修士学位論文

題目

複数反例抽出を用いた時間オートマトンの到達可能性解析手法

指導教員
楠本　真二　教授

報告者
田中　俊彰

平成 23 年 2 月 7 日

大阪大学 大学院情報科学研究科
コンピュータサイエンス専攻

平成 22 年度 修士学位論文

複数反例抽出を用いた時間オートマトンの到達可能性解析手法

田中　俊彰

内容梗概

モデル検査は，システムの設計段階での不具合の発見に有用である．そのため，システムの高信頼設計を目的にシステム設計の初期段階で適用されるようになってきている．さらにそれが実時間のシステムであれば，時間的な制約を持ち，考慮すべき状態数が時間制約に応じて爆発的に増加するため，検査に対するスケーラビリティに大きな制約が加わる．一方，Clarke らが古典的な Kripke 構造を持つオートマトンに対して反例に基づく抽象化洗練 CEGAR を提案し，状態爆発を回避しながらモデル検査を自動で行う枠組みを提案した．この枠組みは時間オートマトンをはじめとして，多くのオートマトンモデルに対しての有用性を示した．

既存研究として，研究グループにおいて時間オートマトンに対する抽象化洗練手法が提案されている．しかしながら，システムの大規模化に伴い，CEGAR のループ回数が増大し，処理に多大な時間が必要となる．そのため，本研究では，既存研究の手法を下敷きに，複数反例抽出という新たなアプローチで CEGAR のループ回数を削減し，処理の高速化を目指す．

本論文では，提案手法の概要を示したのち，具体的な実装方法について並列計算機と *k*- 最短路探索手法を用いた場合について述べる．また，提案手法に対して評価実験を行い，手法の比較考察を行う．評価実験では，既存手法に対し 500 倍程の実行時間の減少するという結果を得ることができ，また既存手法では実用的な時間では検査することのできなかったモデルに対しても本手法を用いることで検査が可能となり，複数反例抽出による高速化手法の有用性を確認することができた．さらに，複数反例抽出において反例の適用順序と CEGAR の高速化についての議論を行い，実験から得られたデータを基に複数の項目を選出し，評価を行う．最後に，今後の課題とまとめについて述べる．

主な用語

モデル検査, 到達可能性解析, 時間オートマトン, CEGAR ループ, 並列処理,　*k*-最短路探索

目 次

1 まえがき **1**

2 関連研究 **3**

2.1 並列計算機の利用 3

2.2 CEGAR ループの工夫 3

3 準備 **4**

3.1 時間オートマトン [9] 4

3.2 CEGAR アルゴリズム 5

3.2.1 一般的な CEGAR の実行例 7

4 提案手法 **9**

4.1 基本アルゴリズム 9

4.1.1 初期抽象化 10

4.1.2 モデル検査 10

4.1.3 シミュレーション 11

4.1.4 抽象モデルの洗練 11

4.2 並列計算機環境下での実装 14

4.3 *k*-最短路探索手法による実装 15

5 評価実験 **18**

5.1 CEGAR の実装 18

5.2 対象とした時間オートマトン 18

5.3 並列計算機環境での実験 18

5.3.1 実験環境 18

5.3.2 対象とする時間オートマトン 19

5.3.3 モデル検査ツール 19

5.3.4 評価項目について 19

5.3.5 実験結果 19

5.3.6 実験結果の考察 20

5.3.7 考察に対する評価実験の結果 21

5.3.8 並列計算機上の実装に対する考察 23

5.4 *k*-最短路探索手法での実験 23

5.4.1 実験環境 23

5.4.2 対象とする時間オートマトン 23

5.4.3 モデル検査ツール 23
5.4.4 評価項目について 23
5.4.5 実験結果 24
5.4.6 考察 25
5.5 評価実験のまとめ 26

6 反例の解析と適用順序の工夫 **27**
6.1 反例の長さを制限した複数反例抽出手法 27
6.1.1 対象とする時間オートマトン 27
6.1.2 実験結果 28
6.2 洗練情報の特性を用いた複数反例の洗練手法 29
6.2.1 優先度の項目について 30
6.2.2 実験環境について 31
6.2.3 実験結果 31
6.2.4 考察 32

7 あとがき **33**

謝辞 **34**

参考文献 **35**

A 付録 **38**

図目次

1 一般的な CEGAR アルゴリズム 6
2 CEGAR の例：元モデル 7
3 CEGAR の例：初期抽象化 8
4 CEGAR の例：抽象モデル 8
5 CEGAR の例：洗練を行った抽象モデル 9
6 アルゴリズム 1 ： 洗練アルゴリズム 12
7 アルゴリズム 2： 洗練アルゴリズム (複数パス) 13
8 並列計算機環境を利用した CEGAR アルゴリズム 14
9 *k*-最短路探索手法を用いた CEGAR アルゴリズム 16
10 アルゴリズム 3 ： *k*-最短路探索手法によるモデル検査アルゴリズム 17
11 ループ回数 (並列環境) ： Fischer 20
12 ループ回数 (並列環境) : Gear Controller 20
13 実行時間 (並列環境) ： Fischer 20
14 実行時間（並列環境）: Gear Controller 20
15 生成状態数 (並列環境) ： Fischer 21
16 生成状態数 (並列環境) : Gear Controller 21
17 全反例数に対する反例の種類 ： Fischer 22
18 全反例数に対する反例の種類 : Gear Controller 22
19 ループ回数 (k-最短路探索手法) ： Fischer 24
20 ループ回数 (k-最短路探索手法) : Gear Controller 24
21 実行時間 (k-最短路探索手法) ： Fischer 24
22 実行時間 (k-最短路探索手法) : Gear Controller 24
23 生成状態数 (k-最短路探索手法) ： Fischer 25
24 生成状態数 (k-最短路探索手法) : Gear Controller 25
25 ループ回数 (k-最短路探索手法：閾値あり) ： Fischer 28
26 ループ回数 (k-最短路探索手法：閾値あり) : Gear Controller 28
27 実行時間 (k-最短路探索手法：閾値あり) ： Fischer 28
28 実行時間 (k-最短路探索手法：閾値あり) : Gear Controller 28
29 生成状態数 (k-最短路探索手法：閾値あり) ： Fischer 29
30 生成状態数 (k-最短路探索手法：閾値あり) : Gear Controller 29

表 目 次

1 最大反例数：Fischer の相互排除プロトコル 25
2 最大反例数：Gear Controller 26
3 Fischer : 8 に対する実験結果 26
4 Fischer-3 並列：昇順 38
5 Fischer-4 並列：昇順 38
6 Fischer-5 並列：昇順 39
7 Fischer-6 並列：昇順 39
8 Gear Controller-1：昇順 39
9 Gear Controller-2：昇順 40
10 Gear Controller-3：昇順 40
11 Gear Controller-4：昇順 40
12 Gear Contorller-5：昇順 41
13 Fischer-3 並列：降順 41
14 Fischer-4 並列：降順 41
15 Fischer-5 並列：降順 42
16 Fischer-6 並列：降順 42
17 Gear Controller-1：降順 42
18 Gear Controller-2：降順 43
19 Gear Controller-3：降順 43
20 Gear Controller-4：降順 43
21 Gear Contorller-5：降順 44

1 まえがき

近年，情報システムの高度化，複雑化と共に，ソフトウェアの信頼性を保証する枠組みが望まれている．モデル検査 [4] は，システムを有限の状態遷移系 (モデル) として形式的に記述し，また検査を行う性質を論理式で入力することで，モデルの全状態を網羅的に探索する技術である．この技術により，設計されたシステムが性質を満たしているかどうかについて，数学的に保証を与えることができ，注目される技術である．しかしながら，システムの振舞いをモデルで表し，全状態を網羅的に探索するモデル検査は，大規模なシステムに対して探索空間が肥大化してしまう状態爆発を起こしてしまう問題を常に内包しており，実際のシステムに適用する上での懸案事項の 1 つとなっている．状態爆発問題により，探索時に状態数の増加によるメモリ容量の肥大化が起こり，検査ができないという事態に陥る [5]. この状態爆発問題を回避する方策の 1 つとして，モデルの状態数を検査する性質ごとに適切に削減するモデル抽象化手法が注目されている．[10] とりわけ，モデル検査時に出力される反例を用いて抽象化を行う CEGAR(CounterExample Guided Abstraction Refinement) ループが大きな注目を集めている [1, 2, 3]. 一方，実際に検査対象となるモデルとしては，並列処理や時間的なシステムを検証する場合，時間オートマトン [9] と呼ばれるモデルが利用される．時間オートマトンは，有限の状態遷移系に，時間的な性質を表すクロック変数を付加したものである．クロック変数は実数値をとり，クロック変数を用いた制約が状態に付与されるため，状態数が通常のモデルよりも指数的に増加することが知られている [6, 7]. そのため，モデル検査を行う際には状態数を適切に削減する抽象化手法が必要となる [13, 14, 15]. 従来研究 [18] では時間オートマトンのモデル検査に CEGAR ループを適用することで，状態数を適切に削減し，状態爆発問題を回避している．しかしながら，従来研究では抽象化の粒度が細かいため，大規模なモデルに対しては膨大な回数の洗練を必要とする場合がある．従来手法では 1 ループあたり 1 回の洗練のみしか適用できないため，そのような場合ループ回数が増加し，結果として実行時間の爆発的な増加を引き起こしている．

そこで本研究では，従来手法 [18] におけるモデル検査の工程で 1 度に複数の反例を抽出し，それぞれに対して洗練を適用することで，全体のループ回数を削減し，実行時間の増加の問題を改善する．一般に 1 回のモデル検査では 1 つの反例しか抽出しないため，本研究ではこの複数反例抽出の考え方を，複数台の計算機を用いた並列処理によって実現する方法と，*k*-最短路探索手法 [19, 20, 21, 22, 23] を用いて 1 台の計算機上で実現する方法で実装し，複数のモデルで評価実験を行った．その結果，提案手法の有用性と問題点を洗い出すと共に，既存手法との比較を行った．さらに，複数反例抽出時に反例の優先度を設定し，洗練の優先順序を操作することで処理時間の減少と状態数の更なる減少が見込めるかについて評価実験を行い，議論を述べている．この評価実験において，一定の知見を得ることができた．

以下，3 では，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，本研究で利用する CEGAR ループについて，さらにモデル検査によって調べられる性質について述べる．次に 4 では，既存手法 [18] における CEGAR の各操作の説明と，本論文における提案手法である複数反例抽

出手法について述べる．また，複数台の計算機を用いた並列実行環境での実装と，k-最短路探索手法を用いた実装の詳細について述べる．5 では実装した手法について複数の時間オートマトンを用いた評価実験を行い，既存手法との比較と手法の有用性を評価する．6 では，出力される反例の優先度の設定と優先度の基準となる評価変数の調査，時間オートマトンを用いた評価実験を行い，7 でまとめを行う．

2　関連研究

本研究では，時間オートマトンに対する CEGAR について，複数反例抽出による高速化手法を提案している．本節では，モデル検査手法の高速化についての関連研究について簡単に紹介する．

2.1　並列計算機の利用

モデル検査において，並列計算機環境による高速化手法は多く提案されている [26][27]．モデル検査での探索において各計算機に探索空間を分割して配布し，各計算機同士で同期を取りながら処理を行う場合が多く，[27] においてはロードバランスの管理なども行うことで高速化を実現している．しかしながら，モデル検査で扱うモデル自体が非決定的な遷移を含む場合や，大規模なモデルに対して適切にモデルを分割することができず，各計算機間での通信によるオーバーヘッドにより処理が遅延するなど，一定以上のモデルに対しては問題がある．

2.2　CEGAR ループの工夫

CEGAR ループの処理を工夫することで，実行時間を減少させようという試みは文献 [28] においてなされている．ここでは，複数の洗練手法を用いた CEGAR ループを並列で動作させる場合や，出力された反例によって洗練を行うかどうかを決定する場合について議論がなされている．しかしながら，文献内ではアイディアの提案のみであり，実験を行い評価を行った等の記述はない．

3 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．

3.1 時間オートマトン [9]

定義 3.1 (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 3.2 (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，$in_1^i n_2$ も同様に $c(C)$ の要素である．

定義 3.3 (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．

定義 3.4 (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 3.5 (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 3.6 (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.

定義 3.7 (クロックリージョン**).** 与えられた 1 つの時間オートマトン $\mathscr{A} = (L, l_0, T, I, C, A)$ に対してクロックリージョン $CR(\mathscr{A})$ を 1 つ定めることができる *[6, 7]*.

クロックリージョンは $|\ C\ |$-次元ユークリッド空間上を点，線分，面などで有限分割したものになる.

$CR(\mathscr{A})$ の要素 (である 1 つのリージョン) を $[u]$ で表記する. $[u] \in CR(\mathscr{A})$ なる $[u]$ について，$[u]$ 中の任意の各点がガード g, インバリアント I を満たすことをそれぞれ $g([u]), I([u])$ と表記する. また, $[u]$ にリセットクロック r を適用することを $r([u])$ と表記する.

$r([u]) = [u][x \mapsto 0]$, where $x \in r$.

定義 3.8 (リージョンオートマトン**).** 時間オートマトン $\mathscr{A} = (L, l_0, T, I, C, A)$ のリージョンオートマトン $\mathscr{A}_r = (L_r, l_{r\,0}, T_r, A)$ は以下のように与えられる.

$L_r \subset L \times 2^{c(C)}$

$l_{r\,0} = (l_0, [0^C])$, ここで $[0^C]$ は $I(l_0)$ を満たす.

$T_r \subset L_r \times A \cup \mathbb{R}_{\geq 0} \times L_r$，$T_r$ の要素は以下のものである.

- $(l, [u]) \overset{d}{\Rightarrow} (l, [v])$ *if* $(l, u) \overset{d}{\Rightarrow} (l, v)$ *for* $d \in \mathbb{R}_{\geq 0}$
- $(l, [u]) \overset{a}{\Rightarrow} (l', [v])$ *if* $(l, u) \overset{a}{\Rightarrow} (l', v)$ *for* $a \in A$

定義 3.9 (DBM(Difference Bound Matrix)[12])**.** 時間オートマトン $\mathscr{A} = (L, l_0, T, I, C, A)$ に対して *DBM* は $|C|$ -次元ユークリッド空間上の凸空間を表現し，クロックリージョン $CR(\mathscr{A})$ の要素の集合として表現される. また，$\mathscr{A}$ のリージョンオートマトン $\mathscr{A}_r = (L_r, l_{r\,0}, T_r, A)$ 上の状態集合を *DBM* D を用いて，$(l, D) = \{(l, [u]) | [u] \in D\}$ のように表現する.

文献 [9] では，DBM に対する操作関数として，任意の遅延を実現する up，DBM と差分不等式との積集合をとる and などが与えられている. また，DBM D を表現するために必要な最小限の差分不等式の集合を $c(D)$ と表す．$c(D)$ は DBM に対するリダクション操作により求めることが可能である. また，ロケーション l のインバリアント $I(l)$ を満たすすべてのリージョンの集合を (l, D_{Inv}) と表す.

3.2 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

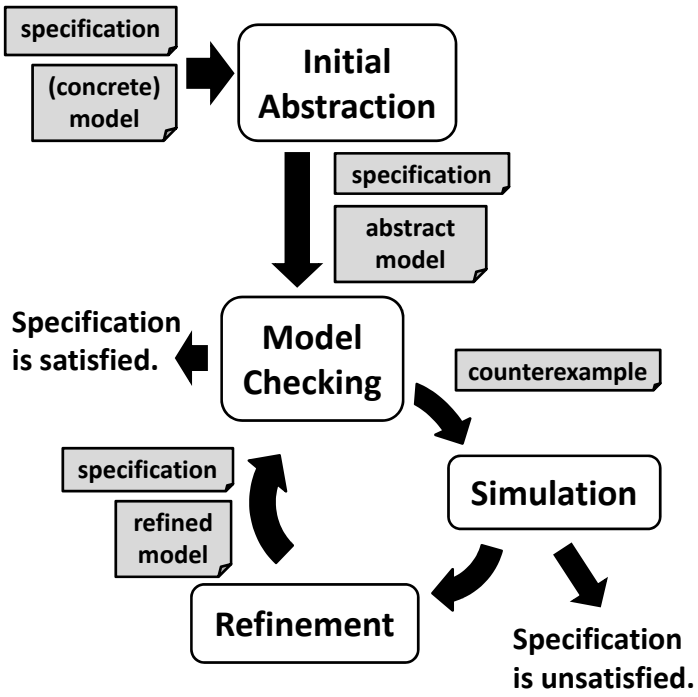

図 1: 一般的な CEGAR アルゴリズム

Abstraction Refinement) と呼ばれるアルゴリズムを提案している (図 1).

CEGAR アルゴリズムの概要について以下に示す．入力は検査対象のモデルと満たすべき性質である．

1. 入力されたモデルを過度に抽象化する．これを，初期抽象化と呼ぶ．初期抽象化によって生成されたモデルを抽象モデルと呼び，入力されたモデルは元モデルと呼称する．

2. 生成された抽象モデルに対してモデル検査を行う．過度に抽象化を行った抽象モデルは元モデルに比べて状態数が少ないため，モデル検査時にかかるメモリ容量を低減されることが可能である．また，初期抽象化により生成される抽象モデルにおいて，抽象モデルが満たすべき性質を満たすのであれば，元モデルでも性質を満たすように抽象化されている．そのため，抽象モデルに対するモデル検査が True であった場合，元モデルでも性質を満たす.

3. もし抽象モデルのモデル検査が False であった場合，モデル検査時に抽出される反例を用い，元モデルでもその反例が存在するかをシミュレーションを行うことで確認する．シミュレーションはモデル検査に比べると探索空間が狭いため，実行に時間がかからない.

4. シミュレーションの結果，抽象モデル上の反例が元モデルでも存在する反例であった場合，モデルは性質を満たさないことが分かるため，False を返し終了する．

5. シミュレーションの結果，元モデル上では存在しない反例であった場合，すなわち反例が偽反例であった場合，抽象モデルの抽象化が正しくないということであるため，抽象モデルの洗練を行う．このとき，抽象モデルでは元の反例が存在しないように洗練を行う．

6. 洗練を行った抽象モデルを再度モデル検査を行い，性質を満たすかどうか検査する．これらの段階を繰り返すことで最終的には状態数の増加を抑えながら，正しい出力を得る，

3.2.1 一般的な CEGAR の実行例

一般的な CEGAR の例について示す．まず，入力として図 2 のようなモデルを想定する．ここで，S_1 は初期状態を表し，黒点で表されている S_7，S_8 は到達してはならない状態を示すものとする．

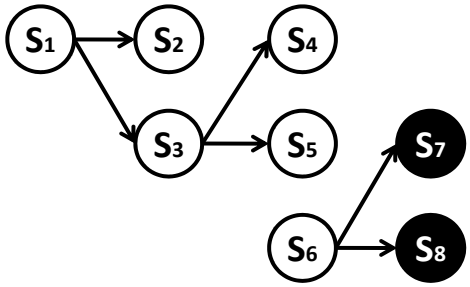

図 2: CEGAR の例：元モデル

図 2 のモデルから，図 3 のモデルのように初期抽象化を行う．例では，状態 S_1 を AS_1, 状態 S_2 と S_3 を AS_2, 状態 S_4 と S_5 と S_6 を AS_3, 状態 S_7 と S_8 を AS_4 に縮約し，状態数を 8 から 4 へと削減している．このとき，初期状態 S_1 を内包する AS_1 は抽象モデル上で初期状態として振る舞い，AS_4 は S_7 と S_8 を内包するため，到達してはならない状態である事を示す．抽象モデルの遷移は元モデルの遷移を内包して表現されるため，遷移の表現は図 4 のモデルのようになる．図 4 の抽象モデルに対してモデル検査を行い，初期状態から到達してはならない状態へ到達可能かどうかについて調べる．

抽象モデル上では初期状態 AS_1 から AS_4 への到達が可能である．そのため，抽象モデルに対するモデル検査は False と反例を返す．反例の系列は $AS_1 \to AS_2 \to AS_3 \to AS_4$ となる．出力された反例に対して，CEGAR ループのアルゴリズムに従ってシミュレーションを元モデル (図 2) 上で行い，反例が元モデルでも存在するかどうかを調べる．

シミュレーションの結果，具体化された反例系列では，初期状態 S_1 から S_9 には到達する系列がないため，抽象モデル上でのモデル検査から得られた反例は本来存在しない反例，すなわち偽反例であることが分かる．よって，抽象モデルの洗練を行う．一般的な CEGAR アルゴリズムでは抽象モデ

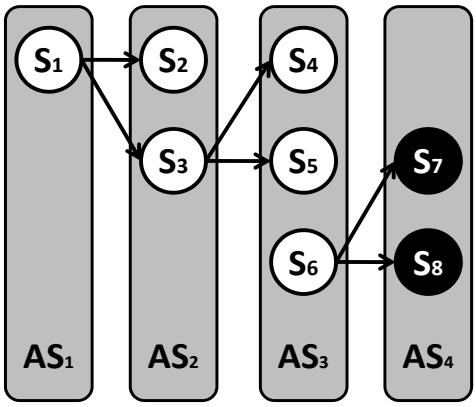

図 3: CEGAR の例：初期抽象化

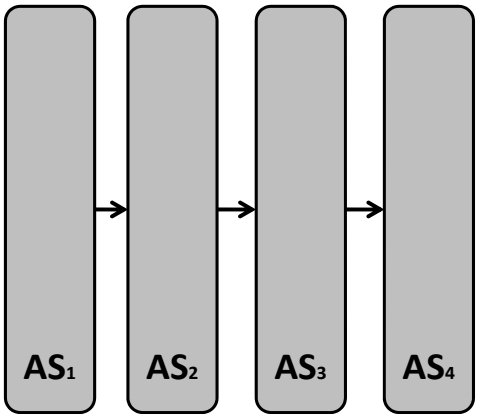

図 4: CEGAR の例：抽象モデル

ルの状態を分割することで洗練を行う．例では，抽象モデル上の状態 AS_3 に対して洗練を行い，新たに抽象モデル上の状態 AS_3' に分割する．洗練を行った抽象モデルは図 5 となる．なお，洗練方法は扱うモデルによって具体的な手法は異なる．また，最適な洗練手法は NP-hard である [1].

洗練を行った抽象モデル (図 4) に対して，再度モデル検査を行う．この例では，洗練を行ったモデルは初期状態から AS_4 に到達することがないため，モデル検査では True を返し，処理を終了する．本例では，終了時の抽象モデルの状態数が 5 であり，元モデルに比べて少ないことが分かる．

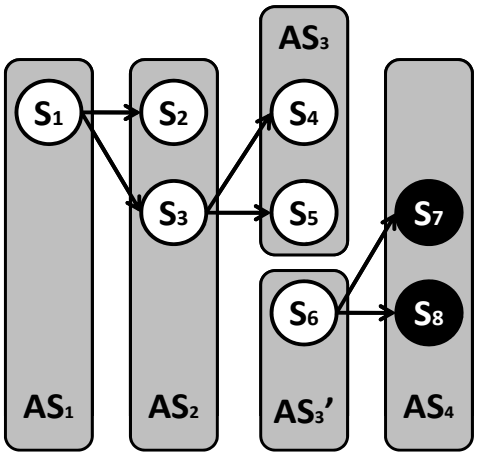

図 5: CEGAR の例：洗練を行った抽象モデル

4 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 [18] 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，複数反例抽出による 1 ループ内での洗練を同時に複数回行っている．

4.1 基本アルゴリズム

時間オートマトンの抽象化洗練手法について基本的なアルゴリズムを示す．提案する抽象化洗練手法では，文献 [18] で示されている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複数反例抽出を行っている．概要を以下に述べる．

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複数の異なったコンフィギュレーションを用いて行う．1 つでもモデル検査の結果を性質を満たし True であれば True を返す．そうでなければ，反例を出力する．この段階で，反例が複数存在する場合は全て出力する．また，同一の抽象モデルに対してモデル検査を行っているので，True と False が混在することはない．

3. 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，抽象化していない実際のモデルでも成立する反例が 1 つでもあるのならば False を，そうでなければ，すなわち全ての反例が偽反例であるのであれば，全てのシミュレーション結果を返す．

4. もし反例が全て実際のモデルでは存在しない偽反例であった場合，求められたシミュレーション結果を統合し，抽象モデルの洗練を行う．

5. 洗練を行った抽象モデルをに対して再度モデル検査を行う．以下この処理が終了するまで繰り返す．

以下，提案手法の各操作について詳細に述べる．

4.1.1 初期抽象化

初期抽象化では，文献 [18] 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 4.1 (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 4.2 (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．

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

i 番目の洗練ループでは i 番目の時間オートマトン $\mathscr{A}_i = (A_i, L_i, l_{i,0}, C_i, I_i, T_i)$ に対して定義 4.2 に従って i 番目の抽象モデル $\hat{M}_i = (\hat{S}_i, \hat{s}_{i,0}, \hat{\Rightarrow}_i)$ が生成される．

定義 4.3 (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．

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

4.1.2 モデル検査

通常，モデル検査ツールにおいて反例の出力はたかだか 1 つであるが，今回は並列計算機を用いる手法や *k*-最短路探索手法を用いた反例出力を行うことで，ある抽象モデル $\hat{M}$ に対して複数の反例を抽出することが可能となる．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．

4.1.3 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 [18] で提案されているシミュレーションアルゴリズムにしたがって，DBM の演算によって到達可能か判定を行う．到達不可能であると判定された状態，つまり時間制約を満たさない状態について，以下に定義する．

定義 4.4 (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．

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．シミュレーション処理はモデル検査処理と比べて計算コストが少ないため，逐次的に行ってもボトルネックにはなりにくいと判断した．

4.1.4 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 [18] 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．洗練で用いる情報はシミュレーション時に元モデルでは存在しない反例と判断された部分であり，これは状態と遷移の集合として以下のように表すことができる．

定義 4.5 (反例から抽出される洗練情報)**.** 抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \Rightarrow)$ 上の反例 $\hat{\rho}$ よりシミュレーションから求められる洗練情報は $\hat{S}$ の連続する状態と遷移の系列である．反例 $\hat{\rho}$ 上の *badstate* を $\hat{s}_i$ とする場合，ある長さ n 反例の洗練情報 $\hat{\sigma}$ は以下のように表わされる．

$$\hat{\sigma} = \langle \hat{s}_i \to \hat{s}_{i+1} \to \hat{s}_{i+2} \to \cdots \to \hat{s}_{n-1} \to \hat{s}_n \rangle$$

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計算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．

洗練時に行う処理　抽象モデルを洗練するとき，次の 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 [18] において定義されている (アルゴリズム 1:図 6).

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 7) を与える．

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 6: アルゴリズム 1 ： 洗練アルゴリズム

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.

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 4.6 で与える.

定義 4.6 (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 7: アルゴリズム 2： 洗練アルゴリズム (複数パス)

定義 4.7. 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 4.1. 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．

証明 4.1. 以下の定理 *4.2*，*4.3* より明らかである． □

定理 4.2. 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 *6* を適用しても大域的に正しい洗練になる．

定理 4.3. 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．

定理 4.2 は自明であるため，ここでは定理 4.3 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 2 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．

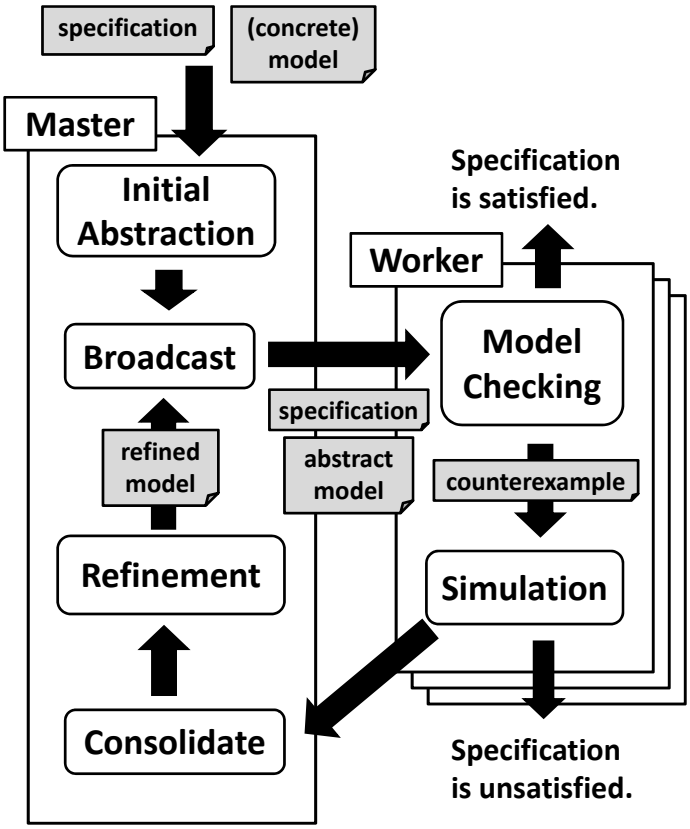

図 8: 並列計算機環境を利用した CEGAR アルゴリズム

4.2 並列計算機環境下での実装

並列計算機環境下での実装した事例について述べる．

並列環境上での CEGAR の実行方法としては，マスター・ワーカ方式を採用する．当初の目的である，複数反例抽出を実現するため，反例を抽出するモデル検査の処理をワーカ計算機の担当とする．これは時間オートマトンに対する CEGAR の処理において，ボトルネックとなるモデル検査の処理を軽減する狙いもある．また，処理の効率化を考慮し，シミュレーションの処理もワーカ計算機の担当とする．以下に処理の流れを示す (図 8).

1. モデルと満たすべき性質を入力としてマスター計算機に与える．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. 抽象化した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，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．各ワーカ計算機に配布される抽象モデルは同一である

ため，仮に 1 台のワーカ計算機が True を返す場合，他のワーカ計算機でも True が返されることが期待される．

4. 反例が出力された場合，各ワーカ計算機上で反例を基にシミュレーションが行われる．このとき，異なる探索戦略でモデル検査を行うことで，出力される反例が各ワーカ毎で異なることが期待される．

5. モデル検査・シミュレーションの結果をマスター計算機に返す．このとき，モデル検査の結果が True である場合はモデルは性質を満たすとして True を，ワーカ計算機の 1 台でもシミュレーション結果が False であった場合はモデルは性質を満たさないとして，False と反例を返す．

6. もしモデル検査の結果が False で，シミュレーションの結果反例が全て実際のモデルでは存在しない偽反例であった場合，得られたシミュレーション結果による抽象モデルの洗練を行う．

7. 洗練を行った抽象モデルを再度各ワーカ計算機に配布し，モデル検査を行う．以上の動作を繰り返す．

並列計算機による実行について，各ワーカー計算機の同期はシミュレーション結果を統合している段階で行う．これにより，1 ループ内で全ての処理を完結することができ，シミュレーションの結果や洗練の結果生じる齟齬の発生防いでいる．

実装例では反例を抽出できる最大数がワーカ計算機の台数に依存してしまう問題点が存在する．また，各ワーカ計算機同士で同期を取っていないため，各ワーカで同一反例が出力される可能性もある．しかしながら，各ワーカで反例出力時に同期を取る場合，1 ループ内でで複数回同期を取ることによるオーバーヘッドがかかってしまい，本稿の研究の目的である実行時間の減少に逆行すると考えられる．そのため，計算機の台数を理論的には上限なく増大させることができると仮定し，同一の反例が出力されることに対しては考慮せず，反例抽出時の同期は行わない．

並列計算機上では，反例の抽出戦略などを各ワーカ毎に変化をさせることが可能になる．また，後述する *k*-最短路探索手法と併用することで，さらなる処理の高速化が見込める可能性もある．本稿ではシンプルな実装にとどめるが，複数のループにまたがった洗練情報の抽出なども行うことが可能であると考えている．

4.3 *k*-最短路探索手法による実装

次に，*k*-最短路探索手法での実装事例について述べる．*k*-最短路探索手法による実装では，並列計算機環境での実装時に問題であった同一反例出力を回避する有用な手段であると考えている．以下に処理の流れを示す (図 9).

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．

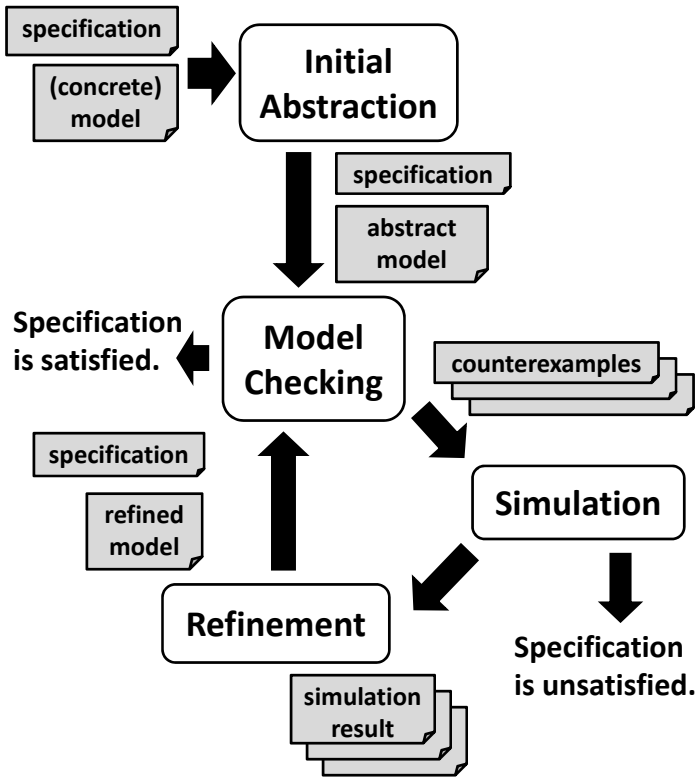

図 9: *k*-最短路探索手法を用いた CEGAR アルゴリズム

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．

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，今回は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 [19] や Jimenez の手法 [20] 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，今回対象とす

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 10: アルゴリズム 3 ： *k*-最短路探索手法によるモデル検査アルゴリズム

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．

本論文では，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．

幅優先探索を行うことにより，反例は最短のものから順に探索することが期待できる．また，反例抽出とシミュレーション，洗練までの処理を一台の計算機で行えるため，抽出された反例に対する洗練情報の適用順序の操作や，反例抽出の同期等も取りやすいという利点がある．

5　評価実験

本章では，提案手法についての評価実験を行う．

5.1　CEGAR の実装

実装は，文献 [18] 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[11] の DBM ライブラリを利用し，時間オートマトンの信頼性を保証している．

5.2　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 [16] と Gear Controller[17] 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 [18] 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．

5.3　並列計算機環境での実験

まず，並列計算機環境で実装した複数反例抽出による時間オートマトンの到達可能性解析手法について述べる．

5.3.1　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．

5.3.2　対象とする時間オートマトン

対象とする時間オートマトンは，Fischer の相互排除プロトコルの 3 並列と 4 並列，Gear Controller とする．Gear Controller は満たすべき性質を 5 種類用意し，性質の違いによる洗練の違いについても評価を行う．

5.3.3　モデル検査ツール

モデル検査は，モデル検査ツール UPPAAL[11] 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．

5.3.4　評価項目について

提案手法では，複数反例を抽出し CEGAR ループのループ回数を減少させることで，実行時間を減少させることを目的としている．よって，評価項目としては CEGAR ループのループ回数と実行時間を取り上げる．

5.3.5　実験結果

実験結果について以下に示す．

実験結果の図は，横軸がワーカ計算機の台数であり，縦軸はそれぞれの計測したい項目についてである．

ループ回数　まず，ワーカ計算機を増やした際の時間オートマトンに対する CEGAR のループ回数について調べる．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 11，12 は，ワーカ計算機台数と実行したループ回数の関係を表している．図より、Fischer，Gear Controller 双方について計算機の台数を増やすごとにループ回数が減少していることがわかる．

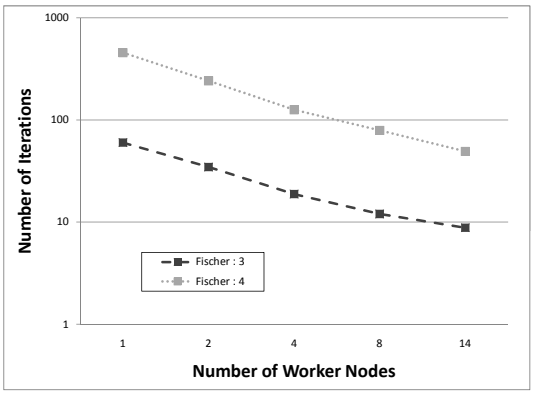

図 11: ループ回数 (並列環境) ： Fischer

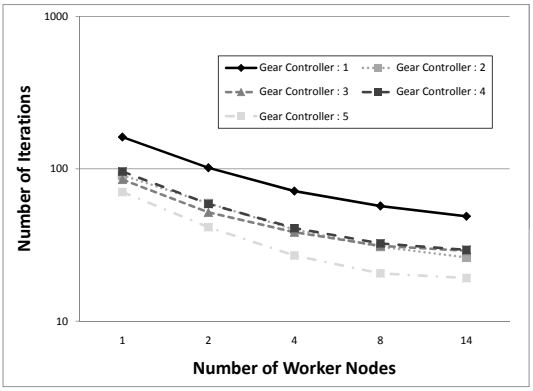

図 12: ループ回数 (並列環境) : Gear Controller

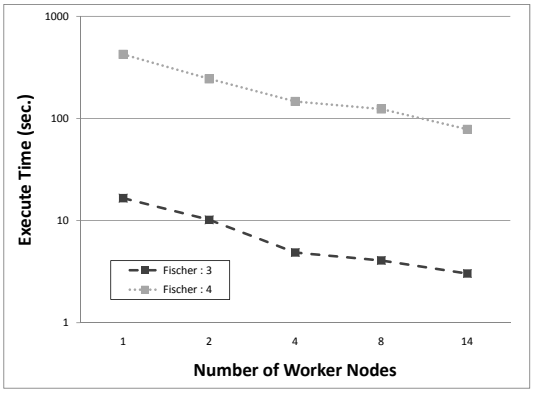

図 13: 実行時間 (並列環境) ： Fischer

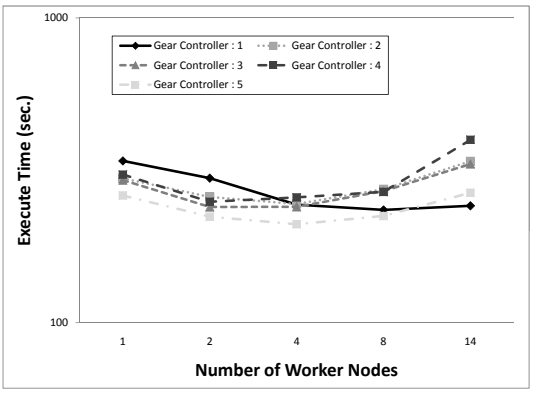

図 14: 実行時間（並列環境）: Gear Controller

実行時間　次に，実行時間の台数効果について調べる．図 13，14 では，ワーカ計算機台数と実行時間の関係を表している．Fischer の相互排除プロトコルでは実行時間はループ回数に比べて減少幅が鈍化しているが，減少傾向であることが分かる．一方，Gear Controller はワーカ計算機を 4 台辺りから増加傾向に転じ，以降はワーカ計算機の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．

5.3.6　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減少幅の鈍化，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，本来は余分な洗練操作の増加により，モデル検査 1 回辺りの実行時間が増加したのではないかと考えられる．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

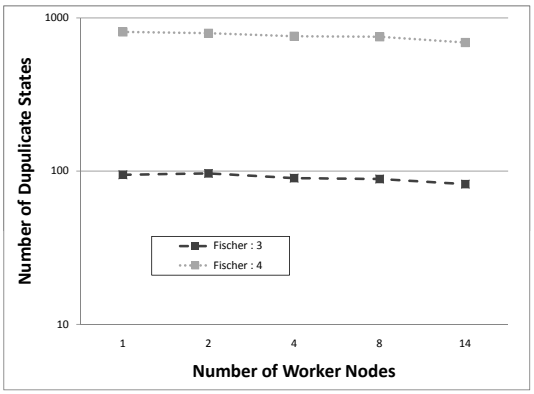

図 15: 生成状態数 (並列環境) ： Fischer

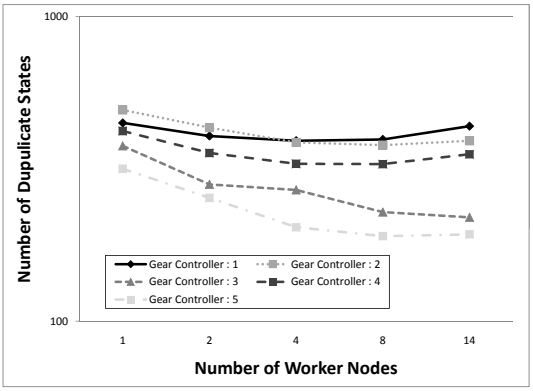

図 16: 生成状態数 (並列環境) : Gear Controller

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全く同一のものであった場合，1 回のループ内で洗練が行われる回数が本来のワーカ計算機台数分行うことができず，ループ初期段階での複数反例抽出による洗練の効果が出ていない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 つの項目について実験を行う．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．

5.3.7 考察に対する評価実験の結果

考察を裏付ける評価実験の結果を以下に示す．実験結果は，横軸を前回と同じくワーカ計算機の台数で，縦軸はそれぞれの計測したい項目についてである．縦軸は状態生成数と同一反例の割合についてであり，全反例数に対する反例の種類は，各ワーカ計算機から出力された反例全てに対し，同一の反例を除外した残りの反例数の割合についてである．値は割合で表記しており，値が 1 に近いほど同一の反例の割合が少なく，ワーカ計算機による台数効果が得られることになる．

定義 5.1 (全反例数に対する反例の種類)**.** 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，出力された全反例数 n に対して同一な反例を除外した反例の種類数を m とすると，全反例数に対す

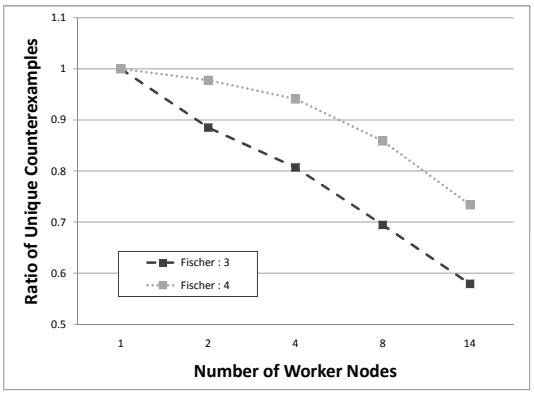

図 17: 全反例数に対する反例の種類：Fischer

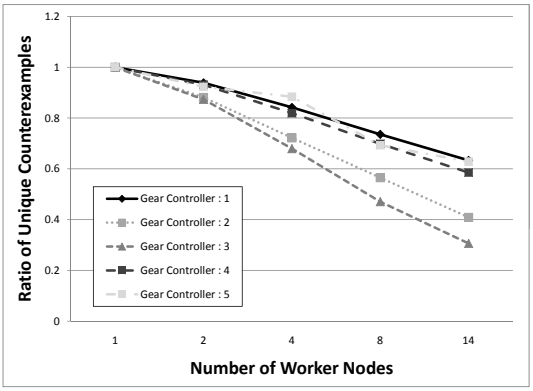

図 18: 全反例数に対する反例の種類：Gear Controller

る反例の種類の割合 $cnum$ は以下のように定義される．

$$cnum = \frac{m}{n}$$

状態生成数　生成状態数の増加量について調べる．図 15，16 は，ワーカ計算機台数と生成状態数の関係を表している．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 15，16 から，Fischer の相互排除プロトコルでは状態生成数は緩やかに減少していることが分かる．Fischer の相互排除プロトコルでは実行時間はワーカ計算機台数と共に減少していたことを考えると，反例数を増やして初期に洗練を進めることで，生成状態数の減少に繋がることを示している．一方，Gear Controller の場合，性質 1，2，4 において状態数がワーカ計算機の台数を増やしても増加する傾向がグラフから見てとれた．性質 2，4 はワーカ計算機の台数と実行時間の関係が 4 並列を境に増加傾向に転じているため，本来では重要ではない洗練による状態生成がモデル検査時間の増加に繋がり，結果として実行時間を増加させている原因になっていると考えられる．

同一反例の割合　次に，同一反例の割合について調べる．図 17，18 は，ワーカ計算機台数と同一反例の割合の関係を示している．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高いといえる．まず，Fischer の相互排除プロトコルではワーカ計算機の台数を増やすとともに同一反例の割合も増加している．これにより，ワーカ計算機の台数を増やした場合，実行時間の減少が鈍化する理由になると考えられる．一方，Gear Controller では，性質 2，3 において同一反例が出力される割合が高いことが図から見てとれる．生成状態数が減少傾向であった性質 3 で実行時間が増加していた理由については，同一反例が多数出力されることによる台数効果の減少であると説明することができる．

5.3.8 並列計算機上の実装に対する考察

以上の 4 つの実験を基に，考察を行う．並列計算機上の実装に対する実験では，提案アルゴリズムの狙いであるループ回数の減少は実現できたが，実行時間の減少についてはモデルによって実現できていない場合があった．阻害要因としては本来不必要な洗練を行ったことによる生成状態数の増加と，同一反例出力によるオーバーヘッドがあったことが考えられる．これらは，実験時に反例抽出をランダムで行ったことに起因すると考えられる．実際，同じ条件下で複数回指向した場合でも，実行時間やループ回数の出力についてかなりの差があることがあり，より厳密に評価実験を行う必要がある．

5.4 *k*-最短路探索手法での実験

次に，*k*-最短路探索手法で実装した複数反例抽出による時間オートマトンの到達可能性解析手法について述べる．

5.4.1 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

5.4.2 対象とする時間オートマトン

本実験では，Fischer の相互排除プロトコル 3 並列から 7 並列について，反例数と各項目の関係についてののデータを掲載する．また Gear Controller に関しては並列計算機上での評価実験と同様，満たすべき性質 5 つそれぞれに対して実験を行う．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．

5.4.3 モデル検査ツール

5.3 章の並列計算機上の実装による実験で利用していたモデル検査モジュール [11] 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，4.3 章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).

5.4.4 評価項目について

並列計算機上の実装と同じく，評価項目は時間オートマトンに対する CEGAR のループ回数，実行時間，更に生成状態数を選択する．同一反例の割合は，*k*-最短路探索手法の性質上，同一反例が出力

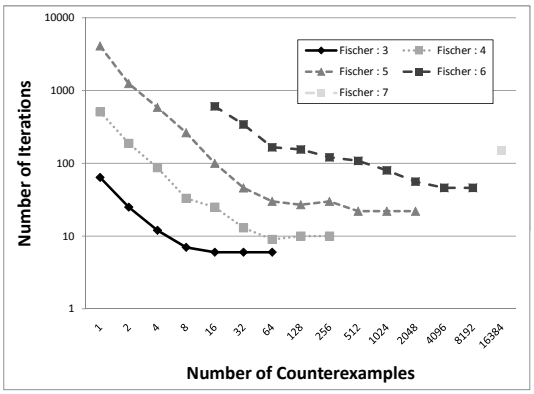

図 19: ループ回数 (k-最短路探索手法)：Fischer

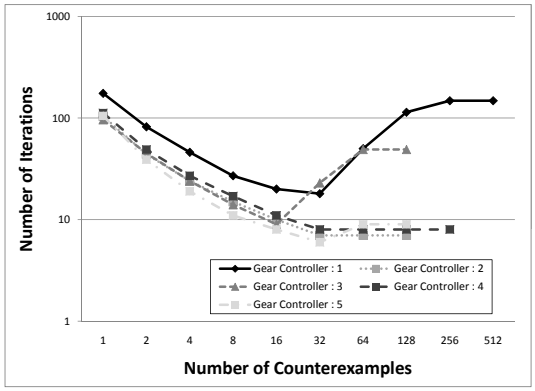

図 20: ループ回数 (k-最短路探索手法)：Gear Controller

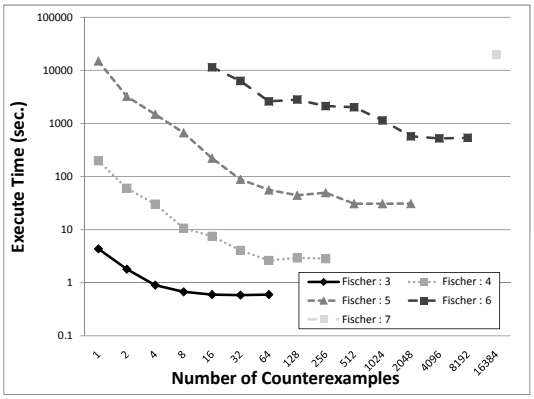

図 21: 実行時間 (k-最短路探索手法) ： Fischer

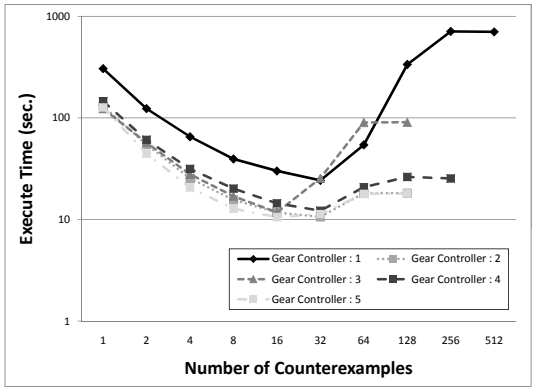

図 22: 実行時間 (k-最短路探索手法)： Gear Controller

されることはないので除外する.

5.4.5 実験結果

実験結果について以下に示す.

実験結果の図は，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. 実験で用いた時間オートマトンの一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.

ループ回数 まず，抽出する反例を増やした際にループ回数の増減量について調べる．図 19，20 は，抽出した反例数に対するループ回数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，反例数が増加するに従ってループ回数が減少している．しかしながら，Gear Controller での

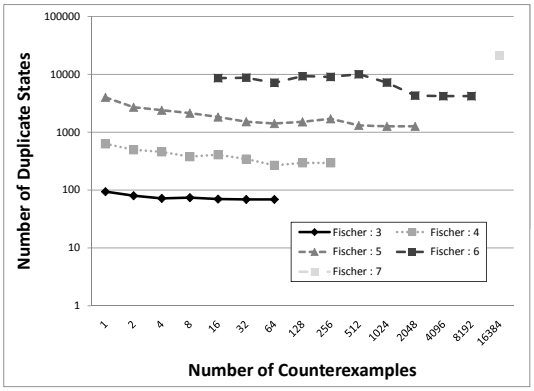

図 23: 生成状態数 (k-最短路探索手法)：Fischer

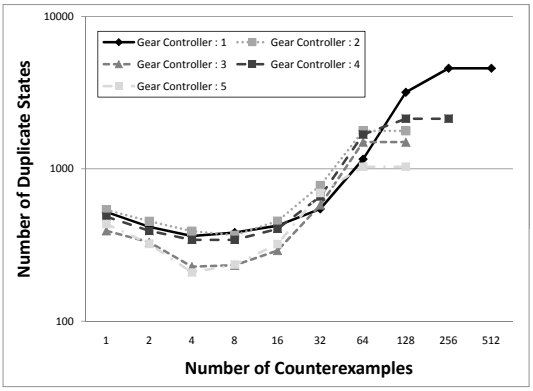

図 24: 生成状態数 (k-最短路探索手法)：Gear Controller

表 1: 最大反例数：Fischer の相互排除プロトコル

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

実験においては，主に反例数 32 を境に横ばいになったり，上昇に転じている．

実行時間　次に，実行時間について評価を行う．図 21,22 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ループ回数と同様に減少しているのに対し，Gear Controller では反例数 32 を境に全ての性質で上昇に転じている．

状態生成数　最後に，洗練によって生成される状態数についての評価を行う．図 23,24 は，抽出した反例数に対する状態生成数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が増加するに従って状態数が緩やかに減少するが，Gear Controller ではやはり反例数 32 を境に上昇に転じている．

5.4.6　考察

以上の実験結果から，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 のループ回数を減少させ，実行時間も減少させることに成功している．また，反例数を増やすごとに洗練で行われる状態の生成についても減少する結果を得ることができ，既存手法に対する状態数の縮減という意味においても優れている，と言える．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

表 2: 最大反例数：Gear Controller

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

ているのだが，それ以上の反例数で実行しようとするとループ回数，実行時間，状態生成数全てにおいて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，GearCotroller 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．

最後に，実用的な時間を考慮に入れず，Fischer の相互排除プロトコルの 8 並列のモデルに対して手法を適用してみた．表 3 はその結果についてである．Fischer の相互排除プロトコルでは，積オートマトンを取った状態で UPPAAL[11] のモデル検査ツールで動作をさせることができない．したがって，提案手法を用いることで今まで検査が不可能であった規模の時間オートマトンに対して，モデル検査が可能になったと言える．

表 3: Fischer : 8　に対する実験結果

Evaluation	Number of Iterations	Execute Time(sec.)	Number of Duplicate States
Fischer : 8	279	177728	40916

5.5　評価実験のまとめ

最後に，全体の実験についてまとめを行う．まず，並列環境下での実装と *k*-最短路探索手法の評価実験を通じて，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，5 並列のモデルに対しては既存手法の 500 倍程度の高速化を確認し，また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 並列から 8 並列までのモデルへの適用も可能となった．一方，Gear Controller 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一定数を境に出ていなかった．これは，本来必要のない反例に対する洗練が行われることにより，不必要な状態の生成が行われモデル検査の実行時間が増大し，結果的に実行時間が増加してしまっているのではないかということが考えられる．この点について，6 章にて評価実験を行う．

6　反例の解析と適用順序の工夫

5 章では，複数反例抽出による洗練手法が実行時間の減少に繋がっていることを示すことができた．本節では，更に実行時間を削減する手法として，反例の解析を行い，また適用順序を変化させることについて実行時間の減少に効果があるかどうかについての議論を行う．

まず，5 章の実験で考察を行った，反例の長さが極端に長いものについての議論を行う．

6.1　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 6.1 (反例の長さ)**.** 抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \Rightarrow)$ 上の反例 $\hat{\sigma} = \langle \hat{s}_0 \rightarrow \quad \hat{s}_1 \rightarrow \hat{s}_2 \quad \rightarrow \cdots \rightarrow \hat{s}_{n-1} \rightarrow \hat{s}_n \rangle$ 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 5 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 6.2 (反例の長さの閾値)**.** 抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \Rightarrow)$ 上の最短の反例の長さを $length_{min} = min$ とするとき，閾値 τ は次のようにあらわされる．

$$\tau = \quad f \times min$$

ここで，*f* は反例長の基準となる係数パラメータである．

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．

6.1.1　対象とする時間オートマトン

対象とする時間オートマトンは，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．

実験環境，利用したモデル検査ツール，評価項目については 5.4 に準ずる．

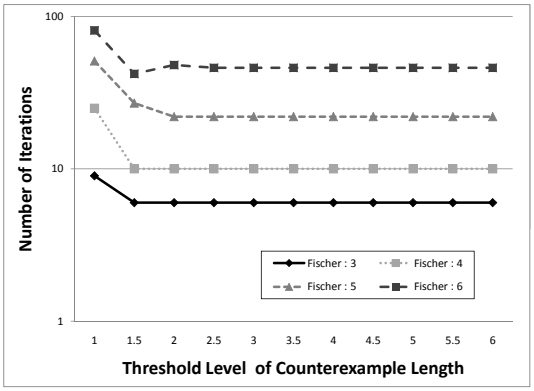

図 25: ループ回数 (k-最短路探索手法：閾値あり) ： Fischer

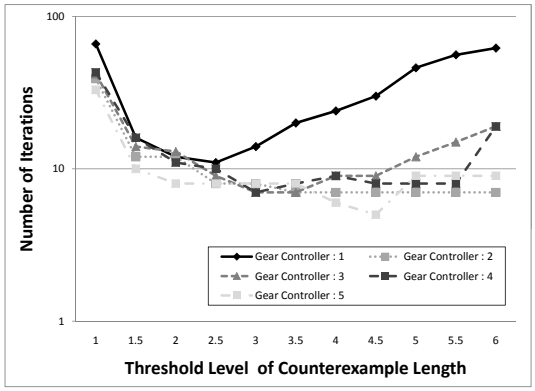

図 26: ループ回数 (k-最短路探索手法：閾値あり) : Gear Controller

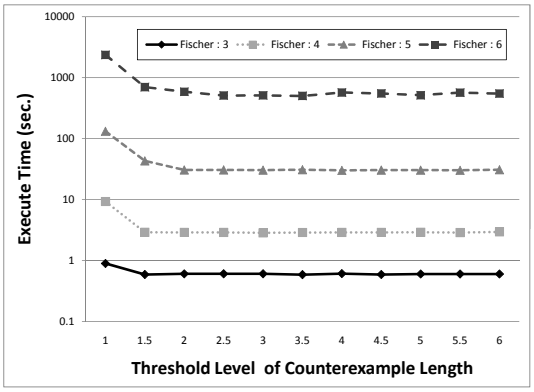

図 27: 実行時間 (k-最短路探索手法：閾値あり) ： Fischer

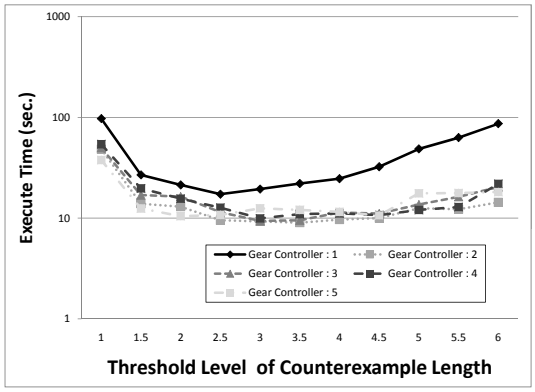

図 28: 実行時間 (k-最短路探索手法：閾値あり) : Gear Controller

6.1.2 実験結果

実験結果について以下に示す.

実験結果の図は，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．

ループ回数　まず，抽出する反例を増やした際にループ回数の増減量について調べる．図 25，26 は，反例の長さの倍率に対するループ回数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，実験結果において差異が出てこない．しかしながら，Gear

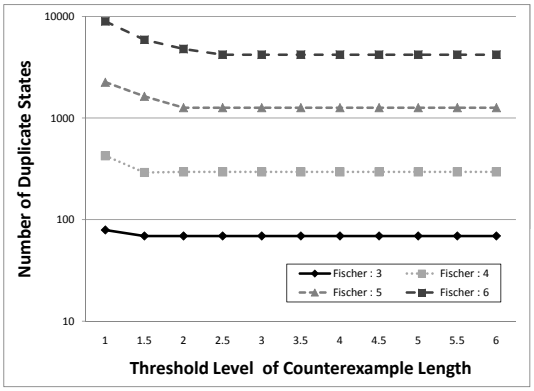

図 29: 生成状態数 (k-最短路探索手法：閾値あり) ： Fischer

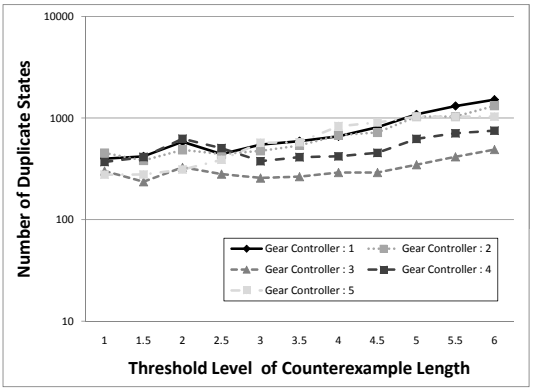

図 30: 生成状態数 (k-最短路探索手法：閾値あり) : Gear Controller

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．

実行時間　次に，実行時間について評価を行う．図 27,28 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ではループ回数と同じく，反例長の閾値を 2 倍以上にしても値の変化がないことに対し，Gear Controller では反例長の閾値を倍率 3 倍程度に限定した場合，最も効果があることが実験結果から見てとれる．

状態生成数　最後に，洗練によって生成される状態数についての評価を行う．図 29,30 は，反例の長さの倍率に対する状態生成数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ループ回数や実行時間に対して傾向が変わらないのに対し，Gear Controller では反例長の閾値の倍率を上げるごとに生成状態数は緩やかに増加している．

実験により，反例の長さに閾値を設けることで，本来あまり重要ではない反例の選別に効果があることを評価することができた．これにより，Fischer の相互排除プロトコルのようなシンメトリな構造を持つモデルや，Gear Controller のようなシンメトリな構造を持たないモデルに対しても同様のアルゴリズムを適用することが可能であることが解り，手法の有用性について評価ができたといえる．

6.2　洗練情報の特性を用いた複数反例の洗練手法

4 章では，洗練時に反例の適用順序により出力されるモデルが誤っている可能性はないが，異なる可能性があることを示している．これは，反例からシミュレーションを通じて求めることのできる洗練情報に重複がある可能性があるためである．ここで，洗練情報を以下のように定義する．

4 章の定義より，反例の適用順序を入れ替えることで，より効率的に洗練をすすめ，実行時間の削減に繋がるのではないかと考えられる．また，適用順序を定義するための評価項目については，モデル検査から抽出された反例の段階で評価を行うよりも，シミュレーションを行い，洗練に必要な情報を抽出した後の方が洗練すべき個所などが明確であり，反例の適用順序を入れ替えるうえで効果が高いと推察する．

反例から抽出される洗練情報は 5 章の評価実験では考察の対象外となっており，反例から抽出される洗練情報は元の反例の長さによって昇順に並べ替えられている．しかしながら，これは反例に対する優先度の重みづけであり，洗練時情報に対する優先度の重み付けではないため，洗練情報に即した基準が必要であると考えている．そのため，反例の適用順序を変化させることで，よりモデルを効率的に洗練することが期待できる．本節では，洗練時に反例の優先度をつけることで，実行時間のさらなる減少や状態生成数の削減を行うことについて議論を行う．

6.2.1 優先度の項目について

まずは，優先度の項目について議論を行う．

洗練情報の深さ　まず，反例に対する洗練情報の開始地点の深さについて取り扱う．時間オートマトンに対する CEGAR[18] において，反例をシミュレーションした結果は，元モデルでは存在しえない状態と遷移の集合として表される．ここで，洗練情報の深さについて，以下のように定義する．

定義 6.3 (洗練情報の深さ)**.** 抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \hat{\Rightarrow})$ 上の洗練情報 $\hat{\sigma} = \langle \hat{s}_i \rightarrow \hat{s}_{i+1} \rightarrow \hat{s}_{i+2} \rightarrow \cdots \rightarrow \hat{s}_{n-1} \rightarrow \hat{s}_n \rangle$ に対し，洗練情報の深さ *depth* は次のようにあらわされる．

$$depth(\hat{\sigma}) = i$$

元モデルでは存在しえない状態と遷移の集合の始点となる状態が初期状態から近ければ近いほど，他の反例もその遷移を含んでいる可能性があり，先に洗練を行うことで他の反例による洗練の省略が期待できる．

洗練情報の長さ　次に，反例に対する洗練情報の長さについても取り扱う．洗練情報を長さについて以下に定義する．

定義 6.4 (洗練情報の長さ)**.** 抽象モデル $\hat{M} = (\hat{S}, \hat{s}_0, \hat{\Rightarrow})$ 上の洗練情報 $\hat{\sigma} = \langle \hat{s}_i \rightarrow \hat{s}_{i+1} \rightarrow \hat{s}_{i+2} \rightarrow \cdots \rightarrow \hat{s}_{n-1} \rightarrow \hat{s}_n \rangle$ に対し，洗練情報の長さ *length* は次のようにあらわされる．

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

反例の抽出時は反例の長さが対象となるが，シミュレーションの結果洗練情報の長さが極端に短い場合も存在する．そのため，反例の長さで評価を行った 5 とはまた異なる結果が出ることが期待される．

洗練情報の新しさ　更に，洗練情報の新しさについても判定を行う．洗練情報内に含まれる状態や遷移が後に生成されたものであるのならば，その反例は前回の洗練によって発生したということを判定することが可能であると考える．また，古い遷移や状態を通る洗練情報においては，新たに追加された遷移により反例が新たに生成された，ということが確認でき，前回の洗練の効果が出ていないことなどを判定することが可能である．検証では，洗練情報内の全状態の新しさの平均，洗練情報の始点の新しさ，また，全遷移の新しさの平均と始点となる遷移の新しさについて評価を行う．新しさの評価については，生成時に付与される $Integer$ 型 ID を利用した．

6.2.2　実験環境について

実験環境について述べる．

モデル検査環境　利用する複数反例抽出手法は *k*-最短路探索による実装で行う．これは，並列計算機は抽出する反例数が計算機の台数に限定されるのに対し，*k*-最短路探索による反例抽出手法は反例を抽出できる最大数まで得ることが可能であるためである．

対象とする時間オートマトン　モデル検査に利用する時間オートマトンは Fischer の相互排除プロトコルの 3 並列から 6 並列までと，Gear Controller を用いる．Gear Controller は検査を行う性質を 5 つ用いることで，同一モデルに対する優先度の違いについても評価を行う．また，反例抽出数はどちらのモデルでも最大数を抽出するとする．

評価項目について　評価項目はループ数，実行時間，状態生成数について行う．

6.2.3　実験結果

実験結果について以下に示す．表の Normal は優先度を設定しなかったもの，Depth は洗練情報の深さ，Length は洗練情報の長さについてそれぞれ降順，昇順に並べ替えたものである．TotalState は状態，Transition は遷移の新しさについて，stp は開始地点のみ評価の対象とし，avg は洗練情報全ての平均値を対象としたものである．

また，Number of Iterations はループ回数を，Execute Time は実行時間を，Number of Duplicate States は状態生成数を表す．実行時間の 単位は ms である．

洗練情報の深さ　まず，洗練情報の深さについて評価を行う．基本的に洗練情報の深さで並べ替えを行ったときは昇順，降順に関わらず極端に各評価項目が悪くなるということは見られなかった．改善されたモデルとしては，昇順では Gear Controller の性質 5(表 12) に対して，降順では Fischer の 5 並列 (表 15) と Gear Controller の性質 1，性質 5(表 17,21) についてである．特に Gear Controller の性質 1 については 7 分の 1 程度に実行時間が減少されている．

洗練情報の長さ　次に，洗練情報の長さについて評価を行う．洗練情報の長さは洗練情報の深さと同じような性質を示し，結果も似通っている．唯一異なっているのは降順での Gear Controller の性質 3(表 19) であり，ここでは洗練情報の深さがほぼ優先度を設定しなかったものと変わりがないのに対し，洗練情報の長さはループ回数，実行時間ともに 4 分の 1 程度に減少している．

しかしながら，洗練情報のの長さ，洗練情報の深さについては一定の傾向が見られたわけではないため，今後さらに複数の異なった性質を持つモデルに対して評価実験を行うことで，その実行時間の削減の効果を見極める必要がある．

洗練情報の新しさ　最後に，洗練情報の新しさについて評価を行う．洗練情報の新しさについても，並べ替えを行ったときに昇順，降順に関わらず各評価項目が悪くなるということは見られなかった．実行時間が短縮された事例としては，昇順では Fischer の 5 並列 (表 6), 降順では Fischer の 6 並列 (表 16) ，Gear Controller の性質 1(表 17) があげられる．

全体的に，昇順に優先度を付けた場合より，Fischer の相互排除プロトコルについては降順で優先度を付けた場合，状態や遷移の新しさの平均においては処理の高速化や状態数の削減などの効果が見られた．

6.2.4　考察

優先度の設定についての考察をまとめる．実験では 6 つの項目を抽出し，それぞれについて実験を通して優先度を設定するための有用性を測った．結果として，Fischer の相互排除プロトコルのようなシンメトリな構造を持つモデルに対して，生成された状態や遷移が新しいものから順に洗練を行うことで，時間オートマトンに対する CEGAR を更に高速化し，状態数の生成を抑えることができるということが分かった．しかしながら，Gear Controller のようなシンメトリな構造を持たないモデルに対しては，今回実験を行った 6 つの項目では有用なものがなかったと結論付けられる．今後は，更に優先度の項目となりえる指標の洗い出しと共に，複数の項目を組み合わせ，評価関数を作ることで，時間オートマトンに対する CEGAR の処理の一つとして組み込むことが可能になると考えている．しかしながら，優先度の評価実験でもわかるように，優先度は各モデルに大きく依存するため，より多くの時間オートマトンで評価実験を行い，採取したデータを基にして回帰分析などを行い，統計的な指標を得ることが重要であると考えられる．

7　あとがき

本研究では，時間オートマトンの時間抽象化とその洗練手法に対して，複数反例抽出による高速化手法を提案した．基本アルゴリズムを提案と共に，並列計算機を用いた実装や k-最短路探索手法を用いた手法での実装を行った．実装した手法に対して評価実験を行い，提案アルゴリズムの有用性について評価を行うとともに，実装による違いの評価や，反例の長さを用いた反例の選別について言及を行った．

加えて，抽出された反例から得られる洗練情報に対して，洗練情報の初期状態からの距離や生成された新しさなど様々な項目を用いて反例の優先度を設定し，評価を行った．まだ有用な指標が得られたとは言い難いが，項目単体でも効果が得られた場合があり，今後の研究において有用な洗練を得る布石になると考えている．

今後の課題としては，k-最短路探索手法を用いたモデル検査ツールの改良と，反例の優先度を求める評価式の策定があると考えられる．k-最短路探索手法を用いたモデル検査ツールは必要最低限の処理しか行っておらず，今後並列計算機上で動作することを想定した実装を行ったり，よりメモリ使用量の削減させるような手法を用いることで CEGAR の高速化が得られると思われる [24][25]．反例の優先度を求める評価式は，優先度を設定する上で有用だと判断された各項目に対して，回帰分析的に評価式を策定することで自動化することができ，CEGAR のフレームワークに組み込むことが可能ではないかと考えている．

謝辞

本研究を行うにあたり，理解あるご指導を賜り，常に励まして頂きました楠本真二教授に心から感謝申し上げます.

本研究の全過程を通じ，適切かつ丁寧なご指導を頂きました岡野浩三准教授に深く感謝申し上げます.

本研究において，常に適切なご助言およびご指導を頂きました肥後芳樹助教に深く感謝致します.

本研究の全過程を通じ，多大な協力，助言を頂きました大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士後期課程 3 年長岡武志氏に深く感謝申し上げます.

その他楠本研究室の皆様のご協力に，心より感謝致します.

最後に，コンピュータサイエンス専攻にてお世話頂きました諸先生方にお礼申し上げます.

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] E M. Clarke, O. Grumberg, and D A. Peled: “Model Checking,” MIT Press, 2000.

[5] E.M. Clarke, E. Emerson, and A. Sistla. “Automatic verification of finite-state concurrent systems using temporal logics,” ACM Transactions on Programming Languages andSystems, Vol. 8(2), pp. 244-263, 1986.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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

[12] 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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] D. Eppstein, “Finding the k shortest paths,” focs, pp.154-165, 35th Annual Symposium on Foundations of Computer Science (FOCS 1994), 1994.

[20] 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.

[21] E.Q.V. Martins, M.M.B. Pascoal and J.L.E. “Dos Santos. Deviation algorithms for ranking shortest paths.” International Journal of Foundations of Computer Science. 10(3): pp. 247-262, 1999.

[22] V.M. Jimenez and A. Marzal, “A Lazy Version of Eppstein’s K Shortest Paths Algorithm.” WEA 2003, LNCS 2647: pp. 179-191, 2003.

[23] T. Han and J. Katoen, “Counterexamples in probabilistic model checking.” Toolsand Algorithms for the Construction and Analysis of Systems (TACAS), pp. 72-86, 2007.

[24] Rong Zhou and Eric A. Hansen, “Breadth-first heuristic search.” Journal Artificial Intelligence, vol. 170: pp. 385-408, 2006.

[25] R. Korf, W. Zhang, I. Thayer and H. Hohwald, “Frontier Search” Journal of the ACM, vol. 52, No. 5: pp.715-748, 2005.

[26] Barnat, J. and Brim, L. and Chaloupka, J. “Parallel breadth-first search LTL model-checking.” Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on, pp. 106-115, 6-10, 2003.

[27] Gerd Behrmann, Thomas Hune and Frits W. Vaandrager. “Distributing Timed Model Checking - How the Search Order Matters.” CAV ’00 Proceedings of the 12th International Conference on Computer Aided Verification, pp. 216-231, 2000.

[28] A. Gupta and O. Strichman. “Abstraction Refinement for Bounded Model Checking.” In Seventeenth Conference on Computer Aided Verification, pp. 112-124, 2005.

A 付録

表 4: Fischer-3 並列：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	6	618	69
Depth	6	591	69
Length	6	596	69
TotalState : avg	6	621	68
TotalState : stp	6	552	68
Transition : avg	6	607	70
Transition : stp	6	607	64

表 5: Fischer-4 並列：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	10	2907	295
Depth	10	2936	295
Length	10	2924	295
TotalState : avg	9	2875	334
TotalState : stp	9	2861	331
Transition : avg	9	2874	330
Transition : stp	9	2674	282

表 6: Fischer-5 並列：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	22	30923	1264
Depth	22	30216	1264
Length	22	30156	1264
TotalState : avg	19	30486	1437
TotalState : stp	12	21830	1315
Transition : avg	22	32212	1436
Transition : stp	21	29662	1287

表 7: Fischer-6 並列：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	46	546922	4199
Depth	46	541140	4199
Length	46	540367	4199
TotalState : avg	56	824297	6088
TotalState : stp	50	745203	6102
Transition : avg	56	816495	6095
Transition : stp	53	644590	4982

表 8: Gear Controller-1：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	148	698864	4568
Depth	148	705973	4706
Length	148	670923	4592
TotalState : avg	21	110894	4823
TotalState : stp	21	101847	4755
Transition : avg	148	693420	4554
Transition : stp	21	104360	4758

表 9: Gear Controller-2：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	7	18307	1774
Depth	8	21261	1804
Length	8	18971	1709
TotalState : avg	9	19628	1822
TotalState : stp	9	19765	1806
Transition : avg	8	21097	1784
Transition : stp	8	19051	1845

表 10: Gear Controller-3：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	49	89424	1498
Depth	49	89600	1484
Length	49	88628	1484
TotalState : avg	50	79938	1590
TotalState : stp	49	78402	1587
Transition : avg	49	93809	1475
Transition : stp	49	76840	1545

表 11: Gear Controller-4：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	8	25457	2132
Depth	8	28265	2196
Length	8	24106	2087
TotalState : avg	8	26066	2259
TotalState : stp	9	26905	2251
Transition : avg	8	26560	2145
Transition : stp	8	23426	2229

表 12: Gear Contorller-5：昇順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	9	17786	1032
Depth	5	11170	976
Length	5	11275	976
TotalState : avg	11	21462	1079
TotalState : stp	7	15068	1047
Transition : avg	5	11165	976
Transition : stp	7	15162	1047

表 13: Fischer-3 並列：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	6	618	69
Depth	6	572	68
Length	6	594	68
TotalState : avg	6	620	70
TotalState : stp	6	540	64
Transition : avg	6	567	65
Transition : stp	6	689	64

表 14: Fischer-4 並列：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	10	2907	295
Depth	10	3046	332
Length	10	3086	332
TotalState : avg	10	2769	277
TotalState : stp	11	2864	270
Transition : avg	11	3050	299
Transition : stp	11	3078	303

表 15: Fischer-5 並列：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	22	30923	1264
Depth	12	22048	1315
Length	12	22736	1315
TotalState : avg	18	23199	1148
TotalState : stp	21	25907	1159
Transition : avg	21	28362	1282
Transition : stp	22	32112	1471

表 16: Fischer-6 並列：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	46	546922	4199
Depth	50	725251	6099
Length	50	786535	6118
TotalState : avg	30	424570	3959
TotalState : stp	30	387435	3754
Transition : avg	32	441573	4063
Transition : stp	33	464214	4233

表 17: Gear Controller-1：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	148	698864	4568
Depth	21	108294	4839
Length	21	107906	4803
TotalState : avg	148	685668	4639
TotalState : stp	111	427916	4617
Transition : avg	111	502354	4736
Transition : stp	111	457287	4672

表 18: Gear Controller-2：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	7	18307	1774
Depth	8	19770	1840
Length	8	18625	1789
TotalState : avg	8	20061	1761
TotalState : stp	8	21294	1827
Transition : avg	8	18389	1677
Transition : stp	8	20245	1774

表 19: Gear Controller-3：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	49	89424	1498
Depth	49	80174	1584
Length	8	17416	1548
TotalState : avg	49	89211	1478
TotalState : stp	49	92323	1496
Transition : avg	49	91985	1490
Transition : stp	49	94534	1511

表 20: Gear Controller-4：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	8	25457	2132
Depth	8	27249	2287
Length	8	27143	2276
TotalState : avg	8	27063	2176
TotalState : stp	8	26907	2274
Transition : avg	8	24553	2129
Transition : stp	8	25798	2228

表 21: Gear Contorller-5：降順

Evaluation	Number of Iterations	Execute Time(ms)	Number of Duplicate States
Normal	9	17786	1032
Depth	7	15039	1050
Length	10	20260	1080
TotalState : avg	10	19065	1012
TotalState : stp	9	18319	1036
Transition : avg	9	18387	1024
Transition : stp	9	18208	1044