抽象化と精密化による実時間モデル検査の改善

中島 一[†] 亀山幸義[†,††]

本研究の目的は，抽象化・精密化の手法を用いて，実時間モデル検査の状態数爆発問題を解決することにある．実時間モデル検査は，実時間システムの検証に用いられる手法で，そのシステムを時間オートマトンで記述する．実時間モデル検査における状態数爆発問題は，探索する状態空間が，実数値をとるクロック変数の数に対し指数的に大きくなり，実用的な時間内で検査が終了しないというものである．抽象化・精密化に基づくモデル検査法では，抽象化によりクロック変数をすべて省いた，初期抽象システムを構築し検査する．しかし，このような抽象化では検証結果として，間違った結果すなわち偽物の反例を得てしまうことがある．その場合，初期抽象システムから抽象化のレベルを低くしたシステムに作り替え，検証をやり直す．これを，正しい結果が得られるまで繰り返す．この手法の最大の課題は，得られた反例を基に，抽象化のレベルが高くかつ検証可能なシステムをどのように作るかということである．提案手法では，取り除いた変数のうち必要なものを復元する精密化を行う．また状態数を抑えるため，システム全体に変数を復元するのではなく，部分的に復元する．さらにクロック変数を復元せず不必要な遷移を除去する，もう 1 つの精密化を併用することで状態数の増加を抑える．本稿では，提案手法の適用例として時間オートマトンの到達可能性解析を試み，実験でその有効性の検証を行った．

Improvement on Real-time Model Checking Using Abstraction-Refinement

Hajime Nakajima[†] and Yukiyoshi Kameyama[†,††]

We address the state explosion problem in real-time model checking. Real-time model checking automatically verifies real-time systems described as timed automata. This method suffers from the state explosion problem: the size of the state space grows exponentially with the number of real-valued clock variables. We use abstraction-refinement for reducing the state space. Using this method, an initial abstract system, which is an over-approximation to behaviors of the timed automaton, is constructed by removing all clock variables, and then the abstract system is verified. This kind of conservative abstraction may lead to a false result, that is a spurious counterexample. In this case, the abstract system is refined on the basis of the counterexample and the refined system is verified. The process is repeated until a correct result is obtained. The difficulty in this method is how the system is refined. In our approach, we extract clock variables needed in the refinement. Additionally, we do not refine the whole system but a part of the system. Besides this refinement, we introduce another refinement removing transitions which are proved to never fire in the timed automaton. Our approach avoids the state explosion problem by combining these refinements. In this work, we apply our techniques to reachability analysis of timed automata.

1. はじめに

近年，情報処理システムの検証手法の 1 つとして，モデル検査法が注目されている．モデル検査法[3),5),10)]は，オートマトンなどによりモデル化されたシステムの状態を網羅的に探索することで，検証すべき性質に対しての自動的な証明を可能とする（これ以降では，モデル化したオートマトン自体のことをシステムと呼ぶ）．しかし，状態数が無限もしくは巨大なシステムに対しては，探索する状態空間が大きくなり，実用的な時間内で検査が終了しなくなってしまう．これを状態数爆発問題と呼ぶ．

状態数が無限となるものの 1 つに，時間オートマトンがある．時間オートマトン[2)]は実時間システムを記述するのに用いられ，時間を計るためのクロックと呼ばれる変数を持つ．クロック変数は非負の実数値をとるため，時間オートマトンの状態空間は無限の大きさ

† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†† 科学技術振興機構
Japan Science and Technology Agency

を持つ．このような時間オートマトンの検証に，モデル検査を適用したものを実時間モデル検査[1),9),11)]と呼ぶ．今日まで，実時間モデル検査の状態数爆発問題を回避する手法が数多く提案されているが，どれも十分な解決ではない．

我々はこの問題を解決するために，抽象化・精密化の手法を用いた実時間モデル検査についての研究を進めている．抽象化・精密化に基づくモデル検査[4),8)]は，次のように行われる．

(1) システムを高いレベルで抽象化し，これを初期抽象システムとする．
(2) 抽象システムでモデル検査を行い性質を満たせば，元のシステムでもその性質は満たされる．抽象システムで性質が満たされない場合，抽象システムにおける反例を検査する．元のシステムで対応する反例が見つかれば，元のシステムは性質を満たさない．
(3) 対応する反例が見つからない，つまり偽物の反例であった場合，精密化を行い抽象化のレベルをわずかに低くした，新しい抽象システムを構築し (2) に戻る．

本稿では抽象化・精密化の手法を用いた，時間オートマトンの到達可能性解析について述べる．到達可能性解析は実時間モデル検査の基本となるもので，時間オートマトンをゾーングラフと呼ばれる有限オートマトンに変換し，到達可能性についてモデル検査を行うものである．ゾーングラフは抽象化のレベルが非常に低い，時間オートマトンの抽象システムであり，クロック変数の増加にともないその状態数は指数的に増加する．我々は抽象化・精密化の手法を用い，検証に不必要なクロック変数を省いた，抽象化のレベルが高い抽象システムを構築する．

しかし，抽象化・精密化を用いたモデル検査の問題点は，抽象化のレベルが高く，かつ偽物の反例を生成しないような抽象システムを，精密化によってどのように作るかということである．我々は，この問題に対して 2 つの精密化を用いることで解決を試みる．1 つ目は，復元すべきクロック変数を抽出するものである．この精密化では，クロック変数を復元して新しい抽象システムを構築するとき，システム全体に復元するのではなく，部分的に復元する．クロック変数をシステム全体に復元すると，状態数が著しく増加することが予想できる．そこで，復元するクロック変数から影響を受けるシステムの一部分を特定し，その範囲に復元することで状態数の増加を抑える．2 つ目の精密化は，元のシステムではけっして実行されない余分な遷移を，抽象システムから除去するものである．クロック変数を復元する精密化は，先にも述べたとおり状態数を増加させる要因となる．また，クロック変数を復元することで，抽象システムを最初から作り直さなければならず，多くの時間を要する．遷移を除去する抽象化は，それまでの抽象システムから遷移を取り除くだけなので，最初から構築しなおす必要はなく，状態数も増加しない．この 2 つの精密化を組み合わせて，抽象化・精密化における問題の解決を試みる．

本稿の構成は，まず従来の手法である時間オートマトンの到達可能性解析について述べ，抽象システムと抽象化・精密化手続き，提案手法の 2 つの精密化，実験結果と関連研究との比較検討という順である．

2. 時間オートマトンの到達可能性解析

時間オートマトンの到達可能性解析について，文献 5) に基づいて説明を行う．

2.1 時間オートマトン

ここでは時間オートマトンの定義と，それが表す無限状態グラフについて述べる．まず，クロック変数の制約について定義を与える．

定義 2.1 （クロック制約）. X をクロック変数の集合としたとき，クロック制約の集合 $C^+(X)$ は $\{TRUE\} \cup C(X)$ で表される．ただし，$C(X)$ は次のようになる．

- $x \prec c \in C(X)$ および $c \prec x \in C(X)$．ただし，$x \in X$，c は非負の整数，$\prec$ は $\leq$ または $<$．
- $\phi_1, \phi_2 \in C(X)$ のとき，$\phi_1 \wedge \phi_2 \in C(X)$．

このとき，時間オートマトンは以下のように定義できる．

定義 2.2 （時間オートマトン）. 時間オートマトンは $A = (L, L_0, \Sigma, X, I, T)$ で表される．

- L はロケーションの有限集合．
- $L_0 \subseteq L$ は初期ロケーションの集合．
- Σ はアルファベットの有限集合．
- X はクロック変数の有限集合．
- $I : L \rightarrow C^+(X)$ はロケーションからクロック制約へのマッピング（ロケーションインバリアント）．
- $T \subseteq L \times \Sigma \times C^+(X) \times 2^X \times L$ は遷移の集合．

A において遷移 $\langle l, a, \psi, \lambda, l' \rangle \in T$ を実行する際，時間は経過せず一瞬で遷移する．このとき各クロック変数の値は，クロック制約 ψ を満たさなければならない（ψ をガードと呼ぶ）．また，クロック変数の集合 λ に含まれるクロック変数は，0 にリセットされる．遷移せずにロケーション l でとどまっていると，時間が経過し，すべてのクロック変数の値が同じ割合で増

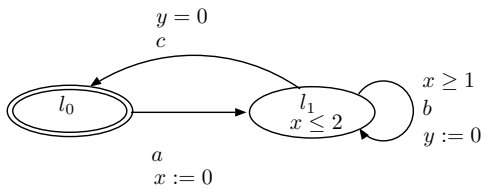

図 1 時間オートマトン
Fig. 1 Timed automaton.

加する．各クロック変数の値が $I(l)$ を満たしている間，時間を経過させることができる．$I(l)$ を満たさなくなると l にとどまることができず，それまでには実行可能な遷移を必ず行うことになる．**図 1** に時間オートマトンの例を示す．この例で $(L, L_0, \Sigma, X, I, T)$ は以下のようになる．

- $L = \{l_0, l_1\}$
- $L_0 = \{l_0\}$
- $\Sigma = \{a, b, c\}$
- $X = \{x, y\}$
- ロケーションインバリアント I
  $I(l_0) = TRUE$
  $I(l_1) = x \leq 2$
- 遷移 T
  $T = \{\ \langle l_0, a, TRUE, \{x\}, l_1\rangle,$
  $\langle l_1, b, 1 \leq x, \{y\}, l_1\rangle,$
  $\langle l_1, c, y = 0, \emptyset, l_0\rangle\}$

このように定義される時間オートマトン A は，次のような無限状態グラフを表現している．

定義 2.3 （無限状態グラフ）. 時間オートマトンの無限状態グラフ表現は $\mathcal{T}(A) = (Q, Q_0, R)$ で与えられる．

- 状態 $Q = (l, \nu)$. ただし $l \in L$. ν は各クロックの非負の実数値への割当て．つまり $\nu : X \to R^+$.
- $Q_0 = \{(l, \nu) \mid l \in L_0 \wedge \forall x \in X\ .\ \nu(x) = 0\}$ は初期状態の集合．
- 遷移関係 R は，2 つの遷移を組み合わせたものである．これらを説明する前に，各クロック変数への値の割当て ν に対する 2 つの操作を導入する．1 つ目は，$d \in R^+$ に対する $\nu + d$ で，$(\nu + d)(x) = \nu(x) + d$ という割当てを意味する．2 つ目は，$\nu[\lambda := 0]$ で，$x \in \lambda$ となる x に対しては 0 を，$x \notin \lambda$ となる x に対しては $\nu(x)$ とする割当てである．これらを用いて，2 つの遷移を説明する．
  – ディレイ遷移. 同じロケーションでとどまっているときの時間の経過に相当し，$(l, \nu) \xrightarrow{d} (l, \nu + d)$ で表す．このとき，$d \in R^+$ であり $0 \leq e \leq d$ となる $\nu + e$ は，$I(l)$ を満たさなければならない．
  – アクション遷移. T の遷移に相当し，$(l, \nu) \xrightarrow{a} (l', \nu')$ で表す．このとき $\langle l, a, \psi, \lambda, l'\rangle \in T$ となるものがある．さらに ν は ψ を満たし，$\nu' = \nu[\lambda := 0]$ でなければならない．

  遷移関係 R は，このディレイ遷移とアクション遷移を組み合わせたものである．状態 (l, ν) と状態 (l', ν') に遷移関係があるとき，$(l, \nu) \xrightarrow{d} (l'', \nu'') \xrightarrow{a} (l', \nu')$ である (l'', ν'') が存在しなければならない．この遷移関係を $(l, \nu)R(l', \nu')$ もしくは，$(l, \nu) \stackrel{a}{\Rightarrow} (l', \nu')$ と書く．

2.2 ゾーングラフ

ここでは，時間オートマトンの到達可能性解析に用いられるゾーングラフについて説明する．ゾーンとは，クロック変数についての不等式を論理記号 $\wedge$ で結んだ論理式である．これ以後ゾーンのことを，その論理式を満たすクロック変数の値の割当ての集合と見なすことにする．以下にゾーンの定義を与える．

定義 2.4 （ゾーンの定義）. クロック変数の集合 $X = \{x_1, x_2 \ldots x_n\}$ に対して，ゾーンは以下のように表される．

$$x_0 = 0 \wedge \bigwedge_{0 \leq i \neq j \leq n} x_i - x_j \prec_{i,j} c_{i,j}$$

ただし x_0 はつねに 0 である特別なクロック変数，$c_{i,j}$ は整数，$\prec_{i,j}$ は $\leq$ または $<$ である．

さらに，ゾーンに対して 3 つの操作を導入する．

INTERSECTION 2 つのゾーン φ_1，φ_2 に対して $\varphi_1 \wedge \varphi_2$ は，以下の集合を表すゾーンとなる．

$$\varphi_1 \wedge \varphi_2 = \{\nu \mid \nu \in \varphi_1 \wedge \nu \in \varphi_2\}$$

CLOCK RESET ゾーン φ とクロック変数の集合 λ に対して，$\varphi[\lambda := 0]$ は以下の集合を表すゾーンとなる．

$$\varphi[\lambda := 0] = \{\nu \mid \exists \nu' \in \varphi\ .\ \nu = \nu'[\lambda := 0]\}$$

ELAPSING OF TIME ゾーン φ に対して，$\varphi^{\Uparrow}$ は以下の集合を表すゾーンとなる．

$$\varphi^{\Uparrow} = \{\nu \mid \exists \nu' \in \varphi\ \exists d \in R^+\ .\ \nu = \nu' + d\}$$

図 2 は，これらの操作によるゾーンの変化を図示したものである．x，y をクロック変数としたとき，ゾーン $x \leq 2 \wedge y \leq 1$ を満たすクロック変数の割当ては，(a) の図で示される領域となる．(b) はこのゾーンに ELAPSING OF TIME の操作を，(c) は y をリセットしたものである．

ゾーンを用いることにより時間オートマトンにおける到達可能性問題を，以下に定めるゾーングラフ上の到達可能性問題に帰着できる．このゾーングラフ上で，

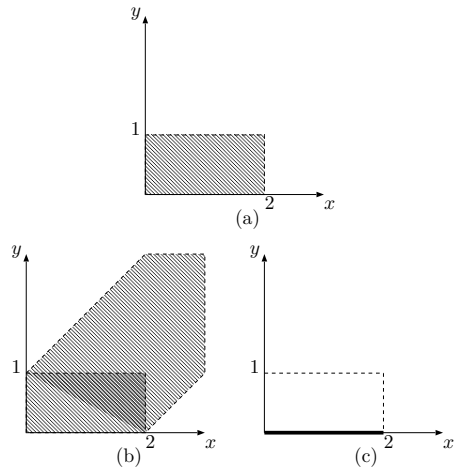

図 2 ゾーンの操作
Fig. 2 Operations on zones.

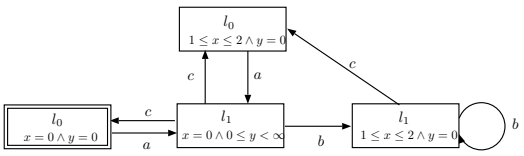

図 3 ゾーングラフ
Fig. 3 Zone graph.

ある特定のロケーションに到達可能であることをモデル検査する．

定義 2.5 （ゾーングラフの定義）. ゾーングラフは有限オートマトンである．時間オートマトン A に対するゾーングラフを $\mathcal{Z}(A)$ と書き，以下のように定義する．

- 状態はロケーション l とゾーン φ の対からなる．つまり (l, φ).
- 初期状態は，
$$\{(l, \varphi) \mid l \in L_0,\ \varphi \text{ は } \forall x_i \in X\ .\ x_i = 0\}$$
- 状態 (l, φ) から A の遷移 $t = \langle l, a, \psi, \lambda, l' \rangle \in T$ のもとでの，次の状態は $(l', succ(t, \varphi))$ で得られる．ただし，$succ$ は以下のように表される．
$$succ(t, \varphi) = ((\varphi \wedge I(l))^{\Uparrow} \wedge I(l) \wedge \psi)[\lambda := 0]$$

$succ(t, \varphi)$ が空集合となる場合は，(l, φ) から t は実行できない．

このようにゾーングラフの状態は，与えられた初期状態から順次 $succ$ を計算することにより求まる，到達可能な状態である．

図 1 の時間オートマトンからゾーングラフを構築したものを**図 3** に示す．

ゾーングラフの状態はロケーションとゾーンの対で表現される．ロケーションはシステムのコンポーネント数に対して指数的に増加し，さらにゾーンもクロック変数の数に対して指数的に増加する．したがって，ゾーングラフを用いた到達可能性解析の状態数爆発問題を解決するためには，この 2 つの要因による状態数の増加を抑えなければならない．本稿では，ゾーンの増加を抑えてモデル検査を行う手法について議論を進め，ロケーションの増加による状態数爆発問題は考慮しない．これは，以下の理由による．ロケーションの増加による状態数爆発問題は，状態変数の増加による状態数爆発問題と同じものであり，通常のモデル検査で提案されている手法を，そのまま用いることができると考えている．しかし，ゾーンの増加は時間オートマトン特有のクロック変数によるものであり，通常のモデル検査における手法を適用しにくい．そこで，我々は抽象化・精密化を用いて，ゾーンの増加を抑える手法について提案する．ゾーンを用いた実時間モデル検査の改善手法としては，ゾーンのデータ表現形式についての研究がある．計算機上でゾーングラフを構成するためには，ゾーンの演算を効率良く行え，メモリの消費が少ないデータ構造が必要である．データ構造は様々なものが提案されている[7),13),14)]が，本稿の提案手法はこれらゾーンのデータ構造に依存せず，独立なものである．これ以降では，抽象化・精密化について説明する．

3. 抽象システム

モデル検査法において，状態数削減のための重要な手法の 1 つが抽象化である．抽象化は元のシステムを，ある種の情報が省略された抽象システムに置き換えるものである．抽象システムは元のシステムの情報を省略したものなので，その情報に関する性質は検証できなくなるが，状態数は元のものより少なくなる．

本稿の検証性質はロケーションへの到達可能性であるため，クロック変数が持つ値には興味がない．もちろん，特定のロケーションへの到達可能性を調べるために必要となるクロック変数は存在するが，すべてのクロック変数が必要である場合は希である．そこで，クロック変数を省略する抽象システムを考える．ロケーション l において省略しないクロック変数の集合を $V(l)$ とすると，抽象システムは以下のように定義できる．

定義 3.1 （抽象システムの定義）. 時間オートマトンを A とし，マッピング $V : L \to 2^X$ に対して，抽象システム $\mathcal{M}(A, V)$ を以下のように定義する．ただし $X = \{x_1, x_2, \ldots, x_n\}$ とする．

- 状態 (l, Φ). ただし $l \in L$. Φ は以下のようなゾー

ンである．

$$x_0 = 0 \wedge \bigwedge_{\substack{0 \le i \ne j \le n \\ x_i, x_j \in V(l)}} x_i - x_j \prec_{i,j} c_{i,j}$$

- 初期状態は $\{(l, \Phi) \mid l \in L_0 \ , \ \Phi$ は $\forall x_i \in V(l) \,.\, x_i = 0\}$.
- 状態 (l, Φ) から A の遷移 $t = \langle l, a, \psi, \lambda, l' \rangle$ のもとでの，次の状態は $(l', SUCC(t, \Phi))$ で得られる．ただし，$SUCC(t, \Phi) = \{\nu \mid \exists \nu' \in succ(t, \Phi)\ \forall x \in V(l) \,.\, \nu(x) = \nu'(x)\}$.

ここで，$SUCC(t, \Phi) = \{\nu | \exists \nu' \in succ'(t, \Phi)\ \forall x \in V(l) \,.\, \nu(x) = \nu'(x)\}$ ではないことに注意されたい．ただし，

$$\begin{aligned} &succ'(t, \Phi) \\ &= ((\Phi \wedge I'(l))^{\Uparrow} \wedge I'(l) \wedge \psi')[\lambda' := 0] \\ &I'(l) \\ &= \{\nu \mid \exists \nu' \in I(l)\ \forall x \in V(l) \,.\, \nu(x) = \nu'(x)\} \\ &\psi' \\ &= \{\nu \mid \exists \nu' \in \psi\ \forall x \in V(l) \,.\, \nu(x) = \nu'(x)\} \\ &\lambda' \\ &= \lambda - V(l) \end{aligned}$$

とする．定義のようにすることで，状態数を抑えつつ，ゾーングラフには存在しない実行系列を抽象システムから除くことができる．4.4 節では実例でこれを説明する．

このような抽象システムの正しさ，いい換えると，この抽象システムで到達可能性について論じることができるかということは，シミュレーション関係から導くことが可能である．

3.1 シミュレーション関係

2 つの有限オートマトンを $G = (S, I, E, AP, \mathcal{L})$, $G' = (S', I', E', AP, \mathcal{L}')$ とする．S は状態の集合で，$I \subseteq S$ は初期状態の集合，$E \subseteq S \times S$ は遷移関係の集合，AP は命題定数の集合，$\mathcal{L}$ は S から AP へのマッピング（また $\mathcal{L}' : S' \to AP$ とする）である．このとき，シミュレーション関係を以下のように定義する．

定義 3.2 （シミュレーション関係）. 以下の条件を満たす関係 $H \subset S \times S'$ があるとき，G' は G をシミュレート（模倣）する，あるいは G と G' はシミュレーション関係にあるといい，$G \preceq G'$ と書く．

(1) $\forall i \in I\ \exists i \in I' \,.\, H(i, i')$

(2) $\forall s \in S\ \forall s' \in S' \,.\, H(s, s') \Rightarrow \mathcal{L}(s) = \mathcal{L}'(s')$

(3) $\forall s, s_1 \in S\ \forall s' \in S' \,.\, (H(s, s') \wedge E(s, s_1) \Rightarrow \exists s'_1 \in S' \,.\, (H(s_1, s'_1) \wedge E(s', s'_1)))$

システム G が性質 f を満たすということを $G \models f$ と書き，$G \preceq G'$ のとき，以下の性質が成立する．証明に関しては，文献 5) を参照されたい．

定理 1. $G \preceq G'$，時相論理式 $f = AGp$ のとき，

$$G' \models f \Rightarrow G \models f$$

ただし，p は時相オペレータ（A, E, G, F）を含まない命題論理式である．また，この論理式を構成する原子命題は AP の要素とする．

簡単に時相オペレータに関して説明すると，以下のような意味になる．

Ap 任意の実行系列に対して p が成立する．

Ep ある実行系列に対して p が成立する．

Gp 実行系列中のすべての状態において p が成立する．

Fp 実行系列中のある状態において p が成立する．

したがって，AGp はすべての実行系列中の各状態において p が成立する，という意味になる．

提案手法の抽象システム $\mathcal{M}(A, V)$ の正しさを示すために，まず，ゾーングラフ $\mathcal{Z}(A)$ と $\mathcal{M}(A, V)$ がシミュレーション関係にあることを示す．

補題 1. $\mathcal{Z}(A) \preceq \mathcal{M}(A, V)$

証明. $\mathcal{Z}(A)$ の状態の集合を $Z = (l_1, \varphi)$ とし，$\mathcal{M}(A, V)$ の状態の集合を $M = (l_2, \Phi)$ とする．ここで，$AP = \{l \mid l$ はロケーション $\}$ とすれば，$\mathcal{L}(Z) = l_1$，$\mathcal{L}'(M) = l_2$ となる．このとき，$Z \times M$ 上の関係 H は，以下のように定義できる．

$$\begin{aligned} &(l_1, \varphi)\ H\ (l_2, \Phi) \\ &\Leftrightarrow (l_1 = l_2) \wedge (\varphi \subset \Phi) \end{aligned}$$

Z における初期状態を $z_0 = (l_0, \varphi_0)$ とすると，M における初期状態 $m_0 = (l_0, \Phi_0)$ は，任意の $\nu \in \varphi_0$ に対して $\nu \in \Phi_0$ なので，$H(z_0, m_0)$ となる．

$H(z, m)$ であるような任意の $z = (l, \varphi) \in Z$, $m = (l, \Phi) \in M$ をとる．このとき，z から遷移 $t = \langle l, a, \psi, \lambda, l' \rangle \in T$ が実行可能であると仮定すると，遷移後の状態は $z' = (l', succ(t, \varphi))$ となる．m から遷移 $t = \langle l, a, \psi, \lambda, l' \rangle \in T$ を実行させたとすると，遷移後は $m' = (l', SUCC(t, \Phi))$ となる．$H(z, m)$ より，任意の $\nu \in \varphi$ に対して $\nu \in \Phi$ であることから，任意の $\nu \in succ(t, \varphi)$ に対して $\nu \in SUCC(t, \Phi)$ がいえればよい．INTERSECTION，CLOCK RESET，ELAPSING OF TIME の操作は，ゾーンの包含関係に関して単調であるので，任意の $\nu \in succ(t, \varphi)$ に対して $\nu \in SUCC(t, \Phi)$ が導ける．

したがって $\mathcal{Z}(A)$ のすべての遷移に対して，$\mathcal{M}(A, V)$ で対応する遷移が存在する． □

本稿の到達可能性は，時相論理式 EFl の充足可能性

として表現される（ただし $l \in L$）．これは，$\neg AG\neg l$ という時相論理式に変換可能なので，以降，検証性質は到達可能性ではなく，$AG\neg l$ つまり「いかなる可能性を鑑みてもつねに $\neg l$ である」という安全性に関するものとする．定理 1 より次の性質が成り立つ．

定理 2. $\mathcal{Z}(A) \preceq \mathcal{M}(A,V)$，$f = AG\neg l (= \neg EFl)$ のとき，

$$\mathcal{M}(A,V) \models f \Rightarrow \mathcal{Z}(A) \models f$$

ただし $l \in L$ であり，ここでは「l にいる」という命題として用いる．

3.2 抽象化・精密化

以上のような抽象システム $\mathcal{M}(A,V)$ を用いて，抽象化・精密化に基づくモデル検査手続きの概要を与える．

抽象化・精密化によるモデル検査手続き

検証性質を $f = AG\neg l$（$l \in L$）とするとき，抽象化・精密化によるモデル検査手続きは以下のようになる．

(1) 時間オートマトン A，$\forall l \in L\ .\ V(l) = \emptyset$ とし，初期抽象システム $\mathcal{M}(A,V)$ を作る．

(2) $\mathcal{M}(A,V) \models f$ をモデル検査で調べる．もし f が満たされるなら，A でも満たされる．

(3) f が満たされないとき，モデル検査により反例 ρ が得られる．その反例が本物か検査する．本物であれば A は f を満たさない．

(4) 反例が偽物の場合，精密化を行う．本手法では以下の 2 つの精密化を導入する．

(a) 遷移を除去する精密化を行い，(2) へ．

(b) クロック変数を復元する精密化を行い，(2) へ．

この手続きに従うと，$\mathcal{M}(A,V) \models f$ が得られれば，定理 2 から $\mathcal{Z}(A) \models f$ を導くことができ，検証は終了である．$\mathcal{M}(A,V) \not\models f$ の場合は，定理から $\mathcal{Z}(A) \not\models f$ を直接導くことができないため，(3) のように反例の検査を行う．次節は反例の検査について述べる．

3.3 反例の検査

検証性質は $f = AG\neg l$ なので，$\mathcal{M}(A,V) \not\models f$ であるときの反例 ρ は，次のような有限の長さの状態と遷移の列で表される．

$$\rho = \langle m_0 \overset{r_0}{\Rightarrow} m_1 \overset{r_1}{\Rightarrow}, \ldots, \overset{r_{n-1}}{\Rightarrow} m_n \rangle$$

反例 ρ の検査とは，抽象システム $\mathcal{M}(A,V)$ 上の実行系列 ρ と対応する実行系列の存在の有無を，ゾーングラフ $\mathcal{Z}(A)$ で調べることである．もし対応する実行系列が $\mathcal{Z}(A)$ に存在すれば，ρ は本物ということであり，$\mathcal{Z}(A) \not\models f$ である．反例を検査するための時間オートマトン A_ρ を定める．A_ρ は，時間オートマトン A から ρ と対応する部分を切り出したような，部分的な時間オートマトンである．

定義 3.3 （反例検査時間オートマトン）. 反例を検査するための時間オートマトンは，反例 ρ に対して，$A_\rho = (L_\rho, L_{\rho,0}, \Sigma_\rho, X_\rho, I_\rho, T_\rho)$ と定義される．ただし，反例 ρ に現れる状態は $m_i = (l_i, \Phi_i)$ とする．

- $L_\rho = \{l_0, l_1, \ldots, l_n\}$.
- $L_{\rho,0} = l_0$.
- $I_\rho : L_\rho \to C^+(X)$．ただし，$l \in L_\rho$ に対して $I_\rho(l) = I(l)$.
- T_ρ は T において r_i と対応する遷移の集合．
- $\Sigma_\rho = \{a \mid \langle l, a, \psi, \lambda, l' \rangle \in T_\rho\}$.
- $X_\rho \subseteq X$．ただし X_ρ は，I_ρ，T_ρ において用いられているクロック変数の集合．

このような時間オートマトン A_ρ からゾーングラフ $\mathcal{Z}(A_\rho)$ を構築し，l_n に到達可能かをモデル検査することで，反例 ρ が本物であるか判定できる．このとき，$\mathcal{Z}(A_\rho)$ の状態数が多くなると検査できなくなるが，ここでの A_ρ の遷移は決定的なものであるため，$\mathcal{Z}(A_\rho)$ の状態爆発問題は考慮する必要はない．

l_n に到達不可能であった場合，ρ は偽物の反例であり精密化を行う必要がある．次章では，提案手法である 2 つの精密化について述べる．

4. 精 密 化

偽物の反例のような実行系列が，抽象システムに存在してしまうのは，抽象化のレベルが高すぎるためである．そこで精密化は，抽象化のレベルをわずかに低くした抽象システムを構築し，偽物の反例を出さないようにする．ここでの抽象システムはクロック変数を省略したものなので，抽象化のレベルを低くするためには，クロック変数を復元した新しいマッピング V' を決定し，$\mathcal{M}(A,V')$ を構築することになる．しかしこのような精密化を行う場合，いくつか考慮すべき点がある．

まず，新たに構築される抽象システムの状態数は，元の抽象システムと比較すると非常に多いということがいえる．つまり，精密化でクロック変数を復元していくと，抽象システムの状態数も指数的に増加していく．また，$\mathcal{M}(A,V')$ は $\mathcal{M}(A,V)$ を利用して構築することができず，時間オートマトン A より最初から作り直すしかない．

これらのことから，3.2 節，抽象化・精密化によるモデル検査手続きで述べた，遷移を除去する精密化を

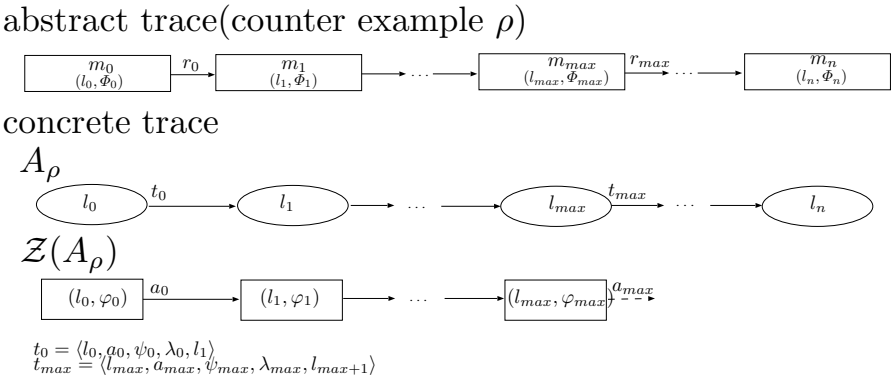

図 4 ρ，A_ρ と $\mathcal{Z}(A_\rho)$ の関係
Fig. 4 ρ,A_ρ and $\mathcal{Z}(A_\rho)$.

導入する．以下，遷移を除去する精密化と，クロック変数を復元する精密化について順に述べる．

精密化を行う必要があるときは，3.3 節の反例 ρ を検査した結果，l_n に到達不可能のときである．このとき到達可能である l_i（ただし $0 \leq i < n$）で i が最大のものを l_{max} とする．**図 4** に ρ，A_ρ と $\mathcal{Z}(A_\rho)$ の対応関係を示す．つまり，反例 ρ のような実行系列では，$\mathcal{M}(A,V)$ の状態 m_{max} と対応する $\mathcal{Z}(A)$ の状態 (l_{max},φ_{max}) から，$\mathcal{M}(A,V)$ の遷移 r_{max} と対応する遷移 t_{max} が実行できないということである．これらのもとで説明を進める．

4.1 遷移の除去

m_{max} と対応する $\mathcal{Z}(A)$ 上の状態 (l_{max},φ_{max}) から遷移 t_{max} が実行できないことは，反例 ρ の検査から分かる．遷移を除去する精密化は，m_{max} と対応する $\mathcal{Z}(A)$ のすべての状態で遷移 t_{max} が実行できなければ，$\mathcal{M}(A,V)$ から遷移 r_{max} を取り除いてもよい，という考えに基づく．m_{max} と対応する $\mathcal{Z}(A)$ のすべての状態で遷移 t_{max} が実行できるかを厳密に調べるためには，$\mathcal{Z}(A)$ を構築しモデル検査しなければならず，状態数爆発問題が発生してしまう．そこで，m_{max} と対応する $\mathcal{Z}(A)$ における状態の上への近似となるような状態を持つゾーングラフを構築し，モデル検査する．このようなゾーングラフは，時間オートマトン A の部分グラフ $A_{Srv(l)}$ のゾーングラフ $\mathcal{Z}(A_{Srv(l)})$ として与えられる．

Srv は，$L \to X$ となるマッピングである．$\phi \in C^+(X)$ に対して，$clk(\phi)$ をクロック制約 ϕ に出現するクロック変数の集合とし，以下に Srv の定義を与える．

定義 4.1 （マッピング Srv）.
遷移 $t_{max} = \langle l_{max}, a_{max}, \psi_{max}, \lambda_{max}, l_{max+1}\rangle$ として，すべての $l \in L$ で以下の計算を行い，収束した結果を Srv とする．

$$
\begin{aligned}
&Srv_0(l)\\
&:= \begin{cases} clk(\psi_{max}) \cup clk(I(l)) & l = l_{max}\text{のとき} \\ \emptyset & \text{それ以外} \end{cases}\\
&Srv_{n+1}(l)\\
&:= Srv_n(l) \cup \bigcup_{\langle l,a,\psi,\lambda,l'\rangle \in T} (Srv_n(l') - \lambda)
\end{aligned}
$$

(l_{max},φ_{max}) から遷移 t_{max} が実行できないということは，$succ(t_{max},\varphi_{max})$ が空集合であるということである．ゾーンの INTERSECTION，CLOCK RESET，ELAPSING OF TIME の操作で，空集合を作り出すことが可能な操作は INTERSECTION のみである．$succ(t_{max},\varphi_{max}) = ((\varphi_{max} \wedge I(l_{max}))^{\Uparrow} \wedge I(l_{max}) \wedge \psi_{max})[\lambda_{max} := 0]$ なので，$\varphi_{max} \wedge I(l_{max})$ もしくは，$(\varphi_{max} \wedge I(l_{max}))^{\Uparrow} \wedge I(l_{max}) \wedge \psi_{max}$ が空集合である．つまり，l_{max} において t_{max} が実行できるかどうかは，ψ_{max} と $I(l_{max})$ に出現するクロック変数の値のみで判定できる．Srv は，l_{max} におけるそれらのクロック変数の値が，影響を受ける範囲を表している．$Srv(l) \neq \emptyset$ のときは，l から l_{max} への実行系列で ψ_{max} と $I(l_{max})$ に出現するクロック変数のいずれかの値を，一度もリセットしない実行系列が存在するということである．逆に $Srv(l) = \emptyset$ のときは，l_{max} にたどり着くまでに必ずリセットされる．

$Srv(l) \neq \emptyset$ の l からなる A の部分時間オートマトンで，m_{max} と対応する状態に到達し遷移 t_{max} が実行できるかを調べれば，r_{max} を除去できるかの判定には十分である．以下に A の部分時間オートマトン $A_{Srv(l)}$ を定義する．

定義 4.2 （遷移検査時間オートマトン）. $A_{Srv(l)}$ は時間オートマトン A と $Srv(l)$ から以下のように求められる．

- $Srv(l) = \emptyset$ に対して，$\langle l', a, \psi, \lambda, l\rangle \in T$ となる遷移を取り除く．
- 遷移を取り除いたことで，遷移が不可能になるロケーションをすべて取り除く．

このように構築される $A_{Srv(l)}$ からゾーングラフ $\mathcal{Z}(A_{Srv(l)})$ を構築し，(l_{max}, φ) に到達でき（ただし $\varphi \subset \Phi_{max}$ を満たす），t_{max} が実行可能かを検査する．ただし，$\mathcal{M}(A, V)$ の状態を $m = (l, \Phi)$ とすると，$Srv(l) = \emptyset$ であり，l が $A_{Srv(l)}$ のロケーションであるような m を，$\mathcal{Z}(A_{Srv(l)})$ の初期状態とする．

t_{max} が実行不可能であった場合，$\mathcal{M}(A, V)$ から遷移 r_{max} を取り除いた $\mathcal{M}(A, V)$ で $\mathcal{M}(A, V) \models f$ を検査する．実行可能であった場合，クロックを復元する精密化を行う．

遷移を除去する精密化の最大の特徴は，$\mathcal{M}(A, V)$ から遷移を取り去るだけなので，簡単に新しい抽象システムが構築でき，状態数も増加しないことである．実際の時間オートマトンでは，遷移として定義はされているが，ガードなどのクロック変数の制約により絶対に実行できないようなものが多数存在する．この精密化は，そのような遷移を検出し抽象システムから除去することが可能なため，状態数を抑えるのに有効である．次節では，遷移を除去する精密化の適用例を示す．

4.2 例 1

ここでは，遷移を除去する精密化の適用例として，鉄道の踏切システムに対する検証例を示す．ここでの鉄道踏切システムは，列車・踏切・制御器の 3 つのサブシステムからなるものを考え，列車が x，踏切が y，制御器が z というクロック変数を持っている．簡略化のために，列車は 1 台であると仮定する．このようなシステムを**図 5** に示す．検証条件は，「いつでも，列車が in のとき踏切は down している」とする．

このシステムの同期積オートマトンは**図 6** であり，これを検証対象システム A とする．A において検証条件を調べるためには，A におけるロケーション $(\mathtt{in}, \mathtt{up}, 1)$, $(\mathtt{in}, \mathtt{coming\ down}, 0)$, $(\mathtt{in}, \mathtt{coming\ up}, 1)$ などに到達可能かを調べればよい（到達可能であれば検証条件を満たさず，到達不可能であれば検証条件を満たすことになる）．

まず，$\forall l \in L\,.\,V(l) = \emptyset$ とした初期の抽象システム $\mathcal{M}(A, V)$ を求める．初期抽象システムではクロック変数をすべて省くので，A のロケーションがそのまま $\mathcal{M}(A, V)$ の状態である．これに対してモデル検査

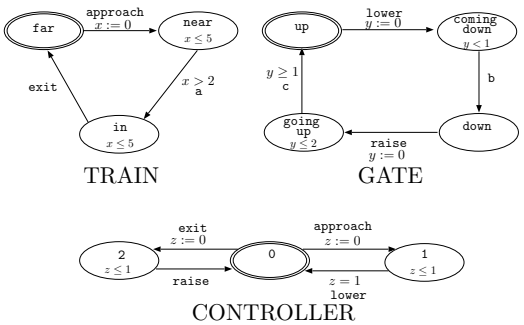

図 5 鉄道踏切システム
Fig. 5 Train crossing example.

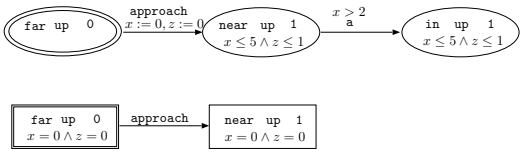

図 7 反例の検査
Fig. 7 Checking the counterexapmle.

表 1 $Srv(l)$
Table 1 $Srv(l)$.

l	$Srv_0(l)$	$Srv_1(l)$
$(\mathtt{near}, \mathtt{up}, 1)$	$\{x, z\}$	$\{x, z\}$
$(\mathtt{near}, \mathtt{going\ up}, 1)$	$\emptyset$	$\{x, z\}$

を行い，たとえば次のような反例 ρ を得ることができる．

$$(\mathtt{far}, \mathtt{up}, 0) \xrightarrow{\mathtt{approach}} (\mathtt{near}, \mathtt{up}, 1) \xrightarrow{\mathtt{a}} (\mathtt{in}, \mathtt{up}, 1)$$

この反例が本物であれば検証は終了だが，偽物の場合，精密化を行う必要がある．そこで反例を検査するために，A_ρ を作りゾーングラフ $\mathcal{Z}(A_\rho)$ でモデル検査を行う．すると，$\mathcal{Z}(A_\rho)$ における状態 $((\mathtt{near}, \mathtt{up}, 1), x = z = 0)$ から遷移 a が実行できず（**図 7**），反例が偽物であるということが分かり，精密化を行う必要がある．

遷移を除去する精密化では，まず $Srv(l)$ を求める．$Srv(l)$ の計算過程を**表 1** に示す．この表は，$Srv_i(l)$ を $i = 0, 1$，$l = (\mathtt{near}, \mathtt{up}, 1), (\mathtt{near}, \mathtt{going\ up}, 1)$ に対して求めたものである．$i = 1$ で $Srv_i(l)$ は収束し，その他のロケーションに対する $Srv_i(l)$ はつねに $\emptyset$ なので省いた．結果，l_1，l_2 を $(\mathtt{near}, \mathtt{up}, 1)$, $(\mathtt{near}, \mathtt{going\ up}, 1)$ とすれば，$Srv(l)$ は以下のようになる．

$$Srv(l) = \begin{cases} \{x, z\} & l = l_1, l_2 \text{のとき} \\ \emptyset & \text{それ以外} \end{cases}$$

この $Srv(l)$ を基に，$\mathcal{M}(A, V)$ の状態 $(\mathtt{near}, \mathtt{up}, 1)$ から遷移 a が実行可能かを調べる検査オートマト

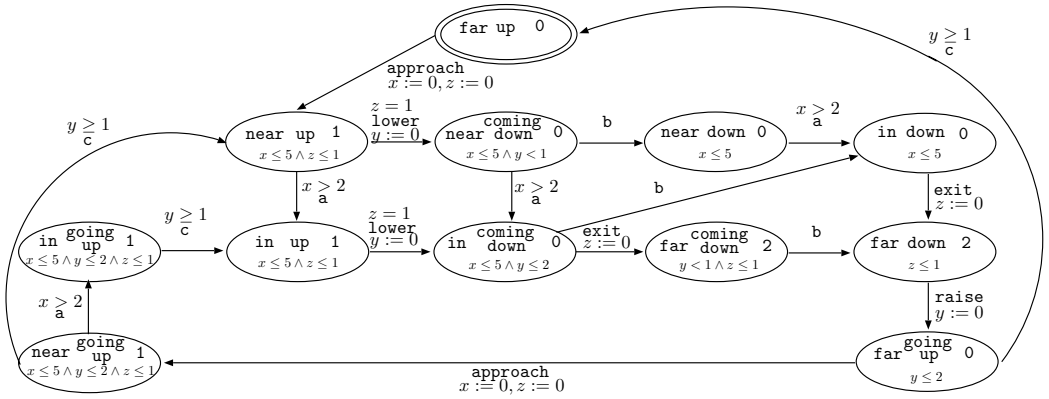

図 6 同期積オートマトン
Fig. 6 Product timed automaton.

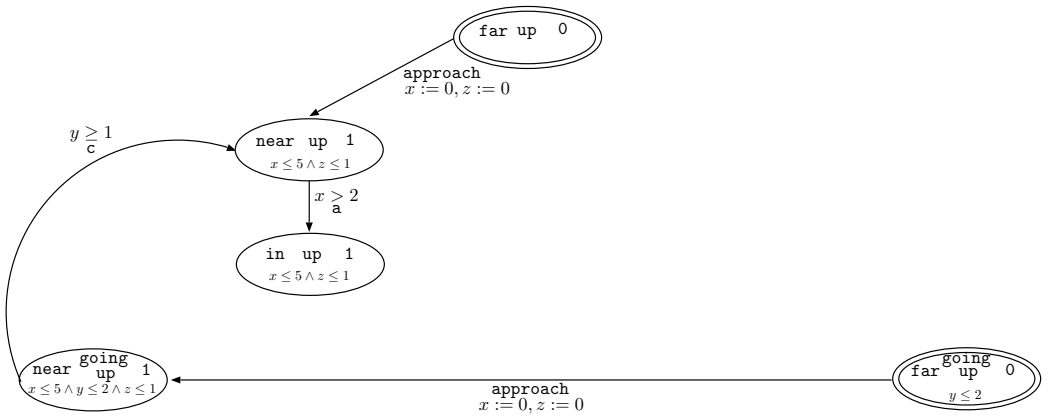

図 8 $A_{Srv(l)}$
Fig. 8 $A_{Srv(l)}$.

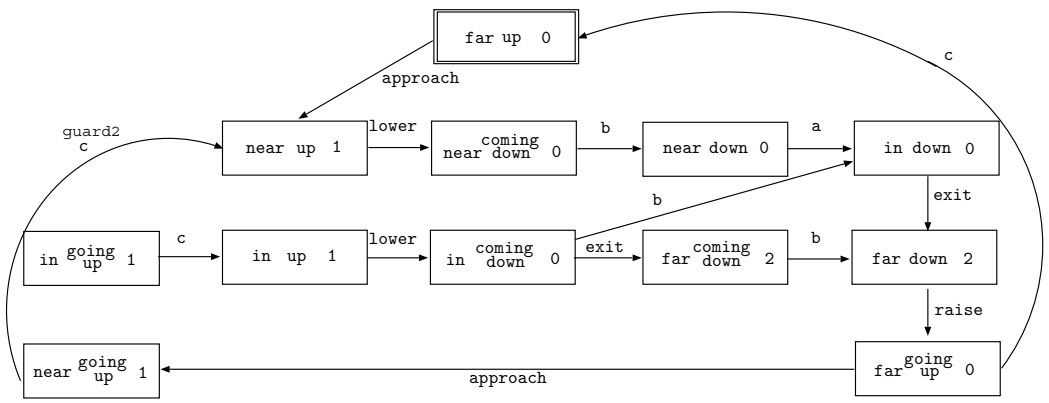

図 9 遷移を除去する精密化により得られる $\mathcal{M}(A, V)$
Fig. 9 $\mathcal{M}(A, V)$ obtained by the transition elimination refinement.

ン $A_{Srv(l)}$ を構築すると，**図 8** のようになる．この $A_{Srv(l)}$ から初期状態を $((\texttt{far}, \texttt{up}, 0), 0 \leq x, y, z < \infty)$，$((\texttt{far}, \texttt{going up}, 0), 0 \leq x, y, z < \infty)$ とした $\mathcal{Z}(A_{Srv(l)})$ でモデル検査を行うと，$(\texttt{near}, \texttt{up}, 1)$ から遷移 a が実行不可能であるということが分かり，この a を $\mathcal{M}(A, V)$ から取り除く．このように a が取り除かれた新しい $\mathcal{M}(A, V)$ で，再びモデル検査を行う．

これらの操作を繰り返すと，$(\texttt{near}, \texttt{coming down}, 0)$

から発火する a，$(\texttt{near}, \texttt{going up}, 1)$ から発火する a も取り除くことができる．よって $\mathcal{M}(A, V)$ は最終的に**図 9** となり，検証条件を満たすという結果を得ることができる．つまりこの例では，クロック変数をまったく用いない，ロケーションが状態であるようなオートマトンに変換した検査が可能となる．

4.3 clock 変数を復元する精密化

遷移の除去の精密化で r_{max} が除去できない場合は，クロック変数を復元することで，偽物の反例 ρ を生成しない新しい抽象システムの構築を試みる．クロック変数を復元する精密化で考慮すべき点は 2 つある．

1 つ目は，復元するクロック変数をどのように選択するかということである．クロック変数を復元した新しいマッピング V' から $\mathcal{M}(A, V')$ を構築しても，クロック変数の選択が適切でないと $\mathcal{M}(A, V')$ に ρ のような実行系列が含まれてしまう．その場合，また抽象システムを精密化で作り直さなければならない．$\mathcal{M}(A, V')$ の構築は，精密化する前の抽象システム $\mathcal{M}(A, V)$ を利用して構築することができないので高いコストを必要とする．したがって，実行系列 ρ が取り除かれていないような $\mathcal{M}(A, V')$ は構築しないことが望ましい．

2 つ目は，状態数爆発問題を考慮して，復元するクロック変数の数はできるだけ少ないほうがよい，ということである．また復元するクロック変数が決定したとき，それらの変数をすべての l に対する $V(l)$ に復元して V' とするのではなく，いくつかの l に対する $V(l)$ に復元し V' とするように，部分的に復元するほうがよい．

以上の 2 点をふまえて，V' を求める手続きを与える．

V' を求めるアルゴリズム 精密化する前のマッピングを V，反例検査オートマトン（定義 3.3）を $A_\rho = (L_\rho, L_{\rho,0}, \Sigma_\rho, X_\rho, I_\rho, T_\rho)$ とすると，以下のように与えられる．

(1) ρ を除去するのに必要なクロック変数の集合 $X_{ref} \subseteq X_\rho$ を求める．

(2) マッピング $V_{ref} : L_{ref} \to X_{ref}$ を求める．ただし $L_{ref} = \{l_0, l_1 \ldots, l_{max}\} \subset L_\rho$．

(3) V' を以下のように与える．

$$V'(l) := \begin{cases} V(l) \cup V_{ref}(l) & l \in L_{ref} \text{のとき} \\ V(l) & l \notin L_{ref} \text{のとき} \end{cases}$$

次に X_{ref} の求め方について説明する．まず，$X_s \in 2^{X_\rho}$ 対して，マッピング $V_s : L_{ref} \to X_s$ （ただし $\forall l \in L_{ref}\,.\, V_s(l) = X_s$）から $\mathcal{M}(A_\rho, V_s)$ を構築することを考える．もし X_ρ の要素数が少ないなら，すべての X_s に対する $\mathcal{M}(A_\rho, V_s)$ で，l_{max} から t_{max} が実行可能かを検査し，実行が不可能でかつ要素数が最小の X_s を X_{ref} とすればよい．先にも述べたが，A_ρ は遷移が決定的なオートマトンなので，$\mathcal{M}(A_\rho, V_s)$ を構築するコストはそれほど高くはない．したがって，このような総当たり的な手法でも十分である．しかし X_ρ の要素数が多い場合，このような手法では X_{ref} を求めるのに多くの時間を要することになってしまう．この場合の X_{ref} の求め方としては様々なものが考えられるが，本稿では**図 10** のアルゴリズムで求めることを提案する．このアルゴリズムから得られる X_{ref} の要素から適当な 1 つを選択し，これを復元するクロック変数の集合 X_{ref} とする．

```
 1: X_ref := {X_ρ} ;
 2: Candidate := ∅ ;
 3: NewCandidate := {X_ρ} ;
 4: RuleSet := ∅ ;
 5:
 6: do
 7:   for each X_s ∈ NewCandidate do
 8:     for each x ∈ X_s do
 9:       if ∃Rule ∈ RuleSet(Rule ⊆ X_s − {x}) then
10:         Candidate := Candidate ∪ {X_s − {x}} ;
11:       end if
12:     end for each
13:   end for each
14:
15:   NewCandidate := ∅ ;
16:   for each X_s ∈ Candidate do
17:     V_s :=create-map(X_s);
18:     if M(A_ρ, V_s) ⊭ reach(l_max+1) then
19:       NewCandidate := NewCandidate ∪ {X_s};
20:     else RuleSet := RuleSet ∪ {X_ρ − X_s};
21:     end if
22:   end for each
23:   if NewCandidate ≠ ∅ then X_ref := NewCandidate ;
24: while NewCandidate ≠ ∅ ;
25:
26: return X_ref ;
```

図 10 X_{ref} 求めるアルゴリズム
Fig. 10 X_{ref} algorithm.

このアルゴリズムは 7 行目から 13 行目で，調べなければならない X_ρ の部分集合を $NewCandidate$ から作り出している．$NewCandidate$ はクロック変数の集合を要素として持つ集合で，最初は X_ρ を要素として持つ．$NewCandidate$ の各要素から，その要素自身が持つ任意のクロック変数を 1 つ取り除いて，新しい X_ρ の部分集合を作り $Candidate$ に加える．

9 行目で，X_ρ のすべての部分集合を構築するのを

避け，集合 $RuleSet$ を用いて枝狩りしている．集合 $RuleSet$ は，ρ を除去するのに必要となるクロック変数の集合を要素として持つ．$RuleSet$ の要素のいずれかを部分集合としないような X_s は，$\mathcal{M}(A_\rho, V_s)$ を構築し調べるまでもなく，ρ を除去することができない．このことはシミュレーション関係がプレオーダであることから説明できる．つまり，$V_s^1(l) \subset V_s^2(l)$ なら，$\mathcal{M}(A_\rho, V_s^1) \preceq \mathcal{M}(A_\rho, V_s^2)$ である．もし $\mathcal{M}(A_\rho, V_s^1)$ で l_{max} から遷移 t_{max} が実行可能なら，$\mathcal{M}(A_\rho, V_s^2)$ でも同様である．したがって，$RuleSet$ の要素を部分集合としない X_s は調べる必要がない．

16 行目から 22 行目では，得られた $Candidate$ の各要素に対して $\mathcal{M}(A_\rho, V_s)$ を構築し，その要素が適当なものであるか調べている．まず `create-map` でマッピング V_s を作り，l_{max} から遷移 t_{max} が実行不可能ならその要素を $NewCandidate$ に加え，X_ρ の新しい部分集合を構築するのに用いる．実行可能であった場合，X_ρ とその要素との差となるクロック変数の集合は，省略することができないものとして $RuleSet$ に加える．すべての要素で実行可能になってしまう場合，24 行目から 26 行目のようにループを抜け出して，X_{ref} を結果として返す．

X_{ref} の決定の次はマッピング $V_{ref} : L_{ref} \to X_{ref}$ を求める．この V_{ref} は，L_{ref} の各ロケーションで復元するクロック変数を表している．もちろん $\forall l \in L_{ref} \,.\, V_{ref}(l) = X_{ref}$ としても反例 ρ は除去できるが，クロック変数が復元されてしまうロケーションをなるべく少なくするために，アクティブクロックというものを用いる．アクティブクロックとは，そのロケーションから到達可能なロケーションにおいて，必要とされるクロック変数のことを意味している．これを使って，L_{ref} のロケーション l で復元されるクロック変数は，ロケーション l のアクティブクロックに含まれ，かつ X_{ref} に含まれているものとする．このことの正しさは，アクティブクロックの性質から述べることができる．アクティブクロックと以下の定理については，詳しくは文献 6) を参照されたい．

定理 3. 時間オートマトンを $A = (L, L_0, \Sigma, X, I, T)$ としたとき，ロケーション $l \in L$ に対するアクティブクロックの集合を $act(l)$ で表現すると，以下のような性質が成り立つ．

$$\mathcal{M}(A, act) \models f \Leftrightarrow \mathcal{Z}(A) \models f$$

この定理から，クロック変数を復元する精密化を行う場合，ロケーション l で復元するクロック変数は，$act(l)$ に含まれるもののみでよい．

マッピング V_{ref} $V_{ref} : L_{ref} \to X_{ref}$ は，以下のように定義する．

$$\forall l \in L_{ref} \,.\, V_{ref}(l) = act(l) \cap X_{ref}$$

部分的に精密化を行うものとして文献 8) があるが，この手法では完全性については保証されていない．ここでの完全性とは，状態数爆発問題を考えなければ，精密化により偽物の反例をすべて除去でき，性質を満たすか否かの結果を得られることを意味する．提案手法のクロック変数を復元する精密化では，これが定理 3 から保証できる．これは精密化を繰り返すことにより，最終的に V は act となるからである．また，文献 8) の手法では，精密化を行う範囲が，偽物の反例に出現する状態だけにとどまらず，広範囲に及ぶことがある．提案手法の精密化の範囲は，反例中のロケーションのみにとどめることができる．

次節では，クロック変数を復元する精密化の適用例を示す．

4.4 例 2

図 5 のシステムについての検証を例として示すが，ここでは，遷移を除去する精密化は行わないことにする．

4.2 節と同様に，初期抽象システムを $\mathcal{M}(A, V)$（ただし $\forall l \in L \,.\, V(l) = \emptyset$）として，モデル検査を行うと**図 11** のような偽物の反例を得る．この反例を ρ とすれば，$X_\rho = \{x, y, z\}$，$L_{ref} = \{l_0, l_1, l_2\}$ となる．ただし，$l_0 = (\texttt{far}, \texttt{up}, 0)$，$l_1 = (\texttt{near}, \texttt{up}, 1)$，$l_2 = (\texttt{near}, \texttt{coming down}, 0)$．ここで X_{ref} を求めるが，この場合は $X_s = \{x\}$ として $\mathcal{M}(A_\rho, V_s)$ を構築すると，**図 12** のように状態 $(l_2, 0 \le x \le 1)$ から遷移 a が実行できない，つまり，$SUCC(t, 0 \le x \le 1)$ が空集合ということである（ただし $t = \langle l_2, a, x > 2, \emptyset, (\texttt{in}, \texttt{coming down}, 0)\rangle$）．したがって，定義 3.1 より $succ(t, 0 \le x \le 1)$ が空集合であり，計算してみると以下のようになる．

$$\begin{aligned} &succ(t, 0 \le x \le 1) \\ &= ((0 \le x \le 1) \wedge I(l_2))^{\Uparrow} \wedge I(l_2) \wedge (x > 2) \\ &= (0 \le x \le 1 \wedge y < 1)^{\Uparrow} \wedge I(l_2) \wedge (x > 2) \\ &= (x \ge 0 \wedge y \ge 0 \wedge -1 < x - y \le 0) \\ &\quad \wedge I(l_2) \wedge (x > 2) \end{aligned}$$

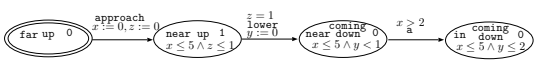

図 11 反例 ρ
Fig. 11 Counterexample ρ.

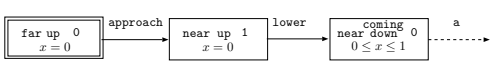

図 12 $\mathcal{M}(A_\rho, V_s)$
Fig. 12 $\mathcal{M}(A_\rho, V_s)$.

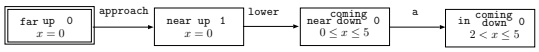

図 13 $succ'(t,\varphi_{V(l)})$ を用いた $\mathcal{M}(A_\rho,V_s)$
Fig. 13 $\mathcal{M}(A_\rho,V_s)$ in $succ'(t,\varphi_{V(l)})$.

$$= (0 \le x < 2 \wedge 0 \le y < 1 \wedge -1 < x-y \le 0) \wedge (x > 2)$$

したがって，$X_{ref} = \{x\}$ である．

もし，$SUCC(t,\Phi) = \{\nu \mid \exists \nu' \in succ'(t,\Phi)\ \forall x \in V(l')\,.\,\nu(x) = \nu'(x)\}$ であると，$\mathcal{M}(A_\rho,V_s)$ は**図 13** のようになり遷移 a が実行できてしまう．$succ'(t,\Phi)$ については定義 3.1 を参照されたい．この場合は，最終的に $X_{ref} = \{x,y,z\}$ となってしまうため，提案手法のほうが少ないクロック変数で反例を除去できる．

$X_{ref} = \{x\}$ とし，V_{ref} を求める．l_0，l_1，l_2 のアクティブクロックの集合はそれぞれ，$act(l_0) = \emptyset$，$act(l_1) = \{x,z\}$，$act(l_2) = \{x,y\}$ である．したがって，V_{ref} は $V_{ref}(l_0) = \emptyset$，$V_{ref}(l_1) = \{x\}$，$V_{ref}(l_2) = \{x\}$ となる．

この精密化で得られる新しいマッピング V' は，V_{ref} から次のようになる．

$$V'(l) := \begin{cases} \{x\} & l = l_1, l_2 \text{のとき} \\ \emptyset & l \neq l_1, l_2 \text{のとき} \end{cases}$$

このようにクロックを復元していく精密化を繰り返し，偽物の反例を除去していくと，最終的なマッピング V は以下のようになり，検証結果は 4.2 節と同様の結果を得られる．

$$\begin{aligned} V((\texttt{far},\texttt{up},0)) &= \emptyset \\ V((\texttt{near},\texttt{up},1)) &= \{x,z\} \\ V((\texttt{near},\texttt{coming down},0)) &= \{x\} \\ V((\texttt{near},\texttt{down},0)) &= \{x\} \\ V((\texttt{in},\texttt{down},0)) &= \{x\} \\ V((\texttt{in},\texttt{going up},1)) &= \emptyset \\ V((\texttt{in},\texttt{up},1)) &= \emptyset \\ V((\texttt{in},\texttt{coming down},0)) &= \emptyset \\ V((\texttt{far},\texttt{coming down},2)) &= \emptyset \\ V((\texttt{far},\texttt{down},2)) &= \{z\} \\ V((\texttt{near},\texttt{going up},1)) &= \{x,z\} \\ V((\texttt{far},\texttt{going up},0)) &= \emptyset \end{aligned}$$

5. 実　　験

従来の時間オートマトンの到達可能性解析と，提案手法である抽象化・精密化を用いた到達可能性解析を，比較するために実験を行った．実験は，Fischer の相互排除アルゴリズムを両手法で検証し，その実行時間，消費メモリ，状態数を比較した．Fischer の相互排除アルゴリズムは，実時間モデル検査アルゴリズムの性能比較によく用いられる代表的な題材であり，時間オートマトンでモデル化する際，プロセス 1 つに対しクロック変数が 1 つ必要となる．したがって，クロック変数の数とプロセスの数は等しい．両手法の実装では，公平を期するためにゾーンのデータ表現形式を DBM [7] で統一した．さらに，実時間モデル検査時の $succ$ の計算アルゴリズムや，状態の探索アルゴリズムも両手法では同じものを用いることにした．また制限として，今回の実験における提案手法の実装では，クロック変数を復元する精密化の部分は実現しておらず，精密化は遷移の除去のみが可能である．クロック変数を復元する精密化の実装は今後の課題であるが，Fischer の相互排除アルゴリズムを検証する限りにおいては，その検証結果に影響はなかった．つまり，遷移を除去する精密化のみで検証することが可能であった．実験に用いた計算機環境は，CPU が Pentium4 の 2.4 GHz で，メモリが 4 GB である．

表 2 Fischer の相互排除アルゴリズムの検証における実行時間の比較
Table 2 Comparison of excution times in the verification of Fischer's protocol.

プロセス数	従来手法の CPU 時間（秒）	提案手法の CPU 時間（秒）
2	0.010	0.010
3	0.050	0.040
4	1.790	0.830
5	-	33.110
6	-	919.260
7	-	28,197.010

まず，実行時間の比較を**表 2** に示す．単位は秒である．従来手法ではプロセス数が 5 のとき，50 秒程度でメモリが不足となってしまい検証できず，それ以上のプロセス数に対しても検証することができなかった．これに対して，提案手法では 7 プロセスまで検証が可能であった．4 プロセス以下で，従来手法と提案手法の実行時間を比較すると，提案手法の方が実行時間が短くなっているのが分かる．4 プロセスの時点で提案手法の実行時間は，従来手法に比べて 2 分の 1 程度に減少している．

次に，両手法のメモリ消費量の比較を**図 14** に示す．横軸がプロセス数で，縦軸が消費メモリ（単位は K バイト）の対数となっている．実線が従来手法で破線が提案手法である．この図から，従来手法と比べて提案手法の方が，プロセス数に対する消費メモリの増加を抑えられることが分かる．

最後に，従来手法における状態数と，提案手法で最

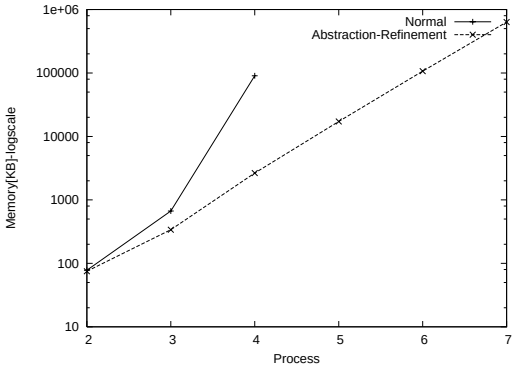

図 14 Fischer の相互排除アルゴリズムの検証における消費メモリの比較
Fig. 14 Comparison of memory sizes in the verification of Fischer's protocol.

表 3 Fischer の相互排除アルゴリズムの検証における状態数比較
Table 3 Comparison of the number of states in the verification of Fischer's protocol.

プロセス数	従来手法の状態数	提案手法の状態数
2	31	18
3	403	65
4	9,678	220
5	-	727
6	-	2,378
7	-	7,737

終的に構築される抽象システムの状態数の比較を**表 3** に示す．提案手法を用いた検証で行われた精密化の回数，つまり除去された遷移の数は，プロセス数が 2，3，4，5，6，7 に対して，それぞれ 2，15，76，325，1,266，4,655 であった．

以上の結果から，状態数爆発問題を抑えるのに提案手法が有効であることが分かる．

6. 関連研究との比較

抽象化・精密化を用いたモデル検査について述べているものには，文献 8) などがある．Henzinger ら[8] は，本来は C 言語のソースコードに対する検証を目的とする，ソフトウェアモデル検査の手法の 1 つである．ソースコードを有限オートマトンに変換し，既存の定理証明器を用いて抽象化・精密化を行う．定理証明器を用いるため，通常の有限オートマトンや時間オートマトンだけでなく，様々な問題の検証に用いることが可能な汎用的な手法である．検証性質については，本手法と同様に安全性について論じている．論文中では時間オートマトンの検証にも利用可能であると述べているが，その有効性については述べられていない．本手法は，時間オートマトンに対する検証にしか用いることができないが，その反面，クロック変数に特化した手法であるため，時間オートマトンの検証には本手法の方が有効であろうと考える．また 4.3 節の最後でも述べたが，提案手法では，完全性を保証できているのに対して，文献 8) では保証できておらず，精密化を施す範囲も広範囲に及ぶ可能性がある．さらに，遷移を除去するような精密化の手法は提案されてはいない．

実時間モデル検査で抽象化を扱ったものとしては，Tripakis ら[12] がある．これは本稿と同じように，時間オートマトンのクロック変数に関する抽象化ではあるが，バイシミュレーション関係の抽象システムを構築することを目的としている．バイシミュレーション関係とは，2 つのシステムを M，M' とすると，$M \models f \Leftrightarrow M' \models f$ という性質が成立することをいう．このため，この関係に基づく抽象化は精密化を必要としない．また，提案手法がロケーションへの到達可能性のみ検証可能なのに対して，この手法では TCTL 論理式の検証が可能である．TCTL 論理式では時間量についての記述が可能で，たとえば「5 単位時間までは，必ず $\neg l$ である」（$AG_{\leq 5}\neg l$）ということなどが記述できる．このような強い性質が成立する反面，バイシミュレーション関係に基づく抽象化では，状態数があまり削減されないことが多い．提案手法はシミュレーション関係に基づくものなので，精密化をする必要があるが，状態数について考えればより多くの削減を望めるものと考える．

7. ま と め

本稿では，抽象化・精密化を用いた時間オートマトンの到達可能性解析の改善のために，遷移を除去する精密化とクロック変数を復元する精密化を提案した．また，提案手法による実験を行い，遷移を除去する精密化の状態数爆発問題に対する有効性を示した．今後の課題としては，クロック変数を復元する精密化の有効性を示すために，より複雑な検証対象へ，本手法を適用した事例研究などを行う必要があると考えている．

謝辞 第 48 回プログラミング研究会においてコメントをいただいた方々，および本稿を査読していただいた方につつしんで感謝の意を表する．

参 考 文 献

1) Alur, R., Courcoubetis, C. and Dill., D.: Model checking in dense real-time, *Information and Computation*, Vol.104, No.1, pp.2–34 (1993).

2) Alur, R. and Dill, D.: Automata for modeling real-time systems, *Proc. 17th ICALP*, LNCS, Vol.433, pp.322–335, Springer-Verlag (1990).
3) Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P. and McKenzie, P.: *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer-Verlag (1999).
4) Clarke, E., Gupta, A., Kukula, J. and Strichman, O.: SAT based Abstraction-Refinement using ILP and Machine Learning Techniques, *Proc. CAV2002*, LNCS, Vol.2404, pp.265–279, Springer-Verlag (2002).
5) Clarke, E.M., Grumberg, O. and Peled, D.A.: *Model Checking*, MIT Press (1999).
6) Daws, C. and Yovine, S.: Reducing the number of clock variables of timed automata, *Proc. 1996 IEEE Real-Time Systems Symp.*, pp.73–81, IEEE Computer Society Press (1996).
7) Dill, D.: Timing assumption and verification of finite-state concurrent systems, *Proc. Automatic Verification Methods for Finite State Systems*, LNCS, Vol.407, pp.197–212, Springer-Verlag (1989).
8) Henzinger, T.A., Jhala, R., Majumdar, R. and Sutre, G.: Lazy Abstraction, *Proc. 29th Annual Symp. on POPL*, pp.58–70, ACM Press (2002).
9) Henzinger, T.A., Nicollin, X., Sifakis, J. and Yovine, S.: Symbolic model-checking for real-time systems, *Proc. 7th Symp. on Logics in Computer Science*, pp.394–406, IEEE Society Press (1992).
10) Huth, M. and Ryan, M.: *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press (2000).
11) Penczek, W., Woźna, B. and Zbrzezny, A.: Towards Bounded Model Checking for the Universal Fragment of TCTL, *Proc. FTRTFT2002*, LNCS, Vol.2469, pp.265–288, Springer-Verlag (2002).
12) Tripakis, S. and Yovine, S.: Analysis of timed systems based on time-abstracting bisimulations, *Formal Methods in System Design*, Vol.104, No.1, pp.25–68 (2001).
13) Wang, F.: Efficient Timed Reachability Analysis Using Clock Difference Diagrams, *Proc. CAV'99*, LNCS, Vol.1633, pp.341–353, Springer-Verlag (1999).
14) Wang, F.: Efficient Data Structer for Fully Symbolic Verification of Real-Time Software Systems, *Proc. TACAS2000*, LNCS, Vol.1785, pp.157–171, Springer-Verlag (2000).

(平成 16 年 2 月 20 日受付)
(平成 16 年 7 月 25 日採録)

中島　一

2003 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科在学．モデル検査法の研究に従事．

亀山　幸義（正会員）

東京大学大学院理学系研究科修士課程修了．現在，筑波大学システム情報工学研究科助教授，科学技術振興機構「機能と構成」領域研究員を兼任．ソフトウェア基礎論，特に，プログラムの論理と検証に興味を持つ．日本ソフトウェア科学会，ACM 各会員．