The 32nd Annual Conference of the Japanese Society for Artificial Intelligence, 2018

ハイブリッドシステムモデリング言語 HydLa における変数と制約階層の動的生成記法の設計と実装

Design and implementation of dynamic variables and constraint hierarchies for HydLa

佐藤柾史 *1　上田和紀 *2
Masashi SATO　Kazunori UEDA

*1 早稲田大学基幹理工学部情報理工学科
School of Computer Science and Engineering, Waseda University

*2 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
Graduate School of Fundamental Science and Engineering, Waseda University

Hybrid systems are dynamical systems that consist of discrete changes and continuous changes. HydLa is a declarative language for modeling hybrid systems by means of constraints. It adopts the concept of hierarchical constraints to describe hybrid systems concisely. HyLaGI is an implementation that simulates HydLa programs by simbolic execution. Variables in HydLa programs that could be simulated by HyLaGI are all global and static. Because of that, some models are difficult to write. To solve this problem, I introduced existential quantifier to create local and dynamic variables and implemented it in HyLaGI. A further problem that arises with existential quantifiers is that we cannot specify priorities among constraints within the scope of existential quantifiers. To solve that, I introduced a notation to specify priorities among generated constraints and implemented it in HyLaGI.

1. はじめに

ハイブリッドシステム [1] とは, 時間進行に依る振る舞いを表す連続変化と, 離散変化を交互に繰り返す系である.

ハイブリッドシステムのモデリング言語として HydLa[3] がある. HydLa は制約プログラミングのパラダイムで設計された宣言型言語であり, モデルの時間変化である微分方程式や離散変化を制約 (時相論理式) を用いて記述する. 制約はモジュールとして定義され, 合成や再利用ができる. 制約階層の概念を取り入れており, モジュール同士に優先度関係を設けることで場合分けを最小限に抑えた記述が可能である.

HydLa で用いられる変数は全て大域的かつ静的であり, 変数の隠蔽や軌道の動的生成ができない. また, 電気回路等のモジュールの再利用性が重要なモデルでは変数の衝突が問題となり, イベントの発生でオブジェクトを動的に生成することが表現できない. もう一つの問題として, 生成した制約同士に優先度関係を設けることが出来ない点がある. 本研究では, 上記の問題を解決するための記法の導入を目的とし, 例題の考察と処理系 HyLaGI への実装を行った.

2. 存在量化子を用いた記法

HydLa の制約は論理式であり, 論理記号は自然に採用することができる. 中でも, 存在量化子記号を採用することによって変数の動的生成を HydLa の意味論を変更することなく実現できる. 本章では, 存在量化子を用いた記法の詳細な意味とそれにより上記の問題が解決することを例題を用いて説明する.

2.1 内部変数の隠蔽

制約のモジュール化が必要なハイブリッドシステムの例として, スイッチ付き RC 回路を 2 つ接続した電気回路を挙げる. 電源・スイッチ・抵抗・コンデンサに対応する制約をモジュールとし, それらを並列合成して組み立てることでモデリングする. コンデンサをモジュール化してモデリングする場合, 媒介変数として内部変数が出現する. その際, コンデンサを複数回利用した場合はその変数の意図しない共有が発生し制約同士が矛盾する. この問題は, その変数を存在量化子を用いて隠蔽することで防ぐことができる. 例えば, コンデンサモジュールは以下のように記述できる.

$$C(vin, vout, i, c) \Leftrightarrow \exists q.(q = 0 \wedge \Box(q = c * (vin - vout) \wedge i = q'))$$

この記法により, コンデンサモジュール C が使用されると量化子の除去が行われ, 他と衝突しない変数が生成される. この変数は明示的に示したスコープの外側から補足できないため, 意図しない衝突が発生することが無い.

この回路をモデリングしたプログラムを図 1 に, 出力の一部を図 2 示す. HydLa のプログラムの詳細については, 文献 [5] を参照して欲しい.

```
R(vin,vout,i,r) <=> [](vin-vout = r*i).
C(vin,vout,i,c) <=> \q.(q=0 & [](q=c*(vin-vout) & i=q')).
E(vin,vout,e0) <=> [](vout-vin=e0).
JUNCTION(i1,i2,i3) <=> [](i1+i2+i3=0).
TIMER(timer) <=> timer=0 & []timer'=1.
SWITCH(vin,vout,i,on)<=>[](on=0 => i=0) & [](on=1 => vin=vout).

TIMERSWITCH(vin,vout,i,time,a) <=> \on.\timer.(
  TIMER(timer) & [](timer<time => on=1-a) &
  [](timer>=time => on=a) & SWITCH(vin,vout,i,on)).
RCL(vin,vout,i) <=> \ve.\vs.\ir.\ic.(
  []vout=0 & E(vs,ve,10) & TIMERSWITCH(vout,vs,ir,10,0)
  & R(ve,vin,ir,3000) & JUNCTION(ir,-ic,-i)
  & C(vin,vout,ic,10^(-3))
).
RCR(vin,vout,i) <=> \vm.\vs.(
  R(vin,vm,i,2000) & C(vm,vs,i,10^(-3))
  & TIMERSWITCH(vs,vout,i,5,1)
).

RCL(vin,vout,i), RCR(vin,vout,i).
```

図 1: スイッチ付き RC 回路を 2 つ接続した電気回路 プログラム

前述のコンデンサの他に, 並列合成に用いたモジュール同士

連絡先: 佐藤柾史 早稲田大学基幹理工学部情報理工学科, 〒 169-8555 新宿区大久保 3-4-1 63 号館 5 階 02 号, 03-5286-3340, masashi(at)ueda.info.waseda.ac.jp

の入出力の隠蔽をするために存在量化子を用いている. 例えば, モジュール TIMERSWITCH ではスイッチとステップ関数を構成する制約を並列合成することで, 時間経過でスイッチを開閉する役割を果たす. 入出力の変数 on, timer を隠蔽しているため, モジュールとして使用しても変数の衝突が起こらない. このプログラムで用いた全ての存在量化子は, 時刻 0 でリネーム・除去される.

```
---------PP 1---------
unadopted modules: {}
positive  : $timer1<10=>$on1=1-0
            $on1=1=>vout=$vs1
            $timer2<5=>$on2=1-1
            $on2=0=>i=0
negative  :
t : 0
$ic1 : 1/300
$ir1 : 1/300
$on1 : 1
$on2 : 0
$q1 : 0
$q2 : 0
$timer1 : 0
$timer2 : 0
```

図 2: スイッチ付き RC 回路を 2 つ接続した電気回路 出力 (一部)

2.2 条件付き制約の後件で用いられた存在量化子の扱い

HydLa における離散変化は, 論理記号の含意記号 (⇒) を用いた制約 (条件付き制約) を用いて表現される. 前件の成立によって後件の制約が発行され, 制約ストアの変化が発生する. 条件付き制約の後件で用いられた存在量化子は前件が成立する度に除去が行われ, 新しい変数に関する制約が生成される. そのため, 離散変化の度に新しい変数を生成することを表現することができる. しかし, 論理式の持つ意味を処理系に自然に導入できない例が存在する. 具体的には, 後件に存在量化子を持つ条件付き制約の前件が連続変化フェーズで成立した場合である. 図 3 にその例を示す. このプログラムでは, 条件付き制約 B の前件が 1-2 秒の間で成立する. 論理式の意味をそのまま採用すると, この間に無限個の変数と制約が生成されることになる. 処理系では, この様な例を検知した場合は実行時にエラーを出力する.

```
A <=> f=0 & []f'=1.
B <=> [](1<=f<=2 => \x.([]x=1)).

A, B.
```

図 3: 連続変化フェーズで成立する条件付き制約

2.3 制約生成の遅延

図 4 は, サーモスタットによる温度制御をモデリングしたプログラムである. 温度が閾値 (62/68 度) を越えようとする度に微分制約を変更することで温度制御を実現する. ここで, 温度の制御に 0.5-1 秒の遅延を伴うものとする. そのような制御の遅延を表現するためには条件付き制約を用いた制約の切り替えを遅延させるのが自然であり, そのためには制御条件を満たす度にタイマーの役割を果たす変数を生成することで実現できる. 項 2.2 で述べたように, 条件付き制約の後件に存在量化子を用いることで前件の成立と共に新しい変数を生成することができる. 図 4 のプログラムでは, モジュール SWITCHON/SWITCHOFF の前件が成立する度に制約 CHANGEMODE が生成され, その制約に記述された存在量化子はその時刻で除去される. x は範囲を持って初期化され, 0.5-1 秒後に微分制約を変更するための条件付き制約を充足する. これにより, 制御の遅延を表現できる. プログラムの出力を図 5 に示す. 前件が成立した時刻で, id の付加された新しい変数が生成されている.

```
INIT <=> p=65 & []k1=1 & []k2=2 & mode=0.
OFF <=> [](mode=0 => p'=-k2).
ON <=> [](mode=1 => p'=k1).
CHANGEMOD(m) <=>
  \x.(0<=x<=0.5 & []x'=1 & [](x=1 => mode=m)).
SWITCHOFF <=> [](p-=68 & mode=1 => CHANGEMOD(0)).
SWITCHON <=> [](p-=62 & mode=0 => CHANGEMOD(1)).

INIT, []mode'=0 << (SWITCHON, SWITCHOFF).
OFF, ON.
```

図 4: 遅延の伴うサーモスタット プログラム

```
---------PP 3---------
unadopted modules: {}
positive  :
  p-=62&mode=0=>CHMOD(05/10, 08/10, 1)
negative  :
t : 3/2
k1 : 1
k2 : 2
mode : 0
p : 62
x1$3 : (1/2, 4/5)
mode' : 0
p' : -2
x1$3' : 1
```

図 5: 遅延の伴うサーモスタット 出力 (一部)

2.4 オブジェクトの動的生成

条件付き制約の後件に存在量化子を用いることで, オブジェクトの動的な生成を表現できる. 例として, 床との衝突で 2 つに割れるボールをモデリングしたプログラムを図 6 に挙げる.

```
INIT(x,y,px,py,vx,vy) <=> x=px & y=py & x'=vx
  & y'=vy.
FALL(x,y) <=> [](y''=-10 & x''=0).
GFALL(x,y) <=> [](y>0 => y''=-10 & x''=0).
BOUNCE(x,y) <=> [](y-=0 => x'=x'- & y'=-4/5*y'-).
BREAK <=>
  [](y-=0 =>
    \nx.\ny.(
      INIT(nx,ny,x-,y-,1/2*x'-,-3/4*y'-)
      & GFALL(nx,ny) & BOUNCE(nx,ny))).

INIT(x,y,0,10,1,0), FALL(x,y) << BOUNCE(x,y).
BREAK.
```

図 6: 床との衝突で 2 つに割れるボール プログラム

変数 x,y は元のボールを表しており, 変数 nx,ny はボールが床に衝突した際に生成される. 同時に, 生成された変数に関する自然落下と床との衝突で跳ね返る制約が生成される.

ボールが床に衝突する度に, 新しい変数が生成されて落下と衝突の制約が与えられる. しかし, このプログラムで生成した

自然落下と床との衝突の制約は, ガード条件を用いて場合分けを行う必要がある. HydLa では制約同士に優先度を設けることで, できるだけ場合分けを行わずに採用される制約の選択を行うことが出来る. しかし, 条件付き制約は論理式であり, 後件は原始論理式の連言である. 従って, 条件付き制約で生成された制約同士に強弱関係を設けることができず, 特に存在量化子を用いた変数生成を採用する限り, 生成された変数に関する制約は全て同じモジュールに属する.

3. 生成された制約同士に優先度を設けるための提案記法

本節では, 項 2.4 で述べた問題を解決するための記法の提案を行う. HydLa では論理式を採用することで意味論を簡潔に定義しているが, 関係を持った制約の発行を論理式のレベルで行うことはできない. 提案記法では, HydLa における論理記号の持つ直感的な意味や, ",", "≪"演算子によってモジュール同士の優先度を規定する文 (以降, モジュール宣言文と呼ぶ) と論理積の類似性に着目して, 既存の記法をできるだけ崩さずに定義することを目的とする.

3.1 構文

提案記法の構文として, 文献 [4] の構文の以下の 2 つを修正する.

(module)	M ::=	C | **G ⇒ P**
(program)	P ::=	M | P, P | P ≪ P
		| Pname($\vec{E}$)
		| ∃ **vname. P**

図 7: 提案記法を追加した構文

現状の HydLa の意味論では論理積で接続された原始論理式を制約の集合とみなしており, それらは対応して定義されたモジュールに属している. 一方, ",", "≪"演算子で接続されたモジュール同士は異なるモジュールに属する制約の集合と言える. どちらも制約の集合とみなすことができるため, 記号 ⇒ と ∃ の意味を一貫して定義できると考えて採用した.

追加した 1 つ目の構文は, G ⇒ P である. この構文は, ガード条件の成立によって優先度を持った制約を追加する役割を持つ. HydLa における記号 ⇒ は制約の生成を表すことに注目し, 優先度を持った制約の追加を表す記号として採用した. この構文を, 条件付きモジュールと呼ぶこととする. 追加した 2 つ目の構文はモジュール宣言文に存在記号を用いることを可能にする. こちらも HydLa における記号 ∃ が変数の生成を表すことに注目し, 異なるモジュールに属する制約から参照される変数を生成するための記号として採用した.

なお, 採用した記号は論理記号としての意味を持っていないことに注意して欲しい.

3.2 意味論

本稿では, 項 3.1 において追加した構文の意味を定義する.

まず, モジュール宣言文の形式的な定義を行う. モジュール宣言文は半順序集合であり, モジュールの集合 MS と順序関係 $\leq$ のペアであるとする. 順序関係は, $M_1 \ll M_2$ を表す順序対 $\langle M_1, M_2 \rangle$ を要素として持つ集合 R とする. HydLa が採用されるモジュールの判定に利用するモジュール宣言文 D は, 条件付きモジュールのガード条件の成立によって動的に拡張される. 従って, D を時刻を引数としその時刻のモジュールの集合と順序関係を返す関数とする. つまり, $D = D(t) = \langle MS(t), R(t) \rangle$ である.

これを前提とし, 基本 HydLa の意味論の拡張を行う. 基本 HydLa の意味論については, 文献 [3] を参照して欲しい.

追加した記法により, 解軌道 $\bar{x}$ に応じてモジュール宣言文 $D(t)$ が拡張される. そこで, $\langle \bar{x}, \langle Q, D \rangle \rangle$ がプログラム P を満たすという関係を定める. 図 10 にその関係を定義する.

文献 [3] の基本 HydLa の意味論を追記・修正した点について説明する. (i) が各モジュールに ロ-閉包条件を求めている所は同じであるが, 追加されたモジュール全てに求めている. (iii) は, $DC(P)$ はプログラムのモジュール宣言を取り出す関数であるが, D が時刻 0 においてプログラムで静的に記述されたモジュール宣言文で初期化されることを示している. (vi) は, 各時刻で D に展開されたモジュールとモジュール同士の優先度関係を以降の時刻で保存するための閉包条件を表す. (v) の主な行について, (s0) の MS は $D(t)$ を入力とする関数であり, モジュールとそれらの半順序関係に基づいてその時刻の採用されるモジュールを導出する. (s4) の各行は, 条件付きモジュールが成立することによってモジュール宣言文が拡張されることを表している. (s4b) の MS', R' は, 前件が成立した条件付きモジュールの後件にあるモジュール宣言文の持つモジュール集合, および順序関係とマッチしている. (s4c) は, 後件のモジュールとそれらのモジュール同士の半順序関係が新しく D に追加されること示す. (s4e) は, 追加したモジュールの制約を Q へ登録する. この時に, always 付の制約は (i) の ロ-閉包条件で以降の時刻に展開される. (s4f) は, 親となる条件付きモジュールより強いモジュールは追加されたモジュールよりも強いことを, (s4g) も同様に, 条件付きモジュールより弱いモジュールは追加されたモジュールよりも弱いことを表す.

モジュール宣言文に記述された存在記号はそのモジュール宣言文が $D(t)$ に登録された時刻で, 束縛している変数を新しい名前に変更して除去が行われる. また, 束縛した変数が持つ制約が所属するモジュール名も現在 MS に存在しない名前にリネームすることとする.

次に, ガード付きモジュールは制約 true を参照するモジュールと定義する. これは, 条件付き制約が前件が充足されない限り影響を及ぼさない制約であることと対応させるためである.

3.3 例題と実行結果

提案記法の 2 つの構文を組み合わせて用いることで, 項 2.4 のモデルをより簡潔に表現することが出来る. 図 8 にプログラムを, 図 9 に出力を示す.

```
INIT(x,y,px,py,vx,vy) <=> x=px & y=py & x'=vx
  & y'=vy.
FALL(x,y) <=> [](y''=-10 & x''=0).
BOUNCE(x,y) <=> [](y-=0 => x'=x'- & y'=-4/5*y'-).
BREAK <=>
  y-=0 => {
    \nx.\ny.(
      INIT(nx,ny,x-,y-,1/2*x'-,-3/4*y'-)
      , FALL(nx,ny) << BOUNCE(nx,ny))}.

INIT(x,y,0,10,1,0), FALL(x,y) << BOUNCE(x,y),
BREAK.
```

図 8: 2 つに割れるボールのモデルを提案記法で書き直したプログラム

```
unadopted modules: {FALL(x,y)}
unsat mod : {BOUNCE(x,y), FALL(x,y)}
unsat cons : {y''=-10, y'=-4/5*y'-}
positive  :
  y-=0=>\nx1$.(\ny1$.(INIT(nx1$, ny1$,
       x-, y-, 1/2*x'-, 2*y'-)&
       FALL(nx1$, ny1$)&BOUNCE(nx1$, ny1$)))
              y-=0=>x'=x'-&y'=-4/5*y'-
negative  :
t : 2^(1/2)
nx1$3 : 2^(1/2)
ny1$3 : 0
x : 2^(1/2)
y : 0
nx1$3' : 1/2
ny1$3' : 2^(1/2)*(-20)
x' : 1
y' : 2^(1/2)*8
nx1$3'' : 0
ny1$3'' : -10
```

図 9: 2 つに割れるボールのモデルを提案記法で書き直したプログラム 出力 (一部)

モジュール BREAK が変更点である. 先頭に時相演算子 always が無い点, 後件が波括弧で囲まれている点, 論理積が優先度を付けるための演算子に変わっている点が図 6 との相違点である. 前件の成立で, 後件の優先度を持った制約同士が発行されるという意味になる. 次に, 項 3.2 の $D(t)$ に登録されるモジュールとそれらの関係を本例題のシミュレーションと照らし合わせながら確認する. 時刻 0 において, 図 10 の条件 (iii) によって D が以下の様に初期化される.

$$INIT(x, y, 0, 10, 1, 0), FALL(x, y) << BOUNCE(x, y),$$
$$BREAK.$$

次に, ボールが地面に衝突して条件付きモジュールの前件が満たされた時刻において, 後件のモジュール宣言文によって $D(t)$ がその時刻で以下の様に拡張される.

$$INIT(x, y, 0, 10, 1, 0), FALL(x, y) << BOUNCE(x, y),$$
$$BREAK, INIT(nx1\$3, ny1\$3, x-, y-, 1/2 * x'-, -3/4 * y'-),$$
$$FALL(nx1\$3, ny1\$3) << BOUNCE(nx1\$3, ny1\$3).$$

これは, この時刻で優先度を持ったモジュールが追加されたことを表す. 追加された全てのモジュールは親であるモジュール BREAK と同じ階層に展開されている. モジュール BREAK が, 生成された優先度を持つ制約をあたかも所持しているかの様にふるまう点が直感的であると考えている.

この後の時刻で条件付きモジュールのガード条件が成立した際, さらに新たな変数とモジュールが生成されてモジュール宣言文が拡張される.

4. まとめと今後の課題

本研究では, 存在量化子記法を用いた変数の動的生成がどのような例題で有効に機能するかを調査し, 処理系に実装した. また, 条件付き制約によって生成された制約同士の優先度が付けられないために, 生成した変数にかかる制約同士の関係を柔軟に記述できない問題を新たな記法を提案することで解決した. 今後の課題は, 提案記法がユーザの直感どおりに機能するかの確認をして, 記法や意味を洗練させることである.

$\langle \bar{x}(t), \langle Q, D\rangle\rangle \models P \Leftrightarrow (i) \wedge (ii) \wedge (iii) \wedge (iv) \wedge (v) \wedge (vi)$, ここで,

$$
\begin{aligned}
&(i) \quad \forall t \forall M \in MS(t)(Q(M) = Q(M)^*) \\
&(ii) \quad \forall M \in P(M^* \subseteq Q(M)) \\
&(iii) \quad D(0) = DC(P) \\
&(iv) \quad \forall t \forall t' \geq t(\\
&\qquad MS(t) \subseteq MS(t') \wedge R(t) \subseteq R(t')) \\
&(v) \quad \forall t \exists E \in MS(D(t))(&& (s0) \\
&\qquad (\bar{x}(t) \models \{Q(M)(t) \mid M \in E\}) && (s1) \\
&\qquad \wedge \neg \exists \bar{x}' \exists E' \in MS(D(t))(&& (s2) \\
&\qquad\quad \forall t' < t(\bar{x}'(t') = \bar{x}(t')) \wedge E \prec E' && (s2) \\
&\qquad\quad \wedge \bar{x}'(t) \models \{Q(M)(t) \mid M \in E'\}) && (s2) \\
&\qquad \wedge \forall d \forall e \forall M \in E((\bar{x}(t) \Rightarrow d) \\
&\qquad\qquad \wedge ((d \Rightarrow e) \in Q(M)(t)) && (s3) \\
&\qquad\quad \Rightarrow e \subseteq Q(M)(t)) && (s3) \\
&\qquad \wedge \forall d \forall M \in E \forall MS' \forall R'(&& (s4a) \\
&\qquad\quad (\bar{x}(t) \Rightarrow d) \wedge ((d \Rightarrow \langle MS', R'\rangle) = M) && (s4b) \\
&\qquad\quad \Rightarrow MS' \subseteq MS(t) \wedge R' \subseteq R(t) && (s4c) \\
&\qquad\quad \wedge \forall M_1 \in MS(t) \forall M_2 \in MS'(&& (s4d) \\
&\qquad\qquad M_2 \in Q(M_2)(t) && (s4e) \\
&\qquad\qquad \wedge \langle M_1, M\rangle \in R(t) \Rightarrow \langle M_1, M_2\rangle \in R(t) && (s4f) \\
&\qquad\qquad \wedge \langle M, M_1\rangle \in R(t) \Rightarrow \langle M_2, M_1\rangle \in R(t)) && (s4g) \\
&(vi) \quad \text{各時刻における } Q(M)(t) \text{ と } MS(t), R(t) \text{ は} \\
&\qquad \text{(i)-(v) を満たす最小の集合である.}
\end{aligned}
$$

図 10: $\langle \bar{x}, \langle Q, D\rangle\rangle \models P$ の定義

参考文献

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

[2] Borning, A., Freeman-Benson, B. and Wilson, M. : Constraint Hierarchies, Lisp and Symbolic Computation, Vol. 5, No. 3, 1992, pp. 223âĂŞ270.

[3] 上田和紀, 石井大輔, 細部博史 : 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, 2011, pp. 306–311.

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

[5] Matsumoto S., Ueda K.: Symbolic Simulation of Parametrized Hybrid Systems with Affine Arithmetic, TIME2016, 2016.