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

信学技報
TECHNICAL REPORT OF IEICE.

実時間システムを対象としたCEGARによる抽象洗練の並列化手法

田中　俊彰† 　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　〒 565–0871 大阪府吹田市山田丘 1–5
E-mail: †{tstanaka,t-nagaok,okano,kusumoto}@ist.osaka-u.ac.jp

あらまし　時間オートマトンの CEGAR を用いた到達可能性解析を行う手法について，高速化に関する技法を提案する．本稿では、初期抽象を行ったモデルに対して複数の反例を生じさせ、反例による洗練結果を統合することにより，より高速な解析手法を提案する．また，提案した手法に対して並列計算機を用いた評価実験を行い，その台数効果による速度向上が行われているかを調べた．その結果，例では台数効果があることを確認した．
キーワード　時間オートマトン，モデル検査，CEGAR，並列処理

Reachability Analysis for Timed Systems using Parallel Processing

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

† Graduate School of Infomation Science and Texhnology, Osaka University
Yamada-oka 1–5, Suita City, Osaka, 565–0871 Japan
E-mail: †{tstanaka,t-nagaok,okano,kusumoto}@ist.osaka-u.ac.jp

Abstract　This report proposes efficient parallel processing of reachability analysis for timed automaton. Our research group has already proposed CEGAR loop for timed automaton. The report proposes parallel processing version of the CEGAR loop. The new version performs in parallel model checking with different parameters and the master node synthesizes multiple counter examples generated by workers and refines the model. We have prototyped a tool and performed experiments. We found that some of the results show the effeciency of the proposed parallel processing.
Key words　Timed Automaton, Model Checking, CEGAR, Parallel Processing

1. ま え が き

本稿では，時間オートマトンに対してクロック変数を除去する時間抽象化を行う抽象洗練手法 [14] に対する，高速化を目的とした手法の改良を提案する. [14] は Clarke らの Counter-example Guided Abstraction Refinement [1] の枠組みを利用しており，時間オートマトンのモデル検査に対して状態爆発を回避する効果的な手法ではあるが，処理速度が通常のモデル検査に対して劣る．これは，状態数を削減する処理に起因する．

そこで，本研究では [14] の手法を見直し，初期の段階で反例抽出処理を並列に行い，一度に複数の反例から洗練を行う手法を用いて，処理の高速化を目指す．また，実際に提案手法に対して速度向上の効果があるかについて実験を行い，その結果を評価する．

文献 [14] は Clarke らの Counter-example Guided Abstraction Refinement [1] の枠組みを利用しており，時間オートマトンのモデル検査に対して状態爆発を回避する効果的な手法ではあるが，処理速度が通常のモデル検査に対して劣る．これは，状態数を削減する処理に起因する．

そこで，本研究では文献 [14] の手法を見直し，初期の段階で反例抽出処理を並列に行い，一度に複数の反例から洗練を行う手法を用いて，処理の高速化を目指す．また，実際に提案手法に対して速度向上の効果があるかについて実験を行い，その結果を評価する．

以下 2. では，まずモデルとして利用される時間オートマトンについて述べる．また，本稿で利用する CEGAR ループついて簡潔に述べる．3. では，本研究で提案する CEGAR ループの反例抽出並列化手法の概要とアルゴリズムおよび，正当性の略証を与える．4. では，提案する手法に対する評価実験を行い，5. でまとめる．

2. 準　　備

本節では, 時間オートマトンの定義とその意味，そして一般的な CEGAR のアルゴリズムについて述べる．

2.1 時間オートマトン

定義 2.1 (C 上の差分不等式)**.** クロックの有限集合 C 上の差分不等式 E の構文と意味を以下のように与える. $E ::= x - y \sim a \mid x \sim a$, ここで $x, y \in C$, a は実数定数リテラル, $\sim \in \{\leqq, \geqq, <$

, >}. 差分不等式の意味は通常の不等式と同じである.

定義 2.2 (C のクロック制約式)**.** クロックの有限集合 C 上のクロック制約式 $c(C)$ を以下のように与える．クロックの有限集合 C 上の差分方程式全てからなる集合を $c(C)$ とする．ある要素 in_1 と in_2 が $c(C)$ の要素である時，$in_1 \wedge in_2$ も同様に $c(C)$ の要素である.

定義 2.3 (時間オートマトン)**.** 時間オートマトン $\mathscr{A}$ は (A, L, l_0, C, I, T) という以下の 6 個の要素から成る

A: アクションの有限集合

L: ロケーションの有限集合

$l_0 \in L$: 初期ロケーション

C: クロックの有限集合

$I \subset (L \to c(C))$: クロック制約式をロケーションに写像したもので，ロケーションインバリアントと呼ばれる

$T \subset L \times A \times c(C) \times \mathscr{R} \times L$, ここで $c(C)$ はクロック制約式であり, ガードと呼ぶ. $\mathscr{R} = 2^C$: リセットクロック集合.

ある遷移 $t = (l_1, a, g, r, l_2) \in T$ は $l_1 \xrightarrow{a,g,r} l_2$ と表記する.

$\nu : C \to \mathbb{R}_{\geqq 0}$ となる ν をクロックの評価関数と呼ぶ.

$d \in \mathbb{R}_{\geqq 0}$ に対して $(\nu + d)(x) = \nu(x) + d$ と定義する.

$r \in 2^C$ に対して、$r(\nu) = \nu[x \mapsto 0], x \in r$ と定義する．この時，$\nu[x \mapsto 0]$ は各クロック x に対する値を 0 とするクロック評価関数を表すとする.

ν の全てからなる集合を N とする.

定義 2.4 (時間オートマトンの意味)**.** 時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ に対して $\mathscr{A}$ の状態集合を $S = L \times N$ とする．$\mathscr{A}$ の初期状態は $(l_0, 0^C) \in S$ で与えられる．状態遷移 $l_1 \xrightarrow{a,g,r} l_2$ $(\in T)$, に対して、次の二つの遷移が定義される. 前者をイベント遷移，後者を時間遷移と呼ぶ.

$$\frac{l_1 \xrightarrow{a,g,r} l_2, g(\nu), I(l_2)(r(\nu))}{(l_1, \nu) \overset{a}{\Rightarrow} (l_2, r(\nu))}, \qquad \frac{\forall d' \leqq d \quad I(l_1)(\nu + d')}{(l_1, \nu) \overset{d}{\Rightarrow} (l_1, \nu + d)}$$

定義 2.5 (時間オートマトンの意味モデル)**.** 時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ について，初期状態から開始するモデルである $\mathscr{A}$ の意味に従って，無限の遷移を持ったシステムであると定義される．$\mathscr{T}(\mathscr{A}) = (S, s_0, \Rightarrow)$ は $\mathscr{A}$ の意味上のモデルであることを示す.

本論文では，あるロケーション l 上の状態とは，l のインバリアントを満たす ν の任意の意味上の状態 (l, ν) を意味する.

2.2 CEGAR アルゴリズム

モデル抽象化は時に実際のモデルの過度な抽象化を行うことがある．これにより，実際のモデルでは存在しない，誤った反例を生成する可能性がある．文献 [1] は CEGAR(Counterexample-Guided Abstraction Refinement) と呼ばれるアルゴリズムを提案している (図 1).

アルゴリズムにおいて，第一段階として実際のモデルを過度に抽象化する (これを初期抽象化と呼ぶ). 次に，生成された抽象モデルに対してモデル検査を行う．この段階で，抽象モデルが与えられた性質を満たすのであれば，実際のモデルでも性質を満たすと結論付けることができる．これは，抽象モデルが実際のモデルの over-approximation であるためである．もしモデル検査器がモデルは性質を満たさないという結果を返してきた場合，検出された反例が本来実行可能でない反例（偽反例と呼ぶ）であるか否かを検証する段階に入る．(これをシミュレーションと呼ぶ) シミュレーションで，もし反例が実際のモデルでも存在するものであるのならば，ループを終了する．そうでないのならば，間違った反例をなくすような抽象モデルの洗練を行う．これらの段階を繰り返し，正しい出力を得る.

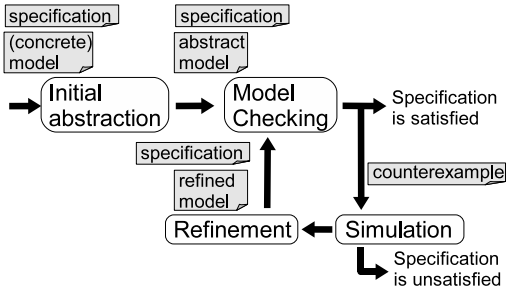

図 1　一般的な CEGAR アルゴリズム
Fig. 1　General CEGAR Algorithm

3. 提案手法

本章では，本稿で提案する確率時間オートマトンの抽象化洗練手法について示す．提案する抽象化洗練手法では，著者らが文献 [14] で提案した時間オートマトンの抽象洗練手法を利用し、さらに高速化のために反例抽出を並列に行っている（図 2）.

（1） 入力として与えられたモデルと満たすべき性質に対して，時間抽象による初期抽象化を行う.

（2） 初期抽象化したモデルをそれぞれのワーカ計算機に配布する．このとき，配布されるモデルは同一のものである.

（3） 抽象モデルを受け取ったワーカ計算機は，モデル検査を行う．この時，性質を満たすのであれば True を，満たさないのであれば反例を出力する.

（4） 反例が出力された場合，反例を基にシミュレーションが行われる．シミュレーションによって，抽象化していないもとモデルでも成立する反例であるのならば False を，元のモデルでは存在しない反例ならばシミュレーション結果を返す.

（5） 各ワーカ計算機で求められたシミュレーション結果を，マスタ計算機で統合する.

（6） マスタ計算機で統合されたシミュレーション結果を元に，抽象モデルを洗練する.

以下，提案手法の各操作について詳細に述べる.

3.1 初期抽象化

初期抽象化では，文献 [14] と同様に，クロック変数に関する制約を全て除去することで，over approximation を満たすように抽象化を行う.

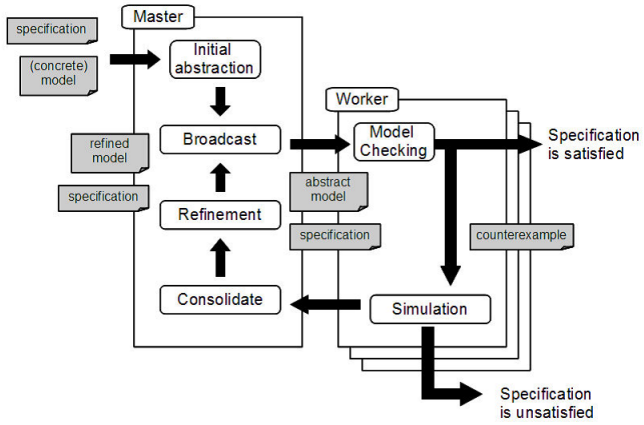

図 2　並列実行環境を利用した CEGAR アルゴリズム
Fig. 2　Our CEGAR Algorithm

定義 3.1 (抽象化関数 h)**.** 時間オートマトン $\mathscr{A}$ とその意味上のモデル $\mathscr{T}(\mathscr{A}) = (S, s_0, \Rightarrow)$ について，抽象化を行う関数 $h : S \to \hat{S}$ を以下のように定義する．

$$h((l, \nu)) = l.$$

その逆関数 $h^{-1} : \hat{S} \to 2^S$ は h を用いて以下のように定義する．$\hat{s} = l$ である抽象モデルに対して $h^{-1}(\hat{s}) = (l, D_{I(l)})$

定義 3.2 (抽象モデル)**.** 時間オートマトン $\mathscr{A} = (A, L, l_0, C, I, T)$ から求められる抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \hat{\Rightarrow})$ とその意味上のモデル $\mathscr{T}(\mathscr{A}) = (S, s_0, \Rightarrow)$ は以下のように定義される．

- $\hat{S} = L$,
- $\hat{s_0} = h(s_0)$
- $\hat{\Rightarrow} = \{(h(s_1), a, h(s_2)) \mid s_1 \overset{a}{\Rightarrow} s_2)\}$.

3.2　抽象モデルの配布

初期抽象化した抽象モデルを，各ワーカ計算機に配布する．この時，配布される抽象モデルは同一のものである．

3.3　モデル検査

モデル検査は、文献 [14] と同様に行う．

定義 3.3 (抽象モデル上の反例)**.** 抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \hat{\Rightarrow})$ 上の反例は $\hat{S}$ の連続する状態と遷移の系列である．ある長さ n の抽象モデルの反例 $\hat{\rho}$ は以下のように表わされる．
$\hat{\rho} = \langle \hat{s}_0 \overset{a_1}{\to} \hat{s}_1 \overset{a_2}{\to} \hat{s}_2 \overset{a_3}{\to} \cdots \overset{a_{n-1}}{\to} \hat{s}_{n-1} \overset{a_n}{\to} \hat{s}_n \rangle$

このとき，到達度解析を行うアルゴリズムの探索戦略を各計算機において変化させることで，抽象モデル上で性質を満たさないと判定された場合に出力される反例が異なることが期待できる．

3.4　シミュレーション

シミュレーションでは, [14] で提案されているシミュレーションアルゴリズムに従って，各ワーカ計算機上で DBM の演算によって到達可能か判定する．

Refinement
Inputs $\mathscr{A}_i, \pi, succ\_list$

$\{\pi = \langle l_0 \overset{a_1, g_1, r_1}{\longrightarrow} l_1 \overset{a_2, g_2, r_2}{\longrightarrow} \cdots \overset{a_n, g_n, r_n}{\longrightarrow} l_n (l_n = e) \rangle\}$
$\{succ\_list = \langle (l_0, D_0), (l_1, D_1), \cdots, (l_k, D_k) \rangle,$
where (l_j, D_j) represents the j-th reachable state set along with π, and l_k is the last location reachable from the initial state. $\}$
$\mathscr{A}_{i+1} := \mathscr{A}_i$
for $j := succ\_list.length$ downto 1 **do**
  $e_j := (l_{j-1}, a_{j-1}, g_{j-1}, r_{j-1}, l_j)$
  $\mathscr{A}_{i+1} := Duplication(\mathscr{A}_{i+1}, succ\_list_j, e_j)$
    {Duplication of the Location and Transitions}
  if $IsRemovable(\mathscr{A}_{i+1}, succ\_list_j, e_j)$ **then**
    $\mathscr{A}_{i+1} := RemoveTransition(\mathscr{A}_{i+1}, e_j)$
      {Removal of Transitions}
    break
  else if $j = 1$ **then**
    $\mathscr{A}_{i+1} := DuplicateInitialLocation(\mathscr{A}_{i+1}, (l_0, D_0))$
      {Duplicate the initial location and transitions from the initial location}
  end if
end for
return $\mathscr{A}_{i+1}$

図 3　アルゴリズム 1 ： 洗練アルゴリズム
Fig. 3　algorithm 1: Refinement Algorithm

3.5　シミュレーション結果の統合

各ワーカ計算機上で計算されたシミュレーション結果を，マスター計算機送り，統合する．このとき，モデル検査やシミュレーションでの終了判定も行う．

3.6　抽象モデルの洗練

抽象モデルの洗練は，文献 [14] で述べられている手法を基にしている．

3.6.1　洗練時に行われる処理

抽象モデルを洗練するとき，以下の 3 つの処理が行われている．

- 状態を複製する
- 状態間の遷移を追加する
- 状態間の遷移を除去する

このとき，状態の複製，遷移の複製，除去に関しての条件は文献 [14] において定義されている (アルゴリズム 1:図 3).

ここで示されているアルゴリズム 1 を，提案手法に対応させるために以下のように変更したアルゴリズム 1 ’(図 4) を与える．

ある反例の集合 $\hat{P}$ に対して，順にアルゴリズム 1 を実行する．その結果は時間オートマトン $\mathscr{A}$ に反映される．もし仮に，反例の badstate を解消できない場合は，アルゴリズム 1 を適用せず，$\hat{P}$ の次の反例に対して処理を繰り返す．

3.6.2　反例の重複

抽象モデルを洗練するときに問題となるのは，複数の反例を抽象モデルの洗練に適用した際，反例の選択順序により誤った洗練を行わないことを保証することである．

RefinementOfCEs

Inputs $\mathscr{A}_i, P$

$\{P = \langle \rho_0, \rho_1, \cdots, \rho_k) \rangle\}$

$\mathscr{A}_{i+1} := \mathscr{A}_i$

for $j := P.length$ downto 1 **do**

$\mathscr{A}_{i+1} := Refinement(\mathscr{A}_{i+1}, \rho_j)$

end for

return $\mathscr{A}_{i+1}$

図 4　アルゴリズム 1'： 洗練アルゴリズム (複数パス)

Fig. 4　algorithm 1' : Refinment Algorithm of CEs

まず，反例の重複について, 定義 3.4 で与える．

定義 3.4 (反例の重複)**.** ある反例 $\hat{\rho}_1$ と $\hat{\rho}_2$ が重複しているということは，反例 $\hat{\rho}_1$ と $\hat{\rho}_2$ が共通する *1* つ以上の初期状態以外の状態 $\hat{s}$ を保持していることである．反例の集合が重複していないとは，その集合のどの *2* つをとっても重複していないことを意味する．

定義 3.5 (badstate)**.** ある反例 $\hat{\rho}$ に含まれる遷移において，時間制約を満たさない最初の抽象モデル上の状態のことを *badstate* とする．

定義 3.6. ある抽象モデル $\hat{M}$ と，与えられた反例の集合 $\hat{P}$ に対して，大域的に正しい洗練 $\hat{M}'$ とは，$\hat{P}$ が $\hat{P}_1(\neq \emptyset), \hat{P}_2$ に分割でき，$\hat{P}_1$ に含まれる反例に対しては *badstate* が解消され，$\hat{P}_2$ の中の反例は $\hat{M}'$ で実行不能な洗練のことである．

定理 3.1. 到達可能性解析においては反例集合の反例をどのような順番でアルゴリズム *1'* を適用しても大域的に正しい洗練である．

証明 3.1. 以下の定理 *3.2*，*3.3* より明らかである．　□

定理 3.2. 重複のない反例の集合に対して，到達可能性解析においては反例集合の反例をどのような順番でアルゴリズム *3* を適応しても大域的に正しい洗練になる．

定理 3.3. 重複のある反例集合に対して，到達可能性解析においては反例集合の反例をどのような順番でアルゴリズム *1'* を適応しても大域的に正しい洗練になる．

定理 3.2 は自明であるため，ここでは定理 3.3 の略証を与える．

略証　反例集合内の反例をすでに n 個についてアルゴリズム 1 'を適用した状況を考える．そのときの抽象モデルを $\hat{M'}_n$ で表す．$n+1$ 番目の反例を任意に選んだとき以下の場合が考えられる．

（1）　$\hat{M'}_n$ 上でその反例は実行不能である．

（2）　$\hat{M'}_n$ 上でその反例は実行可能である．

前者の場合はアルゴリズム 1 'を適用したのちの $\hat{M'}_{n+1}$ は $\hat{M'}_n$ と同形であり，$\hat{M'}_0$ に対して正しい洗練である．

後者の場合は，その反例の badstate がアルゴリズム 1 で新たに実行不可能にある．よって，$\hat{M'}_{n+1}$ は $\hat{M'}_0$ に対して大域的に正しい洗練である．なお，反例集合中最初に選んだ反例は少

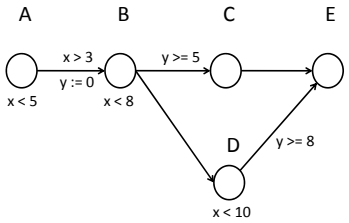

図 5　元モデルの時間オートマトン

Fig. 5　A Timed Automaton

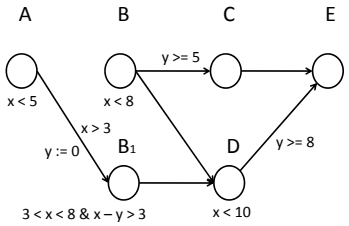

図 6　図 3 において B-C から洗練して得られた時間オートマトン

Fig. 6　A Timed Automaton Refinement of B-C Trantision

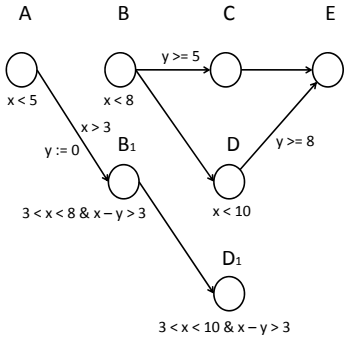

図 7　図 3 において D-E から洗練して得られた時間オートマトン

Fig. 7　A Timed Automaton Refinement of D-E Trantision

なくともアルゴリズム 1 が適用されるため，ρ_1 は空ではない．□

なお，反例の適用順序により一般に得られる抽象モデルは異なり得る可能性がある．また，　逆に適用順に関わらず同一の結果を生み出すこともある．その例を 3.6.3 で示す．

3.6.3　反例解消の例

実際の例を用いて，提案手法の処理について説明する．例えば，図 5 のようなモデルがあるとする．このモデルは，状態 B から状態 C への遷移と，状態 D から状態 E への遷移に対して時間制約を満たさない．このようなモデルに対して，提案手法を適応した場合についての例を示す．

まず、このモデルを提案手法に適応した場合，得られる反例は以下の二つが考えられる．

- $A \to B \to C \to E$
- $A \to B \to D \to E$

この二つの反例が得られたと仮定して，洗練手法を適応する．

まず，反例 $A \to B \to C \to E$ から洗練を行った場合の例を示す．

状態 B から状態 C への遷移は時間制約を満たさないため，

まずは時間制約を満たす状態 B_1 を新たに生成する．次に，生成された状態 B_1 をから状態 D に遷移を生成し，さらに状態 A から状態 B_1 への遷移を生成する．最後に，状態 A から状態 B への遷移を削除することで洗練が終了する（図 6）．

反例 $A \to B \to D \to E$ に対する洗練を行う．前回と同じように時間制約を満たす状態 D_1 を新たに生成する．次に，状態 D_1 の時間制約を満たす状態 B_1 の生成を行うが，前回生成した状態 B_1 が状態 D_1 の時間制約を満たすため生成しない．最後に，状態 B_1 から状態 D への遷移を削除する (図 7).

次に，反例 $A \to B \to D \to E$ から洗練を行った場合の例を示す.

まず，状態 D から状態 E への遷移は時間制約を満たさないため，時間制約を満たす状態 D_1 を新たに生成する．このとき，状態 B は状態 D_1 の時間制約を満たさないため，同時に状態 B_1 も生成する．最後に，状態 A から状態 B への遷移を削除して，洗練を終了する (図 7).

次に，反例 $A \to B \to C \to E$ に対する洗練を行うが，状態 D から状態 E への遷移に対する洗練を行った段階で，状態 A から状態 C に到達するパスは存在しないため，何もせずに終了となる．この場合，反例 $A \to B \to C \to E$ と反例 $A \to B \to D \to E$ の洗練の順序を入れ替えたとしても，最終的には同じ洗練結果が得られることがわかる.

4. 評 価 実 験

本章では提案手法について評価実験を行う.

4.1 実 験 環 境

4.1.1 計算機環境

提案手法を実行する並列計算環境を以下に示す.

マスター計算機

CPU : Intel(R) Core^{TM}2 Duo

CPU L7700 1.80GHz

メモリ : 2.00GB　OS : Ubuntu 10.0.4

ワーカ計算機（14 台）

CPU : Dual Core AMD Opteron^{TM}

Processor 2210 HE 1.80GHz

メモリ : 6.00GB　OS : CentOS 5.4

また，マスター・ワーカ間の通信には Java の RMI フレームワークを利用した.

4.1.2 モデル検査ツール

今回の実験においてモデル検査は，モデル検査ツール UPPAAL [7] のモデル検査モジュールを利用する．探索戦略は深さ優先の最適化探索とし，出力するパスはランダムに設定する．このことで今回目的とする複数種類の反例を出力させる.
反例抽出処理がランダムであるため，今回は出力の均一化のために 1 つの事象につき 5 回の実験を行い，その平均を取って実験結果とする.

4.1.3 対象とした例題

この実験では，Fischer の相互排除プロトコルを利用する．これは実時間モデル検査のアルゴリズムの比較によく用いられる.

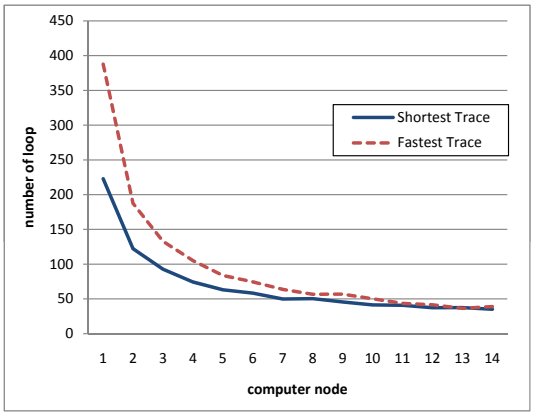

図 8　ループ回数
Fig. 8　Number of iterations

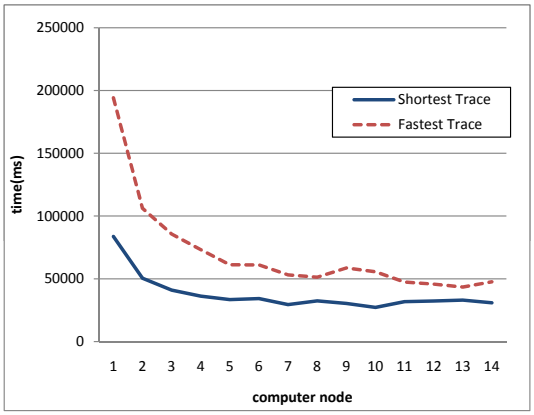

図 9　処 理 時 間
Fig. 9　Excute time

このプロトコルでは，n 個のプロセス間で一つしかない資源の使用を管理するプロトコルである．また，モデルの生成についても，UPPAAL が行う.

4.1.4 探索戦略について

今回の実験では，出力される反例の性質の違いに対してもその違いが現れるかについて実験を行った．反例の選択としては，探索された反例の中で最も早く発見された反例を返す Fastest Trace と，発見された反例の中で最短の長さの反例を返す Shortest Trace の二つについて 実験を行った.

4.2 実 験 結 果

4.2.1 ループ回数

まず，ワーカ計算機を増やした時のループ回数に対する台数効果について傾向を調べる．ここで、ループ回数とは提案手法の処理が行われた回数のことを示している．図 8 は，ループ回数に対する台数効果を表している．利用しているワーカ計算機の台数が増えるほど，図 8 ではループ回数が減少していることが解る.

4.2.2 実 行 時 間

次に，実行時間に対数台数効果について傾向を調べる.
図 9 は，処理時間に対する台数効果を表している．図 8 とは違い，7 台目辺りから処理時間が横ばいであることが図 9 から解

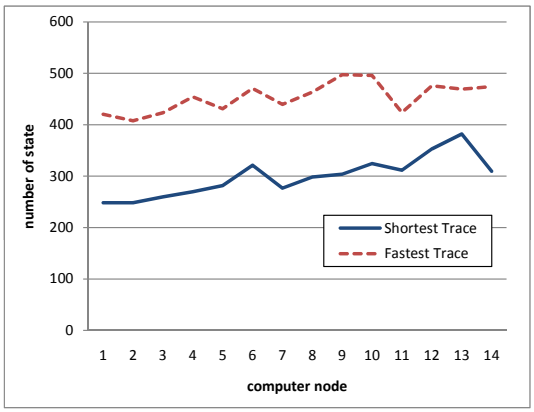

図 10　状　態　数
Fig. 10　Number of states

る．ループ回数は順調に減少しているため，1 回のループ辺りの探索時間が増大していることが考えられる．

4.2.3　生成状態数

最後に，生成状態数の増加量について傾向を調べる．

図 10 は，台数に対する生成状態数の数を表している．新たに状態が生成される場合は主に洗練時であるため，生成状態数が増えることは抽象モデルに対して不必要な洗練が行われている可能性がある，ということを示している．反例パスの探索手法により生成状態数に差があるが，これは出力される反例が短いほうが生成される状態数が少なくなることを示していると考えられる．

4.3　実験結果の考察

実験結果に対する考察を行う．

実験の結果，ループ回数はワーカ計算機を増やすことによる台数効果が出ていることが分かる．しかしながら，図 9,10 が示すように，ワーカ計算機の台数を増やすことにより本来不要な洗練が行われ，そのためにモデルが不必要に肥大化し，結果 1 ループ辺りの処理時間が増大することで台数効果を打ち消している可能性も示している．

また，出力される反例が短い場合，新たに生成される状態数を抑えることが可能であるため探索速度の向上するという結果が得られた．

今回は反例を抽出するモデル検査の処理がランダム出力であったため，詳しい解析はできなかった．この事象をさらに調査するために，今後はランダム性のないモデル検査手法が必要となる．

5.　おわりに

本稿では，時間オートマトンに対する時間抽象化を用いた洗練手法を拡張し，抽象モデルに対して複数の異なった反例抽出，洗練を行うことで手法の高速化を提案，実験を行った．

今後の課題としては，今回の評価実験で見られた，実行時間に対する台数効果の阻害要因を調べるために，$k-$ 最短路による到達度解析を行うモデル検査ツールを実装し，その挙動について解決していきたい．

また，提案手法では，洗練を逐次的に行っていたが，シュミレーション結果から，重複するような洗練，またはある洗練の部分集合となるような洗練をを排除することで，洗練を行う時間を削減し，更なる高速化手法を提案していきたい．

謝辞　本研究の一部は科学研究費補助金基盤 C(21500036) と文部科学省「次世代 IT 基盤構築のための研究開発」(研究開発領域名：ソフトウェア構築状況の可視化技術の普及) の助成による．

文　献

[1] E M. Clarke, O. Grumberg, S. Jha, Y. Lu, and V. Helmut: “Counterexample-guided abstraction refinement for symbolic model checking,” Journal of the ACM, vol.50(5), pp752-794, 2003.

[2] E M. Clarke, A. Gupta, J. Kukula, and O. Strichman: “SAT based Abstraction-Refinement using ILP and Machine Learning Techniques,” In Proc. of the 14th Int. Conf. on Computer Aided Verification, Lecture Notes in Computer Science, vol.2404, pp.695-709, 2002.

[3] E M. Clarke, A. Fehnker, Z. Han, J Ouaknine, O. Stursberg, and M. Theobald: “Abstraction and Counterexample-guided Refinement in Model Checking of Hybrid Systems,” In Int. Journal of Foundations of Computer Science, vol.14(4), 2003.

[4] R. Alur: “Techniques for Automatic Verification of Real-Time Systems,” PhD thesis, Stanford University, 1991.

[5] R. Alur, C. Courcoubetis, and D. L. Dill: “Model-checking for real-time systems,” In Proc. of the 5th Annual Symposium on Logic in Computer Science, IEEE, pp.414-425, 1990.

[6] S. Das, D. L. Dill, and S.Park : “Experience with predicate abstraction,” In Proc. of the 11th Int. Conf. on Computer Aided Verification, Lecture Notes in Computer Science, vol.1633, pp.160-171, 1999.

[7] J. Bengtsson, and W .Yi: “Timed Automata: Semantics, Algorithms and Tools,” In Lectures on Concurrency and Petri Nets, Lecture Notes in Computer Science, vol.3098, pp.87-124, 2004.

[8] F. Wang, K. Schmidt, G D. Huang, F. Yu, B Y. Wang: “Formal Verification of Timed Systems: A Survey and Perspective,” In Proc. of the IEEE, vol.92, No.8, pp.1283-1307, 2004.

[9] G. Behrmann, A. David, and K G. Larsen: “A Tutorial on UPPAAL,” In Proc. of the 4th Int. School on Formal Methods for the Design of Computer, Communication, and Software Systems, Lecture Notes in Computer Science, vol.3185, pp.200-236, 2004

[10] A. David, J. Hakansson, K G. Larsen, and P. pettersson: “Model Checking Timed Automata with Priorities using DBM Subtraction,” In Proc. of the 4th Int. Conf. on Formal Modelling and Analysis of Timed Systems, Lecture Notes in Computer Science, vol.4202, pp.128-142, 2006

[11] H. Nakajima and Y. Kameyama: “Improvement on Real-Time Model Checking using Abstraction-Refinement (In Japanese),” In Transactions of Information Processing Society of Japan, vol.45, No.SIG12 (PRO23), pp.11-24.

[12] S. Kemper, and A. Platzer: “SAT-based Abstraction Refinement for Real-time Systems,” In Proc. of the Third Int. Workshop on Formal Aspects of Component Software, vol.182, pp.107-122, 2006.

[13] H. Dierks, S. Kupferschmid, and K G. Larsen: “Automatic Abstraction Refinement for Timed Automata,” In Proc. of the 5th Int. Conf. on Formal Modelling and Analysis of Timed Systems, Lecture Notes in Computer Science, vol.4763, pp.114-129, 2007.

[14] T. Nagaoka, K. Okano, and S. Kusumoto: “An Abstraction refinement technique for timed automata based on Counterexample-Guided Abstraction Refinement Loop,” IEICE Transactions on Information and Systems, vol.E93-D, No.5, pp.994-1005, 2010.