論理関数主項の複数最小被覆の一導出法

後　藤　公　雄*

A Method for Deriving the Multiple Coverings of Prime Implicants in the Logic Function

Kimio GOTO

Abstract

In this paper is described a method for generating the multiple minimal coverings of prime implicants. An algorithm is proposed in which the Petrick function as the product of sum between several prime implicants is expanded by applying the Shannon's theorem to the binary tree. Using this algorithm, the COMMON LISP language program was made, and run on the LISP oriented machine, ELIS. As a result of this run, the maximum number of the variable for which the multiple minimal coverings of prime implicants could be generated, was seven within the range of allowable computer operating time of 20 sec and beyond the range of allowable Karnaugh map density of 40%.

1. 緒　　言

従来，論理関数の最小化について，主項の生成とその主項の最小被覆の生成の2面から種々の検討が行われてきた。筆者も主項の生成について種々の検討を行ってきた[1]。特に最小項隣接グループ生成法，最小項番号による3分枝展開，さらに分割法および多段分割法は筆者らの研究対象となっている。一方，主項の最小被覆の生成に関してはそれ程多くの研究が行われているとは言えない。これは，主項の最小被覆として複数解が得られるのが普通であり，そのため計算量が膨大になることに起因していると考えられる。しかしながら，単一最小被覆の生成については，述べている文献[2]がある。その方法は主項表で列方向のみではなく，行方向も条件に応じて削除するようにし，最終的に単一解が得られるよう不要な主項を削除する方法によっている。

筆者は，最初からこのような削除法による単一最小被覆生成の研究に入ることを避け，削除することなく複数全最小被覆を求める場合，計算機によって果してどの程度まで演算可能であるか確認し，解の導出の難易度を調べることとした。そこで複数最小被覆生成のためペトリック関数の2進木展開によるアルゴリズムを考案し，これをCOMMON LISP言語プログラムで実現することとした。なお，主項の導出には多変数，高濃度に耐性のある多段分割法と3分枝展開法を併用することとした。次にその詳細について述べる。

平成2年9月28日受理

* 情報工学科

2. ペトリック関数による最小被覆

本章ではペトリック関数[3]を用いて主項の複数最小被覆を生成する方法について述べる。

2.1　ペトリック関数の作成

ある論理関数 f が最小項の論理和として与えられた時，何らかの方法によって考えられるすべての主項が求まったものとする。このような主項にマーク付け（または番号付け）をすると，その主項のマークと主項の含む最小項の番号とを対応ずけることができる。このような主項と最小項との包含関係が，各主項について同時に成立せねばならないことより，ペトリック関数 P_s が導かれる。

その一例を次の4変数の論理関数

$$f=\Sigma(0,\ 1,\ 2,\ 6,\ 7,\ 8,\ 10,\ 11,\ 15) \qquad (1)$$

を用いて説明する。式(1)について主項を求めると，マーク付けして次のように表わされる。

$$\left.\begin{aligned} &a=\{10,\ 11\},\ b=\{0,\ 1\},\ c=\{2,\ 6\},\\ &d=\{6,\ 7\},\ e=\{11,\ 15\},\ f=\{7,\ 15\},\\ &g=\{0,\ 2,\ 8,\ 10\} \end{aligned}\right\} \qquad (2)$$

ここで，主項マークと最小項の包含関係を考えてみる。例えば最小項（最小項番号が i であるような最小項を略して最小項 i と呼ぶこととする）は主項 b（主項マークが b である主項を略して主項 b と呼ぶこととする）または主項 g に含まれている。この関係は，

$$\{0\}=b+g \qquad (3)$$

のように表わされる。他の最小項についても同様にして

$$\left.\begin{aligned} &\{1\}=b,\ \{2\}=c+g,\ \{6\}=c+d,\ \{7\}=d+f,\\ &\{8\}=g,\ \{10\}=a+g,\ \{11\}=a+e,\ \{15\}=e+f \end{aligned}\right\} \qquad (4)$$

が成立する。式(3)および(4)の関係は同時に成立せねばならないから，式(3)と式(4)の各式の論理積をペトリック関数 P_s で表して

$$\begin{aligned} P_s&=\{0\}\cdot\{1\}\cdot\{2\}\cdot\{6\}\cdot\{7\}\cdot\{8\}\cdot\{10\}\cdot\{11\}\cdot\{15\}\\ &=(b+g)\cdot b\cdot(c+g)\cdot(c+d)\cdot(d+f)\\ &\quad\cdot g\cdot(a+g)\cdot(a+e)\cdot(e+f) \end{aligned} \qquad (5)$$

のようになる。この式の右辺に吸収律を適用して，次式が得られる。

$$P_s=b(c+d)(d+f)g(a+e)(e+f) \qquad (6)$$

2.2　ペトリック関数の展開

前節で作成したペトリック関数 P_s は和積形式になっている。これを積和形式に展開したとき，一つの積項の含む全リテラル（主項マーク）により与えられた関数 f を完全に被覆する全主項が得られる。したがってこのような積項のリテラルの最も少ないものを求めれば最小被覆が得られる。

このような展開に当り，リテラル x について Shannon の展開[4]を行うと，

$$P_s=x[P_s]_{x=1}+[P_s]_{x=0} \qquad (7)$$

のようになる。式(7)を式(6)に適用すると，

$$\begin{aligned} P_s&=a[P_s]_{a=1}+[P_s]_{a=0}\\ &=a\{b(c+d)(d+f)g(e+f)\}\\ &\quad+b(c+d)(d+f)ge(e+f)\\ &=abg(c+d)(d+f)(e+f)\\ &\quad+beg(c+d)(d+f) \end{aligned} \qquad (8)$$

が得られる。

ここで式(6)を展開するのに a を用いたが，式(6)で最も発生頻度の高いラテラル d を用いると，

$$\begin{aligned} P_s&=d[P_s]_{d=1}+[P_s]_{d=0}\\ &=dbg(a+e)(e+f)\\ &\quad+bcfg(a+e)(e+f) \end{aligned} \qquad (9)$$

が得られる。式(9)の第2項は式(8)の第1項と第2項に比し，より展開の度合が進んでいることを示している。これは発生頻度の高いリテラルを用いた方が展開が速く行われることを示している。次に式(9)の第1項と第2項にたいし，それぞれ発生頻度の高いリテラルを選び，同様な Shannon の展開を行う。

このような Shannon の展開には2進木を使うと便利である。この2進木展開法は次のようにして行う。先ずペトリック関数 P_s のリテラル，すなわち主項マークの中で，使用頻度が最大のものを max で表す。P_s の和項要素の中で max を含むものを除去した残りの和項要素（勿論単項要素も含む）の積，すなわち式(9)の第1項に相当するものを P_{ts} とし，これを2進木展開の左の処理とする。次いで P_s の和項要素で max を含むものの中の max 自身を除去して得られた和項と，単項を含む残りの和項の積，すなわち式(9)の第2項に相当するものを P_{rs} とし，これを2進木展開の右処理とする。さらに左に得られた P_{ts} と，右に得られた P_{rs} について同様な処理を繰り返す。

このような左処理と右処理に伴って左分枝と右分枝が発生し，左分枝上には展開に用いるリテラル，すなわち主項マークが描かれる。このような展開は，何らかのリテラルが抽出されて終るか，抽出すべきリテラルが何も無くなって終る。前者のような成功終端には，**{1}** を記入し，後者のような失敗終端には，**{0}** を記入することとする。このようにして得られた2進木の成功終端から逆に各分枝を遡り，その分枝上のリテラルを文字列として連結することによって展開後の積項が得られる。式(6)に対する2進木展開の事例を図1に示す。

この2進木展開により，式(6)にたいする最も短い論理積は $bdeg$ となり，結局，主項の最小被覆として

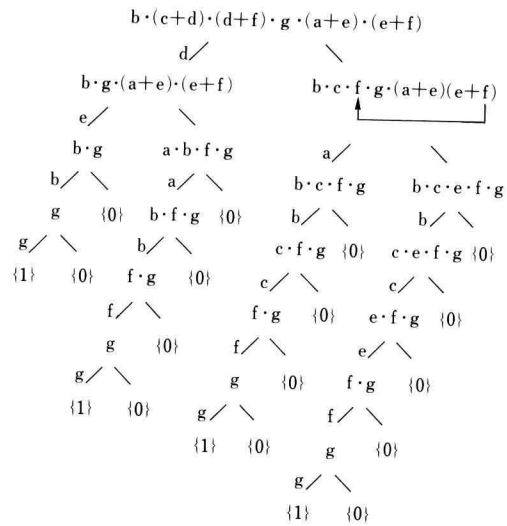

図1.　$P_s=b(c+d)(d+f)g(a+e)(e+f)$ にたいする2進木展開

Fig. 1.　Binary tree expansion for the equation $P_s=b(c+d)(d+f)g(a+e)(e+f)$

{0, 1}, {6, 7}, {11, 15} および {0, 2, 8, 10} が得られることになる。

3. プログラム

前章で述べた考え方に従って，ペトリック関数の展開による主項の最小被覆の導出のためのプログラムを考える。最初にプログラム向きアルゴリズムを述べ，続いてプログラムの概要を述べる。

3.1　プログラム向きアルゴリズム

ここでは2進木によりペトリック関数を展開して主項の最小被覆を生成するアルゴリズムの各ステップを述べる。このアルゴリズムはCOMMON LISP言語によるプログラムを前提としたものである。

［ステップ1］　このステップではペトリック関数を作成する。最小項番号の集合（すなわち最小項番号のリスト）として表される主項を一つずつ配列として記憶する。この配列の番号は対応する主項のマークそのものである。与えられた関数の最小項番号の集合より，最小項番号を一つずつ取り出し，いま作った主項の配列にこの番号が含まれないか調べ，含まれておれば，その配列の番号（すなわち主項マーク）を記憶し，これら配列の番号の集まりを1個のリストとする。もしこのリストの要素が1個しか無ければそれは必須項であり，リストの代りにアトムとする。このようにして作られた各最小項に対応するリストまたはアトムの集まりより成るリストがペトリック関数を表す。すなわち，求まった各アトムは必須項を，各リストは各論理和項を表し，これらのアトムとリストの論理積がペトリック関数を表すことになる。

［ステップ2］　このステップではペトリック関数の吸収処理を行う。まず，必須項に相当する単項（すなわちアトム）と主項の論理和項（すなわちリスト）とに分け，必須項に吸収される論理和項を削除する。つづいて論理和項同士の吸収削除を行う。この際，アトムは必須項を表しているのでそのまま記憶しておき，主項マークのリスト，すなわち論理和項のみの2進木処理を行う。

［ステップ3］　このステップでは最も使用頻度の高い主項マークを探す。ただし，主項マークのリスト（すなわち論理和項）が1個以下ならば，展開を続ける必要はないので，ステップ6へ移る。そうでない時には，主項マークの全リスト中に含まれる要素（すなわち主項マーク）をそのまま一連の要素とするリストを作成する。このように作成したリストに含まれる同じ主項マークの使用頻度を調べ，最高使用頻度の主項マークを決定する。この際，同じ使用頻度の主項マークが他にも存在するか否か調べ，もしそれがあれば，上位番号となる主項マークのものを最高使用頻度のものとして決定する。

［ステップ4］　ステップ3で決定した最高使用頻度の主項マーク（すなわちペトリック関数のリテラル）について展開を行う。ただし，その主項マークの最高使用頻度が1ならば，展開せずにステップ6に移る。そうでなければ，主項マーク（すなわちリテラル）のリストの中で最高の使用頻度を持ったリテラル x を抽出する。ついでこのリテラル x を含まない主項マークの全リスト Σlst を抽出し，リテラル x と Σlst より成るリストを作成する。さらに最高使用頻度を持つリテラル x を含む主項マークの全リストから，リテラル x を除いて得られる主項マークの全てのリスト $\Sigma l'st$ を求める。このような $\Sigma l'st$ と上述の Σlst とからなるもう一つのリストを作成する。このような2つのリストより成るリストが展開の結果を表している。なお，展開される前の式を表すリストの中で最高使用頻度を持ったリテラルが1個の時には，このリテラルについて展開して得られるリストの最後のリスト，すなわち

Σlst と $\Sigma l'st$ より作られる第2のリストは最小被覆からは除去し得る。

［ステップ5］　ステップ4で展開の結果得られた第1リストと第2リストの各々に記憶しておいたアトムを付け加えて，再びステップ2に戻り，主項マークのリストが1個以下になるか，主項マーク（すなわちリテラル）の最高使用頻度が1以下になるまでこれを繰り返す。

［ステップ6］　主項マークのリストが1個以下になった場合には，主項マークリストを展開し，これにアトムを付け加え，この結果を最小被覆候補リストに記憶する。また，最高使用頻度が1以下になった場合も同様な事を行う。

［ステップ7］　主項マークのリストが全て無くなったら，処理を終了し，最小被覆候補リストの中から真に最小被覆になるものを求め，その最小被覆の主項マークを主項に書き直す。

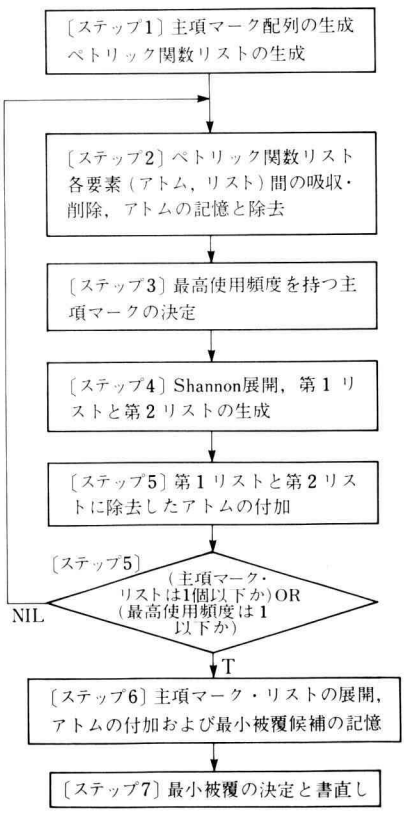

図2.　2進木展開による最小被覆生成のフローチャート

Fig. 2.　Flow chart for generating the minimal covering by the binary tree expansion

このアルゴリズムのフローチャートを図2に示す。

また，アルゴリズムに従って事例を式（1）の論理関数について考えてみる。ステップ1として，主項を表す最小項番号の集合を式（2）の代りに主項の配列 $P(i)$ に代入すると，

$$\left.\begin{aligned} &P(0)=\{10, 11\},\ P(1)=\{0, 1\},\ P(2)=\{2, 6\}\\ &P(3)=\{6, 7\},\ P(4)=\{11, 15\},\ P(5)=\{7, 15\}\\ &P(6)=\{0, 2, 8, 10\}\end{aligned}\right\} \tag{10}$$

が得られる。これよりペトリック関数をリストで表すと，

$$P_s:\ ((1\ 6)1(2\ 6)(2\ 3)(3\ 5)\\ 6(0\ 6)(0\ 4)(4\ 5)) \tag{11}$$

を得る。ステップ2として，式（11）のアトムと主項マークのリストとの間で吸収削除を行う。まず主項マーク・リスト（16）がアトム1に吸収削除される。ついで主項マーク・リスト（26）と（06）とがアトム6に吸収削除され，

$$P_s:\ (1\ 6(2\ 3)(3\ 5)(0\ 4)(4\ 5)) \tag{12}$$

を得る。式（12）でアトム1と6を記憶し，2進木処理は，これらのアトムを除去した

$$P_s':\ ((2\ 3)(3\ 5)(0\ 4)(4\ 5)) \tag{13}$$

について行うようにする。ステップ3として，式（13）で主項マークの最高使用頻度を持ったものを探し，最高使用頻度が2の主項マークとして3と4を得る。この中で上位の主項マークとして3を選択する。ついでステップ4として，主項マーク3について式（13）の P_s' を展開する。この主項マーク（リテラル）3を含まない全主項マーク・リスト Σlst として（04）と（45）が抽出される。これらとリテラル3とで第1のリストとして（3(04)(45)）が求まる。ついで式（13）の3を含む主項マーク・リストから3を除去することにより，リスト $\Sigma l'st$ として2・5が求まる。ここで，$\Sigma l'st$ と Σlst をまとめてリスト（2・5(04)(45)）を得る。ステップ5として，ステップ1で記憶したアトム1と6を，ステップ4で求めた第1のリストと第2のリストに追加し，それぞれ（163(04)(45)）と（1625(04)(45)）を得る。これら2つのリストについて再度ステップ2へ戻って展開処理を続ける。第1のリストについては1，

6および3をアトムとして記憶し，((0 4)(4 5))にたいして展開を行う。ステップ3で最高使用頻度の主項マークは4の1個のみであり，ステップ4で展開し，第2のリストは除去してアトム4を得る。さらにステップ5と6を経て最小被覆候補(1 6 3 4)を得る。また，第1回目の展開で求めた第2のリスト(1 6 2 5(0 4)(4 5))からは，ステップ2で吸収削除を行い，(1 6 2 5(0 4))について展開し，ステップ6で最小被覆候補として(1 6 2 5 0)と(1 6 2 5 4)を得る。

ステップ7として，このような最小被覆候補のリストの中から，真の最小被覆(1 6 3 4)を得る。これは主項マークのリストであるから，最小項の集合で主項を表すと，最小被覆 ***MINC*** は，

$$\boldsymbol{MINC}=\{\{0,1\}\{0,2,8,10\}\{6,7\}\{11,15\}\} \qquad (14)$$

となる。

ここで述べた事例について，2進木展開の模様を図3に示す。

3.2 プログラムの概要

前節で述べたアルゴリズムに従い，COMMON LISP言語を用いてプログラムを作成した。本論文では特に述べてはいないが，与えられた論理関数から主項を生成するのに用いた手法は3分枝展開法[5,6,1]と分割法[7]を併用した方法である。

プログラム用に作成した関数は，複素最小被覆生成用の13個（なお単一最小被覆生成用としてはさらに6個の関数が追加される）であった。なお，主項の生成

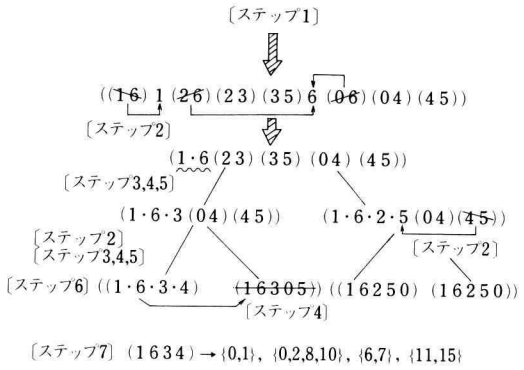

図3. P_s：((16)1(26)(23)(35)6(06)(04)(45))にたいする2進木展開の過程

Fig. 3. Processes of binary tree expansion for the equation, P_s: ((16)1(26)(23)(35)6(06)(04)(45))

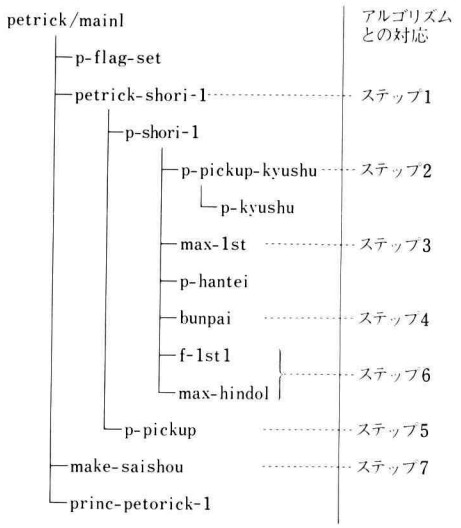

図4. 複数最小被覆を求めるプログラムの関数の相関図とアルゴリズムとの対応

Fig. 4. Correspondence between the correlation diagram of functions used in the program for obtaining the multiple minimal coverings and its algorithm

用には39個の関数を用いている。複数最小被覆生成用のプログラムに用いた関数名とその相関図を図4に示す。また，同じ図に関数とアルゴリズムとの対応を示す。

なお，複数最小被覆生成用プログラムの関数の一例としてbunpaiについて次に述べる。この関数は図2のフローチャートのステップ4に相当するものであって，2進木展開の中心部である。この関数の引数は *fl-lst* であって主項マーク・リストのリストである。この関数bunpaiの動作フローチャートを図5に示す。いま，主項マーク・リストの中でmax-*s*（最高使用頻度を持つリテラル）を含まないリストを一緒にして *l-lst* に設定する。一方，max-*s* を含むリストは，そのリストからmax-*s* を除去したものをまとめて *r-lst* に設定する。リストの要素が1個の場合はアトムとする。ここで，max-*s* と *l-lst* をappendすると，2進木展開の左分枝となる。また，*r-lst* と *l-lst* をappendすると右分枝となる。最後でこれらの2つのリストを一つのリストにまとめる。*fl-lst* ((2 3)(3 5)(0 4)(4 5))を例にとって考える。max-*s* は3となるから，*l-lst* は((0 4)(4 5))となり，*r-lst* は(2 5)となる。よって左分枝は(3(0 4)(4 5))となり，右分枝は(2 5(0 4)(4 5))と

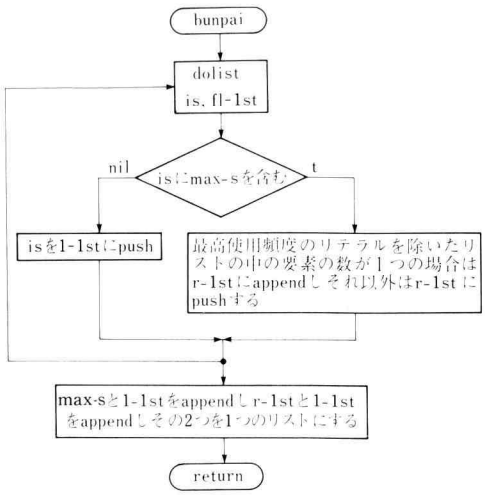

図5.　関数 bunpai の動作フローチャート
Fig. 5.　Flow chart for operating the function, "bunpai"

なる。これらを append すると，((3(0 4)(4 5))(2 5(0 4)(4 5)) が求まる。

なお複数最小被覆を生成する場合には，ペトリック関数の2進木展開を行う過程で次第にリストが増加する。したがって適宜削除するようにしてリストの数を減らすように配慮する。また，2進木展開中にアトムだけになり，成功終端に達したものについては展開中の2進木の展開の各段のリストから削除し，最小被覆候補項のリスト中に記憶するように配慮している。

4.　実験結果とその検討

本章では，前章までに述べた主項にたいする複数最小被覆の生成法にしたがって作成した COMMON LISP 言語プログラムを実行した結果について述べる。勿論，このプログラムの実行には主項生成のプログラムを実行させる必要があり，そのため多段分割による3分枝展開法を用いた。なお，参考のため複数最小被覆の生成結果と単一最小被覆の生成結果とが比較された。この単一最小被覆の生成法では，複数最小被

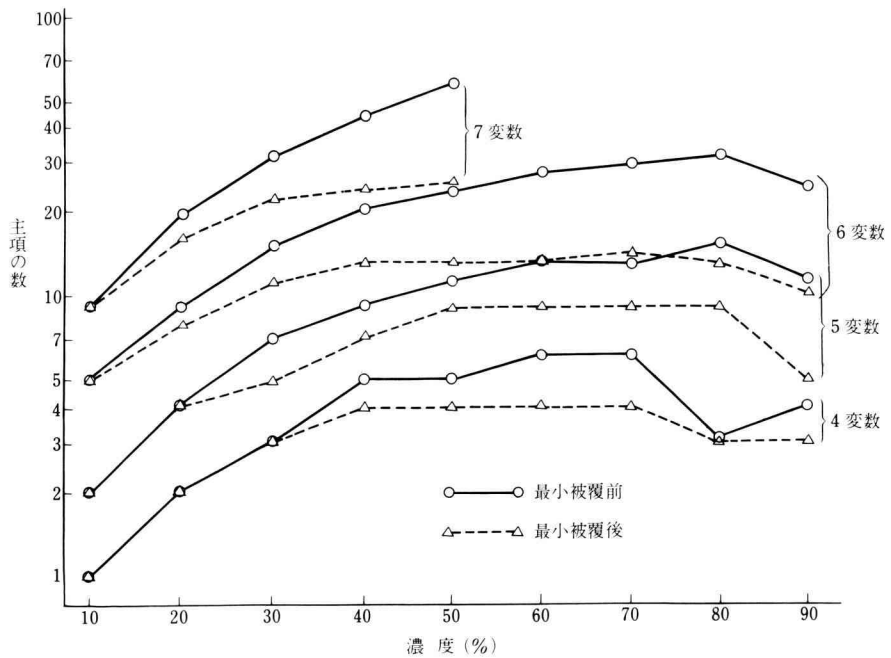

図6.　複数最小被覆の生成前後の主項数の比較
Fig. 6.　Comparison of the numbers of prime implicants before and after generating the multiple minimal coverings

覆生成法のペトリック関数の2進木展開を最小被覆の発見と同時に打ち切るよう配慮している。勿論，これらのプログラムにはすべてCOMMON LISP言語を同いた。

プログラムの実行はLISP専用ワークステーションELIS（NTT-IT製）上で行った。プログラム実行用の入力として，最小項番号を，与えられた濃度に応じた個数だけ乱数を用いて発生させる。この乱数発生は同じ濃度に対し10回繰り返し，その都度プログラムを実行している。得られた各測定結果を10回平均した。ここで濃度とは，カルノー図上の全セル数に対する最小項数の比率（%）を表している。

図6は複数最小被覆を生成する前と後の主項の数を測定したものであり，変数の数として4変数から7変数を扱った。図7は主項および最小被覆生成の演算時間の測定結果を示している。この図では，4～8変数を対象としており，複数最小被覆生成，単一最小被覆生成および主項生成に要した演算時間を示してある。図6および図7の測定では，いずれも主項生成のための演算時間がなるべく短かくなるような分割数と段数が

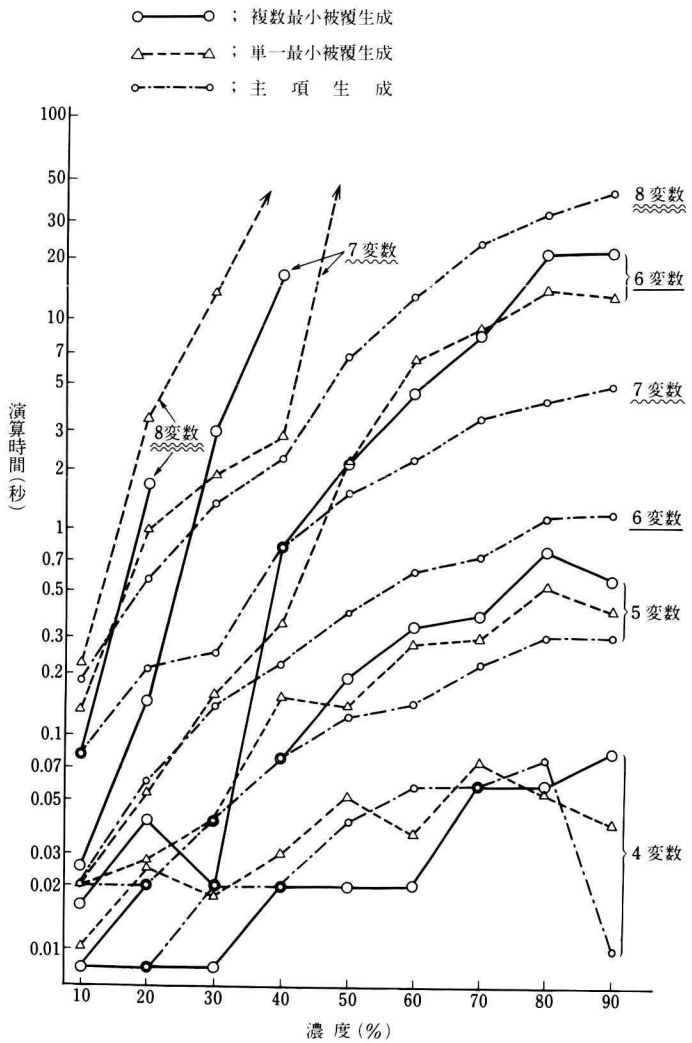

図7.　主項および最小被覆生成の演算時間
Fig. 7.　Computer operating time for generating the prime implicants and the minimal coverings

選んである。

図6の結果は最小被覆を求めた後の主項の数が最小被覆前の考えられる全主項の数よりも大きく減少することを示している。特に濃度が高く，変数が多変数になる程，この傾向が強い，例えば6変数の場合，濃度80%で最小被覆前後で主項数が30個から13個に減少していることが分る。図7の結果より，次のことが言える。

(1)　主項生成に要する演算時間は，低変数の場合，最小被覆生成に要する演算時間と同程度である。しかし高変数になると，主項生成のための演算時間は最小被覆生成のための演算時間の極く小部分しか占めない。例えば6変数で濃度80%の時，複数最小被覆の生成に22秒要するのに対し，主項生成には1.2秒しか必要とならない。

(2)　濃度の増大と変数の数の増大に伴い，最小被覆生成のための演算時間は増大する。

(3)　複数最小被覆の生成に当する演算時間と単一最小被覆の生成に要する演算時間の差は余り明瞭でない。

(1)で述べたことは，変数の数の増大と共に主項の数が増大し，主項の組合せが増大することに起因すると考えられる。(2)で述べた事項も同様に理解できる。(3)で述べた事項は単一最小被覆の生成が複数最小被覆の生成の途中の段階で打ち切られるものであることと矛盾している。これは複数最小被覆の生成と単一最小被覆の生成で測定条件が必ずしも一致しなかったことが第一の理由と考えられる。第二の理由としては，両者とも与えられた全主項の多数の組合せの探索に時間を消費し，特に展開の初期の段階で膨大な時間を要したことが考えられる。

最後に，演算時間として許し得る値を20秒とすると，複数最小被覆の生成では，7変数で濃度40%（この場合の演算時間は17秒），また，8変数で濃度20%（この場合の演算時間は1.7秒）の場合が生成可能の限界であった。特に，実際に最小被覆に使用された主項数は前者の7変数，濃度40%の場合で25であった。

5. 結　　言

本論文では，論理関数の主項に対する複数最小被覆を求める手法としてペトリック関数の2進木展開法を用いる手法について述べた。さらにこの手法をCOMMON LISP言語でプログラム化してワークステーションELIS上で実行させた。この結果7変数まで最小被覆を求めることが可能で，演算時間は約20秒，最小被覆の主項数は約25という結果を得た。この手法は複数個の最小被覆を全部求める場合であるから，メモリの面から苛酷な条件となっている。筆者は必須項のある場合の複数最小被覆の生成法をも検討しているが，その結果はここで得られた結果に近くなっており，本論文の結果はほぼ限界に達していると想像される。

したがって今後は複数最小被覆の生成法を研究することも必要であるが，単一最小被覆の生成法について特に研究を進めたい。本論文で付随的に単一最小被覆の生成の実験結果について触れた。その成果は必ずしも顕著ではなかったが，既に別法を検討中であり，9変数までの単一最小被覆の生成が確認されている。

終りに当り，本論文のアルゴリズムを実現するプログラムの作成に尽力された平成元年度の卒業研究生渡辺泰仁君に謝意を表する。

参 考 文 献

1) K. Goto, “Five methods for simplification of logic function and Comparison of their characteristics,” 1990 IEEE International Symposium on Circuits and Systems, pp. 1122-1125 (1990).
2) S. Muroga, Logic Design and Switching Theory, John wiley & Sons (1979).
3) S.R. Petrick, “On the minimization of Boolean functions,” Proceedings of the International Conference on Information Processing, pp. 422-423 (1960).
4) C.E. Shannon, “A symbolic analysis of relay and switching circuits,” Trans. AIEE, Vol. 57, pp. 713-723 (1938).
5) J.R. Slagle, C.L. Chang, & R.C.T. Lee, “A new algorithm for generating prime implicants,” IEEE Trans., Vol. C-19, No. 4, pp. 304-310 (1970).
6) 矢板　徹，“論理関数の共有展開，” 電子通信学会技術報告，EC 85-2, pp. 11-22 (1985).
7) 後藤公雄，“隣接グループ生成法と分割法による論理関数の簡単化の一手法，” 神奈川工科大学研究報告，No. B-13, pp. 155-160 (1989).