2E1-3

強連結成分の性質を用いた OWCTY モデル検査アルゴリズムの高速化

Speedup of OWCTY Model Checking Algorithm using Strongly Connected Components

川端聡基[*1] 小林史佳[*1] 上田和紀[*2]
Toshiki KAWABATA Fumiyoshi KOBAYASHI Kazunori UEDA

[*1]早稲田大学大学院基幹理工学研究科
Graduate School of Fundamental Science and Engineering, Waseda University

[*2]早稲田大学理工学術院
Faculty of Science and Engineering, Waseda University

Model checking is an exhaustive search method of verification. Automata-based LTL model checking is one of the methods to solve accepting cycle search problems. Model checking is prone to state explosion, and we may expect that parallel processing would be a promising approch. However, the optimal sequential algorithm is P-complete and is difficult to parallelize. Alternative parallel algorithms have been proposed, and OWCTY is one of them. OWCTY is known to be a stable and fast algorithm for models without bugs, but it does not use the characteristics of the automata used in LTL model checking. We propose a new algorithm named SCC-OWCTY that exploits the SCCs (strongly connected components) of property automata. The algorithm removes states that are judged not to form accepting cycles faster than OWCTY. We experimented and compared the two algorithms using DiVinE, and confirmed improvements both in performance and scalability.

1. はじめに

モデル検査は網羅的探索を行う自動検証手法であり，その一つに受理サイクル探索問題に帰結させるオートマトンベースの LTL モデル検査がある．モデル検査は対象となる状態空間が組み合わせ爆発を起こしやすく，その対策として並列化手法が考えられている．しかし，LTL モデル検査で逐次最適とされるアルゴリズムは P-完全で並列化は困難である．そのため代わりの並列アルゴリズムが提案されていて，本研究で扱う OWCTY (One-Way-Catch-Them-Young) はその一つである．OWCTY はバグがないモデルに対して安定して高速に動作するアルゴリズムであると分かっている．しかし，検証対象の特性を考慮した最適化がなされているとは言えない．

そこで本研究ではさらなる高速化を図り，探索する状態遷移グラフの強連結成分 (SCC: strongly connected components) の特性を考え，受理サイクルを作らないと判断できる状態を削除するアルゴリズム SCC-OWCTY を提案する．分散検証環境 DiVinE[1] に SCC-OWCTY を実装し，評価を行った．

2. モデル検査

モデル検査は形式的検証手法の一つであり，対象システムを有限状態遷移グラフとして表現し，そのグラフを網羅的に探索することにより，システムに要求される性質が満たされるかどうかを判定する．モデル検査は網羅的探索を行うためタイミングのバグや非常に稀にしか起こらないバグなどの動作テスト等では発見することが難しいバグを発見することが可能であり，システムの信頼性を高める技術として注目を集めている．

モデル検査の一つに SPIN[3] などでも使用されているオートマトンベースの LTL モデル検査がある．これは要求される性質を線形時間時相論理 LTL で記述し，Büchi オートマトンを用いて受理サイクル探索問題に帰着させる手法である．システムの全実行をシステムオートマトン，検証する LTL 式の否定を性質オートマトンに変換し，これらの同期積をとった積オートマトンに受理状態を含む閉路 (受理サイクル) があるかどうか検査する．受理サイクルがあれば，そのサイクルを通る実行は不正な実行である．

モデル検査が抱える重要な問題に状態爆発問題がある．これはシステムの変数の数やプロセスの数が増加すると，生成される状態数が爆発的に増えてしまうという問題であり，これによりモデル検査問題を解くために時間やメモリを大量に消費してしまう．この問題をいかに回避するかがモデル検査における重要な研究テーマとなっている．

連絡先: 川端 聡基，早稲田大学大学院基幹理工学研究科情報理工学専攻，〒 169-8555 新宿区大久保 3-4-1，03-5286-3340，kawabata(at)ueda.info.waseda.ac.jp

3. 並列 LTL モデル検査

状態爆発問題に対するアプローチの一つに，並列化手法がある．これは多くの計算資源を用いることにより，より大規模な問題を解けるようにすると共に，高速化を目指すものである．

LTL モデル検査における最適な探索アルゴリズムは post-order の DFS に基づいているが，これは P-完全であるとされていて [4] ，並列化は困難と考えられる．そのため，並列環境のための代わりのアルゴリズムがいくつか提案されている [2].

先行研究 [5] により受理サイクルがある問題では MAP (Maximal Accepting Predecessors) アルゴリズムが，受理サイクルがない問題では OWCTY アルゴリズムが高速に動作すると分かっている．MAP は受理サイクルを作る受理状態は自分自身に到達可能であるという考えに基づき，状態遷移時にその状態へ遷移可能な受理状態の情報を伝達させるアルゴリズムである．OWCTY については次節で詳しく説明する．

3.1 DiVinE

DiVinE (DIstributed VerifIcatioN Environment)[1] は並列モデル検査を行うための検証システムであり，多数の並列アルゴリズムが実装されている．また，並列モデル検査用のライブラリでもあり，状態空間の生成や状態保存，通信関数などを提供している．そのため，開発者は探索アルゴリズムだけ実装すればすぐに評価を行うことができ，また既存の各アルゴリズムとの比較も容易となっている．

最新版としては DiVinE-2.2 が 2010 年 2 月に公開されたが，

```
OWCTY_reversed(s)
   V = {}
   REPEAT
      V := build_and_eliminate(V,s)
      V := Reachability_Test(V)
   UNTIL all_reachable(V)
   IF V = {} THEN
      Display "NO ACCEPTING CYCLE exists"
   ELSE
      Display "ACCEPTING CYCLE found"
   ENDIF
STOP
```

図 1: OWCTY_reversed algorithm

今回は昨年公開された DiVinE-Cluster-0.8.3 を使用する．

3.2 並列化方式

DiVinE では partition function を用いて状態空間を分割し，各状態に担当する計算ノードを割り当てている．新しい状態を生成した際，partition function に従ってその状態を担当する計算ノードを決定する．以後その状態に関する情報は担当ノードのみが保持し，計算は全てそのノードが行うようにする．

DiVinE ではデフォルトの状態分割法として Hash-Based 分割を採用している．この手法は，ビット列で表現された各状態のハッシュ値に基づき担当する PE を決める．利点として PE ごとの担当する状態数を均等にできるが，欠点として異なる PE 間を跨ぐ遷移の数を抑えることが困難となり，通信が頻繁に発生するようになる．

関連研究として，状態分割法に性質オートマトンの SCC の特性を利用した Hash On SCC 分割 [6] があり，最大約 30%の実行時間の削減がなされている．しかし，今回はアルゴリズムとの相性を考え Hash-Based 分割を使用する．

4. 既存アルゴリズム

並列 LTL モデル検査アルゴリズムのひとつとして OWCTY がある．OWCTY はグラフの構築と，受理サイクルを作らないと判断できる状態の削除を繰り返す手法である．

OWCTY_reversed は OWCTY の改良版であり，グラフ構築 (フェーズ 1) と，受理状態への到達性検査 (フェーズ 2) との反復で構成されている．状態削除はフェーズ 1 のときに行われ，その条件は以下のようになる．

条件 1 その状態からの遷移が存在しない

条件 2 その状態がどの受理状態へも到達不可能である

OWCTY_reversed の基本的な構成は図 1 のようになる．s,V はそれぞれグラフの初期状態と訪れた状態の集合を表し，遷移関係と受理状態（それぞれ固定）は implicit に与えられているとする．build_and_eliminate 関数がフェーズ 1 のグラフ構築と条件 1 を満たす状態の削除を行う．その後，Reachability_Test 関数がフェーズ 2 を行い，条件 2 に当てはまる状態にフラグを立てる．フラグが立っている状態は次の反復のフェーズ 1 の時に削除される．終了判定には all_reachable 関数を使用し，残っているすべての状態にフラグが立っている場合，反復が起きず終了する．終了したときに削除されていない状態があるならば受理サイクルがあると判断し，すべての状態が削除されたらならば受理サイクルがないと判断する．このすべての処理はノードごとに独立して行えるので，前節で述べた partition function を用いて並列に処理が行われる．

OWCTY は on-the-fly で動かない．つまり，全体のオートマトングラフをまず構築しなければならない．そのため一般に受理サイクルがある問題に対しては不得意であり，そうでない問題を得意とする．

```
SCC-OWCTY(s)
   V = {}
   scc_analyze()
   scc_init := s
   REPEAT
      (V,scc_init) := build_and_scc_eliminate(V,scc_init)
      (V,scc_init) := Reachability_Test(V)
   UNTIL all_reachable(V)
   IF V = {} THEN
      Display "NO ACCEPTING CYCLE exists"
   ELSE
      Display "ACCEPTING CYCLE found"
   ENDIF
STOP
```

図 2: SCC-OWCTY algorithm

5. SCC-OWCTY

このアルゴリズムは OWCTY_reversed を改良したもので，性質オートマトンの強連結成分 (strongly connected component (SCC)) の特性を利用し新たな状態削除条件を追加したものである．

SCC とは次のように定義される．状態 u，v があったとき u から v へ到達可能であり，また v から u へ到達可能なとき u，v は同じ SCC に属すると言う．

5.1 アルゴリズム

SCC-OWCTY は以下の考えに基づく．探索する積オートマトンがシステムオートマトンと性質オートマトンの同期積であるので，積オートマトンの状態は性質オートマトンの SCC の特性を引き継ぐ．つまり，積オートマトンの二つの状態が性質オートマトンの異なる SCC から生成されたならば，その二つが同じ閉路に含まれることはない．従って，受理状態を含まない SCC より生成された状態が受理サイクルを作ることはない．よって，SCC-OWCTY は OWCTY_reversed の状態削除条件に以下のものを加える．

条件 3 受理状態を含まない SCC の状態より生成された

SCC-OWCTY の基本的な構成は図 2 のようになる．OWCTY_reversed からの変更点は以下の 3 つである．一つ目は反復が始まる前に scc_analyze 関数で性質オートマトンの SCC の解析を行う点．二つ目はフェーズ 1 の関数が build_and_scc_eliminate になっている点で，これは build_and_eliminate 関数に新たに状態削除条件 3 を加えただけのものである．三つ目はフェーズ 1 を開始する状態が変わった点である．OWCTY_reversed では初期状態が探索の途中で消えることはないが，SCC-OWCTY では多くの場合消える．そのため，SCC-OWCTY では異なる SCC 間の遷移が起こった場合，その遷移先の状態すべてを scc_init に保存しておき，反復が起こったときはそこからフェーズ 1 を開始する．

5.2 アルゴリズムの性質

このアルゴリズムは以下の三つの削減効果によって既存のアルゴリズムより高速に動作すると期待できる．

- フェーズ 1 での状態削除時の処理の削減
- フェーズ 2 での処理状態数の削減
- 多くの状態を削除することによる反復回数の削減

表 1: 実験環境

OS	CentOS 5.3
CPU	Intel Xeon(R) 2.93GHz, Quad Core × 4
メモリ容量	128GB
gcc version	4.1.2
mpich version	mpich2 1.0.8p1
divine-cluster version	0.8.3

一つ目の削減効果は，条件 1,2 が適用され状態が削除されるより，条件 3 が適用され状態が削除される方が処理量が少ないためである．二つ目は条件 3 によってフェーズ 1 で多くの状態を削除できた場合に起こる．そして，これが三つの中で一番効果が大きいと考えられる．

一方，このアルゴリズムは既存アルゴリズムに比べ，以下の二つの計算を余計に行っている．

- 性質オートマトンの SCC の解析 (scc_analyze 関数)
- 各状態の SCC の参照 (build_and_scc_eliminate 関数)

一番目に関しては，今回扱うモデルは受理サイクル探索を行う積オートマトンの状態数が数千万から数億であるのに対し，性質オートマトンの状態数は十以下であるので無視できる．二番目に関してもこれが大きなオーバヘッドになるとは考えられないが，条件 3 で削除できる状態がまったくない場合，このオーバヘッドが原因で実行速度が低下する可能性は考えられる．

6. 評価実験

6.1 実験概要

OWCTY_reversed と SCC-OWCTY を 1，2，4，8，16PE で実行し比較した．本実験の環境を表 1 に示す．共有メモリを用いた並列マシンを使用し，MPI を用いて通信を行っている．

使用したモデルは BEEM (BEnchmarks for Explicit Model checkers)[7] で提供されているモデル 457 個中，OWCTY_reversed を 1PE で動かしたときに 100 秒以上かかり，かつすべての場合で 1 回でも制限時間以内に解けたモデル 99 個である．制限時間は 7200 秒とし，それ以内に解けなかったモデルは速度向上比の計算では 7200 秒で解けたとみなす．各モデルについて 3 回ずつ測定し一番良い結果を使用する．

6.2 同一 PE 数での速度向上比

OWCTY_reversed に対する SCC-OWCTY の各 PE 数での平均速度向上比を表 2 に示す．なお，これ以降平均とは幾何平均のことを指す．この表によると，SCC-OWCTY は 1PE から 16PE までどの場合でも実行速度が改善されており，使用 PE 数を多くしていけばいくほど OWCTY_reversed より高い性能で動作することが分かる．

最も速度向上比が低いのは 16PE で動かしたときに 0.93 倍である場合だった．そのモデルは性質オートマトンに受理状態を含まない SCC がなく，条件 3 に当てはまる状態がなかった．今回扱った 99 モデルのうち，同じように条件 3 に当てはまる状態を持たないモデルは他に 4 つあった．性質オートマトンの解析はアルゴリズム実行前に行うので，SCC-OWCTY の有効性を考慮し，使用を判断することは可能である．

6.3 フェーズ 2 にかかった時間の割合と速度向上比

OWCTY_reversed で 1 回目の反復にフェーズ 2 にかかった時間の割合と各モデルでの平均速度向上比の関係を図 3 にまとめた．この図には一つプロットしていないモデルがあるが，それについては後で詳しく述べる．この図によると，フェーズ 2 にかかっている時間が長いモデルで高い速度向上比が得られているということが分かる．これはフェーズ 1 において多くの状態を削除できたため，フェーズ 2 で処理する状態数が少なくなり，起こったことだと考えられる．

表 2: 各 PE 数ごとの平均速度向上比

PE 数	1	2	4	8	16
平均速度向上比	1.34	2.09	2.23	2.56	2.94

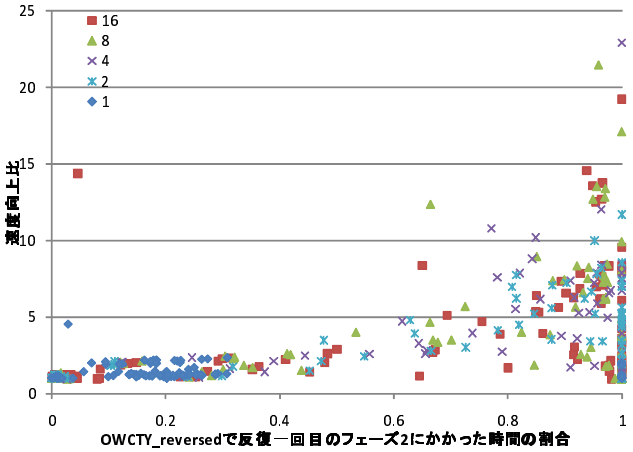

図 3: フェーズ 2 にかかった時間の割合と速度向上比の関係

1PE の場合はフェーズ 2 にかかっている割合が高々30%程度であった．そのため，他の PE の場合に比べ平均速度向上比が低くなったのだと考えられる．

この図で除いたモデルは，16PE で比べたときに速度向上比が 52.3 倍であった．このモデルは SCC-OWCTY では 130 秒程度で解けているのに対し，OWCTY_reversed で実行したときはフェーズ 1 で制限時間が超えていた．また，図 3 にはフェーズ 2 にかかった時間の割合が 10%以下であるにもかかわらず，1PE，16PE において高い速度向上が得られているケースがある．これは同一のモデルでの結果であるが，2，4，8PE ではそのような結果は得られなかった．

6.4 モデルごとの並列効果

各モデルごとの OWCTY_reversed，SCC-OWCTY の逐次実行時と並列実行時の実行時間の関係を図 4，5 にまとめた．グラフの右下にいくほど高い並列効果が得られているということになる．この図を見比べると図 5 の方が右下に配置されている点が多い．しかし，図 4 においてもいくつかの点は SCC-OWCTY と同じぐらいの並列効果が得られている．

図 4，5 において上に横並びしている点は逐次実行したときには解けたモデルが並列実行した場合解けなくなっていることを意味する．同じように右端のデータは逐次実行では解けなかったモデルが並列では解けるようになったことを意味する．図 4，5 を見比べると OWCTY_reversed に比べ SCC-OWCTY は並列実行で解ける問題が増えていることが分かる．

6.5 平均並列効果

両アルゴリズムで逐次実行時と並列実行時の平均の並列効果を表 3 にまとめた．この表によるとどちらのアルゴリズムでも PE 数が少ない場合では逐次より性能が落ちるが，8PE 以上では実行速度が向上しているのが分かる．そして、SCC-

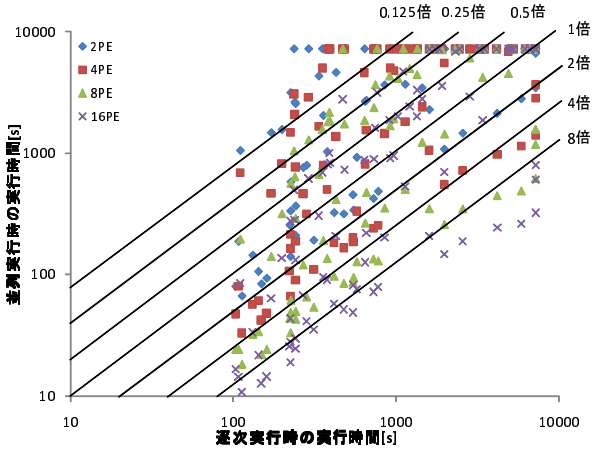

図 4: OWCTY_reversed における逐次と並列の実行時間の関係

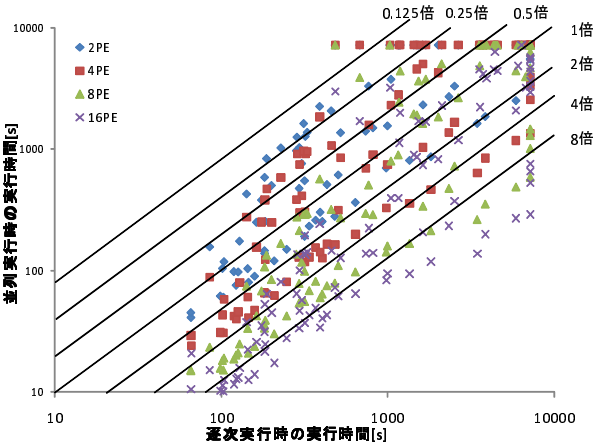

図 5: SCC-OWCTY における逐次と並列の実行時間の関係

OWCTY の方が少ない PE 数でも高い実行速度の向上が得られることが分かる．また，使用 PE 数は 16 よりも多くしても，さらなる性能向上が見られると期待できる．

少ない PE 数で並列実行したときに逐次実行時より性能が落ちるのは，並列実行時にフェーズ 2 での性能が大きく低下する場合があるのが原因と考えられる．少ない PE 数の場合フェーズ 2 の性能低下が並列効果を上回り，使用 PE 数を増やしていくとそれが逆転するためにこの結果が出たのだと考えられる．

このことを確かめるために反復 1 回目のフェーズ 1 にかかった時間のみの平均の並列効果を表 4 にまとめた．これを見ると両アルゴリズム共にフェーズ 1 においては高い並列効果が得られていることが分かる．これによってフェーズ 2 が並列実行時のボトルネックとなっていることが分かった．

図 3 によると逐次実行時のフェーズ 2 にかかる時間の割合は高々30%程度なので，フェーズ 1 だけ並列実行してフェーズ 2 を逐次実行するようにするとさらなる高速化が図れる．並列実行時にフェーズ 2 の性能が低下した理由に関してはさらなる調査が必要である．

7. まとめと今後の課題

強連結成分の性質を用いた受理サイクル探索アルゴリズムとして SCC-OWCTY を設計，実装を行った．DiVinE-Cluster に実装されている OWCTY_reversed と比較したところ，同一 PE 数での実行速度が改善されており，平均で 2.16 倍の速度向上が得られた．並列実行時には，多くのモデルにおいて受理状態への到達性検査に時間がかかっている割合が高く，それを改善することによって実行速度が向上していた．また，いくつかのモデルにおいてはグラフ構築の時間で大幅な改善が見られ，最高で 52.3 倍の速度向上が得られた．グラフ構築時間の改善がなぜ起こったのかさらなる調査が必要である．

表 3: 各 PE 数での平均の並列効果

PE 数	2	4	8	16
OWCTY_reversed	0.41	0.61	1.13	1.73
SCC-OWCTY	0.64	1.01	2.17	3.79

表 4: 各 PE 数でのフェーズ 1 の平均の並列効果

PE 数	2	4	8	16
OWCTY_reversed	1.62	3.15	6.24	7.55
SCC-OWCTY	1.69	3.35	6.87	10.72

次に，OWCTY_reversed，SCC-OWCTY の並列効果を調べたところ，SCC-OWCTY の方が安定して高速に動作することが分かった．どちらも少ない PE 数では並列に実行すると性能が低下するが，4PE を超えたぐらいから性能が上がっていき，16PE のときには逐次の 3.79 倍の性能となった．性能低下は受理状態への到達性検査時に起こってると分かったので，今後その部分の改善が必要である．

また，今回の実装では扱うモデルの SCC の形については考慮していないため，全ての SCC で受理状態を含むような，SCC-OWCTY が有効でない場合でも同じように実行してしまう．性質オートマトンの SCC 解析の結果を反映し，探索開始時に有効なアルゴリズムを選択できるようにすると，最悪の場合を回避して高速に実行できるようになると考えられる．

参考文献

[1] J. Barnat, L. Brim, I. Cerna, P. Moravec, P. Rockai, and P. Simecek: DiVinE- - A Tool for Distributed Verification, in Proc. CAV 2006, LNCS 4144, Springer, pp.278-281 (2006).

[2] J. Barnat, L. Brim, and I. Cerna: Cluster-Based LTL Model Checking of Large Systems, in Formal Methods for Components and Objects, LNCS 4111, Springer, pp.259-279 (2006).

[3] G. Holzmann: The Spin Model Checker, Primer and Reference Manual, Addison-Wesley (2003).

[4] J. Reif: Depth-First Search is Inherently Sequential, Information Processing Letters, Vol.20, No.5, pp.229-234 (1985).

[5] 小林史佳　上田和紀: 巨大なモデルに対する並列モデル検査手法, 日本ソフトウェア科学会第 26 回大会, 7A-2 (2009).

[6] 三輪 真弘, 上田 和紀: 強連結成分ベースのグラフ分割による分散並列 LTL モデル検査の高速化, 情報処理学会第 71 回全国大会, 分冊 1, pp.25-26 (2009).

[7] R. Pelanek: Web Portal for Benchmarking Explicit Model Checkers, Tech. Report FIMU-RS-2006-03, Masaryk University, Czech Republic (2006).