社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ヘテロジニアス MDD を用いた論理関数の表現法

永山　　忍†　　笹尾　　勤†,††

† 九州工業大学大学院 情報創成専攻　〒 820–8502 福岡県飯塚市川津 680–4
†† 九州工業大学 マイクロ化総合技術センタ　〒 820–8502 福岡県飯塚市川津 680–4
E-mail: †nagayama@aries02.cse.kyutech.ac.jp, ††sasao@cse.kyutech.ac.jp

あらまし　本論文では, ヘテロジニアス MDD (Multi-valued Decision Diagram) を用いた論理関数の表現法を提案する. ヘテロジニアス MDD は MDD(k) (Multi-valued Decision Diagram with k bits) を一般化した表現であり, 一般に各入力変数は異なる定義域を持ってよく, 入力変数の分割を考慮することによって, 同じ論理関数の ROBDD (Reduced Ordered Binary Decision Diaram) よりも少ないメモリ量で表現できる. そして, FBDD (Free Binary Decision Diagram) と同程度のメモリ量で表現できる. また, ROBDD および FBDD より短い平均パス長を有する.

キーワード　ヘテロジニアス MDD, ROBDD, MDD(k), FBDD, メモリ量, 平均パス長, 分枝限定法

Compact Representations of Logic Functions using Heterogeneous MDDs

Shinobu NAGAYAMA† and Tsutomu SASAO†,††

† Department of Computer Science and Electronics, Kyushu Institute of Technology　Kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan
†† Center for Microelectronic Systems, Kyushu Institute of Technology　Kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan
E-mail: †nagayama@aries02.cse.kyutech.ac.jp, ††sasao@cse.kyutech.ac.jp

Abstract　In this paper, we propose a compact representation of logic functions using Multi-valued Decision Diagrams (MDDs) that are called heterogeneous MDDs. The heterogeneous MDDs are generalization of MDDs with k bits (MDD(k)s), where each variable may take different domain. By partitioning input variables and representing each partition as a single multi-valued variables, we can produce the heterogeneous MDDs with 16% smaller amount of memory than the Reduced Ordered Binary Decision Diagrams (ROBDDs), and with as small amount of memory as the Free Binary Decision Diagrams (FBDDs). We minimized a large number of benchmark functions to show the compactness of the heterogeneous MDD.

Key words　heterogeneous MDD, ROBDD, MDD(k), FBDD, amount of memory, average path length, branch and bound

1. はじめに

携帯端末や家電製品などに用いる組込みシステムは, コスト, 消費電力, 重さ等の制約で, 使用可能なメモリ量に制限がある. そのため, 限られたメモリ量の制約の下で高速に評価できる論理関数の表現法が求められている.

論理関数の表現法として ROBDD (Reduced Ordered Binary Decision Diagram) [6] が広く知られている. 本論文では, ヘテロジニアス MDD (Multi-valued Decision Diagram) を用いた論理関数の表現法を提案する. ヘテロジニアス MDD は, ROBDD より少ないメモリ量, かつより短いパス長で表現できる. また, 論理関数のコンパクトな表現法として FBDD (Free Binary Decision Diagram) [7], [8] が知られている. 本論文で提案するヘテロジニアス MDD は, FBDD と同程度のメモリ量で表現できる.

この論文は, 第 2 節でヘテロジニアス MDD およびその他の諸定義を行なう. 第 3 節では, ヘテロジニアス MDD に用いる尺度を述べ, 第 4 節では, 尺度に基づいた最適なヘテロジニアス MDD を求めるアルゴリズムを提案する. そして, 第 5 節で実験を用いてヘテロジニアス MDD の有用性を示す.

2. 諸　定　義

本節では, ヘテロジニアス MDD (Multi-valued Decision Diagram) を定義し, 多出力関数の表現法を示す.

2.1 論理関数の表現

論理変数を $X=(x_1,x_2,x_3,\ldots,x_n)$ とする．X の変数の集合を $\{X\}$ で表す．$\{X\}=\{X_1\}\cup\{X_2\}\cup\ldots\cup\{X_u\}$ かつ $\{X_i\}\cap\{X_j\}=\phi\ (i\neq j)$ のとき，$X=(X_1,X_2,\ldots,X_u)$ を X の分割という．また X_i のことを超変数と呼ぶこともある．$|X_i|=k_i\ (i=1,2,\ldots,u)$ とすると，$k_1+k_2+\ldots+k_u=n$ である．このとき論理関数 $f(X)$ は $f(X_1,\ldots,X_u):P_1\times P_2\times P_3\times\ldots\times P_u\rightarrow B$ と表現できる．ここで $P_i=\{0,1,2,\ldots,2^{k_i}-1\}$，$B=\{0,1\}$ である．

また本論文で対象とする論理関数は，冗長な変数を含まないものとする．即ち，冗長な変数があれば，予め除去しておくものとする．

［例 2.1］ 5 つの二値変数 $X=(x_1,x_2,x_3,x_4,x_5)$ の分割 $X=(X_1,X_2)$ を考える．$X_1=(x_1,x_2)$，$X_2=(x_3,x_4,x_5)$ とすると $k_1=2$，$k_2=3$ となり，$P_1=\{0,1,2,3\}$，$P_2=\{0,1,\ldots,7\}$ となる．このとき X_1，X_2 はそれぞれ 4 値変数，8 値変数を表す．5 変数論理関数 $f(X)$ を X_1，X_2 を用いて表現すると $f(X_1,X_2):P_1\times P_2\rightarrow B$ となる． （例終り）

2.2 ヘテロジニアス MDD

BDD (Binary Decision Diagram), ROBDD (Reduced Ordered Binary Decision Diagram) の定義は [6] を，MDD (Multi-valued Decision Diagram), ROMDD (Reduced Ordered Multi-valued Decision Diagram) の定義は [11] を参照されたい．

［定義 2.1］ 入力変数 $X=(x_1,x_2,\ldots,x_n)$ を $X=(X_1,\ldots,X_u)$ と分割したとき，論理関数 $f(X)$ を表現する ROMDD をヘテロジニアス MDD といい，ヘテロ MDD と略記する．本論文では，区別のために，X の分割で $k_1=k_2=\ldots=k_u$ のとき，論理関数 $f(X)$ を表現する ROMDD をホモジニアス MDD といい，ホモ MDD(k) または単に MDD(k) と略記する．ここで $k=k_1=k_2=\ldots=k_u$ である．

ホモ MDD(k) は $f:P^u\rightarrow B$，ヘテロ MDD は $f:P_1\times P_2\times\ldots\times P_u\rightarrow B$，ここで $P=\{0,1,\ldots,2^k-1\}$，$P_i=\{0,1,\ldots,2^{k_i}-1\}$，$B=\{0,1\}$ を表現する．

［定義 2.2］ ホモ MDD(k) では各非終端節点は 2^k 個の枝を持つ．$k=1$ のとき，ホモ MDD(1) は ROBDD を表す．

［定義 2.3］ ヘテロ MDD では，変数 X_i を表現する非終端節点は 2^{k_i} 個の枝を持つ．

［定義 2.4］ DD (Decision Diagram) において，非終端節点の総数を DD の節点数といい，$node(DD)$ と表記する．

［定義 2.5］ 変数 X_i における DD の節点数を DD の X_i における幅と定義し，$width(DD,i)$ で表す．

分割 $(X_1,X_2,\ldots,X_u)$ に対する MDD の節点数は，

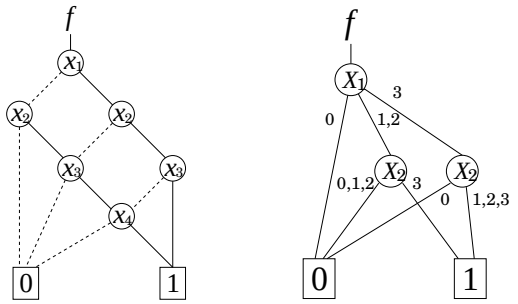
$$node(MDD)=\sum_{i=1}^{u}width(MDD,i)$$

で計算できる．

［例 2.2］ 論理関数 $f=x_1x_2x_3\vee x_2x_3x_4\vee x_3x_4x_1\vee x_4x_1x_2$ を表現する ROBDD を図 2.1(a) に，ホモ MDD(2) を図 2.1(b) に，ヘテロ MDD を図 2.2に示す．図 2.1(a) の ROBDD で実線は 1-枝，破線は 0-枝を示す．また，図 2.1(b) のホモ MDD(2) では $X_1=(x_1,x_2),X_2=(x_3,x_4)$，図 2.2(a) のヘテロ MDD では $X_1=(x_1,x_2,x_3),X_2=(x_4)$，そして図 2.2(b) のヘテロ MDD では $X_1=(x_1),X_2=(x_2,x_3,x_4)$ である． （例終り）

2.3 多出力関数の表現

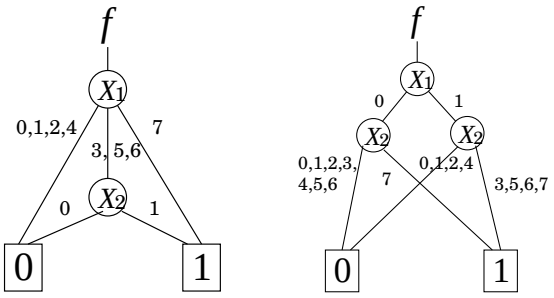
論理回路は，通常多出力である．各出力を独立に表現すると，ほとんどの場合，表現が大きくなりすぎ効率が悪い．従って，能率の良い表現法の考案が重要である．多出力関数を，$F=(f_0,f_1,\ldots,f_{m-1}):B^n\rightarrow B^m,B=\{0,1\}$ と定義する．ここで，n は論理回路の入力数，m は出力数を表す．本論文では，SBDD (Shared Binary Decision Diagram) [18] を用いて多出力関数を表現する．以下 BDD は，特に指定がない限り SROBDD (Shared ROBDD) を示し，ホモ MDD(k) とヘテロ MDD は，この BDD より生成されたものと仮定する．



(a) BDD (b) ホモ MDD(2)

図 2.1 BDD とホモ MDD(2)

Fig. 2.1 BDD and homo-MDD(2)



(a) メモリ量最小の場合 (b) メモリ量最大の場合

図 2.2 ヘテロ MDD

Fig. 2.2 hetero-MDDs

3. ヘテロ MDD の尺度

［補題 3.1］ $X=(x_1,x_2,\ldots,x_n)$ の変数順序を固定したと仮定したとき，X の分割の個数は 2^{n-1} 通り存在する．(注 1)

従って，MDD も 2^{n-1} 種類存在する．この中から，使用目的に応じて最適なヘテロ MDD を構成する必要がある．本節では，最適なヘテロ MDD を構成するための基準となる尺度をいくつか示す．

3.1 メモリ量

［定義 3.1］ 本論文では，DD をメモリに格納するために必要なワード数を**メモリ量**と定義する．

［定義 3.2］ 関数 f が与えられたとき，メモリ量が最小のヘテロ MDD を関数 f の**最小ヘテロ MDD** と呼ぶ．

RODD (Reduced Ordered Decision Diagram) の各節点を表現するには，インデックスと各枝のアドレスが必要である．BDD の各節点は 2 本の枝をもつため，BDD の全節点を表現するためのメモリ量は，

$$(2+1)\times node(BDD)$$

である．MDD(k) の各節点は 2^k 本の枝を持つため，MDD(k) の全節点を表現するためのメモリ量は，

$$(2^k+1)\times node(MDD(k))$$

である．ヘテロ MDD では変数毎に枝数が異なる．ヘテロ MDD の全節点を表現するためのメモリ量は

(注 1)：本論文では，ページ数の制限のために証明を割愛する．

$$\sum_{i=1}^{u}(2^{k_i}+1)\times width(\text{ヘテロ } MDD, i)$$

である.

[例 3.1] 図 2.1(a) の BDD のメモリ量は 18 である. 図 2.1(b) の MDD(2) のメモリ量は 15 である. そして図 2.2(a) のヘテロ MDD のメモリ量は 12, 図 2.2(b) のヘテロ MDD のメモリ量は 21 である. (例終り)

[定理 3.1] 最小ヘテロ MDD では, 全ての超変数 $X_i = (x_j, x_{j+1}, \ldots, x_{j+k_i-1})$ において

$$(2^{k_i}+1)width(\text{ヘテロ } MDD, i) \leqq 3\times\sum_{t=0}^{k_i-1} width(BDD, j+t)$$

が成立する.

[定理 3.2] 真に n 変数に依存する論理関数 f を表現するヘテロ MDD を考える. ヘテロ MDD に必要なメモリ量を $M_{min}(n)$ とすると

$$M_{min}(n) \geqq 2.5n$$

が成立する.

[系 3.1] 真に n 変数に依存する論理関数 f を表現するヘテロ MDD を考える. ヘテロ MDD の部分グラフに必要なメモリ量を $M_{sub}=\sum_{i=j}^{u}(2^{k_i}+1)width(\text{ヘテロ } MDD, i)$ とすると

$$M_{sub} \geqq 2.5n_j$$

が成立する. ここで n_j は部分グラフに現われる二値変数の数を表す.

[定理 3.3] 真に n 変数に依存する論理関数を f とする. f を表現するヘテロ MDD のメモリ量を $Mem(\text{ヘテロ } MDD : f)$ とすると, $n>1$ のとき

$$Mem(\text{ヘテロ } MDD : f) \geqq node(BDD)+2$$

が成立する.

[系 3.2] 真に n 変数に依存する論理関数を f とする. f を表現するヘテロ MDD の部分グラフにおけるメモリ量を $Mem_{sub}($ヘテロ $MDD : f)=\sum_{i=j}^{u} width(\text{ヘテロ } MDD, i)$ とすると, $n_j>1$ のとき

$$Mem_{sub}(\text{ヘテロ } MDD : f) \geqq \sum_{t=1}^{n_j} width(BDD, t)+2$$

が成立する. ここで n_j は部分グラフに現れる二値変数の数を表す.

[定理 3.4] 真に n 変数に依存する論理関数 f を表現するヘテロ MDD を考える. 最小ヘテロ MDD のメモリ量を $M_{max}(n)$ とすると, 以下の関係が成立する.

$$M_{max}(n) \leqq 2^{n-r}+3\cdot 2^{2^r}-5$$

ここで r は, 以下の関係を満足する最大の整数である.

$$n-r \geqq 2^r + log_2 3$$

3.2 平均パス長

[定義 3.3] DD において, 根から終端節点までの経路を DD のパスという. このときパス上の非終端節点数をパス長という.

[定義 3.4] DD において, パスが非終端節点 v_i を通過する確率を節点通過確率とよび, $prob(DD, i)$ で表す.

本論文では, 以下のような計算モデルを仮定する.

- 各変数の値に従い, DD 上の節点を辿ることで論理関数の評価を行う.
- MDD は BDD パッケージを使わずに, 直接実現する [11].
- MDD の各変数に対して, 符号化された入力値が入力される. そのため入力値の変換時間は無視できる. 例えば, $X_1=(x_1,x_2,x_3,x_4)=(1,0,0,1)$ のときは, $X_1=9$ が入力値として与えられる.
- MDD を用いた論理関数の評価では, 節点のアクセスに主な時間を費やす.
- 節点の評価時間は全て等しい.

このとき, DD で表現した論理関数の評価時間は, アクセスした非終端節点の個数 (パス長) に比例する. また, 入力変数 x_i に 0 と 1 が等確率で入力されると仮定する. このような仮定の下で, DD の評価時間を見積もるために平均パス長を使う.

本論文では, 多出力関数の表現法に SDD (Shared Decision Diagram) を用いる. 多出力関数 $F=(f_0, f_1, \ldots, f_{m-1})$ を表現する SDD の平均パス長は, 各単一出力関数 f_i を表現する DD の平均パス長の合計で求められる [20].

[例 3.2] 図 2.1(a) の BDD の平均パス長は, 3.125 である. 図 2.1(b) のホモ MDD(2) の平均パス長は, 1.75 である. そして図 2.2(a) のヘテロ MDD の平均パス長は 1.375, 図 2.2(b) のヘテロ MDD の平均パス長は, 2 である. (例終り)

[定理 3.5] DD の平均パス長は, 節点通過確率の合計に等しい [20].

[定理 3.6] 一つの論理関数を表現する BDD とヘテロ MDD を考えるとき,

$$(\text{ヘテロ } MDD \text{ の平均パス長}) \leqq (BDD \text{ の平均パス長})$$

が成立する.

4. ヘテロ MDD の最適化

本節では, 第 3.節の尺度に基づいたヘテロ MDD 最適化問題の定式化とそれを解くためのアルゴリズムを示す. ある尺度を最適にする解が, 他の尺度を最適にする解になるとは限らないため, 尺度別の最適化アルゴリズムが必要となる.

4.1 メモリ量の最小化

入力変数の順序を固定したと仮定すると, ヘテロ MDD のメモリ量は, 入力変数 X の分割の仕方に依存する. そのため関数毎に, メモリ量を最小にする X の分割を見つける必要がある.

[例 4.1] 図 2.2(a) は, 最小ヘテロ MDD である. 一方, 図 2.2(b) は, メモリ量最大のヘテロ MDD である. (例終り)

メモリ量の最小化問題を次のように定式化する.

[問題 4.1] 論理関数 f を表現する BDD が与えられたとき, 最小ヘテロ MDD を構成する変数 X の分割を求めよ. ここで, 元の BDD の変数順序は変化しないと仮定する.

n 変数論理関数を表現する最適ヘテロ MDD を求めるには, 2^{n-1} 個のヘテロ MDD を構成し, その中から最適なものを選択すればよい. n が小さいとき実用時間内に解が求まる. n が大きい場合には, この手法は時間がかかり過ぎるので次のアルゴリズムを考案する. 問題 4.1を解くための擬似コードをアルゴリズム 4.1 に示す. このアルゴリズムは分枝限定法 (branch-and-bound) に基づく.

[アルゴリズム 4.1] (メモリ量の最小化)

```
1: 最小メモリ量 = BDD のメモリ量;
2: minimize_memory (残りの入力変数の個数,
   超変数 X_i のインデックス i, 使用メモリ量) {
3:    if (残りの入力変数の個数 == 0) {
4:       if (使用メモリ量 < 最小メモリ量) {
5:          最小メモリ量 = 使用メモリ量;
6:          最小にする分割 = 現在の分割;
7:       }
8:    }
```

```
9:    else {
10:     for (s = 1; s < 残りの入力変数の個数; s++) {
11:       k = branch[s] ;
12:       if ((2^k + 1) × width(ヘテロ MDD, i) >
            3 × Σ_{t=0}^{k-1} width(BDD, j + t))
13:         continue ;
14:       使用メモリ量 = 使用メモリ量
            + (2^k + 1) × width(ヘテロ MDD, i) ;
15:       残りのメモリ量 = 最小メモリ量 − 使用メモリ量;
16:       残りの入力変数の個数 = 残りの入力変数の個数 − k ;
17:       if (memory_size_check (残りの入力変数の個数,
            残りのメモリ量) == メモリ容量内) {
18:         現在の分割.k_i = k ;
19:         minimize_memory (残りの入力変数の個数, i + 1,
              使用メモリ量) ;
20:       }
21:     }
22:   }
23:}
24:memory_size_check (残りの入力変数の個数, 残りのメモリ量) {
25:   lower_bound = max((2.5 × 残りの入力変数の個数),
        (Σ_{t=j}^{n} width(BDD, t)+2)) ;
26:   if (残りのメモリ量 < lower_bound)
27:     return メモリオーバ;
28:   else
29:     return メモリ容量内;
30:}
```

アルゴリズム 4.1は, 超変数 X_1 からトップダウン的にメモリ量を計算し, 最適解を探索する. メモリ量が初期値より少ないときのみ初期値を更新するため, 例えば初期値と同じメモリ量の解が見つかったとしてもその解は棄却される. アルゴリズム 4.1の 11 行目は, 分枝 (branch) を行なっている.

$$ratio = \frac{(2^k+1)width(\text{ヘテロ } MDD, i)}{3 \times \sum_{t=0}^{k-1} width(BDD, j+t)}$$

が最も小さくなる k から順に貪欲に分枝していく. また, 12 行目は, 計算量を削減するために定理 3.1 を用いている. *memory_size_chech* 関数内 (25, 26 行目) では定理 3.2, 系 3.1, 定理 3.3, および系 3.2を用いている. このアルゴリズムは問題 4.1に対する最適解を保証する. アルゴリズムの計算量の更なる削減は, 今後の課題とする.

4.2 平均パス長の最小化

ヘテロ MDD の平均パス長も, 入力変数 X の分割の仕方に依存する. 平均パス長を最小にする X の分割は, $X = X_1, k_1 = n$ (1 つの超変数からなる分割) である. しかしこのときのヘテロ MDD のメモリ量は $(2^n + 1) \times width(\text{ヘテロ } MDD, 1)$ となる. そのため与えられたメモリ容量で平均パス長が最小となる X の分割を見つける必要がある. 平均パス長最小化問題を次のように定式化する.

[問題 4.2] 論理関数 f を表現する BDD とメモリ容量 L が与えられたとき, メモリ量が L 以下で, かつ平均パス長最小のヘテロ MDD を構成する変数 X の分割を求めよ. ここで, 元の BDD の変数順序は変化しないと仮定する.

問題 4.2を解くための擬似コードをアルゴリズム 4.2に示す. 本アルゴリズムも同様に分枝限定法に基づく.

[アルゴリズム 4.2] (平均パス長の最小化)

```
1: 最小平均パス長 = BDD の平均パス長;
2: minimize_path (残りの入力変数の個数,
   超変数 X_i のインデックス i, 使用メモリ量, 現在の平均パス長) {
3:   if (残りの入力変数の個数 == 0) {
4:     if (現在の平均パス長 < 最小平均パス長) {
5:       最小平均パス長 = 現在の平均パス長;
6:       最小にする分割 = 現在の分割;
7:     }
8:   }
9:   else {
10:    for (k = 残りの入力変数の個数; k ≧ 1; k−−) {
11:      使用メモリ量 = 使用メモリ量 + (2^k + 1)
           × width(ヘテロ MDD, i) ;
12:      残りのメモリ量 = メモリ容量 L − 使用メモリ量;
13:      残りの入力変数の個数 = 残りの入力変数の個数 − k ;
14:      for (j = 0; j < width(ヘテロ MDD, i); j++)
15:        現在の平均パス長 = 現在の平均パス長
             + prob(ヘテロ MDD, base_address + j) ;
16:      if (memory_size_check (残りの入力変数の個数,
           残りのメモリ量) == メモリ容量内) {
17:        現在の分割.k_i = k ;
18:        minimize_path (残りの入力変数の個数, i + 1,
             使用メモリ量, 現在の平均パス長) ;
19:      }
20:    }
21:  }
22:}
```

アルゴリズム 4.2もアルゴリズム 4.1と同様に, 超変数 X_1 からトップダウン的にメモリ量および平均パス長を計算し, 最適解を探索する. このとき平均パス長の計算は, 定理 3.5 に基づいている. 15 行目の節点通過確率 $prob$(ヘテロ $MDD, base\_address + j$) の計算は, [20] を参照されたい. また $base\_address$ は, 超変数 X_i における節点の先頭アドレスを示している. アルゴリズム 4.2は問題 4.2の最適解を求めるが, 計算時間はアルゴリズム 4.1より大きい.

5. 実験結果

5.1 FBDD との比較

表 5.1は, 最小ヘテロ MDD と FBDD (Free BDD) のメモリ量での比較結果を示している. 表 5.1の OBDD (Ordered BDD) の節点数は [21], FBDD の節点数は [7], [8] より得た結果である. ヘテロ MDD は, この OBDD より生成した. これらの DD は否定枝を用いていることに注意. また OBDD, FBDD のメモリ量は ROBDD のメモリ量を求める公式により計算した. 表中の "時間" は, 最適解が求まるまでの CPU 時間を表している. 以下の性能を持つ計算機上でアルゴリズム 4.1を実装し, CPU 時間を測定した.

- CPU: Pentium III 1GHz
- 1 次キャッシュ: 32KB
- 2 次キャッシュ: 256KB
- 主メモリ: 4GB
- OS: redhat Linux 7
- コンパイラ: gcc -O

また, 表中の "比率の平均" は, OBDD のメモリ量を 1.00 としたときの各 DD のメモリ量の比率の算術平均を示す. 表 5.1より, 最小ヘテロ MDD は FBDD と同程度のメモリ量で論理関数を表現できるといえる. また, アルゴリズム 4.1は実用時間内に 200 入力程度の関数まで最適解を求めることができた.

5.2 BDD とホモ MDD(k) との比較

表 5.2は, 各種ヘテロ MDD と BDD, ホモ MDD(k) (k=2, 3, 4, 5) との比較結果を示している. これらの DD は否定枝を用いていない. また, この実験には表 A.1のベンチマーク関数を用いて総合的な比較を行った. 表 5.2のヘテロ MDD1 は, アルゴリズム 4.1 により求めた最小ヘテロ MDD であり, ヘテロ MDD2 は, アルゴリズム 4.2により求めた平均パス長最小のヘテロ MDD (BDD のメモリ量を上限) である. 表 5.2の各値は, BDD のメモリ量を 1.00 としたときの算術平均を表す.

ヘテロ MDD は平均して, BDD の約 84% のメモリ量で実現できる. 特に, 関数 ex1010 を表現するヘテロ MDD は, BDD の約 46% のメモリ量で実現できた.

ヘテロ MDD は, BDD と同じメモリ量で実現すると, 平均パス長を BDD より, 平均して約 58% にすることができる. またメモリ量最小のヘテロ MDD でも, 平均パス長は BDD の平均

表 5.1　メモリ量最小のヘテロ MDD と FBDD のメモリ量での比較
Table 5.1　Amount of memory for hetero-MDDs and FBDDs

関数名	入力数	出力数	非終端節点数		メモリ量			時間 [sec]
			OBDD	FBDD	OBDD	FBDD	ヘテロ MDD	
C432	36	7	1063	1057	3189	3171	2910	0.02
C499	41	32	25865	25865	77595	77595	59739	0.94
C880	60	26	4052	2798	12156	8394	11934	0.01
C1908	33	25	5525	5047	16575	15141	13493	0.04
C2670	233	64	1771	1062	5313	3186	4805	427.62
C3540	50	22	23827	20999	71481	62997	65198	0.01
C5315	178	123	1718	1478	5154	4434	4855	0.01
C7552	207	107	2159	1594	6477	4782	6462	–
alu4	14	8	349	300	1047	900	855	0.00
apex1	45	45	1245	1177	3735	3531	3117	0.01
apex6	135	99	490	455	1470	1365	1437	0.01
cps	24	102	970	902	2910	2706	2568	0.00
dalu	75	16	688	649	2064	1947	1574	0.01
des	256	245	2944	2902	8832	8706	7589	126.77
frg2	143	139	961	920	2883	2760	2773	0.01
i3	132	6	132	132	396	396	332	0.02
i8	133	81	1275	1190	3825	3570	3825	0.00
i10	257	224	20659	18813	61977	56439	58861	–
k2	45	45	1245	1136	3735	3408	3119	0.00
too_large	38	3	318	286	954	858	859	0.03
vda	17	39	477	467	1431	1401	1088	0.00
				比率の平均	1.00	0.90	0.89	

–: 3600 秒で打ち切り

表 5.2　各種ヘテロ MDD と BDD, ホモ MDD(k) との比較
Table 5.2　Comparison with various hetero-MDDs, BDDs, and homo-MDD(k)s

DD	メモリ量	平均パス長
BDD	1.00	1.00
ヘテロ MDD1	0.84	0.69
ヘテロ MDD2	0.97	0.58
ホモ MDD(2)	1.08	0.69
ホモ MDD(3)	1.54	0.53
ホモ MDD(4)	2.39	0.50
ホモ MDD(5)	4.10	0.46

パス長以下になる. アルゴリズム 4.2は, 実用時間内に 60 入力程度の関数までの最適解を求めることができる.

6. 結論とコメント

本論文では, ヘテロジニアス MDD (Multi-valued Decision Diagram) を用いた論理関数の表現法を提案した. また, メモリ量および平均パス長を最小化するためのアルゴリズムを提案し, プログラムを実装した. 様々なベンチマーク関数に対して実験を行ない, ヘテロ MDD に関して以下のことを示した. 1) FBDD (Free BDD) と同程度のメモリ量でヘテロ MDD を構成できる. 2) BDD の 84% のメモリ量で, かつ 69% の平均パス長を持つヘテロ MDD を構成できる.

特に, ヘテロ MDD は変数順序を変更することも, 否定枝を用いることもなく BDD の 84% という少ないメモリ量で実現できる. また, 最小化プログラムも FBDD に比べ高速で, 各使用目的に応じた最適なヘテロ MDD を比較的容易に構成できる.

またヘテロ MDD は, 面積時間複雑度 [4], [22] や最大パス長を尺度として用い, それらに対する最適化を行なうことも可能である.

本論文で提示したアルゴリズムは 200 入力程度までの関数に対しては最適解を与える. それより大きい関数に対しては, 分枝限定法を用いているため, 適当な時間で計算を打ち切ることにより, 近似解を求めることができる. 本論文では入力変数の順序は固定している. 入力変数の順序の変更までを考慮に入れた最適解を求めるためには, アルゴリズムの改良や発見的手法の考案などが必要となる. これについては今後の研究課題とする.

謝　　辞

本研究は一部, 日本学術振興会, 科学研究費および武田計測先端知財団補助金による.

文　　献

[1] P. Ashar and S. Malik, “Fast functional simulation using branching programs,” *ICCAD’95*, pp. 408–412, Nov. 1995.

[2] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli, E. M. Sentovich, and K. Suzuki, “Synthesis of software programs for embedded control applications,” *IEEE Trans. CAD*, Vol. 18, No. 6, pp.834-849, June 1999.

[3] B. Becker and R. Drechsler, “Efficient graph based representation of multivalued functions with an application to genetic algorithms,” *Proc. of International Symposium on Multiple Valued Logic*, pp. 40-45, May 1994.

[4] R. P. Brent and H. T. Kung, “The area-time complexity of binary multiplication,” *Journal of the ACM*, Vol. 28, No. 3, pp. 521-534, July 1981.

[5] F. Brglez and H. Fujiwara, “Neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN,” *Special session on ATPG and fault simulation, Proc. IEEE Int. Symp. Circuits and Systems*, June 1985, pp. 663-698.

[6] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug. 1986.

[7] W. Günther and R. Drechsler, “Minimization of free BDDs,“ *Asia and South Pacific Design Automation Conference (ASP-DAC’99)*, pp.323-326, Jan. 18-21, 1999, Wanchai, Hong Kong.

[8] W. Günther, “Minimization of free BDDs using evolutionary techniques,“ *International Workshop on Logic Synthesis 2000 (IWLS-2000)*, pp.167-172, May 31-June 2, 2000, Loguna Cliffs Marriott, Dana Point, CA.

[9] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno “A hardware simulation engine based on decision diagrams,” *Asia and South Pacific Design Automation Conference (ASP-DAC'2000)*, Jan. 26-28, Yokohama, Japan.

[10] Y. Iguchi, T. Sasao, M. Matsuura, “Implementation of multiple-output functions using PQMDDs,” *International Symposium on Multiple-Valued Logic*, pp.199-205, May 2000.

[11] T. Kam, T. Villa, R. K. Brayton, and A. L. Sagiovanni-Vincentelli, “Multi-valued decision diagrams: Theory and Applications,” *Multiple-Valued Logic*, 1988, Vol. 4, No. 1-2, pp. 9–62, 1998.

[12] C. Kim, L. Lavagno, and A. S-Vincentelli, “Free BDD-based software optimization techniques for embedded systems,” *Design, Automation and Test in Europe (DATE2002)*, Paris, pp.14-19, March 2000.

[13] H.-T. Liaw, and C.-S. Lin. “On the OBDD-representation of general Boolean function,” *IEEE Transactions on Computers*, Vol. 4, No. 6, pp. 661–664, June 1992.

[14] P. C. McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, “Fast discrete function evaluation using decision diagrams,” *ICCAD'95*, pp. 402–407, Nov. 1995.

[15] S. Minato, N. Ishiura, and S. Yajima, “Shared binary decision diagram with attributed edges for efficient Boolean function manipulation,” *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 52–57, June 1990.

[16] D. M Miller, “Multiple-valued logic design tools,” *Proc. of International Symposium on Multiple Valued Logic*, pp. 2–11, May 1993.

[17] S. Nagayama, T. Sasao, Y. Iguchi, and M. Matsuura, “Representations of logic functions using QRMDDs,” *IEEE International Symposium on Multiple-Valued Logic*, pp. 261-267, Boston, Massachusetts, U.S.A, May 15-18, 2002.

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

[19] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[20] T. Sasao, Y.Iguchi, and M. Matsuura, “Comparison of decision diagrams for multiple-output logic functions,” *International Workshop on Logic and Synthesis*, New Orleans, Louisiana, June 4-7, 2002, pp.379-384.

[21] F. Somenzi, “CUDD: CU Decision Diagram Package Release 2.3.1,“ University of Colorado at Boulder, 2001.

[22] C. D. Thompson, “Area-Time complexity for VLSI,” *Ann. Symp. on Theory of Computing*, May 1979.

[23] S. Yang, *Logic synthesis and optimization benchmark user guide version 3.0*, MCNC, Jan. 1991.

付　　録

A. ベンチマーク関数

第 5.2節の実験に用いた 238 個のベンチマーク関数 [5], [19], [23] を表 A.1に示す. 表中の n と m は, それぞれ論理回路の入力数, 出力数を表す. 括弧内の数字は, 真に依存する変数の個数を表す. また表中の *sequential* 以降のベンチマーク関数は, 本来, 順序回路を表現する関数である. しかし, 本論文では, 組合せ回路のみを対象としているので, 順序回路の FF (Flip-Flop) を取り除き, それを入出力とした組合せ回路に変換する. 本来の関数名に c を付け加え, 変換後の組合せ回路の関数名にしている. 表 A.1の各関数の各実験データは論文のページ数の制限のために割愛する.

表 A.1　実験に用いたベンチマーク関数

Table A.1　Benchmark Functions

関数名	n	m	関数名	n	m	関数名	n	m
3adr6	18 (18)	12	i1	25 (25)	16	radd	8 (8)	5
5xp1	7 (7)	10	ibm	48 (48)	17	rckl	32 (32)	7
9sym	9 (9)	1	in0	15 (15)	11	rd53	5 (5)	3
C432	36 (36)	7	in1	16 (15)	17	rd73	7 (7)	3
C499	41 (41)	32	in2	19 (19)	10	rd84	8 (8)	4
C1355	41 (41)	32	in3	35 (34)	29	rdm16	16 (16)	16
C1908	33 (33)	25	in4	32 (32)	20	rdm17	17 (17)	17
C3540	50 (50)	22	in5	24 (24)	14	rdm18	18 (18)	18
accpla	50 (50)	69	in6	33 (33)	23	rdm8	8 (8)	8
add6	12 (12)	7	in7	26 (26)	10	risc	8 (8)	31
addm4	9 (9)	8	inc	7 (7)	9	root	8 (8)	5
adr2	4 (4)	3	inc16	16 (16)	17	rot16	16 (16)	9
adr3	6 (6)	4	inc17	17 (17)	18	rot17	17 (17)	9
adr4	8 (8)	5	inc18	18 (18)	19	rot18	18 (18)	10
adr8	16 (16)	9	intb	15 (15)	7	rot8	8 (8)	5
adr9	18 (18)	10	jbp	36 (36)	57	ryy6	16 (16)	1
al2	16 (16)	47	k2	45 (45)	45	sao2	10 (10)	4
alcom	15 (15)	38	l8err	8 (8)	8	sct	19 (19)	15
alu1	12 (12)	8	lal	26 (26)	19	seq	41 (41)	35
alu2	10 (10)	6	life	9 (9)	1	sex	9 (9)	14
alu3	10 (10)	8	linrom	7 (7)	36	shift	19 (19)	16
alu4	14 (14)	8	log16	16 (16)	16	signet	39 (39)	8
amd	14 (14)	24	log17	17 (17)	17	spla	16 (16)	46
apex1	45 (45)	45	log18	18 (18)	18	sqn	7 (7)	3
apex2	39 (30)	3	log8	8 (8)	8	sqr6	6 (6)	12
apex3	54 (54)	50	log8mod	8 (7)	5	sqr8	8 (8)	16
apex4	9 (9)	19	luc	8 (8)	27	sqr16	16 (16)	32
apex7	49 (49)	37	m1	6 (6)	12	sym4	4 (4)	1
apla	10 (10)	12	m2	8 (8)	16	sym6	6 (6)	1
b2	16 (15)	17	m3	8 (8)	16	sym10	10 (10)	1
b3	32 (32)	20	m4	8 (8)	16	t1	21 (21)	23
b4	33 (33)	23	m181	15 (15)	9	t2	17 (17)	16
b7	8 (8)	31	mainpla	27 (26)	54	t3	12 (12)	8
b9	41 (41)	21	mark1	20 (20)	31	t4	12 (11)	8
b10	15 (15)	11	max46	9 (9)	1	t481	16 (16)	1
b11	8 (8)	31	max128	7 (7)	24	table3	14 (14)	14
b12	15 (15)	9	max512	9 (9)	6	table5	17 (17)	15
bc0	26 (21)	11	max1024	10 (10)	6	tcon	17 (17)	16
bca	26 (16)	46	misex1	8 (8)	7	term1	34 (34)	10
bcb	26 (16)	39	misex2	25 (25)	18	ti	47 (43)	72
bcc	26 (16)	45	misex3	14 (14)	14	tial	14 (14)	8
bcd	26 (16)	38	misg	56 (56)	23	tms	8 (8)	16
bcddiv	4 (4)	4	misj	35 (35)	14	too_large	38 (38)	3
bench	6 (6)	8	mlp4	8 (8)	8	ts10	22 (22)	16
bench1	9 (9)	9	mlp6	12 (12)	12	ttt2	24 (24)	21
br1	12 (12)	8	mlp8	16 (16)	16	unreg	36 (36)	16
br2	12 (12)	8	mlp9	18 (18)	18	vda	17 (17)	39
bw	5 (5)	28	mlp10	20 (20)	20	vg2	25 (25)	8
c8	28 (28)	18	mp2d	14 (14)	14	vtx1	27 (27)	6
cc	21 (21)	20	mux	21 (21)	1	wgt17	17 (17)	5
chkn	29 (29)	7	my_adder	33 (33)	17	wgt18	18 (18)	5
cht	47 (47)	36	newapla	12 (12)	10	wim	4 (4)	7
clip	9 (9)	5	newapla1	12 (12)	7	x1	51 (51)	35
clpl	11 (11)	5	newapla2	6 (6)	7	x1dn	27 (27)	6
cm150a	21 (21)	1	newbyte	5 (5)	8	x6dn	39 (38)	5
co14	14 (14)	1	newcond	11 (11)	2	x9dn	27 (27)	7
comp	32 (32)	3	newcpla1	9 (9)	16	xor5	5 (5)	1
con1	7 (7)	2	newcpla2	7 (7)	10	xparc	41 (39)	73
cordic	23 (23)	2	newcwp	4 (4)	5	z4	7 (7)	4
count	35 (35)	16	newill	8 (8)	1	sequential		
cps	24 (24)	109	newtag	8 (8)	1	$s27_c$	7 (7)	4
dc1	4 (4)	7	newtpla	15 (15)	5	$s208_c$	18 (18)	9
dc2	8 (8)	7	newtpla1	10 (10)	2	$s298_c$	17 (17)	20
dekoder	4 (4)	7	newtpla2	10 (10)	4	$s344_c$	24 (24)	26
dist	8 (8)	5	newxcpla	9 (9)	23	$s349_c$	24 (24)	26
dk17	10 (10)	11	nrm4	8 (8)	5	$s382_c$	24 (24)	27
dk27	9 (8)	9	nrm8	16 (16)	9	$s386_c$	13 (13)	13
dk48	15 (15)	17	nrm9	18 (18)	10	$s400_c$	24 (24)	27
duke2	22 (22)	29	opa	17 (17)	69	$s420_c$	34 (34)	17
ex1010	10 (10)	10	p1	8 (8)	18	$s444_c$	24 (24)	27
ex5	8 (8)	63	p3	8 (8)	14	$s510_c$	25 (25)	13
ex7	16 (16)	5	p82	5 (5)	14	$s526_c$	24 (24)	27
exam	10 (10)	10	pcle	19 (19)	9	$s641_c$	54 (54)	42
exep	30 (28)	63	pcler8	27 (27)	17	$s713_c$	54 (54)	42
exp	8 (8)	18	pdc	16 (16)	40	$s820_c$	23 (23)	24
exps	8 (8)	38	pm1	16 (16)	13	$s832_c$	23 (23)	24
f51m	8 (8)	8	pope	6 (6)	48	$s1196_c$	32 (31)	32
fout	6 (6)	10	poperom	6 (6)	48	$s1488_c$	14 (14)	25
frg1	28 (28)	3	prom1	9 (9)	40	$s1494_c$	14 (14)	25
gary	15 (15)	11	prom2	9 (9)	21			