DA－90－12

高速な直交分解アルゴリズムと論理合成への応用

松永　裕介

（株）富士通研究所

〒211-88 川崎市中原区上小田中 4-1-1

044-754-2663

yusuke@flab.fujitsu.co.jp

あらまし

本稿では，論理関数の単純直交分解を求める効率の良いアルゴリズムについて述べる．Bertacco と Damiani は二分決定グラフを用いた効率の良いアルゴリズムを提案しているが，彼らのアルゴリズムは不完全であり，正しく直交分解を求めることが出来ない．本稿では彼らのアルゴリズムについて解析を行い，その問題点を指摘した上で，正しく直交分解を求めることの出来る完全なアルゴリズムを提案する．このアルゴリズムは著者が以前，提案した二項分解を求めるアルゴリズムを応用したもので，Bertacco と Damiani のアルゴリズムと同等かより高速に解を求めることができるものである．

キーワード　論理合成，関数分解，二分決定グラフ

An Efficient Disjunctive Decomposition Algorithm and Its Application to Logic Synthesis

Yusuke Matsunaga

Fujitsu Laboratories LTD.

1015 Kamikodanaka, Nakahara-Ku, Kasawaki 211

044-754-2663

yusuke@flab.fujitsu.co.jp

Abstract

Bertacco and Damiani proposed an efficient algorithm finding disjunctive decomposition using Binary Decision Diagrams. However, their algorithm is not complete and does not find all the decompositions. In this paper, we analyze the problem of Bertacco and Damiani's method, and propose an exact and efficient algorithm finding disjunctive decompositions. Experimental results show that our algorithm is comparable or more efficient than Bertacco and Damiani's algorithm and derives accurate decompositions.

key words　logic synthesis, functional decomposition, binary decision diagrams

1 はじめに

論理関数 $F(X)$ が次のように互いに独立な関数を用いて表される場合がある．

$$F(X) = G(H_1(X_1), H_2(X_2), \ldots) \quad (1)$$

このような関数分解を直交分解 (disjunctive decomposition) と呼ぶ [1][1]．任意の関数がこのような分解を持つわけではないが，直交分解は論理設計にとって好ましい性質をいくつか持っている．まず第一に，分解された関数 H_i の入力数は元の関数 F の入力数よりも小さい ($|X| = \sum_i |X_i|$)．n 入力関数を実現する回路は最悪の場合，$O(2^n/n)$ のサイズとなるので，関数をより少ない入力数の部分関数に分解することは，潜在的な回路の複雑度を (指数的に) 下げることになる．第二に，各部分関数の出力数が 1 であるので，通信の複雑度 (communication complexity) が下げられる．これは配線リソースの削減につながる．

直交分解はある意味ユニークに求めることができる[2] ので，多段論理合成の初期解として用いることは妥当であると思われる．もしも，分解された部分関数に対して最適解を求めることができたなら，それらを結合することによってもとの関数に対する最適解を得ることができる．このような分割統治法は論理設計において必要とされる計算時間を大幅に短縮するものである．このように与えられた論理関数に対して直交分解を求める問題は論理設計に対して重要な意味を持つ．

Ashenhurst と Curtis は *decomposition chart* を用いて直交分解を求める厳密アルゴリズムを提案した [1]．しかし，この手法はあらかじめ変数の分割を与えておいて，そのような変数の分割に基づく関数分解が存在するかどうかを判定するだけであるので，すべての変数の分割 (その数は入力数の指数乗となる) を網羅的に列挙して調べなければならない．笹尾は decomposition chart の代りに二分決定グラフ (Binary Decision Diagrams:BDD's)[3] を用いる手法を提案している [2] が，これも一回の処理にかかる計算時間を短縮しているに過ぎず，どのように変数の分割を求めるか，という本質的な問題を解決してはいなかった．

近年，Bertacco と Damiani は二分決定グラフを用いて直交分解を求める効率的なアルゴリズムを提案した [5]．彼らは，もしも関数が直交分解を持つのならば，そのコファクターも同様の分解を持つ，という点に着目した．彼らのアイデアは画期的であり，その実験結果を見る限りでは有効なものと思われたが，笹尾が指摘したように [6]，彼らの結果は正しいものではなかった．事実，彼らのアルゴリズムは不完全であり，ある種の (しかもありふれた) 直交分解を求めることができない．一方，湊と De Micheli は Minato-Morreale のアルゴリズム [9] によって生成された非冗長積和形に対して algebraic decomposition (weak-division[3]) を適用することによってすべての直交分解を求めることができることを示した．この結果は disjunctive decomposition と algebraic decomposition の関係を解析したものとして理論的に興味深いが，論理関数に対して非冗長積和形表現を求める処理を行うぶん Bertacco と Damiani のアルゴリズムよりは効率が悪くなっている．

本稿では，この Bertacco と Damiani のアルゴリズムの問題点を解析し，直交分解を効率よく求める正しいアルゴリズムの提案を行う．このアルゴリズムは Bertacco と Damiani のアルゴリズムと，著者が以前開発した二項分解 (bi-decomposition) を求めるアルゴリズム [11] を組み合わせたものである．実験結果によれば，提案手法は Bertacco と Damiani のアルゴリズムと同等かさらに効率よく，しかも完全な直交分解を求めることができている．

本稿は以下のように構成される．2 章では，直交分解を求めるための必用十分条件について論じ，Bertacco と Damiani のアルゴリズムに対する反例を提示する．3 章ではそれらの問題を考慮した新しいアルゴリズムの提案を行う．4 章で効率的な処理のための実装について述べた後，5 章で実験結果を示す．

2 Bertacco と Damiani のアルゴリズムの解析

2.1 Bertacco と Damiani のアルゴリズム

Bertacco と Damiani のアルゴリズム [5] は以下のようなアイデアに基づいている．

1. 論理関数 F の直交分解は入力／出力の極性と入力順序を適切に正規化することによってユニークに求めることができる．すなわち，論理関数 F に対して，$F = G(A(X_1), B(X_2), C(X_3), \ldots)$ を満たすような関数の集合 $G, A, B, C, \ldots$ は唯一に定まる．ただし，この場合，G は更なる分解を持たないものとする．
2. 論理関数 F が次のような分解を持つものとする．

$$F = G(A(X_1), B(X_2), C(X_3), \ldots)$$

ここで X_1 に属する変数 $x(x \in X_1)$ に対する F

[1] 厳密には，このように各部分関数が 1 出力関数であるような直交分解を単純直交分解 (simple disjunctive decomposition) と呼ぶ

[2] 後述するように AND/OR/XOR の形の分解は結合則が成り立つのでユニークにならない．

[3] 例えば文献 [10] を参照のこと．

のコファクターを考えると，

$$F_0 = G(A_0(X_1 \setminus x), B(X_2), C(X_3), \ldots) \quad (2)$$
$$F_1 = G(A_1(X_1 \setminus x), B(X_2), C(X_3), \ldots) \quad (3)$$

の様になる．ここで，F_0 は F の $x=0$ に対するコファクターを表す ($F_0 = F|_{x=0}$)．同様に，A_0 は A の $x=0$ に対するコファクターを表すものとする．
逆に，もしも F のコファクターが上記の式を満たすのならば，次のような関数分解を得ることができる．

$$F = G(A(X_1), B(X_2), C(X_3), \ldots)$$

つまり，式 (2) および式 (3) は直交分解のための必用十分条件となる．

3. そこで，論理関数 F の直交分解はそのコファクター F_0 と F_1 の直交分解から求めることができる．

例えば，$F = (a+b)(c \oplus d) + \overline{a}\overline{b}(e+f)$ という関数を考えてみると，a に対するコファクターは

$$F_0 = b(c \oplus d) + \overline{b}(e+f)$$
$$F_1 = c \oplus d$$

であり，これらは以下のような分解を持つ．

$$F_0 = ITE(b, c \oplus d, e+f) \quad (4)$$
$$F_1 = c \oplus d \quad (5)$$

ここで，$ITE(x,y,z)$ は *If-Then-Else* 関数を表すものとする．すなわち，$ITE(x,y,z) = xy + \overline{x}z$ である．
もしも，$G = ITE(x,y,z)$, $A = a+b$, $B = c \oplus d$, $C = e+f$ を式 (4), (5) に代入すると，これらの式は直交分解の必用十分条件である式 (2), (3) を満たすことがわかる．とは言っても，この例は上記の必用十分条件を自明に適用できるものではない．これは A_1 が定数 1 となっており，$ITE(1, c \oplus d, e+f)$ が全く見栄えの異なる関数 $c \oplus d$ となっているからである．
そこで，以下のような境界条件を考慮する必要がある．

場合 1 $F = x + A$ or $F = x \cdot A$.

場合 2 $F = x \oplus A$

場合 3 $x \in X_1, |X_1| \geq 2$. すなわち，x は (単純でない) 部分関数 A の変数.

場合 4 $A = x$ もしくは $A = \overline{x}$. すなわち，x 自身が部分関数 A となる．

場合 1 と場合 2 は単純である．場合 3 は，コファクター A_0 と A_1 が定数か否かでさらに 2 つに分類される．

場合 3.a A_0 と A_1 のどちらも定数ではない．

この場合，F_0 と F_1 は A_0 と A_1 の部分を除いて同一の分解を持つことになる．

場合 3.b どちらか一方のコファクター (A_0 か A_1) が定数である．

この場合，定数でないコファクターへ定数値を割り当てることで，F_0 と F_1 が同一となる．

2.2 Bertacco と Damiani のアルゴリズムの反例

例題 1 以下のような関数 F を考える．

$$F = b + ITE(a, c+d, e+f)$$

F の a に対するコファクターは以下のようになる．

$$F_0 = b + e + f \quad (6)$$
$$F_1 = b + c + d \quad (7)$$

どちらのコファクターも *3* 入力 *OR* であり，一つの共通な部分関数 *(*この場合には B というリテラル関数*)* を持つ．そこで，もしも，$b+e+f = b+(e+f)$, かつ，$b+c+d = b+(c+d)$ と認識できれば，$F = b + ITE(a, c+d, e+f)$ という分解を容易に得ることができる．しかし，*Bertacco* と *Damiani* のアルゴリズムは *3* 入力 *OR* 関数をそれ以上分解の出来ない関数として扱っているので彼らのアルゴリズムではこの分解を求めることはできない[4]．

例題 2 以下のような関数 F を考える．

$$F = a(c+d) + b$$

F の a に対するコファクターは以下のようになる．

$$F_0 = b \quad (8)$$
$$F_1 = b + c + d \quad (9)$$

もしも，$c = d = 0$ を F_1 に代入すると，F_1 は b となり F_0 と等しくなる．つまり，$b+c+d = b+(c+d)$ と見なすことで，$F = a(c+d)+b$ の分解を得ることができる．ところが，*Bertacco* と *Damiani* のアルゴリズムではこれを場合 *4* と認識してしまい，分解を求めることができない．

[4] 彼らの論文 [5] では，OR/XOR の分解は場合 4 に関してのみ考慮されている．

彼らのアルゴリズムの問題点は多入力のOR/XOR関数の取り扱いの曖昧さに起因する．明らかに，多入力のOR/XOR関数は異なる分解を持つ．例えば，$a+(b+c)=(a+b)+c=b+(a+c)$のように3入力ORは3通りの二項分解を持つ．BertaccoとDamianiのアルゴリズムでは分解の構造をカノニカルに保つために，極大なOR/XOR関数のみを用いている．しかし，彼らのアルゴリズムは各々の分解が極小である，すなわち，これ以上細かな分解を持たない，という仮定の上に成り立っているので，極大なOR/XOR関数もこれ以上分解できないものと見なすことになる．これが彼らのアルゴリズムが不完全な理由であり，直交分解可能な関数の数の実験結果が他の文献と異なる[6]理由である．

3 直交分解を求める厳密アルゴリズム

直交分解を正しく求めるためにはOR/XOR関数を別個に扱う必要がある．著者は以前に二項分解をボトムアップに求める効率の良いアルゴリズムを開発している[11]．二項分解は直交分解の特別なケースで，$F=G(H_1,H_2)$の形で表されるものである．すべての二入力関数はORかXORのNPN同値類(入力／出力の極性反転，および入力順序の並べ替えによって等しくなる関数の集合)であるので，ここでは$F=H_1+H_2$と$F=H_1\oplus H_2$の2つの場合のみ考慮すれば良い．このアルゴリズムもBertaccoとDamianiのアルゴリズムと同様に，コファクターの分解の情報を用いて，自分自身の分解を行うものである．このアルゴリズムとBertaccoとDamianiのアルゴリズムを組み合わせることによって正しく直交分解を求める厳密アルゴリズムを作ることができる．

3.1 直交分解を表すデータ構造

文献[5]と同様に，ここでも直交分解を表すグラフ構造—**decomposition graph**—を導入する．ただし，このデータ構造は文献[5]のものと少し異なっているので，以下に説明を行う．

分解グラフ$DG(V,E)$は節点の集合Vと枝の集合Eから成る．節点には以下の種類がある．

定数0 定数0を表す．入枝は持たない．このタイプは元の関数が定数関数である場合以外には用いられることはない．

リテラル リテラル関数を表す．変数番号を持つ．入枝は持たない．

OR OR関数を表す．任意の数の入枝を持つ．

XOR XOR関数を表す．任意の数の入枝を持つ．

Other それ以外の(単純でない)関数を表す．任意の数の入枝を持つ．

各々の節点は分解の構造と同時に論理関数も表している．そこで，各々の節点はその関数を表すBDDの節点へのポインタを持っている．

各々の枝は極性の属性を持つ．文献[4]のBDDの実装と同様に，否定の属性を持った枝の表す論理関数は，その枝の指している節点の表す論理関数の否定である．

分解グラフをカノニカルに保つために，以下のような規則を設ける．

- **XOR**節点の入力の枝には否定の属性を付けない．
- **Other**節点の入力の枝には否定の属性を付けない．
- **Other**節点の出力の枝の極性はBDDの枝の極性と同一にする．

3.2 二項分解を扱うための特別な場合分け

ここではOR/XOR二項分解に注目した特別な場合の処理を示す．それ以外の処理はBertaccoとDamianiのアルゴリズムと同様に行える．

ここで，関数Fのxに対するコファクターをF_0およびF_1とし，それらの分解構造を表す枝をe_0およびe_1とする．また，v_0およびv_1をe_0とe_1のソースの節点とする．

場合3.a-OR v_0とv_1が共に**OR**節点で，一つ以上の入力枝が共通であり，かつ，e_0とe_1の極性が同一の場合，共通な入力枝を一つの**OR**節点にマージし，残りの枝と新たな節点を作る(図1)．この場合，枝として(極性も含めて)同一である必要がある．

場合3.a-XOR v_0とv_1が共に**XOR**節点で，一つ以上の入力枝が共通な場合，共通な入力枝を一つの**XOR**節点にマージし，残りの枝と新しい節点を作る．場合3.1-ORと異なり，e_0とe_1の極性が異なっていてもマージすることが可能で，その場合には，極性の差が新しい節点へ伝播することになる(図2)．

場合3.b-OR 一方の節点(ここではv_0とする)が**OR**節点であり，他方の節点v_1がv_0の入力となっており，かつ，e_0からv_1へ至る経路上の極性とe_1からv_1へ至る経路上の極性(e_1の極性自体)が等しい場合，これらをマージする(図3)．

場合 3.b-XOR 一方の節点 (ここでは v_0 とする) が **XOR** 節点であり，他方の節点 v_1 が v_0 の入力となっている場合，これらをマージする (図 4)．場合 3.b-OR と異なり，極性が異なっていてもマージできる．

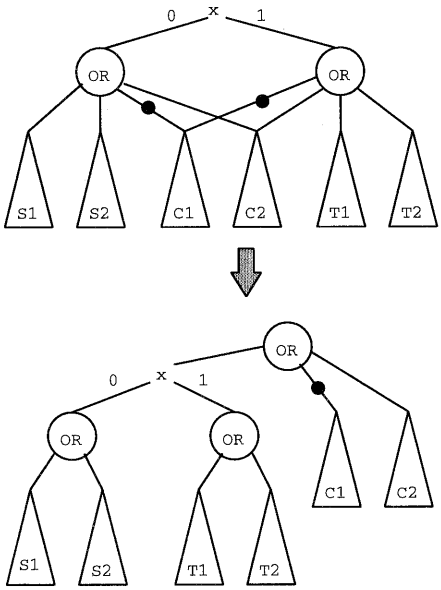

図 1: **OR** 節点のマージ (1)

論理関数 F のコファクターが以下の条件を満たすとき，その論理関数 F は二項分解を持つ [11]．ここで $\circ$ は任意の二項演算である．

$$F_0 = A \circ C \quad (10)$$
$$F_1 = B \circ C \quad (11)$$

この条件は必用条件でもあるので，これが二項分解を持つための必要十分条件となる．上記の 4 つのケースでこの条件をすべてカバーしているので，これらの 4 つのケースを Bertacco と Damiani のアルゴリズムに追加することで，直交分解を完全に求めることができる．

この拡張に伴う計算はすべて，分解グラフ上の局所的な変換であり，BDD を用いた関数処理を必要としないので，この拡張による計算量の増加はわずかなものであり無視できる．

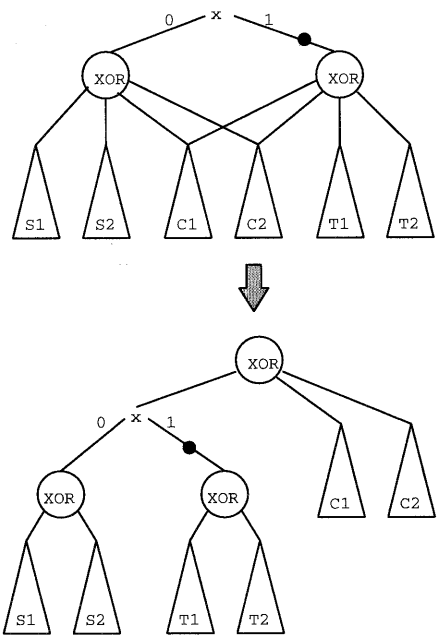

図 2: **XOR** 節点のマージ (1)

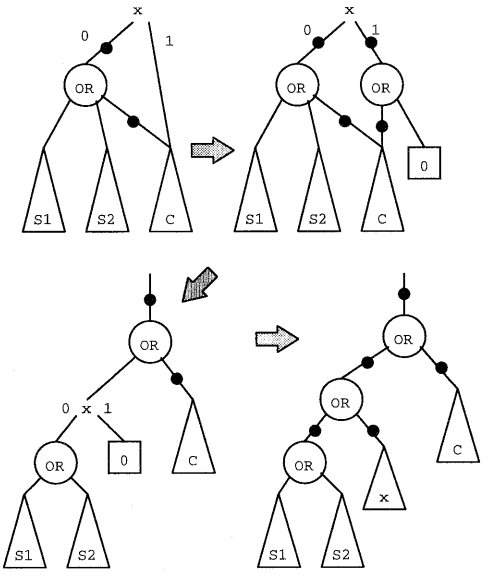

図 3: **OR** 節点のマージ (2)

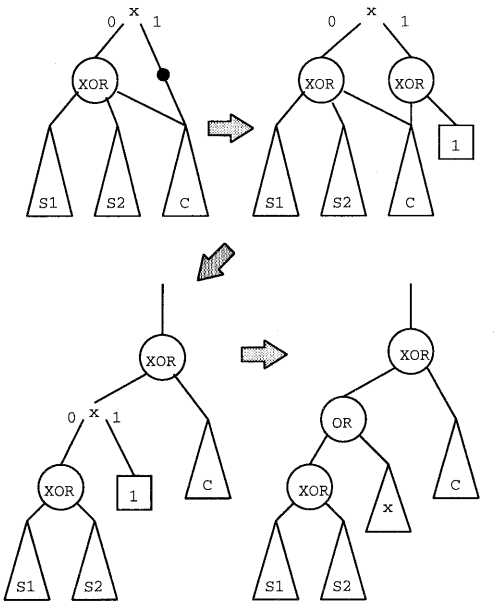

図 4: **XOR** 節点のマージ (2)

4 アルゴリズムの実装

直交分解を求めるプログラムを高速にするためにはアルゴリズムの実装にも細心の注意を払う必要がある．ここでのキーポイントはなるべく新たな BDD の節点を生成しないようにすることである．例えば，F_0 において A_1 を代入した結果が F_1 と等しくなるかを調べる処理を行うには，単純にはまず F_0 に対する $A=1$ のコファクターを計算し，そのコファクターが F_1 と等しいか比べれば良い．これがもっとも当たり前の方法であるが，この場合，中間的に新しい BDD の節点が生成されることになり，複雑な論理関数の場合に処理速度の低下を招く．そこで，"check_with_cofactor" と呼ぶ新しいアルゴリズムを開発した．このアルゴリズムは比較される 2 つの BDD および，これらの BDD の領域を制限する付加的な BDD(1 つもしくは 2 つ) を引数にとり，制限された領域内で 2 つの BDD が等しいかどうか比べるものである．もしも 2 つの BDD が制限された領域内で異なることが分かったら，残りの処理は行われずに false が返される．このように，すべての探索の終了を待たずに結果を返す手法は BDD を用いた処理の高速化の技法の一つとなっている．

もう一つ，直交分解に特化したアルゴリズムの開発を行った．これは次式を満たすような変数 x を効率よく求めるためのものである．

$$F_0(\ldots,x,\ldots)=F_1(\ldots,\overline{x},\ldots)$$

ここではこのような変数を "pivot" と呼ぶことにする．この処理は場合 3 で必要となる．もしも論理関数 F が $G(A,B,C,\ldots)$ の様に分解され，かつ，$A=x\oplus A'$ と表される場合，F のコファクターは以下の様になる．

$$F_0 = G(A',B,C,\ldots) \quad (12)$$
$$F_1 = G(\overline{A'},B,C,\ldots) \quad (13)$$

A' を反転させることで F_0 と F_1 が一致することがわかれば，$G(x\oplus A',B,C)$ という分解を得ることができる．このような pivot 変数を求める最も単純な方法は，"check_with_cofactor" をすべての入力変数に対して適用することであるが，大きな論理関数の場合にはこのチェックに莫大な時間が費やされることになる．また，多くの場合，このような pivot 変数は存在しない．そこで，このような pivot 変数を効率よく 1 ステップで求めるアルゴリズムを開発した．このアルゴリズムは "FindPivotVariable" と呼ばれるもので，以下のようになっている (図 5)．

各再帰ステップにおいて，FindPivotVariable は論理関数 P を返り値として戻す．この P は pivot 変数の条件を表している．もしも変数 x が pivto 変数ならば，P の $x=1$ に対するコファクターは 0 ではない．逆に P の $x=1$ に対するコファクターが 0 の場合には変数 x は pivot 変数ではない．

もしも $F=G$ のとき，$F(=G)$ のサポート変数は pivot 変数ではないので，これらの変数の否定リテラルの積をとったものが P となる (4 行目)．もしも F か G が定数の時 (かつ，それらは等しくない)，この時点で F と G はどのような変数を反転しても一致する望みがないので 0 を返す (7 行目)．F と G の最上位の変数が異なるとき，つまり，F に含まれて G に含まれない変数が存在するか，G に含まれて F に含まれない変数が存在する時，いかなる変数の反転によっても F と G は一致しないので 0 を返す (12 行目)．これらの特別なケースをチェックした後で，同一の計算がすでになされていないかをハッシュ表でチェックし，登録されていない場合にのみ本当の処理を行う．16 行目から 18 行目において，変数 x を反転させた場合の条件 Q を計算する．この結果はコファクターに対する再帰呼び出し FindPivotVariable(F_0, G_1) と FindPivotVariable(F_1, G_0) から作られる．ここでは x を反転させると仮定しているので，反対のコファクター同士がペアになっている．19 行目から 21 行目において，x を反転させない場合の条件 Q を計算している．これらの結果から最終的な条件 P を求める．$P=x\cdot Q+\overline{x}\cdot R$．最後に P はハッシュ表に登録される．

```
int FindPivotVariable(F, G)
{
1:    if (F = G) {
2:        // Supports of F are not pivot variables
3:        S ← support of F;
4:        return ∏_{l∈S} l̄;
5:    }
6:    if (F or G is a constant) {
7:        return 0;
8:    }
9:    x ← top variable of F;
10:   y ← top variable of G;
11:   if (x ≠ y) {
12:       return 0;
13:   }
14:   P ← Precomputed(F, G);
15:   if (P = undefined) {
16:       Q_0 ← FindPivotVariable(F_0, G_1);
17:       Q_1 ← FindPivotVariable(F_1, G_0);
18:       Q ← Q_0 · Q_1;
19:       R_0 ← FindPivotVariable(F_0, G_0);
20:       R_1 ← FindPivotVariable(F_1, G_1);
21:       R ← R_0 · R_1;
22:       P ← x · Q + x̄ · R;
23:       Register(F, G, P);
24:   }
25:   return P;
}
```

図 5: The algorithm "FindPivotVariable"

5 実験結果

上記のアルゴリズムを C++で実装し，評価のための実験を行った．表 1 に実験結果を示す．実験は，ベンチマーク回路の各出力に対する論理関数を求め，それを直交分解するように行われた．PI と PO と書かれたコラムはベンチマーク回路の入力数および出力数を示している．DEC と書かれたコラムは直交分解の存在した関数の数であり，CBF と書かれたコラムは完全に二項分解された関数 (completely bi-decomposable functions[6]) の数を示している[5]．最後の CPU と書かれたコラムは直交分解に要した計算時間を秒の単位で示している．この実験には Pentium-II 233MHz の PC(FreeBSD-2.2.5R) が用いられた．

表 1: 直交分解の結果

name	PI	PO	DEC	CBF	CPU
C1355	41	32	0	0	8.87
C1908	33	25	7	0	1.42
C2670	233	64	41	32	0.21
C3540	50	22	14	4	3.48
C432	36	7	1	1	0.28
C499	41	32	0	0	8.80
C5315	178	123	80	39	0.19
C7552	207	108	107	57	14.09
C880	60	26	26	17	0.92
alu4	14	8	4	3	0.15
apex6	135	99	99	26	0.41
apex7	49	37	37	23	0.37
b9	41	21	21	8	0.02
count	35	16	16	0	0.01
des	256	245	245	4	0.36
frg2	143	139	139	40	0.15
pair	173	137	137	33	7.36
too_large	38	3	3	0	0.09

分解可能な関数の数および完全に二項分解可能な関数の数は文献 [6] と完全に一致しており，提案したアルゴリズムの正しさを裏付けている．一方，文献 [5] ではこれよりも少ない数が報告されており，2 章で述べたように彼らのアルゴリズムに誤りのあることがわかる．笹尾が指摘しているように，C3540 は 4 つの CBF 関数を持つ．例えば，出力 355 の関数は以下のようになっている．

$$F^{355} = \overline{87} + \overline{97} \cdot \overline{107} \tag{14}$$

[5] 文献 [6] と同様に，ここでは $F = x$ のような関数も DEC と CBF に含まれている．

変数順が，$97 \prec 87 \prec 107$ と仮定すると，97 に対するコファクターは，

$$F_0^{355} = \overline{87} + \overline{107} \quad (15)$$
$$F_1^{355} = \overline{87} \quad (16)$$

となる．これは場合3.b-ORにマッチするが，Bertacco と Damiani のアルゴリズムでは分解することが出来ていない．

計算時間に関して言えば，実行する計算機環境の違いがあるが，提案したアルゴリズムが Bertacco と Damiani のアルゴリズム [5] と同等か高速に解を求めていると言える．特に，C1355, C499, C3540 といって比較的時間のかかる例においては，提案したアルゴリズムの性能は文献 [5] を上回っている．これは，4章で述べて実装上の工夫が有効であったものと思われる．

6 おわりに

本稿では直交分解を正確に求める効率の良いアルゴリズムについて述べた．このアルゴリズムは従来手法と同等かより高速に処理を行い，かつ，完全な分解結果を得ることの出来るものである．前述の様に直交分解はユニークに定まり，場合によっては分解できない関数も存在するため，直交分解そのものが完全な論理最適化手法となるわけではないが，直交分解は他の論理最適化手法の妨げになるものではなく，問題を分割することによって各々の処理の高速化に役立つものである．直交分解と，従来もしくは新規の論理最適化手法を組み合わせることで，より強力な論理最適化手法を構築することができるものと思われる．

謝辞

本研究を進めるに当たって貴重な議論をしてくださった，九州工業大学の笹尾勤教授と NTT の澤田宏氏に感謝致します．

参考文献

[1] R.L. Ashenhurst, "The decomposition of switching functions", *In Proceedings of an international symposium on the theory of switching*, pp. 74-116, April 1957.

[2] T. Sasao, "FPGA design by generalized functional decomposition", *in Logic Synthesis and Optimization*, Kluwer Academic Publishers, pp. 233-258, 1993.

[3] R.E. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computer*, C-35(8), pp. 677–691, Aug. 1986.

[4] K.S. Brace, R.L. Rudell, and R.E. Bryant, "Efficient implementation of a BDD package", *In Proceedings of the 27th Design Automation Conference*, Jun. 1990, pp. 40-45.

[5] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions", *In Proceedings of the International Conference on Computer-Aided Design (ICCAD'97)*, pp. 78-82, November 1997.

[6] T. Sasao and T. Matsuura, "Decompos: an integrated system for functional decomposition", *ACM/IEEE International Workshop on Logic Synthesis (IWLS'98)*, June 1998.

[7] T. Sasao and J.T. Butler, "On bi-decompositions of logic functions", *ACM/IEEE International Workshop on Logic Synthesis (IWLS'97)*, May 1997.

[8] S. Minato and G. DeMicheli, "Find All Simple Disjunctive Decompositions Using Irredundant Sum-of-Products Forms", *In Proceedings of the International Conference on Computer-Aided Design (ICCAD'98)*, pp. 111-117, November 1998.

[9] S. Minato, "Fast Generation of Irredundant Sum-of-Products Forms from Binary Decision Diagrams", *In Proccedings of Synthesis and Simulation Meeting and International Interchange (SASIMI'92)*, pp. 64-73, April 1992.

[10] R.K. Brayton, R.L. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: A Multiple-Level Logic Optimization System", IEEE transactions of CAD, Vol. CAD-6, No. 6, pp.1062-1081, Nov. 1987.

[11] 松永 裕介, "論理関数の bi-decomposition を行なうボトムアップアルゴリズムについて", 信学技法 VLD96-90,ICD96-200(1997-03), pp. 25-32, March 1997.