I-06

時間システムの到達可能性解析の並列手法と評価実験

Reachability Analysis for Timed Systems using Parallel Processing and its Experimental Results

田中 俊彰† 長岡 武志† 岡野 浩三† 楠本 真二†

Toshiaki Tanaka Takeshi Nagaoka Kozo Okano Shinji Kusumoto

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の検証を行う．本稿では，著者らが提案している手法に対して，複数の特性の異なるモデルを用いて評価実験を行う．提案されている手法では，ループ初期に反例抽出を平列に行うことで，反例による洗練結果を統合することで高速化を目指している．実験では，モデルの複雑度，反例抽出時に行われる反例探索戦略の変更によって，提案されている手法の優位性や問題点についての評価，考察を行う．

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した抽象洗練手法[15]に対する高速化を目的とした手法について，実験による提案手法の特性と問題点の評価を行う．文献[15]では Clarke らの Counter-Example Guided Abstraction Refinement[1]の枠組みを利用し，時間オートマトンのモデル検査を行う．さらに，文献[16]では手法の高速化を目指し，反例抽出処理を並列で行う手法を提案し，簡単な実験を行っている．

本稿では，文献[16]の手法に対して，複数のモデルでの実験を行い，提案されている手法の有用性と問題点の考察を行う．また，反例出力のための反例選択戦略の変更がどのように手法に影響を与えるかについても言及する．

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 アルゴリズムについて簡潔に述べる．次の 3．では，実験の対象となる文献[16]で提案されている手法について説明する．4．では手法について評価実験を行い，本手法の特性や問題点について考察を行い，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 上の差分方程式すべてからなる集合 $c(C)$ とする．ある要素 in_1 と in_2 が $c(C)$ の要素であるとき，$in_1 \wedge in_2$ 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 $\mathcal{A}$ は

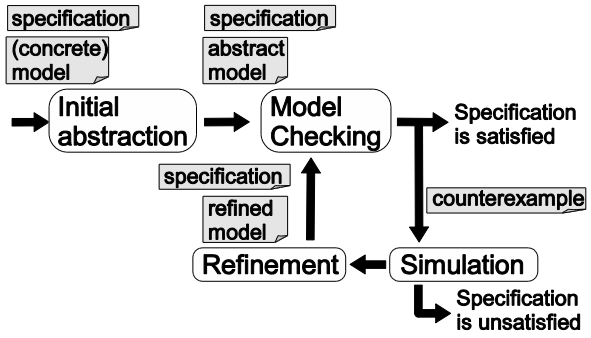

図 1　一般的な CEGAR アルゴリズム
Fig.1　General CEGAR Algorithm

(A, L, l_0, C, I, T) という以下の 6 個の要素からなる

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 ：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

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

定義 2.4(時間オートマトンの意味)．時間オートマトン $\mathcal{A} = (A, L, l_0, C, I, T)$ に対して $\mathcal{A}$ の状態集合を $S = L \times N$ とする．$\mathcal{A}$ の初期状態は $(l_0, 0^c) \in S$ で与えられる．状態遷移 $l_1 \overset{a,g,r}{\to} l_2 (\in T)$ 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．

$$\frac{l_1 \overset{a,g,r}{\to} l_2, g(v), I(l_2)(r(v))}{(l_1, v) \overset{a}{\Rightarrow} (l_2 . r(v))}, \quad \frac{\forall d' \leqq d \quad I(l_1)(v + d')}{(l_1, v) \overset{d}{\Rightarrow} (l_1, v + d)}$$

†大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

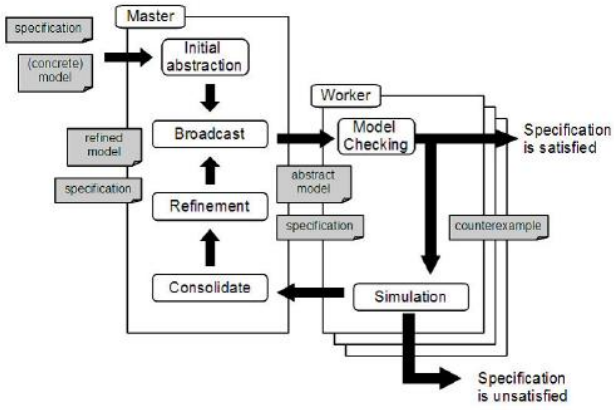

図 2　並列環境を利用した CEGAR アルゴリズム
Fig. 2　Our CEGAR Algorithm

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 $\mathcal{A}=(A,L,l_0,C,I,T)$ について，初期状態から開始するモデルである $\mathcal{A}$ 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．

$\mathcal{T}(\mathcal{A})=(S,s_0,\Rightarrow)$ は $\mathcal{A}$ 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 v の任意の意味上の状態 (l,v) 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．

3. 提案手法

本章では，文献[16]で提案した時間オートマトンの抽象化洗練手法について述べる．この抽象化洗練手法では，著者らが文献[15]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，抽象化していない元のモデルでも成立する反例であるのならば False を，元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結果を，マスター計算機で統合する．

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

以下，文献[16]の各操作について詳細を述べる．

3.1　初期抽象化

初期抽象化では，文献[15]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．

定義 3.1（抽象化関数 h）．時間オートマトン $\mathcal{A}$ とその意味上のモデル $\mathcal{T}(\mathcal{A})=(S,s_0,\Rightarrow)$ 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．

$$h((l,v))=l.$$

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 $\mathcal{A}=(A,L,l_0,C,I,T)$ 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 $\mathcal{T}(\mathcal{A})=(S,s_0,\Rightarrow)$ 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．このとき，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[15]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

ように表される．

$$\hat{\rho} = \left\langle \hat{s}_0 \xrightarrow{a_1} \hat{s}_1 \xrightarrow{a_2} \hat{s}_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} \hat{s}_{n-1} \xrightarrow{a_n} \hat{s}_n \right\rangle$$

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ワーカ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，文献[15]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どうかを判定する．

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，マスター計算機に送り統合する．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[15]で述べられている手法を用いる．

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[15]において定義されている(図3：アルゴリズム1)．

ここで示されているアルゴリズム1を，提案手法に体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 $\hat{\mathrm{P}}$ に対して，順にアルゴリズム1を実行する．その結果を時間オートマトン $\mathcal{A}$ に反映する．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，$\hat{\mathrm{P}}$ の次の反例に対して処理を行う．

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応した際，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．

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，$\hat{\rho}_1$ と $\hat{\rho}_2$ が共有する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．

Refinement

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

$\{\pi = \langle l_0 \xrightarrow{a_1, g_1, r_1} l_1 \xrightarrow{a_2, g_2, r_2} \cdots \xrightarrow{a_n, g_n, r_n} l_n (l_n = e) \rangle\}$

$\{succ\_list = \langle (l_0, D_0), (l_1, D_1), \cdots, (l_k, D_k) \rangle,$

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

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

これより定理を示す．なお，定理の証明については論文[16]で行っている．

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．

定理 3.2．重複のない反例集合に対して，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’に適用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’に適用しても大域的に正しい洗練になる．

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うる可能性がある．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．

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'：Refinement Algorithm of CEs

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

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) CoreTM 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 OpteronTM 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．

4.2　モデル検査ツール

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．反例の探索戦略は深さ優先の最適化探索とし，出力する反例はランダムに設定する．このことで今回の目的としている複数種類の反例を出力させる．

モデル検査による反例抽出処理がランダムで行われるため，出力を均一化するために 1 つの事象につき 5 回ずつ実験を行い，その平均を実験結果として用いる．

4.3　対象とした例題

Fischer の相互排除プロトコル[9]と Gear Controller[14]をそれぞれ利用する．

4.3.1 Fischer の相互排除プロトコル

Fischer の相互排除プロトコルは，n 個のプロセス間で 1 つしかない資源の使用を管理するプロトコルである．1 つのプロセスが 4 つのロケーションしか持たないため，比較的複雑度の低いモデルであるといえる．また，各プロセスがシンメトリな構造を持つため，出力される反例が複数あることが期待できる．そのため，文献[16]の手法に適していると判断した．

4.3.2 Gear Controller

Gear Controller モデルは自動車などの乗り物に用いられるギアの操作をモデル化したものである．このモデルは 5 つの異なる構造をしたプロセスから構成される．そのため，システム全体の複雑度が高く，ロケーション数も多い．Fischer の相互排除プロトコルとは対象的であり，反例出力が複数ない可能性があるため，文献[16]の手法の性能評価に適していると判断した．

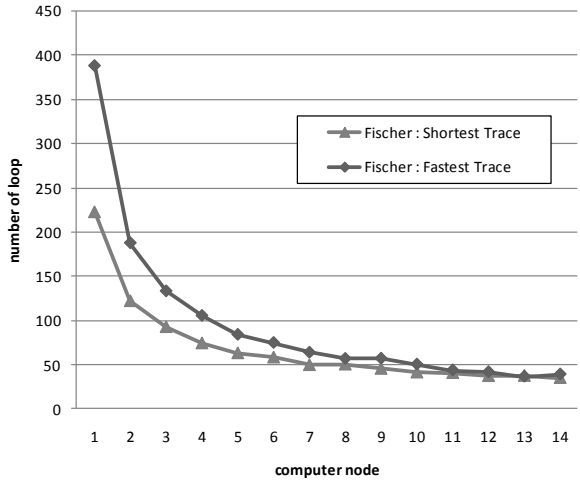

図 5　ループ回数：Fischer
Fig.5　Number of Iterations : Fischer

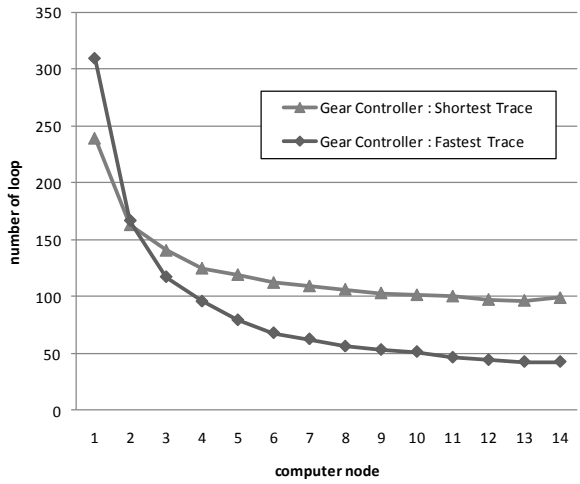

図 6　ループ回数：Gear Controller
Fig.6　Number of Iterations : Gear Controller

モデルの生成については，UPPAAL のモジュールが行う．

4.4　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効果が表れるかどうかについて実験を行った．反例の探索戦略としては，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長さが最短の反例を返す Shortest Trace の 2 つについて評価実験を行った．

4.5.　実験結果

4.5.1　ループ回数

まず，ワーカ計算機を増やした際のループ回数に対する台数効果について調べる．ここで，ループ回数とは提案手法の処理が行われた回数を示している．図 5，6 は，ループ回数に対する台数効果を表している．Fischer の相互排除プロトコルや Gear Controller の Fastest Trace ではワーカ計算機の台数に応じてループ数が減少しているが，

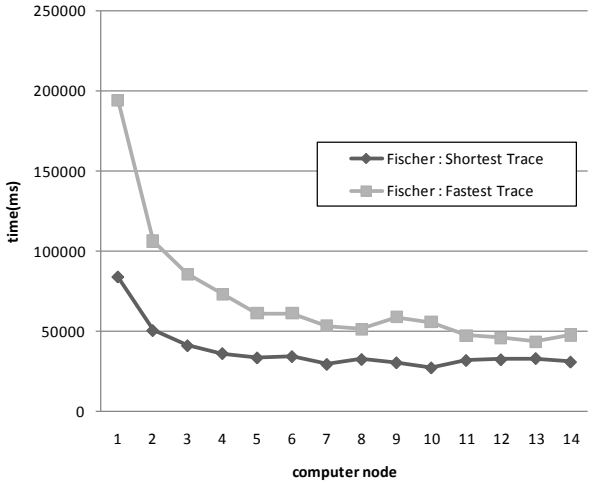

図 7　実行時間：Fischer
Fig.7　Excute time : Fischer

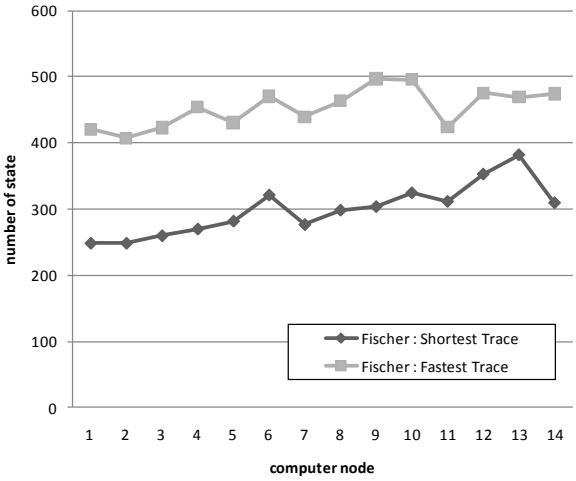

図 9　状態生成数：Fischer
Fig.9　Number of States : Fischer

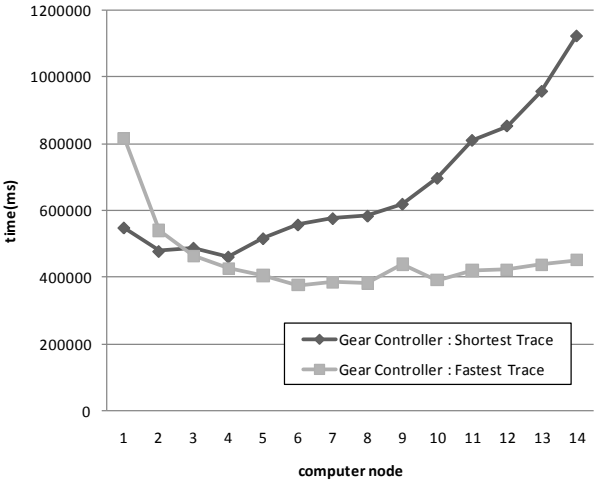

図 8　実行時間：Gear Controller
Fig.8　Excute time : Gear Controller

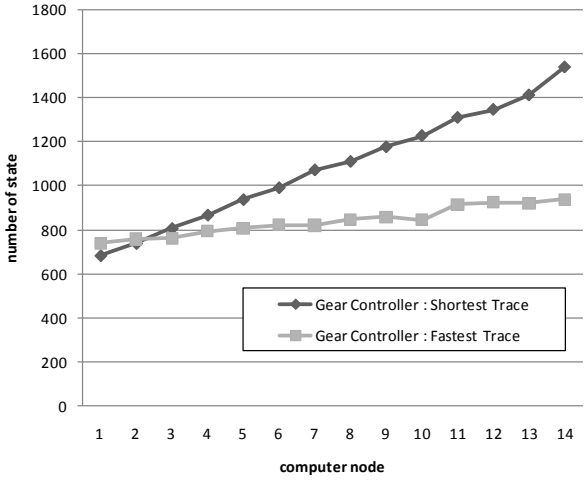

図 10　状態生成数：Gear Controller
Fig. 10　Number of States : Gear Controller

Gear Controller の Shortest Trace のみ，途中からワーカ計算機の台数を増やしてもループ回数が減少していないことがわかる．

4.5.2　実行時間

次に，実行時間の台数効果について調べる．

図 7，8 では，実行時間に対する台数効果を表している．Fi-scher の相互排除プロトコルはどちらも途中から実行時間が横ばいになる傾向が見られた．一方，Gear Controll-er は Fastest Trace では実行時間が途中から横ばいになるが，Shortest Trace では 4 台から増加傾向になることがわかる．

4.5.3　生成状態数

最後に，生成状態数の増加量について調べる．

図 9，10 は，生成状態数に対する台数効果を表している．状態は洗練時に複製されるため，生成状態数が増加することは抽象モデルに対して不必要な洗練が行われ，結果実行時間の増加に繋がるということを示している．

図 9，10 から，Fischer の相互排除プロトコルや Gear Controller の Fastest Trace では緩やかな増加傾向が見られたが，Gear Controller の Shortest Trace では強い比例の関係が見られた．

4.5　実験結果の考察

実験結果に対する考察を行う．

まず，Fischer の相互排除プロトコルについてであるが，ループ回数は台数効果が出ていることがわかる．これは，抽象モデルでの反例が複数出力されていることが期待できる．しかしながら，図 7，9 から，状態数が緩やかに増加しており，また実行時間も低下していない．これは，1 回のループにかかる時間が増加し，結果実行時間が横ばいになるということが考えられる．

次に，Gear Controller についてであるがこちらは Faste-st Trace と Shortest Trace で結果が大きく異なった．Fa-stest Trace では Fischer の相互排除プロトコルとほぼ同等の傾向が表れていたが，Shortest Trace では実行時間の増加などが傾向として現れた．これは，モデルの複雑度

が高く，長さを最短のものと制限した場合に出力される反例が Fischer の相互排除プロトコルよりも少なかったのではないかと推測される．ループ数の削減もほとんど起きていないため，同一の反例を複数のワーカ計算機が返す結果となり，処理の冗長化を招き実行時間が増大したと考えられる．実際，Fastest Trace の場合はある程度 Fischer の相互排除プロトコルと同じ傾向のグラフが表れたため，反例の長さに制約を加えなければ、複数の反例が出力されることが期待できる．
実験では，モデル検査の処理がランダム性に依存するため，詳しい解析ができなかった．今後，実行時間の短縮のためには反例抽出時に反例の有用性についての議論を行う必要がある．また，本稿で評価した手法ではシミュレーション結果の同期をとって取得しているため，ワーカ計算機を増やした際に発生するマスター・ワーカ間の通信オーバーヘッドについても今後解消すべき課題である．

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の並列化について，評価実験を行った．
今後の課題としては，今回の評価実験で得られた，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–最短路探索手法による到達度解析を行い，抽出された反例の評価を行う．また，マスター・ワーカ間の通信を非同期にするなど，並列に実行できる部分について速度向上について改良していきたい．

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), pp.752-794, 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, No.4, pp.583-604, 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 realtime 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]. M. Lindahl, P. Pettersson, and W. Yi: “Formal Design and Analysis of a Gear Controller,”In Proc. of the 4th International Workshop on Tools and Algorithms for the Constraction and Analysis of Systems, vol.1384, pp.281-297, 1998.
[15]. 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.
[16]. 田中俊彰, 長岡武志, 岡野浩三, 楠本真二: ”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” 信学技報, to appear, (2010).
[17]. A. Gupta and O. Strichman: “Abstraction Refinement for Bounded Model Checking,” In Proc. of the 17th Int. Conf. on Computer Aided Verification Lecture Notes in Computer Science, vol. 3576, pp.112-124, 2005.
[18]. G.Behrmann: “Distributed reachability analysis in timed automata,” In Int. Journal on Software Tools for Technology Transfer, vol. 7, No.1, pp.19-30, 2005.