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

2006－ＳＬＤＭ－123（９）
2006／1／17

第一階述語論理のサブクラスを利用したブール関数レベルの等価性判定手法

森友 淳史† 浜口 清治† 柏原 敏伸†

† 大阪大学大学院 情報科学研究科 〒560-8531 大阪府豊中市待兼山町 1-3
E-mail: †{moritomo,hama,kashi}@ics.es.osaka-u.ac.jp

あらまし 近年の半導体回路の大規模化，複雑化に伴い，その設計検証にはより一層の時間と資源が必要となっている．2つの設計間での等価性判定は，典型的な設計検証問題の1つである．設計検証では一般にシミュレーションが主流だが，等価性判定に関しては形式的検証手法の利用が普及しつつある．ブール関数レベルでの形式的な等価性判定を考えると，その検証には膨大な時間と資源がかかる．そこで，第一階述語論理を利用し検証コストを削減する手法が考案されている．特に第一階述語論理のサブクラスである限量子を含まない等号付第一階述語論理での恒真性判定は決定可能であることから，これを用いた自動検証が可能となる．しかし，第一階述語論理では，算術演算などは関数記号の形で抽象化されるため，ブール関数レベルでの等価性判定と同じ結果を与えることが一般にできない．そこで，本報告では可能な部分は限量子を含まない等号付第一階述語論理を利用して検証を行い，そうでない部分のみをブール式へと変換することで検証コストの削減を図る等価性判定アルゴリズムについて述べる．

キーワード 等価性判定，ブール関数，第一階述語論理

Boolean Equivalence Checking Using a Subset of First-Order Logic

Atsushi MORITOMO†, Kiyoharu HAMAGUCHI†, and Toshinobu KASHIWABARA†

† Dept. of Bioinformatic Engineering, Graduate School of Infomation Science and Technology, Osaka University Machikaneyama 1-3, Toyonaka-shi, Osaka, 560-8531 Japan
E-mail: †{moritomo,hama,kashi}@ics.es.osaka-u.ac.jp

Abstract In recent years, design verification becomes more difficult as the scale of semiconductor circuits becomes larger. Designs are commonly checked by simulation-based verification, and exhaustive verification is difficult by simulation-based verification. As for equivalence checking, however, formal verification methods have been spreading. Formal equivalence checking at Boolean function level requires a large amount of cost. Therefore, formal verification based on first-order logic has been considered. Since validity checking of quantifier-free first-order logic with equality which is a subclass of first-order logic is decidable, the verification methods using this logic has been proposed. In this logic, however, arithmetic operations are abstracted away by functional symbols. The result of equivalence checking with this logic can differ from that of the conventional Boolean equivalence checking. This report shows an equivalence checking algorithm of the logic formula using Boolean substitution only for function/predicate symbols which cannot be checked by first-order logic. This algorithm achieves the accuracy of equivalence checking at Boolean level.

Key words Equivalence Checking, Boolean Function, First-Order Logic

1. まえがき

近年の半導体回路の大規模化，複雑化に伴い，その設計と検証にはより一層の時間と資源が必要となっている．回路設計において，十分な検証により正しさが保証されたゴールデンモデルと呼ばれる設計に誤りが混入するのを防ぐため，等価性判定が行われる．設計検証では一般にシミュレーションが主流だが，等価性判定に関しては形式的検証手法の利用が普及しつつある．

ブール関数レベルでの形式的な等価性判定を行う場合，設計の記述量が膨大になり，その検証には多くの時間と資源が必要である．そこで，第一階述語論理を利用し検証コストを削減する手法が考案されている．特に第一階述語論理のサブクラスで

ある限量子を含まない等号付第一階述語論理での恒真性判定は決定可能であることから，これを用いた検証手法が考案されている[3]~[7]．しかし，第一階述語論理では，算術演算などは関数記号の形で抽象化されるため，ブール関数レベルでの等価性判定と同じ結果を与えることは一般にできない．

文献[9]では，可能な部分は第一階述語論理を利用して検証を行い，そうでない部分のみをブール式へと変換する手法を考案した．これにより，ブール関数レベルでは等価であるが，第一階述語論理では等価とみなせなかった論理式の等価性判定が可能となり，設計全体をブール式へと変換する場合に比べてコストの削減が可能だと考えられる．ブール式による意味付けを組み合わせる手法は文献[10]においても，動作合成前後の動作記述とRTL記述の等価性判定に用いられている．この手法では，多数のインスタンスに対して，等価性判定を行うことになるが，関数記号や述語記号を抽象化したままでは扱えないインスタンスについては，そのインスタンス全体をブール式に置換して等価性を判定する．

本論文で提案する手法では，反例を利用し，さらにそれを構造的に分析するアルゴリズムを用いることで，部分的にブール関数に置き換えて判定する．これにより，インスタンス全体をブール関数で置換する必要がない場合には，より効率的な検証が可能になると期待することができる．

以下本報告では，2節では限量子を含まない等号付第一階述語論理について説明し，3節で反例部分をブール式へと置換する等価性判定アルゴリズムを示し，4節では反例を構造的に分析し，ブール式へと変換する部分を削減するアルゴリズムを示す．

2. 限量子を含まない等号付第一階述語論理

限量子を含まない等号付第一階述語論理とは，一般の第一階述語論理から限量子を除いたもので，特別な記号として等号を含む．限量子を含まない等号付第一階述語論理の構文は項と論理式からなる．

項は次のように再帰的に定義される．まず，変数x,y,z，そして，定数a,b,cは項である．また，fがn個の引数を持つ関数記号で，$t_1,t_2,...,t_n$が項ならば，$f(t_1,t_2,...,t_n)$も項である．さらに，αを論理式，t_1,t_2を項とする．このとき，$ITE(\alpha,t_1,t_2)$は項である．ここで，$ITE(\alpha,t_1,t_2)$は*if-then-else*を表し，αが真のときにはt_1を，αが偽の時にはt_2を表す項である．

論理式は次のように再帰的に定義される．まず，*true,false*は論理式である．また，t_1,t_2が項ならば，等式$t_1=t_2$は論理式である．さらに，pがn個の引数を持つ述語記号で，$t_1,t_2,...,t_n$が項ならば，$p(t_1,t_2,...,t_n)$も論理式である．

論理式に，空でない領域Dと，関数記号や述語記号の解釈Iが与えられると，その真偽が次のように決定する．解釈Iにより，引数をk個持つ関数記号には，$D^k \rightarrow D$が割り当てられ，引数をk個持つ述語記号には，$D^k \rightarrow \{true,false\}$が割り当てられる．解釈$I$は，それぞれの変数，定数に$D$の要素を割り当てる．定数に対する解釈$I$は，異なる定数に割り当てられた値がそれぞれ異なるようにDの要素を割り当てる．解釈Iにより，ブール変数は$\{ true,false \}$に割り当てられる．

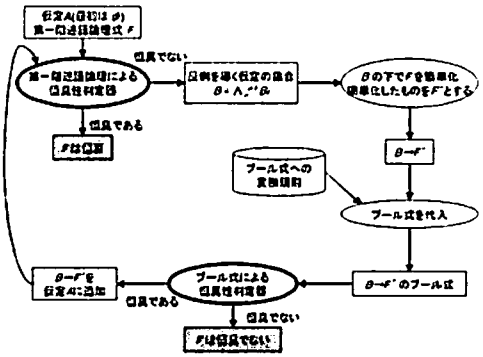

図1 提案アルゴリズム概要

値の評価は次のように行われる．項$t=f(t_1,t_2,...,t_n)$に対して解釈Iが与えられると，$I(t)=I(f)(I(t_1),I(t_2),...,I(t_n))$となる．$ITE(\alpha,t_1,t_2)$項に対しては，$I(\alpha)$が*true*の時は$I(t)=I(t_1)$となり，*false*の時は$I(t)=I(t_2)$となる．論理式$\alpha=p(t_1,t_2,...,t_n)$に対して，$I(\alpha)=I(p)(I(t_1),I(t_2),...,I(t_n))$となる．等式$\alpha=(t_1=t_2)$に対して，$I(t_1)=I(t_2)$の時かつその時のみ$I(\alpha)=true$となる．ブール演算子$\circ$に対して，$\alpha=\alpha_1\circ\alpha_2$は$I(\alpha)=I(\alpha_1)\circ I(\alpha_2)$となる．

恒真性と充足可能性について次のように定義される．論理式αが恒真であるとは，任意の領域Dと解釈Iに対して，$I(\alpha)$が真であることを意味する．また，論理式αが充足可能とは，ある解釈I，領域Dに対して$I(\alpha)$が真であるような解釈Iが存在することを意味する．

以下，本論文では限量子を含まない等号付第一階述語論理を簡単のため単に第一階述語論理と呼ぶ．

3. 第一階述語論理を利用した等価性判定アルゴリズム

以下，3節で基本アルゴリズムを述べ，4節で，その改良手法を示す．

3.1 等価性判定と恒真性判定

ここでは，2つの設計記述の等価性を恒真性判定を使って判定する．正しいことが保証されている設計記述と，それに改良を加えた設計記述を用意し，それぞれの設計記述を，その出力を表す第一階述語論理の項へと変換し，その2つを等号で結ぶ．2つの設計記述の出力が等価であるならば，この等号は常に成り立つので，この式は恒真となる．こうして得られた式の恒真性を判定することで2つの設計記述の等価性を調べる．算術演算などは関数記号で表現されており，これらの関数記号には，ブール関数レベルでの意味が与えられていると仮定している．最終的に証明したいのは，ブール関数レベルの等価性である．

3.2 アルゴリズムの概要

例として，$ITE((x<y),x,y)=ITE(((x-y)<0),x,y)$について考える．述語記号$p_<$を算術比較の$<$を表す記号，関数記号$f_-$を算術演算の$-$を表す記号とすると，この

論理式は $ITE(p_<(x,y),x,y) = ITE(p_<(f_-(x,y),0),x,y)$ で表される．この論理式を第一階述語論理式の恒真性判定手法を利用して恒真性を判定すると，恒真ではないという結果になる．このとき，上述の論理式を偽へと導いた条件は $p_<(x,y)$ と $\neg p_<(f_-(x,y),0)$ である．この2つの条件 $p_<(x,y)$, $\neg p_<(f_-(x,y),0)$ を仮定して，上述の論理式を変形すると $(x = y)$ という式が導き出される．$p_<, f_-, x, y$ の任意の解釈に対し，$p_<(x,y) \wedge \neg p_<(f_-(x,y),0) \Rightarrow (x = y)$ は真とは限らないので，$ITE((x < y),x,y) = ITE(((x - y) < 0),x,y)$ は恒真とは判定されない．

提案するアルゴリズムでは，ここでブール式による解釈を与えて恒真性判定を行う．$p_<$ に不等号，f_- に減算としての意味をそれぞれ与えると $p_<(x,y) \wedge \neg p_<(f_-(x,y),0) \Rightarrow (x = y)$ は恒真となる．$p_<(x,y) \wedge \neg p_<(f_-(x,y),0) \Rightarrow (x = y)$ を恒真である式として仮定した上で，元の論理式について，再度，第一階述語論理式の恒真性判定を行う．このような操作を論理式の恒真性が判定されるまで繰り返す．図1にアルゴリズムの概要を示す．

このようにして，第一階述語論理による恒真性判定では等価であるとみなされなかった論理式を，ブール式に置き換えることで等価性を導き出すことができる．以下，詳しいアルゴリズムを示す．

3.3 恒真性判定アルゴリズム

このアルゴリズムでは，第一階述語論理式の恒真性判定手法とブール式の恒真性判定手法を利用することを仮定している．以下に示すアルゴリズム中では，手続き *Check_V* で第一階述語論理式の恒真性判定手法を，手続き *Check_TF* でブール式の恒真性判定手法を使用している．図1はアルゴリズムの全体的な構成である．入力として，仮定 $A = \{$ 仮定 A_1, 仮定 A_2, ..., 仮定 $A_n\}$ と第一階述語論理式 F をとり，仮定 A の下で論理式が恒真であるか，つまり $\bigwedge_{i=1}^{n} A_i \Rightarrow F$ が恒真であるかを判定する．仮定 A の初期値は空集合 ϕ である．

本論文で提案するアルゴリズムは図2に示す手続き *Validity_Checker* である．この手続きは，与えられた論理式 F と仮定 A に対して第一階述語論理による恒真性判定を行い(2行目)，その結果が恒真ならば入力論理式は恒真であるという結果で終了し，そうでない場合は以下の操作を行う．まず，手続き *Check_V* から返された仮定 $B = \{B_1, B_2, ..., B_m\}$ の下で入力論理式を変形する(4行目)．この変形は仮定 $B = \{B_1, B_2, ..., B_m\}$ の下で入力論理式 F を簡単化し，最終的に等価と判定できなかった式を導出する．この変形の結果得られた論理式を F' とする．次に，$\bigwedge_{i=1}^{m} B_i \Rightarrow F'$ をブール式を使って変換し，その恒真性を判定する(5行目)．このとき，関数記号と述語記号をそれぞれブール式のベクトルとブール式へ対応付けする写像が h である．これが恒真であるならば仮定 A に新たな仮定として $\bigwedge_{i=1}^{m} B_i \Rightarrow F'$ を追加し(6行目)，恒真性判定を繰り返す．そうでないならば出力された仮定は真の反例になり，入力論理式は恒真ではないと判定することができる(7行目)．以上の操作を繰り返すのが手続き *Validity_Checker* である．

```
入力 A : { 仮定 A_1, 仮定 A_2, ..., 仮定 A_n },
     F : 論理式, h : S → β
     ここで S は F 中で使用されている関数記号及び
     述語記号の集合，β はブール式のベクトルの集合
出力 h の下での ⋀_i A_i ⇒ F の恒真性

1  Validity_Checker(A, F, h){
2    while((B := Check_V(A, F)) ≠ "VALID")
3    do begin
4      F' := Transform(F, B);
5      if(Check_TF(B, F', h) ≠ "VALID")
6      then A に ⋀_i B_i ⇒ F' を追加;
7      else return("invalid");
8    end;
9    return("valid");
10 }
```

図2 手続き *Validity_Checker*

```
入力 B : { 仮定 B_1, 仮定 B_2, ..., 仮定 B_m },
     TF : 論理式, h : S → β
出力 ⋀_{i=1}^{m} B_i ⇒ TF をブール式に変換し，恒真性判定を行った
     結果が恒真ならば VALID，そうでないならば INVALID

1  Check_TF(B, TF, h){
2    CTF := Boolean(⋀_{i=1}^{m} B_i ⇒ TF, h);
3    /* ⋀_{i=1}^{m} B_i ⇒ TF をブール式に変換 */
4    if(Bool_Checker(CTF) = "VALID")
5    then return("VALID");
6    else return("INVALID");
7  }
```

図3 手続き *Check_TF*

図2中の手続き *Check_V* は論理式 F について恒真性判定を行う．ここでは第一階述語論理に対する恒真性判定を行っている．恒真であるならば *VALID* を，恒真でないならば入力論理式 F を偽へと導いた条件 $B = \{B_1, B_2, ..., B_m\}$ を返す手続きである．

図2中の手続き *Transform* は入力論理式を偽へと導いた条件 $B = \{B_1, B_2, ..., B_m\}$ の下で論理式 F を変形し，その結果を返す手続きである．

図3で示される手続き *Check_TF* は，$\bigwedge_{i=1}^{m} B_i \Rightarrow TF$ を，手続き *Boolean* によってブール式のベクトルへと変換し(2行目)恒真性判定を行う．ここで恒真性判定を行っている手続き *Bool_Checker*(4行目)は，ブール式の恒真性判定を行う．恒真

の場合は *VALID* を返し (5 行目)，恒真でなければ *INVALID* を返す (6 行目)．

手続き *Boolean* は第一階述語論理をブール式のベクトルに変換する．以下，$t_1, t_2, ..., t_n$ を項とし，f を関数記号，p を述語記号，α を論理式とする．項は s ビットのブール式に，述語はブール式にそれぞれ変換される．β をブール式のベクトルの集合，F を論理式の集合としたとき，以下の写像 $H : F \rightarrow \beta$ が変換手続きである．

[手続き]　ブール式への変換

$H(F)$ は次のように再帰的に定義される．各関数記号，述語記号には対応するブール式のベクトル及び，ブール式がそれぞれ与えられているとする．

- F が定数ならば，$H(F) = (c_1, c_2, ..., c_s)$
  ここで，$c_i \in \{true, false\}$
- F が変数ならば，$H(F) = (b_1, b_2, ..., b_s)$
  ここで，b_i はブール変数
- F が $f(t_1, t_2, ..., t_n)$ ならば，
  $H(F) = h(f)\big(H(t_1), H(t_2), ..., H(t_n)\big)$
- F が $p(t_1, t_2, ..., t_n)$ ならば，
  $H(F) = h(p)\big(H(t_1), H(t_2), ..., H(t_n)\big)$
- F が $t_1 = t_2$ ならば，$H(F) = \big(H(t_1) = H(t_2)\big)$
- F が $F_1 \circ F_2$ ならば，$H(F) = H(F_1) \circ H(F_2)$
- F が $\neg F_1$ ならば，$H(F) = \neg H(F_1)$
- F が $ITE(\alpha, t_1, t_2)$ ならば，
  $H(F) = \big(H(\alpha) \wedge H(t_1)\big) \vee \big(\neg H(\alpha) \wedge H(t_2)\big)$

□

4. 変換部分削減アルゴリズム

前節のアルゴリズムでは，反例を導く仮定 B から導き出された反例 F' をそのまま利用した．しかし，F' 全体をブール式に変換すると，ブール式に変換する必要のない部分まで変換しなければならない場合がある．F' 中の構造的に異なる部分のみをブール式に変換することで，無駄なブール式への変換を削減できる可能性がある．図 4 は等価性判定アルゴリズムの全体図である．図 4 中の '追加処理' が新たに加える処理であり，図 5 はその追加処理アルゴリズムの概要図である．'追加処理' アルゴリズムは，$F' = (F'A = F'B)$ の形の式の $F'A$ と $F'B$ に対して，$F'A$ と $F'B$ を構造的に比較し，対応しているが構造的に異なっている部分式の対を取り出す．その後、取り出した部分式の対をブール式へと変換しその 2 つが等価であるかどうか等価性判定を行う．取り出された全ての部分式の対が等価であれば，$F'A = F'B$ は恒真であるといえる．'追加処理' により恒真であると判定できた場合には，通常の場合と比べブール式へ変換された部分は少ないと期待できる．恒真でないと判定したときは通常の手法で処理を続ける．

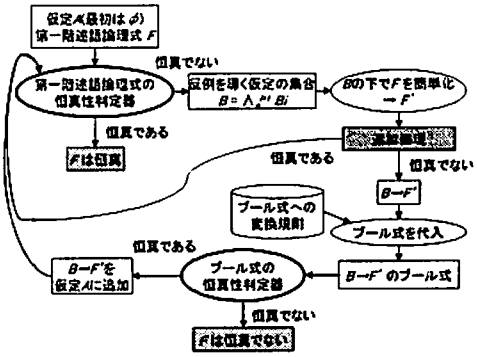

図 4　提案アルゴリズム全体図

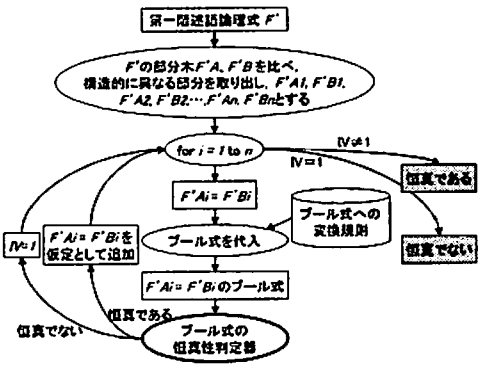

図 5　部分変換アルゴリズム

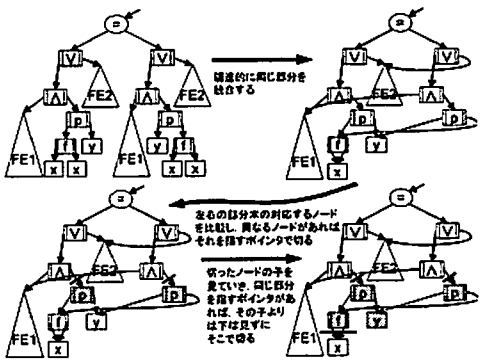

図 6　変換部分の探索

図 6 は提案する追加処理アルゴリズムの中で，構造的に異なる部分を取り出す処理の流れを示しており，図 7 の手続き *Difference* がこの処理を示している．図 7 中の手続き *unique* は，式 F' 中の構造的に同じ部分を統合する手続きである．$FE1$, $FE2$ が論理式を表すとし，F' が $(FE1 \wedge p(f(x,x),y)) \vee FE2 = (FE1 \wedge p(y, f(x,x))) \vee FE2$ という式である場合を考える．この式を表す構文木は図 6 のようになり，それを葉から探索していき，構造的に同じ部分木があれば一つに統合しポインタを付

```
入力 F' : 論理式
出力 D : {F'A_1 = F'B_1, F'A_2 = F'A_2, ..., F'A_m = F'B_m}

1 Difference(F'){
2   F' := unique(F'); /* 構造的に同じ部分を統合する. */
3   D := Divide(F' → arg_1, F' → arg_2, φ);
4   return(D);
5 }
```

図 7　手続き *Difference*

```
入力 F'A : F'の左の部分木, F'B : F'の右の部分木,
     D : {F'A_1 = F'B_1, F'A_2 = F'A_2, ..., F'A_m = F'B_m}
出力 D : {F'A_1 = F'B_1, F'A_2 = F'A_2, ..., F'A_m' = F'B_m'}

1  Divide(F'A, F'B, D){
2    if((F'A = NULL) ∧ (F'B = NULL))
3    then 何もしない;
4    else if((F'A → sym = F'B → sym)
5          ∧(⋀_{i=1}^{n} F'A → arg_i = F'B → arg_i))
6    then 何もしない;
7    else if((F'A → sym = F'B → sym)
8          ∧(⋀_{i=1}^{n} F'A → arg_i → sym = F'B → arg_i → sym))
9    then for(i := 1; i ≤ n; i++)
10        D := Divide(F'A → arg_i, F'B → arg_i);
11   else begin
12     F'A を根とする木の全ノードに対し check_L := 1;
13     F'B を根とする木の全ノードに対し check_R := 1;
14     Replace(F'A);
15     D に F'A = F'B を追加する
16   end;
17   return(D);
18 }
```

図 8　手続き *Divide*

```
入力 F : 論理式

1 Replace(F){
2   if(F = NULL) then return;
3   else if((F → check_L = 1) ∧ (F → check_R = 1))
4   then F を適当な新しい変数で置き換える;
5   else for(i := 1; i ≤ n; i++) Replace(F → arg_i);
6 }
```

図 9　手続き *Replace*

け替える．その結果，図 6 右上にあるような構文木が得られる．

その後，この構文木の根についての左右の部分木に対して，図 8 に示す手続き *Divide* を実行する．手続き *Divide* は左右の部分木を探索していき，左右で対応しているノードを比較して，構造的に異なる部分を探す手続きである．図 8 中の sym はそのノードのシンボルを表し，arg_i はそのノードが持つ i 番目の引数へのポインタである．現在調べている 2 つのノードのシンボルが等しく，かつ，全ての引数ノードのシンボルが等しいとき，その 2 つのノードは構造的に等しいとみなす．構造的に異なるノードが見つかると，そのノードを指しているポインタを区切りの位置とする．その結果，図 6 左下のような構文木が得られる．図 8 中 16 行目に示すように，構造的に異なるノードが見つかると，左部分木の区切りの位置を指すポインタは $F'A$，右部分木の区切りの位置を指すポインタは $F'B$ となっている．$F'A$ が指すノードを根とする部分木の全てのノードに対して，*check_L* の値を 1 にする．同様に，$F'B$ が指すノードを根とする部分木の全てのノードに対して，*check_R* の値を 1 にする．これにより，*check_L* と *check_R* 両方の値が 1 であるノードは左右の部分木で共有されているノードであるということがわかる．共有されている部分木は等価となるので，適当な新しい変数で置き換え，ブール式へ変換する部分を削減する．

その共有部分を適当な新しい変数で置き換える手続きが図 9 に示す *Replace* である．この手続きは *check_L* と *check_R* 両方の値が 1 であるノードを探索し，そのノードを適当な新しい変数で置き換える手続きである．その結果，図 6 右下のように共有する部分は省略することが可能となる．これにより，$F'A$，$F'B$ が構造的に異なる部分として取り出されるので，$F'A = F'B$ をブール式へと変換する部分として決定する．図 6 右下の場合，取り出される部分式は $p(var\_f, y)$ と $p(y, var\_f)$ になる．var_f が新たに導入した変数である．よって，$p(var\_f, y) = p(y, var\_f)$ をブール式へと変換し，ブール式の恒真性判定を行うことになる．

5. まとめと今後の課題

本報告では，ブール関数レベルの等価性判定を第一階述語論理の反例を利用し，効率的に行う手法を提案した．さらに反例の構造を分析することにより，ブール式へと変換する部分を削減するアルゴリズムも示し，更なる効率化を図った．

今後の課題としては，本報告で提案したアルゴリズムの実装と実験が挙げられる．さらに，加算などの結合則が成り立つような演算子に対しては，木構造を変形させるなどの処理を施すことで，よりいっそうの効率化を目指す．

文　献

[1] G. Nelson, D. C. Oppen, "Simplification by Cooperating Decision Procedure", ACM Trans on Programming Languages and Systems, 1(2):pp.245-257, 1979.

[2] R. E. Shostak, "A Practical Decision Procedure for Arithmetic with Function Symbols", Journal of ACM, 26(2):pp351-360, 1979.

[3] J. R. Burch, and D. L. Dill, "Automated Verification of Pipelined Microprocessor Control", CAV, pp.68-80, 1994.

[4] R. B. Jones, D. L. Dill, J. R. Burch, "Efficient Validity Checking for Processor Verification", ICCAD, pp2-6, 1995.

[5] Randal E. Bryant,Steven M. German,Miroslav N. Velev, "Processor verification using efficient reductions of the logic of uninterpreted functions to propositional logic", CAV, pp.93-134, 1999.

[6] D. W. Currie, A. Hu, S. Rajan, M. Fujita, "Automatic Formal Verification of DSP software", DAC 2000, pp.130-135, 2000.

[7] Kiyoharu Hamaguchi, Hidekazu Urushihara, Toshinobu Kashiwabara, "Verifying Signal-Transition Consistency of High-Level Designs Based on Symbolic Simulation", IEICE, E85-D, pp.1587-1594, 2002.
[8] Kiyoharu Hamaguchi, "Symbolic Simulation Heuristics for High-Level Hardware Descriptions Including Uninterpreted Functions", IEICE, E87-D, no.3, pp.637-641, 2004.
[9] Atsushi Moritomo, Kiyoharu hamaguchi, Toshinobu Kashiwabara, "Validity Checking for Quantifier-Free First-Order Logic with Equality Using Substitution of Boolean Formulas", ATVA 2004, pp.108-119, 2004.
[10] 竹中崇, 中田勝, 向山輝, 前川晃, 若林一敏, 山際雄, "動作合成前後の動作記述と RTL 記述の論理等価性検証", 第 17 回回路とシステム軽井沢ワークショップ, pp.555-560, 2004.
[11] CVC Lite, http://verify.stanford.edu/CVCL/
[12] zChaff, http://www.princeton.edu/ chaff/zchaff.html