

二分決定グラフを用いた 論理関数の分離的非単純分解アルゴリズム

甲斐 齊, 儘田佳幸, 高木直史, 高木一義
名古屋大学大学院工学研究科情報工学専攻
464-8603 名古屋市千種区不老町
E-mail: kai@takagi.nuic.nagoya-u.ac.jp

あらまし 論理関数分解は論理合成において重要な役割を果たしており、研究が盛んに行われている。近年、効率の良いアルゴリズムが提案されているが、その多くは論理関数の単純分解を求めるものであった。本稿では非単純分解アルゴリズムを提案する。本アルゴリズムでは論理関数を表す二分決定グラフ (OBDD) の終端ノード側の変数に着目し、それらを束縛集合として分解を検出する。このアルゴリズムを実装し、ベンチマーク回路を用いて実験を行った。いくつかの回路において非単純分解の検出が行えることを示した。

キーワード 論理関数, 関数分解, 二分決定グラフ, 非単純分解

An Algorithm for Non-simple Disjunctive Decomposition of Logic Functions Using OBDD

Hitoshi Kai, Yoshiyuki Mamada, Naofumi Takagi, Kazuyoshi Takagi
Department of Information Engineering, Nagoya University
Furou-cho, Chikusa, Nagoya, 464-8603, Japan
E-mail: kai@takagi.nuic.nagoya-u.ac.jp

Abstract Decomposition of logic function plays an important part in logic synthesis. Recently, several algorithms have been proposed. However, most of them are for finding simple disjunctive decomposition. In this report, we propose an algorithm for non-simple disjunctive decomposition. The algorithm uses Ordered Binary Decision Diagrams (OBDD) to represent logic functions, and the main idea is to bound the variables placed in the lower part. We applied the algorithm to the benchmark circuits and detected non-simple decompositions for several functions.

key words logic function, functional decomposition, binary decision diagrams, non-simple decomposition

1 はじめに

論理関数分解は回路合成等の段階で用いられる重要な処理である。回路合成とは与えられた論理関数を生成する論理回路を作成する過程である。

以前から論理関数分解は複雑なシステムをいくつかの特定用途向け IC(ASIC) に分割するためにも必要となっていた。また、スタンダードセル方式の回路設計では、実現したい論理関数が分解できれば、論理回路全体の分解した各部分でセルの配置を考えればよくなる。現在、小規模な基本論理ブロックを組み合わせて論理回路を実現する FPGA ベースシステムの用途の増加により回路分割問題はますます重要なものとなっている。

多くのランダムな論理関数は分解不可能であることが知られている。しかし計算機で使用する制御回路や算術演算回路などの多くの実用的関数は分解可能であることが実験的に確認されている [5]。

与えられた論理関数 F を論理関数 H と、 F と出力が一致する論理関数 G で表すことができるとき、これを F の論理関数分解という。ただし、 H の入力変数集合は F の入力変数の真の部分集合とし、 G の入力変数集合は H の出力と F の入力変数の部分集合である。関数分解にはいくつかの分類がある。 H の入力変数集合と G の入力変数集合に共通部分がないものを分離的分解、そうでないものを非分離的分解という。 H の出力数が 1 の分解を単純分解、 H の出力数が 2 以上となるものを非単純分解という。

これまでに、単純分解を求めるアルゴリズム [1][2] や非単純分解を求めるアルゴリズム [3][4][8] が提案されている。非単純分解は適用できる範囲が広いが複雑である。また LUT ベースの FPGA 向けの関数分解の手法の研究も行われている [9]。

本稿では論理関数を表す二分決定グラフ (OBDD) の終端ノード側の変数に着目した分離的非単純分解アルゴリズムを提案する。提案アルゴリズムでは論理関数を二分決定グラフを用いて操作する。二分決定グラフの終端ノードから第 k レベルの各ノードはそのノード以下にある k 個の変数を入力とする k 変数論理関数に対応する。この関数の個数について分解可能性の判定を行い、分解可能な場合はその各ノードが表す各 k 変数論理関数を、 $l (< k)$ 個の k 変数論理関数の出力を入力とする l 変数論理関数で表現する。このアルゴリズムを実装し、ベンチマーク回路を用いて実験を行う。

本稿の構成は以下のものである。第 2 章で準備として論理関数分解と二分決定グラフについて説明し、第 3 章で論理関数の新しい非単純分解アルゴリズムを提案し、第 4 章で実験結果を示す。

2 準備

2.1 論理関数分解

論理関数分解とは、与えられた論理関数 F を論理関数 H と、 F と出力が一致する論理関数 G に分ける処理である。ここで H の入力変数集合 $X_B = \{x_1, \dots, x_k\}$ は F の入力変数 $X = \{x_1, \dots, x_n\}$ の真の部分集合とし、 G の入力変数集合は H の出力 $Y = \{y_1, \dots, y_l\}$ と F の入力変数 X の部分集合 $X_F = \{x_{k-s}, \dots, x_n\}$ とする。ただし $0 < k < n, -1 \leq s \leq k-1, 0 < l < k$ とする。すなわち

$$F(X) = G(H(X_B), X_F) \quad (1)$$

$$= G(y_1, \dots, y_l, X_F) \quad (2)$$

と表す。ここで X_B を束縛集合、 X_F を自由集合と呼ぶ。論理関数 F に対して (1) 式を満たすような論理関数 G, H が存在するとき、論理関数 F は分解可能であるという。

論理関数分解の利点について述べる。関数 G, H のそれぞれの入力変数の個数は与えられた論理関数 F の入力変数の個数よりも小さくなる。一般に n 入力関数の回路の大きさは $O(2^n/n)$ となるので、分解することで回路の全体的な大きさが小さくなると考えられている [2]。また、関数 G, H の複雑性は F に対して小さくなると考えてよいので、関数 G, H の最適設計は関数 F の最適設計を導くことになる。

分離的分解は、(1) 式において H の入力変数集合 X_B と G の入力変数集合 (H の出力と F の入力変数 X の部分集合 X_F) に共通部分がない分解である。すなわち $s = -1$ のときであり、 $X_B \cap X_F = \emptyset$ となる。共通部分がある場合、すなわち $s > -1$ のとき非分離的分解という。非分離的分解の例を図 1 に示す。

非単純分解は、(1) 式において H の出力数が 2 以上となるような分解である。すなわち (2) 式において $l \geq 2$ となる分解である。 $l = 1$ の分解を単純分解という。図 2 に分離的非単純分解、図 3 に分離的単純分解の例を示す。

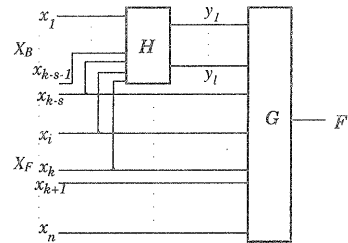


図 1: 論理関数の非分離的分解

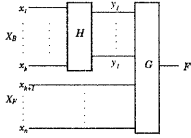


図 2: 論理関数の
分離的非単純分解

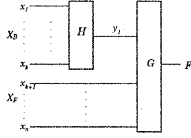


図 3: 論理関数の
分離的単純分解

2.2 二分決定グラフ

二分決定グラフ (Binary Decision Diagram、以下 BDD とする) とは、場合分けグラフによる論理関数の表現方法である [7]。

BDD のノードには非終端ノードと終端ノードがある。非終端ノードは変数でラベル付けされており変数ノードと呼び、終端ノードは論理値でラベル付けされており定数ノードと呼ぶ。変数ノードの中で最上位のノードを根ノードと呼ぶ。終端ノードから根ノードに向かって、終端ノードからの変数ノードの高さをレベルと呼ぶ。終端ノードのレベルは 0 とする。各変数ノードから下へ左右に 2 本の枝が存在し、左の枝を 0 枝、右の枝を 1 枝と呼ぶ。根ノードから出発し、変数の値 (0 または 1) により 0 枝または 1 枝をたどる操作を繰り返す。最終的に到着した終端ノードの論理値が与えられた変数の値割り当てに対する関数の値となる。BDD の各ノードは一つの論理関数を表す。終端ノードから第 k レベルの各ノードはそのノード以下にある k 個の変数を入力とする各 k 変数論理関数を表す。特に BDD の根の表す論理関数を、その BDD の表す論理関数という。

ある変数ノード v に対して 0 枝で指される変数ノードを $\text{low}(v)$ 、1 枝で指される変数ノードを $\text{high}(v)$ とする。変数ノード v にラベル付けされた変数を $\text{var}(v)$ 、終端ノード v にラベル付けされた論理値を $\text{value}(v)$ とする。ノード v が表す論理関数 f_v は、

$$f_v = \begin{cases} \text{value}(v) & (v: \text{定数ノード}) \\ \overline{\text{var}(v)} \cdot f_{\text{low}(v)} + \text{var}(v) \cdot f_{\text{high}(v)} & (v: \text{変数ノード}) \end{cases} \quad (3)$$

と定義される。ここで変数 $\text{var}(v)$ に対し $\text{var}(v)=0$ としたときの関数 $f|_{\text{var}(v)=0}$ 、及び $\text{var}(v)=1$ としたときの関数 $f|_{\text{var}(v)=1}$ を論理関数 f のコファクタと定義すると、

$$f_{\text{low}(v)} = f|_{\text{var}(v)=0} \quad (4)$$

$$f_{\text{high}(v)} = f|_{\text{var}(v)=1} \quad (5)$$

が成り立つ。

提案アルゴリズムでは準既約な順序付き BDD (Ordered BDD) を使用する。OBDD は論理関数 f に対して、根ノードから定数ノードに向かう全てのパスについて出現

する変数の順序が同一な BDD である。既約化とは冗長なノードと等価なノードの削除をこれ以上できなくなるまで繰り返す操作である。等価なノードの統合のみを行い、冗長なノードの削除は行わない操作を準既約化という。論理関数 f を表現する準既約な OBDD は一意に定まる。

例として論理関数 $f(x_1, x_2, x_3) = x_1 \bar{x}_2 + x_3$ を表現する既約な順序付き BDD と準既約な順序付き BDD をそれぞれ図 4、図 5 に示す。以下、BDD は準既約な順序付き BDD のことを指し、BDD の変数順序を終端ノード側から根ノードに向かって $x_1, x_2, \dots, x_n$ と固定して説明する。

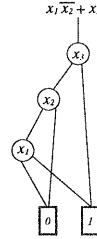


図 4: 既約な順序付
き BDD の例

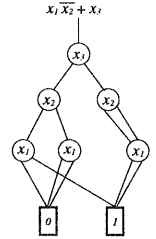


図 5: 準既約な順序
付き BDD の例

3 論理関数の分離的非単純分解アルゴリズム

(1) 式において、論理関数 F の入力変数 $X = \{x_1, \dots, x_n\}$ を BDD の終端ノード側から $X_B = \{x_1, \dots, x_k\}$ と $X_F = \{x_{k+1}, \dots, x_n\}$ に分離的に分割したと仮定する。次に終端ノードから第 k レベルの k 変数論理関数が 2^{2^k} 個以下であるとき、その真理値表よりベクトル表現し、それらを並べた表を考える。次に、その表の各列に着目し列ベクトルを取り出す。この列ベクトルの種類が 2^l 種類であれば l 個の変数 $y_1, \dots, y_l$ により $(y_1, \dots, y_l) = (0, \dots, 0), (0, \dots, 1), \dots, (1, \dots, 1)$ と割り当てができる。これにより $y_1, \dots, y_l$ のベクトル表現を決定できる。このベクトル表現から $x_1, \dots, x_k$ を入力変数とする論理関数 $y_1, \dots, y_l$ をそれぞれ決定できる。

アルゴリズムを以下に示す。入力 は 論理関数 $F(x_1, \dots, x_n)$ とする。出力は $l (< k)$ 個の k 変数論理関数 $y_1(x_1, \dots, x_k), \dots, y_l(x_1, \dots, x_k)$ を含む G する。論理関数 F の終端ノードから第 k レベルの各 k 変数論理関数の集合を $\mathcal{P}(x_1, \dots, x_k) = \{P_0(x_1, \dots, x_k), \dots, P_{|P|-1}(x_1, \dots, x_k)\}$ とし、それらの関数の個数を $|P|$ とする。

異なる関数を取り出した集合を $\text{Vec} = \{V_1, V_2, \dots\}$ とし、それらの関数の個数を $|\text{Vec}|$ とする。すなわち $\text{Vec} \subseteq G$ である。この $|\text{Vec}|$ が列ベクトルの種類の個数である。 $|\text{Vec}|$ が 2^l 個以下ならば Step4 へ進む。そうでなければ l 変数論理関数への非単純分解は不可能である。論理関数 T の BDD を図 7 に示す。

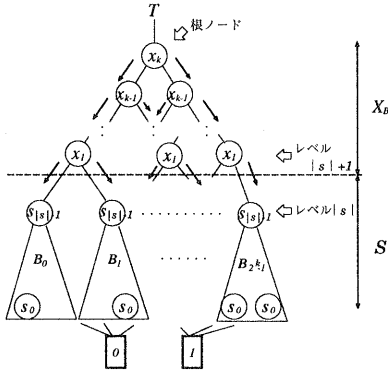


図 7: 論理関数 T の BDD

Step4 において、変数 $y_1, \dots, y_r, \dots, y_l$ を用いて Vec のそれぞれの関数に値割り当てを行う。今回は関数 V_j の番号 j を l ビット 2 進表現したものを変数 $y_1, \dots, y_l$ を用いて関数 V_j にそれぞれ割り当てている。 G_i の種類が V_j であるとき変数 y_r の i ビット目に、 V_j に割り当てた y_r の値を代入する。変数 $y_1, \dots, y_r, \dots, y_l$ のそれぞれのベクトル表現が定まり、 k 変数関数 $y_1, \dots, y_r, \dots, y_l$ を決定することができる。 l 変数論理関数への非単純分解後の F の BDD を図 8 に示す。図 6 と比較すると 2^{2^l} 個以下 ($l < k$) の k 変数論理関数 P が l 個の k 変数論理関数の出力を全て同じ入力とする l 変数論理関数 Q で表現できたことがわかる。各変数順序で Step1 から Step4 を繰り返す。

4 実験結果

3 章で提案したアルゴリズムを C 言語で実装し、UltraSPARC-IIi 333MHz, メモリ 640Mbyte を備えた計算機で実験を行った。BDD 操作には公開されている BDD パッケージ [6] を使用した。表 1 に ISCAS85 ベンチマーク回路に対する実験結果を示す。In は入力数、Out は出力数、Be と Ma はそれぞれ [1]、[2] による単純分解可能な出力数、All は関数分解に成功した出力の合計、smp は単純分解に成功した出力数、mlt は非単純分解に成功した出力数である。今回は BDD の終端ノードから第 3 レベルまでの変数を束縛集合とし、3 変数から 2 変数へ

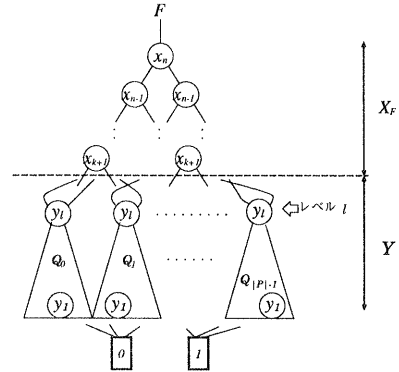


図 8: 非単純分解後の論理関数 F の BDD

の非単純分解に限定し実験を行った。3 変数から 1 変数へ分解できたものを単純分解として検出し、smp の個数で表している。入力数が約 40 以下の回路については各束縛集合において全ての変数順序を考えているが、それ以上のものについては一部の变数順序のみで実験を行った。したがって、単純分解が行える関数でも非単純分解として検出してしまう場合がある。

Be, Ma は単純分解のみを検出しているので、All がこれらより多ければ本アルゴリズムにより非単純分解が検出できているといえる。

表 1 において [1]、[2] と比較したところ、C432、k2、vda などにおいて非単純分解が検出できていることがわかる。[1]、[2] では単純分解可能な出力のみを検出しているが、提案アルゴリズムは単純分解可能な出力だけでなく非単純分解可能な出力も検出できた。

5 おわりに

論理関数の分離的非単純分解アルゴリズムを提案した。提案アルゴリズムでは OBDD を使って、終端ノードから第 k レベルの k 変数論理関数を l ($l < k$) 変数論理関数に置き換えることで論理関数分解を行った。単純分解可能な出力だけでなく非単純分解可能な出力も検出できた。実験結果より非単純分解でのみ分解可能な関数が実用回路でも多く現れると考えられる。提案アルゴリズムについては、列ベクトルの種類に対する値割り当てを工夫すれば関数 H を簡単化できる可能性があり、検討の余地があると思われる。

参考文献

- [1] V.Bertacco and M.Damiani, "The disjunctive decomposition of logic functions", ICCAD-97, pp.78-82, Nov.1997.
- [2] Y.Matsunaga, "An exact and efficient algorithm for disjunctive decomposition," SASIMI'98, pp.44-50, Oct.1998.
- [3] Y. Lai, K. Pan, and M. Pedram, "OBDD-Based Function Decomposition: Algorithms and Implementation", IEEE Trans. Computer-Aided Design, Vol.15, pp.977-990, Aug. 1996.
- [4] H.Sawada, T.Suyama and A.Nagoya, "Logic Synthesis for Look-Up Table Based FPGAs Using Functional Decomposition and Boolean Resubstitution", IEICE TRANS.INF.& SYST., vol.E80-D, No.10, pp.1017-1023, Oct.1997.
- [5] T.Sasao and M.Matsuura, "DECOMPOS : An integrated system for functional decomposition", 1998 International Workshop on Logic Synthesis, Lake Tahoe, June 1998.
- [6] S.Minato, N.Ishiura and S.Yajima, "Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation", Proc. 27th Design Automat. Conf., pp.52-57, 1990.
- [7] R.E.Bryant, "Graph-based algorithm for Boolean function manipulation", IEEE Trans. Comput., vol.35, No.8, pp.677-691, 1986.
- [8] Tsutomu Sasao, Ken-ichi Kurimoto, "Three Parameters to Find Functional Decompositions", ASP-DAC2000, pp.259-264, 2000.
- [9] Shigeru Yamashita, Hiroshi Sawada, Akira Nagoya, "An Efficient Framework of Using Various Decomposition Methods to Synthesize LUT Networks and Its Evaluation", ASP-DAC2000, pp.253-258, 2000.

表 1: ベンチマーク回路の実験結果

Crcr	In	Out	Be	Ma	All	smp	mlt
C1355	41	32	0	0	0	0	0
C1908	33	25	7	7	6	6	0
C432	36	7	1	1	2	1	1
C499	41	32	0	0	0	0	0
C880	60	26	25	26	15	7	8
C17	5	2	-	-	2	0	2
apex6	135	99	99	99	95	79	16
apex7	49	37	36	37	32	21	11
cordic	23	2	-	-	2	0	2
comp	32	3	3	-	0	0	0
count	35	16	16	16	16	16	0
k2	45	45	35	-	36	25	11
x3	135	99	99	-	95	79	16
x4	94	71	65	-	51	51	0
vda	17	39	20	-	34	17	17
rot	135	107	77	-	72	71	1
CM42	4	10	10	-	10	10	0
CM85	11	3	3	-	3	1	2
frg2	143	139	126	139	104	104	0
pair	173	137	125	137	127	126	1
cm150	21	1	-	-	1	0	1
parity	16	1	-	-	1	1	0
cmb	16	4	-	-	4	4	0
f51m	8	8	-	-	4	3	1
lal	26	19	-	-	14	13	1
mux	21	1	-	-	1	0	1
term1	34	10	-	-	9	3	6
ttt2	24	21	-	-	18	18	0
b9	41	21	-	21	17	17	0
too_l g	38	3	-	3	3	2	1