1D5-OS-02b-6in2

変数間の支配関係に基づく論理式の全解列挙手法

Computing All Solutions of Boolean Formulae Using Dominance Relationship between Variables

戸田貴久[*1]　井上武[*2]
Takahisa Toda　Takeru Inoue

[*1]電気通信大学大学院 情報システム学研究科
Graduate School of Information Systems, the University of Electro-Communications

[*2]日本電信電話株式会社 NTT 未来ねっと研究所
NTT Network Innovation Laboratories, NTT Corporation

Computing all solutions of constraint satisfaction problems appears in many areas. Since the computation has a common part, there are advantages to the approach on which we model problems into boolean formulae and solve them with a generic solver. Although this approach can handle a wide variaty of problems from different areas, it is inferior in efficiency to a problem-specific approach. We thus propose an efficient method for computing all satisfying assignments from boolean formulae by exploiting dominance relationship between variables.

1. はじめに

充足可能性問題 *(SAT)* は，命題論理式が与えられるとき，その充足可能性を判定する問題である．SAT を解くソフトウェア（*SAT* **ソルバ**）は，充足可能性だけでなく充足割当も計算する．応用上の重要性から，現実的な時間内に多くの問題例を解くために様々な高速化技法が開発されてきた [鍋島ほか 10]．

AllSAT は，SAT の派生問題の一つであり，すべての充足割当を計算する問題である [長谷川ほか 10]．割当はすべての変数に対する値の割当（**全域割当**）を意味する．未割当の変数の存在を許す立場もあるが，本稿では扱わない．文脈から明らかなときは，充足割当を**充足解**あるいは単に**解**と呼ぶ．

制約充足問題の全解列挙は，モデル検査やデータマイニングなどの応用に現れる．異なる応用ごとに解法を与えるのでなく，共通する計算問題を一手に扱うために，問題を論理式に符号化しその充足解を汎用ソルバ（SAT/AllSAT ソルバ）で計算する手法がある [井上ほか 10, 田村ほか 10]．この解法は汎用的である半面，応用に特化した手法に性能面で劣る．そこで本稿は，論理式の充足解列挙 AllSAT を効率的に解くために，割当に関する論理変数間の支配関係を活用する方法を提案する．変数間の支配関係（従属関係）は以下のように定義される．

定義 1 ϕ を *CNF* とし，ϕ に出現する変数の集合を V とする．変数 $x \in V$ が ϕ において従属とは，変数集合 $S \subseteq V \setminus \{x\}$ が存在して，2 つの任意の充足割当 $\pi, \pi' \colon V \to \{0,1\}$ に対して，S に属するすべての変数への割り当てが一致するならば $\pi(x) = \pi'(x)$ となるときをいう．このとき x は S によって支配されるという．変数が従属でないとき独立という．

例 1 *CNF* ϕ を以下で定義する．

$$(\neg x \vee y) \wedge (\neg x \vee z) \wedge (x \vee \neg y \vee \neg z)$$

これは $x \leftrightarrow y \wedge z$ と論理的に等価なので，変数 y と z への値が定まると，式 ϕ を充足するためには変数 x の値が一意に定まる．それゆえ，x は y と z により支配されている．

連絡先: 電気通信大学大学院情報システム学研究科
〒 182-8585　東京都調布市調布ヶ丘 1-5-1
E-mail: takahisa.toda@is.uec.ac.jp

2. 従来研究

Grumberg らは SAT ソルバを網羅的に解を探索するように変更し，AllSAT 問題を解くソルバを提案した [Grumberg et al. 04]．この方法では，割り当てられた値を知りたい変数（**重要変数**）とそうでない変数（**非重要変数**）に着目して，ソルバは重要変数を優先的に選択しながら[*1]，重要変数の上で網羅的に探索する．矛盾が生じることなくすべての重要変数へ値が割り当てられた後は，非重要変数の上で単解探索を実行する．

Huang らは，CNF から BDD をトップダウンに構築する方法を提案した [Huang et al. 04]．BDD の根節点から 1 終端節点までのパスをたどることで，すべての充足解を列挙できるので，BDD の構築は非明示的にすべての解を計算することに対応する．それゆえ，CNF から BDD を構築するソフトウェア（*BDD* **コンパイラ**）は AllSAT ソルバの一種とみなされる．AllSAT の従来研究ではこのような見方は知られておらず，Toda らにより AllSAT の文脈で明確に位置づけられた [Toda et al. 15b]．本稿では，このような見方に立った BDD コンパイラを *BDD* **コンパイラ型** *AllSAT* **ソルバ**と呼ぶ．

Toda らは，これらのソルバを含む主要な AllSAT ソルバについて多数の問題例を用いた大規模な評価実験を行い，それぞれのソルバの特徴を明らかにした [Toda et al. 15a]．Grumberg らの方法は多数の充足解を持つ問題例にも対応できる上，メモリ消費量が少ないという利点がある．しかし，解を 1 つづつ発見する探索アルゴリズムのため，限られた時間内に発見できる解の個数には限界がある．これに対して，BDD コンパイラ型 AllSAT ソルバは，すべての解を（非明示的にではあるが）表現するので，メモリ消費量が大きい傾向がある．その代わり，動的計画法に似た技法により，等価な計算を複数回行わないので，短時間のうちに莫大な個数の解を発見できるときがある．

BDD コンパイラ型ソルバは，他の方法では到底手に負えないような問題例に対して有効なことがある．しかし，どのような問題例に有効かまだ十分に分かっていなかったり，変数が多くなると性能が低下するなどの課題がある．

[*1] 重要変数の間の変数順は任意のヒューリスティクを利用可能．

3. 準備

本節では BDD および BDD コンパイラの概要を説明する．詳細は文献 [Huang et al. 04, Toda et al. 15a] を参照されたい．SAT の基本的な概念や用語は文献 [井上ほか 10] に従う．SAT ソルバの原理は文献 [鍋島ほか 10] を参照されたい．

3.1 二分決定グラフ：BDD

二分決定グラフ（Binary Decision Diagram; BDD）は，論理関数を有向グラフとして表現するデータ構造である．図 1 の右のグラフが BDD の例である．BDD の非終端節点はラベルを持ち，ちょうど 2 つの有向枝が出る．図において実線で描かれる枝を 1 枝と呼び，点線で描かれる枝を 0 枝と呼ぶ．ちょうど一つの根が存在する．終端節点（子を持たない節点）は，ちょうど 2 つだけ存在し，$\top$ をラベルとして持つ節点を 1 終端節点と呼び，$\bot$ をラベルとして持つ節点を 0 終端節点と呼ぶ．

非終端節点のラベルは論理変数（のインデックス）を表す．非終端節点で 1 枝を選ぶことは対応する変数に値 1 を割り当てることを意味し，0 枝を選ぶことは値 0 を割り当てることを意味する．根から 1 終端節点へのパスは，BDD が表現する論理関数に対する充足割当に対応する．

図 1 の左のグラフは，二分木で論理関数を表している．二分木による表現では，変数が増加するにつれ，指数的に節点数が増加する．BDD は，冗長な節点をスキップしたり（図 2），等価な節点を共有することで（図 3），同一の論理関数をコンパクトに表現する．

3.2 BDD コンパイラ

BDD コンパイラは，CNF を受け取り，BDD を構築する[*2]．この BDD は，CNF のすべての充足割当を 1 終端節点までのパスとして表現する．ここでは，Huang らが提案した BDD のトップダウン構築アルゴリズムの概要を説明する [Huang et al. 04]．

入力 CNF を ϕ と表す．式 ϕ に出現する変数は $x_1, \ldots, x_n$ とし，この順番で変数を選択する．BDD 構築アルゴリズムの基本的な処理は，変数順に従い x_1 から順に値を決めていく．現在，x_1 から x_i まで値が割り当てられたとする．このとき処理は 3 つの場合に分かれる：充足不可能な節が生じる場合，すべての変数への値が割り当てられた場合，それ以外の場合である．

一番目の場合では，それ以降の変数の値をどのように決めても ϕ は充足不可能である．したがって，直近に決定された変数の値をキャンセルし，もう一方の値を割り当てる（バックトラック）．もしキャンセルすべき変数がなければ処理を停止する．二番目の場合では，各変数に値を割り当てる際に充足不可能な節が生じなかったので，現在の割当は ϕ を充足する．ゆえに，現在の割当に対応するパスを BDD に追加する．もし初めて発見された解ならば，1 本のパスだけからなる BDD を作る．その後バックトラックする．三番目の場合では，現在の部分割当が充足可能とも不可能とも決定されていない．ここでナイーブな方法を採用するならば，次の変数 x_{i+1} の値を決定することになる．しかし，この方法では節点共有が一度も生じないので，構築されるグラフは図 1 の左に描かれるような二分木となる．Huang らのアルゴリズムの鍵は，三番目の処理において動的計画法に似たメカニズムを取り入れることで共有可能な節点を発見し，現在までの割当に対応するパスを BDD に追加することである．

以下，三番目の処理をより詳細に述べる．　現在の部分割当を π_i: $\{x_1, \ldots, x_i\} \to \{0,1\}$ とする．　以下で定義される節集合を i 番目の**カットセット**と呼ぶ：$\mathrm{cut}_i := \{C \in \phi \mid \exists l, l' \in C,\ \mathrm{Id}(l) \le i < \mathrm{Id}(l')\}$．ここで，CNF ϕ を節集合とみなし，節 C をリテラル集合とみなしている．さらに，リテラル l を構成する変数の順番を $\mathrm{Id}(l)$ で表す．

ここで重要な点は，カットセット cut_i に属する各節が π_i の下で充足されているか未充足かに関する状態（**カットセットの状態**あるいは単に**計算状態**と呼ぶ；各ビットが該当する節の充足・未充足を表すように表現されたビット列）が，過去に発見された充足割当 π' の計算状態と一致するときである．もし一致するならば，（すでに BDD に表現されているはずの）π' に対応するパス上の x_{i+1} の BDD 節点において，π_i に対応するパスを連結する（x_{i+1} の節点を共有する）．これは，カットセットの状態を見るだけで節点共有の判定が可能であることを意味している．

Huang らのアルゴリズムが三番目の処理において行うことは以下のようにまとめられる．現在の計算状態を算出し，同一の計算状態が登録されているか調べる; もし登録されているならば，π_i のパスをターゲットの節点に連結させ，バックトラックする; 登録されていないならば登録し，次の変数 x_{i+1} の値を決定する．

別の共有判定法として，Huang らは以下で定義される変数集合（i 番目の**セパレータ**）に着目する方法も提案している：$\mathrm{sep}_i := \{x_j \in V \mid j \le i, \exists C \in \mathrm{cut}_i, x_j \in C or \neg x_j \in C\}$．この方法では，$\mathrm{sep}_i$ に属する各変数の値を計算状態とする．

Huang らのトップダウン構築アルゴリズムは，現代的な SAT ソルバの基礎をなす探索アルゴリズム DPLL と矛盾しないので，SAT ソルバのコードに上述の共有判定メカニズムや BDD 構築処理を埋め込むことで実装できる．これにより SAT ソルバの強力な枝刈り能力や単位伝搬などの効率的な実装をそのまま享受できるようになる．また，BDD が一旦構築されると，根から 1 終端までのパスを走査することで全ての充足割当を計算できる．それゆえ，BDD コンパイラを用いて AllSAT 問題を解くことができる．

4. 提案手法

本節では，最初に，変数従属関係を活用する AllSAT 解法の一般的枠組みを与える．その後，実現例として，従属関係の抽出法，変数順序の決定法，BDD コンパイラ型 AllSAT ソルバの内部における従属関係の活用法を順に与える．

4.1 一般的枠組み

本稿で提案する AllSAT 解法の一般的な枠組みは以下の通りである．以下単にソルバというとき AllSAT 問題を解くプログラムを意味する．

1. 解きたい問題を CNF に符号化する際，変数同士の従属関係に関する情報を何らかの方法で抽出する．
2. CNF と変数従属関係をソルバに渡して，ソルバの内部処理において従属関係を活用することで問題を解く．

ステップ 1 は基本的に個々の問題によって処理が異なるが，本稿では問題に依存しない抽出法の一例を後で与える．この方法では，広く用いられている CNF 符号化法の適用過程で従属関係を抽出する．その他にも，例えば，問題ごとの固有の知識に基づき容易に従属関係が決定されたり[*3]，各種のヒューリスティックスを用いて計算されることなどが考えられる．

*2　ここで構築される BDD は簡約規則に関して既約でない．本稿で単に BDD と呼ぶとき既約であることを仮定しない．

*3　モデル検査では，現在状態の変数と次状態の変数との関係が $\bigwedge_i v'_i \leftrightarrow \psi_i(s)$ のように記述される [Abdulla et al. 00]．これは一種の従属関係を表してる．

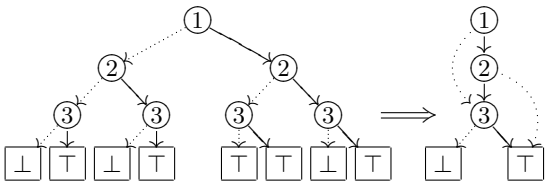

図 1: 同一の論理関数を表す二分木（左）と BDD（右）

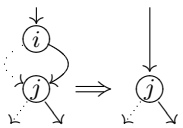

図 2: 節点削除

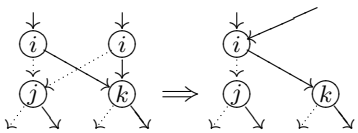

図 3: 節点共有

抽出された従属関係を DIMACS CNF ファイルのコメント行に埋め込むことにより，CNF と一体で管理できる．

ステップ 2で用いるソルバの種類は限定されないが，本稿では BDD コンパイラ型ソルバに対する適用法を後で与える．

4.2 従属関係の抽出

様々な問題を SAT ソルバに帰着して解くアプローチにおいて，問題を命題論理式として一旦モデル化し，その後，CNF に符号化することが一般的である [田村ほか 10, Prestwich 09]．このアプローチにおいて，命題論理式を CNF に変換するために *Tseitin* **変換**は幅広く用いられている [Tseitin 83]．

Tseitin 変換では，入力論理式を構成する各部分論理式に対して新しい変数が導入される．そして，部分論理式と対応する変数との関係を表す節が生成される．構成から，生成された節からなる CNF は元の論理式と充足同値である．

本稿では Tseitin 変換において以下の従属関係に着目する．部分論理式 ψ に対して変数 x が導入されるとせよ．このとき両者の関係は $\psi \leftrightarrow x$ と記述される．簡単のため $\psi \equiv \psi_1 \wedge \psi_2$ とする．ここで ψ_1 と ψ_2 は部分論理式であるので，別の変数 x_1 と x_2 がそれぞれ対応する．ただし部分論理式自体が変数のときは，その変数が対応する．明らかに x_1 と x_2 によって x は支配されるので，この関係を抽出する．その他の論理演算子の場合も同様の議論が当てはまるので説明を省略する．

変数 x_1 と x_2 によって x が支配される事実は，SAT ソルバ（したがって，SAT ソルバを基礎にする AllSAT ソルバ）においてそれほど特別な仕組みを導入しなくても活用されうる．事実，x を x_1 と x_2 の後に選択するならば，x_1 と x_2 に値が割り当てられるとき，論理式 $x_1 \wedge x_2 \leftrightarrow x$ に対応する節により単位伝搬が生じ，x の値が含意される．

4.3 変数順序の決定

抽出された従属関係から，BDD コンパイラ型ソルバに適した変数順序を決定する方法を与える．変数順序の決定は，ソルバの前処理として行われることを想定している．

前述の通り，従属変数はそれを支配する変数より後に配置すべきであるが，カットセットのサイズを抑えるために従属変数を含む節が横長にならないようにしたい．カットセットのサイズが増えると，計算状態の算出コスト増大や共有判定の精度低下などが引き起こされるからである．そこで本稿では，各従属変数はその値を支配する全ての変数が現れた後，すみやかにその従続変数を選択する変数順としたい．図 4 の右の CNF にそのような変数順に変更した CNF を示す．

各従属変数に対して，その値を支配する全ての変数が現れた後に，その従属変数を配置することは必ずしもできないことに注意されたい．というのは，支配関係が循環する場合，例えば x_1 は x_3 を支配し，x_2 は x_1 を支配し，x_3 は x_2 を支配する場合，いづれかの変数はそれを支配する変数の前方に配置せざるを得ないからである．したがって，例えば x_1 と x_3 の間の従属関係だけを採用するなどのように妥協しなければならない．

4.4 ソルバ内部における従属関係の活用

BDD コンパイラ型ソルバにおいて，変数従属関係を用いる効率化手法を与える．BDD および BDD コンパイラ型ソルバに関する詳細は 3. 節を参照されたい．

基本的なアイディアは，従属変数を独立変数と同等に扱わないで，独立変数だけを明示的に処理することである．このために共有判定に関わる処理と BDD 構築に関する処理を効率化する．

最初に，共有判定に関わる処理について述べる．探索の各ステップで共有判定を行うのは計算コストが高い上，多数の計算状態を登録することになりメモリ効率も悪い．そのため，Toda らは，共有の候補となる BDD 節点は決定変数[*4]に対応する節点のみに限定して，共有判定に伴う処理を効率化した [Toda et al. 15b]．具体的には以下の通りである．現在，変数 x_1 から x_i までの値が割当られたとする．変数 x_{i+1} が決定変数でなければ，単位伝搬によりその値は決定されるので，そのような変数の値はすべて単位伝搬で求める．これにより，次に選択する未割当の変数は決定変数となる．したがって，直後の処理において従来の BDD コンパイラと同じ処理を行うだけで，共有されるのは決定変数に対応する節点だけに限定される．

本稿では，従属変数を明示的に扱わないようにするために Toda らの方法 [Toda et al. 15b] を適用する．今回の研究独自の利点は 2 つある．まず，従属変数は含意変数でもあるので，従属変数は共有の候補となる節点には決してならない．さらに，決定変数はかならず独立変数なので，共有判定が行われるのは独立変数の直前である．変数従属関係は，探索処理がはじまる前に分かるので，共有判定が行われうる場所をあらかじめ特定できる．したがって，共有判定のために必要なリソースを限定できる．例えば、探索過程で計算状態を効率的に算出するために，通常，すべてのレベルのカットセットをあらかじめ計算する．しかし，変数従属関係が分かっているならば，決定変数の直前の変数におけるカットセットだけ計算しておけば良い．

次に，BDD 構築に関する処理について述べる．上で述べたように，共有される節点の候補は独立変数に対応する節点である．ゆえに，構築する BDD において従属変数に対応する節点を明示的に表現する理由はない．したがって，BDD コンパイラが充足割当を発見し，対応するパスを BDD に追加するときは，従属変数に対応する BDD 節点を無視し，独立変数に対応する BDD 節点だけからなるパスを追加する．このようにして構築された BDD から充足割当を求めるときには，スキップされている変数があれば，節点削除規則により削除された節点であるか，従属変数に対応する節点のいづれかであると分かる．これらのうちどちらであるかは，スキップされた変数インデックスを見ればただちに判別できる（従属変数のインデックスは

*4 SAT ソルバの単位伝搬によって値が含意されなかった変数を**決定変数**と呼ぶ．決定変数に割り当てる値に関して探索処理が分岐する．

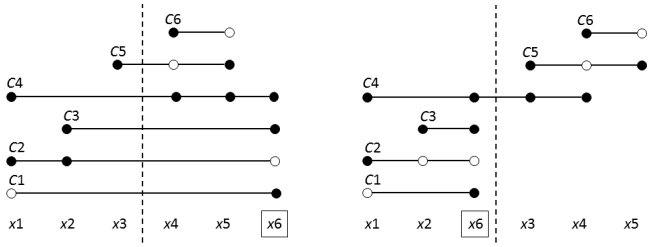

図 4: 変数順変更前（左）と変更後（右）の CNF：区間は節，黒点と白点は正と負のリテラルを表す．変数 x_6 は x_1 と x_2 に支配されている．

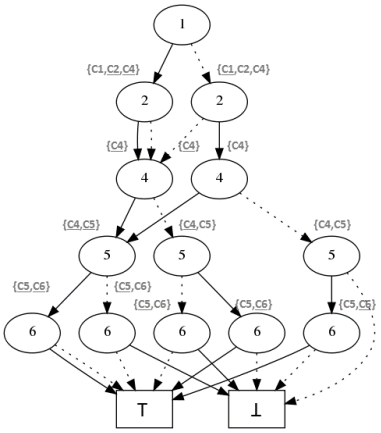

図 5: 提案手法を用いて構築された BDD. 節点ラベルは変数の順番を意味する．各枝にカットセットの状態を付している．下線は充足を意味する．

あらかじめ分かっているから）．さらに，従属変数の値は，入力 CNF と独立変数の値から容易に復元できる．

例 2 図 *4* の左の *CNF* において，変数 x_6 は x_1 と x_2 によって支配されている．図 *5* は，この *CNF* と従属関係から，提案手法に従い構築した *BDD* を示す．*BDD* の各枝には，共有判定のために算出したカットセットの状態を付している．下線はカットセットに属する節が現在の部分割当の下で充足されていることを意味する．ラベル 3 の *BDD* 節点がないのは，変数順序決定後 3 番目の変数，すなわち x_6 が従属変数なので，明示的に表現されていないからである．

左側のラベル 4 の *BDD* 節点ではパスが合流している．これは以下のような処理が行われた結果である．変数 x_1, x_2 の値を，たとえば $x_1 \mapsto 1, x_2 \mapsto 0$ と割り当てた後，x_3 の値が単位伝搬により決定される．現在のところ *CNF* は充足不可能でなく，未割当の変数が残されているので，共有判定の処理に進む．3 番のカットセット cut_3 の状態 $\{\underline{C_4}\}$ を求め過去の登録を参照したところ，同一の状態が存在したので，すでに存在するラベル 4 の節点に現在の割当のパス ① → ② ⇢ をつないで *BDD* を拡大する．

5. まとめ

本稿では，割当に関する変数間の支配関係に着目して，All-SAT 計算を効率化する手法を提案した．AllSAT を解くソルバにはいくつかの種類があるが，最近その計算能力の高さが評価されている BDD コンパイラ型ソルバを扱った．従属関係を活用する AllSAT 解法の一般的な枠組みを提案した．その実現例として、Tseitin 変換の適用時に従属関係を抽出する方法，従属関係を用いて変数順を決定する方法，BDD コンパイラ型ソルバ内部で従属関係を活用する方法を与えた．

参考文献

[井上ほか 10] 井上克己，田村直之: SAT ソルバーの基礎, 人工知能学会会誌 25 巻 1 号（2010 年）．

[鍋島ほか 10] 鍋島英知，宋剛秀: 高速 SAT ソルバーの原理, 人工知能学会会誌 25 巻 1 号（2010 年）．

[田村ほか 10] 田村直之，丹生智也，番原睦則: 制約最適化問題と SAT 符号化, 人工知能学会会誌 25 巻 1 号（2010 年）．

[長谷川ほか 10] 長谷川隆三，藤田博，越村三幸: モデル列挙とモデル計数, 人工知能学会会誌 25 巻 1 号（2010 年）．

[Tseitin 83] G. Tseitin: On the complexity of derivation in propositional calculus. Automation of Reasoning: Classical Papers in Computational Logic, 2:466-483, 1983. Springer-Verlag.

[Prestwich 09] S. Prestwich: Handbook of Satisfiability, chapter CNF Encodings. IOS Press, 2009.

[Grumberg et al. 04] O. Grumberg and A. Schuster and A. Yadgar: Memory Efficient All-Solutions SAT Solver and Its Application for Reachability Analysis, Formal Methods in Computer-Aided Design, LNCS, Vol. 3312, pp.275-289, 2004.

[Huang et al. 04] J. Huang and A. Darwiche: Using DPLL for efficient OBDD construction, In Proceedings of the 7th international conference on Theory and Applications of Satisfiability Testing, pages 157–172, 2005.

[Toda et al. 15a] T. Toda and S. Takehide: Implementing Efficient All Solutions SAT Solvers, CoRR, abs/1510.00523, 2015.

[Toda et al. 15b] T. Toda, K. Tsuda: BDD Construction for All Solutions SAT and Efficient Caching Mechanism, the 30th Annual ACM Symposium on Applied Computing, 2015.

[Abdulla et al. 00] P.A. Abdulla, P. Bjesse and N. Eén: Symbolic Reachability Analysis Based on SAT-Solvers, TACS/ETAPS 2000, LNCS 1785, pp.411-425, 2000.