不完全定義関数を表現する PKDD の簡単化法

松浦 宗寛 [1], 笹尾 勤 [1,2]

[1] 九州工業大学 情報工学部

[2] 九州工業大学 マイクロ化総合技術センター

あらまし: 論理関数を表現する方法の一つとして, 疑似クロネッカ決定グラフ (PKDD) がある. PKDD は二分決定グラフ (BDD) を一般化したものであり, その節点数は BDD よりも多くならず, 多くの場合より少ない節点数で表現できる. 本稿では, ドント・ケアを含む不完全定義関数を PKDD を用いて表現することを考える. 実用時間内に少ない節点数の PKDD を導くヒューリスティック法を開発した. MCNC ベンチマーク関数のうちドント・ケアを含む関数に対して PKDD の生成を行い, 約 14% 節点数を削減できた.

和文キーワード: 不完全定義関数, 二分決定グラフ (BDD), 擬似クロネッカ決定グラフ (PKDD), EXOR, ドントケア

A Method to Minimize The Pseudo-Kronecker Decision Diagrams for Incompletely Specified Functions

Munehiro MATSUURA[1], Tsutomu SASAO[1,2]

[1]Department of Computer Science and Electronics, Kyushu Institute of Technology

[2]Center for Microelectronic Systems, Kyushu Institute of Technology

Abstract: Pseudo-Kronecker decision diagram (PKDD) is a generalization of binary decision diagram (BDD). A PKDD requires not more nodes than a BDD to represent the same function. In this paper, we consider a method to represent incompletely specified functions by using PKDDs. We developed a heuristic method to obtain PKDDs. Many PKDDs for MCNC benchmark functions with *don't cares* are simplified. Experimental results show that the number of nodes of PKDD can be reduced by 14% by considering *don't cares*.

Key words: Incompletely specified function, Binary decision diagram, Pseudo-Kronecker decision diagram, EXOR, Don't care

1 まえがき

現在, 論理関数の表現法として, BDD(Binary Decision Diagram: 二分決定グラフ) がよく使用されている. ここでは, BDD を一般化した擬似クロネッカ決定グラフ PKDD (Pseudo Kronecker Decision Diagram) を用いて論理関数を表現することを考える. PKDD は, BDD を一般化した決定グラフであり, 論理関数を表現するための節点数は BDD よりも多くならない. PKDD で表現された論理関数は, 節点を対応する回路に置き換える事で, そのまま回路実現可能である. また, 関数分解を用いた方法 [12] や三つの節点を一つの 6 入力 LUT としてマッピングする方法 [8] 等で, テーブル参照型 FPGA 設計にも応用できる. 本論文では, 不完全定義関数が与えられた場合に節点数の少ない PKDD を得る方法を提案する. 不完全定義関数を表現する PKDD を最小化するための計算時間のオーダーは, 入力変数の個数を n, ドント・ケアの数を d とすると, $O(2^d \cdot n! 3^n \cdot 3^{2^n-1})$ となる. そのため, 最適解を求める事は, n が小さい時を除きほとんど不可能である. 従って, 実用時間内に少ない節点数の PKDD を導くヒューリスティック法を提案する.

不完全定義関数を表現する PKDD の生成法に関しては, 過去には公表された論文はなく, 節点数がどの程度削減できるか知られていなかった. MCNC ベンチマーク関数のうちドント・ケアを含む関数に対して PKDD を最適化し, その節点数をドント・ケアの最適化を行わない場合と比較した.

2 擬似クロネッカ決定グラフ

2.1 二分決定グラフ (BDD)

任意の論理関数 f は

$$f = \bar{x}_1 f_0 \oplus x_1 f_1 \quad (1)$$
$$f = f_0 \oplus x_1 f_2 \quad (2)$$
$$f = \bar{x}_1 f_2 \oplus f_1 \quad (3)$$

の形で展開可能である. ここで, $f_2 = f_0 \oplus f_1$ である. (1) をシャノン展開, (2) を正極性ダビオ展開, (3) を負極性ダビオ展開と呼ぶ. 関数 f が与えられた時, f と展開すべき変数 x_1 が決まれば, 部分関数 f_0, f_1, f_2 は一意的に定まる. また, f_0, f_1 に対してシャノン展開を再び適用すれば

$$f_0 = \bar{x}_2 f_{00} \oplus x_2 f_{01},\ f_1 = \bar{x}_2 f_{10} \oplus x_2 f_{11}$$

と展開できる. 同様に f_{00}, f_{10} に対して同じ展開を行うと

$$f_{00} = \bar{x}_3 f_{000} \oplus x_3 f_{001},\ f_{01} = \bar{x}_3 f_{010} \oplus x_3 f_{011},$$
$$f_{10} = \bar{x}_3 f_{100} \oplus x_3 f_{101},\ f_{11} = \bar{x}_3 f_{110} \oplus x_3 f_{111} \text{ を得る.}$$

図 1はこの展開を示した二分決定木 (Binary Decision Tree) である. この決定木は, 次の二つの規則を用いて簡単化できる.

1) 部分関数が変数 x に依存しない場合は, x に関しては展開しない.
2) 部分関数が同じ時, 部分木を共有する.

この方法で簡単化したものが二分決定グラフ (Binary Decision Diagram: BDD) である. 関数 f と変数の順序が与えられたとき, BDD は一意的に定まる.

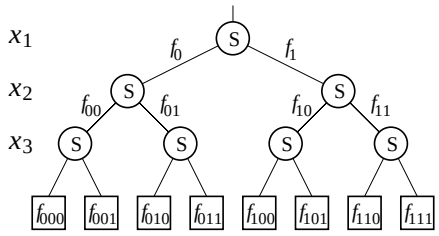

図 1: 二分決定木

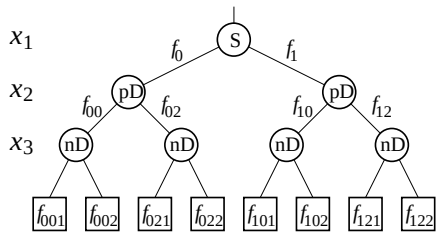

図 2: クロネッカ決定グラフ

2.2 クロネッカ決定グラフ (KDD)

各変数で (1)∼(3) のいずれかを許した展開を考える. 例えば, 図 2では, x_1 に関してはシャノン展開 (図 2で S と略記), x_2 に関しては正極性ダビオ展開 (pD と略記), x_3 に関しては負極性ダビオ展開 (nD と略記) を用いている. このような展開をクロネッカ展開 (Kronecker expansion) と呼び, 簡単化した決定グラフをクロネッカ決定グラフ (KDD: Kronecker Decision Diagram) という. n 変数関数と変数順序が与えられたとき, 異なる KDD は高々 3^n 個存在する.

2.3 擬似クロネッカ決定グラフ (PKDD)

KDD では, 各変数に対して展開法は同じである. しかし, 展開木の各節点において, (1)∼(3) の任意の展開を許すとすると, 例えば, 図 3のような決定木が得られる. この決定木では, x_1 に関してはシャノン展開, x_2 に関しては, 正極性ダビオ展開と負極性ダビオ展開, x_3 に関しては全ての展開法を用いている. このような展開を擬似クロネッカ展開 (pseudo-Kronecker expansion) と呼び, 簡単化した決定グラフを擬似クロネッカ決定グラフ (PKDD) という. 変数の順序を固定した時, 一つの関数に対して高々 3^{2^n-1} 個の PKDD が存在する. 図 4は BDD, KDD, PKDD の関係を示したものである. PKDD が最も広いクラスであり, 関数を表現するために必要な節点数も最も少ない.

2.4 不完全定義関数を実現する PKDD の簡単化の方針

n 変数関数を表現する PKDD を構成する場合

a) 変数の展開順序: $n!$ 通り.
b) 各節点の展開法: 3^{2^n-1} 通り.

存在する. 従って, 厳密な最適解を求めるのは, 現実的ではなく, ヒューリスティック法が提案されている [9, 3]. また, ドント・ケアの個数を d とすると, 2^d 通りの関数が存在し, 節点数が最小になるような関数を生成する割当を見つけることは, 変数順序が固定された BDD の場合でさえ, NP 困

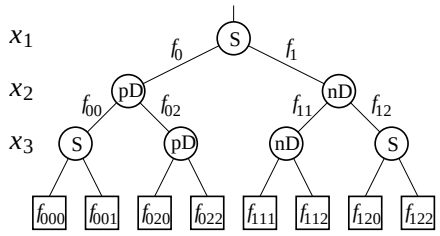

図 3: 擬似クロネッカ決定グラフ

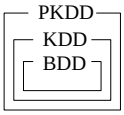

図 4: BDD, KDD, PKDD の関係

難である [13]. 従って, 本論文では, 既存のヒューリスティック法を組み合わせて, 以下の方法で, PKDD を構成する.

1. BDD の節点数をなるべく削減するように, ドント・ケアの値を割り当てる (アリゴリズム 3.1).
2. 変数順序を決定し, 各変数に対して最適な展開を選択し, なるべく小さな KDD を生成する (アルゴリズム 4.2).
3. 2 で求めた展開法を初期値として, PKDD の各節点の展開法を決定し, なるべく小さな PKDD を生成する (アルゴリズム 4.3).

3 不完全定義関数を表現する BDD の簡単化

本節では, 不完全定義関数のドント・ケアの割り当てを行う事により, BDD 節点数を削減する方法を示す [7].

3.1 不完全定義関数

ある入力組合せにおいて, 出力が 0 でも 1 でも良い場合, その値をドント・ケアという. ドント・ケアを含む関数を不完全定義関数という. 不完全定義関数は二つの完全定義関数, f と g の対 $[f, g]$ で表せる. ここで g はケアセット, $f \cdot g$ はオンセット, $\bar{f} \cdot g$ はオフセット, $\bar{g}$ はドント・ケアセットを表す.

定義 **3.1** h が $f \cdot g \subseteq h \subseteq f \vee \bar{g}$ を満たすとき, h は $[f, g]$ のカバーであるという. 特に, どんな $[f_1, g_1]$ のカバーも $[f_2, g_2]$ のカバーであるとき, $[f_1, g_1]$ は $[f_2, g_2]$ の i カバーであるという. また, 二つの不完全定義関数に共通の i カバーが存在するとき, 二つの不完全定義関数はマッチするという.

二つの不完全定義関数に対して共通の i カバーが存在すれば, 二つの関数を同一の関数として表現可能になる. つまり, BDD 上で二つの節点が表す関数に共通の i カバーが存在する場合, 二つの節点を i カバーを表す節点と置き換える事で節点数を減らす事ができる.

3.2 マッチング

二つの不完全定義関数のマッチングを行う場合, ドント・ケアの割り当て方によって三つの基準が考えられる.

$[f_1, g_1]$, $[f_2, g_2]$ をそれぞれ不完全定義関数とする.

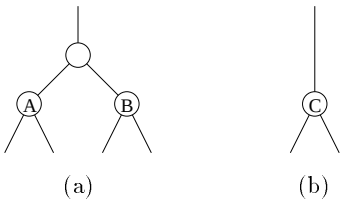

図 5: BDD の簡単化

1. One-sided DC match (OSDM)
   二つの関数 $[f_1, g_1]$, $[f_2, g_2]$ が OSDM であるとは, $g_1 = 0$ の時をいう.
   これは, 一方の関数が全てドント・ケアである.
2. One-sided match (OSM)
   二つの関数 $[f_1, g_1]$, $[f_2, g_2]$ が OSM であるとは, $f_1 \oplus f_2 \subseteq \bar{g}_1$, $\bar{g}_2 \subseteq \bar{g}_1$ の時をいう.
   これは, 一方の関数のドント・ケアを操作することで, 二つの関数を等価にできる.
3. Two-sided match (TSM)
   二つの関数 $[f_1, g_1]$, $[f_2, g_2]$ が TSM であるとは, $f_1 \oplus f_2 \subseteq \bar{g}_1 \vee \bar{g}_2$ の時をいう.
   これは, 二つの関数の両方のドント・ケアを操作することにより, 二つの関数を等価にできる.

マッチングを行う場合, ドント・ケアをなるべく残すように割り当てる. 二つの関数 $[f_1, g_1]$, $[f_2, g_2]$ がマッチした時, それぞれのマッチング基準における, i カバーのドント・ケアの割当を以下のように行う. 1. OSDM: $[f_2, g_2]$, 2. OSM: $[f_2, g_2]$, 3. TSM: $[f_1 g_1 \vee f_2 g_2, g_1 \vee g_2]$.

3.3 マッチングを用いた BDD の簡単化

BDD 上でマッチングを行う二つの節点を選ぶ方法として, ある節点の二つの子に着目する. 二つの子に共通の i カバーが存在すれば, 一つの節点で表現できる. そのとき, 親である節点も削除できる.

例 **3.1** 図 *5(a)* の *BDD* において, 節点 A の表す関数と節点 B の表す関数に共通の i カバーが存在すると仮定する. この時, 節点 A, B を i カバーを表す節点 C に置き換える事で, *BDD* は図 *5(b)* のように簡単化できる. (例終り)

BDD に対してマッチングを行うアルゴリズムを示す.

アルゴリズム **3.1** *BDD* の根から再帰的に以下の処理を行う.

1. ケアセット g が $g = 1$ ならば終了.
2. 二つの子に対して, *OSDM*, *OSM*, *TSM* の順に共通の i カバーが存在するかを調べる.
   - *(a)* 存在しなければ, 二つの子に対して再帰する.
   - *(b)* 存在すれば, この節点を i カバーと置き換え, i カバーに対して再帰する.

4 PKDD の簡単化

KDD と PKDD の簡単化には, EXOR 三分決定グラフ (ETDD) を用いる [11]. ETDD はシャノン展開を用いた際の部分関数 f_0, f_1 を表す二つの節点と, $f_2 = f_0 \oplus f_1$ を表す節点の三つの子を持つ. まず, KDD の簡単化法を示す.

アルゴリズム **4.2** *(KDD の簡単化)*

1. 節点数削減アルゴリズム *[1, 6]* で, *ETDD*, *BDD* のそれぞれに対して変数展開順序を決定する.
2. *1* で求めた二つの変数順序の *ETDD* を構成する.
3. 展開法の初期値として, すべての変数を S 展開とした場合, pD 展開とした場合, nD 展開とした場合のそれぞれについて以下の操作を行い, 節点数最小のものを選択する.
4. *ETDD* の根に近い変数から順に全ての変数について, その変数の展開法を S 展開, pD 展開, nD 展開とした場合の節点数を調べ, 節点数最小の展開法を選択する.
5. 節点数が変化しなくなるまで, *4* を繰り返す.
6. *1* で求めた二つの変数順序のうちで, 節点数が少ない方を選択する.

PKDD の簡単化では, KDD の簡単化結果を初期解とする.

アルゴリズム **4.3** *(PKDD の簡単化)*

1. 節点数削減アルゴリズムで, *ETDD*, *BDD* のそれぞれに対して変数展開順序を決定する.
2. *1* で求めた二つの変数順序の *ETDD* を構成する.
3. 展開法の初期値として, *KDD* の節点数を削減した結果を用いて以下の操作を行う.
4. *ETDD* の根に近い変数から順に全ての変数について, 各節点の展開法を S 展開, pD 展開, nD 展開とした場合の節点数を調べ, 節点数最小の展開法を選択する.
5. 節点数が変化しなくなるまで, *4* を繰り返す.
6. *1* で求めた二つの変数順序のうちで, 節点数が少ない方を選択する.

次に, 不完全定義関数を表現する PKDD の簡単化アルゴリズムを示す.

アルゴリズム **4.4** *(不完全定義関数を表現する PKDD の簡単化)*

1. 全てのドント・ケアの割当を *0* にして, 節点数削減アルゴリズムで *BDD* の変数展開順序を決定する.
2. アルゴリズム *3.1*を用いてドント・ケアの値を割り当て, *BDD* 節点数を削減する.
3. アルゴリズム *4.2*を用いて, *KDD* 節点数を削減する.
4. アルゴリズム *4.3*を用いて, *PKDD* 節点数を削減する.
5. 全てのドント・ケアの割当を *1* にした場合に対して, *1*〜*3* と同じ事を行い *PKDD* の節点数を削減する.
6. *3* と *4* の結果のうち, 節点数が少ない方を選択する.

5 実験結果

MCNC ベンチマーク関数のうちドント・ケアを含む関数に対して実験を行った. 実験に用いた BDD では, 否定エッジは使用していない.

5.1 不完全定義関数を表現する BDD の簡単化

アルゴリズム 3.1を用いて BDD の簡単化を行った結果を表 1に示す. 不完全定義関数 $[f, g]$ の関数 f のドント・ケアの割当をすべて 0 とした場合, および, すべて 1 とした場合のそれぞれについて, BDD の変数順序最適化を行い, アルゴリズム 3.1を適用した. 表 1で Total は各列の節点数の総和を表し, Ratio はそれぞれの関数の節点数の削減率の平均を表す. アルゴリズム 3.1を適用した場合の削減率は約 11% 程度である. $DC = 1$ とした方が削減率は大きいが, 節点数の総和は多い. 特に ex1010 と pdc はドント・ケアの割当によって BDD の節点数が半分近く削減できた.

表 1: アルゴリズム 3.1を適用した BDD の節点数

Name	In	Out	$DC = 0$	Match0	$DC = 1$	Match1
alu2	10	8	70	68	76	70
apla	10	12	115	91	119	93
b10	15	11	290	288	290	288
b11	8	31	99	98	99	98
b3	32	20	409	408	409	408
b4	33	23	219	218	225	224
b7	8	31	99	98	99	98
bca	26	46	894	892	897	895
bcb	26	39	756	751	756	751
bcc	26	45	801	793	780	771
bcd	26	38	606	606	608	608
bcddiv	4	4	19	19	20	19
bench	6	8	85	51	101	52
bench1	9	9	608	365	687	360
bw	5	28	137	122	137	122
dekoder	4	7	27	27	28	25
dk17	10	11	74	69	81	67
dk27	9	9	35	33	42	38
dk48	15	17	81	79	102	72
ex1010	10	10	1421	726	1426	729
exam	10	10	341	161	372	158
exep	30	63	675	675	756	739
exp	8	18	212	187	222	202
exps	8	38	585	583	585	583
fout	6	10	141	114	155	114
inc	7	9	82	75	80	73
l8err	8	8	84	83	84	83
mark1	20	31	119	115	177	105
p1	8	18	208	151	210	147
p3	8	14	134	106	152	106
pdc	16	40	609	337	696	343
sex	9	14	62	61	62	61
spla	16	46	628	608	636	616
t2	17	16	145	125	147	125
t4	12	8	44	44	52	44
wim	4	7	26	26	27	24
Total			10940	9253	11395	9311
Ratio			1.00	0.89	1.00	0.84

$DC = 0$: すべてのドント・ケアに 0 を割り当て, BDD 変数順序最適化を行った際の節点数
Match0: $DC = 0$ の結果に対してアルゴリズム 3.1を適用した際の節点数
$DC = 1$: すべてのドント・ケアに 1 を割り当て, BDD 変数順序最適化を行った際の節点数
Match1: $DC = 1$ の結果に対してアルゴリズム 3.1を適用した際の節点数

5.2 不完全定義関数を表現する PKDD の簡単化

表 2は, 表 1で得られた BDD に対して, アルゴリズム 4.4を用いて, PKDD の簡単化を行った結果を示す. Total は各列の節点数の総和を表す. Ratio は $DC = 0$ を 1.00 とした節点数の削減率の平均である. $DC = 0$ の場合に対して, アルゴリズム 4.4を用いて得られた PKDD 節点数の削減率は約 14% 程度である. 表 1の $DC = 0$ の場合の節点数の総和

表 2: アルゴリズム 4.4を適用した PKDD の節点数

Name	In	Out	BDD	PKDD		
				$DC=0$	$DC=1$	本手法
alu2	10	8	70	60	64	48
apla	10	12	115	96	96	68
b10	15	11	290	249	246	244
b11	8	31	99	56	54	54
b3	32	20	409	308	308	308
b4	33	23	219	157	196	157
b7	8	31	99	56	54	54
bca	26	46	894	797	791	784
bcb	26	39	756	751	664	658
bcc	26	45	801	704	684	670
bcd	26	38	606	536	530	531
bcddiv	4	4	19	12	12	10
bench	6	8	85	61	68	39
bench1	9	9	608	513	540	335
bw	5	28	137	90	90	83
dekoder	4	7	27	17	19	15
dk17	10	11	74	65	63	49
dk27	9	9	35	22	30	22
dk48	15	17	81	60	74	52
ex1010	10	10	1421	1268	1274	684
exam	10	10	341	272	289	116
exep	30	63	675	543	671	581
exp	8	18	212	156	179	152
exps	8	38	585	495	493	489
fout	6	10	141	111	117	92
inc	7	9	82	62	58	55
l8err	8	8	84	72	69	69
mark1	20	31	119	60	124	60
p1	8	18	208	174	179	114
p3	8	14	134	111	127	77
pdc	16	40	609	400	557	248
sex	9	14	62	41	41	41
spla	16	46	628	578	525	531
t2	17	16	145	119	120	105
t4	12	8	44	33	41	29
wim	4	7	26	16	18	14
Total			10940	9121	9465	7631
Ratio				1.00		0.86

$DC=0$: ドント・ケアに 0 を割り当てた関数の PKDD を簡単化した際の節点数

$DC=1$: ドント・ケアに 1 を割り当てた関数の PKDD を簡単化した際の節点数

と, 表 2のアルゴリズム 4.4を用いて得られた PKDD の節点数の総和を比べると 3 割程度削減できた.

例えば, 最も節点数を削減できた ex1010 では, $DC=0$ と割り当てた BDD に対してドント・ケアの割当を行うと, 約 49% 節点数が減り, その結果に対して PKDD 最適化を行うと約 6% 節点数が減った. もとの BDD から考えると, PKDD では約 58% 節点数が減っている.

5.3 各決定グラフの節点数の比率

BDD および PKDD の節点数を比較したものを図 6に示す. BDD は $DC=0$ とした BDD の節点数, BDD Match はドント・ケアを最適化した BDD の節点数, PKDD は $DC=0$ とした PKDD の節点数, PKDD Match はドント・ケアを最適化した PKDD の節点数を示す. $DC=0$ とした BDD の節点数を 1.00 とした時の節点数の割合を棒グラフで表し

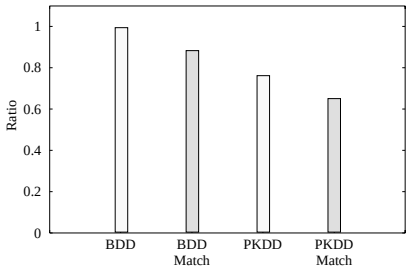

図 6: 各決定グラフの節点数の比較

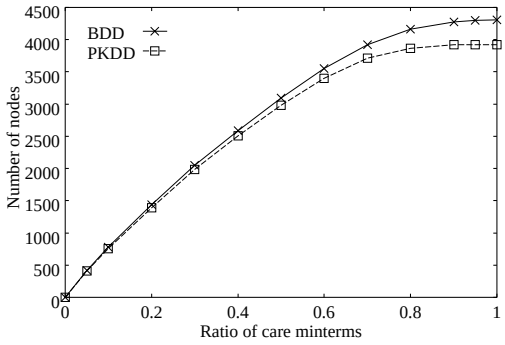

図 7: 乱数関数を表現する決定グラフの大きさ

ている. $DC=0$ とした BDD に比べると, BDD Match は約 11%, PKDD は約 23%, PKDD Match は約 34% 減っている.

5.4 乱数関数

ランダムに不完全定義関数を生成し, BDD, PKDD の節点数を調べた. 最初に真理値表濃度 50% の 15 変数疑似乱数関数 f を作成し, 次に疑似乱数関数 g を作ることで, 不完全定義関数 $[f,g]$ とした. 最後に BDD と PKDD を最小化した. 図 7にケアの濃度に対する BDD と PKDD の大きさを示す. 図 7の結果は 10 個の異なる関数の平均である.

ケア濃度が小さい場合, BDD と PKDD の大きさはほぼ同じであった. しかしながら, ケア濃度が上がると, PKDD と BDD の大きさの差は大きくなった. n 変数の任意の関数を実現するゲート数を $L(n,r)$ とする. ここで r はゲートの最大ファンイン数である. 完全定義関数では, Lupanov[4] は次の関係が成立することを示している.

$$\frac{2^n}{(r-1)n}(1-\varepsilon) < L(n,r) < \frac{2^n}{(r-1)n}(1+o(1)) \qquad (4)$$

これは, n が大きい時成り立つ. 不完全定義関数でも, 彼は下界を得ている. n 変数の任意の関数を実現するゲート数を $L(M,n,r)$ とする. ここで関数は M 個のケア最小項を持つ. すなわち, 2^n-M 個のドント・ケアを持つ.

$$\frac{M}{(r-1)\log_2 M} < L(M,n,r). \qquad (5)$$

式 (4) と (5) から決定グラフの大きさは, $\frac{M}{\log_2 M}$ に比例すると推測できる. 図 7から, 決定グラフの大きさはケア濃度にともなって増加している事がわかる.

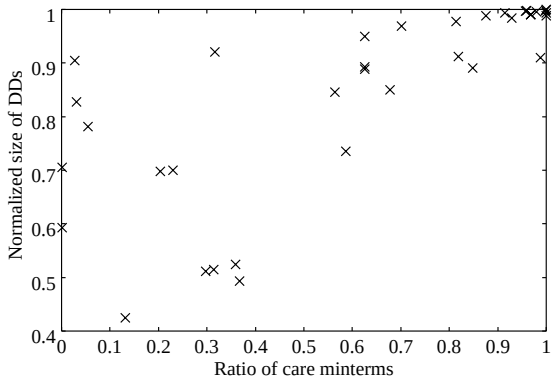

図 8: ベンチマーク関数の BDD の大きさ

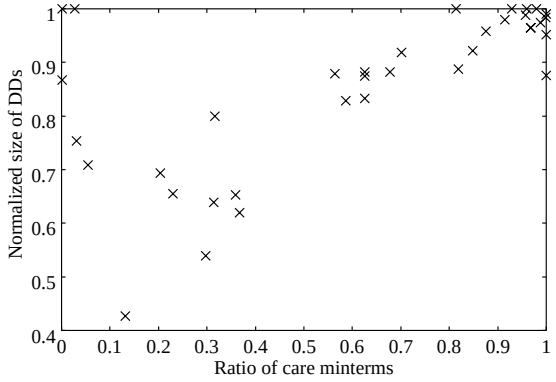

図 9: ベンチマーク関数の PKDD の大きさ

5.5 解析

ドント・ケアが与えられたとき, どの程度決定グラフが削減できるかを考える. ドント・ケアをランダムに生成した関数では, 図 7から削減率を見積もるのは簡単である. しかしベンチマーク関数では, 簡単に見積もることはできない. 解析のため, それぞれのベンチマーク関数に対して, 次に示す (α, β_{bench}) を計算した.

$$\alpha = \frac{\text{ケア最小項の数}}{\text{最小項の数}}$$

$$\beta_{bench} = \frac{\text{ドント・ケアを考慮した決定グラフの大きさ}}{\text{ドント・ケアを考慮しない決定グラフの大きさ}}.$$

ここで, α はケア濃度, β_{bench} はドント・ケアを考慮した場合と, 考慮しない場合の決定グラフの大きさの比である.

図 8と 9は, BDD と PKDD それぞれの (α, β_{bench}) の分布を示している. 大まかな傾向としては, "ケア濃度が増加すると決定グラフは大きくなる" といえる.

例外として, apla, dk17, dk27, dk48, mark1 がある. これらの決定グラフは非常に小さい. これらは図 8と 9の中で, 左上にある.

6 あとがき

本稿では PKDD で不完全定義関数を表現する方法について述べた. PKDD は BDD よりも節点数が少なく, そのまま多段論理回路にマッピングする事もできる. また, 実験により不完全定義関数のドント・ケアを用いて, PKDD 節点数を削減可能である事を示した.

謝辞

本研究は, 一部, 文部省科学研究費補助金による. 不完全定義関数を表現する BDD の簡単化プログラムのプロトタイプは, 大学院卒業生の杉木一也氏による. 明治大学の井口幸洋助教授には, 色々御助言を頂いた.

参考文献

[1] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchange of variables," *ICCAD-91*, pp. 472-475, Nov. 1991.

[2] P. Lindgren, "Applications fo Decision Diagrams in Digital Ciruit Design," Ph. D. Thesis, Lulea University, Sweden, Dec. 1999.

[3] P. Lindgren, R. Drechsler, and B. Becker, "Minimization of ordered pseudo Kronecker decision diagrams," *Proceedings 2000 International Conference on Computer Design*, 504-10, 2000

[4] O. B. Lupanov, "Concerning one method of for the synthesis of circuits," *Izv. Vyssh. Ucheb. Zavad. Radiofiz.*, 1958.

[5] O. B. Lupanov, "On the possibility of synthesis schemes of arbitrary elements," *Tr. Mat. Inst. V. A. Steklova*, 1958.

[6] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *Proc. ICCAD*, pp. 42-47, 1993.

[7] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton, "Heuristic Minimization of BDDs Using Don't Cares," *Design Automation Conference 1994*, pp. 225-231, 1994.

[8] T. Sasao and J. T. Butler, "A design method for look-up table type FPGA by pseudo-Kronecker expansion," *Proc. International Symposium on Multiple-Valued Logic*, pp. 97-106, May 1994.

[9] T. Sasao, H. Hamachi, S. Wada, and M. Matsuura, "Multi-level logic synthesis based on pseudo-Kronecker decision diagrams and logical transformation," *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansions in Circuit Design (Reed-Muller'95)*, Makuhari, Japan, Aug. 27-29, 1995.

[10] T. Sasao and M. Fujita (ed.), *Representation of Discrete Functions*, Kluwer Academic Publishers, Boston, 1996.

[11] T. Sasao, "Representation of logic functions using EXOR operators," Chapter 2 in [10].

[12] 笹尾勤, 栗元憲一, 松浦宗寛, "擬似クロネッカ決定グラフを用いた FPGA 設計手法", 電子情報通信学会 VLD 研究会, 琵琶湖ワークショップ 2000.

[13] M. Sauerhoff and I. Wegener, "On the complexity of minimizing the OBDD size for incompletely specified functions," *IEEE Trans. Computer-Aided Design*, Vol. 15, pp. 1435-1437, Nov. 1996.

[14] S. Yang, "Logic synthesis and optimization benchmark user guide," Version 3.0, MCNC, Jan. 1991.