システムＬＳＩ設計技術　96－9
(2000. 5. 12)

直交でない関数分解の効率的な列挙手法

松永　裕介

（株）富士通研究所

〒211-8588 川崎市中原区上小田中 4-1-1
044-754-2663
yusuke@flab.fujitsu.co.jp

あらまし

本稿では，与えられた論理関数に対して直交でない関数分解を求める効率の良いアルゴリズムについて述べる．このアルゴリズムは著者が以前開発した二分決定グラフを用いて直交分解を行うアルゴリズムを利用したものである．通常，直交でない関数分解は多数存在するので無制限に関数分解の列挙を行うことは難しいので，重複した変数の個数が規定値以下の関数分解のみを列挙する様になっている．7入力程度の論理関数に対して適用したところ，ナイーブなアルゴリズムに比べて5〜6倍の高速化が達成されている．

キーワード　論理合成，関数分解，二分決定グラフ

On Enumerating Non-Disjunctive Decompositions of Logic Functions

Yusuke Matsunaga

Fujitsu Laboratories LTD.

4-1-1 Kamikodanaka, Nakahara-Ku, Kasawaki 211-8588
044-754-2663
yusuke@flab.fujitsu.co.jp

Abstract

This paper describes an efficient algorithm enumerating all the non-disjunctive decomposition of a given function. The algorithm utilizes the disjunctive decomposition algorithm using binary decision diagrams that the authors have previously developed. Since, in general, there exist too many non-disjunctive decompositions for ordinary logic functions, the algorithm restricts to enumerate only decompositions whose duplicated variables are less than the given limit. Comparing to the existing naive algorithm, about 5 or 6 times acceleration has been observed for a case of applying to 7-inputs functions.

key words　logic synthesis, functional decomposition, binary decision diagrams

1 はじめに

論理関数 $F(X)$ を次のような2つの関数 G, H を用いて表すことができるとき，これを関数 F の **関数分解 (functional decomposition)** と呼ぶ．

$$F(X) = G(X_1, H(X_2)) \tag{1}$$

関数分解にはいくつかの分類がある．まず，式 (1) の関数 H が二値の論理関数の場合を単純な分解 (simple decomposition) 呼ぶ．H が多値の場合，または，複数の二値論理関数のベクタの場合を複雑な分解 (complex decomposition) と呼ぶ．次に，変数集合 X_1 と X_2 が互いに素な場合を直交な分解 (disjunctive decomposition) と呼び，そうでない場合を直交でない分解 (non-disjunctive decomposition) と呼ぶ．直交でない分解は複雑な分解の制限された形と見なすことも出来る[1]．

関数分解は古くは PLA の直列分解に用いられるなど論理合成において様々な応用が考えられており，最近では look-up table(LUT) 型の FPGA の合成に用いられている [2, 3]．これは，LUT 型の FPGA においては，その関数がたとえ論理式としては複雑であろうと簡単であろうと，その入力数が一定の値以下であれば同じブロックで実現できることに起因する．そのため，個々の論理関数の入力数を一定の値以下に抑えるために関数分解が用いられる．また，このようなアプローチとは別に，複数の LUT から構成される FPGA のブロックに論理関数をマッチさせるブーリアンマッチング処理に関数分解を用いたものも提案されている [4, 5]．

関数分解を求めるアルゴリズムに関しては比較的古くから研究が行われている [1] が，近年，二分決定グラフを用いて単純な直交分解を効率よく求めるアルゴリズムが提案された [8, 9]．しかし，直交でない分解に関してはあまり効率の良い手法は知られていない．文献 [5] 中では，対象となる関数の入力数が 6〜7 程度と比較的少ないこともあって，しらみつぶし的に探索する手法を用いている．

本稿では，直交分解を求めるアルゴリズムを応用して，直交でない分解を効率よく求めるアルゴリズムの提案を行う．このアルゴリズムは2つの変数集合の重なりが規定値以下の分解をすべて列挙するもので，重なりの小さな分解に対しては効果的である．一般に直交でない分解は直交な分解に比べて極めて多数存在し，また，重なりの多い関数分解は論理合成への応用から考えてあまり実用的な意味がないと思われるので，このように変数集合の重なりがある一定の値以下の分解だけを列挙することは意味のあることだと思われる．

[1] X_2すべてに依存する一つの関数と各々一つの入力のみに依存する $|X_1 \cap X_2|$ 個の関数から H が構成される．

本稿は以下のように構成される．2章では，基本的な用語の定義，および関数分解の説明を行った後で，直交でない分解を求めるために必要な理論的な考察を行う．つづく3章ではそれらを用いた関数分解の列挙アルゴリズムについて述べ，4章で実験結果を示し，考察を行う．

2 直交でない分解を求めるための理論的準備

2.1 用語の定義

ここでは本稿で用いられる用語の定義を行う．

論理関数 $F(X)$ に対して，ある入力変数 $x \in X$ を 0 または 1 に固定した関数 $F|_{x=0}, F|_{x=1}$ を F の x による**コファクター (cofactor)** と呼ぶ．$F|_{x=0}, F|_{x=1}$ は $F_{\overline{x}}, F_x$ と表されることもある．また，コファクターをとる変数 x が自明な場合には F_0, F_1 とも表される．また，同様に，論理関数 F に対して複数の入力変数の値を固定したものもコファクターと呼ぶ．論理関数 F に対して x による2つのコファクターが異なるとき ($F_{\overline{x}} \neq F_x$)，$F$ は変数 x に**依存**しているという．論理関数 $F(X)$ の**サポート (support)** とは F が依存している変数の集合である．論理関数 F が式 (1) の形の分解を持つとき，関数 H のサポート X_2 を**束縛集合 (bound set)** と呼ぶ．また，関数 G のサポートから H を除いた物を**自由集合 (free set)** と呼ぶ．直交でない分解の場合，$X_1 \cap X_2 \neq \phi$ であるが，ここではその重なり $X_1 \cap X_2$ を**共通集合 (common set)** と呼ぶことにする．また，共通集合の要素数 $|X_1 \cap X_2|$ を分解の**多重度**と呼ぶ．

2.2 直交分解と分解グラフ

直交な分解は以下のような性質を持つ．

1. もしも F が $G(X_1, H(X_2)$ と $G'(X_1', H'(X_2'))$ という2つの直交分解を持ち，かつ H および H' はこれ以上分解されない最小の関数だとすると，$X_1 \supseteq X_2'$ であるかまたは $X_1' \supseteq X_2$ である．つまり，複数の直交分解が存在した場合，それらをいかなる順序で適用しても最終的にはこれ以上分解できないユニークな分解が存在する．
2. 論理関数 F が次のような分解を持つものとする．

$$F = G(X_1, H(X_2))$$

ここで X_1 に属する変数 $x (x \in X_1)$ に対する F のコファクターを考えると，

$$F_0 = G_0(X_1 - \{x\}, H(X_2)) \tag{2}$$
$$F_1 = G_1(X_1 - \{x\}, H(X_2)) \tag{3}$$

の様になる．ここで，F_0は F の $x=0$ に対するコファクターを表す $(F_0 = F|_{x=0})$．同様に，G_0は G の $x=0$ に対するコファクターを表すものとする．逆に，もしも F のコファクターが上記の式を満たすのならば，$G=\overline{x}\cdot G_0 + x\cdot G_1$とすることで，次のような関数分解を得ることができる．

$$F = G(X_1, H(X_2)) \tag{4}$$

同様に，x が X_2に属するとすると，

$$\begin{aligned} F_0 &= G(X_1, H_0(X_2 - \{x\})) & (5)\\ F_1 &= G(X_1, H_1(X_2 - \{x\})) & (6)\end{aligned}$$

となり，こちらも $H=\overline{x}\cdot H_0 + x\cdot H_1$とすることで，式 (4) を得ることができる．

Bertacco と Damiani は論理関数を二分決定グラフ (Binary Decison Diagram:BDD) で表し [6]，その二分決定グラフを再帰的にたどることによって直交分解を求めるアルゴリズムを提案した [8]．著者も同様のアルゴリズムを提案している [9]．

ここでは文献 [9] のアルゴリズムで用いられる分解グラフを簡単に説明しておく．分解グラフ $DG(V,E)$ は節点の集合 V と枝の集合 E から成る．節点には以下の種類がある．

定数 0: 定数 0 を表す．入枝は持たない．このタイプは元の関数が定数関数である場合以外には用いられることはない．

リテラル: リテラル関数を表す．変数番号を持つ．入枝は持たない．

OR: OR 関数を表す．任意の数の入枝を持つ．

XOR: XOR 関数を表す．任意の数の入枝を持つ．

Other: それ以外の (単純でない) 関数を表す．任意の数の入枝を持つ．

各々の節点は分解の構造と同時に論理関数も表している．そこで，各々の節点はその関数を表す BDD の節点へのポインタを持つ．

各々の枝は極性の属性を持つ．文献 [7] の BDD の実装と同様に，否定の属性を持った枝の表す論理関数は，その枝の指している節点の表す論理関数の否定である．

分解グラフをカノニカルに保つために，以下のような規則を設ける．

- **XOR** 節点の入力の枝には否定の属性を付けない．
- **Other** 節点の入力の枝には否定の属性を付けない．
- **Other** 節点の出力の枝の極性は BDD の枝の極性と同一にする．

ここで，効率の上で非常に重要なことは，これらのアルゴリズムは個々の分解を列挙するのではなく，分解グラフと呼ばれる一つの木構造を構築するだけだということである．直交分解の数は最悪の場合入力数 n に対して $O(2^n)$ となる[2] が，分解グラフの節点数は高々 $O(n)$ にしかならない．

2.3 直交でない分解の性質

直交な分解とは異なり，直交でない分解にはその列挙アルゴリズムを考える上で望ましくない性質がある．

1. 同時に適用することの出来ない分解が存在する．つまり，F に対して $G(X_1, H(X_2)$ と $G'(X'_1, H'(X'_2))$ という 2 つの直交でない分解があって，$X_1 \cap X'_2 \neq \phi$ かつ $X_2 \cap X'_2 \neq \phi$が成り立つ場合がある．このため，分解の構造がユニークには定まらない．
2. 任意の束縛集合に対して関数分解が必ず存在する．たとえば自由集合の中に束縛集合の変数をすべて含めてしまえば，たとえ H がどのような関数であろうとも G のみで F を表すことが可能である．n 入力関数に対して束縛集合の数は 2^n個あるので，常に $O(2^n)$ 個の分解が存在することになる．

1 番目の性質により分解グラフを用いる直交分解のアルゴリズムをそのまま適用できないことがわかる．また，2 番目の性質からすべての分解を列挙することが容易ではないことがわかる．

そこで，本稿では多重度 (束縛集合と自由集合の重なりの要素数) に制限を設けて関数分解を列挙する問題を考えることにする．これは多重度の少ないほうが列挙しやすいことと，論理合成の応用においては多重度の少ない関数分解の方がより望ましいであろうという仮定に基づくものである．

x_i のみを共通に持つ関数分解 $F = G(X_1, H(X_2))$, $X_1 \cap X_2 = \{x_i\}$ を考える．ここで，F に対して x_iの コファクターを求めると，

$$\begin{aligned} F_0 &= G_0(X_1 - \{x_i\}, H_0(X_2 - \{x_i\})) & (7)\\ F_1 &= G_1(X_1 - \{x_i\}, H_1(X_2 - \{x_i\})) & (8)\end{aligned}$$

の様になる．$X_1 \cap X_2 = \{x_i\}$ なので，F_0および F_1 は直交分解を持つことになる．逆に F のコファクター F_0および F_1が式 (7,8) のような分解を持つとすると，

$$\begin{aligned} G &= \overline{x_i}\cdot G_0 + x_i \cdot G_1 & (9)\\ H &= \overline{x_i}\cdot H_0 + x_i \cdot H_1 & (10)\end{aligned}$$

[2] たとえば n 入力 AND 関数がそうである．

のように G と H を定めることで $F=G(X_1,H(X_2))$ の形の関数分解を得ることができる．つまり，論理関数 F が x_i のみを共通集合とした関数分解を持つための必用十分条件は式 (7,8) のような分解が存在することである．共通集合が複数の変数からなるときも同様の議論が成り立つ．つまり，各々のコファクターに対して同一の変数分割に基づく直交関数分解が存在することが必用十分条件となる．

3 直交でない関数分解を求めるアルゴリズム

前述の性質をもとに直交でない関数分解を求めるアルゴリズムを考えると以下のようになる．

1. foreach 規定値 M 以下の共通集合 C {
2. F の C によるコファクター $F_i(i=0,\dots,2^{|C|}-1)$ を求める．
3. 各 F_i の直交分解の集合 D_i を求める．
4. 各 D_i の共通な要素を選ぶ．
5. }

重要な処理は 3. の直交分解の集合 D_i を求める処理と 4. の直交分解の集合の共通な要素を選びだす処理である．以下，それぞれについて説明を行う．

3.1 直交分解の集合の列挙

与えられた論理関数 F_i に対して文献 [9] のアルゴリズムを用いて分解グラフを求めることができる．前述のように分解グラフは個々の分解を列挙するよりも効率のよいデータ構造であるが，4. の処理で共通な分解の要素を求める際に分解グラフ上でアルゴリズムを考えると複雑になると思われたので，一旦，個々の分解を列挙してからその共通部分を求めるアプローチをとっている．

基本的には分解グラフ上のある節点を根とした部分グラフ (部分木) がある分解の関数 H に相当する (図 1) ので，分解グラフ上の各節点を訪れて，対応する分解を列挙すれば良い．ただし，節点のタイプが OR と XOR の場合には注意が必要となる．たとえば，変数集合 X を入力とする OR 節点を考えると，X の任意の部分集合 X_S を束縛変数とする分解が存在する (図 2)．つまり，入力数 n の OR 節点は非明示的に 2^n 通りの異なる分解を表していると見なすことができる．以上を考慮したアルゴリズムが図 3 である．直交分解の場合，関数に対して変数の分割を与えれば唯一の分解を得ることができる[3] ので，ここでは変数の分割 (X_1,X_2) を

[3] 厳密には束縛集合に依存した関数 H の出力の極性の選び方で 2 通り存在する．

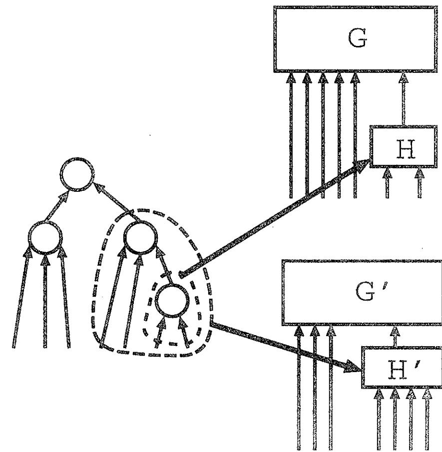

図 1: 分解グラフと直交分解

用いて関数分解を表している．分解グラフの節点から入力側にたどることで自分が依存している変数の集合を得ることができる．それを X_2 として分解を登録している．X_1 は全変数から X_2 を引くことで得ることができる．

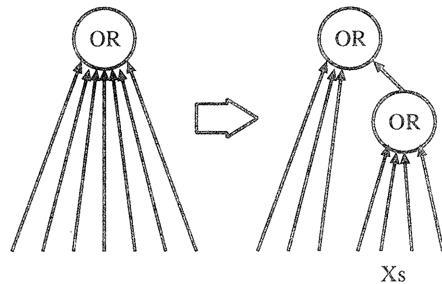

図 2: OR ノードの分解

3.2 共通な分解の選択

ここでは，各コファクターごとに求められた分解の集合を入力として，共通な要素の組を選びだす処理を行う．共通な要素とは基本的には式 (7,8) の様に，変数分割が同じ分解のことであるが，厳密には少し注意が必要となる．例えば，$F=ab+acd+bc$ という関数に対して a のコファクターを計算すると，

$$F_0 = bc$$

```
EnumDec(node, dec_list)
{
1:     node を根とする部分木の表す分解を
       dec_list に登録
2:     if (node のタイプが OR か XOR) {
3:         foreach node のファンインの部分集合 S {
4:             S の表す分解を dec_list に登録
5:         }
6:     }
7:     foreach node のファンイン fi_node {
8:         EnumDec(fi_node, dec_list)
9:     }
}
```

図 3: 直交分解を列挙するアルゴリズム

$$F_1 \quad = \quad b + cd$$

となる．F_0に対しては $(\{b\},\{c\})$ という分解 (変数分割) があり，F_1に対しては $(\{b\},\{c,d\})$ という分解があるが，これらは等しい分割ではない．ところが，実際には F_0には d が含まれないので，$(\{b\},\{c\})$ という分割は $(\{b\},\{c,d\})$ と見なすことができ，すると F_1の分割と等しくなるので，共通の分割 $(\{b\},\{c,d\})$ を持つことがわかる．結果として $F = G(a,b,H(a,c,d))$ という分解を得ることができる[4]．このように，コファクターごとにサポートが異なるため，変数の分割が同一にならない場合がある．そこで，以下のように変数分割に対する両立の概念を定義する．

2つの変数分割 (X_1, X_2)，(X_1', X_2') に対して，$X_1 \cap X_2' = \phi$ かつ $X_1' \cap X_2 = \phi$が成り立つとき，(X_1, X_2) と (X_1', X_2') は両立であると言う．また，3つ以上の変数分割の集合に対しても，その中のすべての要素対が両立であるならばその集合を両立集合と言う．

各コファクターに対する分割集合から要素を一つづつ選びだして組み合わせた集合が両立集合であればそれを記録することで，各コファクターに共通の分解を求めることができる．現在の実装では，単純にすべての組み合わせを列挙して両立かどうかの判断を行っているが，改良の余地はあると思われる．

4 実験結果と考察

上記のアルゴリズムを C++で実装し，評価のための次のような実験を行った．

[4] この例ではもう一つ $F = G'(a,c,d,H'(a,b))$ という分解も存在する．

1. MCNC の論理合成のベンチマーク回路を 2 入力ノードからなるネットワークに分解する．
2. 分解されたネットワークから 1 出力の連結した部分グラフ (クラスタと呼ぶ) を全て列挙する (ただしクラスタの入力数は 7 入力以下)．
3. クラスタの実現している論理関数を求める．これをクラスタ関数と呼ぶ．
4. すべてのクラスタ関数を NPN 同値 (入出力変数の反転，および入力変数順序の入れ替えを行うと等しくなる論理関数間の等価関係) の関係で分類して，各々の NPN 同値類から一つの関数を代表関数として選ぶ．この処理は文献 [10] のアルゴリズムを用いた．
5. 選ばれた代表関数に対して多重度が 0 ～ 2 の関数分解をすべて求める．
6. 比較のため，以下の 2 つの手法を用いる．

手法 1：すべての変数の分割を列挙し，定められた変数の分割にもとづいて BDD 上で分解表 (decomppsition chart) を作り，分解が存在するか判定する．

手法 2：3 章で述べたアルゴリズムを用いる．共通変数を列挙し，その共通変数にもとづいて展開されたコファクターに対して文献 [9] の直交分解アルゴリズムを適用する．

ベンチマーク回路に対してこのような前処理を施しているのは，文献 [4, 5] と同様に，LUT 型 FPGA のマッピングを応用として仮定しているからである．今回の実験では約 4,000 個の 6 入力関数と約 25,000 個の 7 入力関数をサンプルとして用いている．実用上意味のある論理関数のサンプルとしては十分な数と思われる[5]．

表 1 に実験結果を示す．最初のコラムは多重度の規定値を表す．2 番目のコラムは総勢約 30,000 個の論理関数に対して求められた関数分解の数を示している．ただし，ここでは $(\{a,b\},\{c,d\})$ という分解が存在している場合にそれを含むような $(\{a,b,c\},\{c,d\})$ という分解は冗長と判断して数には含めていない．3 番目と 4 番目のコラムがそれぞれ手法 1 および手法 2 に要した計算時間 (単位は秒) である．計算機は Pentium-III 500MHz の PC (FreeBSD-3.4-STABLE) を用いた．

ここで多重度が 0 の場合の関数分解とは直交分解のことであり，この実験に関しては 1 つの関数当り約 11 個程度の関数分解を持つことがわかる．これが多重度 1 で約 16 個，多重度 2 では約 21 個と増えている．処理時間に関しては文献 [9] の直行分解アルゴリズムを用いた手法 2 が明らかに高速であり，この傾向は対象の関

[5] NPN 同値の関係を用いないとサンプル数は数倍～数十倍多くなるが，今回のような関数分解の場合，同一の NPN 同値類に属する関数に対しては全く同様に求めることができる (BDD の変数順の違いにより BDD のサイズは変わる) ので NPN 同値類を考慮しないことでサンプル数を増やしてもあまり意味はない．

表 1: 関数分解の結果

多重度	個数	手法1	手法2
0	333316	690	109
1	488806	3068	402
2	642967	7038	1332

数の入力数が増えるにしたがって顕著になると考えられる．ただし，多重度が 0 の時の速度比が 6.33 : 1 であるのに対して多重度が 2 の時の速度比が 5.28 : 1 と落ちているのはコファクターをとる変数が増えて直交分解を求める対象の関数の入力数が減少しているためかと思われる．この点に関しては対象となる関数の入力数を変えて実験を行う必要があるが，2 入力ノードに分解されたネットワーク中に含まれるクラスタの数はその入力数が増えると指数爆発的に増えるので今回の実験では 7 入力までしか列挙することができなかった．関数分解のためのデータという意味では全てのクラスタを列挙せずに，適当な数のクラスタをサンプリングすることも考えられる．

今回，提案した手法はナイーブなアルゴリズムに比べれば効率的ではあるが，多重度が増えるにしたがって計算時間が 3～4 倍増加しており，計算複雑度の点から見るとナイーブなアルゴリズムと同等である．応用を LUT 型 FPGA のマッピング等に絞った場合,LUT の入力数が 4 なので，多重度としても高々2～3 を考えれば良く，十分実用的と言えるが，他の応用を考えたときには検討しなければならない問題である．

5 おわりに

本稿では直交でない関数分解を列挙するアルゴリズムについて述べた．このアルゴリズムは二分決定グラフを用いて直交分解を高速に求めるアルゴリズムを利用したもので，7 入力関数の多重度 2 以下の分解を求める実験では，ナイーブなアルゴリズムに対して約 5～6 倍高速であった．ただし，ナイーブなアルゴリズムと同様に，多重度に対して計算時間は指数的に増加しており，検討の余地があると思われる．ここでは多重度が規定値以下の全ての関数分解を求める問題を扱ったが，応用としてはある条件を満たした関数分解をただ一つ求めれば良い場合も考えられ，その場合には一旦，全てのコファクターに対して全ての直交分解を列挙してしまう本手法は効率的ではない．今後,LUT 型 FPGA のマッピングなどの関数分解の応用を考えると共に，この応用に適した関数分解のアルゴリズムを検討する必要があると思われる．

参考文献

[1] R.L. Ashenhurst, "The decomposition of switching functions", *In Proceedings of an international symposium on the theory of switching*, pp. 74-116, April 1957.

[2] T. Sasao, "FPGA design by generalized functional decomposition", *in Logic Synthesis and Optimization*, Kluwer Academic Publishers, pp. 233-258, 1993.

[3] S. Yamashita, H. Sawada, and A. Nagoya, "An Efficient Framework of Using Various Decomposition Methods to Synthesize LUT Networks and Its Evaluation", *In Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC 2000)*, pp. 253 – 258, Jan. 2000.

[4] J. Cong, and Y.-Y. Hwang, "Parially-Dependent Functional Decomposition with Applications in FPGA Synthesis and Mapping", *In Proceedings of the ACM 5th International Symposium on FPGA*, pp. 35 – 42, Feb. 1997.

[5] J. Cong, and Y.-Y. Hwang, "Boolean Matching for Complex PLBs in LUT-based FPGAs with Application to Architecture Evaluation", *In Proceedings of the ACM 6thInternational Symposium on FPGA*, pp. 27 – 34, Feb. 1998.

[6] R.E. Bryant, "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computer*, C-35(8), pp. 677–691, Aug. 1986.

[7] K.S. Brace, R.L. Rudell, and R.E. Bryant, "Efficient implementation of a BDD package", *In Proceedings of the 27th Design Automation Conference*, Jun. 1990, pp. 40-45.

[8] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions", *In Proceedings of the International Conference on Computer-Aided Design (ICCAD'97)*, pp. 78-82, November 1997.

[9] Y. Matsunaga, "An Exact and Efficient Algorithms for Disjunctive Decomposition", In Proccedings of the Workshop on Synthesis And System Integration of Mixed Technologies (SASIMI'98), pp. 44 - 50, Oct. 1998.

[10] Y. Matsunaga, "A New Algorithm for Boolean Matching Utilizing Structural Information",

Synthesis and Simulation Meeting and International Interchange(SASIMI'93), pp. 366 – 373, Oct. 1993.

盛光印刷所