アルゴリズム特論（第３回）

北海道大学　大学院　情報科学研究科
アルゴリズム研究室　　湊　真一

前回の内容：　積和形論理式

- 積和形論理式の記述例
  - 実例
  - テーブル表現、論理式表現、CNF/DNF
  - 真理値表との比較（非固定的表現）
  - 論理関数に対する非一意性
- 計算機上での実装
  - 配列表現、リスト表現
- 論理演算アルゴリズム
  - OR, AND, NOT, EXOR
  - 冗長項除去、等価性判定、恒真性判定
- 積和形論理式のデータ量
  - コンパクトな例、不利な例
  - 現実的に扱える範囲

今回の内容

積和形論理式の最小化・簡単化

- 簡単化・最小化・非冗長化
- カルノー図と積和形論理式との対応
  - 最小項、内項、素項、必須素項
- 最小化（Q-M法）
  - アルゴリズム概要と計算量
- 実用的簡単化
  - MINI, Espresso
  - ベンチマーク集合
- 非冗長化（Morreale法）
  - アルゴリズム概要
  - BDDとの組合せ

積和形の非一意性

- 積和形論理式は一般の論理式よりも制約が加えられているが、それでもまだ自由度がある。

（例）　$F = x\,\tilde{y} + x\,z + \tilde{x}\,y + \tilde{x}\,\tilde{z}$

$= x\,\tilde{y} + \tilde{x}\,y + y\,z + \tilde{y}\,\tilde{z}$

$= x\,\tilde{y} + \tilde{x}\,\tilde{z} + y\,z$

同じ論理関数が異なる積和形で表現できる場合がある

→　簡単化・最小化が必要となる。

積和形の論理演算中の冗長項の発生

- 積和形論理式の処理では、演算の途中で冗長項が発生しやすい。一旦発生するとますます増える。
  - ときどき、冗長項を除去する必要がある。
  - 完全に除去するには積和形の最小化・簡単化アルゴリズムを適用するが、非常に時間がかかる。
    - 式の長さの指数またはそれ以上の処理時間
  - 簡単に見つかる冗長項だけを除去する方法はいくつかある。
    - 簡単化規則を繰り返し適用する
    - 式の長さの2乗程度の処理時間
    - 必ずしも最小形にならない

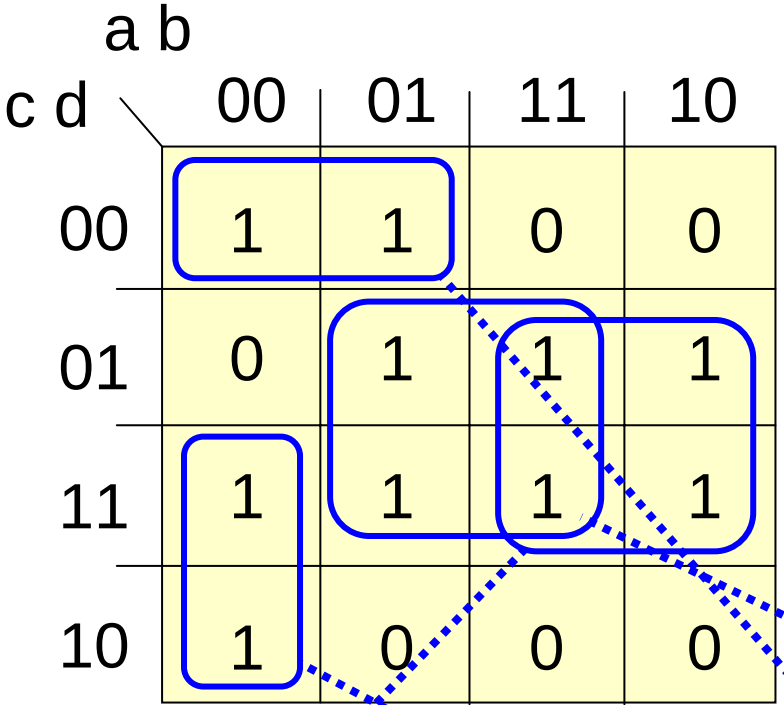
（簡単化規則の例）

F + F = F
F + ~F = 1
x x = x
x ~x = 0
x + ~x F = x + F
x F + ~x = F + ~x

積和形の簡単化・最小化・非冗長化

- 簡単化 (Simplification)
  - 必ずしも最小形でなくても、簡単になればよい。
  - 答は1通りとは限らない
- 最小化 (Minimization)
  - 同じ論理を表す積和形の中で、積項数（または総リテラル数）が最も少ないものを見つけること
  - 1つの論理関数に対して最小形が複数個存在する場合もある
  - 少し複雑な論理式になると非常に時間がかかる
- 非冗長化 (Irredundant SOP)
  - 冗長な項やリテラルを含まない論理式を見つけること
  - 最小形は必ず非冗長形であるが、
    非冗長形でも最小形でない場合がある。
  - 1つの論理関数に対して非冗長形は一般に複数存在する。
    （その中に最小形も含まれる）

カルノー図と積和形論理式

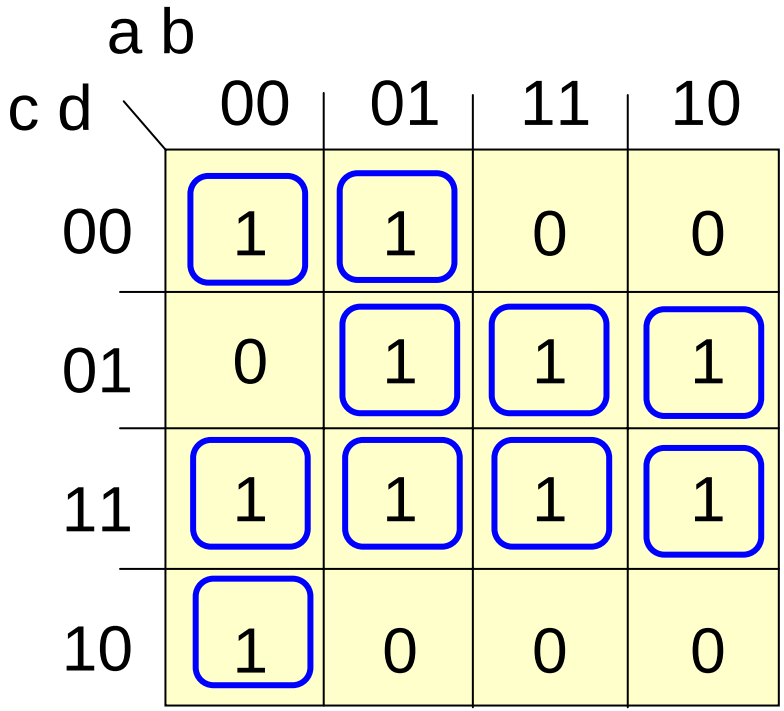


- 各積項はカルノー図の中のループで囲んだ領域に対応する。
- なるべく少ない個数のループで全ての”1”のコマを囲むことができれば、積項数最小となる。
- 大きなループで囲むほど、その積項のリテラル数が少なくなる。

F = a d + ~a ~b c + ~a ~c ~d + b d

計算機上ではカルノー図は使わずに、積和形データを直接操作するが、話をわかりやすくするため、講義ではカルノー図と対比して説明する。

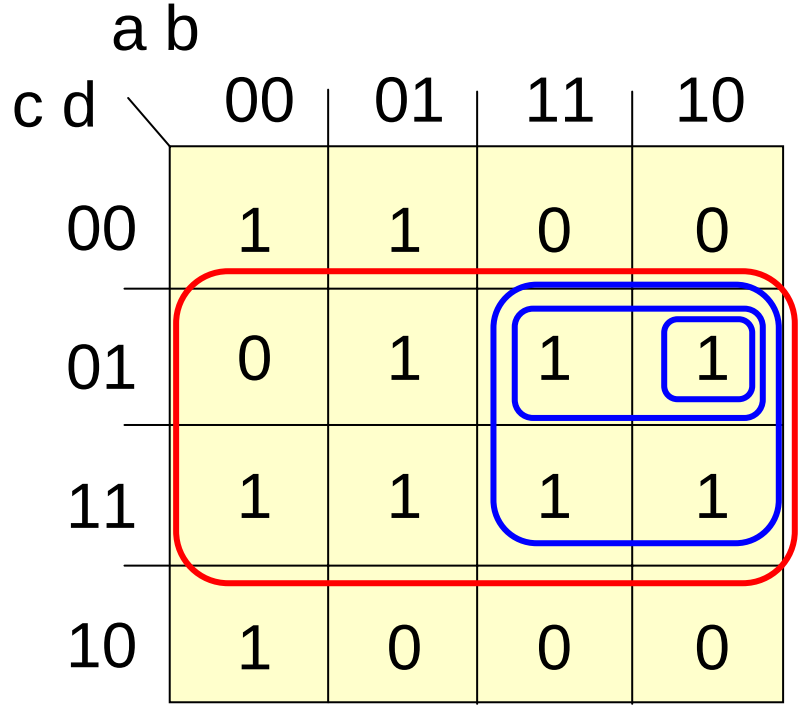
最小項(minterm)展開（積和標準形）



- “1”のコマを１個だけ囲む積項のことを「最小項」(Minterm)と呼ぶ。
- 全ての最小項を集めた積和形論理式のことを「最小項展開」または（積和標準形）と呼ぶ。
- 最小項展開は（ある順序でソートすれば）論理関数に対して一意な形になる。
  - “1”のコマの個数に比例する積項数
  - 各積項のリテラル数は最大

F = a b c d + a b ~c d + a ~b c d + a b ~c d
+ ~a b c d + ~a b ~c d + ~a b ~c ~d
+ ~a ~b c d + ~a ~b c ~d + ~a ~b ~c ~d

内項(implicant)と素項(prime implicant)



内項: a ~b ~c d

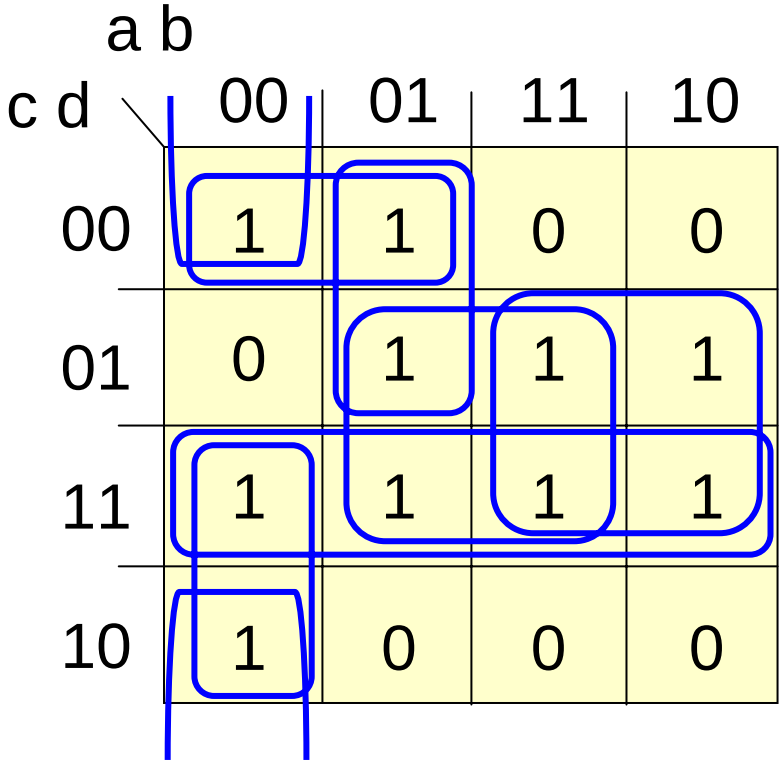
a ~c d

a d

× d

- 論理関数Fの”1”の領域に含まれる積項のことを、Fの「内項」(implicant)と呼ぶ。
  - “0”の領域にはみ出す積項は内項ではない
- 論理関数Fを表す積和形論理式の各積項は、Fの内項の1つである。
  - 最小項は必ず内項である。
- 特に、これ以上大きくできない内項のことを「素項」(prime implicant)と呼ぶ。
  - 主項と呼ぶ教科書もある
- 積和形論理式の簡単化とは、なるべく少ない個数の素項で全ての”1”をカバーする問題である。

素項と必須素項(essential prime implicant)



- 素項は一般に多数存在する。
  - 全ての素項を使わなくても論理関数をカバーできる。
- ある”1”のコマについて、ただ1つの素項にのみカバーされている場合、その素項は必ず使わないといけない。そのような素項を「必須素項」(essential prime implicant)と呼ぶ。
  - 必須素項は必ず存在するとは限らない。論理関数に依存する。
  - 必須素項が多く見つかれば、最小化問題が簡単になる。

Quine-McClusky法

- 1950年代に作られた最小化アルゴリズム
  - 基本的に手作業で行うことを想定して作られた
- アルゴリズムの概要:

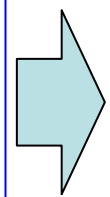
(1) すべての素項を作り、素項のテーブルを作る
- 論理関数を最小項展開して全ての積項を列挙したテーブルを作る
- 積項同士を比較して、隣接する積項を見つけて併合した積項を作る
- 併合できなくなったら、残った積項が素項である
- 素項テーブルは、縦軸に素項を並べ、横軸に最小項を並べる
- 各行について、素項がカバーしている最小項にチェックを入れる

(2) 素項テーブルの最小カバーを求める
- 必須行を求める。縦の列でチェックがただ1つしかついていない列があれば、そのチェックがついている行が必須行となる。必須行があればそれを除去し、必須行にカバーされた列も除去する。
- 被支配行求める。ある行sのチェックが別の行tにすべて含まれるとき、sを被支配行と呼び、これを取り除く。(sを選ぶくらいならtを選んだほうが良いから)
- 支配列を求める。ある列uのチェックが別の列vに全て含まれるとき、vを支配列と呼び、これを取り除く。(uさえカバーすればvも必ずカバーされるから)
- 必須行、被支配行、支配列を繰り返し除去して残った行列をcyclic coreと呼ぶ。あとは、原理的には全ての組合せを試すしかない。

Q-M法の実行例(1)

c d \ a b	00	01	11	10
00	0 1	4 1	12 0	8 0
01	1 0	5 1	13 1	9 1
11	3 1	7 1	15 1	11 1
10	2 1	6 0	14 0	10 0

最小項	[a b c d]
0	[0 0 0 0]
2	[0 0 1 0]
3	[0 0 1 1]
4	[0 1 0 0]
5	[0 1 0 1]
7	[0 1 1 1]
9	[1 0 0 1]
11	[1 0 1 1]
13	[1 1 0 1]
15	[1 1 1 1]



最小項	[a b c d]
0,4	[0 - 0 0]
0,2	[0 0 - 0]
2,3	[0 0 1 -]
3,7,11,15	[- - 1 1]
4,5	[0 1 0 -]
5,7,13,15	[- 1 - 1]
9,11,13,15	[1 - - 1]

Q-M法の実行例(2)

最小項	[a b c d]	
0,4	[0 - 0 0]	A
0,2	[0 0 - 0]	B
2,3	[0 0 1 -]	C
3,7,11,15	[- - 1 1]	D
4,5	[0 1 0 -]	E
5,7,13,15	[- 1 - 1]	F
9,11,13,15	[1 - - 1]	G

c d \ a b	00	01	11	10
00	0 1	4 1	12 0	8 0
01	1 0	5 1	13 1	9 1
11	3 1	7 1	15 1	11 1
10	2 1	6 0	14 0	10 0

	0	2	3	4	5	7	9	11	13	15
A	X			X						
B	X	X								
C		X	X							
D			X			X		X		X
E				X	X					
F					X	X			X	X
G							X	X	X	X

	0	2	3	4	5	7
A	X			X		
B	X	X				
C		X	X			
D			X			X
E				X	X	
F					X	X

(A,C,F,G)
(B,D,E,G)

最小カバー問題の別の解法

	0	2	3	4	5	7
A	X			X		
B	X	X				
C		X	X			
D			X			X
E				X	X	
F					X	X

P = (A+B)(B+C)(C+D)(A+E)(E+F)(D+F)

- 論理関数 Pが1になるようなA,B,C,D,E,Fの組合せの中で、最も変数の数（または各変数のコストの総和）が小さくなる組合せを求めればよい
- Pのような関数をペトリック関数と呼ぶ。
- Pを展開して冗長な項を取り除くと、解が見つかる
  - といっても、Pの展開形を簡単化するのが難しい場合がある。いつもうまく行くとは限らない。

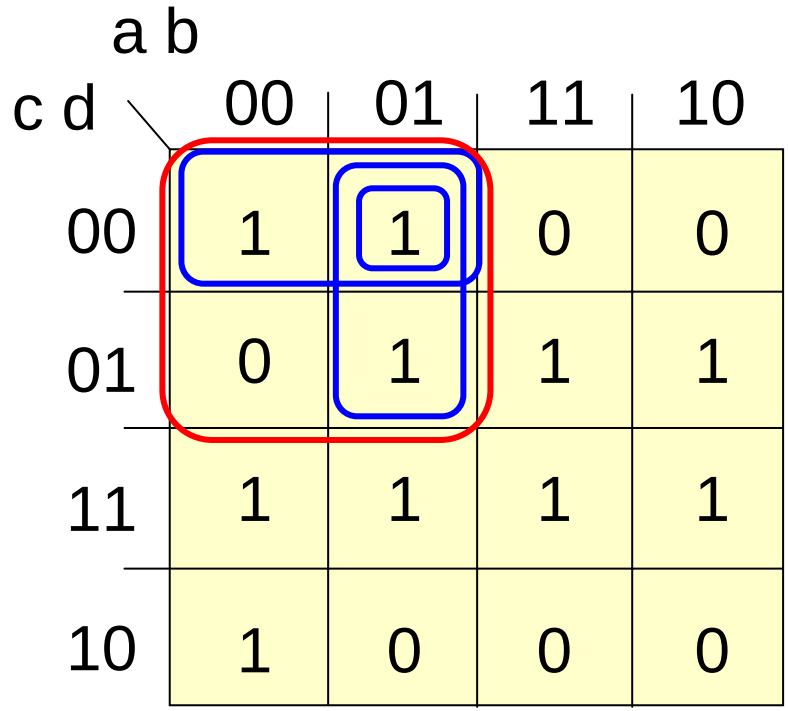
P = (A+B)(B+C)(C+D)(A+E)(E+F)(D+F)
= A B D F + A C D E + A C F + B C E F + B D E

最小カバー問題は、論理最小化の他にも多くの応用がある重要問題

高速簡単化アルゴリズム

- Q-M法は必ず最小形を求めることができるが、少し大規模な論理になると非常に時間がかかる
  - 生成される素項の数が多くなりすぎる。
  - 素項を作れたとしても、最小カバー問題を解くのに時間がかかる。
- 厳密に最小でなくてもよいから、限られた時間内にそこそこ品質の良い簡単化をできるアルゴリズムが研究されてきた。
  - 1970年代～80年代にかけて盛んに研究された
  - 与えられた積和形論理式の各積項の変形を試行錯誤的に繰り返して、簡単化を行う方法（PRESTO, MINI, ESPRESSO等）が一般的
  - BDDで与えた論理関数から 非冗長な積和形を直接生成する方法（Morreale-Minato)もある。

積項の操作(1)「拡大」



内項: ~a b ~c ~d

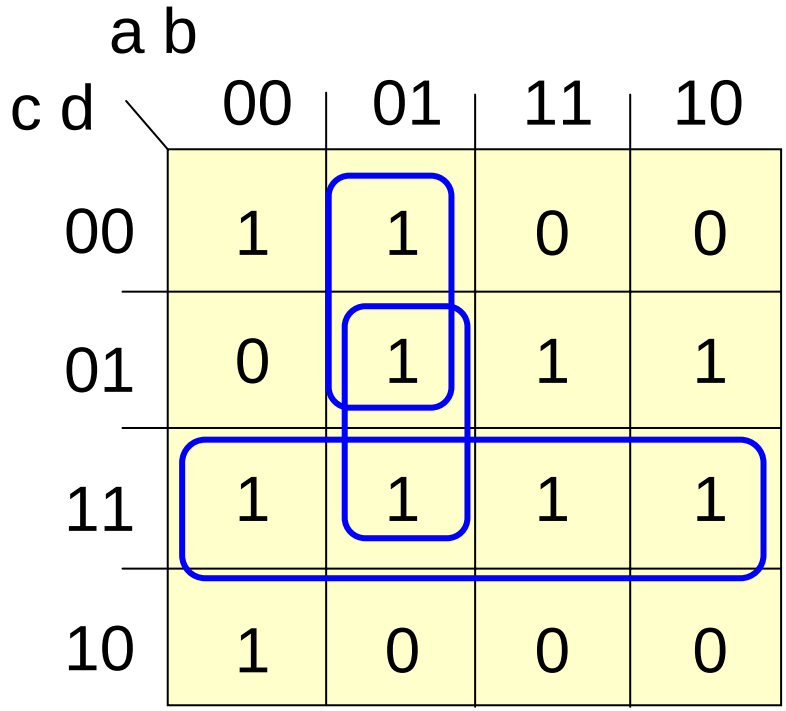
~a b ~c

~a ~c ~d

× ~a ~c

- 与えられた積項のループを、論理に影響を変えない範囲で拡大する操作
  - 積項のリテラルの1つを取り除くことと同じ
  - どのリテラルを取り除くかで拡大する方向が異なる
- ある内項の拡大操作を繰り返すと、素項になる。
- 拡大操作を繰り返すと隣の積項と重なりが発生する

積項の操作(2)「除去」



~a b d

~a b ~c

c d

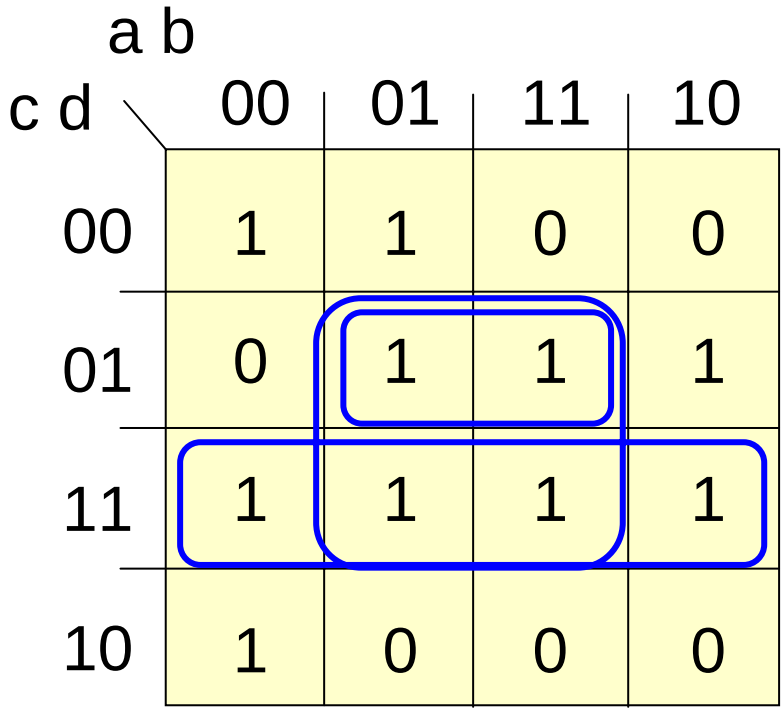
- 与えられた積項が別の積項にカバーされているときに、その積項を取り除く操作
  - 1つの積項にカバーされるときは比較的簡単だが、2つ以上の積項にカバーされている積項を除去するのはやや難しい
- 全ての積項を「拡大」し、不要な積項を全て「除去」すると非冗長な積和形が得られる。
  - 積項の数が多いと計算時間がかかる（総リテラル数の多項式時間）

非冗長積和形(Irredundant SOP)

- どの積項も素項である
  - どのリテラルを消しても論理が変わる
- どの積項1つとっても除去できない
  - どの積項1つ消しても論理が変わる
  - ただし、積項を2つ削除して別の積項1つ足すと、
    もっと小さくできる可能性はある。
- 最小とは限らないが、比較的最小に近い形である。
  - 局所最小解となる

(例)　F = x ~y + x z + ~x y + ~x ~z
= x ~y + ~x y + y z + ~y ~z
= x ~y + ~x ~z + y z

積項の操作(3)「縮小」



- 積項のループを小さくする操作
  - ただし、積和形論理式がカバーする論理を変えない範囲で小さくする
  - 積項にリテラルを追加するのと同じ
  - どのリテラルを追加するかで、縮小する方向が異なる
- 局所最小解（非冗長形）から脱出してより小さい解に到達するために用いる
  - 総リテラル数が一時的に増加する
  - 縮小したのと別の方向に拡大を試みることで、探索の向きを変える

積和形簡単化プログラムの性能

- PRESTO, MINI, ESPRESSO等の簡単化プログラムは、最小形への到達を保証しないが、比較的高速に最小に近い積和形を生成する。
- 積項の「拡大」「併合」「除去」「縮小」を組合せて、さらに必須素項や最小カバー問題の技術も加えて品質と速度を向上させている。

（例）ESPRESSOの実行例（SPARC 2）

論理関数名	入力数	出力数	積項数	総リテラル数	処理時間(秒)
sel8	12	2	17	90	0.2
enc8	9	4	15	51	0.3
add4	9	5	135	819	1.9
add8	17	9	2519	24211	443.1
mult4	8	8	130	874	5.0
mult6	12	12	1893	19340	1126.2

非冗長積和形を直接生成する方法

- Morreale [1970]のアルゴリズム
  - さらにBDDと組合せた高速化方法 - Minato[1993]

アルゴリズムの基本となる展開式:

$$isop = (v \cdot isop_1) + (\sim v \cdot isop_0) + isop_d$$

ある変数vに着目すると、非冗長積和形isopの各積項は、
vを含む積項$isop_1$、~vを含む積項$isop_0$、vに依存しない積項$isop_d$
の3つに分けられ、それぞれが、やはり非冗長積和形となる。

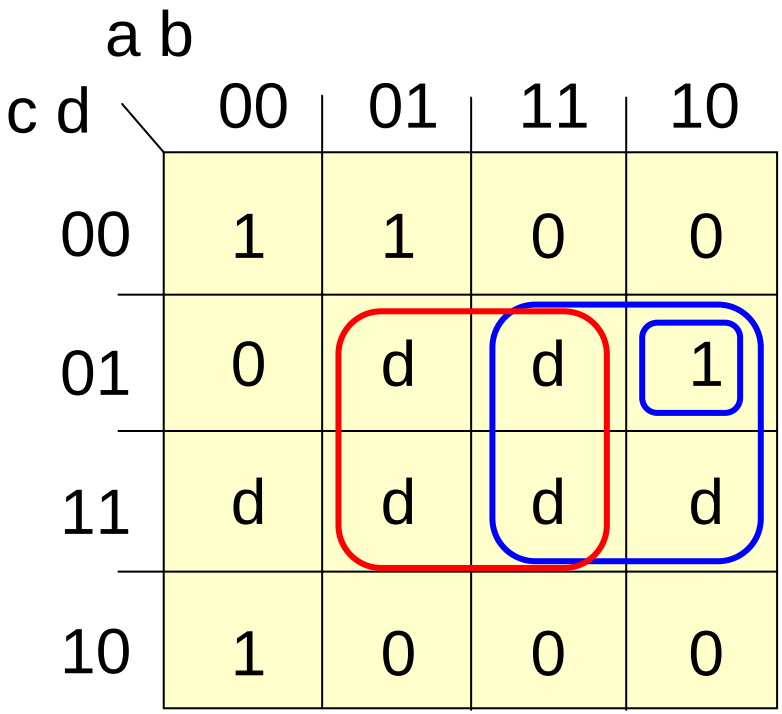
(1) 論理関数Fに関する変数の1つ vを選ぶ。
(2) FをF_1(v=1を代入)とF_0(v=0を代入)の2つの部分関数に分ける。
(3) F_1とF_0がともに"1"になる部分は$isop_d$でカバーできるのでドントケアとして、
どうしてもv, ~vが必要な部分について$isop_1$,$isop_0$を再帰的に求める。
(4) $isop_1$, $isop_0$にカバーされた部分をドントケアとして、$isop_d$を再帰的に求める。

ドントケア(don’t care)を含む論理関数

c d \ a b	00	01	11	10
00	1	1	0	0
01	0	d	d	1
11	d	d	d	d
10	1	0	0	0

- 入力変数の組合せの一部にドントケア”d”を含む論理関数
「不完全指定論理関数」
(incompletely specified logic)
  - “d” の部分は0でも1でも良い
  - ドントケアをうまく使うと、論理式が簡単に表せる場合がある。
- ドントケアを含む論理関数を積和形データとして扱う場合、
2つの積項リストの組
(onset, dcset)で表すことが多い。
  - onset: “1”になる部分を表す
  - dcset: “d”になる部分を表す

ドントケアを含む内項・素項



- “d”の部分は積項に含んでも良い。
  - より大きなループで囲める
- “d”だけをカバーする積項は無駄なので除去してよい。

Morreale法の実行例

[step1]

c d \ a b	00	01	11	10
00	1	1	0	0
01	0	1	1	1
11	1	1	1	1
10	1	0	0	0

$F_0(a=0)$ $F_1(a=1)$

[step2]

c d \ a b	00	01	11	10
00	1	1	0	0
01	0	d	d	1
11	d	d	d	d
10	1	0	0	0

$F_0(a=0)$ $F_1(a=1)$

$isop_0$

$isop_1$

[step3]

c d \ a b	00	01	11	10
00	d	d	0	0
01	0	1	d	d
11	d	1	d	d
10	d	0	0	0

[step4]

c d \ a b	00	01	11	10
00	d	d	0	0
01	0	1	d	d
11	d	1	d	d
10	d	0	0	0

$isop_d$

ESPRESSOとMorreale法との比較実験結果

- 実用的な論理関数を集めたベンチマーク問題で比較
- Morreale法はBDDを使った高速版（Morreale-Minato法）で実験

		ESPRESSO		Morreale-Minato	
関数名	入力数	総リテラル数	時間(秒)	総リテラル数	時間(秒)
mult4	8	874	5.0	945	1.4
mult6	12	19340	1126.2	22274	26.7
achil8p	24	32	2.0	32	0.2
achil8n	24	59049	3512.7	59049	8.7
5xpl	7	347	1.5	366	0.8
9sym	9	609	10.7	1036	0.9
alupla	25	26632	257.3	26734	20.5
bw	5	429	1.4	374	1.1
duke2	22	1036	28.8	1296	3.2
vg2	25	914	42.8	914	1.9
c432	36	time out (>10h)		969037	1744.8
c880	60	time out (>10h)		1986014	1096.6

(計算時間はSPARC 2)

最小化・簡単化アルゴリズムの比較

- Q-M法
  - 最小カバー問題は素項の数に対してNP
  - 素項の数は最悪指数（→　指数の指数）
- MINI, Esspresso
  - 10～20変数、数千リテラルまで現実的に扱える
  - 厳密な計算量は不明
- Morrealeの方法
  - BDD（来週以降説明）と組合せることで
    非常に高速なプログラムを実現
    （Morreale-Minato 法）
  - 50~100変数、百万リテラル程度まで現実的に扱える
  - 厳密な計算量は不明

今回の内容

積和形論理式の最小化・簡単化

- 簡単化・最小化・非冗長化
- カルノー図と積和形論理式との対応
  - 最小項、内項、素項、必須素項
- 最小化（Q-M法）
  - アルゴリズム概要と計算量
- 実用的簡単化
  - MINI, Espresso
  - ベンチマーク集合
- 非冗長化（Morreale法）
  - アルゴリズム概要
  - BDDとの組合せ