計算機アーキテクチャ　103－8
設　計　自　動　化　69－8
(1993. 12. 16)

2分決定グラフの構造に着目したブーリアン・マッチング処理について

松永　裕介
(株) 富士通研究所
〒211 川崎市中原区上小田中1015

あらまし　2分決定グラフの部分グラフの構造比較を用いて探索空間を狭める枝刈り手法にもとづくブーリアン・マッチングのアルゴリズムについて述べる。既存の枝刈り手法と組み合わせることで高速な処理を行なうことが可能となっている。また、テクノロジ・マッピングの中で用いる際には、マッピングを行なうセルの機能とはマッチしない部分回路をその構造から判定することにより、無駄なブーリアン・マッチングの回数を削減している。

和文キーワード　2分決定グラフ，ブーリアン・マッチング，テクノロジ・マッチング

An algorithm for Boolean matching utilizing structural information

Yusuke Matsunaga
Fujitsu Laboratories LTD.
1015 Kamikodanaka. Nakahara-Ku. Kasawaki 211

Abstract　The paper describes a new algorithm for Boolean matching, which is based on BDD structure manipulation. Pruning takes place after partial assignments if subgraphs of two BDD's become inequivalent. Another contribution of the paper is efficient filtering of cluster functions. The topological filter omit clusters which has no hope to be matched so that the calculation of such culster functions are avoided.

英文 key words　binary decision diagram, Boolean matching, technology mapping

1 はじめに

テクノロジ非依存の回路（論理式）に一致するようなセルを見つけるマッチング処理はテクノロジ・マッピングの中で最も重要でありまた最も多くの計算時間を必要とする。このマッチングの手法としては、木状の部分グラフに対するパタン・マッチングを利用するものが最初に提案された [1, 2]。この手法では、予めセルの機能を表す木状のパタンが与えられていれば極めて高速に処理を行なうことができる。しかし、一つのセルに対してその機能を表すパタンは唯一とは限らず、特に入力数の多いセルに対してはその全てのパタンを列挙することが不可能な場合もある。

これに代わるマッチングの手法として、Mailhot らは2つの論理関数の一致を調べるブーリアン・マッチング（Boolean matching）を提案した [3]。彼らのアルゴリズムは再帰的な Shannon 展開に基づくもので、入力変数間の対称性の情報を用いて探索空間の縮小を計っている。Burch らはこれとは全く異なるアプローチでブーリアン・マッチングのアルゴリズムを提案している [4]。彼らはまず、位相反転の同値類に関する canonical form を定義して、論理関数を canonical form に変換する手続きを提案した。また、変数順序の入れ換えに関しても semi-canonical form を定義し、同様に変換手続きを提案した。彼らの手法は、まず前処理として与えられたセル・ライブラリのセルの機能を表す論理関数に対する上記の canonical form を計算しておき、実際のマッチング処理では対象となっている関数の canonical form を求めて一致するものを探すというものである。前処理にどれくらいの時間がかかるかは明らかにされていないが、マッチング処理そのものは極めて高速である。ただし、don't care 条件を考慮することはできない。Savoj らのアルゴリズム [5] は don't care 条件を考慮できるもので、存在作用素（$\exists$）を用いた論理関数処理を行なう点に特徴がある。また、真理値表密度が異なる論理関数同士はマッチングしないという性質を用いて枝刈りを行なっている。

ブーリアン・マッチングでは2つの論理関数が入力変数の位相反転および変数順序の入れ換えによって等しくなるかを確かめる必要がある。論理関数を2分決定グラフ（BDD）[6] を用いて表現すれば2つの論理関数の比較自体は容易に行なえるが、それでも最悪 $O(2^n \cdot n!)$ 通りの位相反転／変数順序変換を試さなければならない。そのため、効率よくブーリアン・マッチングを行なうためには効果的な枝刈りの手法が必要となる。本稿では2分決定グラフの構造に着目して枝刈りを行なうブーリアン・マッチングのアルゴリズムについて述べる。この手法は前述の真理値表密度を用いた枝刈りも併用することができるので、探索空間を一層、狭めることができる。

テクノロジ・マッピングのためにブーリアン・マッチングを用いる場合、実際に与えられたセル・ライブラリのいづれかのセルにマッチする論理関数はほんのわずかで、残りの大部分の関数はマッチングを持たない。さらに、そのような関数は真理値表密度を計算するだけでそのことが簡単に判断できることが多い。本稿では、このような関数をその2分決定グラフを作ることなく判定する手法についても述べる。これは、回路の構造から真理値表密度の近似値を計算するもので、2分決定グラフを作成する時間に比べて極めて高速に計算を行なうことができる。

以下、2章で2分決定グラフの構造に基づくブーリアン・マッチングアルゴリズムについて述べ、3章で回路の構造を用いてマッチングの候補を削減する方法（トポロジカル・フィルター）について述べる。4章でこれらの手法に対する実験結果を示す。

2 2分決定グラフの構造に基づくブーリアン・マッチング

2.1 2分決定グラフ

2分決定グラフ（BDD:Binary Decision Diagram）[6] は論理関数を表す根つきの非巡回有向グラフであり、以下のように定義される。

定義 1 2分決定グラフは次のような7つ組 $B = (N, V_c, V_v, r, e^0, e^1, l)$ で与えられる。ただし、

- $N = \{1, 2, \ldots, n\}$ は変数のインデックスの集合。
- $V_c = \{v_0, v_1\}$ は終端節点の集合。
- V_v は中間節点の集合。
- $r \in V_v \cup V_c$ は根の節点。
- $e^0, e^1 : V_v \rightarrow V_v \cup V_c$ は中間節点の *0*-枝、*1*-枝を表す。
- $l : V_v \rightarrow N$ は各中間節点とインデックスの対応を表す。

各々の節点 v は論理関数 $f_v : \mathcal{B}^n \rightarrow \mathcal{B}$ を表す。

$$f_v = \begin{cases} 0 & v = v_0 \\ 1 & v = v_1 \\ \overline{X_{l(v)}} \cdot f_{e^0(v)} + X_{l(v)} \cdot f_{e^1(v)} & v \in V_v \end{cases}$$

2分決定グラフ Bで表される論理関数 f_Bは、$f_B = f_r$と定義される。 □

通常2分決定グラフと呼ばれているものは順序つき既約2分決定グラフ（reduced and ordered binary decision diagram）で、インデックスの小さい節点から大きな節点に向かう枝しか存在しない（ordered）点と同一の論理関数を表す節点は唯一しか存在しない（reduced）点が特徴となっている。本稿では、以後2分決定グラフを順序つき既約2分決定グラフの意味で用いる。

定義 2 2分決定グラフ Bの節点のうちインデックスが i 以上の節点のみからなる部分グラフをレベル i の下部分グラフとよび、B_iで表す。 □

図1に関数 $f = x_1 \cdot x_2 + x_3$を表す2分決定グラフとそのレベル2の下部分グラフを示す。

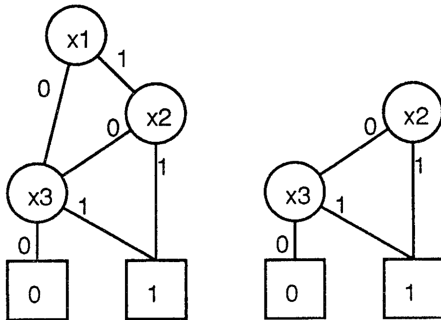

図 1: 2分決定グラフとその下部分グラフの例

2.2 ブーリアン・マッチング

定義 3 ブーリアン・マッチング問題は次のように定義される。与えられた2つの論理関数 $f(x_1, x_2, \ldots, x_n)$ および $g(y_1, y_2, \ldots, y_n)$ に対して、$f = g$となるような変数の割当 $(x_i = y_j$ または $x_i = \overline{y_j})$ を求める。 □

ブーリアン・マッチングは関数 gを表す2分決定グラフを変形して関数 fを表す2分決定グラフに一致させる問題とみなすこともできる。ここで可能な変形は変数の位相反転と順序変更に対応するものである。もしも、$x_i = y_j$ または $x_i = \overline{y_j}$という割当が行なわれた場合、次のような変形を関数 gを表す2分決定グラフに対して施せば良い（ただし、ここでは $i \geq j$とする）。

- インデックス jの全ての節点をインデックス i の位置まで押し下げる。例えば隣接するインデックスの節点同士で交換を繰り返すことでこの処理は行なえる。
- もしも、割当が位相反転を含んでいる場合（$x_i = \overline{y_j}$）、押し下げられたインデックス i の節点の 0-枝と 1-枝を交換する。

ある割り当てに従って関数 gを表す2分決定グラフを変形した結果が関数 fを表す2分決定グラフと等しくなった場合、2つの関数のマッチングが成功したことになる。しかし、このままではマッチングが成功するまで異なる割当を次々に試すことになり、平均的に $O(2^n \cdot n!)$ 通りの割当を試さなければならない。もしも、変数の部分的な割当を行なっただけで、それ以降の変数の割当に関わらずマッチングが成功しないことが分かればその時点で枝刈りを起こすことができ、探索を効率化することができる。以下にそのような枝刈りに必要な定理について述べる。

補題 1 関数 $f = (x_1, x_2, \ldots, x_n)$ を表す2分決定グラフ Bとインデックス $k(1 \leq k \leq n)$ に対して、インデックスが k未満の変数に関する変形を行なってもレベル kの下部分グラフ B_kの構造は変化しない。

そこで、次の定理が導かれる。

定理 1
2つの関数 $f(x_1, x_2, \ldots, x_n)$ と $g(y_1, y_2, \ldots, y_n)$ に対して、インデックスが kより大きい変数に関しては、$x_k = y_k, x_{k+1} = y_{k+1}, \ldots, x_n = y_n$というような割当が決まっているものとする。$f$、$g$を表す2分決定グラフ B^f、B^gのレベル kの下部分グラフ B_k^f、B_k^gが等しくない時、関数 fと gはマッチしない。

証明: インデックスが kより小さな変数の間でどのような割当を行なっても、2分決定グラフのレベル kの下部分グラフは変化しない。つまり、$x_k = y_k, x_{k+1} = y_{k+1}, \ldots, x_n = y_n$という割当が定まった時点で2つの2分決定グラフのレベル kの下部分グラフが異なっていたら、後の割当で等しくなることはない。 □

図2にこの枝刈りを組み込んだブーリアン・マッチングのアルゴリズムを示す。

関数 $pushdown(B, i, j, pol)$ は前述の通り、2分決定グラフ Bのインデックス i の変数をインデックス jの

```
boolean_match(B^f, B^g, k){
    if (k = 0) return FOUND;
    for i = k downto 1 {
        /* x_k = y_i という割当を試す */
        if (|f_{x_k}| = |g_{y_i}|) {
            B^{g1} ← pushdown(B^g, i, k, POSITIVE);
            if (B_k^{g1} ≡ B_k^f) {
                if (boolean_match(B^f, B^{g1}, k − 1)
                = FOUND) {
                    return FOUND;
                }
            }
        }
        /* x_k = \overline{y_i} という割当を試す */
        if (|f_{x_k}| = |g_{\overline{y_i}}|) {
            B^{g2} ← pushdown(B^g, i, k, NEGATIVE);
            if (B_k^{g2} ≡ B_k^f) {
                if (boolean_match(B^f, B^{g2}, k − 1)
                = FOUND) {
                    return FOUND;
                }
            }
        }
    }
    return NOT_FOUND
}
```

図 2: ブーリアン・マッチングのアルゴリズム

位置まで押し下げて、*pol*が NEGATIVE の場合にはさらにその変数に対応する節点の 0-枝と 1-枝を交換する処理を行なう。各割当を試す前にまず関数のコファクターの真理値表密度が等しいかしらべ、等しい場合に *pushdown* を行なった結果の下部分グラフを比べる。これが等しい時のみ、さらに深い再帰呼び出しを行なう。

以下に例を示す。図 3のように $f = x_1 \cdot x_2 + x_3 \cdot x_4$ と $g = y_1 \cdot y_3 + y_2 \cdot y_4$とのマッチングを考える。$|f_{x_1}| = |f_{x_2}| = |f_{x_3}| = |f_{x_4}| = |g_{y_1}| = |g_{y_2}| = |g_{y_3}| = |g_{y_4}|$ であるから、コファクターの真理値表密度では割当の候補を絞ることはできない。まず、最初に $x_4 = y_4$の割当を試してみる。この場合、*pushdown*(B^g,4,4,POSITIVE) なので、gを表す BDD に変化はない。$B_4^f \equiv B_4^g$なので、この割当は成功する。次に、$x_3 = y_3$の割当を選ぶ、これも同様に gを表す BDD を変形させないが、$B_3^f \not\equiv B_3^g$ となるので、$x_4 = y_4$かつ $x_3 = y_3$という割当のもとではマッチングがないことがわかる。次に $x_3 = y_2$という割当が選ばれる（$x_3 = \overline{y_3}$は真理値表密度を用いて除かれる）。この場合、*pushdown*(B^g,2,3,POSITIVE) で、gの 2 番めの変数に対応する節点が 3 番めの位置に押し下げられ、もともと 3 番めの位置にあった節点が 2 番めの位置に上がってくる。すると、その結果は fを表す BDD と等しくなるので、マッチが成功することが分かる。実際にはこの後、$x_2 = y_3$、$x_1 = y_1$という割当の後で、マッチが成功したことが分かる。

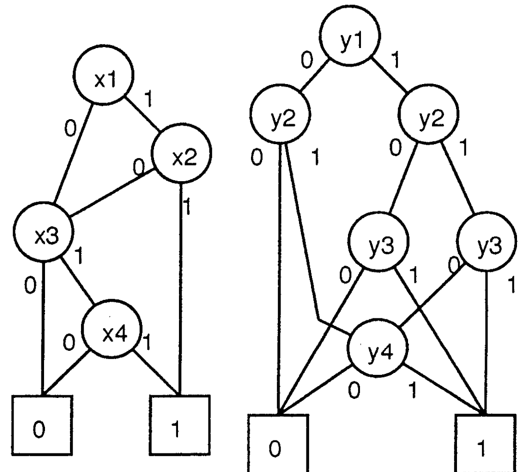

図 3: ブーリアン・マッチングの例

2分決定グラフは論理関数を表す canonical form であるので、以前のマッチング結果を保存しておくことによって、試したことのある関数同士のマッチング結果を表引きだけで取り出すことが可能である。現在の実装では、成功したマッチング結果を適当な大きさのキャッシュに蓄えている。テクノロジ・マッピングの中でブーリアン・マッチングを使用した場合、同一のブーリアン・マッチングを何回も判定する場合が多く、このようなキャッシュを用いた方法は有効であると思われる。ただし、処理時間の効率化については実験結果を参照のこと。

3 トポロジカル・フィルター

通常、テクノロジ・マッピングの対象となる回路は木状の部分回路に分割される。その各々の部分回路はさらに2入力ゲート（例えば2-NAND,2-NOR）のネットワークに分解され、マッチングのための部分回路の構成要素となる [1, 2]。ただし、本稿では一般性を失うことなく、対象となるゲートを2入力 AND ゲートと NOT ゲートのみに限定する。

このマッチングのための部分回路はクラスタと呼ばれ、そのクラスタの実現している論理関数はクラスタ関数と呼ばれる。図 4にクラスタの例を示す。

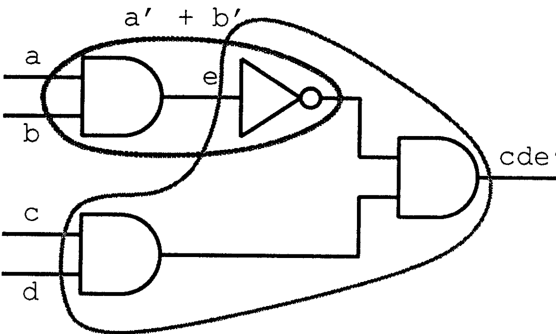

図 4: クラスタの例

ブーリアン・マッチングを利用した多くのテクノロジ・マッパーでは全てのクラスタを陽に列挙して、その各々のクラスタ関数に対してセルの関数とのマッチングを見つけているが、多くのクラスタ関数はセルの関数とはマッチしない。さらに、それらの多くは単に真理値表密度を計算することによってマッチが求まらないことを判定することができることが多い。つまり、全てのクラスタに対してクラスタ関数を表す2分決定グラフを計算することは無駄なことであり、実際にブーリアン・マッチングよりも多くの時間が費やされる場合が多い。そこで、そのようなクラスタをその2分決定グラフを作ることなく判定するために、クラスタの構造を利用したトポロジカル・フィルターを提案する。このトポロジカル・フィルターはクラスタの構造からクラスタ関数の真理値表密度の近似値（上限値、下限値）を計算し、それとセルの関数の真理値表密度と比較することでマッチの可能性があるかを判定するもので、近似値は以下のように計算される。

各ゲートの入力／出力には真理値表密度の上限値、下限値 $[max_appr, min_aprx]$ が定義される。

- クラスタに対する入力

$$max_aprx = \frac{2^n}{2}$$

$$min_aprx = \frac{2^n}{2}$$

- NOT ゲートの出力

  NOT ゲートの入力の近似値を $[max_aprx1, min_aprx1]$ とする。

$$max_aprx = 2^n - min_aprx1$$

$$min_aprx = 2^n - max_aprx1$$

- AND ゲートの出力

  AND ゲートの入力の近似値をそれぞれ $[max_aprx1, min_aprx1]$ 、$[max_aprx2, min_aprx2]$ とする。

  もしもこの2つの入力が共通の外部入力を持たない場合は、

$$max_aprx = \frac{max_aprx1 \times max_aprx2}{2^n}$$

$$min_aprx = \frac{min_aprx1 \times min_aprx2}{2^n}$$

  そうでなければ、

$$max_aprx = MIN(max_aprx1, max_aprx1)$$

$$min_aprx = MAX(0, min_aprx1 + min_aprx2 - 2^n)$$

これらの式から明らかなようにこの値が近似値となるのはクラスタが再収れんを含む場合のみである。この近似値の計算はクラスタのサイズ（ゲート数）に比例した手間で行なうことができ、クラスタ関数を表す2分決定グラフを作成するのに比べて極めて高速である。

表 1: ブーリアン・マッチングの実験結果

circuit	F1	F2	F3	F4
9symml	16597	3895	2261	2150
C1355	1346	768	766	766
C1908	6118	1468	1395	1266
C2670	38944	12269	2913	2485
C3540	127166	35735	11027	9767
C432	14720	3215	1478	1185
C499	1346	768	766	766
C5315	12536	6291	3996	3865
C6288	7455	4610	4584	4556
C7552	117214	25902	11886	9941
apex6	11706	3697	1986	1859
apex7	1560	727	478	458
b9	3170	964	436	385
des	85347	17803	12559	12205
f51m	3440	1594	317	283
rot	19144	6451	2241	1930
z4ml	613	248	134	121
total	468422	126405	59223	53988

4 実験結果

以上のアルゴリズムを実装し、実験を行なった。例題はMCNCの論理合成の多段回路のベンチマークを用い、テクノロジライブラリは富士通のCG31（CMOSの敷き詰め型ゲートアレイ）を用いた。使用計算機はSUN4/670（メモリ128Mbyte）で、各々の回路はテクノロジ・マッピングの前に論理合成システムSISのruggedスクリプト[7]で簡単化された後に、木状の部分回路に分解されている。表1がその時の結果を示している。各コラムはそれぞれ、

- 回路名（*circuit*）
- 総クラスタ数（F1）
- トポロジカル・フィルター通過後のクラスタ数（F2）
- 真理値表密度によるフィルター通過後のクラスタ数（F3）
- マッチの見つかったクラスタ数（F4）

を表している。

全クラスタの70%以上がトポロジカル・フィルターによって取り除かれている。さらに、（厳密な）真理値表密度を用いたフィルターによって、マッチを持たないクラスタのほとんどは取り除かれている。

表2はこの実験のCPU時間を示している。各コラムは、

- 回路名（*circuit*）
- マッチング結果のキャッシュおよびトポロジカル・フィルターを用いた時のCPU時間
- キャッシュを用いずにトポロジカル・フィルターのみを用いた時のCPU時間
- キャッシュは用いてトポロジカル・フィルターを用いなかった時のCPU時間
- キャッシュ上にマッチング結果があった回数
- 全ての成功したマッチングの回数

を表している。時間の単位は秒である。

マッチング結果のキャッシュを行なうことにより全体のほぼ6%に相当するマッチングに対してしか実際のブーリアン・マッチングは実行されていないことがわかる。しかし、CPU時間を見る限りではこのキャッシュの影響はそれほど顕著ではない。これはブーリアン・マッチングそのものが比較的高速なことと、クラスタ関数を表す2分決定グラフの計算に多くの時間が費やされていることによるものと思われる。

トポロジカル・フィルターを用いないと、CPU時間が著しく増大することがわかる。この場合、全てのクラスタに対して2分決定グラフが作成されてから真理値表密度が計算され、その結果マッチがないと判定されたクラスタが捨てられている。トポロジカル・フィルターを用いた場合との相違点は全体の70%程のクラスタに対してその2分決定グラフを作成するか否かということである。その結果が約2倍近い処理時間の差となっている。

5 おわりに

2分決定グラフの構造に着目したブーリアン・マッチングのアルゴリズムについて述べた。このアルゴリズムは非常に簡潔であり、2分決定グラフの下部分グラフの構造比較を用いた枝刈りを既存の枝刈り手法と組み合わせることにより極めて高速に処理を行なうことができる。多くの場合、クラスタ関数の2分決定グラフの作成にブーリアン・マッチングと同等かそれ以上の時間が費やされている。トポロジカル・フィルター

が効果的にマッチの可能性のないクラスタを排除して全体の処理速度の向上をはかっている。

このアルゴリズムでは、ドントケア条件を考慮することはできないが、そのような拡張は可能である。この場合、下部分グラフの等価性ではなく、包含性判定を行なえば良い。2分決定グラフでドントケアを含んだ論理関数を表すための手段はいくつか考えられるが、0,1 の終端節点の他にドントケアを表す第三の節点を設けることによって、この包含性判定も容易に行なうことができる。

参考文献

[1] K. Keutzer, "DAGON:Technology Binding and Local Optimization by DAG Matching" In *Proceedings of 24th DAC*, pp. 341–347, June 1987.

[2] E. Detjens and G. Gannot and R.L. Rudell and A. Sangiovanni-Vincentelli and A.R. Wang, "Technology Mapping in MIS" In *Proceedings of ICCAD*, pp. 116–119, November 1987.

[3] F. Mailhot and G. De Micheli, "Algorithms for Technology Mapping Based on Binary Decision Diagrams and on Boolean Operations" Technical Report of Stanford University, No. CSL-TR-91-486, August 1991.

[4] J.B. Burch and D.E.Long, "Efficient boolean function matching", In *Proceedings of ICCAD*, pp. 408–411, November 1992.

[5] H. Savoj ,M.J. Silva, R.K. Brayton and Alberto Sangiovanni-Vincentelli, "Boolean Matching in Logic Synthesis", In *Proceedings of Euro-DAC*, pp. 168–174, September 1992.

[6] R.E. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computer*, C-35(12), 1986.

[7] H. Savoj, H. Touati and R.K. Brayton, "Improved Scripts in MIS-II for Logic Minimization of Combinational Circuits", In *International Workshop on Logic Synthesis*, May 1991.

表 2: Computational time

circuit	CPU(sec)			matchings	
	cache	no cache	no tpg filter	cache hit	total
9symml	5.82	6.60	9.78	1833	2150
C1355	0.48	0.57	0.35	749	766
C1908	1.60	1.90	2.80	1134	1266
C2670	13.45	13.90	24.20	2296	2485
C3540	38.80	42.68	66.83	9127	9767
C432	3.63	3.87	7.13	1069	1185
C499	0.42	0.50	0.50	749	766
C5315	4.32	4.93	5.05	3685	3865
C6288	2.77	3.55	2.60	4506	4556
C7552	32.80	36.03	64.28	9478	9941
apex6	3.18	3.58	5.65	1701	1859
apex7	0.55	0.58	0.67	375	458
b9	0.90	0.97	1.68	287	385
des	18.63	22.37	35.90	11632	12205
f51m	1.35	1.40	1.87	207	283
rot	5.93	6.33	9.40	1713	1930
z4ml	0.22	0.20	0.27	73	121
total	134.85	149.96	238.96	50617	53988