積和型論理式の因数分解処理による多段合成

5B－7

舛水真樹*　川原友之　正力康也　陳恵茹
凌暁萍　後藤公雄

神奈川工科大学

１．はじめに

本報告は，多出力論理関数が与えられたとき，そのカーネルより構成されるＢＤＤ（２分決定グラフ）を作成し，これを基に，この関数の実現する多段論理回路の最適化を行うことを目標としている．

２．基本アルゴリズム

本報告のアルゴリズムについて述べる．なお，ここでは肯定リテラルをａ，否定リテラルをａ’で表現している．

２．１．入力関数の再構成

後に説明するカーネルの生成及びその比較の作業を容易にする目的で各入力関数の再構成を行う．そのアルゴリズムについて以下に述べる．

まず図１に示すようなＢＤＤを作成する．なお，各アークについている文字は展開に用いた変数を表している．ＢＤＤを作成するためのアルゴリズムを以下に示す．

［ステップ　１］　ＢＤＤのルート（根）に展開する論理関数を与える
［ステップ　２］　ステップ１で与えられた論理関数を任意の変数に０または１を代入することによって展開する．
［ステップ　３］　ステップ１で展開されたノードを任意の変数で同じように展開する．

次に図１のＢＤＤの終端１よりルートに向かい遡行を行う．その際，各アークの展開に用いた変数より積項を作成する．これを全ての終端１について行うことにより積和型論理関数が得られる．

以上の手法で入力関数の再構成を行う．ただし，展開に用いる変数の順序は各入力関数とも共通でなければならない．ここで論理関数の再構成を行う目的は，各入力関数を統一した形式に整えることであり，最適化ではない．そのため，再構成された関数を元の関数と比較した場合，リテラル数やゲート数が増加している可能性もあり得る．最後に入力論理関数の再構成の有用性を示す例を以下に挙げる．

次に示す２つの論理関数を比較する場合について考える．

f=abc'd'+abd,　(1)
g=bc'd'+abc'+abcd.　(2)

これらの関数をＢＤＤにより再構成すると式（１）及び式（２）はそれぞれ式（３）及び式（４）となる（展開に用いた変数の順序はａ→ｂ→ｃ…とする）．

F=abc'+abcd,　(3)
G=a'bc'd'+abc'+abcd.　(4)

式（３）及び（４）の各積項に注目すると式（３）が式（４）に吸収されることが容易に判別できる．この特徴は論理関数同士を比較するときだけでなく，論理関数の部分集合であるカーネルを比較する際にも非常に役立つ．

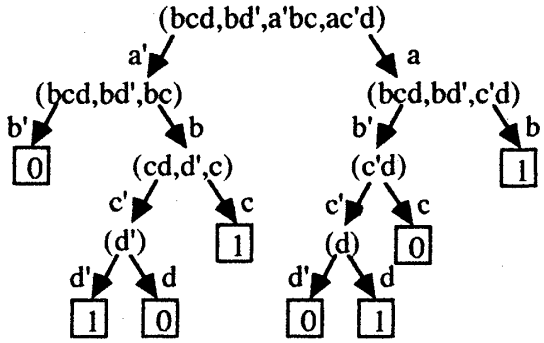

図１　論理関数再構成用ＢＤＤ

２．２　入力関数の多段合成

本報告では多段合成を行うため図２に示すようなＢＤＤを用いている．ＢＤＤ生成のアルゴリズムを以下に示す．

Multilevel Logic Synthesis of Sum-of-Products Logic Function by Factorization Processing
Masaki MASUMIZU
Tomoyuki KAWAHARA
Yasunari SHORIKI
Huei-ru CHEN
Xiao-ping LING
Kimio GOTO
Kanagawa Institute of Technology
1030 Shimo ogino, Atsugi City, Japan.

［ステップ　1］　ＢＤＤのルート（根）に展開する論理関数を与える．展開する論理関数がキューブフリーでない場合，共通因数による除算を行いカーネルを抽出する．
［ステップ　2］　カーネルを任意の展開変数に０または１を代入することによって展開する．
［ステップ　3］　展開された論理関数がキューブフリーでない場合，共通因数による除算を行いカーネルを抽出する．
［ステップ　4］　ステップ１で展開されたノードを任意の展開変数で同じように展開する．

以上の操作を論理関数が単項式になる直前まで繰り返すと，カーネル及びそのコ．カーネルをノードの内容に持つＢＤＤが完成する．次に式（５）をもとにしたＢＤＤの作成例について述べる．

$$f=a'bd+ab'c'+abc+abc'd \qquad (5)$$

式（５）自身は図２のグラフの０ａのノードに格納される．次に式（５）をａ，ｂおよびｄで展開すると式（６）よりノード１ａにｂ’ｃ’＋ｂｃ＋ｂｃ’ｄ，ノード１ｂにａ’ｄ＋ａｃ＋ａｃ’ｄ，さらにノード１ｃにａ’ｂ＋ａｂ’ｃ’＋ａｂｃ＋ａｂｃ’が記憶される．なお，式（６）及び図２に示されているｒは剰余を表している．式（５）をａ，ｂおよびｄで展開したときのあまりは式（５）をａ’ｂ’ｄ’で展開した結果に等しい．

$$f=a(b'c'+bc+bc'd)+b(a'd+ac+ac'd)+d(a'b+ab'c'+abc+abc')+r \qquad (6)$$

次に，生成されたカーネルの比較を行い共通のカーネルの存在を調べる必要があるが，この作業はカーネルの生成と同時に行うべきである．なぜなら，共通のカーネルが発見された際にはそれ以降のカーネルの生成を行わずグラフのアークを修正すればよいからである．

また，全てのカーネルを生成しようとした場合には演算時間の増大が予想されるのでので，本方式ではカーネルの評価を行い，評価の低いカーネルに関しては，それ以上のカーネル生成を行わないものした．評価基準はそのカーネルのリテラル数とし，リテラル数の少ないカーネルは切り捨てる．

最後に入力関数が多出力の場合のＢＤＤの構造であるが，ここでは図３に示すように各論理関数のルートのノード同士をアーク０により接続し１つの大きなＢＤＤとして扱っている．

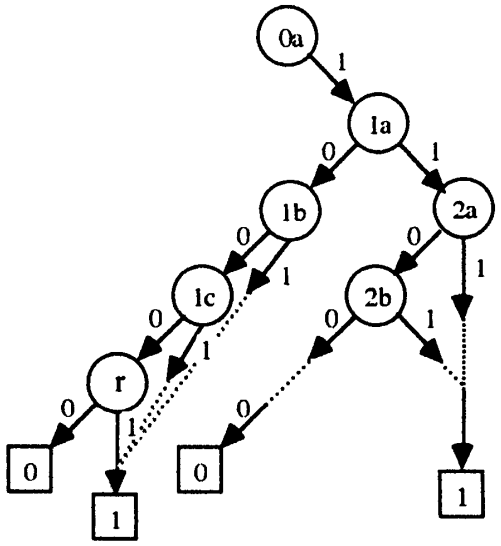

図２　多段合成用ＢＤＤ

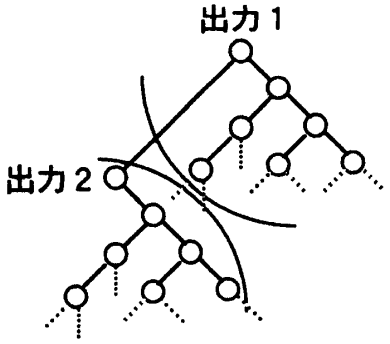

図３　多段合成用ＢＤＤ（２出力関数の場合）

３．まとめ

以上，ここで述べた方式の最大の利点は，生成された全てのカーネルが意味付けされた形で保存されているのでカーネルの集合から多段論理形式に再構成する際にはほとんど手間がかからないことである．しかし，冗長なカーネルをどのように検出するかについての考慮がなされていないなどの問題点も残されている．

参考文献

1) K.Goto, T.Ito, H.Tatsumi, and X.-P. Ling: Generation of All Prime Implicates by Using BDD (Binary Decision Diagrams), Proceedings of the JTC-CSCC'94, Vol2 of 2, pp.667-672, 1994,7.
2) R.K. Brayton: MIS: A Multiple-Level Logic Optimization System, IEEE Transactions Computer-Aided Design, Vol. CAD-6, No 6, 1987,11.