Vol.2013-AL-145 No.3
2013/11/6

論理関数のCNFからBDDの効率的な構築法

戸田　貴久[1,a)]

概要：論理関数の CNF が与えられるとき，その論理式を表す二分決定グラフ BDD を構築するための新しい手法を提案する．CNF は従来から用いられている論理関数の伝統的な表現法の一つである．BDD には論理関数を操作するための様々な演算が備わっているので，CNF よりも BDD を用いる方が適切な多くの問題がある．しかし，CNF から BDD の構築は自明な計算ではない．本稿では，まず CNF を表す中間表現を計算し，それから BDD に変換する二段階の BDD 構築法を提案する．さらに，提案法で中間表現として用いられるデータ構造は，BDD 構築だけでなく，有向グラフ上の列挙問題など様々な問題に対して有用であることも示す．

キーワード：二分決定グラフ, BDD, 論理関数, CNF, 乗法標準形, 双対化, 三分決定グラフ，TDD

Efficient Construction of BDDs from CNFs

Abstract: We propose an algorithm to construct the binary decision diagram (BDD) for a CNF formula of a Boolen function. A CNF formula is a usual representation of Boolean functions and has been used for a long time. On the other hand, various operations to manipulate Boolean functions are available in a BDD, and thus there are many problems where using BDDs are more efficient, however converting CNFs to BDDs is not a trivial task. In this paper, we propose a construction algorithm through two stages: we first compute an intermediate representation of a CNF formula, and then construct the corresponding BDD. We furthermore show that the data structure used as an intermediate representation is useful not only in BDD construction, but also in various problems such as enumeration problems on directed graphs.

Keywords: binary decision diagram, BDD, Boolean function, CNF, conjunction normal form, dualization, symbolic computation，ternary decision diagram, TDD

1. はじめに

論理回路の設計や検証，人工知能，組合せ論などにおける多くの問題において論理関数は基本的な概念である．そのような問題を効率的に計算するために，論理関数の効率的な表現と操作が不可欠である．

CNF は従来から用いられている論理関数の伝統的な表現法の一つである．しかし，応用上重要な関数のサイズが指数になることがあったり，充足可能性判定や等価性判定などの基本操作を扱いにくいなどの欠点がある．他の表現法として，論理関数をグラフとして表す二分決定グラフ BDD [1] がある．BDD は多くの問題に対して実用上効率のよい表現として知られており，広く用いられてきた．例えば，いったん論理関数が BDD として表現されるならば，上述の判定を含む様々なクエリに定数あるいは多項式時間で答えることができる．また，論理演算などの基本操作を効率的に実行できるという利点もある．したがって，CNF から BDD への変換は応用上重要な問題であるが，これは自明な計算ではない．

CNF から BDD を構築するおそらくもっとも広く用いられている方法は，CNF の各節に対応する BDD を構築し，それらの論理積を計算するものである．各節の BDD 構築はその節のリテラル数に比例する時間で可能だが，BDD の論理積の最悪計算時間は入力 BDD サイズの積に比例するので，すべての論理積を計算するためには節の数に関して指数的な時間を要するかもしれない．さらに悪いことに，計算途中で無数の中間 BDD が生成されたり，最終的に得

[1] 科学技術振興機構 ERATO 湊離散構造処理系プロジェクト
ERATO MINATO Discrete Structure Manipulation System Project, Japan Science and Technology Agency
[a)] toda.takahisa@gmail.com

られる BDD よりずっと大きいサイズの BDD が現れたりすることがある．これに対して Huang ら [2] は，充足可能性判定の計算で有用な DPLL 手続きを応用した BDD 構築法を提案している．素朴な構築法の方が速く構築できる場合もあるが，多くのデータに対して DPLL に基づく方法で著しい高速化と省メモリ化が達成されたことが報告されている．堀山ら [3] は，一般の CNF ではなく，ホーン CNF と BDD の間の変換を理論的な観点から扱っている．

本稿では，CNF や DNF などを含むさまざまな対象を表現するデータ構造 ZTDD を導入する．ZTDD はもともと論理関数の項集合の表現として提案されているが，本稿では ZTDD を**符号つき集合族**（通常の集合族の拡張概念）のためのデータ構造としてより一般的な観点から捉え直す．さらに，符号つき集合族を操作するためのいくつかの基本アルゴリズムを提案する．それらのアルゴリズムを組合せて，CNF から BDD を構築できることを示す．さらに，有向グラフ上での列挙問題にも応用できることを示すことで，ZTDD および提案アルゴリズムは，CNF だけに限らず，符号つき集合族として特徴づけられうるさまざまな対象に関する計算問題に対して有効であることを示す．

本稿の構成を以下にまとめる．2 節に本稿で必要となる，論理関数に関する基礎的な概念と結果をまとめる．論理関数のためのデータ構造 BDD も導入する．3 節でもう一つのデータ構造 ZTDD を導入し，その有用性について議論する．4 節で BDD と ZTDD に基づくさまざまな基本アルゴリズムを与える．これらを応用して，CNF から BDD を構築する手法を与える．さらに，有向グラフ上の部分グラフの列挙などへも応用できることを示す．5 節で本研究をまとめる．

2. 論理関数とそのデータ表現

提案手法で用いられる，論理関数に関する基礎的な概念を本節でまとめる．さらに，論理関数のためのデータ表現 BDD を導入し，CNF から BDD を構築する既存手法について解説する．

2.1 論理関数に関する基礎概念

用語は基本的に [4] に従う．n 変数の**論理関数**は $f\colon \{0,1\}^n \to \{0,1\}$ の形の関数である．0 と 1 の間の順序は通常の数の順序とする．二値ベクトル $u, v \in \{0,1\}^n$ の間の順序は，すべての成分 u_i, v_i に関して $u_i \le v_i$ のとき $u \le v$ とする．ただし，ベクトルの i 番目の成分を u_i, v_i と表す．n 変数の論理関数 f, g の間の順序は，すべての $u \in \{0,1\}^n$ に関して $f(u) \le g(u)$ のとき $f \le g$ とする．

リテラルは変数あるいはその否定である．変数を**正リテラル**，その否定を**負リテラル**と呼び，両者を区別する．いくつかのリテラルの論理和を**節**，論理積を**項**と呼ぶ．ただし，節や項は互いに否定の関係にあるリテラルを含まず，さらに同一のリテラルを重複して含まないとする．論理関数 f に対して項 t が $t \le f$ を満たすとき，t は f の**内項**という．ここで，t とそれが表す論理関数とを同一視している．t からどのリテラルを除いても f の内項でなくなるとき，t は f の**主項**という．論理関数 f のいくつかの項 $t_1, \ldots, t_n$ の論理和 $\phi = \vee_{1 \le i \le n} t_i$ が f と等しいとき，ϕ を f の *DNF* と呼ぶ．双対的に，f のいくつかの節 $c_1, \ldots, c_m$ の論理積 $\psi = \wedge_{1 \le i \le m} c_i$ が f と等しいとき，ψ を f の *CNF* と呼ぶ．節をリテラルの集合とみなすとき，CNF は集合族をなす．DNF に関しても同様である．本稿では混同の恐れがないとき，CNF や DNF を集合族と同一視する．

2.2 論理関数のデータ表現

二分決定グラフ（略して *BDD*）は論理関数のグラフ表現である [1]．図 1 に，二変数以上が 1 のとき 1 を返す論理関数を表す BDD の例を示す．最上段の節点は**根**と呼ばれる．内部節点は変数の添数と他の節点への二つの有向枝を持つ．$\mathrm{V}(f)$ は内部節点 f に対応する変数の添数を表し，$\mathrm{LO}(f)$ と $\mathrm{HI}(f)$ は f の有向枝が指す節点を表す（それぞれ f の **LO 節点**，**HI 節点**と呼ぶ）．LO 節点への有向枝は **LO 枝**と呼び，点線矢印で表す．同様に，HI 節点への有向枝は **HI 枝**と呼び，実線矢印で表す．二つの終端節点 $\bot$ と $\top$ がある．

以下の二条件を満たすものを二分決定グラフ BDD と呼ぶ．第一に，内部節点 u が v を指すとき，$\mathrm{V}(u) < \mathrm{V}(v)$ が満たされなければならない．第二に，以下のいずれの簡約規則も適用できないという意味で**既約**でなければならない（図 2）．

(1) 内部節点 u の両方の枝が同一の節点 v を指すならば，u を指す全ての枝を v を指すように変え，u を削除する．
(2) もし節点 u と v を根とする部分グラフ同士が等価ならば，部分グラフを共有する．

BDD は以下のように理解されうる．BDD の根から終端節点に至る道は，論理関数の変数への 0,1 割り当てと関数値を表す．道上でラベル k の節点で HI 枝が選ばれるとき k 番目の変数 x_k には 1 が割り当てられ，LO 枝が選ばれるとき x_k には 0 が割り当てられることを意味する．さらに，その道が $\top$ に到達するとき関数値は 1 となり，$\bot$ に到達するとき 0 となる．例えば，図 1 において道 ① ⇢ ② → ③ → $\top$，① ⇢ ② → ③ ⇢ $\bot$，① → ② → $\top$ はそれぞれ $f(0,1,1) = 1$，$f(0,1,0) = 0$，$f(1,1,0) = 1$，$f(1,1,1) = 1$ を意味する．ここで，三番目の道に ③ が現れていないが，節点削除規則により，③ の両方の枝は $\top$ を指すことが分かり，したがって x_3 に 0 を割り当てても 1 を割り当ても関数値は 1 であることが分かる．

変数集合 V および変数順序が固定されているならば，V 上の論理関数は BDD として一意な形をもつ（例えば [5]）．

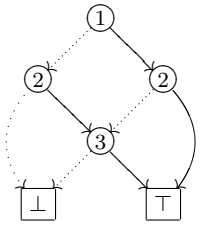
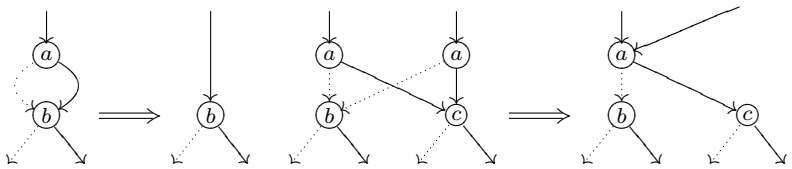


図 1　三変数上の多数決関数を表す BDD
Fig. 1　The BDD for a majority function of three variables



(a) 節点削除規則　(b) 節点共有規則

図 2　BDD の簡約規則
Fig. 2　The reduction rules of BDDs

効率性のため，各 BDD 節点 p はそれが保持する三つ組 $(\mathrm{V}(p), \mathrm{LO}(p), \mathrm{HI}(p))$ に対して一意になるようにハッシュ表で管理されている．すなわち，関数 BDD_UNIQUE は添数，LO 節点，HI 節点の組 (k,l,h) を受け取り，対応する節点がハッシュ表に登録されているときその節点を返す．登録されていないとき，$\mathrm{V}(p)=k$，$\mathrm{LO}(p)=l$，$\mathrm{HI}(p)=h$ を満たす節点 p を作成してハッシュ表に登録して p を返す．これにより等価なグラフを表す異なる節点が作成されなくなる上，グラフの等価性判定を定数時間のうちに行えるようになる．

BDD 上の任意の節点に関して，その節点を根とする部分グラフは BDD である．したがって，混同の恐れがないときには BDD 上の一つの節点とそれを根とする BDD とを同一視する．論理関数 f を表す BDD を $B(f)$ で表す．本稿では $|B(f)|$ は $B(f)$ における内部節点の個数を表し，$B(f)$ の**サイズ**と呼ぶ．

BDD の特徴は，明示的なデータ表現に比べサイズを抑制できること，論理関数に関するさまざまな基本操作を BDD 上で実行できることである．例えば，論理和，論理積，排他的論理和などがある．そのような操作を実現する汎用的な演算として Apply 演算が知られている [1]．Apply 演算は BDD の構造に関して再帰的に実行され，その最悪計算時間は二つの入力 BDD サイズの積に比例する [6]．

2.3　BDD 構築のための既存手法

論理関数の CNF から BDD を構築するための既存手法を解説する（図 3）．一つの節を表す BDD は，その節に現れる変数番号に関して降順にリテラルを見ていき以下の操作を繰り返すことでボトムアップに構築される．現在まで得られた BDD を f とする（最初 $\bot$ で初期化する）．次のリテラルが x_j のとき f を BDD_UNIQUE$(j, f, \top)$ で更新し，次のリテラルが $\neg x_j$ のとき BDD_UNIQUE$(j, \top, f)$ で更新する．明らかに，一つの節を表す BDD の構築にはその節に現れるリテラルの個数に比例する時間を要する．節ごとに BDD を構築し，その後それらの論理積をとることで，CNF を表す BDD を得る．

論理積演算と BDD_UNIQUE は通常の BDD ライブラリにおいて提供されるので，上述の手法を実装するのは容易であるが，その反面いくつかの欠点がある．

- 実際的な効率は論理積を適用する順序に依存する．著者の知る限りにおいて，適切な順序は知られていない．
- 論理積演算を節の個数分だけ繰り返すので，最悪の場合，節数に関して指数的な時間を要するかもしれない．
- 出力 BDD に現れない BDD 節点が計算途中に多数生成されうる．

Huang ら [2] は，充足可能性判定の計算で有用な DPLL 手続きを応用した BDD 構築法を提案している．上述の素朴な構築法の方が速く構築できる場合もあるが，多くのデータに対して DPLL に基づく方法で著しい高速化と省メモリ化が達成されたことが報告されている．

3.　符号つき集合族とそのデータ表現

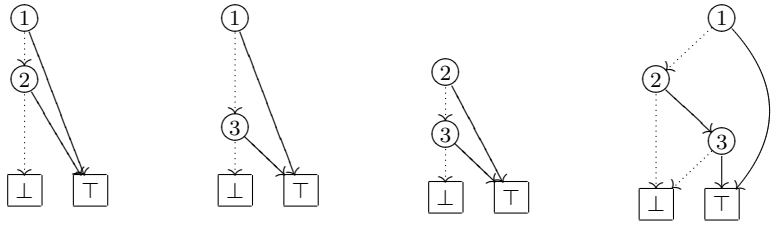
本節では，提案手法で用いられる，もう一つのデータ表現 ZTDD を導入する．ZTDD は論理関数の項集合の表現として提案されているが，本稿では ZTDD を符号つき集合族のためのデータ構造としてより一般的な観点から捉え直し，ZTDD の有用性について議論する．

3.1　符号つき集合族

2.1 節において，CNF（節の集合）や DNF（項の集合）など集合族としての論理関数の表現を導入した．これらの集合族に属する各集合は以下の二条件を満たすものとして特徴づけられる．

(1) 集合は符号つきの要素からなり，各要素は非ゼロ整数とみなされる．
(2) 符号だけ異なる二要素は同一の集合に属さない．

CNF や DNF 以外にも，例えば，有向グラフ上の二節点を結ぶ（単純）道の集合や木の集合など様々な部分グラフの集まりが上の条件を満たす（例えば，有向枝 (i,j) は $i<j$ のとき正，$i>j$ のとき負として符号を定めれば良い）．そのような集合族を扱うたびごとに上の条件を確認するのは煩雑である．本稿では一般性の観点から，これら二条件を満たす集合を**符号つき集合**と呼び，それらの集まりを**符号つき集合族**と呼ぶ．



(a) A (b) B (c) C (d) $A \wedge B$

図 3 $A = x_1 \vee x_2$, $B = x_1 \vee x_3$, $C = x_2 \vee x_3$ のとき対応する BDD．$A \wedge B \wedge C$ を表す BDD は図 1 に示される．

Fig. 3 The BDDs, where $A = x_1 \vee x_2, B = x_1 \vee x_3, C = x_2 \vee x_3$. The BDD for $A \wedge B \wedge C$ is shown in Fig. 1.

3.2 ゼロサプレス型三分決定グラフ

符号つき集合族のためのデータ表現**ゼロサプレス型三分決定グラフ**（略して *ZTDD*）を導入する．ZTDD は項の集合のためのデータ表現として安岡 [7] により提案された．安岡は単に三分決定グラフ TDD と呼んだが，同じ名前で呼ばれる他のデータ表現があるため（例えば [8]），それらと区別するために本稿では ZTDD と呼ぶことにする．

本データ表現は BDD から（より直接的には後述する ZBDD から）派生した表現であり，BDD との相違点は以下の三つである．第一に，各内部節点からは他の三つの節点への有向枝が出ている（図 4）．節点 f に対して $\mathrm{ZERO}(f), \mathrm{NEG}(f), \mathrm{POS}(f)$ はそれぞれ f の三つの有向枝が指す節点を表す（それぞれ f の ZERO **節点**，NEG 節点，POS 節点と呼ぶ）．BDD では ZERO 節点は $\bot$ の別名として使われることがあるが，ここでは $\bot$ を意味しない．ZERO 節点，NEG 節点，POS 節点への有向枝をそれぞれ ZERO **枝**，NEG **枝**，POS **枝**と呼び，点線矢印，破線矢印，実線矢印で表す．節点ラベルは対応する要素の絶対値とする．第二に，ZTDD では以下の簡約規則が採用されている．

(1) 内部節点 u の POS 枝と NEG 枝がともに $\bot$ を指すならば，u を指す全ての枝を u の ZERO 節点を指すように変え，u を削除する（図 5）．
(2) もし節点 u と v を根とする部分グラフ同士が等価ならば，部分グラフを共有する．

節点共有規則は BDD と同じであるが，節点削除規則が異なることに注意されたい．第三に，根から終端節点への道は一つの符号つき集合に対応する（図 4）．すなわち，道上でラベル k の節点で POS 枝が選ばれるとき正の要素 k (> 0) が集合に含まれ，NEG 枝が選ばれるとき負の要素 $-k$ が集合に含まれ，ZERO 枝が選ばれるとき k も $-k$ も集合に含まれない．そして，この道が $\top$ に至るとき，その集合は ZTDD が表す集合族に属し，逆に $\bot$ に至るときその集合は属さない．

BDD と同様に，符号つき集合族は ZTDD として一意な形をもつ．ZTDD 節点はハッシュ表で管理され，関数 ZTDD_UNIQUE は与えられたラベルと三つの子節点をもつ ZTDD 節点を返す．符号つき集合族 $\mathcal{F}$ を表す ZTDD を $Z(\mathcal{F})$ で表す．$|Z(\mathcal{F})|$ は $Z(\mathcal{F})$ における内部節点の個数を表し，$Z(\mathcal{F})$ の**サイズ**と呼ぶ．

3.3 ZBDD の拡張としての ZTDD，その有用性

通常の集合族のデータ表現として**ゼロサプレス型二分決定グラフ**（略して *ZBDD* あるいは *ZDD*）が湊により提案されている [9]．ZBDD もまた BDD から派生したデータ表現である．ZTDD において各節点の NEG 枝は常に $\bot$ を指すものとして忘却し，実質的に POS 枝と ZERO 枝だけを使った場合のデータ表現が ZBDD に相当する．ZBDD は要素が存在するか存在しないかに関して各節点での分岐を表した決定グラフである．一方，ZTDD では負の要素の存在も扱うことができる．負の要素が存在しない集合族を扱う場合は実質的に ZBDD と同じであるから，この意味において ZTDD を ZBDD の拡張とみなすことができる．もちろん，ZBDD においても符号つき集合族を扱うことは可能である．例えば，正の要素 k (> 0) を $2k - 1$，負の要素 $-k$ を $2k$ と符号化すれば良い（例えば [10], pp.83 あるいは [5], pp.277 の練習問題 252）．しかし，この方法では符号だけ異なる二要素は同一の集合に属さないという条件を自然に表現できない．したがって，ZBDD の構造に関する再帰呼び出し（BDD やその派生形において極めて重要な操作）において，節点のラベルの奇偶に応じて計算の振る舞いを変えなければならないという煩雑さが生じる．

ZBDD が様々な組合せ問題の計算やデータマイニングなどへ応用されているのに対して（例えば，[11], [12], [13], [14]），著者の知る限り ZTDD は [7] において論理関数に対する BDD とは別の表現として提案され，VLSI 設計や CAD への応用が示唆されたこと以外に後続の研究はないようである．3.1 節において述べたように，符号つき集合族は CNF や DNF 以外にも有向グラフの部分グラフの集まりとしても現れる．他にもデータマイニングにおいて，ユーザによるアイテム評価に関するデータセット（例えば，

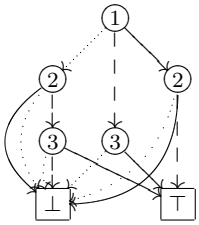


図 4　符号つき集合族 $\{\{1,-2\},\{-1,3\},\{-2,3\}\}$ を表す ZTDD
Fig. 4　ZTDD for $\{\{1,-2\},\{-1,3\},\{-2,3\}\}$

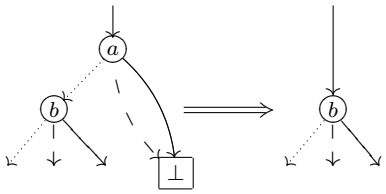


図 5　ZTDD の節点削除規則
Fig. 5　The node elimination rule for ZTDDs

MovieLens データセット）は，各評価を明確に良いあるいは悪い評価とみなせる場合とそれ以外（どちらともとれない評価）の三値に分けるとき，符号つき集合族となる．このように厳密な二値化が困難なデータセット，あるいは不完全なデータや未定義値を含むデータセットなどへの応用が期待される．

4. アルゴリズム

本節では，符号つき集合族から ZTDD を構築するアルゴリズムを与える．そのような ZTDD を受け取り，すべての極大符号つき横断を表す BDD を計算するアルゴリズムも与える．これら二つを組み合わせることで CNF から BDD を構築できることを示す．さらに，論理関数のすべての主項を表す ZTDD を計算するアルゴリズムも与える．以上のアルゴリズムを組み合わせることで，すべての極小符号つき横断を計算できることを示し，この計算の応用例も与える．

4.1 ZTDD の構築アルゴリズム

アルゴリズム 1 では，符号つき集合族 $\mathcal{F}$ が与えられるとき，$\mathcal{F}$ を表す ZTDD を返す再帰関数 COMP が定義されている．ここで，各集合 $S \in \mathcal{F}$ において，絶対値に関して d 番目に大きな要素を d **番目の要素**と呼ぶ．S は符号だけが異なる二要素を同時に含まないので，d 番目の要素は一意に定まる．簡単のため，さらに，各集合 $S \in \mathcal{F}$ は正の無限大 $+\infty$ を要素として含むと仮定する．

アルゴリズム 1 は [15] で提案された ZBDD の構築アルゴリズムと類似の方法に基づいている．各再帰呼び出しにおいて $\mathcal{F}$ を $\mathcal{F}_+, \mathcal{F}_-, \mathcal{F}_0$ に効率的に分割するために，以下の前処理を行う．集合 $S \in \mathcal{F}$ の要素を絶対値に関して昇順に並べ，S を文字列とみなす．$\mathcal{F}$ に対して文字列の整列アルゴリズム（例えば [16]）を適用し，$\mathcal{F}$ を辞書順に整列させる．ただし，符号だけが異なる二要素に関して，正の要素が負の要素より小さいと仮定する．また，$\mathcal{F}$ の各集合（へのポインタ）は配列に格納されているとする．このとき，アルゴリズム 1 における k を d 番目の要素として持つ集合が配列の先頭から連続して位置し，$-k$ を d 番目の要素として持つ集合がその後に続き，それ以外の集合は残りの場所に位置する．したがって，アルゴリズム 1 の各再帰呼び出しにおいて $\mathcal{F}$ を三分割するためには，$\mathcal{F}_-$ の最初の位置と，$\mathcal{F}_0$ の最初の位置を二分探索で発見すれば十分である．

Algorithm 1 符号つき集合族 $\mathcal{F}$ を表す ZTDD を計算する．最初 $d = 1$ で呼び出されなければならない．

```
function COMP(F, d)
    if F = ∅ then
        return ⊥_ZTDD;
    end if
    k ← F の集合で d 番目の要素の絶対値のうち最小値;
    if k = +∞ then
        return ⊤_ZTDD;
    end if
    F_+ ← F の集合で d 番目の要素が k に等しい集合全体;
    F_- ← F の集合で d 番目の要素が −k に等しい集合全体;
    F_0 ← F \ (F_+ ∪ F_-);
    f_+ ← COMP(F_+, d + 1);
    f_- ← COMP(F_-, d + 1);
    f_0 ← COMP(F_0, d);
    f ← ZTDD_UNIQUE(k, f_0, f_-, f_+);
    return f;
end function
```

アルゴリズム 1 の計算時間よりも前処理として行う整列化の方が支配的である．出力 ZTDD はボトムアップに構築され，アルゴリズムの途中で生成される節点は出力 ZTDD に現れるので，必要な記憶量は出力 ZTDD サイズに比例する．

4.2 すべての極大符号つき横断を計算するアルゴリズム

アルゴリズムに進む前に，符号つき横断の概念を導入する．通常の集合族 $\mathcal{F}$ に対する**横断**とは，$\mathcal{F}$ に属するすべての集合と交差する集合である．特に，任意の集合は空集合族 $\emptyset$ の横断とする．本稿では**符号つき横断**を，通常の意味で横断であり，符号つき集合である（符号だけ異なる二つの要素を同時に含まない）ものとして定義する．文脈から明らかなときは単に横断と呼ぶ．包含関係に関して極小な横断を**極小横断**，極大な横断を**極大横断**と呼ぶ．

例 1. 例えば，集合族 $\{\{1,-2\},\{-1,3\},\{-2,3\}\}$ に対して，$\{1,-1,3\}$ は横断であるが符号つき横断ではない．$\{3\}$ や $\{1,-2\}$ などは横断ではない．$\{1,2,3\}$，$\{1,-2,3\}$，$\{-1,-2,3\}$，$\{-1,-2,-3\}$ は極大符号つき横断[*1]である．

[*1] 通常の横断では台集合が最大の横断であるが，符号つき横断では

表 1 $\mathcal{F} = \{\{1,-2\},\{-1,3\},\{-2,3\}\}$ のとき $\chi_{\mathcal{F}}$ の真理値表
Table 1 The truth table for $\chi_{\mathcal{F}}$, where $\mathcal{F} = \{\{1,-2\},\{-1,3\},\{-2,3\}\}$.

x_1	x_2	x_3	$\chi_{\mathcal{F}}$	
0	0	0	1	$\leftarrow \{-1,-2,-3\}$
1	0	0	0	
0	1	0	0	
0	0	1	1	$\leftarrow \{-1,-2,3\}$
1	1	0	0	
1	0	1	1	$\leftarrow \{1,-2,3\}$
0	1	1	0	
1	1	1	1	$\leftarrow \{1,2,3\}$

Algorithm 2 ZTDD f が与えられるとき，すべての極大符号つき横断を表す BDD を計算する．

```
function MAX_TRANS(f)
    if f = ⊤_ZTDD then
        return ⊥_BDD;
    end if
    if f = ⊥_ZTDD then
        return ⊤_BDD;
    end if
    h_- ← MAX_TRANS(NEG(f));
    h_+ ← MAX_TRANS(POS(f));
    t ← BDD_UNIQUE(V(f), h_+, h_-);
    h_0 ← MAX_TRANS(ZERO(f));
    h ← AND(h_0, t);
    return h;
end function
```

$\{1,3\}$，$\{-2,3\}$，$\{-1,-2\}$ は極小符号つき横断である．

符号つき集合族 $\mathcal{F}$ に対するすべての極大符号つき横断を表現するために，極大符号つき横断なら 1 を返し（**受理**し），そうでないなら 0 を返す（**拒否**する）論理関数 $\chi_{\mathcal{F}}$ を定義する（**表 1**）: 各極大符号つき横断 T に対して，$i \in T$ が正の符号をもつとき $x_i := 1$，負の符号をもつとき $x_{|i|} := 0$ とするとき $\chi_{\mathcal{F}}(x_1,\ldots,x_n) := 1$ と定める．関数値が 1 となる 0-1 割り当て以外はすべて関数値を 0 とする．このような論理関数を表す BDD を用いてすべての極大符号つき横断を列挙するためには，BDD の根から $\top$ に至るすべての道をたどれば良い．

アルゴリズム 2 では，符号つき集合族 $\mathcal{F}$ を表す ZTDD $f := Z(\mathcal{F})$ が与えられるとき，$\mathcal{F}$ に対するすべての極大符号つき横断を表す BDD を返す再帰関数 MAX_TRANS が定義されている．関数 AND は，二つの BDD を受け取り，それらの表す論理関数の論理積を表す BDD を返す（2.2 節）．

アルゴリズム 2 の正当性を入力 ZTDD $f = Z(\mathcal{F})$ の構造に関する帰納法で証明する．$f = \top_{\mathrm{ZTDD}}$ のとき，$\mathcal{F} = \{\emptyset\}$ である．空集合に交差できる集合は存在しないので，返り値は $\bot_{\mathrm{BDD}}$ である．一方，$f = \bot_{\mathrm{ZTDD}}$ のとき，$\mathcal{F} = \emptyset$ であり，任意の符号つき集合が横断となる．したがって，返り値は $\top_{\mathrm{BDD}}$ である．よって，f を内部節点とする．T が $\mathcal{F}$ の極大符号つき横断であるための必要十分条件は，以下の二つをともに満たすことである．

(1) T は ZERO(f) の表す集合族の極大横断である．
(2) $\mathrm{V}(f) \in T$ ならば T は NEG(f) の表す集合族の極大横断であり，$-\mathrm{V}(f) \in T$ ならば T は POS(f) の表す集合族の極大横断である．

極大性から，T は $\mathrm{V}(f)$ か $-\mathrm{V}(f)$ のいずれか一方を含まなければならないことに注意されたい．前者に対応する極大横断は，帰納法の仮定より，MAX_TRANS(ZERO(f)) によって表現される．一方，後者に対応する極大横断は，BDD_UNIQUE$(\mathrm{V}(f), h_+, h_-)$ によって表現される．ここで，$h_+ :=$ MAX_TRANS(POS(f))，$h_- :=$ MAX_TRANS(NEG(f)) とする．よって，これら二つの BDD の論理積により，両方の条件を満たす極大横断を表すことができる．以上により，f が内部節点のときも MAX_TRANS が正しい結果を返すことが示された．

以下の定理は，アルゴリズム 2 は再帰呼び出しにおいて生成される中間 BDD のサイズに強く依存することを示している．

定理 1. 入力 ZTDD f に対して，アルゴリズム 2 は $O(|f| \cdot N(f)^2)$ 時間で計算できる．ここで $N(f) := \max\{|\mathrm{MAX\_TRANS}(f')| : f' \text{ は } f \text{ 上の節点}\}$.

証明. 入力 ZTDD 上の各節点（を根とする ZTDD）f' に対して，それを入力としたときの出力 BDD MAX_TRANS(f') をハッシュ表で記憶する．これにより各 f' に対して MAX_TRANS(f') の計算はちょうど一度だけしか行われない．関数 BDD_UNIQUE は定数時間で計算される．$|t| < |h_+| + |h_-| + 1$ から，$O(N(f))$ を得る．AND(h_0, t) の計算時間は入力 ZTDD サイズの積であり，したがって $O(N(f)^2)$ である．ゆえに，MAX_TRANS(f) の計算に要する時間は $O(|f| \cdot N(f)^2)$ である． □

4.3 CNF から BDD の構築アルゴリズム

COMP と MAX_TRANS とを組合わせることで，CNF から BDD を構築できることを示す．$\mathcal{F}$ を CNF の集合族としての表現とする．このとき，$\mathcal{F}$ の極大符号つき横断は，$\mathcal{F}$ のどの節にも現れるリテラルの集まりであり，したがって，すべての節を充足させることのできる変数への 0-1 割り当てに対応する．よって，$\mathcal{F}$ のすべての極大横断を受理する論理関数 $\chi_{\mathcal{F}}$ は，CNF として $\mathcal{F}$ が表現する論理関数に一致する．例えば，表 1 において与えられる $\chi_{\mathcal{F}}$ は，$\mathcal{F} = \{\{1,-2\},\{-1,3\},\{-2,3\}\}$ が CNF として表す論理式 $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$ と等価である．以上から，COMP と MAX_TRANS とを組合せることで，CNF から BDD を構築できることが示された．

そのような最大元は存在せず，一般に複数の極大元が存在する．

Algorithm 3 BDD h が与えられるとき，すべての主項を表す ZTDD を計算する．

```
function PI(h)
    if h = ⊤_BDD then
        return ⊤_ZTDD;
    end if
    if f = ⊥_BDD then
        return ⊥_ZTDD;
    end if
    p0 ← PI(AND(LO(h), HI(h)));
    p− ← DIFF(PI(LO(h)), p0);
    p+ ← DIFF(PI(HI(h)), p0);
    p ← ZTDD_UNIQUE(V(h), p0, p−, p+)
    return p;
end function
```

4.4 論理関数のすべての主項を計算するアルゴリズム

アルゴリズム 3 では，論理関数 f を表す BDD $h := B(f)$ から f のすべての主項を表す ZTDD を返す再帰関数 PI が定義されている．関数 DIFF は，二つの ZTDD を受け取り，第一引数の表す集合族から第二引数の表す集合族の差を表す ZTDD を返す [7].

アルゴリズム 3 は以下の再帰式に基づいている．論理関数 f を変数 x_k に関して $f = x_k \wedge f|_{x_k=1} \vee \neg x_k \wedge f|_{x_k=0}$ と展開[*2]するとき，f のすべての主項の集まり $\mathrm{PRIME}(f)$ は

$$\mathrm{PRIME}(f) = A \cup \{\neg x_k \wedge p \colon p \in \mathrm{B}\} \cup \{x_k \wedge p \colon p \in \mathrm{C}\}$$

を満たす（[5], pp. 278，練習問題 253）．ただし，$A := \mathrm{PRIME}(f|_{x_k=0} \wedge f|_{x_k=1})$, $B = \mathrm{PRIME}(f|_{x_k=0}) \setminus A$, $C = \mathrm{PRIME}(f|_{x_k=1}) \setminus A$ とする．

この再帰式は Coudert と Madre [17] による主項の計算法において用いられた．彼らの方法では，項の集合を Meta-Product と呼ばれる符号化により BDD で表している．すなわち，p を項とするとき，p における各変数 x_i の出現を表す変数 o_i，出現リテラルの正負を表す変数 s_i をもうける．すなわち，$o_i = 0$ ならば x_i も $\neg x_i$ も p には出現しない．$o_i = 1$ かつ $s_i = 1$ ならば x_i が出現する．$o_i = 1$ かつ $s_i = 0$ ならば $\neg x_i$ が出現する．すべての主項を受理する論理関数をこれらの変数を用いて展開できるので，それに従い対応する BDD を構築している．この構築における再帰処理はアルゴリズム 3 と本質的に同じであるが，このようにして得られた BDD は通常の BDD と区別されなければならないので煩雑である．一方，ZTDD では表現されているデータが符号つき集合族であるかぎり，特別な配慮は何も必要ないという利点がある．

4.5 すべての極小符号つき横断を計算するアルゴリズム

以下の定理で示されるように，COMP と MAX_TRANS と PI を組合わせることで，符号つき集合族からすべての極小符号つき横断を表す ZTDD を計算できる．

定理 2. $\mathcal{F}$ を符号つき集合族とする．論理関数 $\chi_{\mathcal{F}}$ の主項は，（集合として見るとき）$\mathcal{F}$ の極小符号つき横断である．逆に，$\mathcal{F}$ の極小符号つき横断は $\chi_{\mathcal{F}}$ の主項である．

証明. p を $\chi_{\mathcal{F}}$ の主項，T を p が表す符号つき集合とする．T がもし横断でないならば，$\mathcal{F}$ の集合 U で T と交差しないものがある．もし T が k (> 0) も $-k$ も持たないならば，U は k と $-k$ のうち高々一つを持つので，U が持っていない方の要素を T に加える．この操作を繰り返すことで，T を含み U と交差しない極大符号つき集合 T' を得る．p は内項なので，T' は $\chi_{\mathcal{F}}$ によって受理されなければならない．よって，T' は極大横断となるが，これは T' が U と交差しないことに矛盾する．ゆえに，T は横断である．T が極小であることは，p が主項であることからただちに導かれる．

逆に，$\mathcal{F}$ の任意の極小符号つき横断 T が（項として見るとき）$\chi_{\mathcal{F}}$ の主項であることは，上と同様の議論で容易に導かれるので証明を省く． □

極小符号つき横断の計算は様々な応用がある．例えば，有向グラフ $G = (V, E)$ 上の固定された二節点 s, t について s から t を結ぶすべての有向道の集まりを $\mathcal{F}$ とするとき，**極小 s-t カットセット**，すなわち G の辺集合でそれらを除去することで s と t を非連結にする極小なものは，$\mathcal{F}$ の極小符号つき横断[*3]に対応する．

5. まとめ

本稿では，CNF や DNF などを含むさまざまな対象を抽象化した概念として符号つき集合族を定義し，符号つき集合族を表現するデータ構造 ZTDD を導入した．ZTDD はもともと論理関数の項集合の表現として提案されているが，本稿では ZTDD をより一般的な観点から捉え直し，その有用性について議論した．符号つき集合族を操作するためのいくつかの基本アルゴリズムを提案した．その応用として，CNF から BDD を構築できることを示した．さらに，有向グラフ上の極小 s-t カットセット列挙に応用できることも示した．

ZTDD および提案アルゴリズムは，CNF だけに限らず，符号つき集合族として特徴づけられうるさまざまな対象に適用可能なので，グラフ列挙やデータマイニングにおける問題への幅広い応用が今後期待される．

[*2] $f|_{x_k=1}$ は f において $x_k = 1$ として得られる論理関数を表し，$f|_{x_k=0}$ は f において $x_k = 0$ として得られる論理関数を表す．

[*3] 極小 s-t カットセットは方向だけが異なる二辺を同時に含まない（従って，符号つき集合である）．なぜなら，極小 s-t カットセット C が (u, v) と (v, u) を含むと仮定せよ．一般性を失うことなく，C のすべての辺を除去した後のグラフで s と u, t と v がそれぞれ同じ連結成分に属すと仮定できる．明らかに $C \setminus \{(v, u)\}$ は依然 s-t カットセットなので C の極小性に反する．

参考文献

[1] Bryant, R.: Graph-Based algorithms for Boolean function manipulation, *IEEE Trans. Comput.*, Vol. 35, pp. 677–691 (1986).
[2] Huang, J. and Darwiche, A.: Using DPLL for efficient OBDD construction, *Proceedings of the 7th international conference on Theory and Applications of Satisfiability Testing*, pp. 157–172 (2005).
[3] Horiyama, T. and Ibaraki, T.: Translation among CNFs, characteristic models and ordered binary decision diagrams, *Inf. Process. Lett.*, Vol. 85, No. 4, pp. 191–198 (2003).
[4] 茨木俊秀：データの論理的解析とブール関数，離散構造とアルゴリズムV（ 藤重悟，編），近代科学社，pp. 147–197 (1998).
[5] Knuth, D.: *The Art of Computer Programming Volume 4a*, Addison-Wesley Professional, New Jersey, USA (2011).
[6] Yoshinaka, R., Kawahara, J., Denzumi, S., Arimura, H. and Minato, S.: Counterexample to the long-standing conjecture on the complexity of BDD binary operations, *Information Processing Letters*, Vol. 112, pp. 636–640 (2012).
[7] Yasuoka, K.: A new method to represent sets of products: ternary decision diagrams, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, Vol. E78-A, pp. 1722–1728 (1995).
[8] Sasao, T.: Ternary Decision Diagrams Survey, *Proc. ISMVL '97*, pp. 241–250 (1997).
[9] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, *30th ACM/IEEE Design Automation Conference (DAC-93)*, Dallas, Texas, USA, pp. 272–277 (1993).
[10] Minato, S.: *Binary Decision Diagrams and Applications for VLSI CAD*, Kluwer Academic Publishers (1996).
[11] Coudert, O.: Solving Graph Optimization Problems with ZBDDs, *the 1997 European Conference on Design and Test*, Paris, France, pp. 224–228 (1997).
[12] Minato, S.: Techniques of BDD/ZDD: Brief History and Recent Activity, *IEICE Transactions on Information and Systems*, Vol. E96-D, No. 7, pp. 1419–1429 (2013).
[13] Toda, T.: Hypergraph Transversal Computation with Binary Decision Diagrams, *Proceedings of 11th International Symposium on Experimental Algorithms (SEA 2013)*, pp. 91–102 (2013).
[14] Minato, S. and Arimura, H.: Frequent closed item set mining based on zero-suppressed BDDs, *Trans. Jpn. Soc. Artif. Intell.*, Vol. 22, pp. 165–172 (2007).
[15] Toda, T.: Fast Compression of Large-scale Hypergraphs for Solving Combinatorial Problems, *Proceedings of Sixteenth International Conference on Discovery Science (DS 2013)*, Singapore, pp. 281–293 (2013).
[16] Bentley, J. and Sedgewick, R.: Fast Algorithms for Sorting and Searching Strings, *Proceedings of 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, pp. 360–369 (1997).
[17] Coudert, O. and Madre, J.-C.: Implicit and incremental computation of primes and essential primes of Boolean functions, *Design Automation Conference, 1992. Proceedings., 29th ACM/IEEE*, pp. 36–39 (1992).