Vol. 34 No. 5　　情　報　処　理　　May 1993

解　説

BDD（二分決定グラフ）

# 2. 計算機上での BDD の処理技法†

湊　　真　一††

## 1. はじめに

論理関数を計算機上で効率よく表現し，高速に論理演算を行うことは，LSI 設計システムの基盤技術として重要であるだけでなく，一般のアルゴリズム論や計算理論にも関わる問題である．論理関数のデータ表現に BDD (Binary Decision Diagrams：二分決定グラフ）を用いる処理方法は，記憶効率や計算速度の面で優れており，最近盛んに用いられている．BDD に関する研究は，計算機上に処理系を実現する技術をもとにして，主として実用的観点から進められてきた[1)~5)]．現在では，グラフの節点数が100万を超えるような非常に大規模な BDD を現実的な時間内で扱うことが可能となっており，BDD の話題を迫力あるものにしている．本稿では，計算機上で BDD を効率的に処理する技法について解説する．

以下では，まず「論理関数の処理」とは具体的に何をすることかを述べ，続いて BDD を用いた論理関数の処理アルゴリズムを解説する．さらに，BDD を応用する際に重要となる入力変数の順序づけの問題について述べる．

## 2. 論理関数の処理とデータ構造

計算機上での論理関数を処理するための基本的操作として，以下のようなものがあげられる．

- 論理関数データに対して AND, OR, NOT などの論理演算を行い，演算結果の論理関数データを生成すること．すなわち，任意のブール式に対応する論理関数データ表現を構築できること．
- 二つの論理関数データの等価性判定が行えること．これは論理関数の恒真・恒偽判定や包含性判定を行う際にも必要となる．
- 論理関数を真にするような入力の組合せ（充足解）を見つけること．あるいは充足解の個数を数えること．

これらの基本的操作を組み合わせることにより，論理関数に対するさまざまな処理を行うことができる．計算機上での論理関数のデータ構造を考える場合，上記の基本的操作を高速に実行できることが要求される．そのためには，なるべく記憶量の少ない表現であることが望ましい．

従来，論理関数の表現形式としては，真理値表や積和形が多く用いられてきた．しかし，真理値表は，簡単な論理式で表せるような論理関数でも，$2^n$ ビット（$n$ は入力数）の記憶量を必要とし，論理演算や等価性判定などの基本的操作にも指数時間を要するため，入力数が多くなると実用的でない．

一方，積和形は，簡単な式で書けるような論理関数に対しては，真理値表よりも少ない記憶量で表せることが多いため，これまで多く用いられてきた．しかし，処理中に発生する冗長な積項を取り除くための簡単化処理が必要で，等価性判定に時間がかかること，算術演算でよく現れるパリティ関数が効率よく表せないこと，及び，否定演算に手間がかかることなどの問題点があった．

これに対して BDD は，論理関数の性質によりサイズは変化するが，パリティ関数や加算器などを含む多くの実用的な論理関数を少ない記憶量で表現することができる．そして BDD データが計算機の主記憶に収まる限りは，本稿で示す処理技法により，上記の基本的操作を，データ量にほぼ比例する時間内で効率よく実行できる．特記すべき点としては，積和形表現が苦手とする否定演算が容易であることと，BDD が論理関数を一意に表すために，等価性判定が即座に行えることがあ

† Techniques for BDD Manipulation on Computers by Shin-ichi MINATO (NTT LSI Laboratories).
†† NTT LSI 研究所

げられる.

## 3. BDD の論理演算アルゴリズム

BDD は，**図-1** に示すような論理関数のグラフによる表現である．これは，論理関数の Shannon 展開（ある入力変数に，0, 1 の値を代入して二つの部分関数を得る手続き）をすべての変数について再帰的に適用した結果を二分木グラフで表し，これを縮約したものである．このとき，展開する入力変数の順序を固定し，冗長な節点の削除と，等価な部分グラフの共有を可能な限り行うことにより「既約」なグラフが得られ，論理関数をコンパクトかつ一意に表すことができる☆.

さらに，複数の論理関数を表す BDD の間においても，入力変数の順序を一致させれば互いに部分グラフを共有することができる．処理中に現れるすべての BDD について，共有可能な部分グラフは必ず共有するようにしたもの（**図-2**）を共有二分決定グラフ（SBDD : Shared BDD）と呼ぶ[5]. SBDD は記憶効率がよいだけでなく，グラフのコピーが不要であること，および論理の一致判定がポインタ比較だけで行えるという強力な特長を有

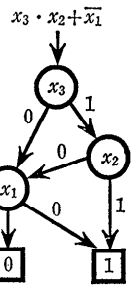



**図-1** BDD（二分決定グラフ）

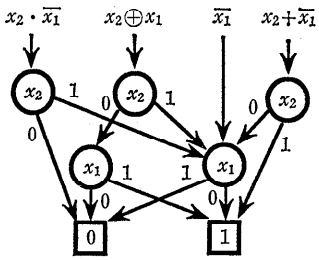



**図-2** Shared BDD（共有二分決定グラフ）

☆一般に BDD という場合には，変数の順序が固定されないものや，既約でないグラフも対象に含まれるが，本稿では上記の簡約化処理を施したものだけを扱う．なお，BDD の概要については，本特集解説[14]を参照されたい.

している.

BDD の論理演算アルゴリズムに関しては，複数 BDD の共有化を行わないオリジナルな技法が Bryant[2] により示されている．現在では複数 BDD を統一的に扱う SBDD の技法が広く用いられており，SBDD 上のアルゴリズムのほうがより簡明であることから，以下の本文では SBDD 技法にもとづく処理系について解説する.

### 3.1 データ構造

BDD 処理系は，すべての BDD の節点を統一的に管理している．各節点は，入力変数の番号，0 枝，1 枝の三つの属性をもち，さらに，グラフを管理するためのポインタやカウンタなどが付属している．典型的な実現においては，節点を格納する記憶領域は 0 番地からの連続領域にテーブルとして確保され，0 枝，1 枝はそれぞれの行き先の節点の番地として格納される．図-2 のグラフをテーブル上に表現した例を**図-3** に示す.

0 および 1 の定数節点は，特別な節点としてあらかじめ定義しておく．(たとえば，0 番地と 1 番地にそれぞれ割り当てる.) ある節点が定数節点であるか否かは，その番地を調べることにより即座に判別できる.

入力変数そのものを表す BDD は，二つの枝がそれぞれ 0，1 の定数節点を指しているような 1 個の節点からなる．なお，本稿では入力変数の番号として自然数を用い，最下位の変数を 1 とし，上位ほど大きな番号で表しているが，上位下位の区別がつけばどんな番号づけでもよい.

### 3.2 節 テーブルによる一意性の保証

BDD 処理系においては，共有可能な部分グラフは必ず共有されていなければならない．すなわち，同じ属性をもつ節点が複数存在してはならない．そこで，変数番号，0 枝，1 枝の三つの属性

| | $v$ | $f_0$ | $f_1$ | |
|---|---|---|---|---|
| 0 | — | — | — | ← 0 |
| 1 | — | — | — | ← 1 |
| 2 | 1 | 0 | 1 | |
| 3 | 1 | 1 | 0 | ← $\overline{x_1}$ |
| 4 | 2 | 0 | 3 | ← $x_2 \cdot \overline{x_1}$ |
| 5 | 2 | 2 | 3 | ← $x_2 \oplus x_1$ |
| 6 | 2 | 3 | 1 | ← $x_2 + \overline{x_1}$ |

**図-3** テーブルによる BDD の表現

をキーとするハッシュテーブルを設け，これに全節点を登録することにより一意性を保証している．このテーブルを「節テーブル」と呼ぶ．新たな節点を生成する前には必ず節テーブルを参照し，等価な節点がすでに登録されていれば，その節点を共有して使用し，重複する節点は一切作らないようにしている．

節テーブルの参照・登録は，BDD の処理中に頻繁に行われるため高速な動作が要求される．ハッシュテーブルが良好に動作している限りは，この操作は節点数に関わらず定数時間で実行できる．

この技法により，BDD 処理系で生成された論理関数は，BDD の根の節点の番地により一意に表現されることになる．したがって，論理の等価性判定や恒真判定を，グラフの節点数に関わらず，根の節点の番地比較だけで行うことができる．

さらに BDD では，ある節点の二つの枝が等価な部分グラフを指している場合には，その節点を削除し，部分グラフに枝を直結させるという簡約化規則がある．そこで，新たな節点を生成する前には必ず二つの部分グラフの等価性判定を行い，冗長な節点は最初から作らないようにしている．この等価性判定も 0 枝，1 枝の指す番地の比較により高速に行える．

### 3.3 二項論理演算

与えられたブール式の論理を表す BDD を生成するには，まず各入力変数を表す BDD を生成し，ブール式の構文にしたがって BDD どうしの二項論理演算処理を繰り返し適用し，式全体の論理を表す BDD を構築していく．二項論理演算アルゴリズムは，BDD 処理技術の中で最も重要な部分である．

ある二つの論理関数 $f, g$ の二項論理演算(AND, OR など) の結果を表す $f \circ g$ の BDD を生成するアルゴリズムについて述べる．このアルゴリズムは，次の Shannon の展開式に基づく．

$$f \circ g = \bar{v} \cdot (f|_{v=0} \circ g|_{v=0}) + v \cdot (f|_{v=1} \circ g|_{v=1})$$

これは，ある一つの入力変数 $v$ に着目したとき，$v=0,1$ のときに場合分け（コファクタを抽出）して，それぞれについて演算すればよいということを示している．この公式にしたがって，BDD の上位の変数から順に展開して，それぞれの部分グラフどうしの演算を再帰的に実行する．定数値に関する自明な演算になったところで，再帰が打ち切られ，結果が返される．

具体的には，次に示す方法で $h(=f \circ g)$ を求める．ここで，$f$ のグラフの根の節点の変数を $f.top$ とし，0 枝および 1 枝の指すグラフをそれぞれ $f_0, f_1$ と書くことにする．

1. $f, g$ のいずれかが定数のとき，
   及び $f=g, f=\bar{g}$ のとき，
   演算子の種類に応じた処理を行う．
   (例) $f \cdot 0 = 0$, $f+f=f$, $f \oplus 1 = \bar{f}$
2. $f.top$ と $g.top$ が同じとき，
   $h_0 \leftarrow f_0 \circ g_0$ ; $h_1 \leftarrow f_1 \circ g_1$ ;
   if $(h_0 = h_1)$ $h \leftarrow h_0$ ;
   else $h \leftarrow Node(f.top, h_0, h_1)$ ;
3. $f.top$ が $g.top$ より上位のとき．
   $h_0 \leftarrow f_0 \circ g$ ; $h_1 \leftarrow f_1 \circ g$ ;
   if $(h_0 = h_1)$ $h \leftarrow h_0$ ;
   else $h \leftarrow Node(f.top, h_0, h_1)$ ;
4. $f.top$ が $g.top$ より下位のとき，
   ($f, g$ を入れ替えて 3. と同様に処理)

上記のアルゴリズム中，新しい節点を作る際に節テーブルを参照して重複を避けることは，前述のとおりである．

たとえば**図-4** に示した BDD において，節点(1)と(5)に関する二項演算を行う場合，その再帰呼び出しの過程は，**図-5** のような二分木となる．通常の逐次処理系では，この二分木を深さ優先順にたどりながら計算を行うことになる☆．

このアルゴリズムをそのまま実現すると，図-5 の(3)-(7)，(4)-(7)，(4)-(8)の組のように，同じ演算を何度も重複して行うことになり処

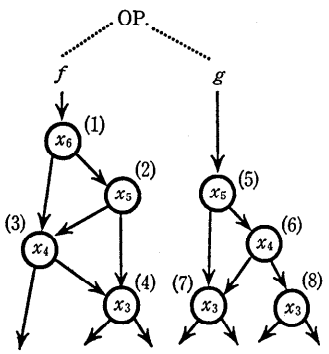



**図-4** 二項論理演算の例

☆ 並列処理系のアルゴリズムについては，本特集解説[18]を参照されたい．

理効率が悪い．そこで，過去に演算を行った節点の組とその演算結果を記録しておく「演算結果テーブル」を用意し，同じ演算がすでに登録されていれば，再帰処理を打ち切り即座に結果を返すという高速化技法が用いられている．この工夫により，$f, g, h$ のグラフの節点数の総和にほぼ比例する時間で論理演算を行うことができる[2]．

演算結果テーブルはハッシュテーブルとして実現されるが，過去のすべての演算を記録すると記憶量が大きくなり過ぎるため，サイズを限定して，最近の演算だけを記憶する手法（キャッシュ）がとられる．このサイズが不十分だと，必要な情報が忘れられて無駄な演算が多くなり，急激に処理速度が低下する．多くの場合，実験的に適当なサイズを決めている（扱う BDD の節点数の数分の1〜数倍程度）．テーブルがヒットする確率を上げるため，枝が集中している節点の演算だけを登録するようにしたり，可換な演算ならばどちらか1通りだけ登録するなどの技法が用いられている．

### 3.4 否定演算と否定枝

ある論理関数 $f$ の否定 $\bar{f}$ を表す BDD は，$f$ の BDD の0，1の定数節点を交換した形となる．積和形表現では否定演算を行うとデータ量が著しく増大することがあるが，BDD では，相補な関係にある論理関数のデータ量は常に等しい．

否定演算を大幅に高速化する手法として「否定枝」の技法が広く用いられている[1),4)]．否定枝とは，BDD の枝に否定演算を表す属性を与えるもので，グラフをたどる際には否定枝を通った回数だけ論理を反転させるという意味をもつ．これにより，互いに否定の関係にある BDD を完全に共有することができ(**図-6**)，記憶量が最大 50% 削減される．また，グラフの根を指している枝の属性を変更するだけで，否定演算処理を瞬時に行うことができる．

否定枝を無制限に使用すると，同じ論理関数を何通りにも表せるため，グラフの一意性が失われてしまう．一意性を保つためには次のような制限を設ければよい[4),5)]．

- 定数節点の値は0だけにする．
（1は0の否定．）
- 0枝には否定枝を用いない．
（必要なら，より上位の節点で用いる．）

なお，否定枝のほかにも，種々の属性を枝に付加する効率化手法が提案されている[5)]．

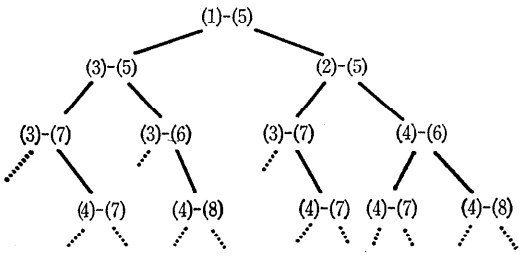



**図-5** 二項論理演算の再帰呼び出し過程

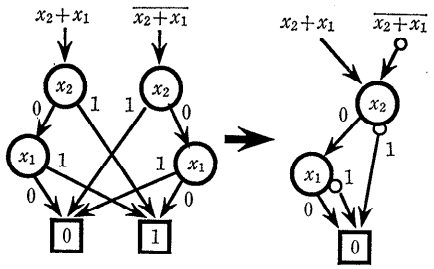



**図-6** 否定枝の使用例

### 3.5 代入（コファクタ）演算

生成した BDD に対して，ある入力変数に 0/1 の定数を代入したときの BDD（コファクタ）を生成する操作が必要になることがしばしばある．最上位の変数に代入する場合は，対応する 0/1 枝の指す部分グラフを返せばよい．その他の場合には，代入する変数が最上位になるまで BDD を展開し，それぞれに代入したあとでグラフを再構築する必要がある．この操作は，二項論理演算の場合と同様に，節テーブルと演算結果テーブルの技法を用いて効率よく実行できる．計算時間は，代入する変数より上位にある節点の数に比例する．

### 3.6 充足解の探索

論理関数の BDD を生成できれば，その充足解を見つけることは容易である．これは，BDD の根から1の定数節点へ達するパス（1-パスと呼ぶ）を見つければよい．BDD の途中の各節点からは，恒偽関数でない限りは少なくとも1個の 1-パスが存在するので，0 の定数節点を避けるようにたどっていけば，必ず 1-パスが得られる．得られたパスを活性化させるような入力値の組合せが充足解となる．このときの計算時間は入力数に比例し，BDD の節点数には依存しない．

一般に充足解は複数存在する．そこで，各入力変数の値に対して，ある種のコストを定義し，コ

ストの総和を最小にするような入力組合せを，BDD の節点数に比例する時間で効率よく探索する方法が知られており[6]，組合せ最適化問題に応用されている（本特集解説[15),17)]参照）

また，充足解の個数を計算することもできる．BDD の各節点において，0/1 枝の指す二つの部分グラフの充足解の個数を求め，これらを加えればよい．(ただし，変数番号に飛び越しがある場合は，飛び越した変数の分だけ2倍する．)各節点における計算結果をキャッシュに記憶しておけば，BDD の節点数に比例する時間で計算することができる．

同様な方法で，真理値表濃度（論理関数を真理値表で表したときの1の割合）を計算することができる．これは，各節点において，二つのサブグラフの真理値表濃度の平均をとることにより求められる．

## 4. BDD 処理系の記憶管理

BDD 処理系が使用する記憶量は，通常，1ノードあたり 20〜30 Byte 程度となる．昨今では，100 MByte を越える主記憶を備えた計算機も珍しくなく，これにより数百万ノードもの BDD を扱うことが可能になっている．しかしそれでもなお主記憶容量がネックとなることが多い．

ブール式から BDD を生成する際の途中計算結果のように，処理中に一時的に生成され，二度と参照されない BDD が多く発生する．そのような不要 BDD は削除して記憶領域の有効利用を図ることが実用上不可欠である．このとき，不要な BDD が他の BDD と節点を共有している場合には，その部分グラフは残しておかなければならない．そこで，各節点ごとに，自分自身を参照している枝の本数を記憶するカウンタを設けることにより，各節点の必要性を判定している．すなわち，ある BDD が不要になったときは，その根の節点のカウンタの値を1減らし，0になれば実際に削除する．削除が実行されたときには，0/1 枝の指している節点が新たに削除される可能性があるので再帰的に処理する．

カウンタによる記憶管理を行う場合には，処理系使用者が勝手に BDD へのポインタを複製することは許されない．たとえばC言語で記述された処理系では，ポインタの複製は，すべて処理系の提供する複製命令を使用し，処理が終了したら削除命令を与えなければならない．そして複製と削除の回数は一致していなければならない*．

節点が削除されたとき，演算結果テーブルにその節点のデータが残っていると，テーブルの正当性が損なわれるので注意が必要である．削除された節点に関するデータをテーブルからすべて選び出して除去することは困難であるため，テーブル全体をいったん初期化するなどの対策が必要となる．処理速度低下を抑えるため，記憶に余裕があればカウンタが0になっても削除せず温存しておき，記憶量が限界に達したときにガベジコレクションを行い，不要な節点をまとめて削除するという技法が有効である．これにより，削除命令の直後に再び必要となった場合でも即座に回復でき，演算結果テーブルの初期化回数も最小限に抑えられる．

ところで，BDD 処理系ではハッシュテーブルを使用しているため，最初にまとめて記憶領域を確保する必要があるが，最終的な BDD の節点数は実行してみないと分からない場合が多い．さりとて，不必要に多く確保するとほかのプログラムから使えなくなり不経済である．この解決策としては，初めに適当に切りのよいサイズを確保しておき，足りなくなったら，次に切りのよいサイズまで拡張するようにして，徐々に増やしていくのがよいと考えられる．

## 5. 入力変数の順序づけ

一般に BDD は，入力変数の順序によって同じ論理関数でも異なる形となり，その節点数も変化する．順序づけの影響の大きさは論理関数の性質に依存し，対称関数ではまったく変化しないが，実用的な論理関数においては，著しく変化することが多い．ブール式や論理回路から BDD を生成する際，変数順が不適当だと BDD の節点数が爆発的に増大し，記憶あふれを起こして処理不能になることもある．入力変数の順序づけは実用上重要な問題である．

任意の論理関数に対して節点数最小となる変数順を求めることは，計算困難な問題と考えられている[2)]．これまでに提案されている解法では

* C++ を用いれば，カウンタの管理はコンパイラに任せることができる．

17変数程度が限界となっており[10]，これより変数が多い場合は最適解を得るのは難しい．ただし，最適でなくても比較的よい順序が得られれば実用的には有効であり，これまでに種々の発見的手法が提案されている．

論理回路から BDD を生成する場合，人手で作成した回路図の順序をそのまま用いると比較的良い結果が得られることが多いことから，回路構造と変数順に関してはなんらかの相関があると考えられる．一般には

- 局所計算性のある入力どうしは近い順位に
- 出力を制御する力の強い入力は上位に

という二つの経験則が知られている．

この経験則をもとに，深さ優先順にネットをたどっていき，先にたどり着いた入力から順に上位に配置する，という戦略にもとづく順序づけ手法が提案されている[7),8)]．そのほかにも，ある規則でネットに重みづけを行い順序を決める方法[5)]やテスト生成用の統計的尺度を利用する方法[9)]などがある．いずれの手法も，主に回路のトポロジカルな情報を利用する方法である．その有効性は回路の構造や論理関数に依存し万能ではないが，実験結果によれば，多くの実用的な論理回路において，無作為な順序に比べ相当良い結果が得られることが示されている．

一方，ある変数順序で生成された BDD について，その変数順序を並べ換えることにより節点数を削減する方法も研究されている．隣接する2変数の交換は，その2変数に関する節点以外は変化しないため比較的容易に行える．そこで隣接変数の交換に基づく逐次改善手法が提案され，良好な結果が報告されている[11)]．またこれを拡張して，隣接する3個以上の変数の交換を行う手法や，乱数を利用したアニーリング法[10)]，および，BDD の「幅」と呼ぶ値に着目して変数を入れ換える手法[12)]などがある．これらの変数の置換にもとづく方法は，まず適当な変数順序で BDD が生成できなければ適用できないが，BDD のもつ論理的な情報を利用できるので効果は大きい．他の順序づけ手法と組み合わせて最後に用いるのがよいと考えられる．

## 6. まとめ

以上で解説した技法にもとづく BDD 処理系は，これまでに世界各地の研究機関で実現及び改良が行われ，その中には BDD パッケージとして無料配布されているものもいくつかある*．これらの処理系を利用して，LSI CAD への応用を中心とする多くの研究が活発に進められている[16)]．

BDD を用いた処理の特長は，変数の展開順序を固定することにより，論理関数が内包する冗長性を自動的に抽出しているということと，「同じ節点は2個もたない，同じ計算は2度しない」ということにより，冗長性を徹底的に排除していることである．BDD 処理の本質は，ハッシュテーブルによる高速検索と，ポインタによるリスト構造の操作にあると考えられるが，これは，主記憶上の任意のデータに定数時間でアクセス可能であるという，現在の計算機モデルの特長を最大限に利用したものであると言える．

**謝辞** 本稿をまとめるにあたり，ご討論いただいた BDD 特集執筆者のかたがたに感謝いたします．また有益なコメントをいただいた閲読者に感謝いたします．

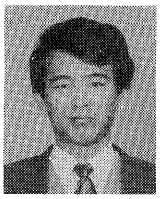
## 参考文献

1) Akers, S. B.: Binary Decision Diagrams, *IEEE Trans. Comput.*, pp. 509-516 (1978).
2) Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. Comput.*, pp. 677-691 (1986).
3) 藤田，藤沢，川戸：2分決定グラフを用いた論理照合アルゴリズムの評価と改良，情報処理学会研究会報告，88-DA-43-2 (1988).
4) Madre, J. C. and Billon, J. P.: Proving Circuit Correctness Using Formal Comparison Between Expected and Extracted Behavior, *ACM/IEEE Proc. 25th DAC*, pp. 205-210 (1988).
5) 湊，石浦，矢島：論理関数の共有二分決定グラフによる表現とその効率的処理手法，情報処理学会論文誌，Vol. 32, No. 1, pp. 77-85 (1991).
6) Lin, B. and Somenzi, F.: Minimization of Symbolic Relations, *IEEE Proc. ICCAD '90*, pp. 88-91 (1990).
7) 藤田，藤沢，松永，角田：2分決定グラフのための変数順決定アルゴリズムとその評価，情報処理学会論文誌，Vol. 31, No. 4, pp. 532-541 (1990).
8) Malik, S., Wang, A. R., Brayton, R. K. and S.-Vincentelli, A.: Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment, *IEEE Proc. ICCAD '88*, pp. 6-9 (1988).
9) Butler, K. M., Ross, D. E., Kapur, R. and Mercer, M. R.: Heuristics to Compute Variable

* 京大工学部矢島研究室では BDD 処理系を公開している．

Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams, *ACM/IEEE Proc. 28th DAC*, pp. 417-420 (1991).
10) 澤田，石浦，矢島：論理関数を表現する二分決定グラフの最小化，信学技報，COMP 91-15, pp. 27-36 (1991).
11) 藤田，松永，角田：大規模回路の多段論理簡単化について，第 41 回情報処理学会全国大会，6-9 (1990).
12) 湊 真一：共有二分決定グラフの「幅」に着目した変数の順序づけ手法，第 42 回情報処理学会全国大会，6-158 (1991).
13) Brace, K. S., Rudell, R. L. and Bryant, R. E.: Efficient Implementation of a BDD Package, *ACM/IEEE Proc. 27th DAC*, pp. 40-45 (1990).
14) 石浦菜岐佐：BDD とは，情報処理，Vol. 34, No. 5, pp. 585-592 (1993)
15) 渡部悦穂，久木元裕治：BDD の応用，情報処理，Vol. 34, No. 5, pp. 600-608 (1993)
16) 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 (1993)
17) 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 (1993)
18) 木村晋二：BDD の並列処理技術，情報処理，Vol. 34, No. 5, pp. 624-630 (1993)

(平成 4 年 12 月 18 日受付)



**湊 真一** (正会員)

1988 年京都大学工学部情報工学科卒業．1990 年同大学院修士課程修了．同年日本電信電話(株)入社．以来，同社 LSI 研究所にて，LSI の論理設計システムの研究開発に従事．特に論理関数の処理手法に興味をもつ．1992 年情報処理学会全国大会奨励賞受賞．電子情報通信学会会員．