設 計 自 動 化 85－21
（1997. 10. 29）

対称変数の検出による関数分解の高速化と多段論理合成への応用

澤田 宏，山下 茂，名古屋 彰

NTT コミュニケーション科学研究所
〒619-02 京都府相楽郡精華町光台2丁目
Tel: 0774-95-1866
Fax: 0774-95-1876
E-mail: {sawada, ger, nagoya}@cslab.kecl.ntt.co.jp

あらまし

本稿では，関数分解 $f(X) = g(h(X^B), X^F)$ の中でも特に，変数集合 X^B と X^F が共通の変数を持たず，h が1出力関数である simple disjunctive decomposition の高速化について議論する．具体的には，検出する分解の形を，1) 変数集合 X^B が対称変数集合である場合，2) 分解後の関数 g の入力数が2である場合，に絞り処理の高速化を図る．また，積和形論理式の多段化に simple disjunctive decomposition を用いることについても議論する．

キーワード　関数分解，simple disjunctive decomposition，対称変数，順序付き二分決定グラフ，多段論理合成

Efficient Simple Disjunctive Decompositions by Detecting Symmetric Variables with Application to Multi-level Logic Synthesis

Hiroshi SAWADA, Shigeru YAMASHITA and Akira NAGOYA

NTT Communication Science Laboratories
2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, JAPAN
Tel: 0774-95-1866
Fax: 0774-95-1876
E-mail: {sawada, ger, nagoya}@cslab.kecl.ntt.co.jp

Abstract

Simple disjunctive decomposition is a special case of a functional decomposition $f(X) = g(h(X^B), X^F)$, where variables are divided into two disjoint sets X^B and X^F, and h is a single-output function. This paper presents that simple disjunctive decompositions, 1) where X^B is a set of symmetric variables, and 2) where g is a 2-input function, can be easily found. We also discuss the application of simple disjunctive decompositions to multi-level logic synthesis.

key words　functional decomposition, simple disjunctive decomposition, symmetric variables, ordered binary decision diagram, multi-level logic synthesis

1 はじめに

論理関数の分解は，多段論理合成において重要な要素技術である．一般に，論理関数 f を $f(X) = g(h(X^B), X^F)$ のように変形することを関数分解 (functional decomposition) と呼ぶ．その中でも，変数集合 X が2つの重なりのない変数集合 X^B と X^F に分けられ，しかも h が1出力関数であるものは，simple disjunctive decomposition と呼ばれる [1, 2, 3]．本稿では，関数分解の中でも特に，この simple disjunctive decomposition について議論する．以下，文脈から明白な場合は，これを単に分解と呼ぶことにする．このような分解は，変数集合を2つの独立なものに分け，それらを結ぶ結線がただ1本であるという点で，非常に性質の良いものである．1出力関数の最適な多段論理表現を求める場合，そのような分解が見つかれば，これを適用すべきである．

関数 f と変数集合の分割 X^B，X^F に対して simple disjunctive decomposition が存在するための条件は，変数集合 X^B に定数値 $\{0,1\}^{|X^B|}$ を割り当てた結果のすべての関数が，ただ2種類だけになるということである．近年，変数集合 X^B が X^F よりも先に展開されるような変数順で，関数 f の順序付き二分決定グラフ (ordered binary decision diagram, OBDD)[4] を構築することにより，分解のための条件を効率的に判定する方法 [5] が提案された．この方法により，ある与えられた X^B に対しては，分解が存在するかどうかが効率的に調べられるようになったが，分解を与える変数集合 X^B を見つけることは，依然として難しい問題である．可能な X^B の組合せの数は，関数の変数の数に対して指数関数的に増大するため，すべての組合せに対して調べることは，大規模な論理関数に対しては現実的ではない．

そこで我々は，大規模な論理関数にも適用できるようにするため，検出する simple disjunctive decomposition の形を以下の2種類に絞り，処理の高速化を図る．

1. 変数集合 X^B が対称変数集合である場合
2. 分解後の関数 g の入力数が2である場合

これらの形に当てはまらないものでも，これらの分解を再帰的に適用することで，発見できる可能性がある．

対称変数集合とは，その集合の中で変数を入れ替えても，もとの論理関数が変化しない変数集合のことを言う．完全指定論理関数の場合，2変数の対称性は同値関係であるため，すべての2変数ペアの対称性を調べることで，対称変数集合を求めることができる．論理関数の2変数の対称性を調べることは，大規模な論理関数に対しては，比較的計算コストのかかることであった．しかし，逆に，非対称な2変数ペアのいくつかを OBDD の形から検出する方法が近年提案され [6]，対称性の計算の効率化が可能になった．本稿では，彼らの手法を改良し，さらに効率的に2変数の対称性を検出する方法を提案する．また，1番目に挙げた分解の形において，対称変数集合がさらに強い条件を満たせば，X^B が X^F よりも先に展開されるような変数順の OBDD を構築することなく分解の存在が分かることも示す．

2番目に挙げた分解後の関数 g の入力数が2である場合は，関数 f の OBDD を構築して，ある変数のすべてのノードがある条件を満たすかどうかで調べることができる．なお，この形は，関数を $f(X) = g(h_1(X_1), h_2(X_2))$ の形 (X_1 と X_2 は重なりのない変数集合) に分解する bi-decomposition [7, 8] の特別な形である．それゆえ，本稿で述べるアルゴリズムは，より単純なものとなっている．

以下では，まず2章で用語を定義し，上記の各手法の詳細を3章で述べる．4章では，多段論理合成における応用として，代数的手法による積和形論理式の多段化 [9] の前処理に simple disjunctive decomposition を用いることについて議論する．5章では実験結果と考察を示し，6章で結論を述べる．

2 準備

2.1 Simple disjunctive decomposition

論理関数 $f(X)$: $\{0,1\}^{|X|} \rightarrow \{0,1\}$ を以下の形式で表現できるとき，simple disjunctive decom-

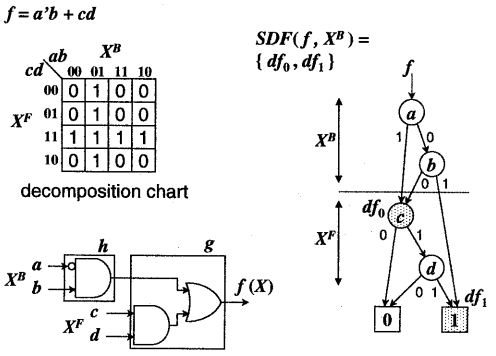

図 1: Simple disjunctive decomposition

position が存在するという．

$$f(X) = g(h(X^B), X^F),$$

ここで，X^B と X^F は，$X^B \cup X^F = X$ かつ $X^B \cap X^F = \emptyset$ を満たす変数の集合であり，g: $\{0,1\}^{|X^F|+1} \to \{0,1\}$ と h: $\{0,1\}^{|X^B|} \to \{0,1\}$ は論理関数である．集合 X^B と X^F はそれぞれ，**bound set**，**free set** と呼ばれる．また，g は分解の**像**と呼ばれ，本稿では h を分解の**部分関数**と呼ぶことにする．

Ashenhurst [1] は，simple disjunctive decomposition が存在するかどうかを調べるために，分解表 (decomposition chart) というものを用いた．これは，列と行にそれぞれ X^B と X^F を対応させた真理値表である．もし，その列に現れるベクトルの種類が 2 であれば，上記の分解が存在することになる．本稿では，関数 f と変数集合 X^B に対し，$SDF(f, X^B) = \{df \mid df = f(\epsilon, X^F), \forall \epsilon \in \{0,1\}^{|X^B|}\}$ なるものを定義する (SDF: Set of Distinct Functions)．これは，分解表における異なる列ベクトルの集合に相当する．Lai ら [5] は，関数 f に対して，X^B の変数が X^F の変数よりも先に展開されるような変数順の OBDD を作ることで $SDF(f, X^B)$ を効率良く計算する技法を提案した．分解が存在する場合，部分関数 h は，関数 f の df_0 と df_1 をそれぞれ 0 と 1 に置き換えることで得られる (ただし，$SDF(f, X^B) = \{df_0, df_1\}$)．また，分解の像 g は，$g = df_0 \cdot h' + df_1 \cdot h$ で与えられる．

例えば，図 1に示す関数 $f = a'b + cd$ に対し，X^B を $\{a, b\}$ とすると，$SDF(f, \{a, b\}) = \{cd, 1\}$ であり，その大きさは 2 であるから simple disjunctive decomposition が存在する．部分関数 h は，df_0 と df_1 をそれぞれ 0 と 1 に置き換えることで得られるので，$h = a'b$ である．像は $g = cd \cdot h' + 1 \cdot h = cd + h$ となる．

2.2 対称変数集合

変数 x の否定を x' のように表記する．関数 f の変数 x_i に関する 2 つの**コファクタ (cofactor)** は，$f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ と $f_{x_i'} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ で与えられる．

関数 $f(x_1, \ldots, x_n)$ が，変数 x_i と x_j(あるいは x_i と x_j') を入れ替えても変化しないとき，$\{x_i, x_j\}$(あるいは $\{x_i, x_j'\}$) に関して**対称 (symmetric)** であるという．関数 f が，$\{x_i, x_j\}$ (あるいは $\{x_i, x_j'\}$) に関して対称であるための必要十分条件は，$f_{x_i x_j'} = f_{x_i' x_j}$ (あるいは $f_{x_i x_j} = f_{x_i' x_j'}$) である．

完全指定論理関数においては，対称性は同値関係である．従って，変数集合に関する対称性は，2 変数の対称性の集合から計算できる．そのような変数集合を**対称変数集合**と呼ぶことにする．例えば，関数 $f = ab'c + d$ は $\{a, b'\}$ や $\{b', c\}$ に関して対称であり，その結果 $\{a, b', c\}$ は対称変数集合となる．

さらに本稿では，[10] に見られる以下の定義を用いる．関数 f が $\{x_i, x_j\}$ に関して対称であり，かつ $\{x_i, x_j'\}$ に関しても対称であるとき，変数 x_i と x_j に関して **multiform symmetric** であるという．また，関数 f において，f_{x_j} の x_i に関する 2 つのコファクタが同じ，すなわち $f_{x_i' x_j} = f_{x_i x_j}$ であるとき，空間 $x_j = 1$ において x_i に関して **single-variable symmetric** であるという．同様に，$f_{x_j'}$ の x_i に関する 2 つのコファクタが同じ，すなわち $f_{x_i' x_j'} = f_{x_i x_j'}$ であるとき，空間 $x_j = 0$ において x_i に関して **single-variable symmetric** であるという．

もし，f が対称変数集合を持ち，その集合のある 2 変数ペアに関して multiform symmetric であれば，f はその集合のすべての 2 変数ペアに関し

て multiform symmetric である．同様に，ある2変数ペアに関して single-variable symmetric であれば，f はその集合のすべての2変数ペアに関して single-variable symmetric である．

3 Simple disjunctive decomposition の高速化

3.1 X^B が対称変数集合である場合

3.1.1 OBDD を用いた対称変数集合の検出

まず，関数 f の対称変数集合を OBDD を用いて調べる．OBDD において，より先に展開される変数の方がレベルが大きいもとのし，レベルから変数添字への写像 π で変数順を表すものとする．2変数の対称性は同値関係であるため，関数 f の対称変数集合を完全に調べるには，すべての $x_{\pi(i)}(n \geq i \geq 2,\ n$ は f の変数の数) に対して，f が $\{x_{\pi(i)}, x_{\pi(j)}\}$ あるいは $\{x_{\pi(i)}, x'_{\pi(j)}\}$ に関して対称である変数 $x_{\pi(j)}(i > j)$ が存在するかどうかを調べれば良い．

OBDD を用いた場合，レベルが隣接している2変数の対称性は，簡単に調べることができる．すなわち，レベル i のすべてのノード v において，$f^v_{x_{\pi(i)}x'_{\pi(i-1)}} = f^v_{x'_{\pi(i)}x_{\pi(i-1)}}$ あるいは $f^v_{x_{\pi(i)}x_{\pi(i-1)}} = f^v_{x'_{\pi(i)}x'_{\pi(i-1)}}$ が満たされれば，$\{x_{\pi(i)}, x_{\pi(i-1)}\}$ あるいは $\{x_{\pi(i)}, x'_{\pi(i-1)}\}$ に関して対称である．

一方，レベルが大きく離れている2変数の対称性を調べることは，それほど簡単ではない．そのため，文献 [6] では，非対称である2変数ペアのいくつかを OBDD の形から求める手法を提案している．これを用いることで，手間をかけて調べる2変数ペアの数を減らせることができる．

定理 1 ([6]) 関数 f が変数順 π の OBDD で表現されているとする．以下の2つの条件のどちらかが満たされるとき，f は変数 $x_{\pi(i)}$ と $x_{\pi(j)}\ (i > j)$ に関して非対称である．

1. レベル i のあるノードが表す関数が，変数 $x_{\pi(j)}$ に依存していない．
2. 根ノードからレベル j のノードに，レベル i のノードを含まない経路で到達できる． □

1の場合は，変数 $x_{\pi(i)}$ に依存する関数 f の一部分が，変数 $x_{\pi(j)}$ に依存していないことがわかる．2の場合は，変数 $x_{\pi(j)}$ に依存する関数 f の一部分が，変数 $x_{\pi(i)}$ に依存していないことがわかる．いずれの場合も，変数 $x_{\pi(i)}$ と $x_{\pi(j)}$ を入れ替えると，もとの関数とは違ったものになるので，これらの2変数は非対称である．なお，これら2つの条件は，各ノードの関数が依存する変数集合を調べておき，根ノードから深さ優先探索を行うことで，簡単に調べることができる．

定理1によって非対称であると判定されなかった2変数ペアに関しては，手間をかけてその対称性を調べる必要がある．一般には，$f_{x_{\pi(i)}x'_{\pi(j)}}$ と $f_{x'_{\pi(i)}x_{\pi(j)}}$ の OBDD を構築して，その一致判定を行う．しかし本稿では，この処理も高速化するため，以下の定理に基づいて対称性を調べる手法を提案する．

定理 2 関数 f が変数順 π の OBDD で表現されているとする．レベル i のすべてのノード v が，$\forall b_k \in \{x_{\pi(k)}, x'_{\pi(k)}\}$，$i-1 \leq k \leq j+1$ に対して，$f^v_{x_{\pi(i)}b_{i-1}\cdots b_{j+1}x'_{\pi(j)}} = f^v_{x'_{\pi(i)}b_{i-1}\cdots b_{j+1}x_{\pi(j)}}$ あるいは $f^v_{x_{\pi(i)}b_{i-1}\cdots b_{j+1}x_{\pi(j)}} = f^v_{x'_{\pi(i)}b_{i-1}\cdots b_{j+1}x'_{\pi(j)}}$ を満たすとき，f は $\{x_{\pi(i)}, x_{\pi(j)}\}$ あるいは $\{x_{\pi(i)}, x'_{\pi(j)}\}\ (i > j)$ に関して対称である． □

これも OBDD における深さ優先探索で調べることができる．その際に，上記の条件が少しでも満たされなければ，非対称であることがわかるので，探索をすぐに止めることができる．

3.1.2 XOR ゲートを用いた分解

関数 f のある対称変数集合 S が，multiform symmetric である2変数を含んでいれば，2.2章で述べたように，S 内のすべての2変数ペアに関して multiform symmetric である．従って，各2変数ペア $a, b \in S$ に対し，$f_{ab'} = f_{a'b}$ かつ $f_{ab} = f_{a'b'}$ が成立する．この場合，$SDF(f, S)$ の大きさは2であり，simple disjunctive decomposition が存在することがわかる．分解の部分関数は，$SDF(f, S)$ の2つの要素を0と1で置き換えることで得られるが．これは XOR ゲートで実現できる．

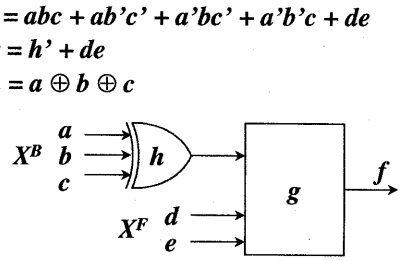

図 2: XOR ゲートを用いた分解

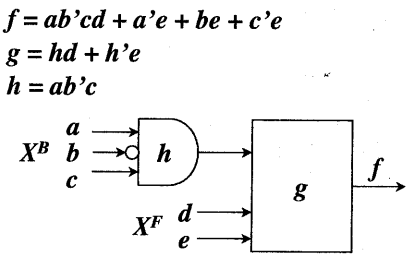

図 3: AND ゲートを用いた分解

例として，図2に示す $f = abc + ab'c' + a'bc' + a'b'c + de$ を考える．対称変数集合は $\{a, b, c\}$ であり，$f_{ab'} = f_{a'b} = c' + de$，$f_{ab} = f_{a'b'} = c + de$ であることから，$\{a, b, c\}$ のすべての2変数ペアに関して multiform symmetric である．従って，simple disjunctive decomposition が存在する．分解の部分関数は，$h = a \oplus b \oplus c$ で与えられ，これは XOR ゲートで実現できる．

3.1.3 AND ゲートを用いた分解

関数 f のある対称変数集合 S が，single-variable symmetric である2変数を含んでいれば，2.2 章で述べたように，S 内のすべての2変数ペアに関して single-variable symmetric である．ここで，各2変数ペア $a, b \in S$ が，$f^N_{a'b'} = f^N_{ab'} = f^N_{a'b}$ を満たすような関数 f^N を考える．これは，関数 f において，S のいくつかの変数の極性を反転させることで得られる．この場合，$SDF(f^N, S)$ の大きさは2であり，simple disjunctive decomposition が存在することがわかる．分解の部分関数は，$SDF(f^N, S)$ の2つの要素を0と1で置き換えることで得られるが，これは AND ゲートで実現できる．従って，f には，AND ゲートといくつかの NOT ゲートで部分関数が実現できるような simple disjunctive decomposition が存在する．

例として，図3に示す $f = ab'cd + a'e + be + c'e$ を考える．対称変数集合は $\{a, b', c\}$ であり，$f_{a'b'} = f_{a'b} = f_{ab} = e$ であることから，$\{a, b', c\}$ のすべての2変数ペアに関して single-variable symmetric である．ここで，変数 b の極性を反転した関数 $f^N = abcd + a'e + b'e + c'e$ を考えると，$f^N_{a'b'} = f^N_{a'b} = f^N_{ab'} = e$ となり，simple disjunctive decomposition が存在する．分解の部分関数は，$h = abc$ であり，AND ゲートを用いて実現できる．NOT ゲートを用いて変数 b の極性を戻すことで，もとの関数 f の分解の形が得られる．

3.1.4 対称関数を用いた分解

関数 f のすべての対称変数集合 S が，multiform symmetric や single-variable symmetric である2変数を含んでいなければ，$SDF(f, S)$ を計算して分解の存在を調べる必要がある．これは，2.1 章で述べたように，OBDD の変数順を変更して計算する．もし simple disjunctive decomposition が存在すれば，その部分関数は一般的な対称関数となる．

3.2 分解の像が2入力関数である場合

次に，分解の像 g が2入力関数である simple disjunctive decomposition も簡単に検出できることを示す．これに当てはまる分解の形とそのための条件は，以下に示す5種類に分類できる．

- $f_{x'} = (f_x)'$ ならば，$f = x \oplus f_{x'}$ と分解できる．
- $f_{x'} = 0$ ならば，$f = x \cdot f_x$ と分解できる．
- $f_x = 0$ ならば，$f = x' \cdot f_{x'}$ と分解できる．
- $f_{x'} = 1$ ならば，$f = x' + f_x$ と分解できる．
- $f_x = 1$ ならば，$f = x + f_{x'}$ と分解できる．

明らかに，$f_{x'}$ と f_x は x に依存していないので，これらの分解は simple disjunctive decomposition

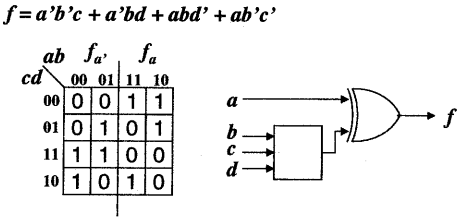

図 4: 分解の像が 2 入力関数である分解

である.

例として，図4に示す関数 f について考える．変数 a の2つのコファクタはそれぞれ $f_{a'} = b'c + bd$ と $f_a = bd' + b'c'$ であり，$f_{a'} = (f_a)'$ が成り立つ．従って，f は $f = a \oplus (b'c + bd)$ の形の simple disjunctive decomposition を持つ．

このような形の分解が存在するかどうかを調べるためには，関数 f のすべての変数 x について，上記の5つの条件のどれかを満たすかどうかを調べれば良い．これは，関数 f の OBDD を構築し，変数 x のすべてのノード v に対して，それが表す関数 f^v が，$f^v_{x'} = (f^v_x)'$，$f^v_{x'} = 0$，$f^v_x = 0$，$f^v_{x'} = 1$，$f^v_x = 1$ のどれかを満たすかどうかを調べればよい．ここで，$f^v_{x'}$ と f^v_x は，v の0枝と1枝が指し示すノードによって表現されている．OBDD に否定枝が導入されている場合は，$f^v_{x'} = (f^v_x)'$ かどうかを判定することは定数時間で可能である．また，$f^v_{x'} = 0$，$f^v_x = 0$，$f^v_{x'} = 1$，$f^v_x = 1$ に関しては，定数ノードとの比較なので，これも定数時間で行える．従って，ここで述べた形の分解は，簡単に検出できると言える．

4 Simple disjunctive decomposition を用いた多段論理回路合成

本章では，積和形論理式を多段化する手法として幅広く用いられている代数的手法 [9] の前処理として，simple disjunctive decomposition を用いることを考える．

1出力関数の多段回路合成を考えた場合，simple disjunctive decomposition が存在すればすぐにそれを適用し，存在しなければ既存の代数的手法等で分解していく，という手法で良い多段回路が得られると考えられる．しかし，多出力関数の場合は，必ずしもそうとは言えない．簡単な例を示すと，$f_1 = ab + ac$，$f_2 = ab$，$f_3 = ac$ の場合，最適な多段回路は，総リテラル数6の $f_1 = f_2 + f_3$，$f_2 = ab$，$f_3 = ac$ であるが，f_1 に存在する simple disjunctive decomposition を適用すると，$x = b + c$，$f_1 = ax$，$f_2 = ab$，$f_3 = ac$ となり，総リテラル数8の多段回路となってしまう．

また，分解を必要以上に行うことは，以下の理由からも不利である．関数の分解を行ったとき，その部分関数の出力に新たな変数を割り当てていく．この変数は本来独立な変数ではなく，外部入力変数に依存するものであるため，この変数を用いている他の関数では，いくらかのドントケア条件が発生していることがある．しかし，代数的手法ではこれを独立な変数として扱うため，そのようなドントケア条件は認識されない．

以上のことから，simple disjunctive decomposition が存在するからといって，必要以上に分解を適用することは望ましくないことがわかる．必要性の少ない関数はその出力側に併合した方が良いと考え，我々は以下のような手順で積和形論理式を多段化している．

1. 各出力の関数に対して，simple disjunctive decomposition が出来なくなるまで再帰的にこれを適用する．
2. 同じ入力と同じ内部関数を持つ関数を共通化する．
3. 必要性の少ない関数をその出力側の関数に併合する．

2では，各出力関数の simple disjunctive decomposition により発生した部分関数の，回路全体における必要性を評価している．すなわち，必要性の高い部分関数はこの処理によって複数の出力を持つことになる．

3では，ある関数 x の出力数が1で，出力先の関数 f において，変数 x がユネイトである (非冗長積和形において x あるいは x' でしか現れない) 場合，その関数 x の必要性が小さいとみなして併合を行っている．例えば，$x = ab$，$f = xc + xd$ の場合，変数 x はユネイトであるので，併合して

表 1: 実験結果

回路			SDD						代数的手法		SDD + 代数的手法		
名前	入	出	SX	SA	SS	2X	2A	時間 (秒)	リテラル	時間 (秒)	リテラル	比	時間 (秒)
5xp1	7	10	0	5	0	4	0	0.32	143	0.87	114	0.797	0.97
alu4	14	8	0	1	0	1	0	0.61	1099	242.88	1083	0.985	241.28
apex1	45	45	2	80	0	0	12	2.43	1703	113.76	1708	1.003	118.17
apex2	39	3	0	18	0	0	3	3.95	395	246.47	385	0.975	128.80
apex3	54	50	2	24	4	0	11	1.67	1631	43.68	1631	1.000	46.48
apex4	9	19	0	0	0	0	4	0.58	2686	393.14	2686	1.000	390.84
apex5	117	88	0	125	0	0	78	3.10	957	11.53	919	0.960	15.18
cordic	23	2	4	7	6	0	0	0.45	97	831.58	90	0.928	1.54
duke2	22	29	0	16	1	0	14	0.53	425	4.02	425	1.000	4.71
pdc	16	40	1	25	0	0	15	1.56	445	3.48	445	1.000	4.97
sao2	10	4	0	3	0	0	5	0.33	188	1.10	188	1.000	1.50
spla	16	46	6	52	7	0	26	1.42	637	12.06	620	0.973	13.20
t481	16	1	4	10	0	0	0	0.49	52	6.28	40	0.769	0.70
合計			19	366	18	5	168	17.44	10458	1910.85	10334	0.988	968.34

$f = abc + abd$ とする.

5 実験結果

本稿で提案した simple disjunctive decomposition の手法を実装し，ベンチマーク回路 [11] 内の積和形論理式に対して実験を行った．表 1に結果を示す．“回路” において，“入” は入力数，“出” は出力数を意味する．

まず，3 章で述べた simple disjunctive decomposition の高速化の効果を見るため，これだけを適用した結果を “SDD” に示す．“SX”，“SA”，“SS”，“2X”，“2A” は，適用された分解の種類を示す．“SX”，“SA”，“SS” は，対称変数集合を検出することによる分解に相当し，“2X”，“2A” は，分解の像が 2 入力関数である分解に相当する．これらにおいて，“X”，“A”，“S” はそれぞれ，XOR ゲート，AND ゲート，対称関数を意味する．“時間” は，Sun Ultra 2 Model 2200 にて実行したときに要した時間で，単位は秒である．

結果より，大規模な論理関数に対しても十分に実用的な時間で実行できることがわかる．また，実用的な論理設計において，本稿で述べた形の simple disjunctive decomposition が多く存在することがわかる．

次に，4 章で述べた前処理の効果を見るため，さらに 2 種類の実験を行った．“代数的手法” には，カリフォルニア大バークレー校の SIS [9] を用いて，代数的手法よる多段化を行った (具体的には，“script.algebraic” という論理合成スクリプトを実行した) 結果を示す．“SDD + 代数的手法” には，4 章で述べたように，まず simple disjunctive decomposition を行い，必要性の少ない関数をその出力側に併合し，そのあと代数的手法による多段化を行った結果を示す．“リテラル” には，多段化が終了した後の回路全体の総リテラル数を示す．

前処理として simple disjunctive decomposition を併用した場合としない場合の多段化結果を比べると，多くの回路では結果はそれほど違わない．しかし，XOR ゲートを多く使う回路に対しては，大きな改善が見られるものがある．まず，“5xp1” や “alu4” に存在する “2X” のタイプの分解は，検出するためには否定の一致判定を必要とするため，代数的手法ではなかなか見つけにくい分解であると思われる．また，“cordic” や “t481” では，処理時間に大きな改善が見られた．これらの元の積和形論理式には，非常に多くの積項が存在するが，XOR ゲートによる分解を検出することで，その数を大きく減らせたのが理由であると思われる．

6 おわりに

本稿では，simple disjunctive decompositionを大規模な論理関数にも適用できるようにするため，X^Bが対称変数集合である場合と，分解の像gが2入力関数である場合に分解の形を限ることで，処理の効率化を図った．また，そのための主要な処理である対称変数の検出の高速化についても述べた．実験結果より，大規模な論理関数に対してもわずかな時間で実行できることと，分解の形を上記のように限っても多くの分解が検出できることが確認できた．

また，simple disjunctive decompositionの論理合成における応用として，代数的手法による積和形論理式の多段化の前処理にこれを用いた．この結果，回路によっては，リテラル数削減や処理時間の削減に効果があることが確認できた．今後，実用的な論理合成システムへの応用を検討して行きたいと考えている．

参考文献

[1] R. L. Ashenhurst, "The Decomposition of Switching Functions," in *Proc. of an International Symposium on the Theory of Switching*, pp. 74–116, Apr. 1957.

[2] H. A. Curtis, *A New Approach to the Design of Switching Circuits.* Van Nostrand, 1962.

[3] J. P. Roth and R. M. Karp, "Minimization Over Boolean Graphs," *IBM Journal*, pp. 227–238, Apr. 1962.

[4] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 667–691, Aug. 1986.

[5] Y.-T. Lai, M. Pedram, and S. Vrudhula, "BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis," in *Proc. DAC*, pp. 642–647, June 1993.

[6] D. Möller, J. Mohnke, and M. Weber, "Detection of Symmetry of Boolean Functions Represented by ROBDDs," in *International Conference on Computer-Aided Design*, pp. 680–684, Nov. 1993.

[7] 笹尾 勤, "論理回路のBi-decompositionについて," 情処研報, 96–DA–82, pp. 9–16, Dec. 1996.

[8] 松永 裕介, "論理回路のbi-decompositionを行うボトムアップアルゴリズムについて," 信学技報, ICD96–200, pp. 25–32, Mar. 1997.

[9] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. CAD*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[10] C. R. Edwards and S. L. Hurst, "A Digital Synthesis Procedure Under Function Symmetries and Mapping Methods," *IEEE Trans. Computers*, vol. c-27, pp. 985–997, Nov. 1978.

[11] S. Yang, *Logic Synthesis and Optimization Benchmarks User Guide Version 3.0.* MCNC, Jan. 1991.