社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－SLDM－127 (17)
2006／11／29

# 〔特別講演〕SATアルゴリズムとその形式的検証への応用

藤田 昌宏†

† 東京大学大規模集積システム設計教育研究センター 〒113-0032 東京都文京区弥生 2-11-16
E-mail: †fujita@ee.t.u-tokyo.ac.jp

**あらまし** まず、論理式の充足可能性判定問題（SAT 問題）を解くアルゴリズム（SAT 手法と呼ぶ）について、DPLL 法を中心にその歴史と現状について解説する。次に、SAT 手法を特に形式的な設計検証に適用する技術について、現状を解説する。近年の SAT 手法の大きな進歩により、実用規模のハードウェア設計検証や、ソフトウェアのバグハンティングが可能になってきている。本稿はその状況を平易に解説することを目標としている。
**キーワード** SAT、SAT アルゴリズム、BCP、形式的検証、等価性検証、モデルチェッキング

# [Invited Talk] SAT algorithms and their application to formal verification

Masahiro FUJITA†

† VLSI Design and Education Center, The University of Tokyo
2-11-16 Yayoi, Bunkyo-ku, Tokyo, 113-0032 Japan
E-mail: †fujita@ee.t.u-tokyo.ac.jp

**Abstract** In this paper, algorithms for Boolean satisfiability checking problems are reviewed concentrating on the algorithms relating to DPLL method. Then their applications to formal verification of hardware and software are presented. Owing to the recent great improvements on SAT algorithms, realistic sizes of hardware designs and software programs can be formally verified or analyzed with the SAT techniques. This paper aims to concisely give the state-of-the-art SAT algorithms and the current status of their application to formal verification.
**Key words** SAT, SAT algorithm,BCP, formal verification, equivalence checking, model checking

## 1. はじめに

充足可能性判定問題 (boolean satisfiability problem; 通称 SAT) とは，与えられた論理関数に対して，その値を 1 にする (論理関数を充足する) ような入力変数の割り当てが存在するかどうかを判定する問題である．

SAT を解くツールを SAT ソルバといい，本論文では、この SAT ソルバで利用されているアルゴリズムとして、最も代表的な DPLL（あるいは DLL）アルゴリズムを元に、近年のめざましい進歩を中心に解説する。最近、米国プリンストン大 Lintao Zhang らによる chaff[?] と呼ばれる SAT ソルバが開発され、ソースが公開されていることから、広く、改良・利用されるようになり、扱える論理式の規模が飛躍的に拡大した。結果として、形式的検証などに応用した場合に、実用規模のハードウェア設計や、プログラムが扱えるようになってきている。また、SAT ソルバの技術を競うワークショップも毎年開催され、ベンチマーク結果では、毎年目を見張る性能改善が連続して実現している。数百万変数からなる論理式の充足可能性も数時間で判定できるまでになっている。図 1 に SAT ソルバの現在までの性能向上の様子を示す。図の横軸は、年を表し、また縦軸は典型的な例題の場合に、いくつの変数を持つ論理式の充足可能性判定ができるか（解を 1 つ見つけるか、解が存在しないことを証明する）を表している。当初数十変数しか扱えなかったものが、2000 年くらいで大きな進歩があり、現在では、数百万変数までも扱えるようになってきていることがわかる。これを受け、SAT 技術をハードウェア設計支援技術 [1] に応用する研究が極めて活発に行われている。特に、検証分野では、従来では考えられなかった規模のハードウェアの検証が可能となってきている。また、ソフトウェアの検証では、LINUX のカーネルコードのモデルチェッキングに成功した例も報告されている。

本論文では、SAT アルゴリズムとその形式的検証への応用について、最近の動向を中心に解説していく。なお、本稿では、特に断らない限り、ハードウェア設計の形式的検証を中心に説明していく。

## 2. SAT 問題

SAT とは，変数 $x$ を入力とする論理関数 $f(x)$ が与えられたとき，$f(x)=1$ となる (論理関数 $f$ を充足する) $x$ が存在する

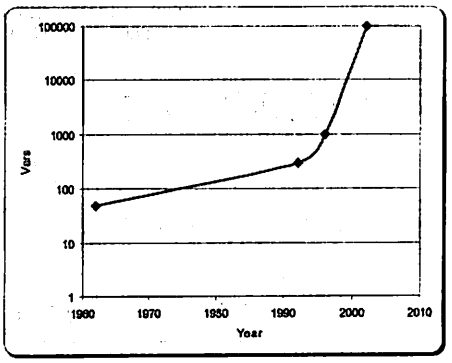


図 1 SAT ソルバの性能

かどうかを判定する問題であり，$x$が存在する場合はその$x$を求め，そうでない場合はそのような$x$が存在しないことを証明する．

SAT は，通常以下のような和積標準形 (CNF) に定式化する．

$$L_i = \{V_p, \overline{V_p}\} \tag{1}$$

$$C_j = (L_{j1} \vee L_{j2} \vee \cdots \vee L_{jm_j}) \tag{2}$$

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n = 1 \tag{3}$$

ここで，$V_p \in V$ は変数 (variable)，$\overline{V}$ は $V$ の否定，$\wedge$ は論理積，$\vee$ は論理和である．$L_i$ は変数 $V_p$ の肯定もしくは否定のいずれかであり，リテラル (literal) と呼ぶ．$C_n$ は一つもしくは複数のリテラルの論理和をとったものであり，クローズ (clause) と呼ぶ．和積標準形 (CNF) は，一つもしくは複数のクローズの論理積で表される．

CNF は，慣習的に，論理積の $\wedge$ や論理和の $\vee$ を省略して表記することがある．例えば，$(a+b)(\overline{a}+\overline{b})$ は，$(a \vee b) \wedge (\overline{a} \vee \overline{b})$ を意味する．ここで，$+$ は論理和である．この式は，$(a=1, b=0)$ もしくは $(a=0, b=1)$ のときに 1 になるため，この SAT は充足可能 (satisfiable) であるという．

一方，$(a+b)(\overline{a}+b)(\overline{b})$ は，変数に対してどのような値を割り当てても値を 1 にすることができないため，充足不能 (unsatisfiable) であるという．

一つの節の中に，最大 $k$ 個のリテラルを含む $(k = \max m_j)$ とき，この SAT は $k-SAT$ と呼ばれる．$k >= 3$ のとき，SAT は NP 完全になることが知られており，SAT を解くための計算量は，変数・節の数に対して指数関数的に増大する．なお，$k >= 4$ の SAT は，中間変数を導入することにより $3-SAT$ に帰着できる．

### 2.1 基本 SAT アルゴリズム

最近の SAT ソルバは、多くの場合、Davis-Putnam-Logemann-Loveland (DPLL) アルゴリズム[2], [3] に基づいているため、本論文では、DPLL アルゴリズムとその改良技術について説明していく。DPLL アルゴリズムは、基本的に場合分けとバックトラックを組み合わせながら、効率的な全解空間探索を行うもので、その処理が終了する場合には、解が存在する場合は必ず見つけ、また、解が存在しない場合は、それを証明できる。基本的な DPLL アルゴリズムを図 2 に示す。

```
sat-solve()
  if preprocess() = CONFLICT then
    return UNSAT;
  while TRUE do
    if not decide-next-branch() then
      return SAT;
    while deduce() = CONFLICT do
      blevel <- analyze-conflict();
      if blevel = 0 then
        return UNSAT;
      backtrack (blevel);
    done;
  done;
```

図 2 DPLL アルゴリズム

$$a(\overline{a}+c)(\overline{b}+c)(a+b+\overline{c})(\overline{c}+e)(\overline{d}+e)(c+d+\overline{e})$$
$$\Downarrow$$
$$c(\overline{b}+c)(\overline{c}+e)(\overline{d}+e)(c+d+\overline{e})$$
$$\Downarrow$$
$$e(\overline{d}+e)$$

図 3 Boolean constraint propagation の再帰的な適用例

まず、プリプロセスにより、自明に充足不可能な場合などを検出する。処理の基本ループでは、まず、従来選んでいない変数を場合分け変数として選び、値を決定する。もしもはやそのような変数が残っていない場合には、解が見つかったことになるため、充足可能とする。場合分けの変数がある場合には、それの値を決め、その値の決定を利用して論理式を簡単化する。これは、一般に Boolean Constraint Propagation (BCP) と呼ばれる処理であり、適用できる限り、再帰的に適用し、論理式を簡単化していく。

図 3 に BCP を適用した例を示す。最初の式では、クローズとして、リテラルが 1 つである $a$ があるため、変数 $a$ は必ず 1 である必要があり、まず、変数 $a$ を定数 1 に置き換える簡単化を行う（これで、2 番目の式になる)。すると今度は、変数 $c$ だけのクローズが現れるので、変数 $c$ を定数 1 に置き換える簡単化を行う（3 番目の式になる)。すると今度は、変数 $e$ だけを含むクローズが現れるので、変数 $e$ を定数 1 に置き換える簡単化を行うことになる。そうすると、一番下の式ではすべてのクローズが成立することがわかるので、元の論理式は充足可能であることがわかる。このように BCP は再帰的に適用可能となることが多い。実際、SAT ソルバにおいては、全処理時間の 80 %程度をこの BCP の処理に費やしている。上記の例では、BCP で充足可能とわかる場合であったが、逆にあるクローズが充足不可能（満たすには、定数値と逆の値が必要となるクローズが存在するなど）であることが検出できることもある。

BCP 処理で充足可能・不可能の結論が出ない場合には、変

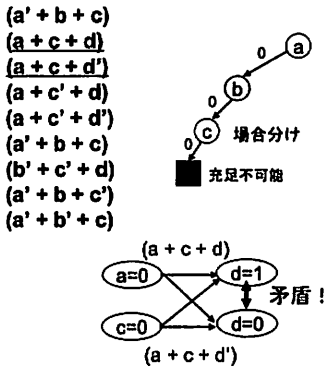


図 4　DPLL アルゴリズムの実行例－場合分け

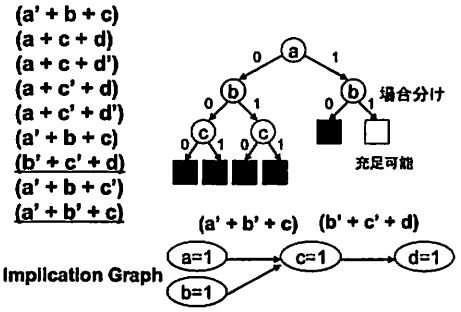


図 5　解を見つけた例

数値による場合分けを続けていく。この BCP の処理には、通常、implication graph と呼ばれるものを作成しながら、効率よく定数とすべき変数を求めていく。

図 4 に場合分けの実行例を示す。今、図に示すような CNF 式の充足可能性を判定するとする。なおここで、変数の後の「'」はその変数の否定を表すとする。図の右に示すように、場合分けとして、変数 $a, b, c$ がすべて 0 の場合を解析しているとする。すると、$a$ と $c$ が 0 であることから、クローズ $(a+c+d)$ と $(a+c+d')$ から、それぞれ、変数 $d$ の値が、1 であり、かつ 0 でなければいけないことがわかる。つまり、矛盾が生じている。この解析は、図の下の示す implication graph を利用して行っている。グラフのノードは変数値の割り当てであり、エッジは因果関係を表現している。これは、CNF 式内のクローズから求まる必要条件となっている。このように implication graph を随時構築することにより、効率的に変数の値の決定を行い、充足可能な場合や矛盾を起こす場合を検出していくことができる。

一方、矛盾が検出された場合には、変数値の場合分け処理をバックトラックし、異なる場合分けを試していく。すべての場合を調べても解が見つからない場合には、与えられた論理式は充足不可能であるとする。図 4 の処理を続けていくと、図 5 に示すように、最終的にこの例題では充足可能と判定される。

### 2.2　SAT アルゴリズムの改良

元の DPLL アルゴリズムでは、バックトラックは、辞書式順序で場合分けの変数を順に値を変えて処理していたが、最近のアルゴリズムでは、矛盾が生じた原因を解析し (analyze-conflict)、その結果に応じて、どこまで一度にバックトラックすべきかを決定している。これは、非辞書式バックトラックと呼ばれるもので、GRASP と呼ばれる SAT ソルバで提案された [4]。また、GRASP では、ラーニングについても提案されている。一般に、矛盾解析の結果、変数の値の決め方により、他の変数の与え方によらず、必ず矛盾になることがわかる場合がある。そのような場合には、矛盾の起こる値を組み合わせを記憶しておき、以降、そのような値の組み合わせを調べないようにすべきであり、conflict-driven ラーニングと呼ばれている。実際には、ラーニングは、矛盾が起きる値の組み合わせの否定を相当するクローズを元の論理式に追加することで行うことができる。

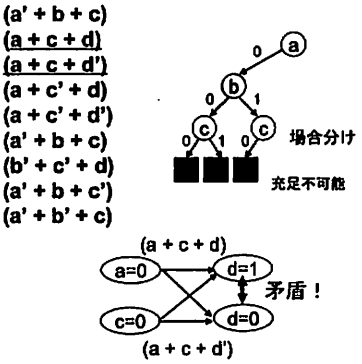


図 6　DPLL アルゴリズムの実行例

図 6 に図 4 と同じ例について、順次変数の割り当てを進め、$a=0, b=1, c=0$ となった場合の状況を示す。この場合も implication graph によって、変数 $d$ の値に矛盾が生じ、充足不可能であることが分かる。注意してみると、これに対する implication graph は図 4 の場合とまったく同じであることが分かる。つまり、$b$ の値は異なるが、それ以外は図 4 と図 6 は同じである。言い換えると、図 4 の状況が生じた際に、矛盾の原因が変数 $a, c$ にのみ依存し、変数 $b$ に依存しないと理解すれば、次の 2 つのことを行うことができ、結果的に図 6 の処理をスキップできる。

- バックトラックの際に変数 $b$ の割り当て変更はスキップして、直接、変数 $a$ の割り当て変更へ戻る（非辞書式バックトラックと呼ばれる）。
- 変数 $a, c$ が共に 0 となると必ず矛盾することが判明しているので、クローズ $(ac)$ を全体の式に付け加えてこのような割り当てを決してしないようにする（ラーニングと呼ばれる）。

これらの処理により、SAT ソルバが扱える CNF 式の規模（変数の数やクローズの数）は飛躍的に増大した。

CNF 式の性質を利用した改良として、SATO [5] と呼ばれる SAT ソルバで導入されたものがある。SATO では、クローズを表現するプログラム上のデータ構造をリストで表現した時に偽となっていないリテラルの最初と最後を指すポインタを導入し、最初の指すポイントの前のリテラルと最後を指すポインタの後のリテラルはすべて偽となるように管理する。このような

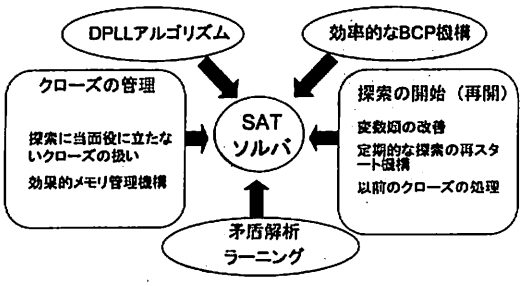


図 7　最新 SAT アルゴリズムの構成

ポインタを効率よく管理することで、BCP 処理が格段に効率化され、結果として、高性能な SAT ソルバを実現できる。ただし、バックトラックの際にも正しく管理する必要があるため、効率には限界がある。これに対し、Chaff と呼ばれる SAT ソルバ [6] では BCP 処理を効率化するために、two literal watching with lazy update と呼ばれる手法が導入されている。これは、2 つ以上のリテラルが残っているクローズは、ある変数の値が決まってもそれだけでは直ちに真となることはないという事実に着目し、値の決定していないリテラルの数が 2 個以上か以下かを中心にクローズの管理を行う。このようにしても、BCP 処理は SATO 手法と同様に効率化される。一方、バックトラック時のポインタの管理は SATO より格段に楽になる。結果として、実際のベンチマーク結果では、当初誰もが驚くほどの結果を示した。Chaff では、場合分けの変数順の決定手法でも新しい提案を行っている。また、BerkMin と呼ばれる SAT ソルバ [7] では、そららの手法がさらに発展されている。これらの手法では、場合分けの変数順を決定する際に、今までの処理で矛盾を発生させたクローズ（図 4 のクローズ $(a+c+d)$ とクローズ $(a+c+d')$ など）に現れる変数を中心に決定するものであり、SAT ソルバの性能向上に大きく貢献している。現在もこれらの SAT ソルバの改良は引き続き活発に行われており、SAT ソルバを競うワークショップも毎年開催され、その性能も毎年大きく改善することが続いている。なお、Chaff などの SAT ソルバの多くは、パブリックドメインツールとしてウェブなどに公開されている。その利用の仕方については、例えば、[8] などを参照されたい。Chaff を自分自身で改良して大きな性能向上を達成している研究者や技術者も多い。

以上をまとめると、最新の SAT ソルバの構成は、図 7 に示すようになる。基本は DPLL アルゴリズムであるが、それに効率的な BCP 処理のための機構が追加されている。また、矛盾解析やそれを元にしたラーニング機能も追加されている。さらに、SAT ソルバの効率的なプログラム実装という観点からは、効率的なクローズの管理機能や、場合分けの順を飛ばしたり、一度リセットして再スタートする機能も追加されている。このような実装により、現在、数百万変数の論理式の充足可能性判定が数時間で行えるようになってきている。

なお、ツールによっては、CNF 式ではなく、例えば論理回路表現をそのまま受け付けるようになっているものもある。また、形式的検証などに応用すると、クローズの集合が順次増大

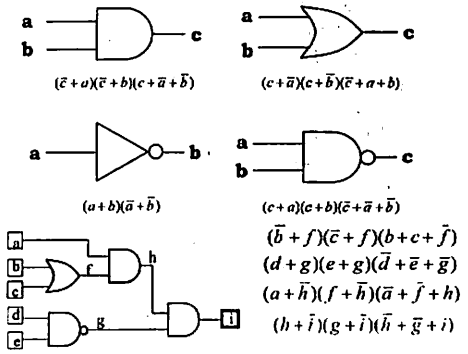


図 8　論理回路から CNF 式の生成

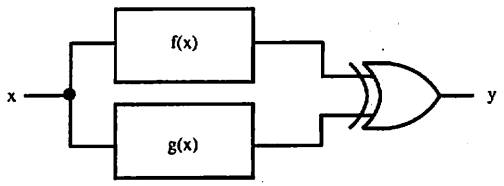


図 9　二つの異なる組み合わせ回路の論理的等価性

していくような問題を次々に解く必要がある場合も多い。このような問題に合わせて、以前の問題の経過を記憶して、次の問題でも利用するインクリメンタルな SAT ソルバ [9] も開発されている。また、SAT 手法と関係が深い、製造故障検出のためのパターン生成を行う ATPG 手法を SAT 問題に適用する研究も行われており [10]、また後述するモデルチェッキングへも応用されている [11]。

## 3.　形式的検証問題への応用

### 3.1　等価性検証への応用

SAT ソルバの最も基本的な形式的検証への応用として、組み合わせ回路の等価性検証がある [12]。そのためには、まず、与えられた組み合わせ論理回路を CNF 式に変換する必要がある。この変換は、各ゲートごとに変換でき、変換された CNF 式の長さ（クローズの数）は、ゲート数に比例する。変換の例を図 8 に示す。2 入力基本ゲートに対しては、2 つの入力変数と出力変数の関係として、3 つのクローズが作られている。

等価性検証を行う場合には、図 9 に示すように、比較する 2 つの回路の出力を排他的論理和で接続し、その出力が 1 となり得るかという問題を SAT ソルバで解く事になる [12]。

また、SAT ソルバを利用し、あるいは拡張して、論理回路より上位の設計記述の等価性検証を行う手法についても研究されている。例えば、C 言語記述に対して記号シミュレーションを実行し、その結果の等価性解析から元の 2 つの C 言語記述の等価性判定を行う手法がある [13]。この種の手法では、ある種の述語論理を SAT 式に変換して解析する手法 [14] など、様々な SAT ソルバの利用技術が研究されている [15]〜[17]。

### 3.2　モデルチェッキングへの応用

モデルチェッキング [18] とは、設計が特定の性質（プロパ

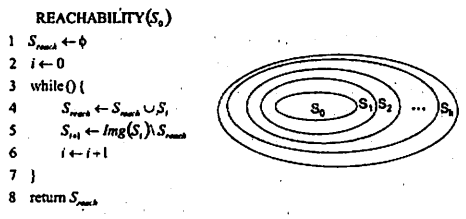


図 10　状態遷移の到達可能状態の計算

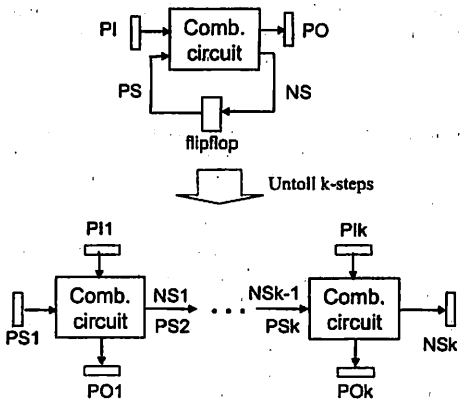


図 11　Bounded な形式的検証手法

ティ）をいつも満たすか否かを検証することを指し、プロパティチェッキンとも呼ばれる。基本アルゴリズムは、図 10 に示すように、初期状態から到達可能な状態を網羅的に調べていくことで、設計を検証していく。到達可能な状態をすべて求めるには、到達可能な状態が最早増加しなくなるまで（fixed point に到達したという）状態遷移を繰り返していく必要がある。元々のアルゴリズムは、状態を 1 つ 1 つ辿るものであったが、状態の集合を一度に処理できる記号モデルチェッキング [19] が開発されて扱える回路規模が一気に増大した。また、設計を一度抽象化してからモデルチェッキングし、必要に応じてその抽象化処理を改良しながら検証する Counter Example Guided Abstraction Refinement （CEGAR）[20] 手法も利用され、大規模設計へのモデルチェッキングの適用も行われている。

SAT 問題として、モデルチェッキングを定式化する方法として、最も簡単なのは、すべての到達可能な状態を求めるのではなく、一定回数の状態遷移だけで到達可能な状態を調べる、Bounded モデルチェッキング [21] と呼ばれるものである。これは、図 11 順序回路中の組み合わせ回路を指定した回数だけ、時間軸方向に展開することに相当する。このように展開すると、検証対象の順序回路は図のように組み合わせ回路として扱うことができ、SAT ソルバで直接扱うことができる。

例えば、いつかはプロパティ $P$ が成立することを調べたい場合には、その否定である、いつも $P$ は成り立たない、という性質を調べる。そうすると、図 12 に示すように、時間軸上に展開された回路に対して SAT 問題として定式化できる。この際、生成される CNF 式の長さは、ゲート数に比例する（すべてが 2 入力ゲートでできている回路ならば、CNF 式の長さは、ゲート数の 3 倍になる）。またこの場合、現れる変数の数は、ネット数と同じなる。したがって、元の順序回路（仮にすべて 2 入力ゲートで構成されているとする）にゲートが N 個あり、それを M 時刻だけ解析する Bounded な形式的検証手法では、生成される CNF 式には、3*N*M 個の変数が現れることになる。最新の SAT プログラムが解ける規模は、大体 100 万変数、数百万項程度なので、1000 時刻解析すると 1000 ゲート規模の回路しか扱えないが、10 時刻の解析でよければ 10 万ゲート規模でも扱えることになる。10 万ゲートという大きさは、RTL から論理回路を自動的に合成する際の回路規模程度であり、形式的検証ツールとして実用的であるといえる。実際、RTL やゲート回路に対する形式的検証ツールは、ハードウェアの設計現場でも一定の利用価値が認められ、導入が進んでいる。

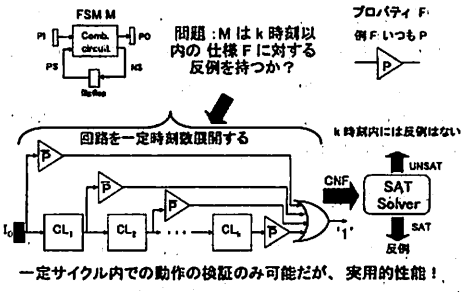


図 12　SAT 手法による Bounded なモデルチェッキング例

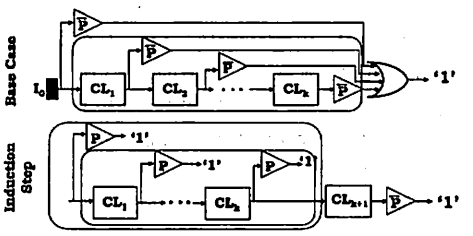


図 13　SAT を利用した帰納法によるモデルチェッキング

なお、最近は、fixed point の計算を SAT 問題として定式化する手法がいくつか提案 [22]〜[25] され、実験的なツールも開発されている。これにより、SAT ソルバの性能向上の恩恵をそのまま受けるモデルチェッキングツールの開発が可能になりつつある。例えば、数学的帰納法を適用する手法がある。これは、図 13 に示すように、帰納法のベースケースと機能ステップをそれぞれ、図のように SAT 問題として定式化して解くというものである。このように、従来 SAT としては定式化が難しかった fixed point 計算についても、SAT 問題としての定式化手法がいくつか考案されている。以前は、2 分決定グラフ（BDD）で実装されてきた検証手法も、SAT ソルバの性能向上によって、SAT での実装に置き換わっている場合も多い。BDD による実装では、一度 BDD サイズが大きくなってしまい、計算機の主記憶容量を超えると、なかなか効率的な処理は難しい。これに対し、SAT 手法では、メモリ消費量が大きく増大することはないので、その意味でも扱いやすい。このため、SAT 手法を利用した研究が活発に行われている。

扱える規模の増大から、SATソルバを利用したソフトウェアのモデルチェッキングについても、実用的な結果が得られるようになってきている[26]。この分野も注目を集めており、SATソルバを利用したモデルチェッキング手法により、LINUXカーネルのコードのバグを発見したという報告もされている。

## 4. おわりに

本稿では、SATアルゴリズムについて、DPLL法を中心とした技術を解説した。アルゴリズム的な改良だけでなく、プログラムとして実装する際の効率化も十分考慮した改良が続けられてきている。結果として、数百万変数からなる論理式の判定も行えるようになり、様々な分野にも応用されるようになっている。その例として、本稿では、形式的検証手法への適用法とその性能について解説した。SAT技術の進歩により、実用規模のハードウェア設計の検証が可能となるとともに、ソフトウェア検証にも応用されつつある。SATソルバは毎年、目覚しい性能改善が報告されており、今後楽しみな分野であるといえる。

現在、手法なSATソルバはパブリックドメインプログラムとして、ウェブからダウンロード可能なものが多い。実際、例えば、Chaff[6]プログラムのソースを見て、場合分けの変数順を自分の応用に合わせて調整するといったこともさかんに行われている。このようにすると比較的簡単に性能を10倍程度高めることもできるとする報告も多い。読者自身でもチャレンジされてみては、いかがだろうか？

残念ながら、SAT手法に関する研究、特に実用的な研究は日本ではさかんであるとはいえない。本稿をきっかけに、興味を持つ人が増えれば幸いである。

## 文　献


[1] 藤田（編著）.: システムLSI設計工学, オーム社, 2006年.

[2] M. Davis, G. Logemann, and D. Loveland.: A Machine Program for Theorem-Proving, Communications of the ACM, 5(7):394-397, July 1962.

[3] M. Davis and H. Putnam.: A Computing Procedure for Quantification Theory, Journal of the ACM, 7(3):201-215, July 1960.

[4] J.P. Marques-Silva and K.A. Sakallah.: GRASP: A Search Algorithm for Propositional Satisfiability, IEEE Transactions on Computers, 48(5):506-521, May 1999.

[5] H. Zhang.: SATO: An Efficient Propositional Prover, Proceedings of the 14th International Conference on Automated Deduction (CADE), July 1997.

[6] M.H. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik.: Chaff: Engineering an Efficient SAT Solver, Proceedings of the 38th Design Automation Conference (DAC), June 2001.

[7] E. Goldberg and Y. Novikov.: BerkMin: a Fast and Robust Sat-Solver, Proceedings of Design Automation and Test in Europe (DATE), March 2002.

[8] 浅田, 藤田（編集）.: システムLSI設計自動化技術の基礎ーパブリックドメインツールの利用法, 培風館, 2005年.

[9] J.P. Whittemore, J. Kim, and K. A. Sakallah.: SATIRE: A New Incremental Satisfiability Engine, Proceedings of the 38th Design Automation Conference (DAC), June 2001.

[10] M.K. Iyer, G. Parthasarathy, and K.-T. Cheng.: SATORI-A Fast Sequential SAT Engine for Circuits, Proceedings of the International Conference on Computer-Aided Design (ICCAD), November 2003.

[11] V. Boppana, S.P. Rajan, K. Takayama, and M. Fujita.: Model Checking Based on Sequential ATPG, Proceedings of the 11th International Conference on Computer-Aided Verification (CAV), July 1999.

[12] E. Goldberg, M.R. Prasad, and R.K. Brayton.: Using SAT for Combinational Equivalence Checking, Proceedings of Design Automation and Test in Europe (DATE), March 2001.

[13] Matsumoto, T. and Saito, H. and Fujita, M.: An Equivalence Checking Method for C Descriptions Based on Symbolic Simulation with Textual Differences, *Trans. IEICE*, Vol. E88-A, No. 12, pp. 3315-3323.

[14] Barrett, C. and Berezin. S.: CVC Lite: A new implementation of the cooperating validity checker, In Proceedings of the 16th International Conference on Computer Aided Verification (CAV).

[15] P.F.Williams, A. Biere, E.M. Clarke, and A. Gupta.; Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking, Proceedings of the 12th International Conference on Computer Aided Verification (CAV), July 2000.

[16] O. Strichman.: On Solving Presburger and Linear Arithmetic with SAT, Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD), November 2002.

[17] H. Jain, D.Kroening, N. Sharigina, E. Clarke.: Word level predicate abstraction and refinement for verifying RTL verilog Proceedings of DAC '05.

[18] E.M. Clarke and E.A. Emerson.: Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic, Proceedings of theWorkshop on Logic of Programs, 1982.

[19] J.R. Burch, E.M. Clarke, D.E. Long, K.L. McMillan, and D.L. Dill.: Symbolic model checking for sequential circuit verification, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13(4):401-424, April 1994.

[20] Clarke, E.M. and Grumberg, O. and Jha, S. and Lu, Y. and Veith, H.: Counterexample-guided abstraction re nement, In Proceedings of CAV, volume 1855, pages 154-169. Springer LNCS, 2000.

[21] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu.: Symbolic Model Checking using SAT procedures instead of BDDs, Proceedings of the 36th Design Automation Conference (DAC), June 1999.

[22] P. Chauhan, E.M. Clarke, J. Kukula, S. Sapra, H. Veith, and D. Wang.: Automated Abstraction Refinement for Model Checking Large State Spaces using SAT based Conflict Analysis, Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD), November 2002.

[23] M.K. Ganai, A. Gupta, and P. Ashar.: Efficient SAT-based Unbounded Symbolic Model Checking using Circuit Cofactoring, Proceedings of the International Conference on Computer-Aided Design (ICCAD), November 2004.

[24] K.L. McMillan.: Applying SAT Methods in Unbounded Symbolic Model Checking, Proceedings of the 14th International Conference on Computer-Aided Verification, July 2002.

[25] P. Chauhan, E. Clarke, and D. Kroening.: Using SAT based Image Computation for Reachability Analysis, Technical report, CMU-CS-03-151.

[26] E. Clarke, D. Kroening, and F. Lerda.: Bounded Model Checking for Software, Proceedings of TACAS '04.