計算機アーキテクチャ　121－1
設　計　自　動　化　82－1
（１９９６．１２．１２）

論理関数のXOR分解アルゴリズムについて

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

[概要]　ファクタードフォームはテクノロジ独立なレベルでの回路構造，および回路面積の見積りやテクノロジマッピングの初期解に適した論理関数表現であるが，AND/OR演算子のみを用いているのでXORを含んだ回路に対しては適切な論理式表現とならない．本稿では，XOR演算子を含んだファクタードフォームを生成するための一手法として，関数のXOR分解について取り上げ，2種類のアルゴリズムを提案する．一つは互いに素なサポートを持つ関数への分解を行なうアルゴリズムであり，もう一つはただ一つの変数を共通のサポートとして持つ関数への分解を行なうアルゴリズムである．どちらも二分決定グラフを用いて効率良く実行することができるので，従来のファクタリングアルゴリズムと組み合わせることで，より簡潔な論理式表現を得ることができる．

キーワード　論理合成，関数分解，二分決定グラフ

On Exclusive-Or Decomposition Algorithms of Logic Functions

Yusuke Matsunaga

Fujitsu Laboratories LTD.

1015 Kamikodanaka, Nakahara-Ku, Kasawaki 211

044-754-2664

yusuke@flab.fujitsu.co.jp

[abstract]　Factored form is a natural way to express a logic function with keeping multi-level logic circuit's structure. Thus, its literal count is generally used for evaluating circuit area in technology independent level. And factored form is also a good start point to technology mapping. However, in the case of technology mapping including XOR cells, factored form is not a good measure nor a good start point, because it consists of only AND/OR terms. So, effective algorithms to find a good logic expression including XOR terms are required. This paper describes novel XOR factoring algorithms that find a disjoint support decomposition or a single overlapping support decomposition. Implemented with Binary Decision Diagrams, these algorithms are very efficient. Combining these XOR factoring algorithms with the conventional AND/OR factoring algorithms, simpler logic expressions are expected to be derived.

Keywords　logic synthesis, function decomposition, binary decision diagrams

1 はじめに

ファクタードフォームは多段論理回路の構造を表すことのできる論理式表現である．そこで，テクノロジ独立なレベルでの論理合成／最適化処理においては，ファクタードフォームのリテラル数が回路の面積を見積もる良い指標として用いられている．ファクタードフォームはまた，テクノロジマッピングのためのよい初期回路ともなっている．tree covering アルゴリズムに基づく多くのテクノロジマッパーは，2入力NAND木のネットワークに分解された回路をその初期解としており[1]，ファクタードフォームはそのようなNAND木を作るためにも用いられている．さらに，リテラル数の少ないファクタードフォームをつくり出すさまざまなアルゴリズムやヒューリスティックが提案されている [2, 3].

しかし，このような手法に問題がないわけではない．たとえば，5入力XOR(排他的論理和)の関数をファクタードフォームで表現することを考えてみると，どのようにしてもコンパクトに表せることはできない[1]．しかるに，この関数は4つの2入力XORゲートで実現可能である．つまり，この場合，ファクタードフォームが実際の回路面積に対するよい評価指標とはなっていない．また，テクノロジマッピングの初期解として不適切な場合がある．例えば，先ほどの例で，5入力XOR関数を一つの (80リテラルの) ファクタードフォームとして表せば数十個の2入力NANDゲートを使った木が生成されてしまうし，図1(a)のようにコンパクトに表すと木ではなくなる(途中のゲートが2つ以上のファンアウトを持っている)ので，tree-covering アルゴリズムでマッピングを行うためには同図(b)の様に，4つの木に分解する必要があり，結果として同図(c)のような4つの2入力XORゲートを用いた回路にしかマッピングできず，もしも，セルライブラリの中に3入力XORゲートが含まれていたとしても，用いられることはない．もちろん，3入力XORの関数を表す2入力NANDゲートの木を作ることは可能であり，その木に対して3入力XORゲートをマッピングすることもできるが，その場合には，逆にその木を2つの2入力XORゲートで構成することはできなくなり，マッピング時の自由度を狭めることになる．

[1] 1つの項に5リテラル必要とする最小項が16必要なので，計80リテラルのファクタードフォームとなる．

このような問題に対する根本的な解決法としては，2入力NAND木への複数の分解に対してマッピングを行なうとか，木ではなくDAG(非巡回有向グラフ)に対するcoveringを行なうなどが考えられるが，本質的に難しい問題であり，実用的な回路規模に対して効率の良いアルゴリズムが存在するとは思えない．そこで，比較的簡単な対処法としてXOR演算子をファクタードフォームに含める方法が考えられる[4]．すると，先の例の5入力XORの関数は図1(c)の様に表されることになり，2入力XORゲートを用いても3入力XORゲートを用いてマッピングすることができるようになる．また，リテラル数も5となり，テクノロジ独立なレベルでの見積りとしては妥当な値となる[2]

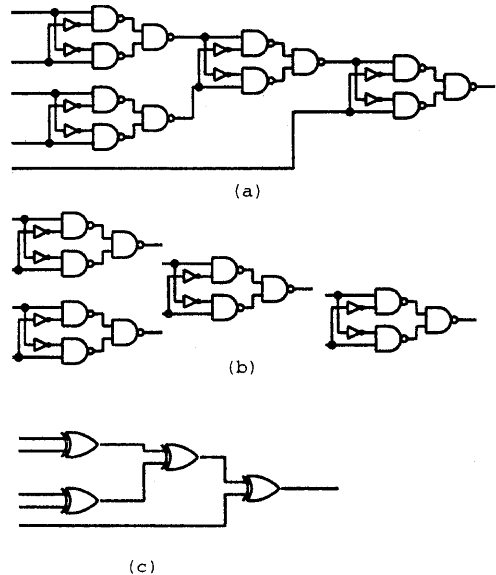

図 1: 5-XOR function and its representations

このようにXOR演算子を論理式表現に加えることで上記の問題に対応できるが，最も重大な問題は，XOR演算子を含んだ“良い”ファクタード

[2] もしも，2入力NANDゲート (AND/OR演算子) と2入力XORゲート (XOR演算子) を同等に扱うのが適切でないならば，リテラル数の代わりに2入力NANDゲートと2入力XORゲートに適当な重みを付けてその重み付きのゲート数の総和を用いれば良い．2入力NANDだけの木の場合には，ゲート数 = リテラル数 − 1 の関係が常に成り立つので，ゲート数をリテラル数の代わりに用いることも妥当と思われる．

フォームを求める効率の良いアルゴリズムが現在のところ知られていない，ということである．従来のファクタリングアルゴリズムでは，与えられた積和形論理式 F から除数 D を求め，その除数を用いて除算を行ない，以下のような商 Q および剰余 R を求めている [2, 3].

$$F = (D \cdot Q) + R \tag{1}$$

この処理を，D，Q，R に対して再帰的に行なうことで，AND 演算子と OR 演算子を用いた論理式 (ファクタードフォーム) を得ることができる．しかし，論理式に対する除算の結果は一意ではなく，また，その計算量も多いので，通常は論理式を代数式 (算術式) と見なして商と剰余を求めるアルゴリズムが用いられている．そのような代数的な除算に適した除数としてカーネル (kernel) と呼ばれる部分論理式が提案されているが，カーネルが XOR 演算子を含んだファクタードフォームを生成するのに効果的かどうかは明らかでない．基本的に積和形論理式を代数式として見なした場合，XOR 演算に基づく除算 (例えば，$F = (D \oplus Q) + R$ や $F = (D \oplus Q) \cdot R$) を代数式的に行うことはできず[3]，従来手法を単純に拡張することは難しい．ESOP(Exclusive-Sum Of Products:XOR を用いた 2 段論理式) を出発点としたファクタリングアルゴリズムはいくつか提案されているが，結果として得られる論理式は初期解に大きく依存しており，また，どのような ESOP が XOR を含んだファクタードフォームに適しているかに関してはほとんど議論がなされていない．

本稿では，XOR 演算子を含んだ論理式を生成するための手法として，2 つの XOR 分解アルゴリズムについて提案する．一つは互いに素なサポートを持つ関数へ分解するアルゴリズムであり，もう一つはただ一つの変数のみを共通のサポートとして持つ関数へ分解するアルゴリズムである．これらのアルゴリズムは関数操作に基づいているので，SOP(Sum Of Products:積和形論理式) や ESOP の式にを必要としないし，それらの形によって結果が異なることもない．もちろん，これらが唯一の手法ではないが，素なサポートを持つ関数へ分解ができるのならば後はその部分関数の最適なファクタードフォームを求めることで，全体としても最適なファクタードフォームを生成することができる．そのような意味で，これらの XOR 分解は他のヒューリスティックを試す前に行っておいて損はない手法ということができる．

[3] $D \oplus Q = D \cdot \overline{Q} + \overline{D} \cdot Q$ であるが，代数式としては D と $\overline{D}$ は全く別個の式であると見なされる．

本稿の構成は以下の様になっている．まず 2 節で用語などの定義を簡単に述べ，続く 3 節で互いに素なサポートを持つ関数への XOR 分解のアルゴリズムを提案する．4 節ではさらにこのアルゴリズムを応用して，ただ一つの変数を共通のサポートとして持つ関数へ分解するアルゴリズムを提案する．5 節で基礎的な実験結果を述べて，6 節で結論を述べる．

2 諸定義

2.1 排他的論理和を含むファクタードフォーム

本稿では通常のファクタードフォーム [3] に対して，排他的論理和を表す項を追加したものを扱う．その再帰的な定義は以下のようになっている [4].

1. **積項**とはリテラルもしくはファクタードフォームの論理積である．
2. **和項**とはリテラルもしくはファクタードフォームの論理和である．
3. **排他的論理和項**とはリテラルもしくはファクタードフォームの排他的論理和である．
4. **ファクタードフォーム**とは，積項，和項，排他的論理和項のいずれかである．

2.2 サポート

論理関数 F のサポートとは，その関数が実際に依存している変数の集合のことであり，ここでは $\mathcal{S}(F)$ で表すものとする．たとえ，論理関数を表す式中に現われる変数でもサポートとはならない例は存在する．例えば，$F = a \cdot b + \overline{a} \cdot b$ に対して，b はサポートに含まれるが，a はサポートではない．論理関数 F が変数 x に依存しているかどうかは，F の x に対するコファクタ (F に $x = 0$ または $x = 1$ を代入してできる論理関数) を求めて，それを比較することで判断できる．変数 x に依存している場合には，2 つのコファクタは等しくならない．

もしも 2 つの関数 F と G のサポートが共通部分を持たない時，すなわち，$\mathcal{S}(F) \cap \mathcal{S}(G) = \phi$ で

ある時，F と G は直交する，と言い，$F \perp G$ と表す．

2.3 Davio 展開

論理関数 F は以下のように展開される．

$$F = F_{\overline{x}} \oplus x \cdot F_{\overset{*}{x}} \quad (2)$$

ここで，

$$F_{\overset{*}{x}} = F_{\overline{x}} \oplus F_x \quad (3)$$

である．

この展開は正 **Davio expansion** と呼ばれる．

3 互いに素なサポートを持つ XOR 分解

3.1 基本概念

論理関数 F が以下のように互いに素なサポートを持つ 2 つの関数に分解されると仮定する．

$$F = G \oplus H \; (G \perp H) \quad (4)$$

この時，$x \in \mathcal{S}(G)$ であり，$x \notin \mathcal{S}(H)$ であるような変数 x が必ず一つは存在する．(4) 式の両辺にそのような変数 x による正 Davio 展開を施すと以下の式を得る．

$$\begin{aligned} F_{\overline{x}} \oplus x \cdot F_{\overset{*}{x}} &= (G_{\overline{x}} \oplus x \cdot G_{\overset{*}{x}}) \\ &\quad \oplus (H_{\overline{x}} \oplus x \cdot H_{\overset{*}{x}}) \end{aligned} \quad (5)$$

H は x に対して独立であるから，

$$H_{\overline{x}} = H \quad (6)$$

$$H_{\overset{*}{x}} = 0 \quad (7)$$

となる．この (7) 式を用いると (5) 式は以下のようになる．

$$F_{\overline{x}} \oplus x \cdot F_{\overset{*}{x}} = (G_{\overline{x}} \oplus H) \oplus x \cdot G_{\overset{*}{x}} \quad (8)$$

(8) 式は x の値に関わらず成立しなければならないので，両辺の各項を比較して，以下の関係を得る．

$$F_{\overline{x}} = G_{\overline{x}} \oplus H \quad (9)$$

$$F_{\overset{*}{x}} = G_{\overset{*}{x}} \quad (10)$$

つまり，$F_{\overline{x}}$ が互いに素なサポートを持つ分解を持ち (それを A と B とする)，その一方の関数 (B) が $F_{\overset{*}{x}}$ に対して直交しているのなら (すなわち，$F_{\overset{*}{x}} \perp B$)，$F$ は互いに素なサポートを持つ関数分解 $F = G \oplus H$ を持つ．ここで，

$$G = A \oplus x \cdot F_{\overset{*}{x}} \quad (11)$$

$$H = B \quad (12)$$

である．

3.2 互いに素なサポートを持つ関数分解を求めるアルゴリズム

(11) 式と (12) 式を利用して，互いに素なサポートを持つ関数分解を行なうアルゴリズムの開発を行った．疑似コードを図 2 に示す．

```
Set-Of-Function xdecomp(Function F) {
        D ← φ
1:      while (F ≠ 0) {
2:          x ← one of S(F)
3:          H ← xdecomp1(F, {x})
4:          G ← F ⊕ H
5:          D ← D ∪ {G}
6:          F ← H
        }
7:      return D
}

Function xdecomp1(Function F,
                  Set-Of-Variables X) {
8:      Y ← X ∩ S(F)
9:      if (Y = φ)
10:         return F
11:     x ← one of Y
12:     Y ← Y − {x}
13:     H ← xdecomp1(F_x̄, Y ∪ S(F_*x))
14:     return H
}
```

図 2: 互いに素なサポートを持つ関数への XOR 分解アルゴリズム

サブルーティン **xdecomp1** は，関数 F を以下

のように分解する．

$$F = H \oplus G \quad (13)$$
$$\mathcal{S}(H) \cap X = \phi \quad (14)$$

部分関数Gはxをサポートとして含み，それ以上はXOR分解できない最小の関数である．もしも，$G = \bigoplus G_i$となるような素なサポートを持つ関数の分割が存在したとすると，その中のG_iのうち少なくとも一つ以上はxには依存しないこととなり矛盾を生ずる．Gと異なり，Hはさらに分解される可能性があるので，Hに対する分解を行なうために**xdecomp1**は再帰的に呼び出される．この再帰(繰り返し)はHが定数になるまで続けられる．**xdecomp1**の求める関数分解はその引数Xに応じて異なったものとなるが，どのような変数を最初に与えたとしても最終的に**xdecomp**の返す結果は唯一となる．つまり，**xdecomp**と**xdecomp1**中の"one of"という処理においてどのような変数を選んでも結果には影響しない[4]．

例題:

次のような関数の分解を考える．

$$F = a \cdot \overline{b} \cdot \overline{d} + \overline{b} \cdot c \cdot \overline{d} + \overline{a} \cdot \overline{c} \cdot d + \overline{a} \cdot b \cdot \overline{c} \quad (15)$$

line 2: aを展開用の変数として選ぶ．

line 3: **xdecomp1**$(F, \{a\})$を呼ぶ．

line 8: $Y \leftarrow \{a\}$.

line 11: aをxとして選ぶ

line 12: $Y \leftarrow \Phi$.

line 13: 正Davio展開を行なう．

$$F_{\overline{a}} = \overline{b} \cdot c \cdot \overline{d} + \overline{c} \cdot d + b \cdot \overline{c}$$
$$F_{\dot{a}} = \overline{c}$$

次に，**xdecomp1**$(F_{\overline{a}} = F', \{c\})$を呼ぶ．

line 8: $Y \leftarrow \{c\}$.

line 11: cをxとして選ぶ．

line 12: $Y \leftarrow \Phi$.

line 13: 正Davio展開を行なう．

$$F'_{\overline{c}} = b + d$$
$$F'_{\dot{c}} = 1$$

続いて**xdecomp1**$(b + d, \Phi)$を呼ぶ．$Y = \Phi$なので，この再帰はここで止まる．

line 14: $b + d$を戻り値として返す．

line 14: $b + d$を戻り値として返す．

line 4:

$$G \leftarrow F \oplus (b + d) = a + c$$

line 5: $D \leftarrow \{(a + c)\}$.

line 6: $F \leftarrow b + d$. aによる正Davio展開を適用する．

line 2,3: bかdを選ぶ．結果として$H = 0$を得る．

line 4,5: $G \leftarrow b+d$かつ，$D \leftarrow \{(a+c), (b+d)\}$となる．

最終的に$F = (a + c) \oplus (b + d)$を得る． □

3.3 二分決定グラフを用いた実装とその計算複雑度

論理関数を表現するためのデータ構造として二分決定グラフ(Binary Decision Diagrams:BDD's)[5]を用いることで，図2のアルゴリズムは効率良く実行することが可能である．計算複雑度は以下のように算定される．

- 全体として，**xdecomp1**は$O(k)$回実行される．ここで，kはFの変数の数である．**xdecomp1**の13行めにおいて，$F_{\overline{x}}$が次の再帰呼び出しの第1引数(F)として用いられている．つまり，$|\mathcal{S}(F)|$は再帰するたびに1以上，減っていることになる．このことは，もし**xdecomp1**がN回呼ばれて，戻り値として関数Hを返したとしたら，$|\mathcal{S}(H)| \leq |\mathcal{S}(F)| - N$であることを意味している．このことより，以下なる場合においても**xdecomp1**の呼び出される回数Nはkを越えないことが分かる．
- **xdecomp1**の計算複雑度は$O(n^2)$である．ここで，nはFを表す二分決定グラフのサイズである．**xdecomp1**中で，必要とされ

[4] しかし，効率には影響する．

る論理関数演算は次の2つである．一つは Shannon 展開であり，もう一つは XOR 演算である ($F_* = F_{\overline{x}} \oplus F_x$ であるので)．もしも，展開する変数として，二分決定グラフの変数順で最も上位にある変数を選ぶものとすると，Shannon 展開は $O(1)$ で実行でき，その結果，$O(n)$ のサイズを持つ2つのコファクタを得ることができる．その $O(n)$ のサイズを持つ二分決定グラフの XOR 演算の計算複雑度は $O(n^2)$ である．

- 各々の **xdecomp1** の呼び出しにおいて，F を表す二分決定グラフのサイズは増加しない．毎回，$F_{\overline{x}}$が新しい F として呼び出されているからである．

以上の考察により，**xdecomp** の計算複雑度は $O(n^2 \times k)$ である．

もしも二分決定グラフの代わりに Functional Decision Diagrams(FDD's)[6] を用いれば，論理関数を表す FDD のサイズを n，変数の数を k として，計算複雑度は $(n \times k)$ となる．しかし，同一の論理関数を表す FDD のサイズと BDD のサイズは同一ではなく，一方が他方の指数倍大きくなる例が (両方とも) 存在する．

4 1変数のみ重複したサポートを持つ関数による XOR 分解

アルゴリズム **xdecomp** を用いることによって，ただ一つの変数をサポートとして共有する分解 ($F = G \oplus H : \mathcal{S}(G) \cap \mathcal{S}(H) = \{x\}$) を求めることができる．

そのような分解が存在したとすると，変数 x によって正 Davio 展開を行ない，

$$F_{\overline{x}} \oplus x \cdot F_{\underset{x}{*}} = (G_{\overline{x}} \oplus x \cdot G_{\underset{x}{*}}) \oplus (H_{\overline{x}} \oplus x \cdot H_{\underset{x}{*}}) \quad (16)$$

$$= (G_{\overline{x}} \oplus H_{\overline{x}}) \oplus x \cdot (G_{\underset{x}{*}} \oplus H_{\underset{x}{*}}) \quad (17)$$

という式を得る．各々の項を比較すると，

$$F_{\overline{x}} = G_{\overline{x}} \oplus H_{\overline{x}} \quad (18)$$

$$F_{\underset{x}{*}} = G_{\underset{x}{*}} \oplus H_{\underset{x}{*}} \quad (19)$$

となる．そこで，もしも $F_{\overline{x}}$ と $F_{\underset{x}{*}}$が素なサポート分解 $U_1 \oplus U_2$と $V_1 \oplus V_2$を持ち，$U_1 \perp V_2$ かつ $U_2 \perp V_1$であるならば，F は変数 x のみを共通のサポートとして持つ2つの関数に XOR 分解できる．ここで，

$$G = U_1 \oplus x \cdot V_1 \quad (20)$$

$$H = U_2 \oplus x \cdot V_2 \quad (21)$$

である．

図3に1変数を共通のサポートとして持つ関数分解のアルゴリズムを示す．

Pair-Of-Functions **xovdecomp**(Function F,
　　　　Variable x) {
　$U = \{u_1, u_2, \ldots\} \leftarrow$ **xdecomp**($F_{\overline{x}}$)
　$V = \{v_1, v_2, \ldots\} \leftarrow$ **xdecomp**($F_{\underset{x}{*}}$)
　/* U と V を $U_G : U_H$ と $V_G : V_H$ に分割する．*/
　各々の節点が u_iと v_jに対応したグラフを作る．
　foreach $u \in U$ {
　　foreach $v \in V$ {
　　　if $u \not\perp v$ then
　　　　u と v に枝を張る．
　　}
　}
　$U_G \leftarrow \{u | u$ は u_1 と (間接的に) 接続している.$\}$
　$U_H \leftarrow U - U_G$
　$V_G \leftarrow \{v | v$ は u_1 と (間接的に) 接続している.$\}$
　$V_H \leftarrow V - V_G$
　$G \leftarrow \bigoplus U_G \oplus x \cdot \bigoplus V_G$
　$H \leftarrow \bigoplus U_H \oplus x \cdot \bigoplus V_H$
　return (G, H)
}

図 3: 1変数を共通のサポートとして持つ XOR 分解のアルゴリズム

最初に $F_{\overline{x}}$と $F_{\underset{x}{*}}$に対する素なサポートを持つ XOR 分解 U および V を求める．続いて，U および V を次のように分割する．

$$\forall u \in U_G, \forall v \in V_H, u \perp v \quad (22)$$

$$\forall u \in U_H, \forall v \in V_G, u \perp v \quad (23)$$

このような分割を求めるために，各々の節点が u および v に対応したグラフを作り，共通のサポートを持つ節点の間に枝を設ける．このようにして作られたグラフ上で，異なる連結成分に属してい

る節点は互いに共通なサポートを持たないことがわかる．そこで，u_1と同一の連結成分に含まれるuの節点の集合をU_G，残りをU_Hとして，同様に，u_1と同一の連結成分に含まれるvの節点の集合をV_G，残りをV_Hとする．このような分割が必ずしも存在するとは限らない．その場合には，U_HもしくはV_Hがϕとなる．

例題:

次のような関数$F=(af+g+e)\oplus(bc+\overline{a}d)$をXOR分解することを考えてみる．この関数を素なサポートを持つ関数にXOR分解することはできないので，1変数を共有する関数に分解することを試みる．もしも，aをそのような重複した変数として選ぶとすると，aで正Davio展開を行なって，

$$F_{\overline{a}} = (g+e)\oplus(bc+d) \quad (24)$$

$$F_{\underset{a}{*}} = (f+g+e)\oplus bc \oplus(g+e)\oplus(bc+d) \quad (25)$$

$$= f\overline{g}\,\overline{e}\oplus d(\overline{b}+\overline{c}) \quad (26)$$

を得る．以下，読みやすさのために各項を以下のように表すものとする．

$$g+e = u_1 \quad (27)$$

$$bc+d = u_2 \quad (28)$$

$$f\overline{g}\,\overline{e} = v_1 \quad (29)$$

$$d(\overline{b}+\overline{c}) = v_2 \quad (30)$$

ここで，以下のようなグラフを作る．$G(V,E)$，$V=\{u_1,u_2,v_1,v_2\}$，$E=\{(u_1,v_1),(u_2,v_2)\}$ (図4)．

図 4: 分割グラフの例

このグラフから，$U_G=\{u_1\}$，$U_H=\{u_2\}$，$V_G=\{v_1\}$，$V_H=\{v_2\}$を得る．最終的に，GとHは以下のようになる．

$$G = (g+e)\oplus a\cdot f\overline{g}\,\overline{e} = af+g+e \quad (31)$$

$$H = (bc+d)\oplus a\cdot d(\overline{b}+\overline{c}) = \overline{a}d+bc \quad (32)$$

よって，Fは以下のように分解される．$F=(af+g+e)\oplus(\overline{a}d+bc)$． □

5 実験結果

提案したアルゴリズムは二分決定グラフを用いて比較的簡単に実装可能であるが，**xdecomp**や**xovdecomp**単独では，ファクタリング処理そのものを行うことはできない．そこで，**xdecomp**と**xovdecomp**のパフォーマンスを見るために以下のような基礎的な実験を行なった．

- K入力のすべてのNPN同値類の関数を列挙する ($K\le 5$)．
- 各々の関数に対して**xdecomp**を適用する．
- 各々の関数に対して**xovdecomp**を適用する．

表1と表2にその結果を示す．

入力数	3	4	5
関数の数	10	208	615904
分解された関数の数	2	11	216
CPU時間の総計 (秒)	0.00	0.06	202.09
CPU時間/関数 (ミリ秒)	0.00	0.29	0.32

表 1: **xdecomp**を用いた実験結果

入力数	3	4	5
関数の数	10	208	615904
分解された関数の数	4	21	751
CPU時間の総計 (秒)	0.00	0.27	1276.67
CPU時間/関数 (ミリ秒)	0.00	1.30	2.07

表 2: **xdecomp**を用いた実験結果

使用計算機はFujitsu S-7/300[5]である．CPU時間に関しては，**xdecomp**，**xovdecomp**ともに良好な結果を得ている．これらの結果は提案したアルゴリズムの効率の良さを示しているといえる．すべてのNPN同値類のうちで，**xdecomp**や

[5] SPECint92 252

xovdecomp で分解できる関数の数はそれほど多くはないが，このことは直ちに提案した XOR 分解が無用であることを導くものではない．今回の実験は与えられた関数が XOR 分解できるかどうかを調べるものであって，実際のファクタリングにおいては従来の AND/OR 分解と組み合わせて用いた場合の効果を示したものではない．実際の論理合成処理の中で提案した XOR 分解の効果を確かめる実験を行うことが今後の課題である．

6 おわりに

本稿では，互いに素なサポートを持つ関数への分解と 1 つの変数を共通なサポートとして持つ関数への分解を行なうアルゴリズムを提案した．もしも，二分決定グラフを用いて元の論理関数を表現できるのであれば，これらのアルゴリズムは非常に効率良く実行可能である．これらの XOR 分解はファクタリング処理のほんの一部分であるが，これらと従来の AND/OR を用いたファクタリングアルゴリズムと組み合わせることによって，より適切なファクタードフォームをつくり出すことが可能であると思われる．

従来，ESOP の簡単化や AND-OR-XOR の簡単化などの高い規則性を持った論理式の簡単化に関しては有効な手法がいくつか提案されているが [7]，多段論理合成の枠組の中で XOR 演算子を適切に扱えるものはほとんど提案されていない．これらの理論的な基礎を固めると共に実用的なアルゴリズム/ヒューリスティックを開発することが重要と思われる．

参考文献

[1] K. Keutzer, "DAGON:Technology Binding and Local Optimization by DAG Matching", in *Proc. 24th Design Automation conf.*, pp. 341–347, June 1987.

[2] 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.

[3] G.D. Hachtel and F. Somenzi, "Logic Synthesis and Verification Algorithms", Kluwer Academic Publishers, 1996.

[4] J.M. Saul, "Toward a mixed exclusive-/inclusive orfactored form", *IFIP Workshop on Applications on Reed-Muller Expansions in Circuit Design*, pp. 2–5, 1992.

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

[6] U. Kebschull, E. Schubert and W. Rosenstiel, "Multilevel Logic Synthesis Based on Functional Decision Diagrams", *IEEE EDAC'92*, pp. 43–47, Mar. 1992.

[7] T. Sasao, "Logic synthesis with XOR gates", Logic Synthesis and Optimization, Kluwer Academic Publishers, pp. 259–285, 1993.