Vol.2011-SE-171 No.7
2011/3/14

複数反例抽出を用いたCEGARによる時間オートマトンの抽象洗練手法

田中俊彰[†1]　長岡武志[†1]
岡野浩三[†1]　楠本真二[†1]

本稿では，時間オートマトンに対するCEGARに対して複数反例抽出を用いた抽象洗練によるを高速化手法を提案する．初期抽象を行ったモデルに対して複数の反例を抽出し，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，提案した手法を *k*-最短路探索手法により実装し，複数の時間オートマトンに対して評価実験を行った．評価実験を通じて提案手法の効果を確認した．

Simultaneous Application of Multiple Counterexample Guided Abstraction Refinement for Timed Automata

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

Our research group has already proposed a CEGAR loop algorithm for timed automata. This report proposes an efficient technique of reachability analysis for timed automata using simultaneous multiple counterexample extraction. We have prototyped a model checker using a *k*-shortest paths search algorithm and performed experiments.We found that some of results show the effeciency of the proposed technique.

1. まえがき

本稿では，時間オートマトンの時間抽象化による抽象洗練手法に対する，複数反例抽出を用いた高速化手法を提案する．文献[13]では，Clarkeらの Counter-example Guided Abstraction Refinement[1]の枠組みを利用し，時間オートマトンに対する抽象洗練手法を提案している．洗練の粒度が小さいため，モデル検査時に状態爆発を回避する効果的な手段であるが，大規模なモデルを適用した場合に実行時間が大幅に増加することが懸念される．

そこで，本稿では1ループ内でモデル検査時に潜在する全ての反例を抽出し，その反例全てに対してシミュレーションを行い，その結果を統合して洗練を行う複数反例抽出手法を適用することで手法[13]の高速化を目指す．

また，文献[16],[17]では並列計算機上で複数反例抽出手法を実装していたが，提案手法では *k*-最短路探索手法による実装を行うことで，同期による実行時間のオーバーヘッドや同一反例出力を回避している．具体的には1つのモデル検査で複数の反例を得て，シミュレーションはこれらの反例を逐次実行し，洗練はこれら複数の反例に対する洗練を同時に行う．手法を実装したプロトタイプに対し，複数の性質の異なる時間オートマトンに対して評価実験を行い，実験結果から提案手法の有用性を示す．

以下，2ではまず時間オートマトンと本稿で利用するCEGAR[1]について簡潔に述べる．3では本稿で提案する複数反例抽出を用いたCEGARのアルゴリズムを概要と，*k*-最短路探索手法による実装について述べる．4で提案手法の評価実験を行い，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, $\sim \in \{\leq, \geq, <, >\}$. 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

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

$l_0 \in L$: 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.

定義 2.4 (クロックの評価関数)**.** $\nu : C \to \mathbb{R}_{\geq 0}$ 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.

ν の全てからなる集合を N とする.

定義 2.5 (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 2.6 (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 1 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．そのため，実際のモデルでも性質を満たすためループを終了し，性質を満たすという出力を行う.

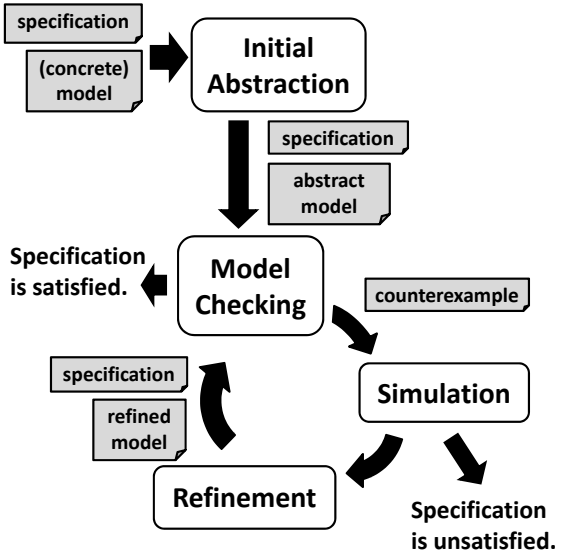

図 1 一般的な CEGAR アルゴリズム
Fig. 1 General CEGAR Algorithm

もしモデル検査においてモデルは性質を満たさないという結果を返された場合，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，モデルは性質を満たさないということとなり，ループを終了し反例の出力を行う.

シミュレーションの結果，検出された反例が偽反例であることが解った場合，その偽反例が検出されないように抽象モデルを改善する段階に入る (これを洗練と呼ぶ)．洗練を行った抽象モデルは再度モデル検査を行い，性質を満たすかどうかを検査する.

これらの段階を繰り返し，状態数の増加を抑え，メモリ使用量を削減しながら正しい出力を得る.

3. 提 案 手 法

本節では，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を示す．提案する抽象化洗練手法では，文献[13] で提案した時間オートマトンの抽象化洗練手法を利用している．さらに，実行時間を削減するために，1 ループ内での洗練を複数反例抽出を同時に適用し 1 回で行っている．以下に処理の流れを示す (図 2).

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

(2) 抽象モデルに対してモデル検査を行う．このとき，性質を満たすのであれば True を返す．性質を満たさないのであれば抽象モデルに存在し得る反例を高々指定パラメータ k 個分，出力する．

(3) 出力された反例を基にシミュレーションを逐次的に行う．シミュレーションの結果，実際のモデルでも存在し得る反例が 1 つでも見つかれば False と反例を返す．

(4) シミュレーションの結果反例が全て偽反例であった場合，シミュレーション結果を基に抽象モデルの抽出された全反例に対する洗練を 1 度に行う．

(5) 洗練を行った抽象モデルを再度モデル検査を行う．以上の動作を繰り返す．

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[13] 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．時間オートマトンのクロック変数に関する制約を全て除去することで，モデル検査における状態数の指数的に増加を回避することが可能となる．

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．

- $\hat{S} = L$,
- $\hat{s_0} = h(s_0)$

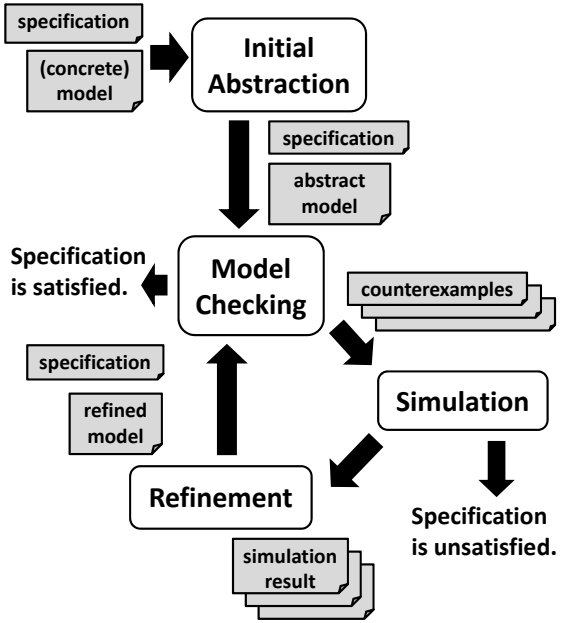

図 2 k-最短路探索手法を用いた CEGAR アルゴリズム
Fig. 2 Our CEGAR Algorithm

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

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．

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

3.2 モデル検査

本手法では，到達可能性解析に限定しているため，モデル検査はグラフの探索問題に帰結することができる．そのため，ある抽象モデル $\hat{M}$ に存在し得る全ての反例を抽出する場合の時間的なコストは，反例をたかだか 1 つのみ抽出する場合とほぼ等価である．

通常のモデル検査器は反例出力は 1 つだけ行うものが多いため，複数反例抽出手法を実現させるためモデル検査器の実装を行った．k-最短路探索アルゴリズムはあるグラフに対してある 2 点間の距離について，最短路から順に最大で k 個まで出力を行うアルゴリズムで

あり，代表的なものとして Eppstein の手法[14] や Jimenez の手法[15] 等があげられる．しかしながら，これらのアルゴリズムがある 2 点間の距離を求めるのに対して，到達可能性解析ではある 1 点から到達できない点を探索するため，*k*-最短路探索アルゴリズムをそのまま適用することはできない．また，対象とする時間オートマトンのモデルでは，遷移条件に整数変数の条件も付与されるため，よりアルゴリズムが複雑になる．

本稿では，幅優先探索をベースに *k*-最短路探索の実装を行った．状態 $\hat{s}(l,v)$ はロケーション l と Integer 変数の値 v より決定し，抽象モデル $\hat{M}$ 上の遷移 $\hat{s}(l,v) \xrightarrow{a} \hat{s}(l',v')$ の条件 a によって次の状態 $\hat{s}(l',v')$ が動的計画法によって探索される．初期状態から順に探索可能な次状態を探索していき，条件が成立しない状態に到達すればそこまでの経路を反例として記録する．以下にアルゴリズムを示す (図 3).

3.3 シミュレーション

シミュレーションでは，抽象モデル $\hat{M}$ に対してモデル検査で求められた反例に対して元となる時間オートマトン上でも実行可能かどうかについて調べる．本手法では，文献[13] で提案されている手法に従う．本手法では抽出された複数の反例に対してそれぞれ逐次的にシミュレーション処理を行う．シミュレーションを各反例に対して逐次行ったとしてもモデル検査処理と比べて時間的なコストが少ないため，ボトルネックにはなりにくいと判断した．

3.4 抽象モデルの洗練

抽象モデル上で発生した偽反例を，元の時間オートマトン上で発生しないように，抽象モデル上で洗練を行う．抽象モデルの洗練は，文献[13] で述べられている状態の複製による手法を用いる．本手法では抽出された複数の反例に対してそれぞれ逐次的に洗練を行う．洗練処理はシミュレーション処理と同等にモデル検査処理に対して時間的なコストが少ないため，逐次的に行ってもボトルネックになりにくい．

3.4.1 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[13] において定義されている (アルゴリズム 2：図 4).

ここで示されているアルゴリズム 2 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 2 ’(図 5) を与える．

Model Checking
Inputs $\hat{M}, error\_list$

$\{errir\_list = \langle (l_0,D_0),(l_1,D_1),\cdots,(l_k,D_k)\rangle\}$
$Passed := \emptyset$
$Waiting := (l_0,D_0)$
$Error := \emptyset$
while $Waiting \neq \emptyset$ **do**
  $select(l,D) from Waiting$
  for $i := error\_list.length$ downto 1 **do**
    if $(l,D) = (l_i,D_i)$ *in* $error_list$ **then**
      $add(l,D)$ *to* $Error$
      $break$
    end if
  end for
  if $for\ all\ tr_i\ in\ (l,D) = true$ **then**
    $calculate(\hat{M},(l,D))$
    $add(l,D)$ *to* $Passed$
    for $j := (l,D).succ\_list.length$ downto 1 **do**
      $add(l',D')$ *in* $Waiting$
    end for
  end if
end while
return $Error$

図 3 アルゴリズム 1 ： *k*-最短路探索手法によるモデル検査アルゴリズム
Fig. 3 algorithm 1 : Model Checking Algorithm using *k*-Shortest Paths Search Algorithm

定義 3.4 (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．

ある反例の集合 $\hat{P}$ に対して，順にアルゴリズム 2 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 2 を適用せず，$\hat{P}$ の次の反例に対して処理を繰り返す．

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 4 アルゴリズム 2 : 洗練アルゴリズム
Fig. 4 algorithm 2 : Refinement Algorithm

3.4.2 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. なお，定理 (3.1-3.3) の証明は文献[16] で行っているため，本稿では定理のみ記述する．

まず，反例の重複について, 定義 3.5 で与える．

定義 3.5 (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．

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 5 アルゴリズム 2' : 洗練アルゴリズム (複数パス)
Fig. 5 Algorithm 2' : Refinement Algorithm of CEs

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 *2'* を適用しても大域的に正しい洗練である．

なお，定理 3.1 は次の定理 3.2，3.3 より導かれる．

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 *2* を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 *2'* を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．

4. 評 価 実 験

本章では，提案手法についての評価実験を行う．

4.1 実 験 環 境

実験環境について以下に示す.

CPU : Intel(R) Xeon(R) CPU E5507 2.27GHz

メモリ：16.00GB　OS : Ubuntu 10.10

4.2 CEGAR の実装

実装は，文献[13] の手法に基づく．シミュレーション時の時間的な性質の検証では，UPPAAL[8] の DBM ライブラリを利用している．また，文献[16],[17] で利用していたモデル検査モジュール[8] は反例を高々1 つだけ出力するため，本手法には適さない．そのため，3.2 で示したアルゴリズムを元に実装を行った．探索手法は幅優先で行い，動的計画法を用いて探索の効率化を行う．入力は通常のオートマトンと到達不可状態を示す性質，出力する最大反例数を表すパラメータ k であり、出力は最大でも高々k 個の反例となる．このモジュールはC++で実装した (4000LOC).

4.3 対象とした時間オートマトン

実験では Fischer の相互排除プロトコル[11] と Gear Controller[12] を利用する．

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，提案手法に適していると判断した．評価実験において，3 並列から 7 並列までのモデルを利用する．特に，6 並列以上の Fischer の相互排除プロトコルでは，初期の状態数が 4096 を超え，文献[13] では実用的な時間でのモデル検査が行うことができない．

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．また，評価実験では満たすべき性質を複数用意し，各性質について実験を行う．

4.4 *k*-最短路探索手法による実装に対する評価実験

まず，*k*-最短路探索手法による実装に対して評価を行う．提案手法により実行時間が減少するかについて調べる．

4.4.1 対象とする時間オートマトン

対象とするモデルは，Fischer の相互排除プロトコルの 3 並列から 7 並列と，Gear Controller である．

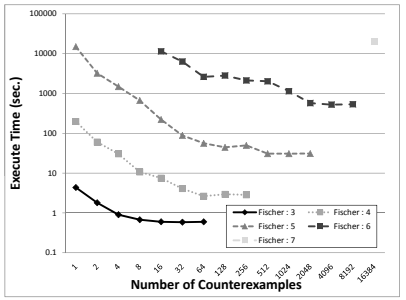

図 6 実行時間 ： Fischer
Fig. 6 Execute Time : Fischer

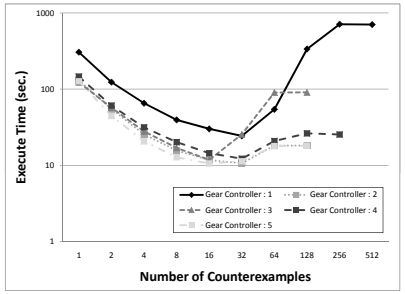

図 7 実行時間 : Gear Controller
Fig. 7 Execute Time : Gear Controller

表 1 最大反例数：Fischer の相互排除プロトコル
Table 1 The Maximum Number of Counterexamples : Fischer

Fischer : 3	Fischer : 4	Fischer : 5	Fischer : 6	Fischer : 7
18	108	540	2430	10206

表 2 最大反例数：Gear Controller
Table 2 The Maximum Number of Counterexamples : Gear Controller

Gear Controller : 1	Gear Controller : 2	Gear Controller : 3	Gear Controller : 4	Gear Controller : 5
162	58	58	74	40

4.4.2 評価項目について

評価項目については，抽象洗練手法の実行時間について行う．また，本実験では実行時間の上限を 5 時間と設定し，5 時間経過した時点でタイムアウトとした．

4.4.3 実 験 結 果

実験結果は，横軸を抽出した反例の数とし，縦軸はそれぞれの計測したい項目についてである．単位は秒である．実験で用いた時間オートマトンの 1 ループあたりに抽出された反例の最大数を表 1,2 に示す．

図 6,7 は，抽出した反例数に対する実行時間の増減について表している．Fischer の相互排除プロトコルでは反例数を増やすことで減少しているのに対し，GearController では反例

数 32 を境に上昇に転じている．

4.4.4 考　　察

実験結果から，考察を行う．まず，Fischer の相互排除プロトコルについての評価実験では，ほぼ想定通りの結果を得ることができた．提案手法により反例を 1 ループ内で複数抽出することで時間オートマトンに対する CEGAR の実行時間を減少させることに成功している．しかしながら，Gear Controller においては，反例数が 32 までは Fischer の相互排除プロトコルと同等の結果が出ているのだが，それ以上の反例数で実行しようとすると実行時間が増加する結果が得られた．このことについてある仮説が考えられる．

k-最短路探索手法は得られた反例について，最短のものから順に出力する．一方，極端に長い反例を洗練で利用した場合，本来必要ない洗練が行われる可能性が高く，実行時間が長くなる傾向にある．この 2 点から，Gear Cotroller においては抽出する反例数が 32 以上である場合，極端に長い反例が出力され，本来必要ではない冗長な洗練が行われる可能性がある．

この仮説を評価するために，以下の実験を行う．

4.5 反例の長さを制限した複数反例抽出手法

反例の長さによりループ回数や実行時間が変化するかについて，実験による評価を行う．ここで，反例の長さについては以下のように定義する．

定義 4.1 (反例の長さ)**.** 抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \Rightarrow)$ 上の反例 $\hat{\sigma} = \langle \hat{s}_0 \to \hat{s}_1 \to \hat{s}_2 \to \cdots \to \hat{s}_{n-1} \to \hat{s}_n \rangle$ に対し，反例の長さ *length* は次のようにあらわされる．

$$length(\hat{\sigma}) = n$$

k-最短路探索手法による実装では，反例は最短のものから順に出力される．よって，1 ループ内で洗練を行う反例を増やすことは，反例の長さが長い反例に対しても洗練を行うことを意味している．Gear Controller の場合，実験 4.4 では反例数が 32 程度までは反例数と共に実行時間も減少していた．つまり，あまりに長い反例を 1 ループ内で洗練を行うことは，洗練時に余分な状態を生成し，洗練の効率化に寄与しないばかりか状態数を増やすことでモデル検査時間を増大させる可能性がある．

そこで，反例の長さに閾値を設け，閾値を超える長さの反例に関しては抽出されたとしても洗練を行わないことで，実行時間の短縮に繋がるかについて評価を行う．ここで，閾値については以下のように定義する．

定義 4.2 (反例の長さの閾値)**.** 抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \Rightarrow)$ 上の最短の反例を $\hat{\sigma}_{min}$ とするとき，閾値 τ は次のようにあらわされる．

$$\tau = f \times length(\hat{\sigma}_{min})$$

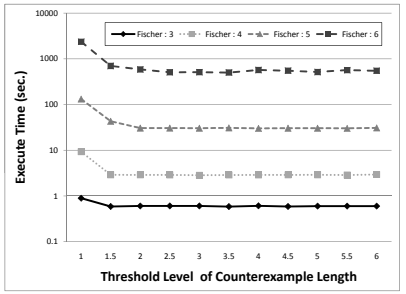

図 8　実行時間 (閾値あり) ： Fischer
Fig. 8　Execute Time (with a threshold) : Fischer

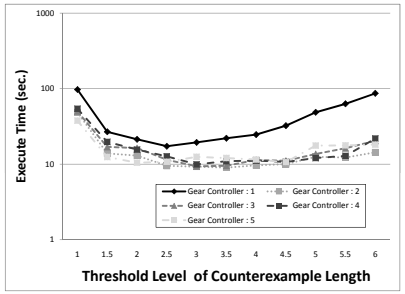

図 9　実行時間 (閾値あり) : Gear Controller
Fig. 9　Execute Time (with a threshold) : Gear Controller

ここで，*f* は反例長の基準となる係数パラメータである．

次の評価実験では，係数 *f* の値を変化させることで，最も実行時間の短縮になる閾値の評価を行う．

4.5.1 対象とする時間オートマトン

対象とするモデルは，Fischer の相互排除プロトコルの 3 並列から 6 並列と，Gear Controller である．

4.5.2 実 験 結 果

実験結果は，横軸を抽出した反例に対し，閾値とする反例の長さを求める基準として，最短反例からの長さの倍率の値である．最小値を最短反例の 1 倍とし，最大で 6 倍までの長さまで実験を行った．縦軸はそれぞれの計測したい項目についてである．

図 8,9 は，反例の長さの倍率に対する実行時間の増減について表している．Fischer の相互排除プロトコルについては，反例の長さ 2 倍以上にしても値に変化がない．これは，反例の長さを 2 倍以上にした場合，全ての反例が閾値以内に収まるためで，実験結果において差異が出てこない．事前の実験 4.4 において Fischer の相互排除プロトコルにおいて反例数と実行時間の減少は相関があったため，本実験においても有効性は揺るがないことが確認できる．一方，Gear Controller での実験においては，主に反例の長さの倍率を 3 倍程度に限定した場合，最も効果があることが分かる．それ以上の倍率で洗練を行った場合，冗長な洗練が行われている可能性が高いことを示している．

4.5.3 考　　察

評価実験を通じて，複数反例抽出による時間オートマトンに対する CEGAR の高速化が実現できた．特に，Fischer の相互排除プロトコルについて，既存手法では実用的な時間では検査できていなかった 6 並列と 7 並列のモデルへの適用が可能となった．一方，GearController については，単純に適用する反例の数を増やすだけでは高速化の効果が出ず，反例の長さに閾値を設定するなどの工夫を行う必要があった．しかしながら，Gear Controller に対しても高速化の効果が実現できていることが確認でき，シンメトリな構造を持たないモデルに対しても有用であることを示すことができた．

5. あ と が き

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 1 ループで複数の異なった反例を抽出，洗練を行う複数反例抽出手法による高速化を提案し，*k*-最短路探索手法による実装による評価実験を行った．実験の結果，複数の時間オートマトンに対して有効であることを示すことができ，また抽出した反例に対して選択の工夫を加えることで汎用性の高い手法にすることが可能となった．

今後の課題として，さらに多くの時間オートマトンに対して適用するとともに，複雑度が高く，状態数の多い時間オートマトンに対して実験を行うことで，手法の有用性を実証していきたい．また，反例の適用順序を変化させることで，実行時間のさらなる減少を目指していきたい．

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) 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
9) 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.
10) 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.
11) J. Bengtsson, K G. Larsen, F. Larsson, P. Pettersson, and W. Yi: “UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems,” Hybrid Systems 1995, pp232-243.
12) 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.
13) 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.
14) D. Eppstein, “Finding the k shortest paths,” focs, pp.154-165, 35th Annual Symposium on Foundations of Computer Science (FOCS 1994), 1994.
15) V.M. Jimenez and A. Marzal. “Computing the K shortest paths: A new algorithm and an experimental comparison.” WAE 1999, LNCS 1668: pp. 15-29, 1999.
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” 電子情報通信学会信学技報, vol. 110, no. 169, SS2010-22, pp. 35-40, 2010.
17) 田中俊彰，長岡武志，岡野浩三，楠本真二，“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” 情報処理学会研究報告， Vol.2010-SE-170, No.15, 2010.