# 修 士 論 文 概 要 書

CD

2008 年 2 月提出　　学籍番号 3606U077 – 0

| 専攻名 | 情報・ネットワーク | 氏 名 | 西澤亮太 | 指導教員 | 上田　和紀　印 |
|---|---|---|---|---|---|
| 研究指導名 | 並列知識情報処理 | | | | |
| 研究題目 | ハイブリッド並行制約プログラムのハイブリッドオートマトンへの変換 | | | | |

## 1　はじめに

ハイブリッドシステムとは、連続変化と離散変化を持つ系を言う。ハイブリッドシステムをモデリングできる高水準言語としてハイブリッド並行制約プログラミング (HCC) が考案されている。HCC は、高水準言語から数値計算を用いてシミュレーションできる反面、処理の複雑さや計算の脆弱性などの問題点が挙げられる。
ハイブリッドシステムをモデル化する最も有名なフレームワークにハイブリッドオートマトン (HA) がある。HA は、状態遷移図を用いた低水準な表現方法で、精度保証を用いた計算などのアルゴリズムが開発されている。
これまでに、HCC でハイブリッドシステム検証を行うため、HA への変換について考えられたことがある。しかし、検証のための最低限の変換しか行っていない。今回は、HCC の静的解析を行い、HA を対象に研究されているアルゴリズムを応用できるような変換を目的とする。また、今後のシステムの開発のために高水準言語 HCC を低水準な表現に落とし込むことで HCC の操作的意味論を議論し易くなる。
HCC から変換した HA が、実行を不足なく変換し、できるだけ冗長でないように生成することを目指す。

## 2　ハイブリッド並行制約プログラム

ハイブリッド並行制約プログラミング (HCC) でモデリングするためのエージェントを記す。

| | | | |
|---|---|---|---|
| $A$ | := | $c$ | (tell) |
| | \| | if $ac$ then $A$ | (Positive ask) |
| | \| | unless $ac$ then $A$ | (Negative ask) |
| | \| | hence $A$ | (Unit Delay) |
| | \| | $A, A$ | (Parallel Composition) |
| | \| | new $X$ in $A$ | (Hidding) |

HCC の実行は、二つのフェーズの繰り返しからなる。状態の離散変化を計算するポイントフェーズと状態の連続変化を計算するインターバルフェーズがある。プログラムは、ポイントフェーズから計算し、二つのフェーズを交互に繰り返す。
HCC の実行は、状態変数 X$\in R^n$ を制御する不等式を tell 制約 c により定義し、tell 制約の集合よりフェーズで状態変数を制御する制約ストアで実行する。ask 制約 ac は、ask エージェントによって不等式 ac を X $\in$ ac を満たすか制約ストアに確認を行う不等式である。

## 3　ハイブリッドオートマトン

ハイブリッドオートマトンは、離散変化事象系を表すオートマトンの概念を各離散状態間に連続で変化する状態を取り入れることで、ハイブリッドシステムを表現できるように拡張したものである。
HCC から変換されるハイブリッドオートマトン H=(V,E) を以下のように定義する。実行に使用する状態変数 X は、HCC のプログラムで使われている変数と同じ物である。HCC では、変数宣言が存在しないので、各制約内の変数を抽出する。

### 3.1　node

$Invariant : I(v) = \{ac1, \cdots, acn \subseteq R^n \mid v \in V\}$
ノードに停滞していられる状態変数の条件を表す。不等式 ac の連言で表すことが出来る。Flow:$F(v) = \{X \rightarrow f(X,t) \mid X \in I(v), v \in V\}$(node における実行)
$f(X,t) = \{\int C(X) \mid (c1, \cdots, cn) = C\}$
ノードにおいて、時間と共に変化する状態変数を tell 制約の集合で表す。

### 3.2　edge

init=e、状態変数の初期状態を定め、最初の連続実行系をターゲットするエッジを init として持つ。init は、s(e)=NULL となるエッジである。
$Sorce : s(e) = \{v \mid e = (v, v')\}$ 有向辺の元。
$Target : t(e) = \{v' \mid e = (v, v')\}$ 有向辺の先。
$Guard : G(e) = \{ac1, \cdots, acn \subseteq R^n \mid e \in E\}$ X で G(e) が満たされたら、制御が辺のソースからターゲットへ移る。条件は ask 制約の連言で表す。
$Reset\_map : R(e) = \{X \rightarrow reset(X) \in I(t(e)) \mid X \in G(e), e \in E\}$(リセットにおける実行)$reset(X) = \{X \rightarrow C(X) \mid (c1, \cdots, cn) = C\}$
G(e) が満たされ、制御が移る時に R(e) によって状態変数の値を書き換えることが可能。c の連言で表す。

## 4　アルゴリズム

HCC から HA を生成するアルゴリズムについて説明する。HCC で記述したモデルに対し、HA で正確な実行が可能で、記号的な変換ができ、複雑な数値計算を扱わず、なるべく簡潔な HA を出力できるアルゴリズムを考える。

### 4.1 エージェントの変換

$CURRENT = \{A_1, \cdots\}$, このフェーズで処理しなければいけないエージェントの集合。$NEXT = \{A_1, \cdots\}$ 次のフェーズで処理するエージェントの集合。$ASK = \{ac1, \cdots\}$ask 制約の集合 $STORE = \{c1, \cdots\} = R(e) = F(e)$tell 制約の集合。これらをノード、エッジの構成と呼ぶ。各エージェントの変換処理を説明する。

**tell** $q : \langle CURRENT, (STORE, c), ASK, NEXT\rangle$

**parallel** $q : \langle (CURRENT, A, A), STORE, ASK, NEXT\rangle$

**hidding** $q : \langle (CURRENT, A), STORE, ASK, NEXT\rangle$

**ask** $q : \langle (CURRENT, A), STORE, (ASK, ac), NEXT\rangle$
$q : \langle CURRENT, STORE, (ASK, \neg ac), NEXT\rangle$

**unless** $q : \langle CURRENT, STORE, (ASK, ac), NEXT\rangle$
$q : \langle (CURRENT, A), STORE, (ASK, \neg ac), NEXT\rangle$

**hence** $q : \langle CURRENT, STORE, ASK, (NEXT, A)\rangle$
(point phase)
$q : \langle (CURRENT, A), STORE, ASK, (NEXT, A)\rangle$
(interval phase)

ask エージェントでは、新たに一つフェーズを生み出すが、ask 制約の成立関係を任意にユーザが指定することで制限できる。

### 4.2 HA の生成

エージェントの変換処理を用いて以下の 5 つのステップで変換できる。

1. init の生成。HCC のプログラムを Agent として HA の init となるエッジ e の CURRENT に入力する。e の構成は最初全て空である。
   $init : \langle (CURRENT, A), STORE, ASK, NEXT\rangle$
   今回の HA は、初期値を限定せずに init を作れる。ask(t(init)) を任意に ac を代入することで生成することができる。
   $t(e) : \langle CURRENT, STORE, (ASK, ac), NEXT\rangle$

2. エッジ e をつくる。$\forall A \in$ CURRENT に処理を行なう。Reset=STORE,Guard=ask で HA の $e \in E$。t(e) は次のステップで作る。

3. ターゲットの決まっていない、$e \in E$ を取り出して、ノード v の候補を作る。
   t(e)=v。R(e)$\in$ G(e),CURRENT(e)=NEXT(e) が成り立つ e は、実行が継続するので、e の構成がそのまま v の構成になる。計算の手間と、オーダーを削減できる。成り立たなければ、
   CURRENT(v)=CURRENT(e),(NEXT(v)=NEXT(e)) で生成。

4. 候補 v の $\forall A \in$ CURRENT を処理する。処理後、$STORE(v) = STORE(v')$ ならば実行が等しく、$NEXT(v) = NEXT(v')$ ならば次のエッジの生成が等しいため $v = v'$ である。$v' \in V$ なら v は破棄し、t(e)=v' とする。これにより、有限時間でないモデルも扱える。v' が存在しなければ、
   Flow=STORE,Invariant=ask とし、v$\in$V。

5. s(e)=v となるエッジを生成する。到達する範囲を $\exists$ac$\in$ask(v),$F(v)(0 < t < \infty) \in \neg$ac で特定し、
   $e : \langle CURRENT, STORE, (ASK, \neg ac), NEXT\rangle$
   とする。条件を満たす ac が存在しなければ、終了。e が生成されたら、2 から繰り返す。到達不能な ac は任意に指定し、到達可能性解析の役に立つ。

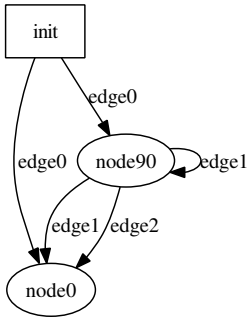
## 5 まとめと今後の課題

まず、冗長だが正確な実行を得られる HA を作れるアルゴリズムを考えた。正確な実行の変換は、[1] で述べられている。次に、冗長な HA をモデルから得られる情報を利用して削減する方法を組み込んだ。その実装の例を以下に示す。まだ冗長な部分があるのと自動化できることが分かっている部分があるので、今後改良を加えたい。

```
x = 10[c0],always{
  cont(x)[c9],
  if x = 0[ac3] then x '=-0.5*prev(x')[tell2],
  else[ac4]
     {x''=-9.8[c3],if prev(x')<0[ac12]thenx'=0[c13]}
}
```

実行結果 order2size5 と Graphviz による出力

| state | STORE | ASK | INVERSE | output |
|---|---|---|---|---|
| node0 | flow:9,13 | $I^+$:4,12 | $I^-$:3 | edge: |
| node90 | flow:9,1 | $I^+$:4 | $I^-$:3,12 | edge:1,2 |
| edge0 | reset:0,9,1 | $G^+$:4 | $G^-$:3,12 | node:0,90 |
| edge1 | reset:9,2 | $G^+$:3 | $G^-$:4 | node:0,90 |
| edge3 | reset:9,,13 | $G^+$:4,12 | $G^-$:3 | node:0 |




## 参考文献

[1] V. Gupta, R. Jagadeesan, and V. A. Saraswat, "Hybrid cc, hybrid automata and program verification" in Hybrid Systems III, R. Alur, T. A. Henzinger, and E. D. Sontag, Eds: Springer-Verlag, 1996, pp. 52–75. LNCS 1066