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

2007－ＳＬＤＭ－129／2007－ＥＭＢ－4（19）
2007／3／16

論理回路の SAT ベース形式的検証の高速化のための BDD を用いた CNF 式生成手法

中村一博† 成瀬智啓†† 高木一義† 高木直史†

† 名古屋大学大学院情報科学研究科 〒464-8603 愛知県名古屋市千種区不老町
†† 名古屋大学工学部 〒464-8603 愛知県名古屋市千種区不老町
E-mail: †{nakamura,ktakagi,ntakagi}@is.nagoya-u.ac.jp, ††naru@takagi.i.is.nagoya-u.ac.jp

あらまし 本稿では、SAT を用いた論理回路の形式的検証の高速化を目指し、論理回路から SAT-solver の実行時間が短かくなるような和積標準形 CNF 論理式を生成する、CNF 式生成処理フレームワークを提案する。提案するフレームワークは、回路分割と、部分回路の BDD 生成、BDD から CNF 式への変換から構成される。また本稿では、提案フレームワークに基づく、中間変数の間隔を考慮した回路分割、BDD 生成、BDD から CNF 式への変換アルゴリズムを示す。提案する CNF 式生成手法と広く用いられている CNF 式生成手法を実装し、SAT-solver の実行時間について比較を行ったところ、提案手法により実行時間が短縮されることが確認された。
キーワード 充足可能性判定、二分決定グラフ、和積標準形論理式、形式的検証、論理回路

Efficient Translation of Logic Circuits to CNF Formulae with BDD for Acceralating SAT-based Formal Verification

Kazuhiro NAKAMURA†, Tomohiro NARUSE††, Kazuyoshi TAKAGI†, and Naofumi TAKAGI†

† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, Aichi, 464-8603 Japan
†† School of Engineering, Nagoya University
E-mail: †{nakamura,ktakagi,ntakagi}@is.nagoya-u.ac.jp, ††naru@takagi.i.is.nagoya-u.ac.jp

Abstract In this paper, we present a translation framework of circuits to CNF formulae for acceralating SAT-based formal verification. The translation framework consists of three elements, circuit partitioning, construction of BDDs and conversion of BDDs to CNF formulae. We also present a translation method of circuits to CNF formulae based on the translation framework. Experimental results show that the proposed method has effect on the reduction of SAT-solver execution time.
Key words SAT, BDD, CNF formulae, formal verification, logic circuits

1. まえがき

近年、集積回路技術の発展により、ますます大規模な論理回路の設計が行われるようになってきている。設計される論理回路の大規模化に伴い、設計誤りを犯す危険性が高まっている。そのため、設計された論理回路に誤りがないことを数学的に保証する形式的検証の研究が行われている。

また、論理式の充足可能性判定（Satisfiability; SAT）問題を解く SAT-solver の性能向上に伴い、SAT を用いた形式的検証手法の研究が盛んに行われている [1]。

SAT 問題は、与えられた論理式が真となるような変数割り当てが存在するかどうかを判定する問題である。SAT-solver が入力とする論理式は和積標準形（Conjunctive Normal Form; CNF）論理式であため、SAT-solver を用いた形式的検証では、論理回路を CNF 式に変換する必要がある。一般に論理回路の CNF 式表現は一意ではない上、充足可能性判定に要する時間は SAT-solver に与える CNF 式により変化する。そのため、CNF 式の生成処理は、SAT ベースの形式的検証の高速化をはかる上で重要な問題である。

これまでに、マイクロプロセッサの検証において、連続する ITE（If Then Else）ゲートがコンパクトな CNF 式に変換できることに注目し、プロセッサの回路記述から ITE を多く含

む最適化された Bool 式を生成し、その式を CNF 式に変換する方法が提案されている [2], [3]。また、与えられた CNF 式に対して、変数や節を減らす最適化を行う SAT-solver のプリプロセッサも提案されている [4]。

本稿では、論理回路の二分決定グラフ BDD (Binary Decision Diagram; BDD) 表現の BDD 節点が、ITE に対応することに着目する。そして、SAT を用いた一般の論理回路の形式的検証の高速化を目指し、CNF 式生成処理フレームワークを提案する。提案するフレームワークは、SAT-solver の実行時間が短くなるような CNF 式を生成する問題を 3 つの処理に分けて解く枠組であり、回路分割と、部分回路の BDD 生成、BDD から CNF 式への変換から構成される。このフレームワークは、論理回路を分割し、分割で得られた部分回路の BDD を構築し、部分回路の BDD から CNF 式を生成する。SAT-solver の実行時間の短縮に効果のある、回路分割、部分回路の BDD 生成、BDD から CNF 式への変換の、各アルゴリズムの研究を進めることにより、論理回路の形式的検証の高速化を目指す。

また本稿では、提案フレームワークに基づく、中間変数の間隔を考慮した回路分割、BDD 生成、BDD から CNF 式への変換アルゴリズムを示す。この CNF 式生成手法と、SAT を用いた形式的検証において広く用いられている CNF 式生成手法 [5] を実装し、比較を行った。順序回路の sequential depth 計算 [6] と組合せ回路の等価性判定の CNF 式生成における SAT-solver [4], [7], [8] の実行時間を比較したところ、提案手法により実行時間が短縮されることが確認された。

以下 2 では準備、3 では CNF 式生成処理フレームワーク、4 では回路分割と BDD を用いた CNF 式生成手法、5 では実験と結果を示す。

2. 準　　備

2.1 ブーリアン・ネットワーク

ブーリアン・ネットワーク [9] は非巡回有向グラフ $G=(V,E)$ で、ノード $j \in V$ がノード $i \in V$ に依存するときかつそのときに限り、ノード i からノード j への辺 $e \in E$ が存在する。入次数が 1 以上のノード $v \in V$ は論理関数 F_v をもつ。

論理回路の各論理ゲートは、ブーリアン・ネットワークではノードで表される。ブーリアン・ネットワーク中の入次数が 0、出次数が 0 のノードは、それぞれ論理回路の外部入力、外部出力に対応する。外部入力に対応するノードの論理値が定まれば、外部出力に対応するノードの論理値が計算できる。任意の論理回路はブーリアン・ネットワークで表現できる。以下では、論理回路を構成する論理ゲートは、XOR, OR, AND, NOT の 4 種類とする。

2.2 論理式の充足可能性判定

$x_i (i=1,...,p)$ を真または偽の値をとる論理変数とする。$\overline{x}_i$ は論理変数 x_i の論理否定を表す。論理変数と論理変数の否定をリテラルという。$\vee$ と $\wedge$ は、それぞれ論理和と論理積を表す。リテラルを論理和でつないだものを節という。節を論理積でつないだものを和積標準形 (Conjunctive Normal Form; CNF) 式という。以下では、真を 1 に、偽を 0 に対応づける。

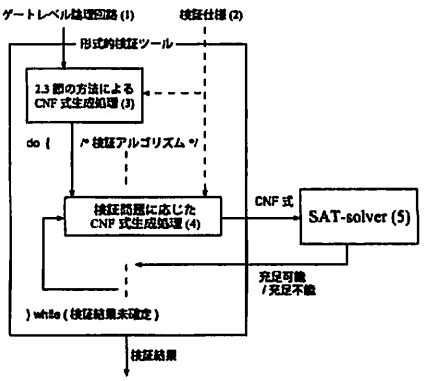


図 1　SAT を用いた形式的検証の処理フロー

変数 $x_1,...,x_p$ からなる CNF 式 f が与えられたとき、その充足可能性判定問題とは、f を真にする変数の値の割り当てが存在するかどうかを判定する問題である。そのような変数の割り当てが存在するとき、f は充足可能であるといい、存在しないとき f は充足不能であるという。

この SAT 問題を解くプログラムが SAT-solver である。SAT-solver に与える入力は CNF 式であるため、SAT を用いた論理回路の形式的検証では、論理回路を CNF 式に変換する必要がある。

2.3 論理回路から CNF 式への変換

本節では、論理回路から CNF 式への変換手法として、全ての論理ゲートに中間変数を導入して CNF 式を生成する方法を示す [5]。この方法は、変換処理が簡単なため SAT を用いた論理回路の形式的検証で広く用いられている。論理回路中の各論理ゲートに中間変数が導入され、論理ゲート毎に CNF 式に変換される。変換規則は以下のように、ゲート毎にあらかじめ決まっており、論理回路から簡単に CNF 式を生成することができる。ここで、$\wedge, \vee, \overline{\ }, \neq, \equiv$ は、それぞれ AND, OR, NOT, XOR, 等価 (XOR の否定) を表す。

- $(i \equiv (x \wedge y)) \Rightarrow (i \vee \overline{x} \vee \overline{y}) \wedge (\overline{i} \vee x) \wedge (\overline{i} \vee y)$
- $(i \equiv (x \vee y)) \Rightarrow (\overline{i} \vee x \vee y) \wedge (i \vee \overline{x}) \wedge (i \vee \overline{y})$
- $(i \equiv \overline{x}) \Rightarrow (\overline{i} \vee \overline{x}) \wedge (i \vee x)$
- $(i \equiv (x \neq y)) \Rightarrow (\overline{i} \vee \overline{x} \vee \overline{y}) \wedge (\overline{i} \vee x \vee y) \wedge (i \vee \overline{x} \vee y) \wedge (i \vee x \vee \overline{y})$
- $(i \equiv (x \equiv y)) \Rightarrow (\overline{i} \vee \overline{x} \vee y) \wedge (\overline{i} \vee x \vee \overline{y}) \wedge (i \vee \overline{x} \vee \overline{y}) \wedge (i \vee x \vee y)$

例えば、$v_1' = (\overline{v}_1 \wedge x) \vee (v_2 \wedge \overline{x})$ を中間変数 i_1, i_2, i_3, i_4 を導入すると $(i_1 \equiv (v_1' \equiv i_2)) \wedge (i_2 \equiv (i_3 \vee i_4)) \wedge (i_3 \equiv \overline{v}_1 \wedge x)) \wedge (i_4 \equiv (v_2 \wedge \overline{x}))$ となり、これに上記の変換規則を適用すると CNF 式が得られる。

2.4 SAT を用いた論理回路の形式的検証

図 1 に、SAT を用いた論理回路の形式的検証の処理フローを示す。(1) ゲートレベルの論理回路と (2) 検証仕様が、形式的検証ツールへの入力である。検証仕様は、検証の問題に応じて、論理回路の場合とプロパティの場合がある。また、検証仕様は、検証の問題により検証アルゴリズム中に組込まれている

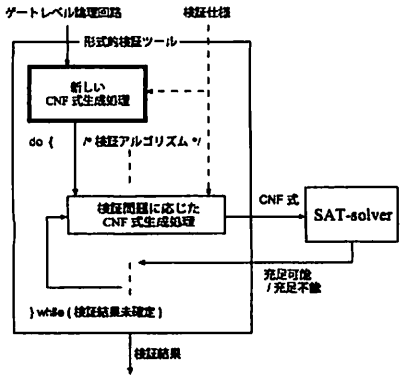


図 2 CNF 式生成処理の置き換えによる SAT を用いた形式的検証の高速化の高速化

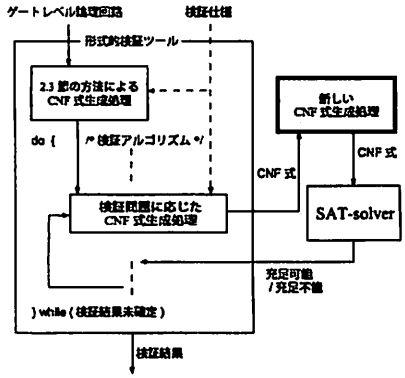


図 3 SAT-solver のプリプロセッサによる SAT を用いた形式的検証の高速化

場合もあり、与えられないこともある。まず、論理回路は (3) 2.3 節の CNF 式生成手法により CNF 式に変換され、次に、この論理回路の CNF 式と検証仕様から (4) SAT-solver に渡される CNF 式が生成される。そして、(5) SAT-solver が実行され、その結果に応じて検証アルゴリズムは終了するか、CNF 式の生成処理を行う。SAT-solver は外部のプログラムである。

一般に論理回路の CNF 式表現は一意ではない上、SAT-solver の実行時間は与えられる CNF 式に左右されるため、SAT-solver の実行時間が短くなるような CNF 式を生成することが、SAT を用いた論理回路の形式的検証を高速化する上で重要である。

高速化のアプローチとして、次の 2 つのアプローチがある [2]〜[4]。SAT-solver の実行時間が短くなるような CNF 式の生成処理を導入した SAT ベース形式的検証の処理フローを図 2、図 3 に示す。図 2 の構成は、図 1 の CNF 式生成処理を、最適化された処理に置き換えた構成である [2], [3]。また、図 3 の構成は、形式的検証ツールは変更せず、CNF 式の最適化手法を SAT-solver のプリプロセッサとして持つ構成である [4]。図 2 の構成は、回路構造の情報を用いた CNF 式の最適化は行いやすいが、既存の形式的検証ツールの再利用性は低い。図 3 の構成は、既存の形式的検証ツールとの親和性が高いという特徴がある。

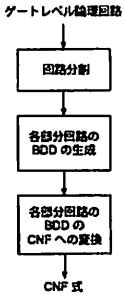


図 4 提案する CNF 式生成処理フレームワーク

3. SAT ベース形式的検証の高速化のための CNF 式生成処理フレームワーク

我々は、一般の論理回路を BDD 表現した際の BDD の節点が ITE に対応することに着目し、BDD を用いた CNF 式生成手法処理フレームワークを提案する。このフレームワークの各要素アルゴリズムの研究を進めることにより、論理回路の形式的検証の高速化を目指す。

3.1 CNF 式生成処理フレームワーク

図 4 に、提案する CNF 式生成処理フレームワーク示す。このフレームワークは 3 つの処理、回路の分割処理、各部分回路の BDD 生成処理、BDD から CNF 式への変換から成る。回路分割では、入力として与えられるゲートレベルの論理回路を部分回路に分割する。部分回路の BDD 生成では、部分回路毎に BDD を作成する。BDD から CNF 式への変換では、部分回路の BDD 毎に CNF 式を生成する。

このフレームワークは、論理回路から SAT-solver の実行時間が短くなるような CNF 式を生成するという問題を、3 つの処理に分けて解く枠組である。SAT-solver の実行時間の短縮に効果のある、各処理のアルゴリズムを開発することにより、論理回路の形式的検証の高速化を目指す。

以下では、提案フレームワークに基づき、SAT-solver の実行時間の短縮を目的とした、回路の分割アルゴリズム、BDD の生成アルゴリズム、BDD から CNF 式への変換アルゴリズムを示す。

4. 回路分割と BDD を用いた CNF 式生成手法

4.1 中間変数の間隔を考慮した回路分割

2.3 節の CNF 式生成手法 [5] では、回路をゲート単位で分割していたが、提案手法では図 5 のように、ゲートが統合された形の部分回路に分割する。外部入力と、中間変数間の間隔 `lev` が閾値 `LevelThre` 以下になるように導入された中間変数とで囲まれる部分が、分割処理により得られる部分回路である。`LevelThre` の値を大きく（小さく）設定すると、分割により得られる部分回路のサイズは大きく（小さく）なり、部分回路の個数は少なく（多く）なる。`LevelThre` の値を 20 程度まで大

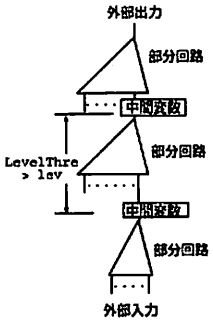


図 5 中間変数の間隔を考慮した回路分割

```
LevelbasedDepthFirstPartitioning (G(V, E))
   G(V, E): ブーリアン・ネットワーク
      POs: ブーリアン・ネットワークのノードの集合
        g: ブーリアン・ネットワークのノード
begin
 1:  POs = ∪ G(V, E) の出次数 0 のノード (外部出力) ;
 2:  foreach g ∈ POs
 3:    LevelbasedDepthFirstPartitioning_sub (g, 1);
end

LevelbasedDepthFirstPartitioning_sub (v, lev)
          v: ブーリアン・ネットワークのノード
        lev: 中間変数からノード v までの段数
     v1, v2: v のファンイン
  LevelThre: 中間変数を導入する閾値
begin
 1:  // 変数の場合、リターン
 2:  if (v が中間変数 or v が外部入力)
 3:    return;
 4:
 5:  // AND, OR, XOR ゲートの場合、更に探索
 6:  if (v が AND ゲート
 7:           or v が OR ゲート or v が XOR ゲート ) {
 8:    if (LevelThre == lev) {
 9:      ノード v に中間変数を導入;
10:      lev = 1;
11:    } else
12:      lev = lev + 1;
13:      LevelbasedDepthFirstPartitioning_sub (v1, lev);
14:      LevelbasedDepthFirstPartitioning_sub (v2, lev);
15:      return;
13:  }
14:
15:  // NOT ゲートの場合、更に探索
16:  if (v が NOT ゲート) {
17:    if (LevelThre == lev) {
18:      ノード v に中間変数を導入;
19:      lev = 1;
20:    } else
21:      lev = lev + 1;
22:      LevelbasedDepthFirstPartitioning_sub (v1, lev);
23:      return;
24:  }
end
```

図 6 中間変数の間隔を考慮した回路分割アルゴリズム

きくすると、回路分割の次の処理の BDD 生成において BDD のサイズが爆発的に大きくなる可能性がある。SAT-solver の実行時間が最も短くなるような LevelThre の値は、回路によって異なるため、LevelThre の値は、2 から 20 程度までのいくつかの値で CNF 式を生成し、SAT-solver を実行して、その実行時間の傾向から決める。

```
mkBDD (G(V, E))
   G(V, E): ブーリアン・ネットワーク
     IMVs: 中間変数の集合
begin
 1:  IMVs = ∪ 中間変数の導入されたノード;
 2:  foreach v ∈ IMVs
 3:    mkBDD_sub (v, 1);
end

mkBDD_sub (v, flg)
      f: BDD
    flg: フラグ、部分回路の根の時 1 それ以外 0
  begin
 1:  // 変数の場合、リターン
 2:  if ((flg が 0 and v が中間変数) or v が外部入力)
 3:    return ノード v に対応する BDD ;
 4:
 5:  // AND, OR, XOR ゲートの場合
 6:  if (v が AND ゲート) {
 7:    f = mkbdd_sub (v_1, 0);
 8:    g = mkbdd_sub (v_2, 0);
 9:    return BDDand (f, g);
10:  } else if (v が OR ゲート) {
11:    f = mkbdd_sub (v_1, 0);
12:    g = mkbdd_sub (v_2, 0);
13:    return BDDor (f, g);
14:  } else if () {
15:    f = mkbdd_sub (v_1, 0);
13:    g = mkbdd_sub (v_2, 0);
14:    return BDDxor (f, g);
15:  }
16:
17:  // NOT ゲートの場合
18:  if (v が NOT ゲート) {
19:    f = mkbdd_sub (v_1, 0);
20:    return BDDnot (f);
21:  }
end
```

図 7 部分回路の BDD の作成アルゴリズム

図 6 に、中間変数の間隔を考慮した回路分割アルゴリズムを示す。このアルゴリズムは、2 つの処理、LevelbasedDepthFirstPartitioning (LBDFP) と、LevelbasedDepthFirstPartitioning_sub (LBDFPsub) から構成される。LBDFP への入力は論理回路の組み合わせ回路部のブーリアン・ネットワークで、LBDFP の 1 行目では、外部出力の集合を PSs にセットしている。2 行目、3 行目では、PSs の各ノードに対して、中間変数の間隔を考慮した回路分割 LBDFPsub を行っている。LBDFPsub は、論理回路を解析して中間変数を導入する。解析は論理回路の外部出力からスタートし、論理ゲートを外部入力に向かって深さ優先探索により行う。lev の値は、1 に初期化されており、12 行目、21 行目では、再帰処理の度にインクリメントされる。lev の値が決められた閾値 LevelThre になった時、9 行目、18 行目では、そのゲート v に中間変数が導入される。

4.2 部分回路の BDD の作成

図 7 に、部分回路の BDD を作成するアルゴリズムを示す。このアルゴリズムは、アプライ演算 [10] により BDD を構成するもので、mkBDD と、mkBDD_sub から構成される。mkBDD の 1 行目では、中間変数を導入したノードの集合を $IMVs$ にセットしている。2 から 3 行目では、$IMVs$ の各ノードに対して、mkBDD_sub を実行する。mkBDD_sub は、ブーリアン・ネット

ワークを探索しノード毎に BDD を生成する。探索は、部分回路のブーリアン・ネットワークの根からスタートして、外部入力に向かって深さ優先探索によりノードをたどる。そしてノード v が変数の時、mkBDD_sub の 3 行目の処理で、変数 v に対応する BDD 変数を返す。また、ノード v が論理ゲートの時は、mkBDD_sub の 9 行目の BDDand、13 行目の BDDor、14 行目の BDDxor、20 行目の BDDnot の処理で、各論理演算に対応した演算を行った BDD を返す。mkBDD_sub 中の flg は v が部分回路のブーリアン・ネットワークの根のノードかどうかを表している。flg が 1 の時は中間変数の導入されている根 v でリターンしない。

この部分回路の BDD 生成処理により、外部入力の変数と、中間変数の変数から成る各部分回路の BDD が構築される。BDD の変数順序は、外部出力から深さ優先探索で到達する外部入力変数の順、ならびに、中間変数の導入された順序とした。

4.3 BDD から CNF 式への変換

図 8 に、各部分回路の BDD から CNF 式を生成するアルゴリズムを示す。アルゴリズムは、PathbasedBDDtoCNFTrans (PBBCT) と、PathbasedBDDtoCNFTrans_sub (PBBCTsub) から構成される。PBBCT の 1 行目では、各部分回路の BDD の集合を $SubCircBDDs$ にセットしている。2 から 3 行目では、$SubCircBDDs$ の各 BDD に対して、PBBCTsub を実行している。PBBCTsub は、BDD を解析してパス毎に CNF 式の節を生成する。解析は BDD の根からスタートして、BDD を葉に向かって深さ優先探索により行われる。葉に到達した時、根からその葉までパスに対して CNF 式の節を生成する。PBBCTsub の 1 行目の $f_{x_i=0}$, 13 行目の $f_{x_i=1}$ は、それぞれ BDD f の変数 x_i に 0, 1 を代入することを表している。また、配列 $path[]$ は BDD の根から葉へのパスを記憶する配列で、2 行目、14 行目では、x_i に代入した値に対応するリテラルを $path[i]$ にセットしている。3 行目、6 行目、16 行目、19 行目では、変数 x_i への値の代入により、BDD の定数節点に到達したかどうかを判定している。ここで、定数節点に到達していれば、4 行目、7 行目、17 行目、20 行目で、根から葉までのパスを記憶した配列 $path[]$ の、$path[0]$ から $path[i]$ に格納されているリテラルの否定の和を出力する。また、5 行目と 18 行目では、このリテラルの否定の和に部分回路の中間変数のリテラル $\overline{y}$ を追加出力し、8 行目と 21 行目では中間変数のリテラル y を追加出力している。15 行目では、$f_{x_i=0}$ と $f_{x_i=1}$ が同じ BDD なら、16 行目以降の処理を省いている。

```
PathbasedBDDtoCNFTrans (G(V, E))
        G(V, E): ブーリアン・ネットワーク
    SubCircBDDs: BDD の集合
              f: BDD
         path[]: リテラルを格納する配列
begin
 1: SubCircBDDs = ∪ 中間変数の導入されたノードの BDD;
 2: foreach f ∈ SubCircBDDs
 3:   PathbasedBDDtoCNFTrans_sub (f, path[], x0,
                                  f の部分回路の根に導入された中間変数);
end

PathbasedBDDtoCNFTrans_sub (f, path[], xi, y)
   f, f0, f1: BDD
     path[]: BDD の根節点から定数節点へのパス、リテラルの配列
         xi: BDD のレベル i の変数
          y: 部分回路の根に導入された中間変数
 begin
 1: f0 = f_{xi=0};     // f の変数 xi に 0 を代入
 2: path[i] = BDD の変数 x̄i に対応するリテラル;
 3: if (f0 が 0 節点) {
 4:   path[0] から path[i] の各リテラルの否定の和 ∨ を出力;
 5:   更に ȳ も追加で和をとり出力;
 6: } else if (f0 が 1 節点) ){
 7:   path[0] から path[i] の各リテラルの否定の和 ∨ を出力;
 8:   更に y も追加で和をとり出力;
 9: } else {
10:     PathbasedBDDtoCNFTrans (f0, path[], xi+1, y)
11: }
12:
13: f1 = f_{xi=1};     // f の変数 xi に 1 を代入
14: path[i] = BDD の変数の否定 x̄i に対応するリテラル;
15: if (f0 == f1) { return; }
16: if (f1 が 0 節点) {
17:   path[0] から path[i] の各リテラルの否定の和 ∨ を出力;
18:   更に ȳ も追加で和をとり出力;
19: } else if (f1 が 1 節点) ){
20:   path[0] から path[i] の各リテラルの否定の和 ∨ を出力;
21:   更に y も追加で和をとり出力;
22: } else {
23:   PathbasedBDDtoCNFTrans (f1, path[], xi+1, y)
24: }
end
```

図 8 BDD から CNF 式への変換アルゴリズム

5. 実験と結果

提案フレームワークに基づく CNF 式生成手法と、2.3 節の CNF 式生成手法を実装し SAT-solver の実行時間について実験を行った。実験では、これら二つの手法によって生成される CNF 式を SAT-solver に与え、その実行時間を比較した。実験は、順序回路の sequential depth 計算 [6] における CNF 式生成と、組み合わせ回路の等価性判定における CNF 式生成において行った。なお、実験は Core 2 Extrem 2.93GHz, メモリ 4 GB の計算機上で行った。また、SAT-solver として、Davis-Putnam method [11] に基づいた SAT-solver である chaff-2004.11.15 [7] と miniSAT2-061208 [4], [8] を使用した。BDD による論理関数処理の実現には、BDD パッケージ CUDD [12] を使用した。

5.1 順序回路の sequential depth 計算における比較

順序回路の sequential depth は、順序回路の初期状態から最も遠い状態までの深さを表すものである [6]。CNF 式生成処理に、提案手法を用いた sequential depth 計算プログラムと、2.3 節の CNF 式生成手法を用いた sequential depth 計算プログラムを作成し実験を行った。本実験では、SAT-solver として、chaff-2004.11.15 [7] を使用した。

10-bit カウンタの sequential-depth 計算が、深さ 400 に達するまでの実行時間は、表 1 のようになった。表 1 の CNF 式生成手法は、CNF 式の生成に使用した手法を表す。LevelThre は、提案手法で中間変数の間隔を考慮した回路分割を行う際に使用した閾値 LevelThre の値である。Time は sequential-depth 計算に要した時間を表し、CNF 式の生成に要する時間も含まれている。時間変化率は、“提案手法を用いた時の Time”/“2.3 節の手法を用いた時の Time” である。この結果から、提案手法

表 1 10-bit カウンタ の sequential depth 計算への適用結果

CNF 式生成手法	LevelThre	Time(s)	時間変化率
2.3 節の手法	—	148819	—
提案手法	8	142431	0.96
	12	43167	0.29
	16	38273	0.26

表 2 2.3 節の CNF 式生成法を用いた組み合わせ回路の等価性判定における SAT-solver の実行時間

等価性判定の対象の出力信号線名	2.3 節の CNF 式生成法			
	CNF 式			miniSAT
	#var	#lit	#cl	Time (s)
out[9]	1667	9073	4077	3.7
out[10]	1997	10906	4894	12.6
out[11]	2358	12952	5800	64.6
out[12]	2766	15255	6819	261.5
out[13]	3192	17657	7885	604.1
out[14]	3655	20278	9044	3364.9
out[15]	4142	23034	10266	23543.3
out[16]	4664	25992	11574	43889.3

において LevelThre の値を変化させることにより、時間変化率が変化することが分かる。また、提案手法で LevelThre を 16 とした場合、2.3 節の手法を用いた場合と比較して sequential depth 計算要する時間が 26% に短縮できた。

5.2 組合せ回路の等価性判定における比較

CNF 式生成処理に、提案手法を用いた等価性判定プログラムと、2.3 節の CNF 式生成手法を用いた等価性判定プログラムを作成し実験を行った。本実験では、SAT-solver として、2.4 節の 図 3 のような構成で、与えられた CNF 式を最適化する機能を持った miniSAT2-061208 [4], [8] を使用した。また、提案手法における LevelThre の値は、いくつかの値を試し、最も結果の良かった 2 とした。

2.3 節の CNF 式生成手法と提案手法を用い、32-bit Wallace 乗算器と 32-bit 配列型乗算器の等価性判定を乗算器の 9-bit 目から 16-bit 目までの各出力について行ったところ、結果はそれぞれ、表 2、表 3 のようになった。#var, #lit, #cl は、それぞれ CNF 式中の変数の数、リテラルの数、節の数を表す。miniSAT Time は、等価性判定に要した時間である。この時間に CNF 式生成の時間は含まれていないが、両手法ともに 1 秒以下で論理回路から CNF 式の生成が可能である。また、表 3 の時間変化率は、"表 3 の miniSAT Time" /"表 2 の miniSAT Time" を表す。この結果から、提案手法により、2.3 節の手法と比較して、変数の数、リテラルの数、節の数の少ない CNF 式が生成されていることが分かる。また、提案手法では、2.3 節の手法を用いた場合と比較して、等価性判定に要する時間が out[11], out[12], out[14], out[15], out[16] において、それぞれ 49%, 99%, 18%, 13%, 72% に短縮できた。

6. むすび

SAT を用いた形式的検証の高速化のための CNF 式生成生成処理フレームワークを提案し、提案フレームワークに基づく、CNF 式生成手法を示した。既存の CNF 式生成手法との比較を行ったところ、提案手法により SAT-solver の実行時間が短縮されることが分かった。今後、提案フレームワークの各処理のアルゴリズムの改良を行う予定である。

表 3 提案する CNF 式生成法を用いた組み合わせ回路の等価性判定における SAT-solver の実行時間

等価性判定対象の出力信号線名	提案する CNF 式生成手法 (LevelThre=2)				
	CNF 式			minSAT	
	#var	#lit	#cl	Time (s)	時間変化率
out[9]	947	8312	2891	3.7	1.0
out[10]	1126	10085	3473	14.3	1.13
out[11]	1296	11745	4041	31.5	0.49
out[12]	1525	13752	4749	259.6	0.99
out[13]	1777	16170	5558	779.1	1.29
out[14]	2053	18750	6432	614.9	0.18
out[15]	2332	21476	7347	3097.6	0.13
out[16]	2624	24188	8279	31449.8	0.72

謝辞 日頃から御討論いただく、名古屋大学大学院報科学研究科 川島裕崇さんをはじめ高木研究室の皆様に感謝します。なお、本研究は、一部文部省科学研究費補助金 18700043 若手研究 (B) による。

文 献

[1] 藤田昌宏, "SAT アルゴリズムとその形式的検証への応用," 電子情報通信学会技術報告書 (VLD2006-67) , pp. 15-20, 2006.
[2] M.N. Velev, "Efficient Translation of Boolean Formulas to CNF in Formal Verification of Microprocessors," in Proc. of Asia and South Pacific Design Automation Conference(ASP-DAC'04), 2004.
[3] M.N. Velev, "Exploiting Signal Unobservability for Efficient Translation to CNF in Formal Verification of microprocessors," Proc. of Design Automation and Test in Europ(DATE'04), 2004.
[4] Niklas Eén and Armin Biere, "Effective Preprocessing in SAT through Variable and Clause Elimination," Proc. of SAT'05, 2005.
[5] D. Plaisted and S. Greenbaum, "A Structure-Preserving Clause Form Translation," *Journal of Symbolic Computation 2*, pp. 293-304, 1986.
[6] Maheer Mneimneh and Karem Sakallah, "SAT-based Sequential Depth Computation," Proc. of ASP-DAC'03, 2003.
[7] M.W. Moskewicz, C.F. Madigan, Y.Zhao, L. Zhang and S. Malik, "Chaff: Engineering an Efficient SAT Solver," Proc. of the 38th Design Automation Conference(DAC'01), 2001.
[8] Niklas Eén and Niklas Sörensson, "An Extensible SAT-solver," Proc. of SAT'03, 2003.
[9] R. K. Brayton, R. L. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang, "MIS: a multiple-level logic optimization system," *IEEE Trans. on CAD*, CAD-6, no.6, pp. 1062–1081, 1987.
[10] 湊真一, "計算機上での BDD の処理技法," 情報処理, Vol. 34, No. 5, pp. 593-599, 1993.
[11] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," Journal of the Association for Computing Machinery, 7, pp. 201-215, 1960.
[12] Fabio Somenzi. Cudd: Cu decision diagram package release 2.4.1.