Vol. 34 No. 5　　情　報　処　理　　May 1993

解　説

BDD（二分決定グラフ）

1. B D D　と　は†

石浦　菜岐佐††

1. はじめに

ブール代数や論理回路の演習に，「与えられたブール式—たとえば $F=(x_1\overline{x_2}+x_3)\ (\overline{x_1}+x_3)$—を計算し簡単にせよ.」という問題をよく見かける．初歩的な問題ではあるが，大規模なものを効率よく解く方法が見つかれば，現在の情報工学の分野における多くの難問が解決できることになる．論理合成，設計検証，故障検査など，論理回路の計算機援用設計の分野における諸問題が，ブール式や論理関数の効率的な処理法に依存していることはいうまでもないが，整数計画法，被覆問題，n クイーン問題などの最適化問題も，論理関数の処理に帰着して解くことができる．**二分決定グラフ**（*Binary Decision Diagram*，以下 *BDD* と略する）は，論理関数の表現法の一つであるが，この望みを「いくらか」かなえてくれるものとして，LSI の CAD (computer-aided design：計算機援用設計）の分野を中心に最近注目を集めているものである．

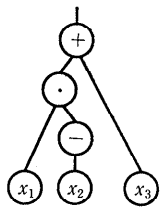
論理関数を計算機上で表現する方法としては，これまで，1)真理値表表現，2)解析木表現，3)積和形表現，などが用いられていた．真理値表は**図-1**(a）のように，入力変数ベクトルの辞書順に関数の値を列挙したものである．解析木表現は，**図-1**(b）のように論理関数を表すブール式の構文木を記憶するものである．積和形表現は，$x_1\overline{x_2}+x_3$ のように関数を変数およびその否定の論理積の論理和の形で表すものである．優れた論理関数表現の条件としては，

a）表現が小さいこと，

b）関数間の論理演算（f と g の表現から $f+g$ の表現を求める），等価，恒偽，包含の判定の計算量が小さいこと，

があげられる．n 変数論理関数の総数は 2^{2^n} であるから☆，これらを識別するためには 2^n に比例する記憶量はどうしても必要となる．したがって，a）は，「実際の応用を扱う際に出現するできるだけ多くの論理関数について」ということになる．論理関数の等価，恒偽，包含の判定は，ブール式の充足可能性判定問題が NP 完全であることより，一般には指数時間が必要であると考えられるので，b）についても a）と同様のことがいえる．

$x_1x_2x_3$	f
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	1
1 1 0	0
1 1 1	1



(a) Truth-table　(b) Parse tree

図-1 論理関数 $x_1\overline{x_2}+x_3$ の真理値表と解析木による表現

この条件に照らしてみると，真理値表表現は，どのような n 変数論理関数に対しても 2^n の記憶量が必要になる—たとえ恒偽関数のような簡単な関数に対しても，100 変数を扱っていれば 2^{100} の記憶量を必要とする—という問題点がある．演算についても同様である．解析木表現は，a）の条件を満たし，論理演算も容易であるが，一つの論理関数に対する表現が一意でないため，等価，恒偽，包含の判定の計算量が膨大となる．積和形表現は a），b）ともある程度満足するが，パリティ関数 $(x_1\oplus x_2\oplus\cdots\oplus x_n)$ などの実用上重要な関数の表現の大きさが $O(2^n)$ となることや，論理否定演算に時間がかかること，等価，恒偽，包含の判

† An Introduction to Binary Decision Diagrams by Nagisa ISHIURA (Department of Information Systems Engineering, Faculty of Engineering, Osaka University).

†† 大阪大学工学部情報システム工学科

☆ 2^n 個の入力割当てのおのおのに対して 0,1 の 2 通りの出力値が選べることより，n 変数論理関数の総数は 2^{2^n} となる．

定が必ずしも容易でない，などの問題点がある．

BDD はグラフによる論理関数の表現である．はじめに Akers が表現そのものを考案した[1]のは1978年であるが，1985年に Bryant が効率的な演算法を考案[3]して以来，一躍注目を集めるようになった．

BDD は，ある条件の下では，論理関数に対する表現が一意に定まり標準形となるという特長がある．上記 a)，b) の点で積和形よりも優れていると考えられ，現在，LSI の CAD のさまざまな応用に用いられるようになっている．これらの中には BDD を利用することによってはじめて実用時間で計算可能になったものも多く，今後 CAD 技術の実用化にも大きな影響を与えていくものと考えられる．

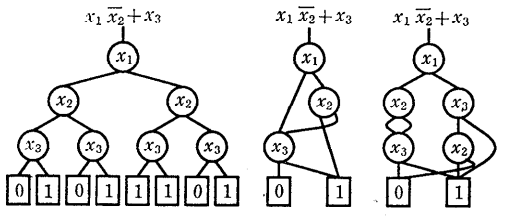
本稿では，BDD とはどのような表現で，どのような特長をもつかを簡単に解説した後，これがどのように利用できるのかを紹介する．

2. 二分決定グラフ (BDD)

2.1 BDD とは

図-2(a),(b),(c) はいずれも論理関数 $x_1\overline{x_2}+x_3$ を表現する BDD である．このグラフがどのようにして $x_1\overline{x_2}+x_3$ を表現しているかであるが，直観的には，雑誌などでよく見かける「性格診断」などと同じである．一番上の節点（根）より始め，丸い節点に記された変数の値が0であれば左の枝を，1であれば右の枝をたどるという操作を繰り返す．最終的にたどり着く四角い節点に記された論理値が，与えられた変数の値に対する関数の値になるというものである．

BDD 中の非終端節点（丸い節点）は変数でラベル付けされており，**変数節点**と呼ばれる．終端節点（四角い節点）は論理値でラベル付けされており，**定数節点**と呼ばれる．変数の値が0,1のときたどる枝はそれぞれ**0枝**，**1枝**と呼ばれる．変数節点 v の0枝で指される節点を low(v)，1枝で指される節点を high(v) と表すことにする．また，変数節点 v にラベル付けされた変数を var(v)，定数節点 v にラベル付けられた論理値を value(v) で表すことにする．



(a) BDD equivalent to truth table　(b) Reduced ordered BDD　(c) Unordered BDD

図-2 二分決定グラフ (BDD)

BDD の各節点は一つの論理関数を表す．節点 v の表す論理関数 f_v は，次のように定義される．

f_v = value(v)（恒偽関数，恒真関数）
　　… v が定数節点のとき

$f_v = \overline{\text{var}(v)} \cdot f_{\text{low}(v)} + \text{var}(v) \cdot f_{\text{high}(v)}$
　　… v が変数節点のとき

すなわち，BDD は論理関数の Shannon 展開をグラフで表現したものとみることができる．逆にいえば，v を変数節点，$f|_{x=0}$，$f|_{x=1}$ をそれぞれ f の変数 x に 0,1 を代入して得られる関数（f の $x=0$，$x=1$ に関するコファクタ）とすると，

$f_{\text{low}(v)} = f_v|_{\text{var}(v)=0}$

$f_{\text{high}(v)} = f_v|_{\text{var}(v)=1}$

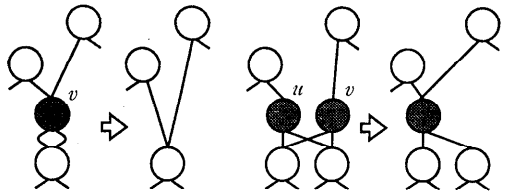
が成立する．BDD の根の表す論理関数を，その BDD の表す論理関数という．

2.2 BDD の標準形

図-2 のように同じ論理関数を表す BDD は複数存在するが，この中で重要な意味をもつのは，**既約な順序付き BDD** (ROBDD: reduced ordered BDD) である．

いま，図-2 の BDD に対して $x_1<x_2<x_3$ という変数の全順序を考える．図-2 の (a)，(b) の BDD では，根から定数節点に向かって節点をたどったときに，どのようなパスを通っても，変数の出現する順序はこの全順序に従っている．このように，ある変数の全順序が存在し，根から定数節点に至るすべてのパスについて，変数の順序がその全順序関係に矛盾しないとき，その BDD は**順序付き BDD**(orderedBDD) と呼ばれる．図-2(a)，(b) は順序付き BDD であるが，(c) はそうではない（x_1, x_2, x_3 という出現順と x_1, x_3, x_2 という出現順が混在している）．

節点 v が low(v)=high(v) を満たすとき，v は**冗長**な節点と呼ばれる．このような節点は**図-3**(a) のように，BDD の表現する論理関数を変更することなく削除することができる．節点 u と v が low(u)=low(v)，high(u)=high(v) を満たすと



(a) Removal of a redundant node　(b) Removal of an equivalent node

図-3 冗長節点，等価節点の削除

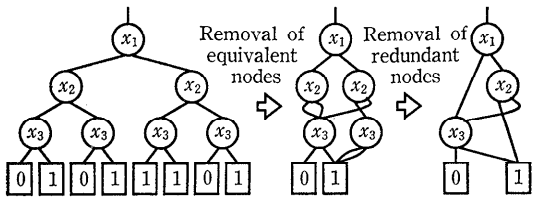


図-4 BDD の既約化

き，u と v は**等価**であると言われる．u と v は同一の論理関数を表現する．等価な節点の一方は，**図-3**(b) に示すように，BDD の表現する論理関数を変更することなく削除することができる．冗長な節点と等価な節点の削除をもうこれ以上できなくなるまで繰り返す操作を**既約化** (reduction) と言う．順序付き BDD に既約化をほどこして得られるものが，**既約な順序付き BDD** (以下単に**既約な BDD**) である．図-2(a) の順序付き BDD に既約化を行うと，(b) の既約な BDD が得られる (**図-4** 参照)．

3. BDD の特長

既約な BDD は次に示す三つの著しい特長をもっている．

(1) 変数順序を固定すれば，既約な BDD は論理関数の標準形となる．

(2) 多くの実用的に重要な論理関数が現実的な節点数で表現できる．

(3) 論理関数に対する演算が表現のサイズに比例する時間で行える．

以下，これらの性質について解説する．

3.1 既約表現の一意性

論理関数 f と変数の順序が与えられたとき，f をこの順序で表現する BDD は複数存在するが，これらに既約化をほどこして得られる BDD はすべて同形となることが知られている．すなわち，一つの変数順序のもとで f を表す既約な BDD は一意に定まる．詳細な証明は，文献 3) に示されているが，冗長な節点と等価な節点の除去は，その適用順序によらず同じ結果を生むことを考えれば，この結果は直観的に自明といえる．

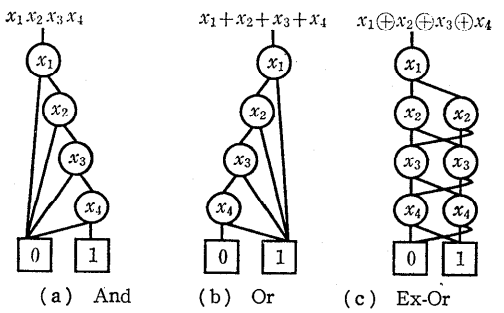
こうして，変数順序を一つ定めれば，既約な BDD は論理関数の標準形となる．これは論理関数の等価性判定が BDD，すなわち根付き非巡回有向グラフの同形判定に帰着できることを意味し，論理関数の BDD 表現さえ求めることができれば，等価性判定が現実的な時間で解けることを意味する．

3.2 BDD の節点数

前述のとおり，n 変数論理関数を識別するためには 2^n に比例する記憶量が必要である．BDD でもこれは例外ではなく，n 変数論理関数を表現するための最悪の場合の節点数は，$O(2^n/n)$ となる☆．しかし，実用上よく用いられる多くの関数の BDD 表現が n の多項式でおさえられることが示されており，ここに大きな意義がある．いくつかの具体例を通じ，どのような論理関数が現実的な記憶量で表現できるかをみてみることにしよう．

(1) n 変数の論理積，論理和，パリティ

図-5(a)，(b)，(c) はそれぞれ 4 変数の論理積，論理和，排他的論理和をとって得られる論理関数を表す BDD である．これらの否定の論理関数の BDD はこれらの BDD の定数節点の 0 と 1 を交換することにより得られる．いずれの BDD も，入力変数の数を n とすると，節点数は $O(n)$ となることが分かる．特に，積和形表現では $O(2^n)$ 個の積項が必要となるパリティ関数が $O(n)$ で表現できる意義は大きいと考えられる．



(a) And　(b) Or　(c) Ex-Or

図-5 論理積，論理和，排他的論理和の BDD

☆ 一つの節点を記憶するのに $O(n)$ ビット必要となるので，全体の記憶量は $O(2^n)$ ビット必要である．

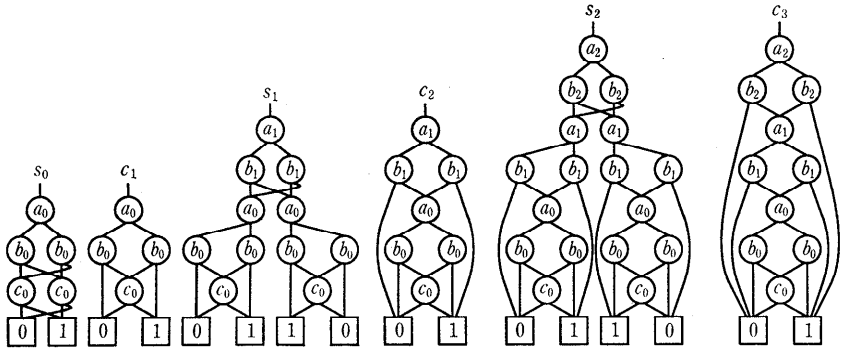


図-6　2進加算の BDD

（2） 算術加減算，比較

図-6 は2進数 (a_2, a_1, a_0) と (b_2, b_1, b_0) およびキャリー c_0 の加算結果 (c_3, s_2, s_1, s_0)，および中間結果である桁上げ c_1，c_2 の論理関数を表す BDD である．（図を見やすくするため，s_1，s_2 は一部既約化を行っていない．）2進数のビット数を n とすると，それぞれの出力の論理関数の表現の大きさはやはり $O(n)$ となる．また，2進数の比較も $O(n)$ の大きさの BDD で表現できる．

（3） セレクタの論理関数

図-7 はセレクタ（選択入力 s_0, s_1, s_2 によって，データ入力 $d_0, d_1, \cdots, d_7$ から一つを選ぶ回路）の論理関数を表現する BDD である．データ数が d のときの節点数は $O(d)$ である．論理演算，加減算，セレクタが効率良く表現できることより，ALU の論理関数なども効率良く表現することが可能である．

（4） 対 称 関 数

変数を交換しても関数の値が変わらない関数を対称関数という．（図-5 に示した論理関数はすべて対称関数である．）対称関数は，言い換えれば入力中の1の数によって値が決まる関数であり，**多数決関数**がその一例である．対称関数はしばしば制御論理にも現れるが，一般には積和形表現が入力変数の指数に比例して大きくなることが知られている．**図-8** は対称関数の BDD の一般的な形を表したものである．（一般には定数節点にさまざまな0，1の値が入り，その後に既約化がほどこされる）．左から i 列目の節点は，入力変数中の1の数が $i-1$ であることを表していると考えればよい．この図より n 変数対称関数は一般に $O(n^2)$ の節点数で表現可能であることが分かる*．

以上のように，BDD を用いれば実用上重要な論理関数の多くが，入力変数の数を n としたとき，$O(n)$ ないし $O(n^2)$ の記憶量で表現できる．したがって，このような論理関数であれば，たとえ数万ゲートからなる論理回路であってもその関数が表現可能である．ただし，BDD の形や大きさは変数の順序に依存する．実際の応用においては，変数の順序を決定する問題が重要になる．

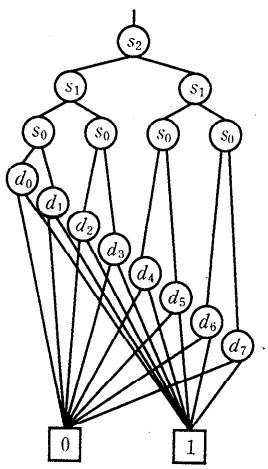


図-7　セレクタの論理関数の BDD

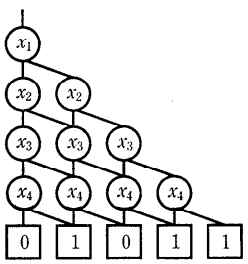


図-8　対称関数の BDD

* 文献 27) では対称関数を表現する BDD の節点数を詳しく論じている．

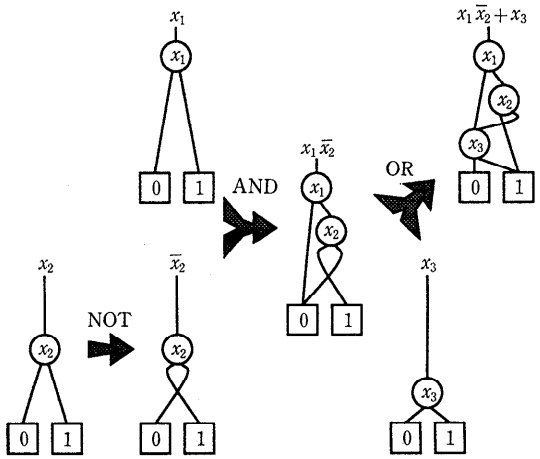


図-9 apply 演算による BDD の構築

反対に，どのような変数順によっても，BDD の大きさが $O(2^n)$ になる論理関数も知られている．実用的な関数の中では，乗算回路の出力に現れる論理関数の BDD はどのような変数順のもとでも指数規模となることが証明されている[4]．どのような論理関数が表しやすくどのような論理関数が難しいかについては，**5.**で述べる．

3.3 BDD の演算効率

ある論理関数の BDD を人手で得る最も簡単な方法は，真理値表から図-2(a) のような二分木を構成し，これを既約化するというものである．しかし，真理値表の大きさは 2^n であり，計算機上で実現する場合にはこの時点で 2^n の記憶量が必要となって，BDD 表現を用いる利点がなくなってしまう．

目的の論理関数の BDD を構築する効率的な方法は，論理関数に論理演算を適用していく方法である．たとえば，$x_1\bar{x}_2+x_3$ の BDD を構築する場合，**図-9** のようにまず，x_2 という論理関数（一変数論理関数）の BDD に論理否定演算をほどこして $\bar{x}_2$ の BDD を作り，次に x_1 との論理積演算，x_3 との論理和演算によってそれぞれ $x_1\bar{x}_2$，$x_1\bar{x}_2+x_3$ の BDD を得るというものである．

このように，論理関数に対して論理演算を行って新たな論理関数を得る演算を **apply 演算**と呼ぶ．そのアルゴリズムの詳細は後の解説[25]に譲るが，apply 演算はオペランドとなる BDD の大きさの積に比例する時間で実行できる．とくに，積和形表現では否定を計算する際に膨大な計算時間が必要となることがあったが，BDD の場合は与えられた BDD の大きさに比例する時間（後の解説[25]で紹介される否定エッジを用いれば定数時間）でこれを計算することが可能である．

また apply 演算のほかに，BDD の表す関数において，ある変数に 0 または 1 を代入して得られる関数の BDD を求める代入演算も，グラフの大きさに比例する時間で計算できる．これを用いて，ある変数に関数を代入する演算—たとえば $f(x_1, x_2, \cdots, x_n)$ に $x_1=g(x_2, \cdots, x_n)$ を代入し，$f(g(x_2, \cdots, x_n), x_2, \cdots, x_n)$ を求める—や，変数の交換を行う演算—$f(x, y)$ から $f(y, x)$ を求める—も効率良く行うことができる．

BDD では，論理関数が充足可能かどうか（すなわち恒偽関数でないかどうか）をただちに判定することができるが，さらに，充足可能である場合には論理関数の値を 1 にする入力割当を求めることができる．この計算時間は BDD の大きさではなく，入力変数の数に比例する．また，論理関数の値が 1 になる割合（真理値表密度）や，各入力変数が 1 をとる確率が与えられているときに論理関数の値が 1 になる確率も，BDD の大きさに比例する時間で計算することができる．

3.4 BDD の改良

BDD の記憶効率や演算効率をさらに向上させるため，さまざまな改良が試みられている．詳細は後の解説[25]で紹介されるが，一つは複数の BDD 間でのグラフの共有と既約化（global share, **図-10** 参照）[2),23)]であり，もう一つはさまざまな属性エッジの導入である[23]．グラフの共有化を行うと，多数の論理関数が同時に効率良く表せるだけでなく，演算の効率が向上し，特に関数の等価性判定が BDD の大きさに依存しない定数時間で行

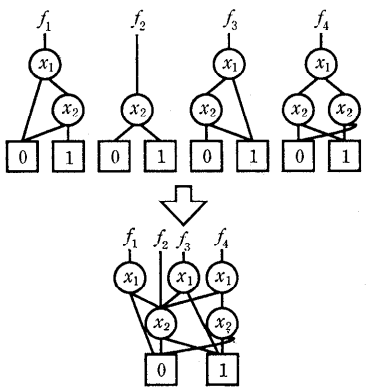


図-10 共有二分決定グラフ

えるようになる．属性エッジには，否定エッジ，入力反転エッジ，変数シフトエッジなどさまざまなものが考えられているが，これらの導入により，記憶効率と演算効率が改善される．

4. BDD の応用

上記 BDD の特長をまとめると，「多くの実用的な論理関数の BDD は多項式サイズであり，多項式時間で構築できる．また，多項式サイズの BDD に対しては，等価性や充足可能性の判定も多項式時間で行える．」ということになる．これは，(多少強引ではあるが)「NP 完全問題や co-NP 完全問題の多くの実用的な問題例 (instance) が多項式時間で解ける」ことを意味する．

では，実際にどの程度 NP 完全問題が解けるかーは問題にもよるが，LSI の CAD の分野では，組合せ回路の等価性判定問題や順序回路の検証問題など，BDD を用いることによってはじめて実用規模の問題が解けた例が存在する．

BDD の応用の詳細は以降の解説[12),31),30)]で詳しく述べられるので，ここでは，BDD の応用されている問題を簡単に分類するにとどめる．

(1) 論理関数を用いて直接定式化できる問題：

二つの組合せ回路の実現する論理関数の等価性は，二つの回路の論理関数を表現する BDD を apply 演算によって構築すれば，たちどころに判定できる[11),19)]．組合せ回路のテスト生成も，正常回路の論理関数 f と故障回路の論理関数 f_α を計算し，$f \oplus f_\alpha$ を1にする割当てを求めることにより行える[8),13),28)]☆．また，論理合成において，従来真理値表や積和形であった内部表現を BDD に置き換えるという手法[20)]なども，この方法に分類される．

一般の最適化問題は，問題のインスタンスを2進符号化し，制約条件を論理関数で表現すれば，全許容解の集合を求められることを利用して解くという試みがなされている[31)]．

(2) 集合の「特徴関数」を BDD で表現することにより解かれる問題：

有限集合 U の各要素を2進符号化し，U の部分集合 S に対して $\chi_S(x)=1$ iff $x \in S$ なる関数 χ^S (S の**特徴関数**と呼ばれる) を考える．S を表現する際，χ_S の BDD 表現を用いれば，リストなどを用いてその要素を直接列挙する方法では表現が不可能な巨大な集合を表現することが可能になる．和集合，積集合，補集合などの集合演算は，それぞれ，特徴関数に対する論理和，論理積，論理否定演算に置き換えることができる．この方法を用いれば，巨大なグラフを扱うことが可能になり，従来，限られた大きさの状態遷移グラフにしか適用できなかった順序回路の検証法や順序回路のテスト生成法が，1000 ゲート以上，30 フリップフロップ以上の回路に対しても適用できるようになる[5),7),9)]．また，故障リストが巨大となる多重故障の故障シミュレーションに適用した例[29)]もある．

(3) BDD のグラフとしての性質を利用して解かれる問題：

文献 17), 21) では，集合被覆問題が，ある BDD の根から 1 でラベル付けされた定数節点へ至る最短パスを求める問題に帰着されることを利用して，この問題を効率良く解いている．また，文献 15) では，論理関数が1になる確率が容易に計算できることを利用して，回路の誤動作が起こる確率を求めている．

5. BDD の能力と限界

どのような論理関数が BDD で効率良く表現でき，どのような関数ができないのかは BDD を利用する上で重要な問題である．完全な解答はまだ出されていないが，文献 14) および文献 4) にそれぞれ入力変数の数 n の多項式の節点数で表現できる関数のクラス，n の指数の節点数が必要な関数のクラスに関する議論がなされている．

n の多項式の節点数で表現できる関数のクラスを考えるときには，対称関数の BDD の形がヒントになる．この BDD は，見方を変えれば，$x_1, x_2, \cdots, x_n$ の値が逐次的に入力される順序機械

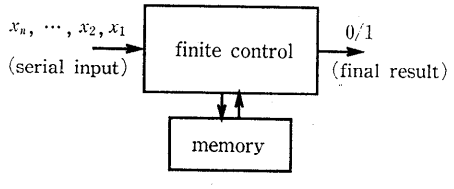


図-11 順序機械

☆ ただし，従来のテスト生成法に比べると，必ずしも効率が良いとはいえない．

(図-11) の状態遷移図とみることができる．左から i 番目の「状態」は，これまでに入力された1の数が $i-1$ 個であることを記憶するものである．BDD の高さは n であるから，BDD の幅，すなわち各時刻での状態数が多項式でおさえられていれば全体の節点数が多項式でおさえられる．さらに言えば，この状態を記憶するための記憶量（フリップフロップ数）が $O(\log n)$ でおさえられていれば BDD は現実的な大きさで済むことになる．

論理演算や2進数の加算，比較などは有限の状態（定数個のフリップフロップ）の順序機械で計算できる関数であるから，BDD でも効率良く表現できる（節点数 $O(n)$）．また，セレクタ，対称関数などはデータ数を数えることができれば計算できるので，$O(\log n)$ 個フロップで計算でき，BDD でも効率良く表現できることになる．

逆に逐次的な計算において，多くのデータを記憶せねばならない関数の場合には指数個の節点が必要になる．これは逐次的に入力されるデータを，計算のいろいろな時点で参照せねばならない場合などにおこる．文献 4) では，このような関数を LSI 上に実現すると，回路を2分割したときに断面を横切る信号線数が多くなることに基づいて，このような関数のクラスの面積計算時間積の複雑さに関する結果を導いている．

BDD を利用して問題を効率的に解こうとするときには，変数の順序づけが重要な問題になる．最適な（目的の BDD が最小になる）変数順序を求めることは計算困難な問題と考えられ，$O(n^2 3^n)$ のアルゴリズム（n は入力変数の数）しか知られていない[10]．このため，実際には最小の BDD 表現が入力変数の数の線形となるような論理関数に対しても，17 変数程度までしか扱うことができない[16]．発見的な手法は種々提案されているが[6),11),19),23),26)]，万能のものは存在しないようである．実用化を考える上では，その応用に適した良い変数順序の計算法が最も重要な課題になると考えられる．

6. むすび

本稿では BDD とその特長や応用を紹介した．BDD を用いることによって，今まで実際的な時間で解くことのできなかった問題の中に，解けるものが現れてきており，今後の実用化に向けての研究が期待される．

謝辞 本稿をまとめるにあたり，ご討論いただいた NTT の湊真一氏，富士通研究所の藤田昌宏氏，九州大学の安浦寛人教授，CMU の E. M. Clarke 教授と R. E. Bryant 教授，京都大学矢島研究室の諸氏，および大阪大学白川研究室の諸氏に感謝する．

参考文献

1) Akers, S. B.: Binary Decision Diagrams, *IEEE Trans. Comput.*, Vol. C-27, No. 6, pp. 509-516 (June 1978).
2) Brece, K. S., Rudell, R. L. and Bryant, R. E.: Efficient Implementation of a BDD Package, *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 40-45 (June 1990).
3) Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trads. Comput.*, Vol. C-35, No. 8, pp. 677-691 (Aug. 1986).
4) Bryant, R. E.: On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication, *IEEE Trans. Comput.*, Vol. C-40, No. 2, pp. 205-213 (Feb. 1991).
5) Burch, J. R., Clarke, E. M., Mcmillan, K. L. and Dill, D. L.: Sequential Circuit Verification Using Symbolic Model Checking, *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 46-51 (June 1990).
6) Butler, K. M., Ross, D. E., Kapur, R. and Mercer, M. R.: Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams, *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 417-420 (June 1991).
7) Cho, H., Hachtel, G. D. and Somenzi, F.: Fast Sequential ATPG Based on Implicit State Enumeration, *Proc. IEEE Int. Test Conf.*, pp. 67-74 (Oct. 1991).
8) Cho, K. and Bryant, R. E.: Test Pattern Generation for Sequential MOS Circuits by Symbolic Fault Simulation, *Proc. 26th ACM/IEEE Design Automation Conf.*, pp. 418-423 (June 1989).
9) 崔 渓鎔，小原隆司，石浦菜岐佐，白川 功：共有二分決定グラフを用いた順序回路のテスト生成法，情報処理学会研究会報告，設計自動化 62-28 (May 1992).
10) Friedman, S. J. and Supowit, K. J.: Finding the Optimal Variable Ordering for Binary Decision Diagrams, *IEEE Trans. Comput.*, Vol. C-39, No. 5, pp. 710-713 (May 1990).
11) Fujita, M., Fujisawa, H. and Kawato, N.: Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Dia-

grams, *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 2-5 (Nov. 1988).
12) 藤田昌宏, Clarke, E. M.: BDD の CAD への応用, 情報処理, Vol. 34, No. 5, pp. 609-616 (May 1993).
13) Gaede, R. K., Mercer, M. R., Bulter, K. M. and Ross, D. E.: CATAPULT: Concurrent Automatic Testing Allowing Parallelization and Using Limited Topology, *Proc. 25th ACM/IEEE Design Automation Conf.*, pp. 597-600 (June 1988).
14) 石浦菜岐佐, 矢島脩三: 多項式サイズの二分決定グラフで表現可能な論理関数のクラス, 情報処理学会アルゴリズム研究会, 20-5 (Mar. 1991).
15) Deguchi, Y., Ishiura, N. and Yajima, S.: Probabilistic CTSS: Analysis of Timing Error Probability in Asynchronous Logic Circuits, *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 650-655 (June 1991).
16) Ishiura, N., Sawada, H. and Yajima, S.: Minimization of Binary Decision Diagrams Based on Exchanges of Variables, *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 472-475 (Nov. 1991).
17) Lin, B. and Somenzi, F.: Minimization of Symbolic Relations, *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 88-91 (Nov. 1990).
18) Madre, J. C. and Billon, J. P.: Proving Circuit Correctness Using Formal Comparison between Expected and Extracted Behavior, *Proc. 25th ACM/IEEE Design Automation Conf.* pp. 205-210 (June 1988).
19) Malik, S., Wang, A. R., Brayton, R. K. and Sangiovanni-Vincentelli, A.: Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment, *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 6-9 (Nov. 1988).
20) Matsunaga, Y. and Fujita, M.: Multi Level Logic Optimization Using Binary Decision Diagrams, *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 556-559 (Nov. 1989).
21) 松本 忍, 矢島脩三: ブール式処理による不完全指定順序機械の最小化, 情報処理学会論文誌, Vol. 31, No. 11 (Feb. 1990).
22) McGeer, P. C., Saldanha, A., Stephan, P. R., Brayton, R. K. and Sangiovanni-Vinceltelli, A.: Timing Analysis and Delay-Fault Test Generation Using Path-Recursive Functions, *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 180-183 (Nov. 1991).
23) Minato, S., Ishiura, N. and Yajima, S.: Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation, *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 52-57 (June 1990).
24) 湊 真一: 二分決定グラフからの非冗長積和形の高速生成手法, 電子情報通信学会技術研究報告, VLD 91-107 (Dec. 1991).
25) 湊 真一: 計算機上での BDD の処理技法, 情報処理, Vol. 34, No. 5, pp. 593-599 (May 1993).
26) Minato, S.: Minimum-Width Method of Variable Ordering for Binary Decision Diagrams, *IEICE Japan Trans. Fundamentals*, Vol. E75-A, No. 3 (Mar. 1992).
27) Ross, D. E., Butler, M. K. and Mercer, M. R.: Exact Ordered Binary Decision Diagram Size When Representing Classes of Symmetric Functions, *J. Electronic Testing: Theory and Applications*, Vol. 2, pp. 243-259 (1991).
28) Stanion, T. and Bhattacharya, D.: TSUNAMI: A Path Oriented Scheme for Algebraic Test Generation, *Proc. IEEE Int. Symp. on Fault-Tolerant Computing*, pp. 36-43 (June 1991).
29) Takahashi, N., Ishiura, N. and Yajima, S.: Fault Simulation for Multiple Faults Using Shared BDD Representation of Fault Sets, *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 550-553 (Nov. 1991).
30) 渡部悦穂, 久木元裕治: BDD の応用, 情報処理, Vol. 34, No. 5, pp. 600-608 (May 1993).
31) 柳谷雅之: 組合せ最適化問題の BDD による解法, 情報処理, Vol. 34, No. 5, pp. 617-623 (May 1993).

(平成5年1月26日受付)

石浦菜岐佐 (正会員)

昭和36年生. 昭和59年京都大学工学部情報工学科卒業. 昭和61年同大大学院修士課程修了. 昭和62年1月京都大学工学部助手. 平成3年3月京都大学工学博士. 平成3年5月より大阪大学工学部情報システム工学科講師. 論理回路の設計検証, テスト, 論理合成の研究に従事. 電子情報通信学会, IEEE 各会員.