LMNtal 並列モデル検査における状態生成数削減及び高速化

Reduction of the Number of States and the Acceleration of LMNtal Parallel Model Checking

安田 竜
Ryo YASUDA

早稲田大学大学院基幹理工学研究科情報理工学専攻
Graduate School of Fundamental Science and Engineering, Waseda University
yasuda@ueda.info.waseda.ac.jp

吉田 健人
Taketo YOSHIDA

(同上)
taketo@ueda.info.waseda.ac.jp

上田 和紀
Kazunori UEDA

早稲田大学理工学術院
Faculty of Science and Engineering, Waseda University
ueda@ueda.info.waseda.ac.jp

keywords: model checking, accepting cycle search, nested depth first search, parallelization

Summary

SLIM is an LMNtal runtime. LMNtal is a programming and modeling language based on hierarchical graph rewriting. SLIM features automata-based LTL model checking that is one of the methods to solve accepting cycle search problems. Parallel search algorithms OWCTY and MAP used by SLIM generate a large number of states for problems having and accepting cycles. Moreover, they have a problem that performance seriously falls for particular problems. We propose a new algorithm that combines MAP and Nested DFS to remove states for problems including accepting cycles. We experimented the algorithm and confirmed improvements both in performance and scalability.

1. はじめに

モデル検査とは、対象となるシステムを状態遷移系としてモデル化し、網羅的に探索することによって、システムに要求された性質が満たされているかどうかを判定する検証技術である。網羅的かつ機械的な検証を行うことができるため、並列システムにおけるデッドロックなど動作テスト等では発見が難しいエラーを確実に見つけることが可能であり、システムの信頼性を高める方法として非常に有効である。しかし、モデルの規模が大きくなると爆発的に状態数が増加し、膨大な処理時間がかかるという問題がある。

この問題を解決するため、階層グラフ書換え言語 LMNtal [乾 08] をモデリング言語とする SLIM モデル検査器 [後町 11] では、並列化等による高速化が行われてきた。第 2・3 節で述べるように、LTL モデル検査で逐次最適な探索アルゴリズムとされている Nested DFS は P-完全であり並列化は困難であるため、SLIM モデル検査器には、並列探索アルゴリズムである OWCTY や MAP が実装されている。しかし、OWCTY, MAP はエラーを含む問題に対して状態生成数が多くなり易い。調査の結果、

階層グラフという強力なデータ構造で状態を扱う SLIM モデル検査器は、状態の対称性を用いて状態数を削減できるのが大きな特徴であるが、状態展開にグラフ同型性判定を含む高コストな計算を含むため状態空間の構築が処理時間のほとんどを占めており、状態数が増大すると処理時間に大きな影響を与えることが分かった。また、OWCTY, MAP は特定の問題に対して極端に性能が落ちてしまうという問題点がある。

このような背景から、本研究では Nested DFS と MAP を組合わせた新たな並列アルゴリズムの設計と実装を SLIM 上で行い、その性能を評価した。その結果、状態展開に高コストな処理を含むモデル検査器において、エラーを含む多くの問題で状態生成数が減少し、高速かつ安定した性能が出ることが確認できた。

2. モデル検査

2.1 オートマトンを用いたモデル検査

オートマトンベースのモデル検査とは、要求された性質を LTL (Linear Temporal Logic, 線形時相論理) で記述し、オートマトンを用いて受理サイクル (受理頂点を含む閉路) 探索問題に帰着させる手法である。

検証対象となるシステムを A 、性質を S とする。 A が取り得るすべての実行（状態遷移列）の集合を $L(A)$ 、 S によって許されるすべての実行の集合を $L(S)$ とした場合、 $L(A)$ 、 $L(S)$ が以下の関係にあるとき、システム A は性質 S を満たす。

$$L(A) \subseteq L(S) \quad (1)$$

ここで、(1) は $L(S)$ の補集合 $\overline{L(S)}$ を用いて次のように書くことができる。

$$L(A) \cap \overline{L(S)} = \emptyset \quad (2)$$

これは、性質 S に反した A の実行が存在しないことを表しており、もし仮に $L(A) \cap \overline{L(S)}$ が空でないならば、それは不正な実行（エラー）となる。したがって、 $L(A) \cap \overline{L(S)}$ をオートマトン（同期積オートマトン）として表現し、このオートマトンによって受理される実行が空かどうかを判定することで、システム A が性質 S を満たすかどうかを検査することができる [Clarke 99]。同期積オートマトンを作成することを状態空間の構築、受理サイクルの探索を行うことを状態空間の探索という。

モデル検査には、仕様を満たさない状態の有無の検査 (safety) と、仕様を満たさない状態遷移列（受理サイクル）の有無の検査 (liveness) とがある。

2.2 On-the-fly level

状態空間探索アルゴリズムの重要な性質の一つとして、On-the-fly level [Barnat 12] がある。状態空間探索における On-the-fly とは、状態空間構築を行いながら探索を行うことで、探索中に受理サイクルを発見した場合は探索および構築を打ち切り、全状態空間が構築される前に状態空間探索と構築を終了することができる。On-the-fly の性能は以下の3段階に分類される。

- level 0: On-the-fly 動作をしない。受理サイクルが存在する場合でも全状態空間を構築する。
- level 1: On-the-fly 動作をする場合がある。受理サイクルが存在する場合、全状態空間を構築する前に探索が終了することがある。
- level 2: On-the-fly 動作をする。受理サイクルが存在する場合、全状態空間を構築する前に探索が終了する。

モデル検査では、システム変数等の増加により生成される状態数の組み合わせが爆発しがちであるので、全状態空間を構築する前に探索を打ち切るとは実行時間やメモリ使用量に関して非常に重要であると言える。

2.3 モデル検査アルゴリズム

モデル検査に用いられる並列状態空間構築アルゴリズムとしては、SPIN モデル検査器 [Holzmann 87] の Version 5 以降で用いられている Stack-Slicing アルゴリズム [Holzmann 08] 等がある。また、状態空間探索アルゴリ

ズムとしては SPIN モデル検査器で用いられている Nested DFS、分散検証環境 DiVinE [Barnat 06] で用いられている OWCTY、MAP、及び OWCTY と MAP を組み合わせた OWCTY(+MAP) アルゴリズムなどがある。以下で各アルゴリズムの概要について述べる。

Stack-Slicing アルゴリズムは、全スレッド (Worker) で状態空間を共有し、深さ優先探索 (DFS) をパイプライン分割する並列アルゴリズムである。全スレッドは、状態を受け取る Work Queue を固有に持ち、通信方向を持った論理的な輪を構成する。DFS のスタックの深さが一定の閾値を超えた場合、隣接スレッドの Work Queue へ未展開の新規状態を送信する。各スレッドは、Work Queue から取り出した状態を根にした DFS を繰り返す。しかし、1 状態あたりの遷移数が少ない場合や、合流する遷移数、遷移先を持たない状態が多い場合に、実行に参加するスレッド数によっては全てのスレッドに処理すべき状態がうまく行き渡らないという問題がある。

Nested DFS (NDFS) [Courcoubetis 90] は、On-the-fly level 2 の逐次探索アルゴリズムで、LTL モデル検査において逐次最適な探索アルゴリズムである。Nested DFS は、二段階の DFS (Blue DFS, Red DFS) により構成される。Blue DFS が状態空間の構築を行い、Red DFS が状態空間の探索を行う。Red DFS は、ある受理状態から自分自身へ到達できるかを DFS により判定する。Blue DFS は、DFS による状態展開を行いながら Red DFS の起点となる受理状態を探索する。Red DFS の起点となる受理頂点は postorder 順に選択される。これにより、同一の枝に対するトラバースが発生しなくなり、計算量の逐次最適性が保証される。しかし、受理頂点を postorder 順に選択しなければならないという制約があるため、状態の展開や探索を複数のスレッドで分担することができず、並列化が困難である。SPIN モデル検査器では、Red DFS と Blue DFS を並列に行わせる Dual DFS が並列アルゴリズムとして用いられているが、liveness の検証において使用できる最大スレッド数は 2 である。

MAP (Maximal Accepting Predecessors) [Brim 04] は、On-the-fly level 1 の並列探索アルゴリズムである。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遷移可能な受理状態の情報を伝達させていく。

OWCTY (One Way Catch Then Young) [Černá 03] は、On-the-fly level 0 の並列探索アルゴリズムである。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 は、On-the-fly level が 0 であるため、全状態空間を構築してから探索を行う必要があるが、一部 MAP の手法を導入することにより On-the-fly level 1 で動作するようになる。この場合、受理サイクル発見のタイミングは MAP に依存しているため、状態生成数は MAP と同等になる。

3. SLIM におけるモデル検査アルゴリズム

SLIM モデル検査器では、状態空間構築アルゴリズムとして動的負荷分散を行うように拡張した Stack-Slicing アルゴリズムを、状態空間探索アルゴリズムとして Nested DFS, MAP, OWCTY(+MAP) を用いており、状態空間構築・探索処理の並列化がなされている [後町 11]。状態空間探索に MAP を用いた場合の処理を図 1 に示す。SLIM における Stack-Slicing アルゴリズムは、第 2.3 節で述べた問題点を解決するため、隣接スレッドがアイドル状態を主張した場合に、閾値を待たずに状態の送信を行う処理と、アイドル状態になったスレッドが、他のスレッドの Work Queue から状態を取得する処理を追加することで負荷分散を行っている。StackSlicing は、スレッドが自身の Work Queue (*myWorkQueue*) から取得 (*dequeue(myWorkQueue)*) した状態から DFS による状態展開処理 (*dfsParallel*) を繰り返し行う手続きである。dfsParallel は状態空間構築を行う手続きで、DFS スタックから取得 (*stackPop*) した状態 *s* に対して遷移先の計算 (*expand*) を行う。その後、MAP の伝搬や受理サイクルの探索 (*mapStart*) を行う。求めた *s* の遷移先状態の中から新規状態をスレッド自身の DFS スタックへ追加 (*stackPush(succ)*) するか隣接するスレッドへ送信 (*handoff(succ)*) するかを選択して処理を進める。選択の条件式である *loadBalancing* は、DFS スタックの深さが閾値 (Cutoff Depth) を超えている場合と、隣接するスレッドがアイドル状態を主張している場合に真を返す。

StackSlicing において、スレッドは自身の Work Queue が空である (*empty(myWorkQueue)*) 場合に、他のスレッドの Work Queue から状態の取得を試みる (*workStealing*)。手続き *terminationDetection* は終了検知を行うもので、全てのスレッドの Work Queue が空であり、アイドル状態を主張している場合、実行を終了する。

これらの処理において、状態空間構築処理では高い並列効果が出ているが、状態空間探索処理においては On-the-fly level が 0~1 の並列アルゴリズムしか実装されていないほか、性能の評価も十分に行われていなかった。探索アルゴリズムの性能評価を行った結果、SLIM のように状態展開に高コストな処理を含んでいる場合、これらの探索アルゴリズムを用いた場合の受理サイクルを含む問題に対しての状態生成数増大は影響が大きく、十分な性能が出ていないことが分かった。

4. 新たな並列モデル検査アルゴリズム

4.1 概要

On-the-fly level が 0~1 の並列モデル検査アルゴリズムである MAP や OWCTY では、状態生成数が On-the-fly level 2 の Nested DFS に比べて多く、状態の展開に高コストな計算を含む場合に十分な性能がでない。そこで、Nested DFS と MAP を組み合わせた On-the-fly level が

```

procedure StackSlicing
  while  $\neg$ terminationDetection() do
    if empty(myWorkQueue) then
      workStealing()
    else
      stackPush(dequeue(myWorkQueue))
      dfsParallel()
    end if
  end while
end procedure

procedure dfsParallel
  STATE s := stackPop()
  expand(s)
  mapStart(s)
  foreach succ in newSuccessors(s) do
    if loadBalancing() then
      handoff(succ)
    else
      stackPush(succ)
      dfsParallel()
    end if
  end for
end procedure

```

図 1 SLIM の並列モデル検査アルゴリズム

2 の並列モデル検査アルゴリズムを作成した。

このアルゴリズムは MAP ではなく OWCTY(+MAP) を用いても実現できる。しかし、OWCTY(+MAP) の状態生成数は第 2.3 節で述べた通り MAP の状態生成数と等しい。状態展開に比べて状態空間探索時間が無視できるほど小さい場合、状態数が等しければ全体の実行時間も MAP とほぼ同じとなる。よって、今回は OWCTY ではなく実装上扱い易い MAP を採用した。また、状態空間構築アルゴリズムは第 3 章で述べた動的負荷分散機能付きの Stack-Slicing アルゴリズムを用いている。

On-the-fly level 2 の並列アルゴリズムとしては、複数スレッドで Nested DFS を同時に動かすことによって並列化を行う Multicore Nested DFS [Evangelista 12] も存在して、状態空間構築が低コストならば良いスケーラビリティが出ることが報告されている。しかし、各スレッドで Nested DFS を実行する関係上、各スレッドで Nested DFS と同様に postorder 順の探索を行わなければならない、状態の受け渡しを行って状態展開の並列化を行う Stack-Slicing が併用できず、負荷分散も行えなくなる。このため、状態生成数を削減できても、Stack-Slicing を用いた場合よりも状態の展開自体が遅くなってしまう可能性があり、受理サイクルの存在する問題に関しても十分な性能が出ることが保証できない。さらに、受理サイクルが存在しない問題の場合、全状態空間が構築されるため、状態空間構築において効率的な負荷分散が行えない Multicore Nested DFS では十分な性能が出せない可能性がある。よって、今回は独自のアルゴリズムを導入することとした。

4.2 方針

既存の SLIM 並列モデル検査アルゴリズムでは図 1 のように Stack-Slicing による状態空間構築と MAP による状態空間探索を行っていた。既存のアルゴリズムでは *N* スレッドが MAP を実行していたが、図 2 に示すように、1

スレッド (*ndfsWorker*, 図 2 における Thread1) で Nested DFS を実行し, 残りの $N - 1$ スレッド (*mapWorker*, 図 2 における Thread2 ~ N) で MAP を実行するように変更する. こうすることにより, 全 Worker で状態空間を共有し, N スレッドで並列で状態空間の構築を行いながら Nested DFS による早期の受理サイクル検出が可能となる. 受理サイクルを検出するタイミングは Nested DFS と同じになるため, 状態の生成数は Nested DFS と同じ On-the-fly level 2 のアルゴリズムと同等になる. それに加え, 同時に MAP を動作させることにより, MAP の方が Nested DFS よりも受理サイクルの検出が早いような問題や, エラーを含まない問題にも対応できる. ただし, 1 スレッドを Nested DFS に使うことにより, 既存のアルゴリズムと比較して MAP アルゴリズムの探索速度は劣る. しかし, 状態の展開が高コストなモデル検査器の場合, 効率的な探索を行うことよりも状態生成数を削減することによる状態展開処理の削減を行った方が効率的な場合がある. 第 5 章で用いた問題と同様の問題に対して, SLIM モデル検査器を用いて状態空間構築及び状態空間探索の処理時間を計測してみたところ, 状態空間構築に 99% 以上の時間がかかっていた. このため, 状態生成数削減により高速化が期待できるはずである.

次に，新たな並列アルゴリズムを図 3 に示す．新たな並列アルゴリズムでは，*ndfsWorker* が Nested DFS による探索 (*ndfsStart*) と状態空間探索のための postorder 順の計算 (*calcPostorder*) を行い，*mapWorker* は既存のアルゴリズムと同様に MAP による探索を行う．*ndfsStart* は，*newDfsParallel* でバックトラックしてきたとき，つまり状態 *s* が既に展開済みであったとき，かつ状態 *s* が受理状態である場合に実行され，状態 *s* から自分自身への到達性解析を行う．postorder 順の計算は，状態 *s* の successor の中で *ndfsWorker* によって未到達の状態を DFS スタックに積み直すことで行われる．次に，状態空間構築のため，新規 successor を自身または他 Worker の *myWorkQueue* に push するが，*ndfsWorker* は postorder 順の計算を行わなければならないことから，*workStealing* による状態の取得や，*handoff* によって隣接 Worker から渡された状態を処理する処理を行うことができない．そこで，状態の *handoff* は *ndfsWorker* から *mapWorker* への *handoff* または *mapWorker* から *mapWorker* への *handoff* のみ可能にするように処理を変更する (*newHandoff*) ．*workStealing* に関しても，*ndfsWorker* は *workStealing* を行わず，*mapWorker* のみが行うように変更を加える．

5. 評価実験の概要

評価実験は表 1 に示す 48 コアの共有メモリマシンを使用した．評価実験は，受理サイクルを含む問題 60 個と，受理サイクルを含まない問題 24 個を対象に，48 スレッドを用いた場合の全実行時間及び状態生成数を MAP と

図2 新たな並列アルゴリズムの概念図

```

procedure newStackSlicing
  while  $\neg$ terminationDetection() do
    if empty(myWorkQueue) and worker_is_mapWorker then
      workStealing()
    else
      stackPush(dequeue(myWorkQueue))
      newDfsParallel()
    end if
  end while
end procedure

procedure newDfsParallel
  STATE s := stackPeek()
  if is_expanded(s) then
    if worker_is_ndfsWorker and is_accepting_state(s) then
      ndfsStart(s)
    end if
    if is_visited(s) then
      stackPop()
      return
    end if
  end if
  if  $\neg$ is_expanded(s) then expand(s) end if
  if worker_is_ndfsWorker then
    calcPostorder()
  else if worker_is_mapWorker then
    mapStart(s)
  end if
  foreach succ in newSuccessors(s) do
    if loadBalancing() then
      newHandoff(succ)
    else
      stackPush(succ)
      newDfsParallel()
    end if
  end for
end procedure

```

図3 新たな並列モデル検査アルゴリズム

新たな並列アルゴリズムで比較する．Stack-Slicing の閾値は 5 とする．計測時間が 2000[s] を超えるものはタイムアウトとみなし，計測結果は 2000[s] とする．タイムアウトした問題の状態生成数は計測できないため，グラフ中には表記しない．また，受理サイクルを含まない問題の場合，どのアルゴリズムを用いても全状態空間を構築することから，状態生成数は全アルゴリズムで等しくなるので，結果は省略する．

使用した問題は，BEEM Database (<http://anna.fi.muni.cz/models/>) 上で公開されている，SPIN モデル検査器用のベンチマーク問題などを中心に，Promela で記述されたモデルを LMNtal でエンコードして利用している．問題には，食事をする哲学者問題，Bakery Mutex Algorithm 等の排他制御モデル，Sliding Window Proto-

col などの通信プロトコル問題, N-Queen 問題などのパズル問題, その他システムモデルや計算モデル等の問題が含まれている. 問題は, MAP により生成される状態数の上限が 700 万状態程度の問題を主な対象とした. 受理サイクルを含む問題 60 個に関して, 評価実験を行った結果を図 4, 図 5 に示す. 次に, 受理サイクルを含まない問題 24 個に関して評価実験を行った結果を図 6 に示す.

6. 新たな並列アルゴリズムに対する考察

6.1 状態生成数

図 4 は, 新たな並列アルゴリズムと MAP をそれぞれ 48 コアで動作させた場合の状態生成数を表している. 約 60% の問題で状態生成数は 0.5 倍以下に, さらに約 30% の問題で状態生成数が 0.125 倍以下になっている. また, 計測結果のうち, MAP の処理時間が 100ms を超えた 35 題に対して状態空間削減率の幾何平均を求めたところ, 約 14% にまで状態生成数を削減できていることが分かった. 一方で, MAP よりも状態生成数が多くなってしまった問題は 7 つのみであり, いずれの問題に関しても MAP と比較して大きく超過していないことがグラフから分かる. これらの問題は, いずれも Nested DFS による受理サイクルの発見タイミングよりも MAP による受理サイクルの発見のタイミングの方が早い問題であった. この場合, 状態生成数は MAP と同等になるはずだが, 若干超過してしまっている. これは, 逐次実行の場合と違い, 並列実行では同一のアルゴリズムを実行しても状態生成数に若干の差異が生じるためであると考えられる. 状態生成数がどの程度削減できるかは問題に依存するが, 計測結果から多くの問題で削減可能であることが確認できる. 一部の問題 (bubble10) では, MAP で 6970337 状態であったものが新たな並列アルゴリズムでは 5346 状態にまで削減されるなど, 非常に良く効果が出ていることが分かる. これは, MAP アルゴリズムが, 各受理状態の順序付け次第で探索の効率が大きく変わる不安定なアルゴリズムであるが, Nested DFS による早期の受理サイクル検出によって不安定さを補うことのできた例である.

6.2 実行時間

図 5 は, 新たな並列アルゴリズムを用いた場合と MAP アルゴリズムを用いた場合の実行時間を表している. 数

10ms で終了するような比較的小規模の問題では MAP よりも実行時間が長くなっている問題があるが, MAP の処理時間が 100ms を超え, タイムアウトしなかった問題 30 題に関しては 87% 以上の問題で速度向上が見られた. さらに, これらの問題における速度向上比の幾何平均を求めたところ, 約 4.96 倍となっていることが分かった. 状態数が爆発しやすく, 既存のアルゴリズムでは解くのが困難であった問題も高速に解けるようになってきていることが分かる. 受理サイクルを含まない問題に関しては状態生成数が変わらないため, 多くの場合で MAP と同等の性能となっていることが図 6 から確認できる. ただし, MAP や OWCTY は反復探索処理を行うため, 受理状態が一列に繋がっているような問題 (peterson_4_p1 など) では反復探索の回数が極端に増加し, 本来は 1% 未満であった状態空間探索の処理時間が全体の 90% 以上を占めるようになり極端に性能が低下してしまうが [小林 09], 新たな並列アルゴリズムで用いている Nested DFS は反復探索を行わないため, これらの問題も高速に解けている.

7. まとめと今後の課題

7.1 ま と め

Nested DFS と MAP を連携させた新たな並列モデル検査アルゴリズムの設計・実装を行った. 新たな並列モデル検査アルゴリズムと SLIM に実装されている MAP を比較したところ, 受理サイクルを含む多くの場合で状態生成数が削減され, それに伴い高速化が行われた. さらに, MAP では解けなかった一部の大規模問題にも対応できるようになった. 受理サイクルを含まない問題でも MAP と相性の悪い問題に関しては高速化を行うことができた. これにより, 状態展開が高コストなモデル検査器において, 状態空間探索アルゴリズムの On-the-fly level を上昇させることによって状態生成数を削減することにより, 全実行時間を短縮できることが確認できた.

図 4 状態生成数比較 (受理サイクル有り)

表 1 実験環境

CPU	AMD Opteron(tm) Processor 6176SE
CPU 周波数	2.3GHz
コア数	48 (12 × 4 Processors)
Memory	256 GBytes
Cache Size	L1: 128KBytes (コア占有)
	L2: 512KBytes (コア占有)
	L3: 12MBytes (12 コア共有)

図5 実行時間比較（受理サイクル有り）

図6 実行時間比較（受理サイクル無し）

7.2 今後の課題

今回導入したアルゴリズムでは検証条件の性質を考慮していないが、例えば強連結成分（SCC）による分割を利用し、問題の性質を考慮するように改良された Nested DFS [Edelkamp 01] の導入などにより、さらなる状態数削減効果が得られるのではないかと考えられる。また、SLIM モデル検査器のように状態展開が高コストなモデル検査器では状態空間構築処理に多くの処理時間が割かれるため状態空間探索処理の高速化は効果が薄い。よって、さらに高速化していくためには状態空間構築の改善が必要であると考えられる。状態空間構築の高速化を行うには、ある状態を起点にした経路集団から代表経路を決定することで実際に展開する状態数を削減する Partial Order Reduction [Peled 93] の応用などが考えられる。

本研究の一部は、科学研究費（基盤研究(B)23300011）の補助を得て行った。

◇ 参考文献 ◇

- [Barnat 06] Barnat, J., Brim, L., Černá, I., Moravec, P., Ročkal, P. and Šimeček, P.: DiVinE – A Tool for Distributed Verification, in *Proc. CAV 2006*, LNCS 4144, Springer, pp. 278–281 (2006)
- [Barnat 12] Barnat, J., Brim, L. and Ročkal, P.: On-the-fly Parallel

Model Checking Algorithm that is Optimal for Verification of Weak LTL Properties, *Science of Computer Programming*, Vol. 77, No. 12, pp. 1272–1288 (2012)

- [Brim 04] Brim, L., Černá, I., Moravec, P. and Šimša, J.: Accepting Predecessors are Better than Back Edges in Distributed LTL Model-Checking, in *Proc. FMCAD 2004*, LNCS 3312, Springer, pp. 352–366 (2004)
- [Černá 03] Černá, I., Pelánek, R.: Distributed Explicit Fair Cycle Detection, in *Proc. SPIN 2003*, LNCS 2648, Springer, pp. 49–73 (2003)
- [Clarke 99] Clarke, E. M., Grumberg, O. and Peled, D. A.: *Model Checking*, The MIT Press (1999)
- [Courcoubetis 90] Courcoubetis, C., Vardi, M. Y., Wolper, P. and Yannakakis, M.: Memory Efficient Algorithm for the Verification of Temporal Properties, in *Proc. CAV 1990*, LNCS 531, Springer, pp. 233–242 (1990)
- [Edelkamp 01] Edelkamp, S., Lafuente, A. L. and Leue, S.: Directed Explicit Model Checking with HSF-SPIN, in *Proc. SPIN 2001*, LNCS 2057, Springer, pp. 57–79 (2001)
- [Evangelista 12] Evangelista, S., Laarman, A., Petrucci, L. and van de Pol, J.: Improved Multi-Core Nested Depth-First Search, in *Proc. ATVA 2012*, LNCS 7561, Springer, pp. 269–283 (2012)
- [後町 11] 後町将人, 堀泰祐, 上田和紀: LMNtal 実行時処理系の並列モデル検査器への発展, *コンピュータソフトウェア*, Vol. 28, No. 4, pp. 137–157 (2011)
- [Holzmann 87] Holzmann, G. J.: The Model Checker SPIN, *IEEE Trans. on Software Engineering*, Vol. 23, No. 5, pp. 279–295 (1997)
- [Holzmann 08] Holzmann, G.: A Stack-Slicing Algorithm for Multi-Core Model Checking, *Electronic Notes in Theoretical Computer Science*, Vol. 198, No. 1, pp. 3–16 (2008)
- [乾 08] 乾敦行, 工藤晋太郎, 原耕司, 水野謙, 加藤紀夫, 上田和紀: 階層グラフ書換えモデルに基づく統合プログラミング言語 LMNtal, *コンピュータソフトウェア*, Vol. 25, No. 1, pp. 124–150 (2008)
- [小林 09] 小林史佳: LMNtal 実行時処理系 SLIM の LTL モデル検査機能の並列化, 早稲田大学大学院理工学研究科修士論文 (2009)
- [Peled 93] Peled, D.: All from One, One for All: on Model Checking Using Representatives, in *Proc. CAV 1993*, LNCS 697, Springer, pp. 409–423 (1993)

〔担当委員：大和田 勇人〕

2013 年 8 月 20 日 受理

著者紹介

安田 竜

2013 年 3 月早稲田大学基幹理工学部情報理工学科卒業。現在早稲田大学大学院基幹理工学研究科情報理工学専攻修士課程在学中。並列計算、モデル検査などの研究に従事。

吉田 健人

2013 年 3 月早稲田大学基幹理工学部情報理工学科卒業。現在早稲田大学大学院基幹理工学研究科情報理工学専攻修士課程在学中。Lock-Free アルゴリズムなど、並列計算の研究に従事。

上田 和紀 (正会員)

1978 年東京大学工学部計数工学科卒業。1986 年同大学院情報工学博士課程修了、工学博士。1983 年 NEC 入社。1985 年～1992 年（財）新世代コンピュータ技術開発機構出向。1993 年より早稲田大学理工学部情報学科。現在、同理工学術院情報理工学科教授。プログラミング言語、並行・並列計算、高性能検証などの研究およびシステム開発に従事。第 7 回日本 IBM 科学賞など受賞。2004～2007 年度日本ソフトウェア科学会編集委員長など歴任。