社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2007-29 (2007-08)

UPPAAL拡張時間オートマトンの反例に基づく抽象化改良ループによるモデル抽象化手法

長岡 武志† 岡野 浩三† 楠本 真二†

† 大阪大学大学院情報科学研究科
〒 560–8531 大阪府豊中市待兼山 1–3

あらまし 近年，情報システムの高信頼設計においてモデル検査は重要な役割を果たすようになってきている．実時間システムのモデル検査では，モデルの状態数が状態変数やクロック変数に対して指数的に増加し，スケーラビリティに厳しい制限が生じる．本稿では，時間モデル検査ツールである UPPAAL で用いられる拡張時間オートマトンに対する具体的なモデル抽象化手法を提案する．モデル抽象化技法はこのスケーラビリティ改善のための主要な手法として注目を浴びている．本手法では，Clarke らが提案した反例に基づいた抽象化改良ループ CEGAR を適用しており，抽象化から検査までの全ての工程を自動的に行うことが可能である．また，反例によるモデルの再抽象化においては時間オートマトンの検証で用いられるデータ構造 DBM を活用する工夫を行なっている．

キーワード モデル検査，時間オートマトン，モデル抽象化，CEGAR，UPPAAL

Abstraction of Extended Timed Automata for UPPAAL Based on Counterexample-Guided Abstraction Refinement Loop

Takeshi NAGAOKA†, Kozo OKANO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University
Machikane-yama 1–3, Toyonaka City, Osaka, 560–8531 Japan

Abstract In the model checking of the realtime systems, the number of states of models increases exponentlly with the number of the state variables and the clock variables. Such explosion severely limits the scalability of model checking. This paper proposes a concrete model abstraction technique for extended timed automata used in UPPAAL, a famous model checking tool for realtime systems. In the proposed technique, the abstraction refinement loop based on the counterexample that Clarke et al. proposed is applied, and all the processes from the abstraction can be performed automatically. We gives concrete procedures for the abstraction loop with utilization of DBM.

Key words Model Checking, Timed Automata, Model Abstraction , CEGAR, UPPAAL

1. まえがき

近年，情報システムの高信頼設計にモデル検査を活用することが強く望まれている．モデル検査手法はシステムを有限の状態遷移系として記述し，状態遷移系の全状態を探索することで，システムが仕様を満たすかどうかを証明する手法である．しかし，大規模なシステムに対しては状態数爆発を起こすなど，スケーラビリティの弱さが課題となっている．モデル検査のスケーラビリティの弱さを改善する手法として，検査する性質ごとに，モデルの状態数を適切に削減するモデル抽象化手法が注目されている [1], [2], [7].

一方，実時間システムの動作検証には，有限の状態遷移系に実時間制約を付加した時間オートマトン [4], [5] と呼ばれるモデルが用いられる．時間オートマトンでは，有限のロケーションと呼ばれる状態に，実数値をとるクロック変数を用いた制約が付加されるため，時間オートマトンは無限の状態空間を持つことになる．モデル抽象化を行わない従来の時間モデル検査では時間領域が実質，有限個に押さえられることを利用し有限状態のモデルに対し検査を行う．しかし，この状態数はロケーションやクロックの個数に対して指数的に増加するため，状態数を適切に縮小するモデル抽象化手法が必要となる．

本稿では，時間モデル検査ツール UPPAAL [6] で用いられる拡張時間オートマトンに対する具体的なモデル抽象化手法を提案する．抽象化には有限モデルやハイブリッドオートマトンなどの抽象化に用いられている「反例に基づいた抽象化改良ループ [1], [3]」を適用しており，抽象化から検査までの全ての工程

を自動的に行うことが可能である.「反例に基づいた抽象化改良ループ」では，抽象化を行ったモデル (抽象モデル) に対してモデル検査を適用し，検査の結果生じた反例が抽象化を行う前のモデル (具象モデル) 上で再現できるかどうかを調べ，具象モデル上で再現不可能な反例 (見せかけの反例) を検出した場合は，そのような反例が発生しないように抽象モデルを改良する，という操作を繰り返すことで適切な抽象モデルを生成する手法である．本提案手法では，時間オートマトンの状態空間の表現手法である DBM (Difference Bound Matrix) [6] に関するアルゴリズムを利用して抽象モデル上で発生した反例の再現を実現している．抽象モデルの改良には，見せかけの反例が生じた原因である抽象状態を分割，遷移の除去を行うことによって見せかけの反例が発生しないように改良を行う.

関連研究としては，文献 [7] では SAT ソルバによるモデル検査をベースとした時間オートマトンの抽象化手法を提案している．この手法では，反例が再現可能かどうかの判定にも SAT ソルバを利用し，抽象モデルの改良では，見せかけの反例が発生しないように，モデルを表現している命題論理を改良している．しかし，時間変数が残ったままであるので純粋な SAT ソルバを適用できないなどの問題点がある.

以降，2. では準備として時間オートマトンについて説明する．次に 3. でモデル抽象化に関する定理を与える．4. で「反例に基づいた抽象化改良ループ」の具体的なアルゴリズムについて述べ，5. では提案した抽象化手法を適用した例を挙げる．6. で考察を与え，最後に，7. でまとめる.

2. 時間オートマトン

時間オートマトンは有限状態のオートマトンに時間経過を表現するクロック変数を付加したオートマトンである．本節では，標準的な時間オートマトンの構成要素や意味，更に時間モデル検査器 UPPAAL で扱っている拡張時間オートマトンについて簡単に紹介する.

2.1 標準的な時間オートマトン

標準的な時間オートマトンは以下のように与えられる [6].

定義 2.1 (時間オートマトン)**.** 時間オートマトン TA は 6 項組 (L, l_0, C, A, E, I) である．ここで L はロケーションの集合であり，$l_0 \in L$ は初期ロケーションである．C はクロックの集合，A はアクションの集合である．アクションは他のプロセスと同期を取るアクションと 1 つのプロセス内だけの τ-アクションから構成される．$E \subseteq L \times A \times B(C) \times 2^C \times L$ はロケーション間の状態遷移の集合であり，アクション，ガード，リセットするクロックの集合を持つ．$B(C)$ はクロックに関する条件式の論理積の集合である．$I : L \to B(C)$ は各ロケーションに対してインバリアントを割り当てる.

定義 2.2 (時間オートマトンの意味)**.** 時間オートマトン TA は次のような遷移システム $(S, s_0, \to)$ として定義される．$S \subseteq L \times \mathbb{R}^C$ は状態の集合であり，$s_0 = (l_0, u_0)$ は初期状態である．また，$\to$ は次のような遷移関係を表現する.

- $(l, u) \xrightarrow{d} (l, u + d)$ if $\forall d' : 0 \leqq d' \leqq d \Rightarrow u + d' \in I(l)$
- $(l, u) \xrightarrow{a} (l', u') \exists e = (l, a, g, r, l') \in E$ s.t $u \in g, u' = [r \mapsto 0]u, u' \in I(l')$

$u \in g$ は $u \in \mathbb{R}^C$ がガード g を満たすことを示している．同様に，$u \in I(l)$ は $u \in \mathbb{R}^C$ がロケーション l でのインバリアントを満たすことを示している．また，$[r \mapsto 0]u$ は u において r に含まれる全てのクロックを 0 にリセットすることを意味している.

2.2 時間オートマトンの状態空間の表現

UPPAAL などのツールでは，時間オートマトンの無限の状態空間の表現方法として，ゾーンと呼ばれる表現を用いている [5]．ゾーンはクロックに関する制約の集合であり，ゾーンが持つ制約を満たす全ての状態を表現する．時間オートマトンが持つ状態空間は有限個のゾーンから構成されるゾーングラフで表現することができる．また，ゾーンが持つ制約の集合は，DBM(Difference Bound Matrix) とよばれる行列として表現することができる．DBM では，時間制約が一般に 2 クロック変数の差分不等式の集合で表されることを利用し，時間オートマトンが持つ n 個のクロックに対して $(n+1) \times (n+1)$ の行列として全ての制約を表現する．DBM のクロック制約を満たす全ての解の空間を D とすると，時間オートマトンの状態集合として $(l, D) = \bigcup_{u \in D}(l, u)$ として表現することとする．また，全ての $l \in L$ に対して $D_l^{inv} = \bigcup_{u \in I(l)} u$ と表現する.

2.3 UPPAAL における時間オートマトンの拡張

UPPAAL では，標準的な時間オートマトンに対して拡張を行った拡張時間オートマトンを検証モデルとしている [6]．標準的な時間オートマトンに対して UPPAAL 拡張時間オートマトンが追加している主な特徴を以下に示す.

bounded integer variable　インバリアントやガード制約，変数の代入文に含めることが可能な整数変数．とり得る値の範囲の指定が可能.

urgent synchronisation　この同期が可能な状態では時間経過が不可能となる同期.

urgent location　ロケーション上での時間経過が不可能であるロケーション.

commited location　ロケーション上での時間経過が不可能であり，次の瞬間にこのロケーションから抜け出す必要のあるロケーション.

これらの拡張点のうち，整数変数については抽象化による状態数の削減が可能であると考えられる．しかし，以降で述べる抽象化手法は標準的な時間オートマトンが持つクロック変数の抽象化であり，整数変数の抽象化は 6. の考察で述べている.

3. モデル抽象化

この節では時間オートマトン TA によって生成される状態遷移系 $M = (S, s_0, \to)$ の抽象モデル $\hat{M} = (\hat{S}, \hat{s_0}, \hat{\to})$ を与える．なお，簡単化のため，ここでは標準的な時間オートマトンを対象とする．抽象化関数 $h : S \to \hat{S}$ は S に含まれる具体的な状態から $\hat{S}$に含まれる抽象状態 へのマッピングである．文献 [2] より M に対する抽象化関数 h は定義 3.1 のように与えられる.

定義 3.1 (抽象化関数 h). 状態遷移系 $M = (S, s_0, \rightarrow)$ を抽象化関数 h によって抽象化されたモデル $\hat{M} = (\hat{S}, \hat{s_0}, \hat{\rightarrow})$ は以下のように与えられる.

(1) $\hat{S} = \{\hat{s} | \exists s.s \in S \wedge h(s) = \hat{s}\}$

(2) $\hat{s_0} = h(s_0)$

(3) $\hat{\rightarrow} = \{(\hat{s_1}, \hat{s_2}) | \exists s_1 \in S. \exists s_2 \in S. (s_1 \rightarrow s_2) \wedge h(s_1) = \hat{s_1} \wedge h(s_2) = \hat{s_2}\}$

また, $h^{-1} : \hat{S} \rightarrow 2^S$ を h の逆関数として与える.

本稿で提案する抽象化手法では, システムの異常な状態を表すロケーション $e \in L$ に対して, $AG\neg e$ を検査する. これは e への遷移は存在しないということを表す検査式である. この検査式に対する反例は必ずループを含まない.

文献 [2] では, 命題論理 p に対し検査式 AGp を検査するための状態縮約にともなう意味変化をなるべく起さないという意味で保守的な抽象化についての定理が与えられている. 与えられた定理から以下の定理を導くことができる.

定理 3.1. 時間オートマトン $TA = (L, l_0, C, A, E, I)$ によって生成される状態遷移系 $M = (S, s_0, \rightarrow)$ に対し, 抽象化関数 h を用いて抽象化された抽象モデルを $\hat{M} = (\hat{S}, \hat{s_0}, \hat{\rightarrow})$, システムの異常な状態を表すロケーションを $e \in L$, 抽象モデルにおけるエラー状態の集合を $\hat{Er} = \bigcup_{u \in I(e)} (h(e, u))$ とする. $h(s) \models \neg e \iff h(s) \notin \hat{Er}$ とした時, 全ての $s \in S$ に対して

$$h(s) \notin \hat{Er} \Rightarrow s \notin \bigcup_{u \in I(e)} (e, u) \qquad (1)$$

ならば, $\hat{M} \models AG\neg e \Rightarrow M \models AG\neg e$.

Proof. 文献 [2] の定理 1 より, 命題論理 p に対して, 抽象化関数 h が全ての $s \in S$ について

$$h(s) \models p \Rightarrow s \models p \qquad (2)$$

を満たすならば, 状態遷移系 M を h によって抽象化したモデルを $\hat{M}$ とすると, $\hat{M} \models AGp \Rightarrow M \models AGp$ である. $s \models \neg p$ は $s \notin \bigcup_{u \in I(e)} (e, u)$ であり, 式 1 と式 2 は同値である. 従って文献 [2] の定理 1 より, 定理 3.1 を証明できる. □

定理 3.1 より, 抽象モデルに対して $AG\neg \bigvee_{\hat{e} \in \hat{Er}} \hat{e}$ を示すことができれば, 具象モデルでも $AG\neg e$ を示すことができる.

4. 反例に基づいた抽象化改良ループ

モデル抽象化では, 過剰な抽象化や誤った抽象化によって本来のモデルでは発生するはずのない見せかけの反例が発生してしまう場合がある. 文献 [1] では, 最初に十分な抽象化を行い, 抽象モデル上で見せかけの反例が発生すればその反例が発生しないように抽象化されたモデルを改良する, 反例に基づいた抽象化改良ループを提案している. 抽象化改良ループを図 1 に示す. 本節では, 時間オートマトン TA が生成する状態遷移系 $M = (S, s_0, \rightarrow)$ に対する具体的な抽象化改良ループを与える.

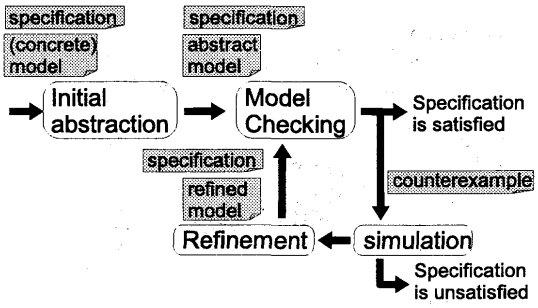

図 1 抽象化改良ループ

4.1 初期抽象化

初期抽象化は, 具体的な状態遷移システム $M = (S, s_0, \rightarrow)$ に対して適用する最初の抽象化である. 提案手法では, 時間オートマトンのクロック変数を全て省略するという方法で初期抽象化を行う. したがって初期抽象化関数 h は以下のように定義できる.

- $\forall l \in L, \forall u_1 \in I(l), \forall u_2 \in I(l). h((l, u_1)) = h((l, u_2)) = l$
- $\forall (l_1, u_1) \in S, \ \forall (l_2, u_2) \in S. \ h(l_1, u_1) = h(l_2, u_2) \iff l_1 = l_2$

このとき, 初期抽象化関数 h による抽象モデルは式 (1) を満たす. また, 提案手法の初期抽象化では, 2 つの異なるロケーションに属する状態を 1 つの抽象状態にマッピングする抽象化は行っていない. したがって, 以下で述べる改良を施した抽象モデルにおいても, 全ての抽象状態は 1 つのロケーション上の状態を縮約した状態である. なお, 後述の改良ステップにより, 必要なら, 抽象オートマトンの状態の細分化が時間領域の情報を用いて行なわれることとなる.

4.2 モデル検査

モデル検査では, 抽象モデル $\hat{M}$ に対してモデル検査を行う. 定理 3.1 より, 抽象モデルを生成する抽象化関数 h が式 1 を満たすならば, 抽象モデル $\hat{M}$ 上で性質を満たせば具象モデル M 上でも性質を満たすことは保障される. しかし, $\hat{M}$ 上で反例が発生した場合は, 反例が具象モデル M で再現できるものかどうかを確かめる必要がある.

4.3 シミュレーション

抽象モデル上で発生した反例が具象モデル上で再現できるかどうかの検査を行う. 抽象モデル上で発生した反例を $\hat{T} = \langle \hat{s_0}, \ldots, \hat{s_k} \rangle$ とする. このとき, 反例 $\hat{T}$ はロケーション遷移 $T = \langle l_0, \ldots, l_k \rangle (\bigwedge_{i=0}^{k} \exists u \in \mathbb{R}^C. (l_i.u) \in h^{-1}(s_i))$ に対応する.

提案手法では, 具象モデル上でロケーション遷移 T を再現し, ロケーション l_k へ到達可能性を調べることで反例が再現可能かどうかを調べている. ロケーション l_k へ到達可能性については, ロケーション遷移 $T = \langle l_1, \ldots, l_k \rangle$ に対して, 具象モデル上において初期状態から到達可能なゾーンを求めることで実現する. 各ロケーション l_i 上で到達可能なゾーンを D_i^{reach} とすると, $D_i^{reach} = \emptyset$ となる $i \leq k$ が存在すれば, 反例 $\hat{T}$ は再現不可能であると判断できる. 逆に $D_k^{reach} \neq \emptyset$ であれば反例

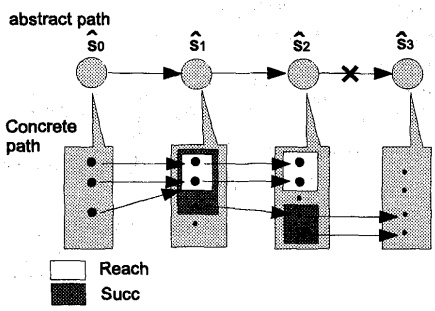

図2 反例パスの再現

$\hat{T}$ は再現可能であり，ループは終了する．

文献[5]ではゾーンを表現するDBMに対するアルゴリズムが紹介されており，これらのアルゴリズムを用いることで D_i^{reach} を求めることが可能である．なお，T に現れる制約(各ロケーションのインバリアント，各遷移におけるガード)の個数を m とすると，反例の再現は $O(mn^2)$ の時間計算量で実現できる．

4.4 抽象化改良

抽象モデル上で発生した見せかけの反例 $\hat{T} = \langle \hat{s_0}, \ldots, \hat{s_k} \rangle$ が発生しないように，抽象モデルの改良を行う．本手法では，文献[1],[3]で提案されている，抽象状態の分割，遷移の除去による改良手法を応用する．

見せかけの反例が生じる原因は，反例を具象モデル上で再現した際に，反例の初期状態から到達可能な状態集合 ($Reach$) と次状態へ遷移可能な状態集合 ($Succ$) の積集合が $\emptyset$ となるような抽象状態が存在するからである．このとき，$Reach$ に含まれる状態からは反例上の次状態への遷移は存在しないため具象モデル上では反例を再現できないが，$Succ$ に含まれる状態からは次状態へ遷移可能であり，抽象モデル上では次状態への遷移が生じてしまうのである．図2では抽象モデル上の反例 $\langle \hat{s}_0, \hat{s}_1, \hat{s}_2, \hat{s}_3 \rangle$ を具象モデル上で再現した例を示している．図の各抽象状態 $\hat{s}_i$ の下の図は状態集合 $h^{-1}(\hat{s}_i)$ を表している．図では抽象状態 $\hat{c}$ において，$Reach \cap Succ = \emptyset$ となっており，反例は再現不可能であることがわかる．

文献[1],[3]では，見せかけの反例を除去するために，抽象状態を分割し，$Reach$ と $Succ$ を別の抽象状態にマッピングする改良を行っている．

時間オートマトンにおいて，抽象状態を $Reach$ と $Succ$ に分割するには，クロックによる状態空間を分割する必要がある．しかし，状態空間の分割を行った場合，虫食い状の空間を形成してしまうなど，分割後の状態空間の処理が複雑化してしまうという問題が発生する．そこで本稿では，初期状態から到達可能な状態空間を複製することによって，反例の除去を行う手法を提案している．ここからは，到達可能な状態空間の複製による反例の除去を行う手法について，1) 状態の複製，2) 遷移の除去の2つの工程に分けて説明する．

1) 状態の複製

整数 f を反例 $\hat{T}$ において $D_f^{reach} = \emptyset$ となるような最小の整数とする(図2では $f = 3$)．このとき，抽象状態 $\hat{s}_{f-1}$ は初期状態から到達可能であり，かつ次状態へは遷移不可能な状態である．状態の複製では，$\hat{s}_{f-1}$ に対して，初期状態から到達可能な状態集合 $Reach$ を縮約する抽象状態として $\hat{s'}_{f-1}$ を生成する(図3参照)．時間オートマトンでは，到達可能な状態集合 $Reach$ はデータ構造DBMを利用することによって求めることができる．なお，$\hat{s}_{f-1}$，$\hat{s'}_{f-1}$ への遷移については，$h^{-1}(\hat{s}_{f-1}) \setminus h^{-1}(\hat{s'}_{f-1})$ への遷移が可能な状態のみ $\hat{s}_{f-1}$ への遷移が可能であるとし，このとき $\hat{s'}_{f-1}$ への遷移は不可能であるとする．

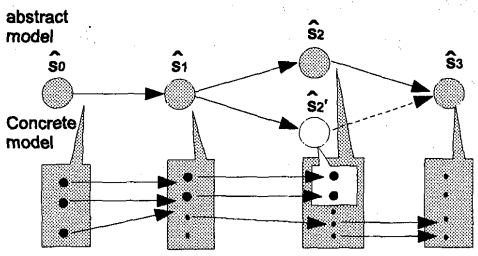

図3 状態の複製

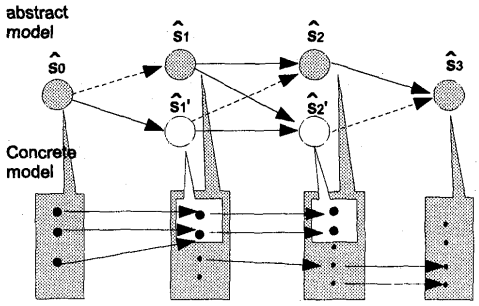

図4 改良した抽象モデル

2) 遷移の除去

まず，$\hat{s'}_{f-1}$ から $\hat{s}_f$ への遷移は不可能であるため，遷移 $(\hat{s'}_{f-1}, \hat{s}_{f-1})$ は除去することができる．さらに，$\hat{s}_{f-2}$ から $\hat{s}_{f-1}$ への遷移が不可能であれば，遷移 $(\hat{s}_{f-2}, \hat{s}_{f-1})$ を除去することができ，このとき，反例 $\hat{T}$ を除去することができる．しかし，遷移を除去できない場合は反例 $\hat{T}$ を除去することができない．この場合は，さらに反例 $\hat{T}$ の1つ前の状態 $\hat{s}_{f-2}$ に対して状態の複製，遷移の除去を行う，という操作を繰り返すことによって反例を除去することができる．これは，最終的に $\hat{s}_1$ の複製を行った場合，$\hat{s}_0$ から $\hat{s}_1$ への遷移は不可能であり，$(\hat{s}_0, \hat{s}_1)$ は除去可能であるため，反例を除去することができるのである．図3では，遷移 $(\hat{s'}_2, \hat{s}_2)$ は除去可能であるが，遷移 $(\hat{s}_1, \hat{s}_2)$ は除去不可能である．したがって図3のモデルの抽象状態 $\hat{s}_1$ に対して状態の複製を行う必要がある．$\hat{s}_1$ に対して状態の複製を行ったモデルを図4に示す．図4のモデルでは，遷移 $(\hat{s'}_1, \hat{s}_2)$，$(\hat{s}_0, \hat{s}_1)$ を除去可能であり，反例を除去することができる．

本改良手法は，クロック空間の複製，遷移の除去からなるが，時間オートマトンは有限の個数のゾーンから表現でき，遷移の個数も有限であり，本改良手法を用いた抽象化改良ループは必ず終了する．

最後に，提案手法による抽象モデルの改良手法の正しさにつ

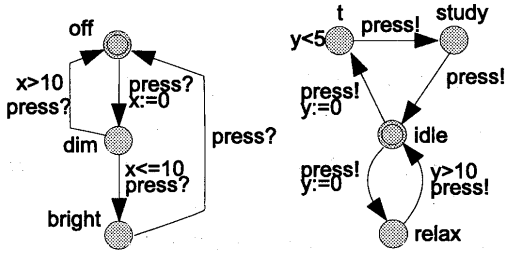

図5　照明システム

いて，各工程ごとに以下で簡単に説明する．

c1)　状態の複製

i 回目のループにおける抽象モデルを $\hat{M}_i$ とし，$\hat{M}_i$ において，$\hat{M}_i$ 上の状態 $\hat{s}_j$ の複製である $\hat{s'}_j$ を追加した抽象モデルを $\hat{M}_{i+1}$ とする．このとき，$\hat{M}_i$ における全ての遷移は $\hat{M}_{i+1}$ 上に存在する．したがって，$\hat{M}_{i+1}$ における全ての遷移が $\hat{M}_i$ 上の遷移へ置き換え可能であることを示すことで，2つの抽象モデルが同値であることを説明する．

まず，$\hat{M}_{i+1}$ における遷移 $(\hat{a},\hat{b})$ について，$\hat{a} \neq \hat{s'}_j$ かつ $\hat{b} \neq \hat{s'}_j$ であるとき，遷移 $(\hat{a},\hat{b})$ は $\hat{M}_i$ でも存在する．$\hat{a} = \hat{s'}_j$ である場合は，遷移 $(\hat{s'}_j,\hat{b})$ は $\hat{M}_i$ 上の遷移 $(\hat{s}_j,\hat{b})$ へ置き換え可能である．同様に $\hat{b} = \hat{s'}_j$ である場合，遷移 $(\hat{a},\hat{s'}_j)$ は $\hat{M}_i$ 上の遷移 $(\hat{a},\hat{s}_j)$ へ置き換え可能である．したがって，$\hat{M}_{i+1}$ 上の全ての遷移は $\hat{M}_i$ 上の遷移へ置き換え可能である．

c2)　遷移の除去

文献[1],[3]では，抽象モデル上における遷移 $(\hat{s}_1,\hat{s}_2) \in \hat{\rightarrow}$ に対して，$h^{-1}(\hat{s_1})$ のどの状態からも $h^{-1}(\hat{s_2})$ に含まれる状態に遷移不可能な場合に，遷移 $(\hat{s_1},\hat{s_1})$ の除去は抽象モデルの改良であるという補題が与えられている．

提案手法では上記の補題の条件を満たさない遷移の除去が生じる．以下では，補題の条件を満たさない遷移の除去について抽象化の改良であることを説明する．まず，反例 $\hat{T}$ 中の状態 $\hat{s}_j$ の複製 $\hat{s'}_j$ について，$D_j = h^{-1}(\hat{s}_j) \setminus h^{-1}(\hat{s'}_j)$ とし，$h^{-1}(\hat{s}_{j-1})$ の次状態の集合を $Succ_{j-1}$ とする．このとき提案手法では，$D_j \cap Succ_{j-1} = \emptyset$ である，つまり $h^{-1}(\hat{s}_{j-1})$ から遷移可能な状態は $h^{-1}(\hat{s'}_j)$ に含まれる状態のみである場合は，$\hat{s}_{j-1}$ から $\hat{s}_j$ への遷移は不可能であるとし，遷移を除去している．この場合，$h^{-1}(\hat{s}_{j-1})$ からは到達不可能である D_j への遷移は除去されているが，$h^{-1}(\hat{s}_{j-1})$ から到達可能な状態集合 $Succ_{j-1}$ への遷移は残されているおり，このような遷移の除去は抽象モデルの改良であるといえる．

5.　抽象化改良例

本節では，具体的な時間オートマトンのモデルに対して本稿で提案している抽象化手法適用した例を示す．適用したモデルは時間に依存した照明システム[5]である．照明システムのモデルを図5に示す．照明システムは照明のスイッチモデル(図左)とユーザモデル(図右)から構成される．照明システムのエラー状態はユーザモデルが $idle$ 状態である時にスイッチモデルが dim または $bright$ である状態とする．

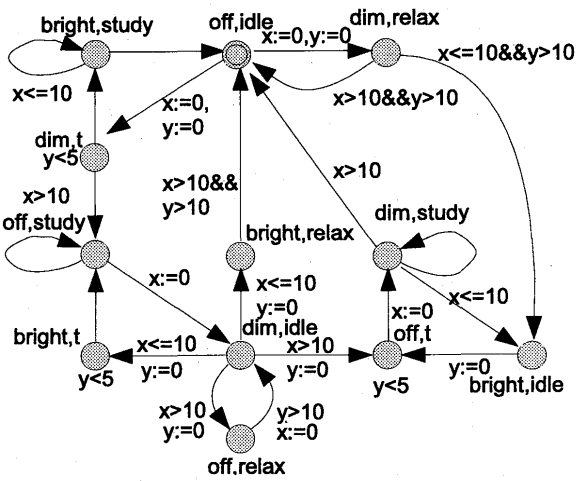

図6　照明システムの積オートマトン

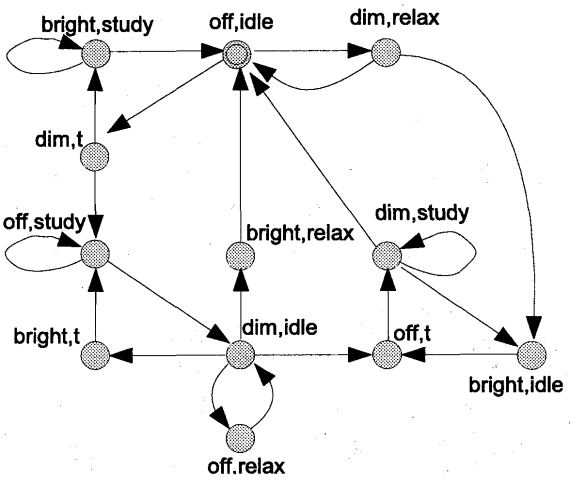

図7　初期抽象モデル

照明システムの二つのオートマトンを結合した積オートマトンを図6に示す．この積オートマトンに対して抽象化を適用する．したがって検査する性質は

$$AG\neg((dim,idle) \vee (bright,idle)) \qquad (3)$$

とする．式3の性質を図6のモデルに対してUPPAALで検査を行うと，validと出力される．したがって図6のモデルは式3の性質を満たしている．

ここからは，図6に示す積オートマトンに対して，本稿で提案している抽象化を適用する．まず，積オートマトンに対して初期抽象化を適用したモデルを図7に示す．初期抽象化では，積オートマトンが持つクロック変数 x，y を省略する抽象化を行っている．

抽象モデルに対して，UPPAALを用いて式3の制約式を検査した結果，抽象モデルに対して2回の改良が行われた．1回目の改良を図8に示す．1回目の改良は反例 $\hat{T}_1 = \langle(off,idle),(dim,relax),(bright,idle)\rangle$ に対して行われた．図6の具象モデル上で反例 $\hat{T}_1$ を再現した時，初期状態 $(off,idle)$ から $(dim,relax)$ に遷移した時に到達可能

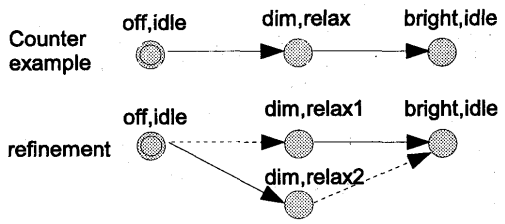

図8 改 良 1

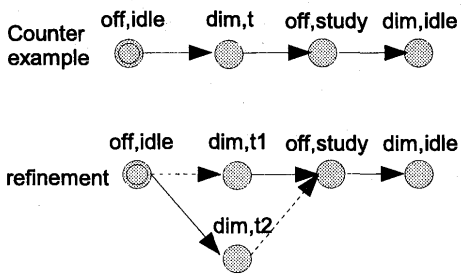

図9 改 良 2

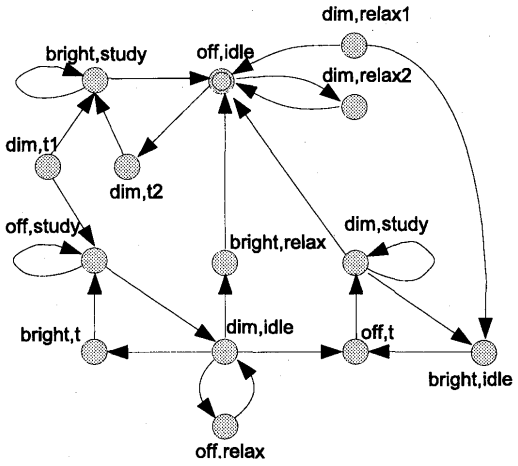

図10 最終的な抽象モデル

なクロック空間は $x = y$ である．しかし $x = y$ であるクロック空間は，遷移 $((dim, relax), (bright, idle))$ のガード $x \leqq 10 \wedge y > 10$ を満たさないため，反例 $\hat{T}_1$ は再現不可能である．この反例を除去するため，図 8 の図下のように改良を行う．図の状態 $(dim, relax1)$ は具象モデル上の状態 $(dim, relax)$ のインバリアントを満たす状態を表現し，状態 $(dim, relax2)$ は $x = y$ を満たすクロック空間のみを表現する抽象状態である．2 回目の改良は図 9 に示す．2 回目の改良は反例 $\hat{T}_2 = \langle (off, idle), (dim, t), (off, study), (dim, idle) \rangle$ を除去するために行われた．

これらの改良をした結果として最終的に得られる抽象モデルを図 10 に示す．最終的な抽象モデルに対して UPPAAL で検査を行ったところ，valid という結果を得ることができた．したがって，この例ではクロック変数を全て省略した抽象モデルを用いて具象モデルと同様の結果が得られることがわかる．

6. 考 察

本稿では，標準的な時間オートマトンが持つクロック変数の抽象化について具体的な手法を提案している．しかし，提案手法では，UPPAAL 拡張時間オートマトンが持つ整数変数に対する抽象化には対応していない．ここでは，整数変数の抽象化についての考察を与える．整数変数は非連続値であり，有限なドメインであるため，プログラム中の述語に着目した述語抽象化 [8] が適用できると考えられる．述語抽象化では，プログラム中の全ての状態を，プログラム中に現れる述語に対する真偽についての同値類に分類し，抽象化を行う．UPPAAL では整数変数に関する述語はロケーションのインバリアントや遷移におけるガードに現れるため，インバリアントやガードに含まれる述語に着目した同値類を形成することで，整数変数に対する抽象化を与えることができると考えられる．

7. おわりに

本論文では，「反例に基づいた抽象化改良ループ」を利用し，時間モデル検査ツール UPPAAL で用いられる拡張時間オートマトンに対する具体的なモデル抽象化手法を提案した．モデル抽象化には，反例に基づいた抽象化改良ループ CEGAR を適用し，抽象化から検査までの全ての工程を自動的に行うことが可能である．また，反例によるモデルの再抽象化においては時間オートマトンの検証で用いられるデータ構造 DBM を活用する工夫を行なっている．

今後の課題としては，提案手法を実用的な例題に対して適用し，本手法の有効性を評価することが挙げられる．また，本稿では UPPAAL 拡張時間オートマトンが持つ整数変数の抽象化には対応しておらず，整数変数の抽象化も課題として挙げられる．その際，6. で述べた手法の適用を考えたい．

文 献

[1] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and V. Helmut: "Counterexample-guided Abstraction Refinement, Computer Aided Verification:12th International Conference,vol1855,pp154-169,July,2000.

[2] E. Clarke, A, Gupta, J. Kukula, and O. Strichman: "SAT based Abstraction-Refinement using ILP and Machine Learning Techniques, Computer Aided Verification:14th International Conference,vol2404,pp695-709,July,2002.

[3] E. Clarke, A. Fehnker, Z. Han, J Ouaknine, O. Stursberg, amd M. Theobald: "Abstraction and Counterexample-guided Refinement in Model Checking of Hybrid Systems, International Journal of Foundations of Computer Science,Vol.4,No.4,2003.

[4] R. Alur: "TimedAutomata, Computer Aided Verification:11th International Conference,vol1633,pp688,July,1999.

[5] J. Bengtsson, and W.Yi: "Timed Automata: Semantics, Algorithms and Tools, Lectures on Concurrency and Petri Nets,vol3098,pp87-124,2004.

[6] G. Behrmann, A. David, and K.G. Larsen : "A Tutorial on Uppaal, Formal Methods for the Design of Real-Time Systems,vol3185,pp200,236,2004.

[7] S. Kemper, and A. Platzer: "SAT-based Abstraction Refinement for Real-time Systems, In Third International Workshop on Formal Aspects of Component Software, 2006.

[8] S. Das, D. L. Dill, and S.Park : "Experience with predicate abstraction, In Computer-Aided Verification, volume1633,pp160-171, 1999.