OUKA
Osaka University Knowledge Archive

Title	時間システムを対象とした到達可能性解析の高速化手法の提案
Author(s)	田中, 俊彰; 長岡, 武志; 岡野, 浩三 他
Citation	情報処理学会研究報告. ソフトウェア工学研究会報告. 2010, 2010-SE-170(15), p. 1-8
Version Type	VoR
URL	https://hdl.handle.net/11094/50246
rights	ここに掲載した著作物の利用に関する注意 本著作物の著作権は情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。
Note	

Osaka University Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

Osaka University

Vol.2010-SE-170 No.15
2010/11/12

時間システムを対象とした 到達可能性解析の高速化手法の提案

田中俊彰[†1]　長岡武志[†1]
岡野浩三[†1]　楠本真二[†1]

時間オートマトンの CEGAR を用いた到達可能性解析について，高速化手法の提案とその評価実験を行う．本稿では，著者らが既に提案している時間オートマトンの CEGAR を用いた到達度解析手法に対して，反例抽出部分の処理を並列化することで処理の高速化を目指す．また，手法に対して複数の時間システムを用いて実験を行った．実験で得られた結果を基にして手法の優位性や問題点についての評価を行い，さらにいくつかの改良点について考察を行う．

Fast Method of Reachability Analysis for Timed Systems

TOSHIAKI TANAKA,[†1] TAKESHI NAGAOKA,[†1]
KOZO OKANO[†1] and SHINJI KUSUMOTO[†1]

This report proposes an efficient method of reachability analysis for timed systems and experimental results. Our research group has already proposed CEGAR loop for timed automaton and parallel processing of the CEGAR loop for the timed system.We have prototyped a tool and performed experiments. The report mainly focuses on the experimental results and some improvements.

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[15] に対する，高速化を目的とした手法の改良を提案する．[15] は Clarke らの Counter-example Guided Abstraction Refinement[1] の枠組みを利用しており，時間システムを対象とした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，大規模なモデルに対しては処理速度が通常のモデル検査に対して劣る傾向にある．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[15] の手法を基に，初期段階での反例を複数抽出し，抽出された反例に基づく洗練結果を統合することで手法の高速化を目指す．また，並列計算機を用いて手法の高速化に取り組んだ文献[18] の手法に対して，複数の時間システムを用いて実験を行い，実験結果から提案されている手法の有用性を示すとともに，問題点の考察を行い，文献[18] の手法の改良を行う．

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について述べ，4 で手法に対する評価実験を行う．5 で手法問題点とその改良に対する考察を行い，6 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, $\sim \in \{\leq, \geq, <, >\}$. 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
$I \subset (L \to c(C))$: ロケーションからクロック制約式への写像，ロケーションインバリア

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

© 2010 Information Processing Society of Japan

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

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

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.

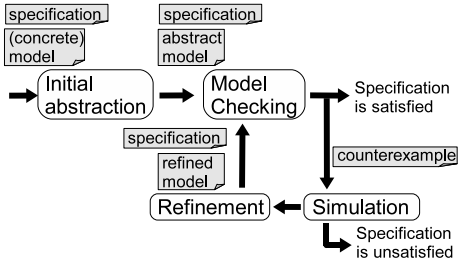

図 1 一般的な CEGAR アルゴリズム
Fig. 1 General CEGAR Algorithm

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[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.

ⓒ 2010 Information Processing Society of Japan

Vol.2010-SE-170 No.15
2010/11/12

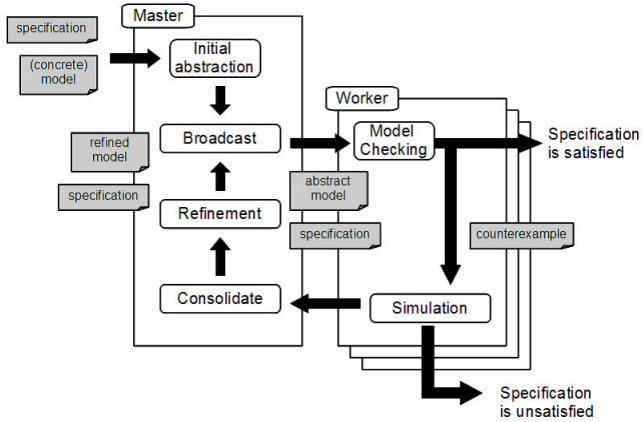

図 2 並列実行環境を利用した CEGAR アルゴリズム
Fig. 2 Our CEGAR Algorithm

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

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

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[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 $\hat{\rho}$ は以下のように表わされる．

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

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，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[10] の演算によって到達可能か判定する．

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[15] 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

© 2010 Information Processing Society of Japan

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

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．

まず，反例の重複について，定義 3.4 で与える．

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

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．

以下の定理は文献[18] で与えられており，並列化された提案手法の正しさの一部を保証するものである．

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．

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．

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，逆に適用順にかかわらず同一の結果を生み出すこともある．

ⓒ 2010 Information Processing Society of Japan

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) 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．

4.2 モデル検査ツール

モデル検査は，モデル検査ツール UPPAAL[9] 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，同階層の状態に対する選択戦略はランダムに設定する．このことで今回のアルゴリズムの主目的である異なる複数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 つのプロセスはたかだか 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であると言える．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[18] 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照的であり，シンメトリな構造を持たないため出力される反例が複数ない可能性があるため，手法の性能を評価する上で適していると判断した．

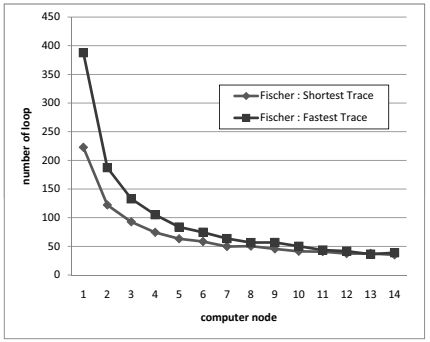

図 5 ループ回数 ： Fischer
Fig. 5 Number of Iterations : Fischer

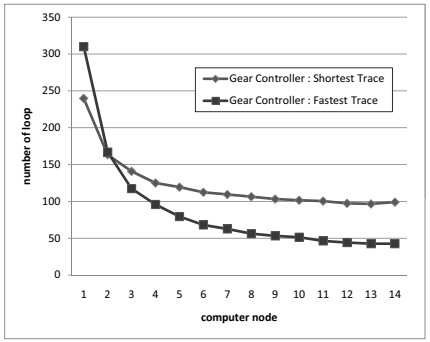

図 6 ループ回数 : Gear Controller
Fig. 6 Number of Iterations : Gear Controller

4.4 反例抽出の優先度について

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 評価項目について

評価項目について述べる．提案手法では，反例を 1 ループ内で複数洗練することでループ回数を削減し，高速化を実現させようとしている．そのため，1 ループ内で反例を複数洗練することがループ回数削減に繋がっているかを確認する必要がある．また，最終的な目標である実行時間の短縮について，計算機台数に対する実行時間も評価する必要がある．この 2 つの項目について評価を行う．

4.6 実 験 結 果

実験結果について以下に示す．

4.6.1 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，

ⓒ 2010 Information Processing Society of Japan

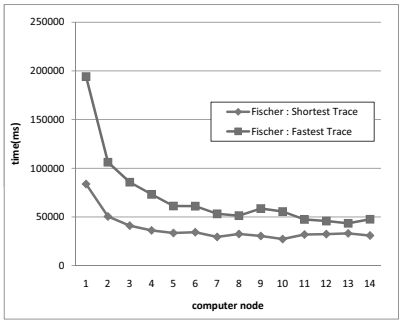

図 7 実行時間 : Fischer
Fig. 7 Excute Time : Fischer

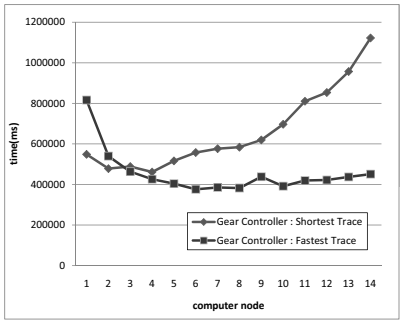

図 8 実行時間 : Gear Controller
Fig. 8 Excute Time : Gear Controller

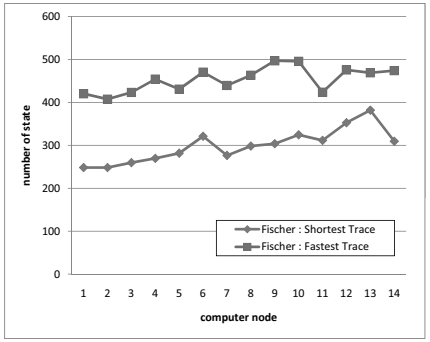

図 9 状態生成数 : Fischer
Fig. 9 Number of States : Fischer

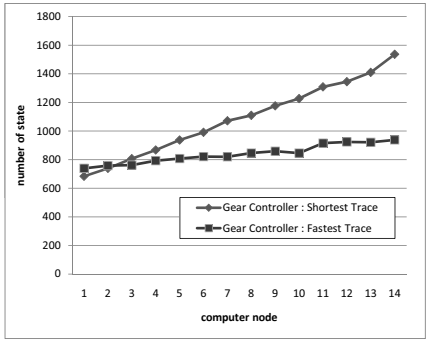

図 10 状態生成数 : Gear Controller
Fig. 10 Number of States : Gear Controller

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．これは，Fischer，Gear Controller 双方においてワーカ計算機の台数に応じてループ数が減少していることを示している．ただ，Gear Controller の Shortest Trace の場合，ループ回数の減少が他よりも鈍化していることがわかる．

4.6.2 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．Fischer 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 Controller 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り，台数を増やすごと処理時間が増加傾向にあることがわかる．

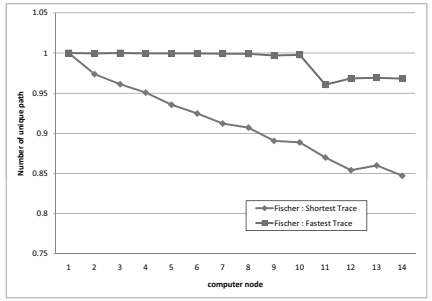

図 11 同一反例の割合 : Fischer
Fig. 11 Ratio of the Same Counterexample : Fischer

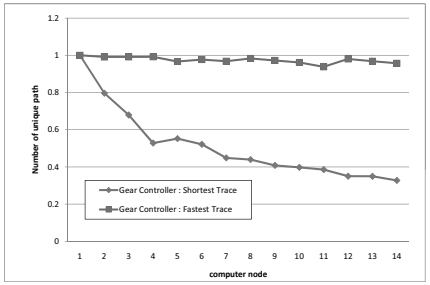

図 12 同一反例の割合 : Gear Controller
Fig. 12 Ratio of the Same Counterexample : Gear Contorller

4.7 実験結果の考察

ループ回数と実行時間の実験結果に対する考察を行う．まず，ループ回数に対する台数効果は実験結果を見る限り出ていると考えられる．このことは，提案手法が効果的であるということを表している．しかし，実行時間はループ回数が減少しているにも関わらず途中で横ばい，あるいは増加傾向にある．この現象について，2 つの可能性が考えられる．

1 つは，1 ループあたりの実行時間が増加している可能性である．複数の反例に対して同時に洗練を行う際，本来はあまり重要度の高くない，洗練の必要性のない個所まで洗練を行っている可能性がある．洗練手法は状態を複製して行われるため，状態数は増加する．状態数が増加するとモデル検査にかかる時間が増大し，結果 1 ループあたりの実行時間が増加してしまう．

もう 1 つは，出力される反例に同一のものが含まれている可能性である．各ワーカーマシンで出力される反例が同一のものであった場合，本来の台数効果を求めることができなくなり，結果として実行時間が横ばいもしくは増加してしまう可能性が考えられる．

以上の考察を裏付けるために，さらに 2 つの項目についてデータの詳細分析を行う．まず，状態生成数について調べる．これは，前者の可能性を裏付けるための指標である．台数に応じて状態が増加しているのであれば，本来では優先度の低い反例が抽出され，洗練が行

ⓒ 2010 Information Processing Society of Japan

われて状態数が増加していることになる．そして，同一反例の割合について調べる．これは，後者の可能性について調べるためで同一反例が多ければ多いほど台数効果が出ていないということになる．

4.7.1 状態生成数

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 では強い相関の関係が見られた．

4.7.2 同一反例の割合

次に，同一反例の割合について調べる．図 11，12 は，出力された全反例に対する，ユニークな反例の割合である．つまり，値が 1 に近ければ近いほど，出力された反例の同一反例の割合は低いと言える．しかしながら，Fischer，Gear Controller 共に，Fastest Trace では殆ど同一反例は出力されていないことが見て取れるが，Shortest Trace においては同一反例が相当数出力されていることがわかる．特に Gear Controller では 14 台並列時の同一反例の割合が 7 割近くに上っていることがわかる．

5. 手法の問題点の考察

4 で得られた実験結果より，提案手法の問題点と手法の改良について考察する．

5.1 1 ループの実行時間の増加への対処

1 ループ内での実行時間の増大の原因は，出力される反例を選別せずに洗練を行ったため，与えられた性質を検査する上で本来必要のない状態の生成が行われたためだと考えられる．この問題を解決するためには，反例の優先度を設定する必要がある．優先度の規則としては，まず反例の長さから優先度を設定する規則が考えられる．図 7 の結果を見てもわかるように，短い反例を出力する Shortest Trace 戦略 の方が台数が少ない場合においては実行時間が少ない．また，図 9 では状態生成数も Shortest Trace 戦略の方が低減されていることがわかる．また，反例の洗練箇所によって優先度を設定する規則も考えられる．例えば，洗練箇所が初期状態から近いものに対する優先度をあげる規則や，逆に遠いものの優先度を上げる規則などが考えられる．このような規則に対して，実験による評価を行う必要があると考えられる．

5.2 同一反例への対処

同一反例が出力される可能性については，$k-$ 最短路探索を用いることで解決できると考えている．$k-$ 最短路探索のアルゴリズムを用いた到達度解析を行うモデル検査ツールを自作することで，問題の解決を行えると考えている．$k-$ 最短路探索のアルゴリズムは Eppstein の手法[16] や Jiménez らの手法[17] 等があるが，実装の方針を考える上で，どのアルゴリズムが有用であるかを以下で検討する．

5.3 閉路の扱いについての考察

次に，閉路の扱いについて述べる．一般的に時間オートマトンは閉路を持つことができる．閉路の回数が異なるだけの反例を複数取得しても洗練箇所が同一である可能性がある．洗練箇所が同一な反例を複数抽出することは探索の冗長化になる恐れがある．しかし，閉路の回数が異なるだけの反例だとしても，洗練箇所が異なる場合もあるため，閉路を含む反例の扱いについては工夫が必要となる．解決策として，閉路の部分を抽象的に表現することで反例を包括的に扱う手法を考えている．閉路の抽象化によりシミュレーション段階での反例の見落としなどを防ぐことができ，また，閉路による余分な反例探索を削減することが可能となるためである．

5.4 $k-$ 最短路探索の実装についての考察

実装についての方針について考察を行う．今回対象とする時間オートマトンはクロック変数の他に整数変数を有する．モデル検査は CEGAR ループの抽象化によりクロック変数が除去されたモデルに対して行われるため，整数変数の扱いについて考察を行う．整数変数の性質を満たす反例を探索する手法としては，以下のものが考えられる．

- 整数変数と状態を引数とした状態を生成することで，整数変数を静的に扱う手法
- 一度全ての反例を探索してから，反例に対して整数変数の条件を満たすかをシミュレーションする手法
- 整数変数を動的に定義し，反例探索時に評価をしていく手法

整数変数を最も簡単に扱うには，整数変数を静的に扱う手法であるが，これは状態を指数的に増加させる要因となり，結果としてモデル検査に要する時間が増大するため有用な手段とは言えない．また，全ての経路を探索してから整数変数の評価を行う場合，条件に合わない反例を相当数抽出させる結果になるため，効率的であるとは言い難い．実装の方針としては，整数変数を動的に定義して，反例探索時に評価していく手法を採用する．整数変数の評価による前向き枝刈りの効果も期待できる．そのため，$k-$ 最短路探索アルゴリズムは動的計画法をベースとし，後ろ向きに探索を行う Jiménez らの手法より，前向き且つ幅優先的に

ⓒ 2010 Information Processing Society of Japan

探索を行う Eppstein の手法の方が有用であると考えている．

6. 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．更に，実験結果を基にして高速化の阻害要因になっている問題を考察し，手法の改良に言及した．

今後の課題として，5 で述べたような手法について実装を速やかに行い，手法の改良とその評価を行いたい．時間オートマトンは各状態間の遷移に重みを持たないため，全ての重みを統一させることで短い反例を優先的に選択することが可能になる．また，同じ閉路を複数含む反例は洗練箇所が重複している可能性が高いため，優先度を低くするなどの機能を追加することで，手法のさらなる高速化を実現していきたい．

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．

参考文献

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) 中島 一, 亀山 幸義: “抽象化と精密化による実時間モデル検査の改善,” 情報処理学会論文誌プログラミング, vol.45, No.SIG12 (PRO23), pp.11-24, 2004.
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) D. Eppstein: “Finding the k shortest paths,” focs, pp.154-165, 35th Annual Symposium on Foundations of Computer Science (FOCS 1994), 1994.
17) V.M. Jiménez and A. Marzal: “Computing the K shortest paths: A new algorithm and an experimental comparison.” WAE 1999, LNCS vol.1668, pp.15-29, 1999.
18) 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, vol. 110, no. 169, SS2010-22, pp. 35-40, 2010.

ⓒ 2010 Information Processing Society of Japan