ブール関数の主項を生成する
—再帰的プログラムについて

松　田　秀　雄

緒　　言

再帰的プログラミングはデータサイズnの問題を解く手続きが，いくつかの，より小さなサイズn_1（$<n$）の手順に分けて，自分自身を副手順として階層的に呼び出していく方法で，プログラミングの形がエレガントになる，メモリが節約できる，問題によっては計算時間の短縮が可能であるなどの特長があるとされている[1]。本論文ではブール関数の主項（Prime Implicant）を計算機で能率よく求めるために筆者らが提案している部分マップ法をFORTRAN言語で再帰的プログラミングしてみた結果について述べている。前回の部分関数法[2]の場合に比べ，分割する部分問題の数がふえるので，合成過程が複雑となり，スタックカウンタを又スタックレジスタに入れておかなければならない，いわゆる二重スタックが必要となるなどの様相を明らかにする。

1．理　　論

1.1　部分マップ法

1.1.1　諸定義　　マップ上のセルは2進座標で表わされるが，表1のように，これを1の数の少ない順に，同一数なら2進数として小さい順に番号付けを行う。図1に4変数のマップの例を示す。セルiの座標で1をとる変数の肯定リテラルの積項で表わされるマップ上のキューブを許容キューブiと呼びP_iと記す。セル1は肯定変数をもたないが特別なものとして，P_1＝マップ全体とする。$P_2=x_4$, $P_3=x_3$, ……, $P_{16}=x_1x_2x_3x_4$である（図1に例）。セルiの座標で0をとる変数の否定リテラルの積項で表わされるマップ上の部分を部分マップiと呼びSで記す。このときもセル16は否定変数をもたないが特別なものとして，S_{16}＝マップ全体とする。他のセルは定義通り，$S_{15}=x'_4$, $S_{14}=x'_3$（図2参照），……, $S_1=x'_1x'_2x'_3x'_4$でセル1のみからなる部分マップである。又部分マップiの許容キューブjを$P_{i,j}$で表わす。S_{16}は元のマップに等しいので，$P_{16,j}=P_j$である。このように定義すると，セル番号iが大きい程，部分マップは大きくなり，

$$S_{16}\geq S_{15}\geq\cdots\cdots\geq S_2\geq S_1$$

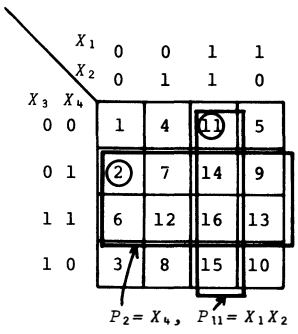


図1　許容キューブの例

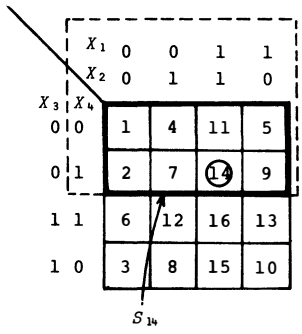


図2　部分マップの例

表1　ベクトルの並べ方

セル番号	2進ベクトル X_1	X_2	X_3	X_4	$P_{14,4}$
①	0	0	0	0	
②	0	0	0	1	
3	0	0	1	0	
④	0	1	0	0	4
⑤	1	0	0	0	
6	0	0	1	1	
⑦	0	1	0	1	7
8	0	1	1	0	
⑨	1	0	0	1	
10	1	0	1	0	
⑪	1	1	0	0	11
12	0	1	1	1	
13	1	0	1	1	
⑭	1	1	0	1	14
15	1	1	1	0	
16	1	1	1	1	

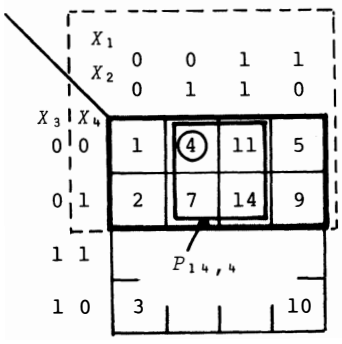


図3　部分マップ14の許容キューブ4

逆に，許容キューブの方では小さくなり

$$P_{16} \leq S_{15} \leq \cdots\cdots \leq P_2 \leq P_1$$

である。但し，部分マップS_i及び許容キューブP_iの大小関係とは，S_i及びP_iがそれぞれ含むセルの数でいう。

S_i及び$P_{i,j}$は計算機上容易に発生させることができる。それには表1の順で並べた2進ベクトルを用いる。例えば図2の部分マップ14はセル14の座標が（1，1，0，1）から，x_3が0となるセルを表1で，セル14から上に向かって探し出していけば得られる。すなわち，左欄で○印を付したセル集合がこれにあたる。又，部分マップ14の許容キューブ4，すなわち$P_{14,4}$は図で示したセルの集合 {4，7，11，14} からなるが，これも表1で，セル4の座標が（0，1，0，0）からx_2が1となっているセルをセル4から下にたどって，部分マップ14に含まれるセルの集合の中から選び出せばよい。すると，右端欄に上げた，{4，7，11，14} が$P_{14,4}$として得られる。他のS_i，$P_{i,j}$も同様な方法で表1を操作して発生させることができる。

1.1.2　基本部分マップ法　　部分マップ法については，すでに文献(3)，(4)※で詳述しているので，ここでは大筋だけを述べる。

(1)　論理関数Fをマップ上で与える。

(2)　部分マップS_iをセル番号の大きい順に発生する（但し，trueのセルのみ）。

(3)　各S_iごとに許容キューブ$P_{i,j}$をセル番号jの小さい順に発生する（但し，trueのセルのみ）。

(4)　関数Fと$P_{i,j}$の論理積をとる。$F \cap P_{i,j} = P_{i,j}$ なら，既知の主項と包含関係を調べ，含まれなければ主項として登録する。

(5)　(2)から(4)を繰り返す。

但し，大きな$P_{i,j}$が主項となれば，それに含まれるセルの許容キューブの発生は不用である，又，ある部分マップのtrueのセルが許容キューブのいくつかでことごとくカバーされるなら，その部分マップに含まれるセルの部分マップは省略できる，などの諸性質で(2)から(4)の繰り返し数が大幅に減少するので，比較的早くすべての主項が求まる。

1.2　再帰的方法

再帰的プログラミングとはデータサイズnの処理手順を考えるとき，複数個のより小さなサイズn_1（$<n$）の問題に変えて，自分自身の手順を階層的に呼び出していく方法であるから，部分マップ法を再帰的にするためには，まず，問題を部分問題に分割する方法から考えねばならない。そこで，n変数（x_1，x_2，…，x_n）のマップをn_1（$<n$）変数（$x_1, x_2, \cdots, x_{n_1}$）座標（0，0，…，0），（0，0，…，1），…，（1，1，…，1）で分ける。その結果n_2（$=n-n_1$）変数（$x_{n_1+1}, x_{n_1+2}, \cdots, x_n$）のマップが$2^{n_1}$個でき，それぞれ$M_1, M_2, \cdots, M_{2^{n_1}}$と表わす。又，関数$F$もこれらの分割にしたがって，各マップ上で，それぞれ

※文献(3)，(4)では縮少カルノー図法といっている。

$F_1, F_2, \cdots, F_{2^{n_1}}$となるものとする。これを例示したのが図4で，$n=5$変数のマップを$n_1=3$変数（$x_1, x_2, x_3$）の座標で分割し，$n_2=2$変数（$x_4, x_5$）のマップが$2^3=8$個得られている。それぞれ$M_1, M_2, \cdots, M_8$と付してある。ここで○印のセルが与えられた関数$F$のtrueのセルを表わすものとする。$F$も分割されて，それぞれ$F_1, F_2, \cdots, F_8$となる。

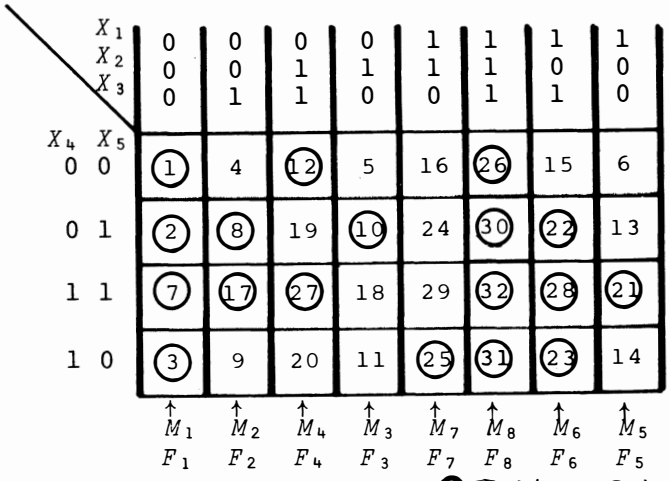


図4　マップ及び関数の分割

さて，そこでn_2変数のマップで一つの許容キューブ$P_{i,j}$を発生し，各F_k（$k=1, 2, \cdots, 2^{n_1}$）と論理積をとり，

$$P_{i,j} \cap F_k = P_{i,j} \qquad (1)$$

が成り立てば，kに対応するn_1変数のマップ上のセルを1（true），成り立たなければ0（false）として，新たな関数$f_{i,j}$を作る。

今の例で，$n_2=2$変数のマップで，例えば部分マップ4の許容キューブ2，つまり$P_{4,2}$を発生して，$F_1, F_2, \cdots, F_8$の各関数と論理積をとって$f_{4,2}$を作っていく様子を表わしたのが図5である。F_1, F_2では$P_{4,2}$に含まれるすべてのセルが○印なので，式(1)が成り立ち，$f_{4,2}=1$，F_4，F_5では$P_{4,2}$に含まれるセルの中で○印でないものがあるので式(1)が成り立たず，0となっている。

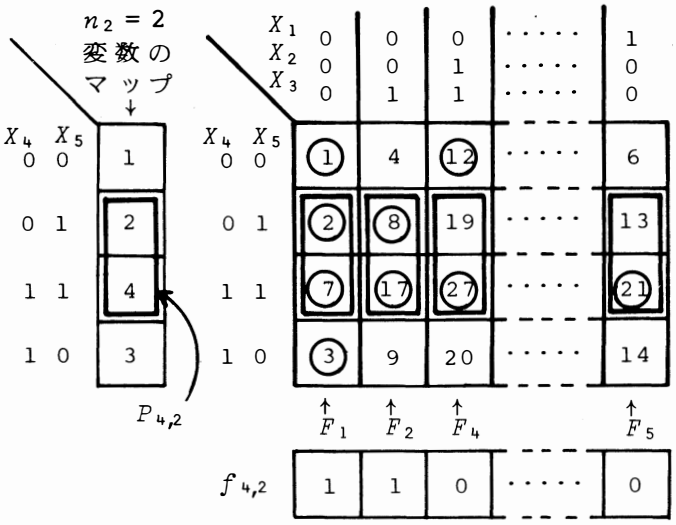


図5　新しい関数$f_{i,j}$の発生例

このようにして得られた$f_{i,j}$に基本部分マップ法を適用して主項を求める。これらの主項に$P_{i,j}$のリテラルを組合わせると元の関数Fの主項が求まる。例えば図5で求めた，$f_{4,2}$の主項の一つにx_1x_3が得られるが，これに$P_{4,2}$のリテラルx_5を付加した$x_1x_3x_5$は元の関数F（図4）の主項の一つとなる。

上記の原理で，n変数の論理関数Fの主項は3^{n_2}個（実際はこれよりずっと少ない）のn_1（$=n-n_2$）変数の論理関数$f_{i,j}$の主項を求め，これらに必要なリテラルの合成を行って求められる。ところで，ここで$f_{i,j}$に更にこの手順を繰り返すと，いくつかのより小さなn_1'（$<n_1$）変数の関数$f'_{i,j}$が得られ，更にもう一度…と繰り返すと，希望のn_0変数の関数まで変数の数を減らせる。ここで部分マップ法で主項を求め，逆に今迄の手順をさかのぼって次々とリテラルを合成していけば，やはり元の関数Fの主項が求まる。この導出過程をアルゴル流に書くと図6となる。この手続きはプログラムの中で自分自身の手続きを呼び出しているので，再帰的プログラムといわれている。

図6は上述の手法の流れだけを書き上げたもので，途中で得られた主項を蓄えたり、主項の包含関

係を調べたりする操作など，細かなことは一切省略している。

2．FORTRAN プログラム

図6の再帰的プログラムをFORTRANプログラムで組むとすると，副手順がその手続き中に自分自身を呼び出すことを禁じているので，特別な工夫が必要となる。一つのプログラムを手を変え品を変えて使うため，その手順に入る前に，一々，それまでに使われていた各種パラメータやデータを一ったんソフトウェア的に作ったスタックレジスタに記憶し，後ほど，その呼び出しまでに復帰してきたとき，それらのデータを再び使って計算することになるので，それにそなえねばならない。このような操作は適当な配列とスタックカウンタと呼ばれるポインタとでできる。

```
                procedure SUBMAP(F,n):
    begin
1.   if  n ≤ n0  then
       begin
2.       F に部分マップ法を適用して主項を求める;
3.       return
       end
4.   else
       begin
5.       F を F1 ,F2 ,··· F2n1  の関数に分ける;
         begin
6.         Pij を発生;
7.         Pijと各 Fk との論理積をとりfij を作る;
8.         SUBMAP(fij,n1);
9.         fij の主項に Pij のリテラルを合成
         end      (6. ～ 9. は必要なキューブの数
10.      return                だけ繰り返す)
       end
    end
```

図6　再帰的プログラム

ここで，計算機上，各種データがどのような形で表現されているかを述べておく。関数Fは配列$FF1(K)$を使って2進ベットルで表わす。部分マップ，許容キューブはそれぞれに含まれるセル番号で表わす。ある許容キューブが式(1)の論理積をみたし，主項となるなら，それをリテラルの積項で配列$XDQ(I,J)$に記憶しておく。例えば，4変数なら4次元ベクトルで表わし，図1の$P_2=x_4$はx_1, x_2, x_3が任意変数として陽に表われていないので，1，2，3の座標を2とし，x_4は肯定変数なので4の座標を1とおいた（2，2，2，1）で記憶する。もし否定変数が表われればその座標を0とおく。すると，$P_{11}=x_1x_2$は（1，1，2，2），図3の$P_{14,4}=x_2x'_3$は（2，1，0，2）と表わして記憶される。

図7から図9まではFORTRANプログラムの概略図である。左端に付したLine-Numberによって説明していこう。*14.*から*37.*まで（但し，*24.*, *31.*～*35.*をのぞく）が本プログラムの基本部分である。NS変数の関数$FF1$が与えられたとき（*14.*），2^{NS1}個の$NS2$変数の関数$FBT(I,J)$に分割する（*18.*），但し，$NS=NS1+NS2$。$NS2$変数のマップで部分マップごとに許容キューブ$P_{IQL,IQ}$を発生（*19.*, *21.*）し，分割した関数FBTとの論理積をとって，$NS1$変数の関数$FFF(I)$を作る（*22.*）。これに部分マップ法を適用して，$FFF(I)$の主項をまず求め，それに$P_{IQL,IQ}$のリテラルを合成して，$FF1$の主項を求めていく（*25.*から*29.*まで），プログラムになっている（但し，簡単のため*28.*, *29.*では$P_{FLQ,JQ}$に$P_{IQL,IQ}$のリテラルを付加したものを単に$P_{FLQ,JQ}$と表わしている）。

本プログラムではこの部分を基本的手順SABMAPとして再帰的呼び出しを行っているので，プロ

グラムの他の部分は，データの一時蓄えや，戻り番地の記憶など，再帰的プログラムを組むために必要となった手続きである。したがって，他の部分と基本部との関連にふれながら各部の説明を行っていく。

まず，プログラムの最初の部分 *1.*, *2.*, *3.* で関数 F を与える。NSA は F の変数の数だが，F 及び NSA は $FFALL$ 及び $NALL$ を通して，SUBMAP（*11.* 以下 *43.* までの手順）で $FF1$（*14.*）及び NS（*12.*）にそれぞれ引き渡される。$NEND$ は図6の n_0 に相当し，何度か手順を呼んで関数を分割し，変数の数 NS が $NEND$ に等しくなったところで，それ以上の分割を止め，部分マップ法を適用して主項を求めるプログラムに入る判定常数で SUBMAP の *24.* で使われている。*6.* の変数 $GOSEI$ は SABMAP を呼んで分割し続ける間は0で，一ったん主項が求まり，リテラルを次々と合成していく手順の復帰過程では1の値をとるパラメータである。*7.* の $STACK$ ははじめ0で，SUBMAP を呼び出すごとに1ずつふえる（*11.*），再帰的呼び出しの深さを表わす，スタックカウンタである。*8.* の配列 $RE(STACK)$ は SABMAP を呼び出して，この手順が完了したときに，どの番地へ戻ればよいかを表わすためのスタックである。RE に情報が蓄えられるのは *9.* と *48.* の二個所で，いずれも SUBMAP を呼び出す直前である。*9.* で RE

```
3
4
1.        NSA = 6
2.        NEND = 4
.         関数 F を FFALL(I) に与える
.         NALL(1) = NSA
          ............
6.        GOSEI = 0
7.        STACK = 0
8.        RE(1) = 1
9.        GO TO 740
10.     735 GO TO 4100
      C   PROCEDURE SUBMAP
11.     740 STACK = STACK+1
12.       NS = NALL(STACK)
          ............
14.       関数を FFALL から FF1 へ移す
            FF1(I) = FFALL(I)
15.       NS1 = NS - 1
16.       NS2 = NS - NS1
17.       STACK2 = 0
          ............
18.       FF1 を 2^NS1 個の NS2 変数関数 F_i(i=1,2^NS1)
          に分割、ここでは FBT(J,I) で表わす
19.       NS2 変数のマップでセル番号 IQL の大きい順に
          部分マップを発生して、4000 までを繰り返す
20.       但し、分割した関数 FBT(J,I) がすべて
          セル IQL で0ならば、直ちに 4000 へ跳ぶ
21.       許容キューブ P_IQL,IQ を大きい順に
          発生して、以下 3000 までを繰り返す
22.       P_IQL,IQ と F_k の論理積から
          NS1 変数の関数 FFF(I) を作る
23.       FFF がすべて 0 なら直ちに 3000 へいく
24.       IF (NS.GT.NEND) GO TO 3100
      C   以下 2800 まで FFF に部分マップ法適用
25.       NS1 変数のマップでセル番号 FLQ の大きい順に
          部分マップを発生して、2800 までを繰り返す
26.       許容キューブ P_FLQ,FJQ を大きい順に
          発生させて 2800 までを繰り返す
27.       論理積 P_FLQ,FJQ ∩ FFF = P_FLQ,FJQ か？
            NO → 2800
            YES ↓
28.       すでに XDQ(I,J) の第 C1 列目から第 C2 列
          目までに入っている主項と P_FLQ,FJQ
          との包含関係を照合する
                ↓
          ..... 次の図へ続く .....
```

図7　FORTRANによるプログラム（その1）

(1)＝1, *48.*で$RE(STACK)$＝2（このとき，$STACK \geq 2$となっている）とおいて，それぞれ呼び出しの場所を区別する。やがて，SABMAPの呼び出しが終わって，*41.*, *42.*に達したとき，$RE(STACK)$の内容が1か2かに応じて，文番号735か3200かのいずれかへ復帰する。特にREの内容が1となるのは*9.*で呼び出したときだけで，しかもここでの一回だけであるから，もし，ある呼び出しの手順で*41.*, *42.*に達し，$RE=1$で，735へ戻ったときにはNSA変数の関数Fの主項がすべて求まったという状態になっている。それで文番号4100(図9)へとび，これらの結果を印刷して，プログラムは終了する。

だが，一般にはここまでに到達するまでに何度もSABMAPの呼び出しが行われる。手順を呼んでも$NS>NEND$の状態が続くから，*24.*で文番号3100（図8）へとぶ。図8のプログラム*43.*から*48.*では関数を分割するため，更に手順を呼び出す前準備を行う。すなわち，$FF1$を分割してできた$FBT(J,I)$(*18.*)や変数の数NSなどをスタック$FBTALL(J,IC)$や$NALL(STACK)$へ蓄える。現在の呼び出しでのFBTの内容を$FBTALL$の何行目から何行目までに入れておくかを表わすポインタもそれ専用のスタックへ蓄える。その他にこの呼び出しでは，どの部分マップのどの許容キューブについて計算

```
29.          包含されなければ PFLQ,FJQ を主項として
             XDQ(I,J)    に新しく追加
30.     2800 CONTINUE
31.     2850 STACK2 =STACK2 + 1
32.          IF (GOSEI.EQ.1) GO TO 2900
33.          C1D(STACK,STACK2) = C1
34.          GO TO 3000
35.     2900 C1D(STACK,STACK2) = C1D(STACK+1,1)
36.     3000 CONTINUE
37.     4000 CONTINUE
38.     4010 IF (NS.NE.NEND) GO TO 3235
39.     4030 REC = RE(STACK)
40.          STACK = STACK - 1
41.          IF (REC.EQ.2) GO TO 3200
42.          GO TO 735
          C  ****************************************
          C    更に分割するため手順を呼ぶ前準備
43.     3100 FBT(J,I)  をスタック FBTALL(J,IC) へ記憶
44.          NSALL(STACK) = NS
             その他、IQL, IQ など必要なデータやパラメ
             ータを各スタックへ蓄える
             ........................
45.          FFALL(I) = FFF(I)
46.          NALL(STACK+1) = NS1
47.          RE(STACK+1) = 2
48.          GO TO 740
          C  ****************************************
          C    ある呼び出し中の手順で、NS1 変数の関数
          C     FFF の主項が求まったので、これにNS2 変数
          C    のマップの許容キューブ PIQL,IQ のリテラ
          C    ルを合成する処理プログラム
49.     3200 NS  = NSALL(STACK)
50.          IQL = IQLALL(STACK)
51.               など必要なデータやパラメータを
             スタックから変数へもどす
52.          GOSEI = 1
53.          XDQ(I,J)    に入っている第C1D( STACK+1,1)
               列目から第 C2 列目までの主項に PIQL,IQ
               のリテラルをつける
54.          GO TO 2850
          C  ****************************************
                          ↓    ..... 次の図へ続く .....
```

図8　FORTRANによるプログラム(その2)

```
      C     ある呼び出し中の手順ですべての部分マップ
      C     のすべての許容キューブにつき主項となるも
      C     のにリテラルを合成しおわったのでそれらの
      C     間の包含関係を調べるプログラム
55.    3235 第 C1D(STACK,1) 列目から第 C2 列目まで
           XDQ(I,J) に蓄えられている主項の包含関係
           を調べる
56.        GO TO 4030
      C     ************************************
      C     呼び出し中のすべての手順が復帰したので
      C     結果が求まる
57.    4100 CONTINUE
58.         関数 F の主項がすべて求まった
           ので印刷する
59.        STOP
```

図9　FORTRANによるプログラム(その3)

を行っているのかを表わす情報*IQL*や*IQ*の値もスタックへ入れる。その他10個余りのデータやパラメータを蓄えるためのスタックを用いているが省略してある。*45.*, *46.*の命令で，今求められている*FFF*や*NS1*が次のSUBMAPの呼び出しで，その手順での*FF1*や*NS*に変わる。なお，SUBMAPの手順中*16.*で，*NS1*=*NS*－1となっているが，これは分割ごとに関数の変数の数が1ずつ減っていくわけで，*NS1*=*NS*−*NS2*, *NS2*は*NS*より小さな任意の整数とおけば，任意の数だけ減少させるようにできる。*47.*は先述したように*48.*のSABMAPの呼び出しで，この手順が終了したときには（*41.*で）次の文番号3200へとぶよう，*RE*の内容を2としているのである。

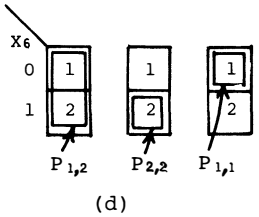
何度かSUBMAPを呼び出せば，*NS*=*NEND*になる筈である。このとき，SUBMAPの手順*25.*以下が実行されて，分割された関尊*FFF*の主項が求まり，リテラルが合成されて，*XDQ*(*I,J*)の第*C*1列目から第*C*2列目までに蓄えられる（*25.*～*30.*）。*31.*で*STACK*2が現われるが，これは第2のスタックカウンタで，手順*33.*の*C*1*D*(*STACK*, *STACK*2)として使われている。*C*1*D*は二つのスタックカウンタをもち，*STACK*2が実は又スタックに蓄えられる必要があるという意味で二重スタックといおう。*C*1*D*の働きや，*32.*から*35.*までの命令については次章で詳述する。

NS=*NEND*について基本部の処理が一わたり完了すると，*38.*に達するが，いまの場合*39.*, *40.*と続き，*STACK*の数を1だけ減らして，文番号3200へ復帰する。これはより上位でSUBMAPを呼び出していた手順のある部分マップ*IQL*のある許容キューブ$P_{IQL,IQ}$について作った関数*FFF*の主項が求まったことにあたり，その手順での次の段階，つまり次の許容キューブについての処理へと進む。そのためには，その段階での呼び出しで使われていた変数の数*NS*や*IQL*, *IQ*などをスタックから呼び戻す必要がある。その手続きが*49.*から*51.*である。ここでこの段階での$P_{IQL,IQ}$のリテラルを合成するため，*GOSEI*=1とおき，*XDQ*に記憶している主項にだけ$P_{IQL,IQ}$のリテラルをつける(*53.*)。そして*54.*で文

```
F=0100010001*0010101011*0000001011*0001000101*0100100000*1000001000*0010
                                                  * 10けたずつの区切り記号
                    (a)

FBT(1,J)=0000101011011100010000110101001 1
FBT(2,J)=1000000001010100001000000000000 0
                    (b)

    FFF =1000000001010100000000000000000 0
                    (c)
```



(d)

図10　実行例に使われた関数と1変数のマップ

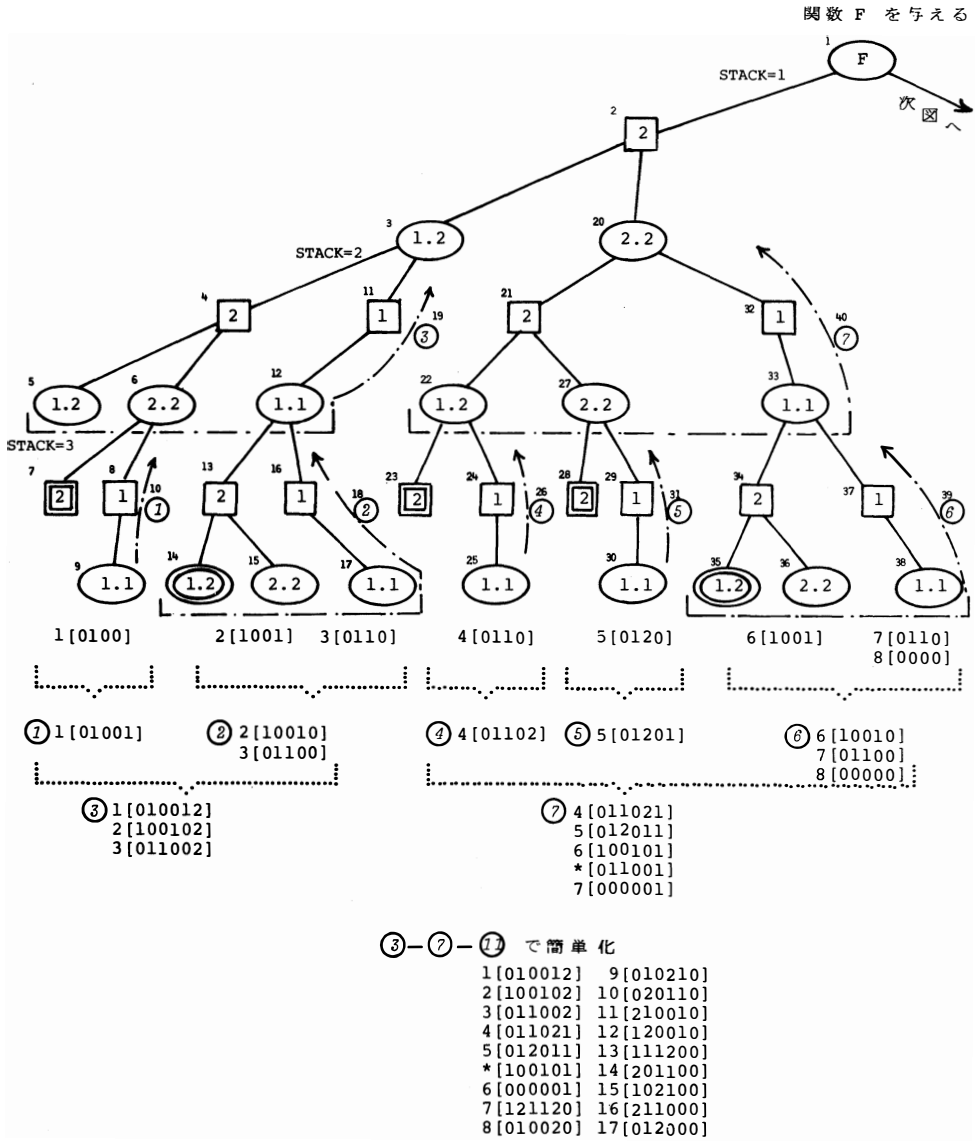


図11　計算処理手順の流れ(その１)

番号2850へとび，*31.*, 35.の処理をして，次の許容キューブ$P_{IQL,IQ+1}$へと移る(*36.*, *21.*)。

この呼び出し手順でのすべての処理が完了すると再び*38.*に達するが，$NS=NEND$のとき以前に呼び出した手順へさかのぼっているので，$NS>NEND$となっている。それで文番号3235（図９）へとぶ。ここで，今リテラルを付加した主項の中だけで，他のものに含まれるものがないかどうかを調べる（プログラムではステップ数を減らすため，SABMAP中の*28.*, *29.*と手順を共用している)。そして4030（図８）へ戻って，$STACK$の内容を１だけ減らして，更にその先に呼び出している手順へと復帰していく。

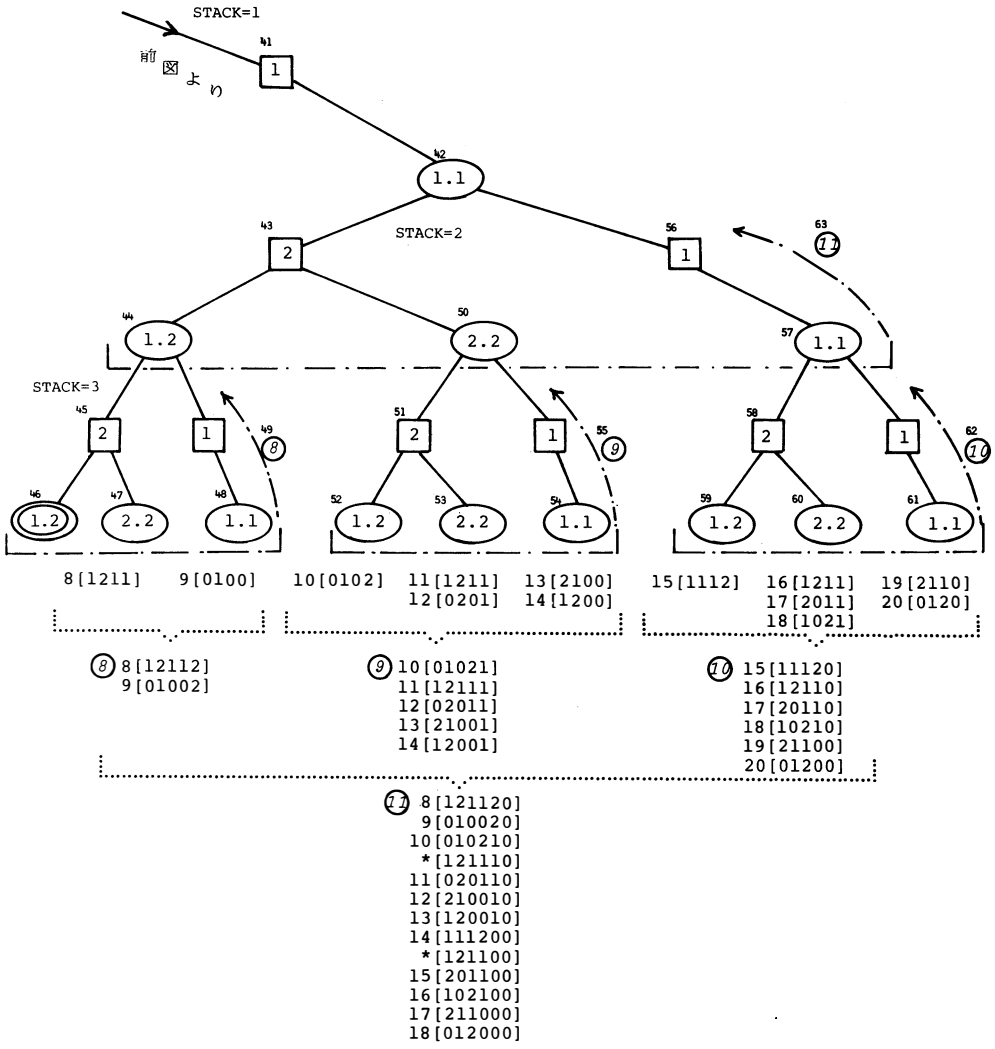


図12　計算処理手順の流れ(その２)

3 .実 行 結 果

図11及び図12は前章のプログラムを実行させたときの一例について，その計算処理手順の流れを図で表わしたものである。小文字の肩付き数字が計算順序を表わし，四角のます形の中の数字がどの部分マップの処理（プログラムの手順*19.*，以下単に*19.*のように略）を行っているかを，又だ円の中の数字がその部分マップのどの許容キューブについての処理(*21.*)が以下に行われるかをそれぞれ表現している。

まず小文字１で関数*F*が与えられる。*F*は図10(ａ)で示した６変数の関数である。直ちに，SUBMAP

が呼ばれ，$STACK=1$ となる。続いて関数が分割されるわけだが，プログラムでは$NS1=NS-1$，$NS2=1$ であるから，関数が$2^{NS1}=2^5=32$個の1変数関数に分割される。なお，$NS2=1$ 変数として，その呼び出しで使われる関数の最上位の変数，ここではx_6が選ばれるものとする。図10(b)の$FBT(I,J)$はこの分割された関数を示す。1変数のマップでは図10(d)に示すように部分マップが2と1の二つ，又部分マップ2には許容キューブが$P_{2,1}$と$P_{2,2}$の二個，部分マップ1には$P_{1,1}$ただ一個の許容キューブがある。図11，図12では部分マップ2についての処理(*19.*)をしているときには[2]で，部分マップ1の処理をしているときときには[1]で，又許容キューブ$P_{2,1}$，$P_{2,2}$及び$P_{1,1}$について処理(*21.*)しているときには(1.2)，(2.2)及び(1.1)とそれぞれ表わしている。

小文字2，3と進むと，部分マップ2の許容キューブ$P_{2,1}$を発生して関数FBTと論理積をとり，FFFを作る(*22.*)。$P_{2,1}$はセル {1, 2} であるから，$FBT(1,J)$と$FBT(2,J)$の両方に1がある列を1，その他の列を0とおいて得られるベクトルがFFFで，図10(c)にこれを示す。FFFは5変数関数なので，ここで再びSUBMAPが呼ばれ，$STACK=2$ と変わる。小文字4，5と進んで，この呼び出しでの$P_{2,1}$とFBTとの論理積の結果はFFFのことごとくの成分が0となり，*23.*で$P_{2,1}$についての処理が終了，次の小文字6の$P_{2,2}$に移る。図11，図12で二重だ円の点はすべて，このように*23.*で手順が終了していることを示す。6で$P_{2,2}$とFBTとの論理積の結果，別のFFFが生ずるが，これも5変数関数なので，再々度SUBMAPを呼び出し，$STACK=3$ とかわる。分割され$FBT(1,J)$ができるが，$FBT(2,J)$の方が全部0となるので，部分マップ2についての手順はここでは不用となる(*20.*)。図11，図12ではこのように手順*20.*で終わる点は二重四角で囲んである。

次に小文字8，9と進むが，すでに関数は4変数で$NS=NEND$になっているので，これ以上の呼び出しを止め，$P_{11,1}$の論理積でできたFFFの主項を求める。3変数x_1, x_2, x_3の関数FFFに対し，主項[010]が得られ，それに$P_{1,1}$のリテラル$x'_4=0$ を付加して[0100]が配列$XDQ(I,J)$の第1列目に記憶される。図11の小文字9の(1.1)の下の1[0100]と示したのはこの様子を表わす。図11，図12の最下部だ円の下のベクトルはすべて，このような計算過程で得られた主項で，[ ]の左外側の数字は配列XDQに蓄えられる列番号を表わしている。さて，ここで4変数の関数$FF1$（小文字6の呼び出しでみればFFF）が求まったので，リテラルを合成して，上位の呼び出しへ復帰していく（*41.*, *49.*～54)。小文字10，①の一点鎖線はこの様子を表わし，小文字6の$P_{2,2}$ のリテラル$x_5=1$ が付加され，小文字6の点に復帰する。①のベクトル[01001]は小文字10の合成過程①でこのような主項が得られたということを表わしている。図11，図12で◯番号を付した一点鎖線はすべて，合成過程を表わし，その過程で得られた主項が同じ◯番号を付けて下の適当な場所にベクトルとして並べてある。小文字6へ復帰すると，$STACK$は1減り2となる。これは小文字3の点でのSUBMAPの呼び出しが現在進行中で，いま小文字4，つまり部分マップ2の処理が終了した時点にいることを意味している。続いて小文字11，すなわち部分マップ1についての処理が行われ，以下，小文字12，13,…の順で計算処理が進行する。

ここで，例えば合成過程④，⑤，⑥，⑦について考えてみると，④では主項[0110]に2を，⑤では[0120]に1をリテラルとして追加するだけでよいが，⑥ではXDQの6列目[1001]から8列目[0000]までの主項に0をリテラルとして合成する必要がある。又更に⑦では④，⑤，⑥で得られた各主項に第6の成分リテラルとして1を合成するが，これは第4列目から第8列目までである。本プログラムではメモリ節約のため，主項の記憶，リテラルの合成，包含関係の照合を全部配列XDQで行っているため，これらの処理をXDQの第何列目から，第何列目まで行うかという情報が欠かせない。後の第何列目までの方はXDQに記憶されている主項の現在高$C2$でよいのだが，はじめの第何列目からの方の情報は⑥，⑦でみたように合成過程の局面局面で異なってくるので，特別の工夫をしている。本プログラムではこの情報を得るため二重スタック$C1D(STACK, STACK2)$ を用いている。$STACK2$は，

あるSUBMAPの呼び出しで，ある許容キューブについて主項が得られると1となり，その次の許容キューブについて又主項が得られると，1ふえて2となり，そして最高3までなりうるカウンタである。$C1D$の値は，SUBMAPがどんどん呼ばれて関数が分割され，$NS=NEND$となって4変数の主項が求められる局面*33.*で$C1$，つまり前回までに蓄えられていたXDQの中味＋1に設定され，合成過程のときは*35.*で一回先の合成過程での$C1D(STACK+1,1)$の値に設定される。図11の小文字25で$C1D(3,1)=4$となり，26をへて22で$C1D(2,1)=C1D(3,1)=4$，30で$C1D(3,1)=5$，31をへて27で$C1D(2,2)=5$，36で$C1D(3,1)=6$，38でも$C1D(3,2)=6$，39をへて33で$C1D(2,3)=6$，40をへて20では$C1D(1,2)=4$となっている。この$C1D$を用いると，合成過程のプログラム*53.*にしたがって，⑥では（このとき，$STACK=STACK-1=2$となっている）$C1D(3,1)=6$列目から，$C2=8$列目までに入っている主項にリテラル0を，又⑦では$C1D(2,1)=4$列目から$C2=8$列目までの主項にリテラル1を合成すればよい。他の合成過程の場合も$C1D(STACK+1,1)$によって同様に行われる。

最後には図12の小文字42まで計算が進み，③，⑦，⑪で得られた主項の包含関係が図9の*55.*で調べられ，図11の③—⑦—⑪と書き表わしたベクトルのように最終結果が得られる。これが図10(a)で与えた関数Fの主項である。但し，図中［ ］左外側に＊印がついているものは他の主項に包含されるので除去される。

結　　言

再帰的プログラムは図6のようにプログラムで表わすと，簡単であるが，その計算の流れを追跡し説明するのは難かしい。図7，図8，図9で示したFORTRANプログラムの計算処理手順を図11，図12のように表現して，再帰的呼び出しが行われるごとにプログラムがいかに進行していくかを示した。特に結果を蓄えておく配列$XDQ(I,J)$について，ある列からある列までの主項にリテラルを合成する操作に，二重スタック$C1D(STACK, STACK2)$が必要となり，どのような仕組で使われたかを明らかにした。本プログラムは約500ステップ，計算時間は6変数，7変数，8変数の各関数で，それぞれ，1分11秒，3分39秒，11分20秒（各100個の関数の平均）であった。使用計算機はOKITAC**70**システム**50**モデル**40**（電気工学科設置）である。

参 考 文 献

1）A.V.エイホ他；アルゴリズムの設計と解析，サイエンス社，(1977)
2）松田；富山大学工学部紀要，**32**，1，(1981)
3）松田，宮腰；富山大学工学部紀要，**31**，1，(1980)
4）宮腰，松田；信学技報，**AL78**，17，(1979)

A Recurcive Program for Generating All the Prime Implicants of Boolean Functions

Hideo MATSUDA

This paper describes a recurcive program for generaing the prime implicants of Boolean functions by utilizing the submap method. The procedure is written in FORTRAN, and the steps of this computing process which consists of two parts of decomposition and composition is explained in datail by a chart form. Especially in the composition process, it is shown that we need a double stack,which has two stack counters, for adding literals to only prime implicants stored in between from one position to the other position of a stack register.

〔英文和訳〕

ブール関数の主項を生成する一再帰的プログラムについて

松　田　秀　雄

本論文は部分マップ法をつかって，ブール関数の主項を生成する再帰的プログラムについて述べている。手続きはFORTRANで書かれ，分解と合成の２つからなるこの計算の処理過程が図式形式でくわしく説明されている。特に合成過程においては，スタックレジスタのある位置からある位置までに蓄えられている主項にのみリテラルを付加するために，スタックカウンタを２つもつ，二重スタックが必要となることを示している。

（1981年11月20日受理）