最強事後条件の計算を用いたハイブリッドオートマトンの帰納的検証

石井 大輔　Guillaume Melquiond　中島 震

ハイブリッドシステムの安全性検証は物理環境と結合されたシステムの開発において重要な技術である．本発表ではハイブリッドオートマトンを対象とした自動的な論理的分析手法を提案する．提案手法は従来のモデル検査法と較べて広範な問題に適用可能であり，記号パラメタや非線形式で記述されるモデルを扱うことができる．提案手法では，まずハイブリッドオートマトンの実行トレースを逐次プログラムに変換し，その後，帰納法にもとづくプログラムのループ構造への変換と最強事後条件の計算により，安全性を証明するための検証条件群を生成する．検証条件をMathematica 等の算術ソルバーで妥当性判定することにより元モデルの安全性を示すことができる．提案するアルゴリズムは，展開する実行の長さの調整と，証明できなかった検証条件からループ不変条件を生成する処理をおこないながら，帰納的検証手法を自動化する．文献由来の例題群に対し実験を実施し，実用性を示す結果を得た．

1　はじめに

連続的ダイナミクスをともなう遷移系であるハイブリッドシステムは，物理環境と結合した組み込みシステムの直截的なモデルであり，その高信頼な検証技術の確立は重要かつチャレンジングな課題である．

ハイブリッドシステムの検証手法としてモデル検査法と論理的分析法の 2 つのアプローチが提案されている．モデル検査のアプローチでは，ハイブリッドオートマトンを区分線形系等に抽象化し，状態の網羅的探索により検証をおこなう．モデル検査法は自動化することができ，HyTech [10]，PHAVer [6]，HybridSAL [20] 等の多くのツールが開発され，実用的な問題に応用されてきた．2 つ目の論理的分析アプローチでは，演繹的推論に基づきハイブリッドシステムの検証に必要となる検証条件を導出し，それらの妥当性を示すことにより検証をおこなう．論理的分析法は，理論的な面では多くの研究がなされてきたものの，ツール実用化の面では遅れていた．例外的な研究成果として KeYmaera [16] があり，モデル検査アプローチでは扱えなかったパラメタを含む系や非線形系等，広範なクラスの問題が検証可能であることを示した．論理的分析法の欠点は，系が大きく複雑になると，検証プロセスが自動的でなくなり，ユーザにモデルの理解と検証戦略の選択等の対話的作業を強いる点である．

本論文では，算術ソルバーを利用して半自動的にハイブリッドシステムの論理的分析をおこなうツールを提案する．提案手法はハイブリッドオートマトンの安全性検証を対象とする．

まず本論文では，ハイブリッドオートマトン (2 節) の安全性を示すための有限長の検証条件を導出する健全な方法を与える．提案手法では，ハイブリッドオートマトンの実行を直線的な逐次プログラムに変換し (3 節)，プログラムに対して帰納法を適用することによりループ構造を構成する (4 節)．すなわち，ハイブリッドオートマトンのすべての実行が，最大 m ステップの連続変化と離散変化の後，最大 n ステップ周期で状態空間中のある領域を繰り返し訪れるようなループをなすことを示すためのプログラムを構成す

Inductive Verification of Hybrid Automata with Strongest Postcondition Calculus.
Daisuke Ishii, 東京工業大学, Tokyo Institute of Technology.
Guillaume Melquiond, INRIA サクレ / パリ第 11 大学, INRIA Saclay / Université Paris Sud 11.
Shin Nakajima, 国立情報学研究所, National Institute of Informatics.

$$\frac{\phi(0)=\nu \quad t>0 \quad \forall\tilde{t}\in[0,t]\ \frac{d\phi(\tilde{t})}{dt} = F_l(\phi(\tilde{t}))\wedge I_l[\phi(\tilde{t})]}{\langle l,\nu\rangle \xrightarrow{t} \langle l,\phi(t)\rangle} \qquad\qquad \frac{G_{l_1,l_2}[\nu_1]\quad R_{l_1,l_2}[\nu_1,\nu_2]\quad I_{l_2}[\nu_2]}{\langle l_1,\nu_1\rangle \xrightarrow{0}\langle l_2,\nu_2\rangle}$$

図 **1**　ハイブリッドオートマトンの操作的意味論

る．その後，最強事後条件の計算 (3.2 節) を用いて逐次プログラムから検証条件を導出する．これらの検証条件は実数上の算術式や常微分方程式を含んだ一階述語論理式である．実際に数式処理系 Mathematica を用い，線形および一部の非線形なハイブリッドオートマトンに関する検証条件が証明可能であることを示す．上記の検証プロセスを提案するアルゴリズムにより自動化する (4.2 節)．ただし (i) ループ構造を構成するために実行を展開するステップ数 m および n と，(ii) ループ構造の冒頭で成立すべきループ不変条件を効率よく見つけるためにユーザの対話的作業を要する．代数的手法によるループ不変条件の生成についても述べる (4.3 節)．提案手法の実装を用い (5 節)，文献由来の例題群を検証した実験結果を述べる (6 節)．

2　ハイブリッドオートマトン

ハイブリッドオートマトン (hybrid automata) [9] はハイブリッドシステムを表す数理モデルである．

定義 1. ハイブリッドオートマトンを以下の要素からなる 7 つ組 $HA = \langle L, V, \mathit{Init}, \mathcal{G}, \mathcal{R}, \mathcal{F}, \mathcal{I}\rangle$ で表す．

- ロケーションの有限集合 $L = \{l_1, \ldots, l_p\}$.
- 実変数の有限集合 $V = \{x_1, \ldots, x_q\}$．変数へのすべての付値の集合を $\mathbb{R}^V$ で表す．
- 初期条件 $\mathit{Init} \subseteq L\times\mathbb{R}^V$.
- ガード条件の族 $\mathcal{G} = \{G_{l,l'}\}_{l\in L, l'\in L}$．ただし $G_{l,l'}$ は $\mathbb{R}^V$ に関する条件である．
- リセット関数の族 $\mathcal{R} = \{R_{l,l'}\}_{l\in L, l'\in L}$．ただし $R_{l,l'} : \mathbb{R}^V \to \mathbb{R}^V$ とする．
- ベクトル場の族 $\mathcal{F} = \{F_l\}_{l\in L}$．ただし $F_l : \mathbb{R}^V \to \mathbb{R}^V$ とする．
- ロケーション不変条件の族 $\mathcal{I} = \{I_l\}_{l\in L}$．ただし I_l は $\mathbb{R}^V$ に関する条件である．

列 $\sigma_0 \xrightarrow{t_1} \sigma_1 \xrightarrow{t_2} \cdots$ を (有限長または無限長の) 実行という．ただし $\sigma_i \in L\times\mathbb{R}^V$ と $\mathit{Init}[\sigma_0]$ がなりたつとする．$\xrightarrow{t_i}$ を $t_i > 0$ のとき連続変化，$t_i = 0$ のとき離散変化とよび，図 1 の 2 つのルールで定義する．

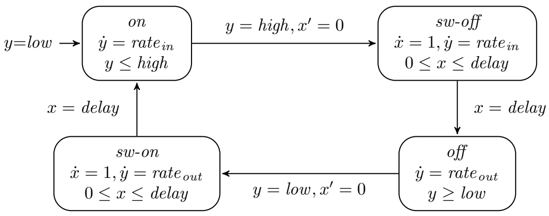

図 **2**　水槽の水位制御のモデル.

$\sigma_0 \xrightarrow{t_1} \sigma_1 \xrightarrow{0} \sigma_2 \xrightarrow{t_3} \cdots \xrightarrow{0} \sigma_{2k}$ の形をした実行を長さ k の標準形と呼ぶ．

本論文では，複数の離散変化が同時刻に起こらず，最初に $\langle l_0, \nu_0\rangle \xrightarrow{0} \sigma_1$ のように離散変化が起こらないと仮定する．一般的な実行に関する検証は今後の課題とする．

定義 2. 安全性 (不変性ともいう) を式 $\Box P$ で表す．ただし P は $L\times\mathbb{R}^V$ に関する条件である．式 P は，各実行の初期状態に関して成り立ち，各連続変化および離散変化において保存される．

例 1 (水槽 [3] [13])．　水位制御装置付きの水槽を図 2 のハイブリッドオートマトンでモデリングする．ロケーション *off* および *sw-on* では水槽の出力孔から水が放出されるため，水位 y が一定速度 $rate_{out}$ で減少し，ロケーション *on* および *sw-off* では制御装置が水を注入するため，y が一定速度 $rate_{in}$ で増加する．制御装置は y が値 low あるいは $high$ に達するのに応じてロケーション *on* と *off* を切り替える．さらにロケーション *sw-on* および *sw-off* によって上記の切り替えに遅延 $delay$ が発生することを表している．安全性の例として，水位 y が限界値 min と max の間につねに収まっていることを，つぎのように記述できる．

$$\Box(min \le y \le max) \tag{1}$$

また各定数パラメタについて下記のように制約する．

$$\begin{aligned} &min \le low \wedge high \le max \wedge low < high \wedge \\ &delay > 0 \wedge max \ge high + rate_{in}\cdot delay \wedge \\ &min \le low + rate_{out}\cdot delay \end{aligned} \tag{2}$$

3 ハイブリッドオートマトンの実行から逐次プログラムへの変換

本節ではプログラム検証の技術を利用してハイブリッドオートマトンの有限長・無限長の実行を扱うための準備をおこなう．まず，実行を記述するための簡単な逐次言語を導入する．つぎに，各プログラム文と事前条件から導出される最強事後条件を説明する．

3.1 逐次言語

ハイブリッドオートマトン HA について，逐次言語 $\mathbf{Imp}_{HA}$ を与える．$\mathbf{Imp}_{HA}$ は 2 つのコマンド `evolve` および `trans` と，それらの列からなる．コマンド `evolve` は連続変化，`trans` は離散変化を記述する．

定義 3. 言語 $\mathbf{Imp}_{HA}$ の構文を以下のように定める．

$$s ::= \texttt{skip} \mid s;\ \ s \mid \texttt{evolve}\ t \mid \texttt{trans}$$

定義 4. プログラム状態 (S や S_i で表す) は変数名から変数値への写像である．変数 x_s は HA の実行における「現在」状態 ($\in L \times \mathbb{R}^V$) を表す．また当該状態を変数群 $x_s = \langle x_l, \cdot \rangle = \langle \cdot, x_v \rangle = \langle \cdot, (x_1, \ldots, x_i, \ldots, x_q) \rangle$. により部分的に表すことができるものとする (いずれかの変数値を更新した場合，上記の等価性が保たれるよう他の変数値も暗黙的に書き換わる)．

図 3 に $\mathbf{Imp}_{HA}$ の操作的意味論を示す．$[\![e]\!]_S$ は表現 e 中の自由変数をプログラム状態 S が与える値で置き換えることを表す．$S\{x \mapsto v\}$ はプログラム状態 S を変数 x に値 v を与えるよう更新したプログラム状態を表す．`skip` と連結構文のルールは典型的なものである．

任意の HA の有限長実行について対応する $\mathbf{Imp}_{HA}$ プログラムを与えることができる．

補題 1. 以下の形をした任意の HA の実行 $\sigma_0 \xrightarrow{t_1} \sigma_1 \xrightarrow{0} \cdots \xrightarrow{t_k} \sigma_{2k-1}$ に対し，下記の $\mathbf{Imp}_{HA}$ プログラムを与えることができる．

$$\texttt{evolve}\ t_1;\ \texttt{trans};\ \cdots;\ \texttt{evolve}\ t_k$$

このプログラムは下記すべてをみたす実行をもつ．

- 初期プログラム状態において $x_s = \sigma_0$ が成立．
- プログラムが `skip` に簡約される．
- 終了プログラム状態において $x_s = \sigma_{2k-1}$ が成立．

上記の HA の実行は標準形に 1 つ連続変化を加えた形になっている．上記プログラムは途中でブロックする (`skip` に簡約されない) 実行や HA の異なるロケーションをたどる実行ももつことに注意されたい．上記の $\mathbf{Imp}_{HA}$ プログラムを用いて安全性の判定条件を与えることができる．

補題 2. もし任意の上記の形の $\mathbf{Imp}_{HA}$ プログラムについて，初期プログラム状態 S_0 が $[\![x_s]\!]_{S_0} \in \mathit{Init}$ をみたすとき，性質 P が終了プログラム状態においてなりたつならば，HA は安全性 $\Box P$ をみたす (ただし Zeno 時刻が存在するならば最初の Zeno 時刻までに関する)．

3.2 最強事後条件

本節ではプログラム検証の基本概念である最強事後条件 [11] [5] を用いて，$\mathbf{Imp}_{HA}$ プログラムの安全性のための検証条件を導出する方法を説明する．

提案手法では到達性ではなく安全性の検証を目的としており，プログラムが非ブロッキングであることを示す必要はない．そのため，最弱事前条件と最強事後条件の計算は双対となる．本稿では前向き到達性に対応する最強事後条件を用いるが，最弱事前条件を用いて検証を実施することも可能である．

補題 3 (最強事後条件と健全性)**.** 任意の $\mathbf{Imp}_{HA}$ プログラム s について，もし初期状態が性質 P をみたすならば，終了状態は $SP(P, s)$ をみたす (s が `skip` に簡約可能とする)．ただし SP は図 4 のように定義する．

証明. プログラム状態 S と S' について $[\![P]\!]_S$ と $S, s \rightsquigarrow^* S', \texttt{skip}$ がなりたつと仮定する．$[\![SP(P, s)]\!]_{S'}$ がなりたつことを示せばよい．それには s の構造に応じた SP の定義の各部が $\mathbf{Imp}_{HA}$ の操作的意味論 (図 3) を含意することを帰納的に確認すればよい．実際，SP の定義の各部は ($\mathbf{Imp}_{HA}$ の操作的意味論で前提としている) ハイブリッドオートマトンの操作的意味論 (図 1) に基づいており，明らかである． □

例 2. 例 1 において $x_l = \mathit{on} \wedge y = \mathit{low}$ をみたす状態を考える．長さ t の連続変化により到達する状態が $y \leq \mathit{max}$ をみたすことを証明する．補題 2 および 3 から，任意のプログラム状態について下記がなりた

$$\frac{}{S, (\texttt{skip};\ s) \rightsquigarrow S, s} \qquad \frac{S_1, s_1 \rightsquigarrow S_2, s_2}{S_1, (s_1;\ s_3) \rightsquigarrow S_2, (s_2;\ s_3)} \qquad \frac{}{S, \texttt{evolve}\ 0 \rightsquigarrow S, \texttt{skip}}$$

$$\frac{[\![x_s]\!]_S \xrightarrow{t} \sigma}{S, \texttt{evolve}\ t \rightsquigarrow S\{x_s \mapsto \sigma\}, \texttt{skip}} \qquad \frac{[\![x_s]\!]_S \xrightarrow{0} \sigma}{S, \texttt{trans} \rightsquigarrow S\{x_s \mapsto \sigma\}, \texttt{skip}}$$

図 3　$\mathbf{Imp}_{HA}$ の操作的意味論.

$$SP(P, \texttt{skip}) := P \qquad SP(P, s_1;\ s_2) := SP(SP(P, s_1), s_2)$$
$$SP(P, \texttt{evolve}\ t) := \exists\phi\ P[x_v \leftarrow \phi(0)] \wedge \phi(t) = x_v \wedge (\forall\tilde{t} \in [0, t]\ \tfrac{d\phi(\tilde{t})}{dt} = F_{x_l}(\phi(\tilde{t})) \wedge I_{x_l}[\phi(\tilde{t})])$$
$$SP(P, \texttt{trans}) := \exists\langle l', x'_v\rangle\ P[x_s \leftarrow \langle l', x'_v\rangle] \wedge G_{l', x_l}[x'_v] \wedge x_v = R_{l', x_l}(x'_v) \wedge I_{x_l}[x_v]$$

図 4　$\mathbf{Imp}_{HA}$ プログラムの最強事後条件.

つことを示せばよい.

$SP((x_l = on \wedge y = low), \texttt{evolve}\ t) \Rightarrow y \leq max$

ここで左辺をみたすプログラム状態にいると仮定し，$y \leq max$ がなりたつことを示す．SP の定義から，下記のような関数 ϕ が存在する.

$$(x_l = on \wedge \phi_y(0) = low) \wedge \phi(t) = (x, y) \wedge$$
$$(\forall\tilde{t} \in [0, t]\ \tfrac{d\phi(\tilde{t})}{dt} = F_{x_l}(\phi(\tilde{t})) \wedge I_{x_l}[\phi(\tilde{t})])$$

上式中の微分方程式を解くことにより式 $y = low + rate_{in} \cdot t$ を求め，ロケーション不変条件 I_{x_l} を展開して $y \leq high$ で置き換える．さらに制約 (2) を考慮し，線形算術によりゴール $y \leq max$ を証明することができる.

注意 1. 検証条件を自動化ツールで扱う際，事前にできる限り限量子を除去しておくのが望ましい．たとえば $SP(P, \texttt{trans})$ は $\exists l'\ Q[l']$ という形をしているが，これは式 $Q[l_1] \vee \ldots \vee Q[l_p]$ ($l_1, \ldots, l_p$ で全ロケーションを表す) と等しい．$SP(P, \texttt{evolve}\ t)$ では，例 2 で述べたように常微分方程式を閉形式に変換できれば，$\exists\phi$ を除去できる.

4　帰納的検証手法

本節では補題 2 および数学的帰納法にもとづきハイブリッドオートマトンの安全性検証をおこなうアルゴリズムについて述べる.

4.1　帰納法による安全性検証

補題 2 では無限個のプログラムについて安全性を確認する必要があり，実用的ではなかった．本節では安全性検証のための前提条件を弱め，有限個のプログラムをもとに検証を実施する手法を示す.

提案手法では，まず $P^+ \Rightarrow P$ をみたす表明 P^+ を仮定し，ハイブリッドオートマトン HA が下記をみたすことを期待する.

- HA の初期状態 $Init$ から始まる高々m ステップ長の任意の実行について，すべての途中状態が安全で，P^+ をみたす状態に到達する.
- P^+ をみたす状態から始まる高々n ステップ長の任意の実行について，すべての途中状態が安全で，P^+ をみたす状態に到達する.

上記を条件式としてエンコードし，妥当性判定することにより安全性検証を実施する．本アプローチの成否は実行の長さ m および n と表明 P^+ を見つけることができるかどうかに依存する.

安全性の推論規則として，$m = 0, n = 1$ とした単純形を示す.

定理 1 (単純形)**.** 下記の推論規則がなりたつ.

$$\frac{\begin{array}{rl} VC_0: & Init \Rightarrow P^+ \\ VC_1: & \forall t \geq 0\ SP(P^+, \texttt{evolve}\ t) \Rightarrow P \\ VC_{-1}: & \forall t \geq 0\ SP(P^+, \texttt{evolve}\ t;\ \texttt{trans}) \Rightarrow P^+ \end{array}}{HA \models \Box P}$$

証明. VC_0 は初期状態が表明 P^+ をみたすことを確認している．VC_{-1} は連続変化と離散変化からなる任意の実行 $\sigma_i \xrightarrow{t_{i+1}} \sigma_{i+1} \xrightarrow{0} \sigma_{i+2}$ について，σ_i が P^+ をみたすとき，σ_{i+2} もまた P^+ をみたすことを帰納的に確認している．VC_1 は上記連続変化の途中で安全性がつねになりたつことを確認している．　□

上記定理 1 の推論規則を任意の m, n について一般化する．

定理 2 (展開形)**.** 下記の $m+n+2$ 個の最強事後条件を考える．

$$
\begin{array}{lcl}
SP_1 & \equiv & SP(\mathit{Init} \land \neg P^+, \texttt{evolve } t_1) \\
SP_2 & \equiv & SP(SP(SP_1, \texttt{trans}) \land \neg P^+, \texttt{evolve } t_2) \\
 & \vdots & \\
SP_m & \equiv & SP(SP(SP_{m-1}, \texttt{trans}) \land \neg P^+, \\
 & & \qquad \texttt{evolve } t_m) \\
SP_0 & \equiv & SP(SP_m, \texttt{trans}) \\
SP_{m+1} & \equiv & SP(P^+, \texttt{evolve } t_1) \\
 & \vdots & \\
SP_{m+n} & \equiv & SP(SP(SP_{m+n-1}, \texttt{trans}) \land \neg P^+, \\
 & & \qquad \texttt{evolve } t_n) \\
SP_{-1} & \equiv & SP(SP_{m+n}, \texttt{trans})
\end{array}
$$

下記の推論規則がなりたつ．

$$
\frac{
\begin{array}{lcrl}
VC_1 & : & \forall t_1 \geq 0 & SP_1 \Rightarrow P \\
VC_2 & : & \forall t_1, t_2 \geq 0 & SP_2 \Rightarrow P \\
 & & \vdots & \\
VC_m & : & \forall t_1 \dots t_m \geq 0 & SP_m \Rightarrow P \\
VC_0 & : & \forall t_1 \dots t_m \geq 0 & SP_0 \Rightarrow P^+ \\
VC_{m+1} & : & \forall t_1 \geq 0 & SP_{m+1} \Rightarrow P \\
 & & \vdots & \\
VC_{m+n} & : & \forall t_1 \dots t_n \geq 0 & SP_{m+n} \Rightarrow P \\
VC_{-1} & : & \forall t_1 \dots t_n \geq 0 & SP_{-1} \Rightarrow P^+
\end{array}
}{HA \models \Box P}
$$

証明. 検証条件 VC_1 から VC_0 により，高々m ステップ長の初期の実行が領域 P^+ に到達することを確認している．検証条件 VC_{m+1} から VC_{-1} により，領域 P^+ から始まる高々n ステップ長の実行がふたたび領域 P^+ に到達することを確認している． □

VC_{m+1} 以外の検証条件では P^+ が不成立を仮定していることに注意されたい．これは n ステップ未満で領域 P^+ にふたたび到達する場合を検証するための措置である．

4.2 検証アルゴリズム

図 5 のアルゴリズムは，ハイブリッドオートマトン HA，安全性 $\Box P$，実行を展開する最大長 m_{max}, n_{max} を入力とし，定理 2 中のすべての前提条件がなりたつことを確認し，$HA \models \Box P$ を判定する．アルゴリズムは各 $m \leq m_{max}, n \leq n_{max}$ について帰納的検証をおこなう (1 行目)．アルゴリズムはまず初期実行について計算し (4 行目)，つぎにループ部について計算する (7 行目)．手続き Validate は引数の論理式が妥当であれば true を返し，妥当性を示すことができなければ false を返す．すべての検証条件の妥当性を示すことができれば検証が成功する (10 行目)．

Input: HA; P; $m_{max} \in \mathbb{N}_{\geq 0}$; $n_{max} \in \mathbb{N}_{>0}$
Output: true: $HA \models \Box P$;
false: $m_{max} + n_{max}$ ステップ以下で $\Box P$ を判定不可

1: **for** $m \in \{0, \dots, m_{max}\}$; $n \in \{1, \cdots, n_{max}\}$ **do**
2: $P^+ := P$
3: **while** $P^+ \not\equiv \mathit{false}$ **do**
4: **if** $\neg \forall i \in \{0, \dots, m\}$ Validate(VC_i) **then**
5: **break**
6: **end if**
7: **if** $\exists j \in \{m+1, \dots, m+n, -1\}$ $\neg$Validate(VC_j) **then**
8: $P^+ := P^+ \land$ Learn(VC_j)
9: **else**
10: **return** true
11: **end if**
12: **end while**
13: **end for**
14: **return** false

図 5 検証アルゴリズム．

ループ部の検証がうまくいかないとき，妥当性判定に失敗した検証条件の証明がうまくいくよう手続き Learn によりループ不変条件を詳細化し (8 行目)，検証を再度試みる．Learn がループ不変条件を強め過ぎた場合には (3 行目)，内部のループから抜け出し別の m, n による検証を試み，それでも成功しなければ false を返す．

アルゴリズムにおいて，初期実行の検証 (4–6 行目，以下この手続きを BaseCase とよぶ) とループ部の検証 (7-11 行目，以下 Induction) は独立しており，これらの順序を逆にしてもかまわない．

4.3 ループ不変条件の生成

手続き Learn の実装について述べる．ループ部の検証において条件 $VC_i \equiv \forall t_1 \dots t_i \geq 0\ SP(P^+, s) \Rightarrow P$ の検証に失敗したとする．このとき Learn(VC_i) は，ループ不変条件を $P^+ := P^+ \land Q$ と更新すれば VC_i

が妥当となるような式 Q を生成する．Learn は VC_i に代数的操作を施し，$SP(Q,s) \Rightarrow VC_i$ がなりたつような Q を求める．

Q に出現する現在状態を表す変数 x_s は P^+ が成立する時点での状態を表すべきであるが，VC_i 内の x_s はプログラム s の終了時点での状態を表している．この不一致を解決するため，Learn は，限量子除去 (quantifier elimination, QE) を用い，Q から特定時点に依存する変数群を消去する．

$$Q := \mathsf{QE}(\forall x_s \forall t_1 \ldots t_i \; SP((P^+ \wedge x_0 = x_s), s) \Rightarrow P)[x_0 \leftarrow x_s]$$

複数の実数変数と関数変数の限量子を含んだ式に対する限量子除去は，一般に困難な問題である．提案手法における検証条件の単純化について注意 1 と次節を参照されたい．

限量子除去の結果，しばしば論理和の形の大きな式が求まり，このような式を Q として用いても検証がうまくいかないことがある．とくに Q の部分式が HA の実行において決して有効にならない挙動を表している場合，検証条件が必要以上に複雑になっている．そこで次節の実験においては，下記の方法により Q の簡略化・強化をおこなった．

- 式の分割．Q を最上位の論理和演算子で分割し，1 つまたは複数の部分式を採用する．
- ロケーションの否定．Q を分割した際，結果にロケーション l に関する部分式が含まれるとき，部分式を式 $x_l \neq l$ で置き換える．この方法はループ展開と組み合わせると有効である．

5 実装

前節までに説明した手法を記号的に算術式を扱うことが可能な数式処理系 Mathematica 8.0.4[†1] を用いて実装した[†2]．本実装では手続き Learn を完全には自動化しておらず，4.3 節で述べた方法をユーザが手動で適用する必要がある．Validate は Mathematica の組み込み手続き FullSimplify, Reduce, FindInstance を用いて 3 通りに実装した．ハイブリッドオートマトン中の微分方程式は組み込み手続き DSolve を用いて閉形式に変換している．

BaseCase と Induction の実装では，2 通りの最適化を施した．まず，各 VC_i を個別に妥当性判定するのではなく，共通の前提条件を再利用するようにした．VC_i の妥当性を判定する際，アルゴリズムはまず VC_i が依拠するプログラム s_i の実行結果を表す SP_i を計算してから，$SP_i \Rightarrow P/P^+$ を妥当性判定する．すでに SP_i を計算していれば，SP_{i-1} は効率よく計算できる．つぎに，プログラム s が複数ロケーションにまたがる実行集合に対応している場合の計算を効率化した．実装では，最強事後条件をロケーション毎に複製・管理し，妥当性判定を複数回おこなう (注意 1 を参照)．Validate を呼び出す回数は増えるが，一般に検証プロセス全体を効率化できる．

例 3. 実装を用いて例 1 の安全性 (1) を検証する例を示す．アルゴリズムに従い，まず $m=0, n=1$ について計算する．Init が $P^+ \equiv P$ を含意することを示すべく BaseCase を実行すると，true が得られる．つぎに Induction を実行し，連続変化と離散変化のシミュレーションをおこなう．Induction はロケーション $\mathit{on}, \mathit{sw\text{-}off}, \mathit{off}, \mathit{sw\text{-}on}$ 毎に最強事後条件・検証条件を生成し，それらの妥当性判定をおこなう．VC_1 の判定では，ロケーション on および off については成功するが，$\mathit{sw\text{-}off}$ および $\mathit{sw\text{-}on}$ については失敗する．そこで，Learn により下記の補助式を生成する．

$$\begin{aligned} Q_{\mathit{sw\text{-}off}} \equiv\, & \mathit{min} + x \cdot \mathit{rate}_{\mathit{in}} \leq y + \mathit{delay} \cdot \mathit{rate}_{\mathit{in}} \leq \\ & \mathit{max} + x \cdot \mathit{rate}_{\mathit{in}} \vee x = \mathit{delay} \vee \\ & y + \mathit{delay} \cdot \mathit{rate}_{\mathit{in}} < \mathit{low} + x \cdot \mathit{rate}_{\mathit{in}} \\ Q_{\mathit{sw\text{-}on}} \equiv\, & \mathit{min} + x \cdot \mathit{rate}_{\mathit{out}} \leq y + \mathit{delay} \cdot \mathit{rate}_{\mathit{out}} \leq \\ & \mathit{max} + x \cdot \mathit{rate}_{\mathit{out}} \vee x = \mathit{delay} \vee \\ & \mathit{high} + x \cdot \mathit{rate}_{\mathit{out}} < y + \mathit{delay} \cdot \mathit{rate}_{\mathit{out}} \end{aligned}$$

ここで，4.3 節の 2 つの方法を用いて 2 通りにループ不変条件を改良することができる．ロケーションの否定を適用した場合，P^+ に下記の補助式を加えることになる．

$$Q_1 := x_l \neq \mathit{sw\text{-}off} \wedge x_l \neq \mathit{sw\text{-}on}$$

更新後の P^+ を用い，$m=0, n=2$ とすると検証が成功する．補助式の分割を適用した場合，式 $Q_{\mathit{sw\text{-}off}}$

---

†1 `http://www.wolfram.com/mathematica/`

†2 `http://www.ueda.info.waseda.ac.jp/~ishii/pub/mathybrid/` にて公開．

と $Q_{sw\text{-}on}$ を最上位の論理和で 3 つの部分式に分割し，各補助式の最初の部分式 ($Q_{sw\text{-}off,1}$ と $Q_{sw\text{-}on,1}$ とする) を用いることで検証が成功する．具体的には P^+ に下記の補助式を加える．

$$Q_2 := (x_l = sw\text{-}off \Rightarrow Q_{sw\text{-}off,1}) \wedge (x_l = sw\text{-}on \Rightarrow Q_{sw\text{-}on,1})$$

すると $m = 0, n = 1$ のもと検証が成功する．

6 実験結果

提案手法の実用性を評価するため，複数の例題を検証する実験を実施した．また，比較のために同じ例題を既存ツール (HyTech, PHAVer, KeYmaera) を用いて検証した．実験結果を表 1 に示す．各列は，各 HA のロケーション数，変数の数，ループ展開数 (初期実行/ループ部)，ループ不変条件 P^+ を補助定理で改良した回数，提案手法で検証条件の判定に要した CPU 時間，HyTech (バージョン 1.04f，H) あるいは PHAVer (バージョン 0.38，P) による検証に要した時間，KeYmaera (バージョン 3.0) による検証に要した時間，を示している (「–」は検証に失敗したことを示す)．実験は 3.4GHz Intel Xeon プロセッサと RAM 4GB を備えたマシン上で実施した．ただし，提案手法の実行時間はループ不変条件 P^+ の改良を終えてから検証にかかる時間のみを計測していることに注意されたい．

6.1 例題

水槽 (例 1)．提案手法を用い，例 3 で述べたように検証できる．流量を表すパラメタ変数による非線形項のため，モデル検査器はこの問題を扱うことができなかった．また KeYmaera でも検証することができたが，モデルに (例 3 のものよりも複雑な) ループ不変条件を追加する必要がある [15]．

ガスバーナー． 2 つのロケーション $L = \{leaking, non\text{-}leaking\}$ からなる線形ハイブリッドオートマトンの検証問題である．実装を用いると Induction が最初の連続変化の検証に失敗した．Learn を用いてロケーション *leaking* に関する補助式を生成，P^+ に追加し，ロケーション *non-leaking* に関してはロケーションの否定を P^+ に追加した．すると $m = 4, n = 2$ のもと，検証が成功した．HyTech は効率よくこの問題を検証することができた．KeYmaera ではループ不変条件を追加しても検証できなかった．

温度制御 [3]．あるロケーションから複数のエッジが出ているハイブリッドオートマトンについて，ロケーション `shutdown` に決して到達しないことを検証する問題である．安全性条件に `shutdown` への離散変化のガード条件の否定を加え，強めるよう変更した後，実装を用いて検証することができた．Induction での妥当性判定の失敗から $Q_1 \vee Q_2 \vee Q_3$ という形の補助式を得たが，各部分式を P^+ に加えただけでは検証が成功しなかった．そこで $Q_1 \wedge (Q_2 \vee Q_3)$ という形の式を P^+ に加えたところ，$m = 1, n = 1$ のもと検証に成功した．HyTech は効率よくこの問題を検証することができた．KeYmaera ではループ不変条件を追加しても検証できなかった．

跳ね返るボール．典型的な非線形ハイブリッドオートマトンの例である．ボールの初期位置，初速，床の反発係数をパラメタ化し，反発係数が 1 以下ならばボールの高さがボールの初期エネルギーを超えないことを検証した．4 つの検証条件は Mathematica を用いて証明した．このようにパラメタ化したモデルはモデル検査器では検証できなかった．KeYmaera は同様のモデルを効率よく検証できた．

鉄道制御 [14] [8] [2]．位置 z にいる鉄道が位置の上限 m を越えないことを検証する問題である．[14] でのモデルに対し，[2] における解析に基づき，ガード条件を詳細化した．$m = 0, n = 1$ のもと検証を実施し，検証条件 VC_1 の判定失敗から P^+ を改良した後，成功した．本問題は [14] [8] においても独自の検証戦略，モデル変換，不変条件生成を駆使して検証されている．モデル検査器では非線形式に起因して検証できなかった．KeYmaera は [14] で述べられているようにパラメタ・制約式を変更したモデルを検証することができた．

高速道路 [12]．n 台の自律走行する車からなる高速道路のモデルである．実装を用いて $n = 9$ と $n = 19$ のインスタンスを検証することができた．[12] では同問題を専用手法を用いて検証している．PHAVer は $n = 9$ のインスタンスを検証できたが，$n = 19$ では

表 **1** 実験結果.

例題	locs	vars	unroll	lemmas	CPU 時間	モデル検査器	KeYmaera
水槽 (例 1)	4	2	0/1	2	0.85s	–	1.8s
ガスバーナー	2	3	4/2	3	2.22s	0.004s (H)	–
温度制御	4	3	1/1	4	2.82s	0.012s (H)	–
跳ね返るボール	1	2	0/1	1	0.49s	–	0.9s
鉄道制御	2	3	0/1	1	4.48s	–	3.1s
高速道路 9	10	9	0/2	1	0.22s	0.22s (P)	–
高速道路 19	20	19	0/2	1	3.64s	–	–

使用メモリが限界に達し失敗した．KeYmaera では検証することができなかった.

6.2 考察

モデル検査器 HyTech および PHAVer を用いて 3 つの例題を効率よく検証することができたが，他の問題は検証できなかった．提案手法はいくつかの点で優れているといえる．まず，提案手法はパラメタを含むモデルを扱うことができる．跳ね返るボールの例はパラメタを含むためにモデル検査器で検証できなかった (具体化すれば検証できる)．つぎに，提案手法はスケーラビリティが高い．[12] の実験では，PHAVer は高速道路の例を $n = 15$ までしか検証できていない.

KeYmaera を用いて多くのハイブリッドシステムを自動的に検証できるが，ハイブリッドオートマトンとしてモデル化したハイブリッドシステムに関しては検証がうまくいかないことが多い．水槽の例ではモデルにループ不変条件を付加する必要があった．自動検証がうまくいかない場合，ユーザは定理証明器を対話的に用いて演繹的な証明をおこなうこともできるが，KeYmaera は 141 の推論ルールを備え，使いこなすのは習熟を要する [16]．本稿の提案手法も全自動化されていないものの，検証戦略の数は限られており，また対話的作業を要するループ展開の試行やループ不変条件の導出を自動化することも可能だと考えている.

7 関連研究

ハイブリッドシステムのための論理的分析ツールが複数提案されている [13] [1] [7] [8] [2] [19]．これらのツールではハイブリッドシステムを既存の検証フレームワークに変換して扱う．いずれのツールにおいても検証プロセスが完全には自動化されておらず，ユーザが付加的な情報を入力したり，対話的な定理証明をおこなったりする必要がある.

Platzer らが開発した KeYmaera [16] [14] が近年，実用的な検証ツールとして注目されている．KeYmaera は独自のモデリング言語を備え，ハイブリッドシステムを hybrid program として記述し，differential (algebraic) dynamic logic で仕様や検証条件を付加する．また，検証には独自の定理証明器を用いる [14]．これに対して本研究では，標準的なハイブリッドオートマトンを対象とし，簡単な逐次言語と帰納法にもとづく限られた検証戦略からなる「軽量な」検証手法を提案している.

ハイブリッドシステムの多項式不変条件を生成する代数的手法 [18] [17] が提案されている．将来課題として，これらを提案手法に組み込むことが考えられる.

提案手法は有界モデル検査法との類似性がある．無限長の実行を扱う有界モデル検査法が提案されているものの (例 [4])，これらの手法は離散系に適用されるにとどまっている．多くのハイブリッドシステムの有界モデル検査器は有限長の実行に関してのみ検証をおこなっている．Hybrid SAL Relational Abstracter [20] はハイブリッドシステムを離散系に抽象化して有界モデル検査をおこなうが，提案手法では抽象化せず直接ハイブリッドシステムを扱う点が異なる.

8 まとめ

本稿では，多くの線形および非線形のハイブリッドオートマトンの無限長の実行に関する安全検証が可能な，論理的分析アプローチによるツールを提案した．提案手法では，多数の導出規則を用意して対話的な定理証明にもとづくツールを提供するのではなく，検証のための戦略を最強事後条件の計算と帰納法の適用とに絞り込み，自動的な検証プロセスの実現を目指した．実験結果では複数のベンチマーク問題を実用的な計算時間で検証可能なことを示した．提案手法では，一般のプログラム検証の場合と同様に不変条件を求める作業が必要となるが，証明に失敗した検証条件から Mathematica を用いて不変条件を生成する方法を示した．現状では不変条件生成においてユーザの対話的作業を必要としており，完全な自動化が今後の課題として挙げられる．

謝辞 本研究の一部は科研費 23-3810 の補助を得ておこなった．

参 考 文 献

[1] Ábrahám-Mumm, E., Steffen, M., and Hannemann, U.: Verification of hybrid systems: Formalization and proof rules in PVS, *ICECCS*, 2001, pp. 48–57.

[2] Abrial, J.-R., Su, W., and Zhu, H.: Formalizing Hybrid Systems with Event-B, *ABZ*, *LNCS* 7316, 2012, pp. 178–193.

[3] Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S.: The algorithmic analysis of hybrid systems, *Theoretical Computer Science*, Vol. 138, No. 1(1995), pp. 3–34.

[4] De Moura, L., Ruess, H., and Sorea, M.: Bounded model checking and induction: From refutation to verification, *CAV*, *LNCS* 2725, 2003, pp. 14–26.

[5] Dijkstra, E. W.: Guarded commands, nondeterminacy and formal derivation of programs, *Communications of the ACM*, Vol. 18, No. 8(1975), pp. 453–457.

[6] Frehse, G.: PHAVer: Algorithmic Verification of Hybrid Systems past HyTech, *International Journal on Software Tools for Technology Transfer (STTT)*, Vol. 10, No. 3(2008), pp. 263–279.

[7] Ghosh, R., Tiwari, A., and Tomlin, C.: Automated symbolic reachability analysis: with application to delta-notch signaling automata, *HSCC*, *LNCS* 2623, 2003, pp. 233–248.

[8] Hasuo, I. and Suenaga, K.: Exercises in Nonstandard Static Analysis of hybrid systems, *CAV*, *LNCS* 7358, 2012, pp. 462–478.

[9] Henzinger, T. A.: The Theory of Hybrid Automata, *Verification of Digital and Hybrid Systems (NATO ASI Series F: Computer and Systems Sciences)*, Vol. 170(2000), pp. 265–292.

[10] Henzinger, T. A., Ho, P.-H., and Wong-Toi, H.: HyTech: A model checker for hybrid systems, *STTT*, Vol. 1(1997), pp. 110–122.

[11] Hoare, C. A. R.: An Axiomatic Basis for Computer Programming, *Communications of the ACM*, Vol. 12, No. 10(1969), pp. 576–580 and 583.

[12] Jha, S. K., Krogh, B. H., Weimer, J. E., and Clarke, E. M.: Reachability for Linear Hybrid Automata Using Iterative Relaxation Abstraction, *HSCC*, *LNCS* 4416, 2007, pp. 287–300.

[13] Manna, Z. and Sipma, H.: Deductive Verification of Hybrid Systems Using STeP, *HSCC*, *LNCS* 1386, 1998, pp. 305–318.

[14] Platzer, A.: *Logical Analysis of Hybrid Systems*, Springer, 2010.

[15] Platzer, A.: Guide for KeYmaera Hybrid Systems Verification Tool, http://symbolaris.com/info/KeYmaera-guide.html, 2012.

[16] Platzer, A. and Quesel, J. D.: KeYmaera: A hybrid theorem prover for hybrid systems, *IJCAR*, *LNCS* 5195, 2008, pp. 171–178.

[17] Rodriguez-Carbonell, E. and Tiwari, A.: Generating Polynomial Invariants for Hybrid Systems, *HSCC*, *LNCS* 3414, 2005, pp. 590–605.

[18] Sankaranarayanan, S., Sipma, H., and Manna, Z.: Constructing invariants for hybrid systems, *HSCC*, *LNCS* 2993, 2004, pp. 539–554.

[19] Su, W., Abrial, J.-R., and Zhu, H.: Complementary Methodologies for Developing Hybrid Systems with Event-B, *ICFEM*, *LNCS* 7635, 2012, pp. 230–248.

[20] Tiwari, A.: HybridSAL relational abstracter, *CAV*, *LNCS* 7358, 2012, pp. 725–731.