（1991．12．13）

Boolean Unificationの組合せ論理合成への応用

久木元 裕治　　藤田 昌宏
東京大学工学部　　富士通研究所

Boolean unificationとは、与えられたブール方程式の最汎解を与える手続きで、求まった最汎解は、方程式を満たす解の自由度を完全に表現できる。本稿では、再設計、多段論理簡単化、Boolean relationの簡単化などの組合せ論理合成のさまざまな問題が、Boolean unificationを用いることによって統一的に扱えることを示す。

Application of Boolean Unification to Combinational Logic Synthesis

Yuji Kukimoto　　Masahiro Fujita
University of Tokyo　　FUJITSU LABORATORIES LTD.

Boolean unification is an algorithm to obtain the general solution of a given Boolean equation. Since the general solution provides a way to represent complete don't care sets in a functional form, Boolean unification can be a powerful method when applied to logic synthesis. In this paper we present various applications of Boolean unification to combinational logic synthesis. Three topics of combinational logic synthesis: redesign, multi-level logic minimization and minimization of Boolean relations are discussed. All these problems can be uniformly formalized as Boolean unification problems. Experimental results are also reported.

1 はじめに

Boolean unification(ブール単一化) [10, 9, 3] は、与えられたブール方程式を満たす最汎解 (most general unifier) を求める手続きで、LSI-CAD の分野では、制約論理型言語 CHIP [5] の応用という形でさまざまな研究が行なわれてきている。CHIP はブール制約解消系を Boolean unification を用いて実装することによって、ブール式で与えられた制約を能動的に解く機構を持っている。これまでに、論理検証 [13] やテスト・パターン自動合成 [12] への適用が報告されている。

本稿では、Boolean unification を組合せ論理合成へ適用する手法を提案する。論理合成の多くの問題は、設計仕様が入力と出力に関するブール制約式で表現されるので、この制約式に Boolean unification の手続きを適用することによって、仕様を満たす出力関数を入力変数の関数として求めることができる。また、この際、制約を満たす設計のすべてが最汎解という形で表現されるため、与えられた仕様の自由度を完全にとらえることができる。言いかえると、Boolean unification の結果として得られる最汎解は、完全な Don't care set を表現していると見なせる。論理合成においては、仕様を満たす設計の中から、コストの小さいものを選択することが不可欠なので、設計空間を完全に把握することは高品質な合成結果を得る上で重要な意味を持つ。

ここでは以上の考えに基づき、再設計、多段論理簡単化、Boolean Relations の簡単化の 3 つの問題に対して、Boolean unification を用いた形式化手法を示す。これらの問題は、Boolean unification を適用することによって、すべて統一的に扱えることが示される。

本稿では、まず第 2 章で Boolean unification について概説する。第 3 章では、組合せ論理合成への応用例と形式化手法について述べ、第 4 章で最汎解から効率的な回路を設計する手法を提案する。第 5 章では、Boolean relation の多段簡単化について提案した手法を適用した実験結果を示し、第 6 章で結論が述べられる。

2 Boolean unification(ブール単一化)

2.1 概要

Boolean unification [10, 9, 3] は、与えられたブール方程式を満たす最汎解を求めるアルゴリズムである。アルゴリズムとしては、Boole 自身が提案したものなどいくつかが知られているが、近年ブール制約を扱う制約論理型言語の制約解消系の実現に用いられ注目されている。

いま、式 (1) が与えられたとする。ここで、$p_1, p_2, \ldots, p_n$ はブール変数、$q_1, \ldots, q_m$ はブール定数である。

$$f(p_1, p_2, \ldots, p_n, q_1, q_2, \ldots, q_m) = 0 \qquad (1)$$

Boolean unification とは、式 (1) を変数 p_i について解くアルゴリズムである[1]。ここで、式 (1) に、Boolean unification を適用すると、ブール変数 $p_1, \ldots, p_n$ の最汎解が求まる。最汎解は、一般的には $q_1, q_2, \ldots, q_m$ と新しく導入された変数 (以後ダミー変数と呼ぶ) $r_1, r_2, \ldots, r_n$ に依存する関数として、次のように表現できる。導入されるダミー変数の数は、解くべき変数の数以下であることが分かっている。

$$\begin{aligned} p_1 &= mgu_1(q_1, q_2, \ldots, q_m, r_1, r_2, \ldots, r_n) \\ &\vdots \\ p_n &= mgu_n(q_1, q_2, \ldots, q_m, r_1, r_2, \ldots, r_n) \end{aligned} \qquad (2)$$

ダミー変数は、変数 $q_1, \ldots, q_n$ によって構成される任意のブール関数をとることができ、何らかの代入を最汎解に対して施すことによって、与えられたブール方程式 (1) の特解が得られる。すなわち、最汎解中のダミー変数は解の自由度を表現するために導入される。また、与えられた方程式を満たす解は、すべて最汎解中のダミー変数に何らかの代入を行なうことによって表現できる。Boolean unification を用いることによって、式 (1) で表現されている変数 p_i 間の関係を、式 (2) のように関数として表現することが可能となることに注意されたい。

例えば、次式 (3) に対して Boolean unification を行なうことを考えよう。(以下、$'$ は否定を、$\oplus$ は排他的論理和 (exclusive-or) を意味する。)

$$f(p_1, p_2, q_1, q_2) = p_1' \oplus p_2' \oplus p_1' p_2' \oplus q_1 \oplus q_2 = 0 \qquad (3)$$

p_1, p_2 の最汎解は次のように求められる。(実際のアルゴリズムについては次節で述べる。)

$$\begin{aligned} p1 &= q_1 r_1 r_2' \oplus q_2 r_1 r_2' \oplus q_1 \oplus q_2 \oplus 1 \\ p2 &= q_1 r_2 \oplus q_2 r_2 \oplus q_1 \oplus q_2 \oplus 1 \end{aligned}$$

ここで、2 つのダミー変数 r_1, r_2 に対して、それぞれ q_1, q_2 を代入すると次のような特解が求まる。

$$\begin{aligned} p_1 &= q_1 q_1 q_2' \oplus q_2 q_1 q_2' \oplus q_1 \oplus q_2 \oplus 1 = q_1 + q_2' \\ p_2 &= q_1 q_2 \oplus q_2 q_2 \oplus q_1 \oplus q_2 \oplus 1 = q_1' + q_2 \end{aligned}$$

[1] 任意のブール方程式は式 (1) の形に変形できる。例えば、

$$f = g$$

は、次式を解くことと等価である。

$$f \oplus g = 0$$

```
unify(f(p)){
  if (p = ()) then {
    if (f(p) = 0) then return () else fail
  }
  else {
    G(y) = unify(f(0,y) · f(1,y))
    return(
      ((f(0,G(y)) ⊕ f(1,G(y)) ⊕ 1) · r1 ⊕ f(0,G(y)), G(y))
    )
  }
}
```

Note: $\mathbf{p} = (p_1, p_2, \ldots, p_n), \mathbf{y} = (p_2, \ldots, p_n)$

図 1: Boolean unification アルゴリズム ($f(\mathbf{p}) = 0$)

2.2 Boolean unification のアルゴリズム

Boolean unification アルゴリズムとしては、Boole の手法と Löwenheim の手法の 2 つが良く知られている。Boole の手法は、変数消去を行なうことによって最汎解を構成する。一方、Löwenheim の手法では、与えられた方程式の特解を用いることによって、簡単な手続きで最汎解を求める。

図 1に Boole の提案した Boolean unification アルゴリズムを示した。このアルゴリズムは、シャノン展開を行なって再帰的にサブ・フォーミュラに unification アルゴリズムを実行することによって最汎解を求める。アルゴリズム中で用いられるのは通常の論理演算に限られるので、BDD [4] を論理関数表現として用いることにより効率的に実装できる。また、unification アルゴリズムにおいて、変数消去の際にコンセンサス演算が行なわれるが、あらかじめ解くべき変数を BDD の変数順で上位に配置し、その順にしたがって変数消去を行なえば効率的に実行できる。アルゴリズムの詳細については、[10] を参照されたい。

3 組合せ論理合成への応用

3.1 再設計

既に設計された回路に対して、その回路を活かしつつ、新しい仕様を満たすように設計を変更することをここでは再設計 (redesign) [6]、あるいは逐次合成 (incremental synthesis) [16] と呼ぶ。再設計においては、現在の回路はレイアウトまで完成していることを想定する。この既存の回路を有効利用するためには、外部入出力や境界上の端子などに小規模な回路を付加することによって、新たな仕様を満足できるか否かを調べることとなる。

図 2(a) に示すように既存の回路の入力の前に回路を付加する場合を考えよう。このとき、既に設計された回路の論理を $f(t,i)$、新しい仕様を $s(i)$ とすると、再設計の問題は次の方程式を満たす t を i を用いて表現する問題に帰着する。ここで、i, o は入力変数、出力変数のベクトルを、また t は付加回路の出力変数ベクトルを表現している。

$$o = f(t,i) = s(i) \tag{4}$$

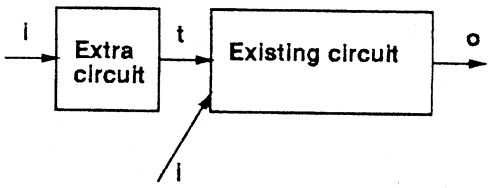
出力 o は、既に設計されている回路について考えると t, i に依存する論理として表現できるが、一方で与えられた仕様 $s(i)$ を満たしていなければならないことを上式は意味している。式 (4) は、次のように式 (1) の形の標準形に変形できる。

$$f(t,i) \oplus s(i) = 0$$

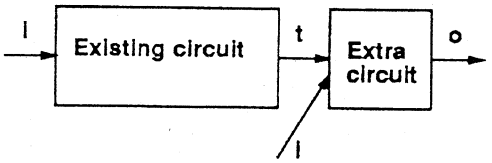
ここで、t は解かれるべき変数、i は定数に相当する。得られた最汎解は、付加回路が満たすべき入出力特性を最も一般的な形で表現している。よって付加回路は最汎解の中から実装にあたりコストが最小のものを選択すればよい。また、入力段に付加回路を付与するだけで修正しきれない場合には、unification の結果、解が存在しないことが分かる。出力段に回路を付加する場合 (図 2(b)) や、これらの組合せ (図 2(c)) についても同様に形式化できる。

3.2 多段論理簡単化

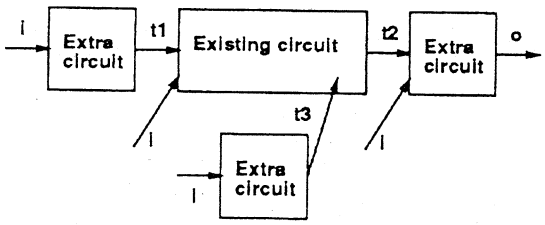
Boolean unification によって得られる最汎解は、与えられたブール方程式を満たすすべての解をダミー変数というパラメータを用いることにより表現している。論理合成の立場から考えると、最汎解は完全な don't care set を表現しているため、論理簡単化の有力な手段となることが期待できる。従来の多段論理簡単化では、基本的に don't care set は 1 つの



(a) 入力側での修正



(b) 出力側での修正



(c) さまざまな修正手法の組合せ

図 2: 再設計

ネット、あるいはゲートに対して定義されており、トランスダクション法 [11] のように回路構造に変更を加えて簡単化を行なう際に、同時に複数のネットの論理を変更することはできなかった。(CSPF [11] は例外といえるが、かなり限定された don't care set である。) 例えば、MSPF [11] は他のノードの論理が変更しないという条件のもとでの、あるノードの許容関数として定義される。

しかし、もし他のノードの論理が変わらないという制限のもとで、同時に複数のノードの論理を変更することができれば、回路構造をより大きく変えることができるようになるため、さらに強力な簡単化を実行できる。例えば、図 3に示した回路中の 2 つのノード x, y の論理が同時に変更可能であるとしたときの don't care set は、単一のノードだけを考慮した don't care set より大きな設計の自由度を表現できることになる。Boolean unification を用いると、これらの don't care set を簡単に求めることができる。図 3の場合には、まず出力関数 o を i, x, y を用いて次のように 2 通りに表現する。

$$o = f(x, y, i) = s(i) \tag{5}$$

式中の $s(i)$ は仕様の論理関数である。式 (5) は、次

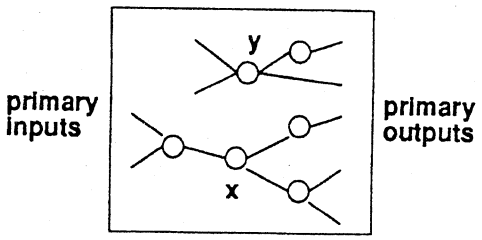


図 3: 多段ネットワーク

$i_1 i_2$	$o_1 o_2$
00	{01,10}
01	{11}
10	{01,11}
11	{01,10,11}

表 1: Boolean relation

式のように変形できる。

$$f(x, y, i) \oplus s(i) = 0 \tag{6}$$

ここで、i を定数と見なして式 (6) を x, y について解くと、x, y は i とダミー変数 r_1, r_2 に依存する論理関数として次のように表現される。

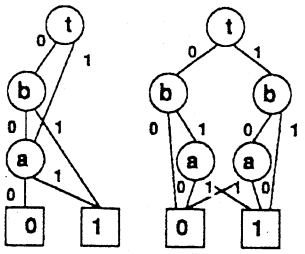
$$\begin{aligned} x &= g_1(i, r_1, r_2) \\ y &= g_2(i, r_1, r_2) \end{aligned}$$

実際にダミー変数 r_1, r_2 に何らかの代入を行なうと、仕様を満たす範囲で許される (x, y) の論理関数の組が得られる。すなわち、最汎解は x, y の許容関数の組の集合を表現しているといえる。ダミー変数のもつ自由度は、各ノードが取り得る論理の don't care 情報と 2 つのノード間の don't care の関係を表現している。単一のノードについて同様の計算を行なって得られる don't care set は MSPF に相当する。

3.3 Boolean Relation の多段簡単化

Boolean relation [2, 14] は多出力不完全指定論理関数の一般形として Somenzi らによって提案されたもので、それぞれの入力最小項に対して複数の出力最小項の記述を許すことによって、従来の don't care で表現できなかった自由度を完全に表現できることが知られている。

表 1に、Boolean relation の例を示した。各入力パターンに対して許容される出力パターンを列挙することにより仕様が与えられている。例えば、この Boolean relation では、入力パターン $i_1 i_2$=00 に対して、2 つの出力パターン $o_1 o_2$={01,10} のいずれかをとることが指定されている。このような仕様は、



(a) BDD for x (b)BDD for y

図 4: 式 (7) の BDD

通常の don't care では表現できないことに注意されたい。出力パターンの集合が 1 つの積項で表現できるときに限り、従来の don't care を用いた不完全指定論理関数で表すことができる[2]。

これまでに、Boolean relation の積和形をターゲットとした簡単化についてさまざまな研究 [14, 7, 15] が行なわれているが、多段簡単化についてはまだ有効な手法は報告されていない。ここでは、Boolean unification を用いることによって、Boolean relation の多段簡単化が形式化できることを示す。

Boolean relation は、与えられた relation によって許されている入出力パターンについてのみ 1 を出力する関数 (特性関数) を用いて表現できる。表 1に示した Boolean relation については、特性関数は次のように定義される。

$$f(i_1, i_2, o_1, o_2) = \\ i_1' i_2' (o_1' o_2 + o_1 o_2') + i_1' i_2 o_1 o_2 \\ + i_1 i_2' (o_1' o_2 + o_1 o_2) + i_1 i_2 (o_1' o_2 + o_1 o_2' + o_1 o_2)$$

従って、与えられた Boolean relation を満たす出力関数としては、特性関数が必ず 1 となるものを選ぶ必要がある。

$$f(o_1, o_2, ..., o_n, i_1, i_2, ..., i_m) = 1$$

上式を Boolean unification を用いて出力変数 $o_1, \ldots, o_n$ について解くことによって各出力について許される論理関数が求められる。Boolean relation では、従来の不完全指定論理関数と異なり出力間には依存関係が存在するが、それらはダミー変数によって表現される。各出力を関数の形で表現できれば、従来の多段論理簡単化の手法を容易に適用できる。

4 最汎解からの回路合成

[2]入力パターン $i_1 i_2$=10 のときには、出力パターン {01,11} が指定されているが、これは don't care を用いて *1 と表現することができる。

Boolean unification によって得られた最汎解の中から、効率的な回路を合成するためには、ダミー変数に対する適切な代入を選び、簡単な特解を見つける必要がある。筆者らは、Boolean unification アルゴリズムを BDD によって実装したため、論理関数の複雑度を BDD におけるノード数を指標として判断し、論理が簡単な特解を選ぶことにした。例えば、次のような最汎解が得られたとしよう。(a, b は入力、x, y は出力、t はダミー変数である)

$$\begin{aligned} x &= a + bt' \\ y &= ab + a't \end{aligned} \qquad (7)$$

x, y の BDD 表現を図 4に示した。ダミー変数 t に対しては、この場合変数 a, b によって構成されるすべての論理関数を代入できる。一般に、n 変数の論理関数は 2^{2^n} 種類あるので、すべての代入を試みてその中から最良の特解を求めることは実用的ではない。そこで、ここでは簡単のためダミー変数には 0 あるいは 1 のみの代入を許すとする。このとき、もし t に 0 を代入した場合は、BDD のノード数の合計が 8(図 5)、t に 1 を代入した場合は 7(図 6) となることより、1 を代入した方がより簡単な結果が得られると期待できる。

- $t = 0$ のとき

$$\begin{aligned} x &= a + b \\ y &= ab \end{aligned}$$

- $t = 1$ のとき

$$\begin{aligned} x &= a \\ y &= a' + b \end{aligned}$$

しかし、実際には最も簡単な特解は t に b を代入したときに得られる。

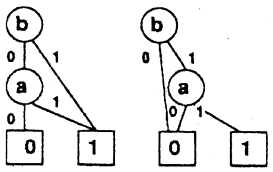
$$\begin{aligned} x &= a \\ y &= b \end{aligned}$$

以上からもわかるように、定数の代入だけに限ると最汎解の自由度を有効利用できず、必ずしも良い解が得られない。最汎解の表現する don't care set を完全に利用できるような論理簡単化手続きについては現在検討中である。

実装に際しては、図 4に示したように、ダミー変数を BDD の変数順で上位にとることによって、定数値を代入したときの結果が簡単に得られるようにした。また、BDD のノード数に関しては、グラフ間で共有されているノードは重複せずに数えることにより、複数関数間での論理関数の共有の効果を考慮した。

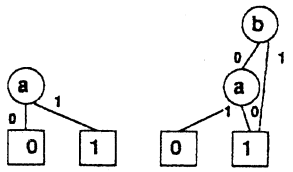
	GYOCRO + MIS	Boolean unification + MIS	
	literals(fac/sop)	literals	CPU time(sec)
int3	13/19	12/15	0.03
int9	26/30	23/27	0.26
int15	437/588	447/560	700.5
gr	270/300	371/447	113.7
b9	162/284	163/278	5.69
vtx	130/2780	67/85	12.66

表 2: Boolean Relation の多段簡単化



(a) BDD for x (b)BDD for y

図 5: $t = 0$ とした場合の BDD



(a) BDD for x (b)BDD for y

図 6: $t = 1$ とした場合の BDD

5 実験結果

Boolean unification と最汎解からの回路合成を行なうプログラムを SBDD [8] を用いて Sparc station2 上に実装した。表 2に示したのは、Boolean relation のベンチマークに、提案した手法を適用した結果である。ただし、ここではダミー変数に対する代入としては定数の他にシングル・リテラルも考慮している。CPU 時間は、Boolean unification と簡単な特解の選択に要した時間である。

Boolean unification を用いた多段簡単化は次の手順で行なった。

1. 特性関数の計算
2. Boolean unification の実行
3. BDD ノード数最小の特解の選択
4. 2 段回路への collapse
5. Espresso による 2 段簡単化
6. MIS2.2 [1] スタンダード・スクリプトによる多段簡単化

GYOCRO [15] はカリフォルニア大学で提案された Boolean relation のための 2 段ヒューリスティック・ミニマイザである。ここでは、生成された 2 段回路を MIS スタンダード・スクリプトで多段化し、提案した手法と比較している。

Boolean unification を用いる手法では、ダミー変数に対する代入を制限しているが、大きな回路についてはかなり良い結果が実用的な時間で得られていることが分かる。CPU 時間の大部分はダミー変数の選択のために用いられており、Boolean unification 自体の計算時間は BDD で実装する限りでは特に重い処理ではない。

6 おわりに

Boolean unification がさまざまな組合せ論理合成の問題に応用できることを示した。unification の結果得られた最汎解は、従来の don't care を用いた手法に比べて広範囲な設計の自由度を表現できるため、複数ノードの論理の変更を許す多段論理簡単化や don't care でとらえることができない Boolean relation の簡単化に適用することができることを示した。また、unification アルゴリズムは BDD を用いて実装すれば効率的に実行できることがわかった。現在は、最汎解の持つ自由度を効果的に扱える簡単化アルゴリズムの検討を行なっている。また、Boolean unification に適した BDD の変数順についても考察していきたい。

謝辞

Boolean relation のベンチマークと GYOCRO による簡単化結果を提供して頂いたカリフォルニア大学の渡部氏に感謝致します。

参考文献

[1] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A multiple-level interactive logic optimization system. *IEEE Transactions on Computer-Aided Design*, Vol. 6, No. 6, pp. 1062–1081, November 1987.

[2] R. K. Brayton and F. Somenzi. Boolean relations and the incompletely specification of logic networks. In *Proceedings of International Conference on Very Large Scale Integration*, pp. 231–240, August 1989.

[3] Frank Markham Brown. *Boolean Reasoning*. Kluwer Academic Pubilshers, 1990.

[4] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computer*, Vol. C-35, No. 8, pp. 677–691, August 1986.

[5] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graph, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of International Conference on Fifth Generation Computer Systems (FGCS-88)*, December 1988.

[6] M. Fujita, T. Kakuda, and Y. Matsunaga. Redesign and automatic error correction of combinational circuits. In *Proceedings of the IFIP TC10/WG10.5 Workshop on Logic and Architecture Synthesis*, pp. 253–262. North Holland, May 1990.

[7] A. Ghosh, S. Devadas, and A. R. Newton. Heuristic minimization of boolean relations using testing techniques. In *Proceedings of IEEE International Conference on Computer Design*, pp. 277–281, September 1990.

[8] Shin-ichi Minato, Nagisa Ishiura, and Shuzo Yajima. Shared binary decision diagrams with attributed edges for efficient boolean function manipulation. In *Proceedings of 27th ACM/IEEE Design Automation Conference*, pp. 52–57, June 1990.

[9] Ursula Martin and Tobias Nipkow. Unification in boolean rings. *Journal of Automated Reasoning*, Vol. 4, pp. 381–396, 1988.

[10] Ursula Martin and Tobias Nipkow. Boolean unification — the story so far. *Journal of Symbolic Computation*, Vol. 7, pp. 275–293, 1989.

[11] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney. The transduction method – design of logic network based on permissible functions. *IEEE Transactions on Computer*, Vol. 38, No. 10, pp. 1404–1424, October 1989.

[12] H. Simonis. Test generation using the constraint logic programming language CHIP. In *Proceedings of 6th International Conference on Logic Programming*, June 1989.

[13] H. Simonis, N. Nguyen, and M. Dincbas. Verification of digital circuits using CHIP. In *The Fusion of Hardware Design and Verification, Proceedings of the IFIP10.2 Working Conference on the Fusion of Hardware Design and Verification*, pp. 421–442, July 1988.

[14] F. Somenzi and R. K. Brayton. An exact minimizer for boolean relations. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pp. 316–319, November 1989.

[15] Y. Watanabe and R. K. Brayton. Heuristic minimization of boolean relations. In *Proceedings of MCNC International Workshop on Logic Synthesis*, May 1991.

[16] Y. Watanabe and R. K. Brayton. Incremental synthesis for engineering changes. In *Proceedings of MCNC International Conference on Logic Synthesis*, May 1991.