 廣瀬賢一, 大谷順司, 石井大輔, 細部博史, 上田和紀 : 制約階層によるハイブリッドシステムのモデリング手法, 日本ソフトウェア科学会第 26 回大会論文集, 2D-2, 2009.

[8] 廣瀬賢一 : 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の提案と実装, 早稲田大学大学院基幹理工学研究科, 修士論文, 2010.

[9] 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 処理系の実行アルゴリズムの検討, 第 8 回ディペンダブルシステムワークショップ, 2010.