2018

Minimalizacja funkcji boolowskich

TADEUSZ ŁUBA

FUNKCJA BOOLOWSKA , UKŁADY KOMBINACYJNE, MIMIMALIZACJA, REDUKCJA ARGUMENTÓW, DEKOMPOZYCJA FUNKCJONALNA I LINIOWA, ANALIZA DANYCH, REDUKCJA ATRYBUTÓW

Rewolucyjny rozwój technologii mikroelektronicznych spowodował, że projektowanie systemów cyfrowych może być realizowane wyłącznie za pomocą komputerowych narzędzi projektowania przystosowanych do przetwarzania dużych ilości różnorodnych danych. Celem materiałów dydaktycznych jest omówienie zaawansowanych metod syntezy logicznej niezbędnych zarówno w projektowaniu systemów cyfrowych jak też w analizie i eksploracji danych. Dlatego głównymi zagadnieniami omawianymi w materiałach są metody minimalizacji i dekompozycji funkcji boolowskich, jak też redukcja atrybutów i indukcja reguł decyzyjnych. Takie ujęcie tych zagadnień jest zgodne z pilną potrzebą ważnych zastosowań takich jak: dystrybucja adresów IP, skanowanie wirusów, wykrywanie niepożądanych danych, itp. Nie mniejsze potrzeby stosowania zaawansowanych metod syntezy logicznej dotyczą analizy i eksploracji danych. Inaczej mówiąc celem tego podręcznika jest przygotowanie przyszłych inżynierów do umiejętnego wykorzystania ogromnego potencjału syntezy logicznej o czym świadczą wyniki prezentowanych metod i algorytmów.

Spis treści

1 Minimalizacja funkcji boolowskich

1.1 Funkcje boolowskie

Układ kombinacyjny jest podstawowym układem logicznym umożliwiającym realizację funkcji boolowskich. Układ kombinacyjny konstruujemy z elementów logicznych po to, aby realizować funkcje lub ich zespoły opisujące bardziej skomplikowane układy cyfrowe. Dlatego rozważania o układach kombinacyjnych rozpoczynamy od pojęcia funkcji boolowskiej.

Pojęcie funkcji boolowskiej jest pojęciem podstawowym umożliwiającym modelowanie zjawisk fizycznych reprezentowanych jako odwzorowanie ciągów (wektorów) binarnych należących do zbioru ***X*** w ciągi binarne (wektory) ze zbioru ***Y***, gdzie zbiory ***X***, (***Y***) są podzbiorami *n*-krotnego, (*m*-krotnego) iloczynu kartezjańskiego zbioru ***B*** = {0, 1}.

Formalnie funkcją boolowską zmiennych binarnych $x_1, \ldots, x_n$ nazywamy odwzorowanie:

f: $\boldsymbol{X} \rightarrow \boldsymbol{Y}$, gdzie $\boldsymbol{X} \subseteq \boldsymbol{B}^n$, $\boldsymbol{Y} \subseteq \boldsymbol{B}^m$.

Jeżeli $\boldsymbol{X} = \boldsymbol{B}^n$, to funkcję taką nazywamy zupełną; w przeciwnym przypadku jest to funkcja niezupełna, zwana również funkcją nie w pełni określoną.

Najczęściej stosowane reprezentacje funkcji boolowskich to tablica prawdy oraz formuła (wyrażenie) boolowskie.

Funkcja f może być przedstawiona w postaci tablicy prawdy. Jest to tablica o n+1 kolumnach i 2^n wierszach. W kolejnych wierszach są zapisywane wszystkie wartości ciągu $x_1, \ldots, x_n$, czyli wszystkie wektory ***x***. W ostatniej kolumnie podana jest wartość y przyporządkowywana danemu wektorowi lub „–”, jeżeli funkcja dla tego wektora nie jest określona. Kolejne wektory są numerowane, przy czym wartość i podana z lewej strony w dodatkowej kolumnie jest dziesiętnym odpowiednikiem wektora ***x*** traktowanego jako liczba w zapisie dwójkowym:

$$A_D = L(A_{NKB}) = \sum_{j=0}^{n-1} a_j \cdot 2^j = a_{n-1} \cdot 2^{n-1} + \cdots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

Na przykład przeliczenia liczb binarnych $(0101)_B$ oraz $(1010)_B$ (podanych w NKB – naturalnym kodzie binarnym) na liczby dziesiętne 5_D i 10_D dokonujemy następująco:

$(0101)_B = 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5_D$

$(1010)_B = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 10_D$

Przykłady tablicowej reprezentacji funkcji boolowskiej podane są w tabl. 2.1a (funkcja zupełna) i tabl. 2.1b – funkcja niezupełna.

Tablica 2.1

a)

	x_1	x_2	x_3	f
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

b)

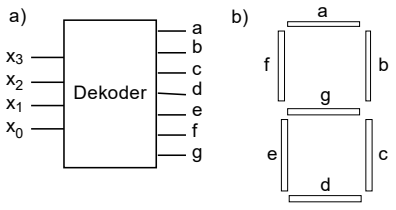
	x_1	x_2	x_3	f
0	0	0	0	0
1	0	0	1	1
2	0	1	0	–
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	–
7	1	1	1	1

Tablice te specyfikują funkcje boolowskie, których wektory wejściowe są dodatkowo określone liczbami dziesiętnymi. Stąd pomysł, aby specyfikacje funkcji boolowskich podawać w uproszczonej formie. Dla funkcji z tabl. 2.1a odpowiedni zapis powinien być:

$$f = \Sigma(1, 3, 5, 6, 7)$$

Natomiast funkcję z tabl. 2.1b zapisać należy z dodatkowym specyfikowaniem wektorów nieokreślonych:

$$f = \Sigma[1, 3, 5, 7, (2, 6)].$$



Rys. 2.1. Dekoder (a) wskaźnika siedmiosegmentowego (b)

Typowym zastosowaniem tablic prawdy do specyfikacji funkcji boolowskich może być opis dekodera wskaźnika siedmiosegmentowego powszechnie stosowanego do wyświetlania cyfr w urządzeniach pomiarowych i monitorujących. Dekoder taki to układ kombinacyjny o wejściach x_3, x_2, x_1, x_0 oraz wyjściach a, b, c, ..., g (rys. 2.1a), odpowiadających segmentom wskaźnika (rys. 2.1b). Na wejścia x podawane są liczby binarne 0000, 0001,...,1000,1001 (dziesiętnie: 0,1,...,8,9). Wyjścia a, b, c, ..., g sterują diodami świecącymi. Dioda zostaje podświetlona, gdy odpowiednie wyjście jest w stanie 1. W rezultacie działanie dekodera jest opisane tablicą prawdy, podaną w tablicy 2.2. W tym przypadku mamy do czynienia z funkcjami niezupełnymi. Jak się później przekonamy, omówiony dekoder nie nastręcza żadnych trudności w syntezie odpowiadającego im układu kombinacyjnego.

Tablica 2.2

	x_3	x_2	x_1	x_0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
0	1	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Można jednak podać zastosowania bardziej skomplikowane. Na przykład układ kryptograficzny realizujący tzw. algorytm DES (*Data Encryption Standard*), jest wyposażony w klucze *S*, których działanie specyfikuje się funkcjami boolowskimi o 6 wejściach i 4 wyjściach. Przykładowa specyfikacja dla klucza S_1 jest podana w tablicy 2.3.

Tablica 2.3

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Jeśli wejścia klucza S_1 oznaczymy $x_1,...,x_6$, a wyjścia y_1, y_2, y_3, y_4, to zapis w tablicy 2.3 rozumiemy następująco: dla wektora $\boldsymbol{x}$ = (000000) wyjścia y są liczbą binarną, której zapis dziesiętny jest L = 14 (czyli 1110), dla $\boldsymbol{x}$ = (000001), L = 4, wreszcie dla $\boldsymbol{x}$ = (111111), L = 13.

Funkcje boolowskie reprezentowane odwzorowaniem f, jakkolwiek możliwe do bezpośredniej realizacji technicznej, nie są najlepszą formą do zastosowań w strukturach bramkowych. Znacznie wygodniejsze w tym przypadku są reprezentacje funkcji w postaci wyrażeń boolowskich (formuł boolowskich). Ich zaleta wynika przede wszystkim z łatwej realizacji elementów logicznych zwanych bramkami logicznymi, które to elementy stanowią naturalną realizację wyrażeń boolowskich, gdzie występują w postaci operatorów.

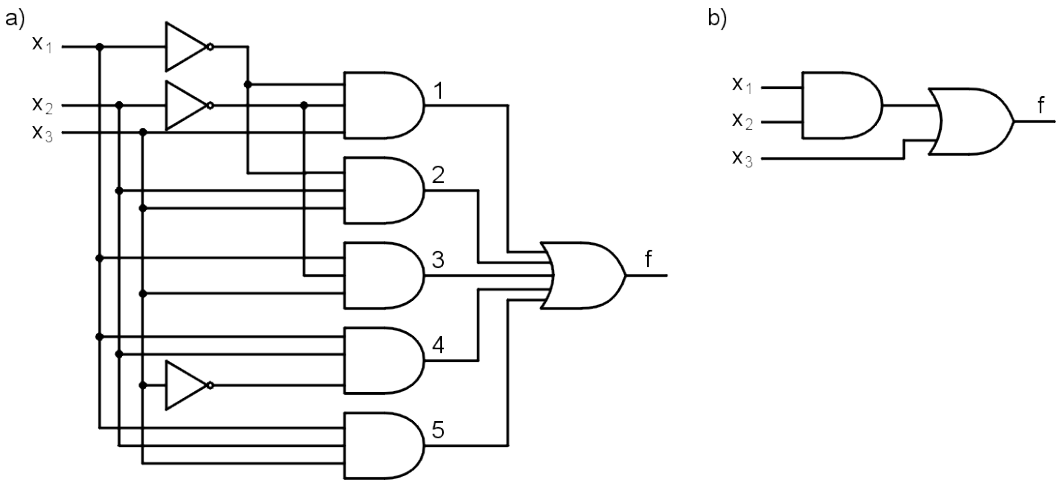
Dla funkcji opisanej tablicą prawdy podaną w tabl. 2.1a odpowiednia formuła boolowska jest następująca:

$$f = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3 + x_1x_2x_3$$

a jej realizacja na bramkach AND, OR, NOT pokazana jest na rys. 2.2.

W układzie kombinacyjnym z rys. 2.2 funkcja *f*, realizowana na jego wyjściu *f*, reprezentuje odwzorowanie z tabelki prawdy, co łatwo sprawdzić wprowadzając na wejścia układu odpowiednie wektory binarne i obliczając wartość uzyskaną na wyjściu *y*. Na przykład dla $x_1 = x_2 = 0$, $x_3 = 1$ na wyjściu bramki AND1 pojawi się sygnał o wartości 1, i w rezultacie wyjście *y* przyjmie wartość 1.

Natomiast dla $x_1 = x_2 = x_3 = 0$ na wyjściach wszystkich bramek AND będzie 0, a więc jednocześnie *y* przyjmie wartość 0, co jest zgodne z tablicą prawdy.



Rys. 2.2. Realizacja funkcji *f*: a) bezpośrednio, b) po uproszczeniu wg zasad algebry Boole'a

Niekwestionowaną zaletą formuł boolowskich jest możliwość ich upraszczania, a co zatem idzie możliwość uzyskiwania realizacji oszczędniejszych z punktu widzenia liczby bramek. Zasady formalne upraszczania formuł boolowskich związane są z prawami i własnościami algebry Boole'a. Stosując prawa algebry Boole'a, poprzednio podane wyrażenie na *f* można uprościć w sposób podany w następującym przykładzie.

PRZYKŁAD 2.1

$$f = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3 + x_1x_2x_3 =$$
$$= \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3 + x_1x_2x_3 + x_1x_2x_3 =$$
$$= \bar{x}_1x_3(\bar{x}_2 + x_2) + x_1x_3(\bar{x}_2 + x_2) + x_1x_2(\bar{x}_3 + x_3) = \bar{x}_1x_3 + x_1x_3 + x_1x_2 =$$
$$= x_3(\bar{x}_1 + x_1) + x_1x_2 = x_3 + x_1x_2$$

Ostatecznie wyrażenie to można zrealizować w układzie kombinacyjnym, którego struktura – znacznie prostsza od poprzedniej realizacji – jest pokazana na rys. 2.2b.

Sens fizyczny minimalizacji i jej ogromne znaczenie praktyczne wynika z faktu, że oba układy: pierwotny i zminimalizowany działają identycznie. Zatem upraszczając wyrażenia boolowskie będziemy mogli jednocześnie uprościć ich realizację, np. zmniejszyć liczbę zastosowanych bramek co decyduje o

kosztach realizacji i tym samym jest głównym czynnikiem zwiększającym konkurencyjność naszego produktu na rynku.

Zasygnalizowany tu proces upraszczania wyrażeń boolowskich ma ogromne znaczenie praktyczne i opracowano dla jego potrzeb wiele zaawansowanych metod syntezy, które z technicznego punktu widzenia nazywa się metodami *minimalizacji funkcji boolowskich*. Wiele z nich doczekało się realizacji w postaci zaawansowanych narzędzi komputerowych i stanowi podstawę nowoczesnej syntezy logicznej.

1.2 Formy kanoniczne funkcji boolowskich

Niech wektorowi $\boldsymbol{w} = (x_1, \ldots, x_j, \ldots, x_n)$ odpowiada następujące wyrażenie zapisane w formie iloczynu zmiennych prostych i zanegowanych:

$$x_1^{e_1} \ldots x_j^{e_j} \ldots x_n^{e_n},$$

gdzie $x_j^{e_j} = \begin{cases} \bar{x}_j & dla\ e_j = 0 \\ x_j & dla\ e_j = 1 \end{cases}$

Oznacza to, że składowej 0 wektora odpowiada w iloczynie zmienna zanegowana, a składowej 1 – zmienna prosta. Iloczyn taki nazywamy iloczynem pełnym, gdyż zawiera on wszystkie zmienne. Iloczyn pełny przyjmuje wartość 1 tylko dla tych wektorów w tablicy prawdy funkcji, dla których wartość funkcji jest 1; dla innych wektorów wartość jego wynosi 0. Iloczyn pełny będziemy także oznaczać P_i.

Zbiorowi wektorów $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_i, \ldots, \boldsymbol{w}_r\} \subseteq \{0,1\}^n$, gdzie dla każdego i, $f(\boldsymbol{w}_i)=1$, odpowiada wyrażenie:

$$F(x_1, \ldots, x_n) = \sum_{i=0}^{2^n-1} P_i f(i)$$

Jest to tzw. kanoniczna postać sumy. Stanowi ona sumę wszystkich iloczynów pełnych mnożonych przez wartość funkcji dla kombinacji i. W wyrażeniu tym pozostają w efekcie te iloczyny (tzw. mintermy), którym odpowiada wartość funkcji 1; iloczyny, którym odpowiada wartość funkcji 0, znikają. Kanoniczną postać sumy można utworzyć bezpośrednio z tablicy prawdy przez wybranie wierszy, dla których wartość funkcji wynosi 1, utworzenie dla każdego takiego wiersza iloczynu pełnego, oraz utworzenie sumy iloczynów pełnych.

Kanoniczna postać sumy funkcji z tabl. 2.1a jest zatem następująca:

$$f = \bar{x}_1\bar{x}_2x_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1x_2\bar{x}_3 + x_1x_2x_3$$

Postępowanie dualne prowadzi do tzw. kanonicznej postaci iloczynu. Wektorowi $\boldsymbol{w}_i = (x_1, \ldots, x_j, \ldots, x_n)$, gdzie $f(\boldsymbol{w}_i) = 0$, odpowiada następujące suma logiczna:

$$x_1^{e_1} + \cdots + x_j^{e_j} + \cdots + x_n^{e_n},$$

gdzie $x_j^{e_j} = \begin{cases} x_j & dla\ e_j = 0 \\ \bar{x}_j & dla\ e_j = 1 \end{cases}$

Oznacza to, że składowej 0 wektora odpowiada w sumie zmienna prosta, a składowej 1 – zmienna zanegowana. Suma taka nazywa się sumą pełną, gdyż zawiera ona wszystkie zmienne w postaci prostej lub zanegowanej. Suma pełna przyjmuje wartość 0 dla wektora $\boldsymbol{x}_i$; dla innych wektorów wartość jej wynosi 1. Sumę pełną będziemy także oznaczać S_i.

Zbiorowi wektorów $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_r\} \subseteq \{0,1\}^n$, gdzie dla każdego i, $f(\boldsymbol{x}_i) = 0$, odpowiada iloczyn:

$$F(x_1, \ldots, x_n) = \prod_{i=0}^{2^n-1} \left(S_i + f(i)\right)$$

Wyrażenie to stanowi iloczyn zawierający wszystkie sumy pełne z dodanymi wartościami funkcji dla kombinacji i. Te sumy S_i, dla których $f(i) = 1$, znikają; zostają sumy (tzw. maxtermy) z $f(i) = 0$.

Jest to tzw. kanoniczna postać iloczynu, której interpretacja za pomocą funkcji z tabl. 2.1a jest następująca:

$$f = (x_1 + x_2 + x_3)(x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + x_3)$$

W tym przypadku synteza funkcji sprowadza się do wybrania wierszy, dla których wartość funkcji wynosi 0, utworzenia dla każdego takiego wiersza sumy pełnej oraz utworzenia iloczynu sum pełnych.

Tablica 2.4

i	P_i	$x_1\ x_2\ x_3$	S_i
0	$\bar{x}_1\bar{x}_2\bar{x}_3$	0 0 0	$x_1+x_2+x_3$
1	$\bar{x}_1\bar{x}_2x_3$	0 0 1	$x_1+x_2+\bar{x}_3$
2	$\bar{x}_1x_2\bar{x}_3$	0 1 0	$x_1+\bar{x}_2+x_3$
3	$\bar{x}_1x_2x_3$	0 1 1	$x_1+\bar{x}_2+\bar{x}_3$
4	$x_1\bar{x}_2\bar{x}_3$	1 0 0	$\bar{x}_1+x_2+x_3$
5	$x_1\bar{x}_2x_3$	1 0 1	$\bar{x}_1+x_2+\bar{x}_3$
6	$x_1x_2\bar{x}_3$	1 1 0	$\bar{x}_1+\bar{x}_2+x_3$
7	$x_1x_2x_3$	1 1 1	$\bar{x}_1+\bar{x}_2+\bar{x}_3$

Należy mieć świadomość, że kanoniczna postać iloczynu może być uproszczona w analogiczny sposób jaki stosowaliśmy przy uproszczeniu kanonicznej postaci sumy tej funkcji. Stosując zasady algebry Boole'a łatwo sprawdzić, że funkcję tę, zapisaną w postaci iloczynu sum reprezentuje następujące wyrażenie:

$$f = (x_1 + x_3)(x_2 + x_3)$$

W tablicy 2.4 zestawiono dla przykładu wszystkie iloczyny pełne i sumy pełne dla trzech zmiennych x_1, x_2, x_3.

PRZYKŁAD 2.2

Dla funkcji $f_1 = \Sigma[0, 1, 2, 5, 8, 9, 10, (4, 12, 13)]$ podanej w tablicy 2.5 odpowiednie wyrażenie boolowskie wg kanonicznej postaci sumy jest:

Tablica 2.5

i	x_1	x_2	x_3	x_4	f_1	f_2
0	0	0	0	0	1	1
1	0	0	0	1	1	–
2	0	0	1	0	1	1
3	0	0	1	1	0	–
5	0	1	0	1	1	1
6	0	1	1	0	0	0
7	0	1	1	1	0	1
8	1	0	0	0	1	0
9	1	0	0	1	1	–
10	1	0	1	0	1	0
11	1	0	1	1	0	–
13	1	1	0	1	–	1
14	1	1	1	0	0	0
15	1	1	1	1	0	1

$$f_1 = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1\bar{x}_2x_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3x_4 + x_1\bar{x}_2\bar{x}_3\bar{x}_4 + x_1\bar{x}_2\bar{x}_3x_4 + x_1\bar{x}_2x_3\bar{x}_4$$

Kolejne iloczyny odpowiadają i = 0, 1, 2, 5, 8, 9, 10.

Natomiast dla funkcji $f_2 = \prod[6, 8, 10, 14, (1, 3, 4, 9, 11, 12)]$ kanoniczna postać iloczynu jest następująca:

$$f_2 = (x_1 + \bar{x}_2 + \bar{x}_3 + x_4)(\bar{x}_1 + x_2 + x_3 + x_4)(\bar{x}_1 + x_2 + \bar{x}_3 + x_4)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3 + x_4)$$

Kolejne sumy odpowiadają i = 6, 8, 10, 14.

1.3 Specyfikacje i realizacje funkcji boolowskich

Omówione w poprzednich podrozdziałach zasady reprezentacji funkcji boolowskich są w praktyce projektowania układów cyfrowych bardziej sformalizowane. Stosowanie formalnych zasad specyfikacji funkcji boolowskich jest ważne ze względu na coraz powszechniejsze stosowanie komputerowych systemów projektowania. Zasady te – w przypadku języków opisu sprzętu – są mocno rozbudowane i przy

założonej tematyce tego podręcznika mało istotne dla omawiania zaawansowanych algorytmów syntezy logicznej.

Z tych powodów ograniczymy się do tzw. standardu berkeley'owskiego stosowanego w komputerowym systemie Espresso [2.1].

W tablicy 2.6 jest podana tablica prawdy funkcji boolowskiej reprezentującej działanie dekodera wskaźnika siedmiosegmentowego (por. tabl. 2.2). Istotą takiej reprezentacji (plik typu pla) jest specyficzny zapis nagłówka określającego:

.type fr – typ danych

.i 4 – liczba wejść

.o 7 – liczba wyjść

.p 10 – liczba wektorów (kostek)

Tablica 2.6

```
type fr
.i 4
.o 7
.p 10
0000 1111110
0001 0110000
0010 1101101
0011 1111001
0100 0110011
0101 1011011
0110 1011111
0111 1110000
1000 1111111
1001 1111011
.e
```

Realizacje funkcji boolowskich

Dysponując różnymi bramkami logicznymi możemy realizować funkcje boolowskie w różnych strukturach. Do najbardziej typowych i najczęściej stosowanych należą:

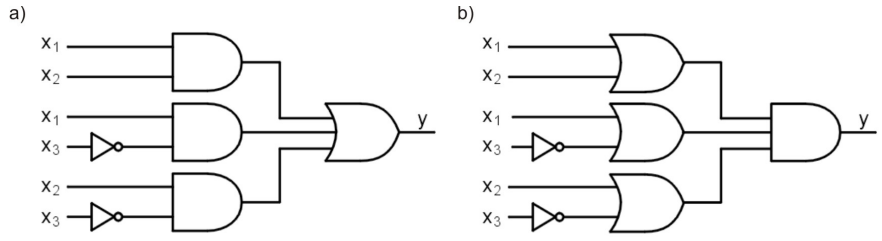
realizacja AND-OR,

realizacja OR-AND.

Na rys. 2.3a pokazano realizację AND-OR funkcji opisanej wyrażeniem:

$y = x_1x_2 + x_1\bar{x}_3 + x_2\bar{x}_3$.

.



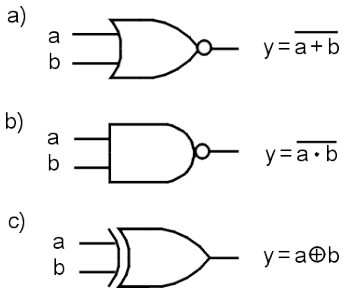
Rys. 2.3. Realizacja AND-OR (a) i OR-AND (b)

Realizacja AND-OR jest bezpośrednim odwzorowaniem wyrażenia boolowskiego typu SOP (*Sum of Product*). Z kolei realizację OR-AND uzyskujemy z wyrażenia typu iloczyn sum. Na rys 2.3b pokazano realizację OR-AND funkcji opisanej wyrażeniem $y = (x_1 + x_2)(\, x_1 + \bar{x}_3)(x_2 + \bar{x}_3)$

.

W dwuelementowej algebrze Boole'a wprowadza się też inne działania (operatory). Do najważniejszych z nich należą: zanegowana suma (NOR), zanegowany iloczyn (NAND), oraz suma wyłączająca (tzw. suma modulo 2 lub różnica symetryczna, oznaczana EXOR).

Określenia tych działań podano w tablicy 2.7, a odpowiednie symbole bramek na rys. 2.4.



Rys. 2.4. Symbole bramek NAND, NOR i EXOR

Dysponując bramkami logicznymi NAND, NOR możemy realizować funkcje boolowskie w innych strukturach. Do najbardziej typowych i najczęściej stosowanych należą:

realizacja NAND,

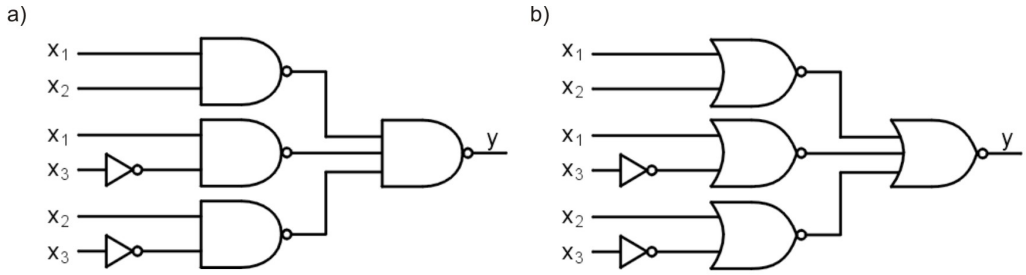
realizacja NOR.

Tablica 2.7

a b	$\overline{a+b}$	$\overline{a \cdot b}$	$a \oplus b$
0 0	1	1	0
0 1	0	1	1
1 0	0	1	1
1 1	0	0	0
	NOR	NAND	EXOR

Realizację NAND uzyskujemy z wyrażenia typu suma iloczynów, które przekształcamy zgodnie z prawem De Morgana do postaci:

$$y = \overline{\overline{x_1 x_2} \cdot \overline{x_1 \bar{x}_3} \cdot \overline{x_2 \bar{x}_3}}$$



Rys. 2.5. Realizacja NAND (a) i NOR (b)

Realizację NAND podano na rys. 2.5a.

Z kolei przekształcenie wyrażenia typu iloczyn sum wg prawa De Morgana :

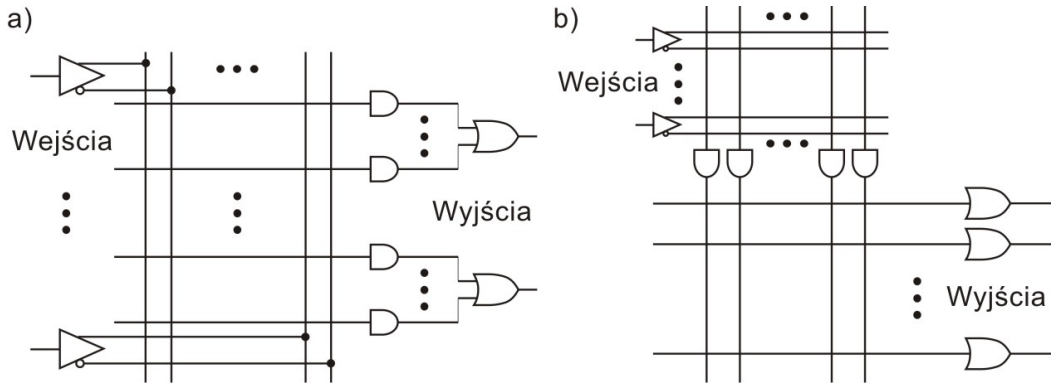
$$y = \overline{\overline{x_1 + x_2} + \overline{x_1 + \bar{x}_3} + \overline{x_2 + \bar{x}_3}}$$

bezpośrednio prowadzi do realizacji NOR, którą podano na rys. 2.5b.

Realizacje takie polegają na konfigurowaniu (programowaniu) matrycy AND-OR, która jest podstawowym elementem konstrukcyjnym układów PLD. Matryca AND-OR może być matrycą typu PAL (*Programmable Array Logic*), której cechą charakterystyczną jest programowalna matryca AND i stała matryca OR lub typu PLA (*Programmable Logic Array*), gdzie obie matryce, AND i OR są programowalne. Znaczy to, że PAL jest zespołem bramek iloczynowych dołączonych do oddzielnych bramek OR (rys. 2.6a), a

w PLA każda bramka AND może być dołączona do dowolnej bramki OR (rys. 2.6b). Programowanie polega na realizacji połączeń w punktach styku linii poziomych i pionowych.

Całkowicie odmienne możliwości realizacji funkcji boolowskich powstały wraz z wprowadzeniem struktur programowalnych typu FPGA (*Field Programmable Gate Array*). Wynika to z faktu, że w układach FPGA typową strukturą jest prostokątna macierz elementów logicznych zwanych komórkami, rozmieszczonych w środowisku komutacyjnym kanałów połączeniowych.



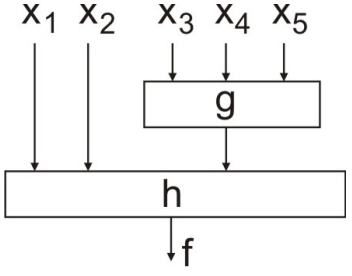
Rys. 2.6. Matryca typu PAL (a) i typu PLA (b)

Komórka struktur FPGA jest zbudowana z uniwersalnego układu kombinacyjnego, umożliwiającego realizację dowolnej funkcji logicznej (zadanej tablicą prawdy) o kilku wejściach i jednym lub dwóch wyjściach. Z tych powodów taką komórkę nazywa się komórką typu LUT (*Look Up Table*), a jednocześnie klasę układów zbudowanych z takich komórek nazywa się układami FPGA typu LUT. Dlatego realizacje w strukturach FPGA mają całkiem odmienne wymagania na struktury układów logicznych. Bezpośrednie odwzorowanie struktur bramkowych na komórki logiczne układów FPGA jest sprzeczne z naturą komórki o tyle, że z punktu widzenia syntezy jest ona przystosowana do realizacji dowolnej funkcji logicznej o argumentach wprowadzonych na jej wejścia. Z tych powodów dla struktur FPGA znacznie skuteczniejszą metodą syntezy okazuje się dekompozycja funkcji boolowskich, której istotą jest synteza funkcji boolowskich w strukturach wielopoziomowych złożonych z komponentów w postaci bloków logicznych typu LUT specyfikowanych pierwotnymi tablicami prawdy.

Tablica 2.8

	x_1	x_2	x_3	x_4	x_5	f
1	0	0	0	0	0	0
2	0	1	0	1	1	0
3	0	0	1	0	1	0
4	0	1	1	1	1	0
5	0	0	1	1	0	0
6	0	1	0	0	1	1
7	0	0	1	0	0	1
8	0	1	1	1	0	1
9	1	0	1	0	1	1
10	1	1	0	0	1	1
11	1	0	0	0	1	1

Na przykład funkcję podaną w tablicy 2.8 można zrealizować w strukturze FPGA, jeśli tylko bloki *g* i *h* realizują odwzorowania podane w tablicach 2.9a i 2.9b, co pokazano na rys. 2.7. Można sprawdzić, że dla każdego wektora złożenie funkcji *g* oraz *h* wytwarza na wyjściu *h* = *f* tę samą wartość.



Rys. 2.7. Realizacja funkcji *f* z tab. 2.8

Tablica 2.9

a)

x_3	x_4	x_5	g
0	0	0	0
0	1	1	1
1	0	1	0
1	1	1	1
1	1	0	0
0	0	1	0
1	0	0	1

b)

x_1	x_2	g	h
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	1	0	1

1.4 Metody minimalizacji funkcji boolowskich

Minimalizacja funkcji boolowskich jest zagadnieniem intensywnych prac badawczych od początku lat 50. XX wieku. Później, w latach 70. pojawienie się układów scalonych średniej i wielkiej skali integracji spowodowało wyraźne zmniejszenie zainteresowania tym problemem. Ale ponowny wzrost zainteresowania minimalizacją funkcji boolowskich powstał w latach 80. Bezpośrednią przyczyna tej sytuacji stała się możliwość realizacji układów logicznych w strukturach scalonych o złożoności milionów bramek logicznych.

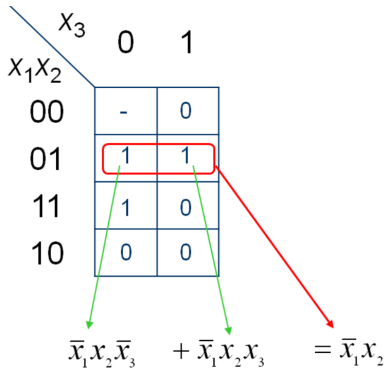
Najbardziej znaną i powszechnie stosowaną metodą graficzną jest metoda tablic Karnaugha. Klasyczną metodą analityczną jest metoda Quine'a – McCluskey'a. Obie metody opracowane zostały w połowie lat 50. XX wieku. Niestety zarówno metoda Karnugha, jak też Quine'a – McCluskey'a nie są przystosowane do wymagań dzisiejszych technologii.

Wadą pierwszej jest graficzny sposób obliczeń ograniczający z natury liczbę zmiennych minimalizowanych funkcji, natomiast w przypadku drugiej barierą ograniczająca jest złożoność obliczeniowa stosowanych algorytmów.

Dlatego powstały nowe, całkowicie odmienne metody syntezy dwupoziomowej. Przede wszystkim znalazły one zastosowanie w klasycznym już algorytmie ESPRESSO opracowanym na Uniwersytecie Kalifornijskim w Berkeley w latach 80. Znajdują one minimalne lub suboptymalne rozwiązania nawet dla bardzo skomplikowanych zadań. Syntezie mogą być poddawane zespoły nie w pełni określonych funkcji boolowskich, o wielowartościowych wejściach i o setkach argumentów, a czas obliczeń jest stosunkowo krótki.

1.4.1 Metoda Karnaugha

Mimo swoich niedoskonałości metoda Karnaugha jest najczęściej wykładaną metoda minimalizacji funkcji boolowskich. Paradoksalnie to co jest jej wadą – prostota – jest jednocześnie jej zaletą, gdyż można szybko ją „opanować" i stosować w obliczeniach ręcznych. Należy jednak pamiętać, że w praktyce projektowania inżynierskiego metoda tablic Karnaugha jest absolutnie nieprzydatna, gdyż zakres jej zastosowań kończy się na funkcjach 6 – 7 argumentów, a typowe zadania praktyczne operują funkcjami o kilkudziesięciu argumentach.



Rys. 2.8. Zasada sklejania

W metodzie Karnaugha każda kratka odpowiedniej tablicy (zwanej tablicą Karnaugha) odpowiada wektorowi zmiennych binarnych. Można więc powiedzieć, że ciąg wartości tych zmiennych tworzy adres kratki. Kratki są ponumerowane w taki sposób, że numer *i* jest liczbą dziesiętną odpowiadającą kombinacji zmiennych (wektorowi zero-jedynkowemu) traktowanej jako liczba dwójkowa. W poszczególnych kratkach wpisane są wartości funkcji, tj. 0 lub 1 przyjmowanej przez funkcję dla tej kombinacji lub symbol „–", jeżeli funkcja nie jest określona (rys. 2.8). W tablicy Karnaugha różniącym się tylko o negację pełnym iloczynom przyporządkowane są leżące obok siebie pola tablicy (sąsiednie kratki), które się łączy (skleja) odpowiednią pętelką. Korzysta się z faktu, że dla dowolnego A, $A\bar{x} + Ax = A$.

Tablica 2.10

0	0
0	1
1	1
1	0

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

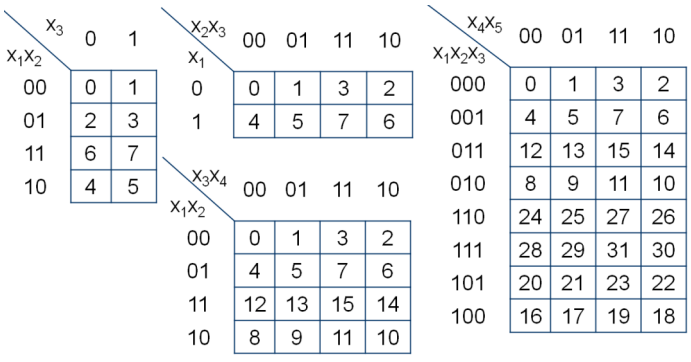
Dla uzyskania efektu sąsiedztwa współrzędne pól opisuje się tzw. kodem Gray'a. Kod Gray'a jest takim ciągiem wektorów binarnych, w którym każde dwa sąsiednie wektory różnią się wyłącznie na jednej pozycji. Przykłady kodu Gray'a dla dwóch i trzech zmiennych podane są w tabl. 2.10.

Minimalizacja metodą Karnaugha polega na wykonaniu dwóch czynności:

a) wpisanie funkcji do tablicy,

b) zakreślanie pętelek i kojarzenie z nimi iloczynów zmiennych prostych i zanegowanych.

Wpisywanie funkcji do tablicy Karnaugha ułatwia numeracja kratek. Na rys. 2.9 podajemy przykłady tablic Karnaugha z ponumerowanymi kratkami według naturalnego kodu binarnego (NKB), odpowiednikami. Wzorce tych tablic ułatwiają znajdowanie kratek odpowiadających wektorom w tablicy prawdy.



Rys. 2.9. Przykłady tablic z ponumerowanymi kratkami

PRZYKŁAD 2.3

Minimalizacja funkcji f z poprzedniego rozdziału, której tablicę prawdy powtarzamy w tabl. 2.11 wymaga zatem wpisania funkcji do tablicy Karnaugha.

Tablica 2.11

	x_1	x_2	x_3	f
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Wystarczy w tym celu wyszukiwać kratki tablicy, których współrzędne odpowiadają wektorom tablicy prawdy i do ich wnętrza wpisywać wartość funkcji.

Tablica 2.12

x_1x_2 \ x_3	0	1
00	0	1
01	0	1
11	1	1
10	0	1

Tablica Karnaugha funkcji f podana jest w tabl. 2.12. Następnie zakreślamy pętelki. Pętelki muszą obejmować kratki wypełnione „1", których współrzędne reprezentują pełne iloczyny zmiennych podlegające uproszczeniu zasadami algebry Boole'a. Łatwo stwierdzić, że „1" małej pętelki są opisane wyrażeniem:

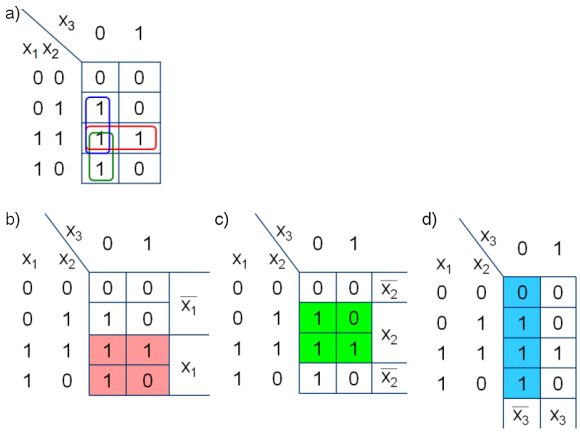
$$x_1 x_2 \bar{x}_3 + x_1 x_2 x_3 \qquad (2\text{-}1)$$

natomiast dużej:

$$\bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 x_3 + x_1 x_2 x_3 + x_1 \bar{x}_2 x_3 \qquad (2\text{-}2)$$

Stosując zasady algebry Boole'a można wyrażenie 2-1 uprościć do iloczynu x_1x_2, natomiast wyrażenie 2-2 do jednej zmiennej x_3, zatem $f = x_1x_2 + x_3$.

Niestety zakreślanie pętelek i kojarzenie z nimi odpowiednich iloczynów jest trudniejsze. Omawiamy ten proces na bardziej skomplikowanym przykładzie (rys. 2.10), w którym dla poszczególnych kopii tablicy Karnaugha trzech zmiennych zaznaczamy pola odpowiadające pojedynczym zmiennym (prostym i zanegowanym). Pokrywanie się tych pól określa odpowiedni iloczyn zmiennych prostych



Rys. 2.10. Ilustracja minimalizacji funkcji boolowskiej

Minimalizacja funkcji za pomocą tablic Karnaugha sprowadza się do wykonania czynności opisanych w następującym algorytmie.

Algorytm 2.1

1) Wpisujemy funkcję do tablicy Karnaugha.

2) Zakreślamy pętelki.

a) Pętelki zakreślamy wokół grup sąsiadujących kratek zawierających „1” albo „1” i „–” (kreski).

b) Liczba kratek objętych pętelka musi wynosić: 1, 2, 4, …,2^k.

c) Staramy się objąć pętelką jak największą liczbę kratek.

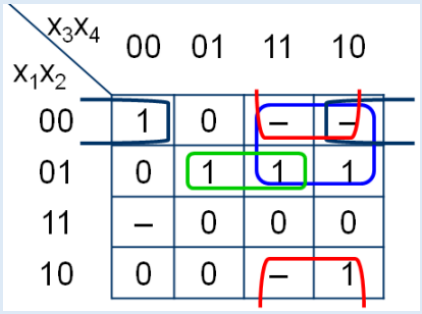
3) Pętelki zakreślamy tak długo, aż każda „1” będzie objęta co najmniej jedną pętelką, pamiętając o tym aby pokryć wszystkie „1” możliwie minimalną liczbą pętelek.

4) Z każdą pętelką kojarzymy iloczyn zmiennych prostych lub zanegowanych. Suma tych iloczynów, to minimalne wyrażenie boolowskie danej funkcji.

PRZYKŁAD 2.4

Należy zminimalizować funkcję f zapisaną symbolicznie w postaci SOP liczbami dziesiętnymi: $f = \Sigma[0, 5, 6, 7, 10, (2, 3, 11, 12)]$

Tablica 2.14



Korzystając z odpowiedniego wzoru tablicy z rys. 2.9 znajdujemy kratki o odpowiednich numerach i wpisujemy do nich „1” i ”–„ (kreski). W pozostałe wpisujemy „0”. Po wpisaniu zakreślamy pętelki (tabl. 2.13), z którymi kojarzymy odpowiednie iloczyny, uzyskując w rezultacie minimalne wyrażenie boolowskie w postaci sumy iloczynów:

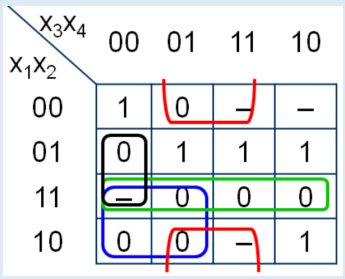
W podobny sposób postępujemy przy obliczaniu minimalnego wyrażenia w postaci iloczynu sum. Jedyną różnicą jaką w tym przypadku należy wprowadzić do algorytmu 2.1 jest zmiana w punkcie w punkcie 2a, a mianowicie: pętelki zakreślamy wokół grup sąsiadujących kratek zawierających „0” albo „0” i „–” (kreski); inny jest również sposób interpretacji pętelek: pętelkom odpowiadają sumy zmiennych

PRZYKŁAD 2.5

Minimalizacja funkcji z przykładu 2.4 do postaci iloczynu sum przedstawiona jest na tabl. 2.14.

Rozwiązanie

Tablica 2.14



Poszczególnym pętelkom na tej tablicy odpowiadają sumy zmiennych, których iloczyn tworzy następujące wyrażenie:

$$f = (\bar{x}_1 + x_3)(\bar{x}_1 + \bar{x}_2)(x_2 + \bar{x}_4)(\bar{x}_2 + x_3 + x_4)$$

PRZYKŁAD 2.6

Uprościć następujące wyrażenie:

$$Y = (\bar{A} + \bar{B} + C + D)(A + \bar{B} + \bar{C} + D)(A + \bar{B} + C + D)(\bar{A} + B)(A + \bar{D})$$

Rozwiązanie

Nanosimy wyrażenie *Y* na tablicę Karnaugha (tab. 2.15) i upraszczamy tworząc pętelki obejmujące zera. Następnie wypisujemy rozwiązanie, dla kanonicznego iloczynu sum argumenty z 0 są proste, argumenty z 1 są zanegowane.

Tablica 2.15

AB \ CD	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	1	1	1
10	0	0	0	0

Po minimalizacji uzyskaliśmy następująca funkcję:

$$Y = (A + \bar{B})(\bar{A} + B)(A + \bar{D})(\bar{B} + C + D)$$

Przykład 2.4 posłuży teraz do wprowadzenia jednego z najważniejszych pojęć minimalizacji jakim jest implikant.

DEFINICJA

Implikant danej funkcji f jest to iloczyn literałów (zmiennych prostych i zanegowanych) taki, że odpowiadający mu zbiór wektorów binarnych nie zawiera wektora „zerowego" funkcji.

Prosty implikant jest to implikant, który zmniejszony o dowolny literał przestaje być implikantem.

Na przykład $\bar{x}_1 x_3 x_4$ jest implikantem funkcji f, bo zbiór wektorów reprezentowanych przez ten implikant: 0111 oraz 0011 nie zawiera żadnego wektora (mintermu), dla którego wartość funkcji jest 0. Są to bowiem mintermy, które w naturalnym kodzie binarnym odpowiadają liczbom 7 i 12, i jak widać z zapisu funkcji $f = \Sigma[0, 5, 6, 7, 10, (2, 3, 11, 12)]$, nie należną one do zbioru wektorów „zerowych" tej funkcji.

Tablica 2.16

x_1x_2 \ x_3x_4	00	01	11	10
00	1	0	–	–
01	0	1	1	1
11	–	0	0	0
10	0	0	–	1

Pojęcie implikantu można zinterpretować na tablicy Karnaugha. W tablicy 2.16 zakreślono dwie pętelki. Mniejsza odpowiada implikantowi $\bar{x}_1 x_3 x_4$. Nie może to być implikant prosty, gdyż pętelkę tę można powiększyć. Większej pętelce odpowiada implikant prosty $\bar{x}_1 x_3$, w którym nie można już usunąć żadnego literału.

PRZYKŁAD 2.7

Zminimalizować metodą Karnaugha następującą funkcję boolowską:

$f = \Sigma[4,5,10,11,15,18,20,24,26,30,31, (9,12,14,16,19,21,25)]$

Po zakreśleniu „pętelek" (tab. 2.17) uzyskujemy następujące wyrażenie boolowskie:

$$f = \bar{x}_2 x_3 \bar{x}_4 + \bar{x}_1 x_2 x_4 + x_2 x_3 x_4 + x_1 \bar{x}_3 \bar{x}_5$$

Tablica 2.17

$x_1 x_2 x_3$ \ $x_4 x_5$	00	01	11	10
000	0	0	0	0
001	1	1	0	0
011	–	0	1	–
010	0	–	1	1
110	1	–	0	1
111	0	0	1	1
101	1	–	0	0
100	–	0	–	1

W dotychczasowych przykładach minimalizowane były pojedyncze funkcje boolowskie, dla których – w celu uzyskania rozwiązania z minimalnym kosztem – tworzone były wyłącznie implikanty proste. W ogólnym przypadku minimalizacji zespołów funkcji boolowskich tworzenie minimalnego rozwiązania wyłącznie na podstawie implikantów prostych nie zawsze prowadzi do najlepszego rezultatu końcowego. Problem sygnalizuje przykład 2.8.

PRZYKŁAD 2.8

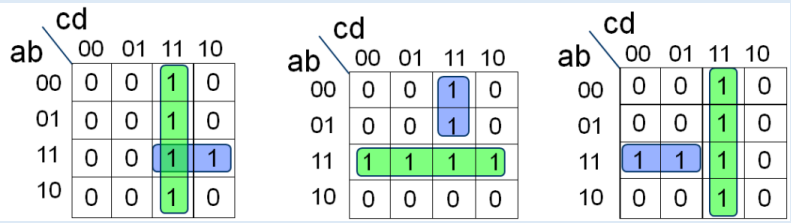
Należy zrealizować zespół trzech funkcji czterech argumentów:

$f_1 = \Sigma(3,7,11,14,15)$

$f_2 = \Sigma(3,7,12,13,14,15)$

$f_3 = \Sigma(3,7,11,12,13,15)$

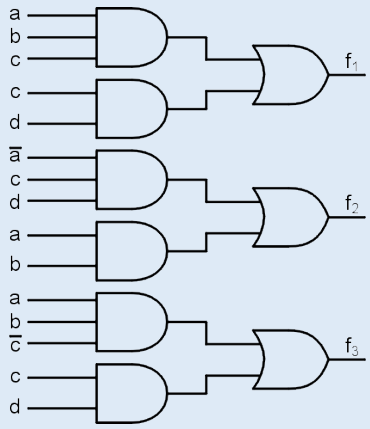
Tablica 2.18



Jeśli każdą funkcję zminimalizujemy oddzielnie (tablice Karnaugha poszczególnych funkcji pokazane są w tabl. 2.18), to uzyskamy następujące wyrażenia:

$$f_1 = abc + cd, \qquad f_2 = ab + \bar{a}cd, \qquad f_3 = ab\bar{c} + cd,$$

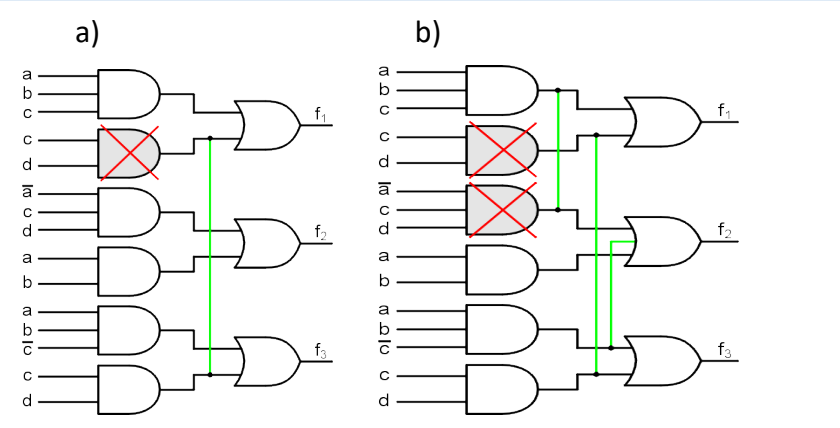
których realizację pokazano na rys. 2.11. Do realizacji tych trzech funkcji potrzebujemy 9 bramek



Rys. 2.11. Realizacja funkcji z tablicy 2.18

Zauważmy jednak, że bramka AND dla f_1 może być usunięta przez wykorzystanie bramki AND z f_3 (rys. 2.12a). Natomiast bramkę AND z f_2 można usunąć przez wykorzystanie faktu $ab = abc + ab\bar{c}$, co prowadzi do struktury zbudowanej zaledwie z 7 bramek (rys. 2.12b).

PRZYKŁAD 2.8. c.d.



Rys. 2.12. Realizacja funkcji z tablicy 2.18: a) z uproszczoną funkcją f_1;

b) z uproszczonymi funkcjami f1 i f2

Przykład sugeruje, że w realizacji zespołu funkcji stosowanie minimalnej sumy implikantów prostych nie zawsze prowadzi do rozwiązania z minimalnym kosztem. Kolejny przykład ilustruje w jaki sposób należy dobierać implikanty zespołu funkcji, aby uzyskać rozwiązanie z minimalnym kosztem.

PRZYKŁAD 2.9

Należy zminimalizować zespół funkcji:

$y_1 = \Sigma(2,3,5,7,8,9,10,11,13,15)$

$y_2 = \Sigma(2,3,5,6,7,10,11,14,15)$

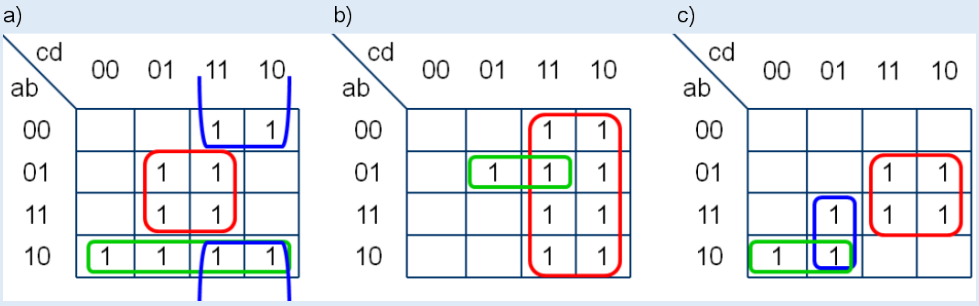
$y_3 = \Sigma(6,7,8,9,13,14,15)$

W tablicach 2.19a, b, c podane są wykresy Karnaugha poszczególnych funkcji, odpowiednio: y_1, y_2, y_3. Po zakreśleniu „pętelek" obejmujących implikanty proste uzyskujemy następujące wyrażenia boolowskie:

$$y_1 = a\bar{b} + bd + \bar{b}c \qquad y_2 = c + \bar{a}bd \qquad y_3 = bc + a\bar{c}d + a\bar{b}\bar{c}$$

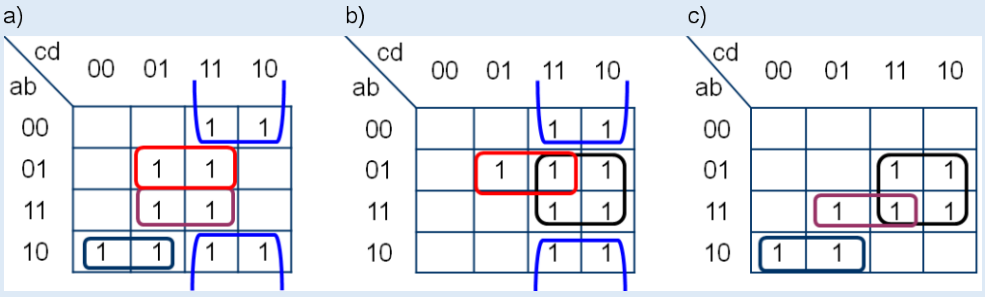
Realizacja tych wyrażeń wymaga zastosowania 7 bramek AND i 3 bramek OR.

Tablica 2.19



Znacznie lepszą realizację uzyskamy dokonując minimalizacji w sposób przedstawiony w tablicach 2.20a, b, c.

Tablica 2.20



Pętelki w tych tablicach zakreślamy tak, aby uzyskiwać implikanty wspólne dla co najmniej dwóch spośród trzech funkcji. Spełnienie tego warunku wymusza zakreślanie pętelek wokół grup kratek reprezentujących implikanty, które nie są implikantami prostymi. W rezultacie uzyskujemy następujące wyrażenia:

$$y_1 = \bar{b}c + \bar{a}bd + abd + a\bar{b}\bar{c} \qquad y_2 = \bar{b}c + \bar{a}bd + bc \qquad y_3 = abd + a\bar{b}\bar{c} + bc$$

Realizacja powyższych wyrażeń łącznie, mimo pozornie większego skomplikowania dla poszczególnych funkcji, jest oszczędniejsza niż poprzednio; całość wymaga bowiem zastosowania 5 bramek AND i trzech bramek OR.

1.4.2 Metoda systematyczna

W metodzie systematycznej zbiór wektorów (w ogólności kostek), dla których funkcja $f = 1$ oznacza się ***F***. Natomiast zbiór tych wektorów (kostek), dla których funkcja $f = 0$ oznacza się ***R***.

W tabl. 2.21 przedstawiono tablicę prawdy funkcji boolowskiej 7-argumentowej. Funkcję tę w dalszej części wykładu oznaczać będziemy EXTL. Wektory zbioru ***F*** oznaczono symbolami $k_1, \ldots, k_5$.

Tablica 2.21

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	f
	1	0	0	0	1	0	1	0
	1	0	1	1	1	1	0	0
	1	1	0	1	1	1	0	0
	1	1	1	0	1	1	1	0
k_1	0	1	0	0	1	0	1	1
k_2	1	0	0	0	1	1	0	1
k_3	1	0	1	0	0	0	0	1
k_4	1	0	1	0	1	1	0	1
k_5	1	1	1	0	1	0	1	1

Metoda systematyczna jest procesem działającym na kostkach zbiorów ***F*** i ***R***, a jej celem jest uzyskanie dla danej $k \in \boldsymbol{F}$ kostki k' tak dużej, jak to tylko możliwe (tzn. z możliwie dużą liczbą pozycji o wartości $*$) i nie pokrywającej żadnego wektora zbioru ***R***. W swoich obliczeniach metoda ta wykorzystuje tzw. macierz blokującą ***B***.

Macierzą blokującą (kostkę k) nazywać będziemy macierz:

$$\boldsymbol{B}(k, \boldsymbol{R}) = [b_{ij}], \qquad 1 \le i \le |\boldsymbol{R}|, \qquad 1 \le j \le n,$$

w której każdy element $b_{ij} \in \{0,1\}$ jest definiowany następująco:

$b_{ij} = 1$, jeśli $k_j = 1$ oraz $r_{ij} = 0$ lub $k_j = 0$ oraz $r_{ij} = 1$;

$b_{ij} = 0$, w pozostałych przypadkach.

W przypadku funkcji opisanej wektorami binarnymi macierz blokująca dla danej kostki $k \in \boldsymbol{F}$ powstaje z macierzy ***R*** przez zanegowanie tych kolumn ***R***, których pozycje są wyznaczone przez pozycje jedynek w kostce $k \in \boldsymbol{F}$.

PRZYKŁAD 2.10

Wyznaczymy macierz blokującą dla $f = (\boldsymbol{F}, \boldsymbol{R})$ zadanej w tablicy 2.21 i kostki k_2. Jak wynika z tej tablicy, zbiory $\boldsymbol{F}$ i $\boldsymbol{R}$ są opisane macierzami:

$$\boldsymbol{F} = \begin{bmatrix} 0\,1\,0\,0\,1\,0\,1 \\ 1\,0\,0\,0\,1\,1\,0 \\ 1\,0\,1\,0\,0\,0\,0 \\ 1\,0\;\;1\,0\,1\;\;0\,1 \\ 1\,1\,1\,0\,1\,0\,1 \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} 1\,0\,0\,0\,1\,0\,1 \\ 1\,0\,1\,1\,1\,1\,0 \\ 1\,1\,0\,1\,1\,1\,0 \\ 1\,1\,1\,0\,1\,1\,1 \end{bmatrix}$$

Skoro $k_2 = (1000110)$, to dla uzyskania $\boldsymbol{B}$ wystarczy w macierzy $\boldsymbol{R}$ „zanegować" kolumny pierwszą, piątą i szóstą. Zatem $\boldsymbol{B}(k_2, \boldsymbol{R})$:

$$\boldsymbol{B} = \begin{matrix} 1\,2\,3\,4\,5\,6\,7 \\ \end{matrix}\begin{bmatrix} 0\,0\,0\,0\,0\,1\,1 \\ 0\,0\,1\,1\,0\,0\,0 \\ 0\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,0\,0\,1 \end{bmatrix}$$

Pokryciem kolumnowym macierzy $\boldsymbol{B} = [b_{ij}]$, $i \in \{1, \ldots, w\}$, $j \in \{1, \ldots, n\}$ jest zbiór $L \subseteq \{1, \ldots, n\}$ taki, że dla każdego $i \in \{1, \ldots, w\}$ istnieje $j \in L$, dla którego $b_{ij} = 1$.

Pokrycie kolumnowe jest pokryciem, w którym elementami pokrywanymi są wiersze $\boldsymbol{B}$, a pokrywającymi – kolumny tej macierzy. Jednak brak formalnych elementów pokrywanych i pokrywających skłania do wprowadzenia nazwy „pokrycie kolumnowe".

PRZYKŁAD 2.11

Obliczenia pokrycia kolumnowego omówimy na przykładzie macierzy blokującej wyznaczonej dla kostki k_2 funkcji z poprzedniego przykładu.

$$\boldsymbol{B} = \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Oznaczając kolumny macierzy $\boldsymbol{B}$ kolejno L_1 do L_7 zapisujemy poszczególne wiersze tej macierzy oznaczeniami tych kolumn, które w danym wierszu mają „1". Na przykład wiersz pierwszy zapiszemy L_6, L_7, drugi L_3, L_4, itd. Traktując kolumny L_i jako zmienne boolowskie tworzymy iloczyn sum kolumn poszczególnych wierszy:

$$(L_6 + L_7)\,(L_3 + L_4)\,(L_2 + L_4)\,(L_2 + L_3 + L_7)$$

Następnie iloczyn sum przekształcamy do wyrażenia boolowskiego typu suma iloczynów:

$$(L_4 + L_2)(L_4 + L_3)(L_7 + L_6)(L_7 + L_2 + L_3) = (L_4 + L_2L_3)(L_7 + L_6(L_2 + L_3)) = (L_4 + L_2L_3)(L_7 + L_2L_6 + L_3L_6) =$$
$$= L_4L_7 + L_2L_4L_6 + L_3L_4L_6 + L_2L_3L_7 + L_2L_3L_6$$

Składniki tego wyrażenia reprezentują zbiory minimalnego pokrycia kolumnowego:
$\{L_4, L_7\}, \{L_2, L_4, L_6\}, \{L_3, L_4, L_6\}, \{L_2, L_3, L_7\}, \{L_2, L_3, L_6\}$

Macierz blokująca $\boldsymbol{B}(k, \boldsymbol{R})$ pozwala wyznaczyć uogólnioną kostkę k oznaczaną $k^+(L, k)$ w sposób następujący:

$$k^+(L,k) = \begin{cases} k_j, \text{gdy } j \in L \\ *, \quad \text{w pozostałych przypadkach.} \end{cases} \qquad (2\text{-}3)$$

Uogólnioną kostkę $k^+(L, k)$ nazywa się również ekspansją kostki k, co oznacza, że wszystkie składowe kostki k należące do L nie ulegają zmianie, natomiast składowe nie należące do L przyjmują wartość $*$.

Można wykazać, że $k^+(L, k)$ jest implikantem funkcji $f = (\boldsymbol{F}, \boldsymbol{R})$. W szczególności $k^+(L, k)$ jest implikantem prostym, gdy L jest minimalnym pokryciem kolumnowym macierzy $\boldsymbol{B}(k, \boldsymbol{R})$. Wykorzystując pojęcie macierzy blokującej można obliczyć implikanty proste dla poszczególnych kostek zbioru $\boldsymbol{F}$.

PRZYKŁAD 2.12

Kontynuujemy obliczenia dla macierzy blokującej ***B*** = ***B***(k_2, ***R***) z przykładu 2.10.

Zbiór *L* = {4,7} jest pokryciem kolumnowym ***B***, a więc $k^+(L,k) = (***0**0)$, czyli implikantem ***F*** jest $\bar{x}_4\bar{x}_7$. Natomiast dla *L* = {2,4,6} (inne pokrycie kolumnowe) $k^+(L,k) = (*0*0*1*) = \bar{x}_2\bar{x}_4x_6$. Postępując podobnie dla pozostałych pokryć kolumnowych można uzyskać następujący zbiór implikantów prostych funkcji f generowanych przez kostkę k_2:

$\bar{x}_4\bar{x}_7$, $\bar{x}_2\bar{x}_4x_6$,.$\bar{x}_3\bar{x}_4x_6$,. $\bar{x}_2\bar{x}_3\bar{x}_7$,. $\bar{x}_2\bar{x}_3x_6$

Każdy z obliczonych w ten sposób implikantów może zastąpić (często mówimy pokrywa) wiele innych kostek pierwotnych funkcji *f*, gdzie relację pokrycia definiujemy następująco:

$K \subseteq K'$ (K' pokrywa K) wtedy i tylko wtedy, gdy $k_i = k_i' \; lub \; k_i = *$ dla każdego *i*, $1 \le i \le n$,
gdzie: $K = (k_1, \ldots, k_n)$ i $K' = (k_1', \ldots, k_n')$,

Na przykład obliczony wyżej implikant pokrywa kostki k_2, k_3, k_4, co zapisujemy formalnie:
$\bar{x}_4\bar{x}_7 \supseteq k_2, k_3, k_4$.

Relacja pokrycia dla kostek i sposób postępowania omówiony w przykładzie 2.12 nasuwa pomysł systematycznej metody obliczania minimalnych realizacji funkcji boolowskich.

Kolejne etapy takiej metody są następujące.

1. Dla każdej k_i należącej do zbioru ***F*** obliczyć macierz blokującą.
2. Wyznaczyć wszystkie minimalne pokrycia kolumnowe tej macierzy.
3. Wyznaczyć wszystkie implikanty proste.
4. Obliczyć sumę zbiorów implikantów prostych generowanych przez poszczególne kostki.
5. Z tak utworzonego zbioru wszystkich implikantów prostych usunąć implikanty powtarzające się.
6. Spośród obliczonych implikantów prostych wyselekcjonować minimalny podzbiór implikantów pokrywający wszystkie kostki zbioru ***F***.

Niezależnie jednak od metody ograniczania liczności zbioru implikantów prostych proces ich selekcji wykonuje się za pośrednictwem tzw. tablicy implikantów prostych.

Tablica implikantów prostych jest to tablica elementów binarnych 0 lub 1, o liczbie wierszy równej liczbie kostek zbioru ***F*** oraz liczbie kolumn równej liczbie wyznaczonych implikantów prostych funkcji *f*.

Tablicę implikantów prostych konstruujemy w następujący sposób. Jeśli implikant I_j pokrywa kostkę k_i, to na przecięciu wiersza k_i z kolumną I_j piszemy 1, w przeciwnym przypadku piszemy 0.

PRZYKŁAD 2.13

Utworzymy tablicę implikantów prostych dla funkcji f podanej w tabl. 2.21. W tym celu obliczamy minimalne (i o najmniejszej liczności) pokrycia kolumnowe dla każdej kostki zbioru ***F*** tej funkcji. Oznaczając przez J_i implikanty generowane przez kostkę k_i uzyskujemy kolejno:

$$J_1 = \bar{x}_1 \qquad J_3 = \bar{x}_5 \qquad J_5 = x_2\bar{x}_6 \text{ oraz } x_3\bar{x}_6$$
$$J_2 = \bar{x}_4\bar{x}_7 \qquad J_4 = \bar{x}_4\bar{x}_7$$

Usuwając implikanty powtarzające się ostateczną listę implikantów prostych zapisujemy jak następuje:

$$I_1 = \bar{x}_1 \qquad I_3 = \bar{x}_5 \qquad I_5 = x_3\bar{x}_6$$
$$I_2 = \bar{x}_4\bar{x}_7 \qquad I_4 = x_2\bar{x}_6$$

Na tej podstawie dla każdego obliczonego wyżej implikanta (ale interpretowanego jako kostka) wyznaczamy wszystkie kostki zbioru ***F*** pokrywane przez ten implikant. Przykładowo:

$$I_1 = \bar{x}_1 \supseteq k_1$$
$$I_2 = \bar{x}_4\bar{x}_7 \supseteq k_2, k_3, k_4$$
$$I_4 = x_2\bar{x}_6 \supseteq k_1, k_5$$

Tablicę implikantów prostych dla funkcji *f* pokazano w tabl. 2.22 .

Tablica 2.22

	I_1	I_2	I_3	I_4	I_5
k_1	1	0	0	1	0
k_2	0	1	0	0	0
k_3	0	1	1	0	1
k_4	0	1	0	0	0
k_5	0	0	0	1	1

Tablica implikantów prostych umożliwia wybór (selekcję) takiego minimalnego zbioru implikantów, który pokrywa wszystkie kostki funkcji pierwotnej.

Minimalny zbiór implikantów prostych reprezentujących funkcję f można wyznaczyć obliczając minimalne pokrycie kolumnowe tablicy implikantów prostych. W tym celu tworzymy wyrażenie CNF, $I_2\ (I_1 + I_4)\ (I_2 + I_3 + I_5)\ (I_4 + I_5)$, które po oczywistych uproszczeniach zamieniamy na DNF: $I_2I_4 + I_1I_2I_5$

Zatem minimalne pokrycie tworzą implikanty I_2, I_4. Czyli,

$$f = \bar{x}_4\bar{x}_7 + x_2\bar{x}_6$$

Obliczeniem decydującym o eksplozji kombinatorycznej zadania minimalizacji funkcji boolowskiej jest obliczenie wszystkich pokryć kolumnowych tablicy implikantów prostych. O złożoności tego problemu decyduje szybko rosnąca (ze wzrostem liczby argumentów) liczność rodziny tych implikantów.

Otóż w przypadku funkcji z tablicy 2.21 liczba wszystkich implikantów jest 5: tym samym odpowiednia tablica implikantów (tabl. 2.22) ma 5 kolumn. W rezultacie obliczenie minimalnych pokryć kolumnowych tej tablicy można wykonać „ręcznie” – jest widoczne „gołym okiem”. Ale nie zawsze tak musi być. Zjawisko występującej w tym problemie złożoności obliczeniowej lepiej wyjaśni następny przykład, którego wyniki obliczono programem InstantRS.

PRZYKŁAD 2.14

Dla funkcji podanej w standardzie typu pla (tabl. 2.23) zbiór implikantów jest następujący:

I1 = !x6!x7
I2 = !x7x8
I3 = x2x3
I4 = x3x4
I5 = x3!x5
I6 = x3x6
I7 = x3!x7
I8 = x3!x8
I9 = x1x4
I10 = x2x4
I11 = x4!x5
I12 = x4x6
I13 = x4!x7
I14 = x4!x8
I15 = x2!x7
I16 = x6x8
I17 = !x1x2
I18 = x2x5
I19 = !x6!x8

Tablica 2.23

```
. type fr
.i 8
.o1
.p12
11000110 0
00000100 0
10101011 0
10000100 0
10001110 0
11000011 0
00011011 0
00000001 1
11110100 1
11010111 1
01001111 1
11000010 1
.e
```

zatem do obliczenia wszystkich minimalnych pokryć kolumnowych (tabl. 2.24) trzeba formalnie obliczyć wyrażenie typu DNF dla następującego wyrażenia w postaci CNF:

I19(I1 + I2)(I3 + I4 +.....+ I15)(I9 + I10 + I11 + I12 + I16)(I16 + I17 + I18).

Tablica 2.24

#	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	I16	I17	I18	I19
c1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c2	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
c3	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0
c4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
c5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Nie jest to zadanie proste i zawiera minimalne pokrycia kolumnowe reprezentujące 42 rozwiązania, czyli 42 minimalne reprezentacje funkcji z Tabl. 2.23, przykładowo:

01] !x6!x7 + x2x3 + x6x8 + !x6!x8

02] !x6!x7 + x3x4 + x6x8 + !x6!x8

03] !x6!x7 + x3!x5 + x6x8 + !x6!x8

04] !x6!x7 + x3x6 + x6x8 + !x6!x8

05] !x6!x7 + x3!x7 + x6x8 + !x6!x8

06] !x6!x7 + x3!x8 + x6x8 + !x6!x8

…….

41] !x7x8 + x4!x5 + x2x5 + !x6!x8

42] !x7x8 + x4x6 + x2x5 + !x6!x8

Można się zatem spodziewać eksplozji kombinatorycznej, której rezultat dla funkcji Kaz podanej w Tabl. 2.25 prowadzi do wniosku, że w tym przypadku liczba wszystkich minimalnych implikantów jest 416, co uniemożliwia systematyczne obliczenie minimalnych wyrażeń boolowskich tej funkcji.

Ograniczenia w stosowaniu systematycznego algorytmu obliczania pokrycia kolumnowego spowodowały potrzebę użycia szybkich, heurystycznych algorytmów redukcji tablic boolowskich. Opracowane zostały dwa algorytmy iteracyjne: MaxCol i MinRow.

Algorytm MaxCol wykorzystuje strategię, w której dla uzyskania minimalnego pokrycia kolumnowego najbardziej istotne są kolumny macierzy, które zawierają największą liczbę jedynek.

Postępowanie w algorytmie MaxCol

1. Policzenie wystąpień wartości 1 w każdej kolumnie
2. Zapisanie kolumny z największą liczbą 1 – jeżeli jest kilka kolumn o tej samej liczbie 1, arbitralnie wybrana zostaje ta z mniejszym indeksem
3. Usunięcie wszystkich wierszy, dla których w wybranej kolumnie znajduje się wartość 1
4. Jeżeli macierz nie jest pusta (zawiera elementy), następuje powrót do kroku 1.
5. Zwrot zapisanych kolumn

Algorytm MinRow opiera swoje iteracyjne działanie analizowaniu wierszy o najmniejszej ilości jedynek.

Tablica 2.25

.type fr
.i 21
.o 1
.p 31

10011001011001111110 1
11101111011110111100 1
00101010100001111000 0 1
00100110110011011000 1 1
10011001001101100110 1 1
10010110010011011001 1 1
00110010011101001101 1 1
00110110001101101100 1 1
11011001001100100110 1 1
10011011001101001001 1 1
11001101101101000110 0 1
01000101000000110011 1 0
10011010101111111010 0 0
11100111101111001100 0 0
10110101110001011110 0 0
11010110000101010000 0 0
11011011011110001011 0 0
11000010001111001000 1 0
00100100010111110110 1 0
10010001111110011011 0 0
10001100011001101111 0 0
11010100011010110000 1 0
11011000110110110011 1 0
01000011100100000000 1 0
00100110010111111000 0 0
10010011111100111001 0 0
00001000111000110110 1 0
10100001010000111000 0 0
10100011010101001111 1 0

Kroki algorytmu MinRow

1. Policzenie wystąpień wartości 1 w każdym wierszu
2. Wybranie wierszy z najmniejszą liczbą wystąpień 1
3. Policzenie wystąpień wartości 1 w kolumnach, dla których dowolny wybrany w kroku 2. wiersz ma wartość 1
4. Zapisanie kolumny z największą liczbą 1 wśród tych z kroku 3. – jeżeli jest kilka kolumn o tej samej liczbie 1, arbitralnie wybrana zostaje ta z mniejszym indeksem

Strategię MinRow do obliczania minimalnego zbioru implikantów zastosowano w programie Hummingbird. Skuteczność tej strategii potwierdza wynik minimalizacji funkcji Kaz (Tabl. 2.25):

F = x4!x13x21 + x7!x15x16 + !x4!x17!x21

Warto zauważyć, że minimalizacja tej funkcji programem Espresso wcale nie jest lepsza:

F = !x2x14!x19x21 + !x8!x11!x12 + x5x8!x20

1.4.3 Minimalizacja metodą sekwencyjnego pokrywania

Strategia sekwencyjnego pokrywania zakłada uogólnienie pojedynczej kostki zbioru *F*, usunięcie kostek pokrytych przez kostkę uogólnioną, a następnie powtórzenie procesu dla pozostałych elementów zbioru *F*. Pozwala to na wyznaczenie zbioru kostek pokrywających wszystkie kostki zbioru *F*. Celem etapu uogólniania kostki jest zmniejszenie liczby jej zmiennych tak, aby uzyskana kostka pokrywała jak największą liczbę kostek zbioru *F*, nie pokrywając przy tym żadnej kostki zbioru *R*:

1. Oblicza się ekspansję kostki k_1 (ozn. $E(k_1)$) – tylko jedną.
2. Wykreśla się z macierzy ***F*** wszystkie wektory pokrywane przez $E(k_1)$.
3. Powstaje nowa macierz ***F'***, w której bierzemy pierwszą kostkę i postępujemy tak samo,
4. Proces kończy się, gdy pokryjemy (wykreślimy) wszystkie kostki ***F***,
5. Wynik (zminimalizowaną funkcję) tworzą kolejno obliczone ekspansje.

PRZYKŁAD 2.15

Funkcję boolowską opisaną zbiorami *F* i R zminimalizować metodą sekwencyjnego pokrywania.

F:	*R*:
00000	11101
11000	00010
11010	00110
01110	10001
11100	01100
01011	

Rozwiązanie

	k			
F:	1	00000	*R*:	11101
	2	11000		00010
	3	11010		00110
	4	01110		10001
	5	11100		01100
	6	01011		

Liczymy ekspansję k_1.

Ponieważ k_1 = (00000), to macierz blokująca B_1 jest identyczna z macierzą *R*.

	1 2 3 4 5	
B_1:	11101	~~1,2,3,5~~
	00010	4
	00110	~~3,4~~ $\Longrightarrow$ 3,4,5
	10001	1,5
	01100	2,3

Wypisujemy w każdym wierszu numery kolumn z jedynkami. Następnie wybieramy taki zbiór, który zapewni minimalne pokrycie kolumnowe. Do pokrycia wybieramy *L* = {3,4,5}; ($x_3\, x_4\, x_5$).

k_1^+ dla k_1=00000 będzie --000: $\bar{x}_3\bar{x}_4\bar{x}_5$ $k_1^+ \geq k_2$. Pokryta została także kostka k_2

k_3	11010		1 2 3 4 5	
	11101	B_3:	00111	3,4,5
	00010		11000	~~1,2~~
R:	00110		11100	~~1,2,3~~
	10001		01011	2,4,5
	01100		10110	~~1,3,4~~

PRZYKŁAD 2.15. c.d

Minimalne pokrycie zapewnia L={1,5} (x_1 x_5)

$$k_3^+ = 1\text{---}0 = x_1\bar{x}_5 \quad k_3^+ \geq k_5.$$

Kostka k_3^+ pokrywa także k_5, do dalszych obliczeń zostają kostki k_4, k_6

			1 2 3 4 5	
k_4	01110			
	11101	B_4:	10011	~~1,4,5~~
	00010		01100	~~2,3~~
R:	00110		01000	2
	10001		11111	~~1,2,3,4,5~~
	01100		00010	4

Minimalne pokrycie zapewnia L={2,4}; (x_2 x_4)

$$k_4^+ \geq k_6 = \text{-1-1} = x_2x_4 \qquad k_4^+ \geq k.$$

Zbierając wszystkie kostki pokrycia otrzymujemy funkcję minimalną:

$$f = \bar{x}_3\bar{x}_4\bar{x}_5 + x_1\bar{x}_5 + x_2x_4$$

Funkcja z przykładu 2.15 zminimalizowana programem InstantRS jest reprezentowana następującymi wyrażeniami:

1] !x1!x2!x4 + x1!x5 + x2x4

2] !x1!x3!x4 + x1!x5 + x2x4

3] !x2!x4!x5 + x1!x5 + x2x4

4] !x3!x4!x5 + x1!x5 + x2x4

Taki sam wynik uzyskamy z programu Hummingbird: f = !x2!x4!x5+x1!x5+x2x4. Nie oznacza to jednak, że strategia sekwencyjnego pokrywania jest tak samo skuteczna jak metoda systematyczna z algorytmem MinRow. Świadczy o tym wynik minimalizacji funkcji Kaz, obliczonej programem Blink. Blink reprezentuje dane wyjściowe w postaci tzw. Reguł decyzyjnych omawianych w rozdziale 5. Reguły te – dla funkcji Kaz – są następujące:

(x3=0) & (x4=1) & (x18=1) => (y=1)

(x2=1) & (x3=1) & (x5=1) => (y=1)

(x1=0) & (x3=1) & (x5=1) => (y=1)

(x1=0) & (x6=1) & (x9=1) => (y=1)

(x3=0) & (x6=1) & (x15=0) => (y=1)

(x1=0) & (x3=1) & (x13=0) => (y=1)

(x2=0) & (x5=1) & (x8=1) => (y=1)

Oczywiście możemy te reguły zapisać sumy iloczynów zmiennych boolowskich:

!x3x4x18 + x2x3x5 + !x1x3x5 + !x1x6x9 + !x3x6!x15 + !x1x3!x13 + !x2x5x8

Natomiast Kaz zminimalizowany programem Hummingbird jest wyrażeniem:

f = x4!x13x21 + x7!x15x16 + !x4!x17!x21

nieco prostszym od wyrażenia uzyskanego programem Espresso:

y1 = !x2x14!x19x21 + !x8!x11!x12 + x5x8!x20

1.5 Uzupełnienie (Complement)

Zadaniem procedury COMPLEMENT (uzupełnienie) jest obliczenie zbioru ***D***, tj. zbioru wszystkich nieokreśloności funkcji opisanej zbiorami ***F*** i ***R***. Zbiór ten jest wykorzystywany między innymi w procedurze pokrycia, a jego zwarta specyfikacja jest bardzo istotna z punktu widzenia złożoności obliczeń. Potrzebę obliczania ***D*** wyjaśnia dobrze przykład funkcji boolowskiej o 30 argumentach, opisanej tablicą prawdy o 200 wierszach (wektorach). Otóż zbiór wszystkich nieokreśloności takiej funkcji ma liczność $2^{30} - 200$, a więc bezpośrednia specyfikacja odpowiedniego ***D*** staje się niemal nierealna.

Procedura uzupełniania polega na iteracyjnym rozkładzie zbioru kostek reprezentującego funkcję *f* na kofaktory. Kofaktory te są obliczane tak długo, aż odpowiadające im zbiory kostek staną się „łatwe" do obliczenia ich uzupełnienia. Proces kończy „scalanie" wyników cząstkowych. Podstawą takiego postępowania jest spostrzeżenie, że uzupełnienie funkcji reprezentowanej rozkładem Shannona:

$$f = x_j f_{x_j} + \bar{x}_j f_{\bar{x}_j} \qquad f = x_j f_{x_j} + \bar{x}_j f_{\bar{x}_j}$$

można wyznaczyć obliczając najpierw ko faktory f_{x_j} oraz$f_{\bar{x}_j}$, a następnie ich uzupełnienie, czyli [1]:

$$\bar{f} = x_j \bar{f}_{x_j} + \bar{x}_j \bar{f}_{\bar{x}_j} \qquad (2\text{-}4)$$

Zatem $\bar{f}$ powstaje w wyniku „scalenia" wyników cząstkowych, tj. $\bar{f}_{x_j}$ oraz $\bar{f}_{\bar{x}_j}$.

W przypadku, gdy wynikiem rozkładu Shannona jest jednorodny zbiór kostek, obliczenia przejmuje procedura UNATE_COMPLEMENT, spełniająca rolę procedury wyznaczającej uzupełnienie funkcji jednorodnej. Wynika to z faktu, że dla funkcji monotonicznych, a w szczególności dla:

a) monotonicznie rosnącej w punkcie x_j,

b) monotonicznie malejącej w punkcie x_j,

uzupełnienie (2-4) upraszcza się do następujących wzorów [1]:

$$\bar{F} = \bar{x}_j \bar{F}_{\bar{x}_j} + \bar{F}_{x_j} \qquad \text{(2-5a)}$$

$$\bar{F} = x_j \bar{F}_{x_j} + \bar{F}_{\bar{x}_j} \qquad \text{(2-5b)}$$

Łatwo to wykazać, gdyż na przykład dla (5-6a) mamy, że $F_{x_j} \supseteq F_{\bar{x}_j}$ a więc, korzystając z implikacji $\boldsymbol{A} \supseteq \boldsymbol{B} \Rightarrow \boldsymbol{A} \cap \boldsymbol{B} = \boldsymbol{B}$, uzyskujemy:

$$F = x_j F_{x_j} + \bar{x}_j F_{\bar{x}_j} = x_j F_{x_j} + F_{\bar{x}_j} = F_{x_j}\left(x_j + F_{\bar{x}_j}\right)$$

czyli:

$$\bar{F} = \bar{x}_j \bar{F}_{\bar{x}_j} + \bar{F}_{x_j}$$

Obliczenie uzupełnienia funkcji jednorodnej rozpoczynamy od wyznaczania zero-jedynkowej macierzy ***M*** funkcji jednorodnej danej macierzy ***F***.

Przeliczenie macierzy ***F*** na macierz ***M*** przeprowadzamy według następujacego schematu:

M[*i*, *j*] = 0, jeżeli ***F***[*i*, *j*] = *,

M[*i*, *j*] = 1, jeżeli ***F***[*i*, *j*] = 1 lub ***F***[*i*, *j*] = 0.

Transformacja taka jest jednoznaczna, gdyż macierz ***F*** z założenia jest jednorodna (w żadnej z jej kolumn nie występują jednocześnie zera i jedynki). Następnie wyliczany jest wektor *V*, w którym „zapamiętuje się" polaryzacje zmiennych występujące w każdej kolumnie. Na przykład, zakładając macierz ***F*** o postaci:

$$\boldsymbol{F} = \begin{bmatrix} 01*0 \\ **1* \\ 0*1* \end{bmatrix}$$

otrzymamy macierz:

$$\boldsymbol{M} = \begin{bmatrix} 1100 \\ 0010 \\ 1010 \end{bmatrix}$$

oraz wektor $V = [0110]$

Otrzymaną macierz ***M*** będziemy rozkładać rekursywnie (stosując rozkład Shannona), aż do wystąpienia szczególnych postaci uzyskanych kofaktorów.

Obliczenie kofaktorów otrzymanej macierzy ***M*** rozpoczynamy od wyboru zmiennej do rozkładu. Odpowiedni wybór zmiennej ma istotne znaczenie dla redukcji obliczeń. Wybór zmiennej przeprowadzamy według następującego algorytmu:

1) wybieramy kostkę (wiersz macierzy ***M***) z największą liczbą zer,
2) w wybranej kostce wybieramy zmienne, które mają jedynkę w tej kostce,
3) spośród wybranych w punkcie 2) zmiennych wybieramy tę, która ma najwięcej jedynek w swojej kolumnie.

Postępowanie to uzasadniamy następująco. Otóż obliczenie kofaktora powoduje ustawienie na zero kolumny odpowiadającej wybranej zmiennej. Zatem wybranie zmiennej według powyższego algorytmu i obliczenie kofaktora ustawi na zero kolumny, które mają największą liczbę jedynek (p. 3). Ułatwi to uzyskanie sytuacji, w której w kofaktorze będzie wiersz samych zer (p. 1), odpowiadający tautologii.

Kofaktory macierzy ***M*** obliczamy według następującego schematu.

Kofaktor jedynkowy macierzy ***M*** względem zmiennej x_j otrzymujemy przez ustawienie wszystkich pozycji *j*-tej kolumny macierzy ***M*** na zera. Inaczej mówiąc, kofaktor jedynkowy jest przepisaniem macierzy ***M*** z „ustawieniem *j*-tej kolumny na zero".

Kofaktor zerowy macierzy ***M*** względem zmiennej x_j otrzymujemy przez wypisanie z ***M*** tych kostek (wierszy), w których zmienna x_j przyjmuje wartość zero.

Powyższy sposób obliczania kofaktorów jest bezpośrednią konsekwencją definicji kofaktora [2.8].

Na przykład dla macierzy ***M***:

$$\boldsymbol{M} = \begin{bmatrix} 1100 \\ 0010 \\ 1010 \end{bmatrix}$$

wybór zmiennej x_3 prowadzi do kofaktorów:

$$\boldsymbol{M} = \begin{bmatrix} 1101 \\ 0000 \\ 1000 \end{bmatrix} \text{ dla } x_3 = 1$$

oraz wektor $[1101]$ dla $x_3 = 0$

Następnym etapem obliczeń jest próba uzupełnienia otrzymanych kofaktorów. W szczególności, jeżeli którykolwiek z kofaktorów zawiera wiersz samych zer, to należy go usunąć, gdyż uzupełnienie takiego kofaktora jest zbiorem pustym.

Proces rozkładu na kofaktory prowadzimy rekursywnie, tzn. otrzymane kofaktory rozkładamy według tej samej zasady, aż do uzyskania kofaktorów, które zawierają tylko jedną kostkę (wiersz). Warto dodać, że jeżeli na którymś z poziomów rekursji w kolumnie odpowiadającej wybranej zmiennej są tylko jedynki, to kofaktor zerowy takiej macierzy jest pusty.

PRZYKŁAD 2.16

Dla zilustrowania metody rozważmy przykład jednorodnej funkcji *F* danej następującą macierzą:

$$\boldsymbol{F} = \begin{bmatrix} 0\ 1 * 0 \\ * * 10 \\ 01 ** \\ 0 ** 0 \end{bmatrix}$$

Obliczamy macierz ***M***:

$$\boldsymbol{M} = \begin{bmatrix} 1101 \\ 0011 \\ 1100 \\ 1001 \end{bmatrix}$$ oraz wektor $V = [0110]$.

Następnie przeprowadzamy rozkład jak na rys. 2.13. Rozkład jest tu realizowany kolejno według zmiennych x_i, tworząc odpowiednie wyniki rozkładu; z lewej strony zapisujemy wynik dla zmiennej x_i w postaci prostej, a z prawej dla tej zmiennej w postaci zanegowanej. Wybór zmiennej do rozkładu jest dokonywany według poprzednio podanej zasady: wybieramy wiersz z największą liczbą zer, a spośród kolumn wskazywanych (w tym wierszu) jedynką, wybieramy kolumnę z największą liczbą jedynek. Dlatego do pierwszego rozkładu wybieramy x_4 (drugi wiersz, czwarta kolumna w macierzy ***M***).

Możliwe są również inne wybory. Kofaktor dla x_4 powstaje przez zamianę wszystkich pozycji kolumny czwartej na zera. Kofaktor dla $\bar{x}_4$ powstaje przez wypisanie wierszy, które w czwartej kolumnie mają zera. Przechodząc do następnego rozkładu wykreślamy powtarzające się wiersze i wyznaczamy nową zmienną. W tym przypadku będzie to x_3 (może być również x_1). Decydując się na x_3 uzyskujemy odpowiednie macierze: dla x_3 jest to macierz z wierszem samych 0, czyli tautologia; a więc kończy to rozkład w tej gałęzi. Dla $\bar{x}_3$ musimy rozkładać dalej, względem zmiennej x_1. Rezultatem rozkładu jest macierz reprezentująca tautologię (gałąź x_1) i macierz reprezentująca pusty zbiór kostek, czyli ϕ.

Podstawą uzupełnienia funkcji jednorodnej jest rozszczepianie macierzy ***M*** na podzbiory kostek, których uzupełnienie łatwo można obliczyć. Taką postacią jest kofaktor o jednym wierszu. Uzupełnienie takiego kofaktora (w notacji ***M***) oblicza się według następujących zasad:

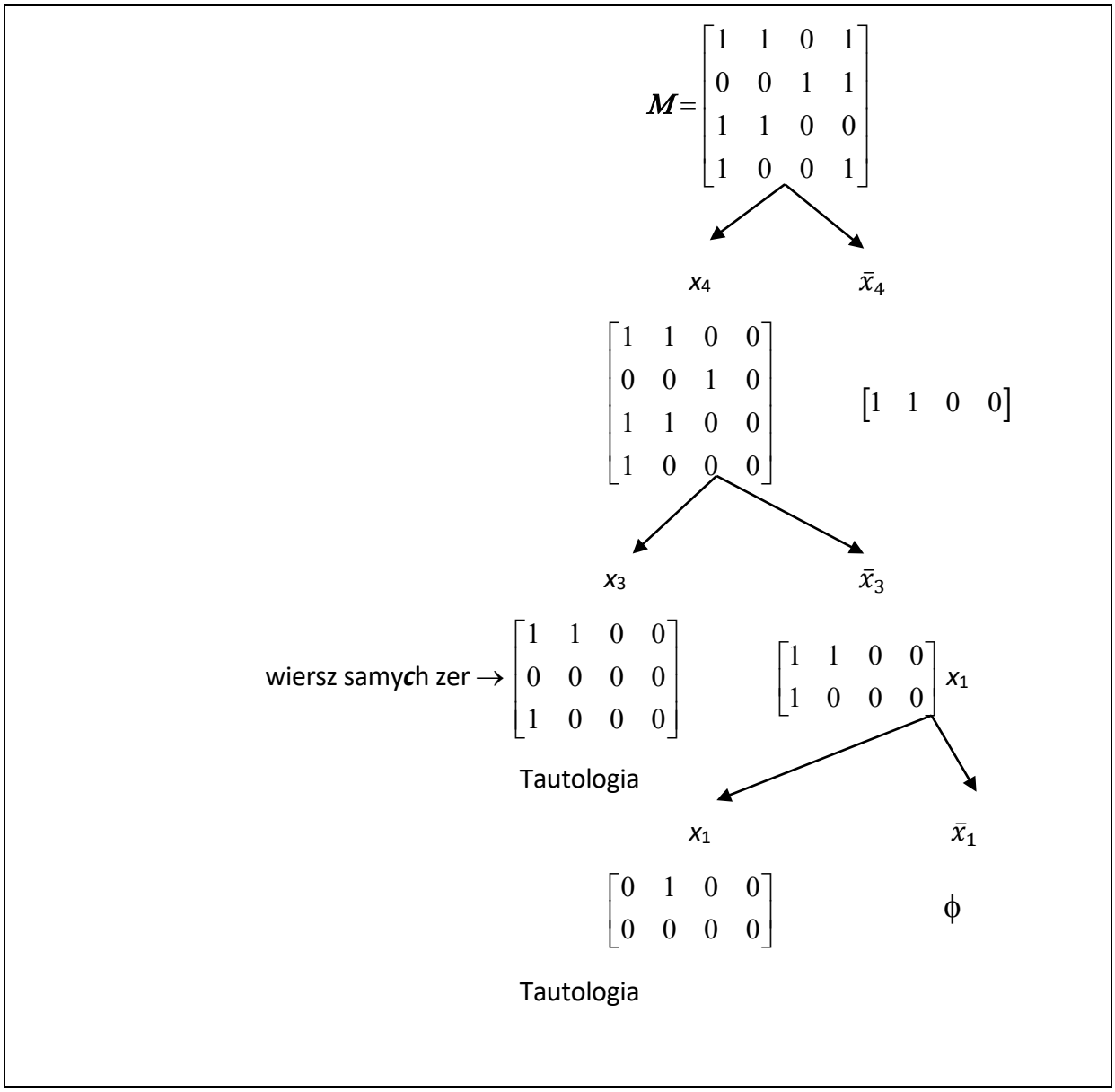
1) jeżeli kofaktor zawiera tylko jedną jedynkę, jego uzupełnienie jest identyczne jak ko faktor;

2) jeżeli kofaktor zawiera więcej niż jedną jedynkę, jego uzupełnienie zawiera tyle kostek (wierszy), ile jest jedynek w kofaktorze, przy czym wszystkie wiersze mają jedynkę (pozostałe pozycje zera) na pozycjach odpowiadających kolejnym jedynkom kofaktora.

Stosując te zasady do kofaktora uzyskanego dla $\bar{x}_4$: $[1100]$uzyskujemy:

$$\begin{bmatrix} 1000 \\ 0100 \end{bmatrix}$$

Uzasadnienie tego zapisu wynika bezpośrednio z prawa De Morgana. Również oczywiste jest uzupełnienie zbioru pustego, który powstał na drodze $x_4\bar{x}_3\bar{x}_1$; jest nim tautologia.



Rys. 2.13. Rozkład funkcji *F* z przykładu 2.16

Po obliczeniu uzupełnień na poszczególnych „piętrach" rozkładu możemy przystąpić do scalania poszczególnych wyników. Scalanie przeprowadzamy zgodnie z przetransformowanym do notacji ***M*** wzorem (2-5):

$$\boldsymbol{F} = x_j \cdot \boldsymbol{F}_0 + \boldsymbol{F}_1 .$$

Znaczy to, że jeżeli otrzymany kofaktor był zerowy (ozn. $\boldsymbol{F}_0$), to mnożymy go przez odpowiednie x_j i dodajemy, a jeżeli był jedynkowy (ozn. $\boldsymbol{F}_1$), to tylko dodajemy. Czyli (dla $\bar{x}_4$):

$$\begin{bmatrix}1000\\0100\end{bmatrix} \cdot [0001] = \begin{bmatrix}1001\\0101\end{bmatrix}$$

oraz dla $x_4\bar{x}_3\bar{x}_1$; kolejne obliczenia są następujące:

[1 0 0 0] + ϕ

[1 0 0 0] · [0 0 1 0] = 1 0 1 0.

Ostatni wynik należy dodać do wyniku uzyskanego dla $\bar{x}_4$):, czyli:

$$[0001] \cdot \begin{bmatrix} 0100 \\ 1000 \end{bmatrix} + [1010]$$

Łącząc wreszcie wyniki cząstkowe, uzyskujemy $\bar{\boldsymbol{M}}$, która po transformacji według wektora *v* prowadzi do $\bar{\boldsymbol{F}}$**:**

$$\bar{\boldsymbol{M}} = \begin{bmatrix} 1001 \\ 0101 \\ 1010 \end{bmatrix} \overset{V}{\rightarrow} \bar{\boldsymbol{F}} = \begin{bmatrix} 1 ** 1 \\ * 0 * 1 \\ 1 * 0 * \end{bmatrix}$$

Otrzymana macierz $\bar{\boldsymbol{F}}$ jest uzupełnieniem jednorodnej funkcji *F*. W ten sposób, mając funkcję rozłożoną na funkcje jednorodne, łatwo można obliczyć uzupełnienie każdej z nich.

1.6 Redukcja argumentów

1.6.1 Metoda klasyczna

Z praktycznego punktu widzenia istotne są takie realizacje funkcji boolowskich, w których argumenty $x_{i_1}, \ldots, x_{i_t}$, stanowią podzbiór zbioru $X\{x_1, \ldots, x_n\}$, gdyż uzyskuje się wtedy bezpośrednio realizację *n*-argumentowej funkcji *f* w układzie kombinacyjnym o *t* wejściach (*t* < *n*), co ma znaczenie w syntezie na typowych modułach PLD i w strukturach FPGA. Istotne jest również to, że (przy dużych *n*) zmniejszenie liczby argumentów z *n* do *t* upraszcza ewentualnie dalsze zabiegi optymalizacyjne, gdyż zmniejsza wymiary zadań dla procesu minimalizacji i dekompozycji. W szczególności redukcja argumentów stanowi podstawę dekompozycji równoległej, którą omówimy pod koniec niniejszego rozdziału.

Realizację funkcji $f = f(x_1, \ldots, x_n)$ taką, że:

$$f = h(x_{i_1}, \ldots, x_{i_t}), \qquad t < n$$

nazywać będziemy minimalno-argumentową realizacją funkcji f wtedy i tylko wtedy, gdy nie istnieje zbiór zmiennych:

$\left\{x_{j_1}, \ldots, x_{j_{t-1}}\right\} \subseteq X$, taki że $f = h\left(x_{j_1}, \ldots, x_{j_{t-1}}\right)$,

Zbiór $x_{i_1}, \ldots, x_{i_t}$ oznaczać będziemy X_t.

Niech $a \in \boldsymbol{B}^n$, $b \in \boldsymbol{B}^n$. Utwórzmy dla wektorów a, b wyrażenie boolowskie:

$$\delta(a,b) = \bigvee_i x_i$$

gdzie sumowanie jest po wszystkich i takich, że $a_i \neq b_i$. Inaczej mówiąc, x_i jest składnikiem $\delta(a,b)$ tylko wtedy, gdy wektory a oraz b mają różne składowe a_i, b_i. *Reprezentacją argumentową* funkcji f (oznaczaną dalej przez RA_f) nazywać będziemy wyrażenie boolowskie:

$$RA_f = \bigwedge_{a,b} \delta(a,b) \qquad (2\text{-}6)$$

gdzie iloczyn logiczny $\wedge$ jest brany po wszystkich parach $a, b : f(a) \neq f(b)$.

Można wykazać, że dla każdego rozwiązania równania $RA_f = \mathbf{1}$:

$$RA_f(x_{i_1}, \ldots, x_{i_t}) = \mathbf{1} \qquad (2\text{-}7)$$

istnieje funkcja h o argumentach $x_{i_1}, \ldots, x_{i_t}$, dla której $f(x_1, \ldots, x_n) = h(x_{i_1}, \ldots, x_{i_t})$

Dla dowodu powyższego wystarczy zauważyć, że jeżeli $RA_f(x_{i_1}, \ldots, x_{i_t}) = \mathbf{1}$, to dla każdej pary wektorów $a, b : f(a) \neq f(b)$, wektory $(a_{i_1}, \ldots, a_{i_t})$ oraz $(b_{i_1}, \ldots, b_{i_t})$ różnią się co najmniej jedną składową.

Jeżeli zbiór zmiennych $X_t = x_{i_1}, \ldots, x_{i_t}$, spełniających równanie (2-7) jest zbiorem o możliwie najmniejszej liczności, to funkcja h jest nazywana minimalno-argumentową realizacją funkcji f [1)].

Z konstrukcji wyrażenia RA_f wynika, że jest to koniunkcja czynników o postaci $(x_{i_1} \vee \ldots \vee x_{i_k})$ Rozwiązanie równania $RA_f = \mathbf{1}$ jest więc typowym *problemem pokrycia*, a odpowiednie obliczenia można uprościć wykorzystując pojęcie elementu zasadniczego, którego rolę w redukcji argumentów spełniać będzie omówione dalej pojęcie zmiennej niezbędnej.

Rozważmy funkcję f z tablicy 2.26a. Postaramy się zmniejszyć liczbę argumentów tej funkcji usuwając z niej po jednej zmiennej. Usuwając z tablicy 2.26a kolumnę odpowiadającą zmiennej x_4 uzyskujemy tablicę, w której dwa wiersze, o numerach 2 i 8, (tabl. 2.26b) są identyczne, ale wierszom tym odpowiadają różne wartości funkcji. Tablica jest więc sprzeczna. Stąd wniosek, że argumentu x_4 usunąć nie można.

[1)] Zbiór X_t jest również nazywany reduktem [2.10].

Tablica 2.26

a)

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	f
1	1	0	0	0	1	0	1	0
2	1	0	1	1	1	1	0	0
3	1	1	0	1	1	1	0	0
4	1	1	1	0	1	1	1	0
5	0	1	0	0	1	0	1	1
6	1	0	0	0	1	1	0	1
7	1	0	1	0	0	0	0	1
8	1	0	1	0	1	1	0	1
9	1	1	1	0	1	0	1	1

b)

	x_1	x_2	x_3	x_5	x_6	x_7	f
1	1	0	0	1	0	1	0
2	1	0	1	1	1	0	0
3	1	1	0	1	1	0	0
4	1	1	1	1	1	1	0
5	0	1	0	1	0	1	1
6	1	0	0	1	1	0	1
7	1	0	1	0	0	0	1
8	1	0	1	1	1	0	1
9	1	1	1	1	0	1	1

c)

	x_2	x_4	x_6	x_7	f
1	0	0	0	1	0
2	0	1	1	0	0
3	1	1	1	0	0
4	1	0	1	1	0
5	1	0	0	1	1
6	0	0	1	0	1
7	0	0	0	0	1
8	0	0	1	0	1
9	1	0	0	1	1

Natomiast usuwając z tablicy 2.26a zmienną x_3 uzyskujemy tablicę, w której wszystkie wiersze są różne. Można nawet usunąć jeszcze zmienne x_1 i x_5 (tabl. 2.26c), a łączny rezultat tych usunięć nie spowoduje pojawienia się sprzeczności. Spostrzeżenie powyższe prowadzi do następującej definicji.

Oznaczmy przez $T_f(X - \{x_i\})$ tablicę specyfikacji funkcji f uzyskaną z T_f przez usunięcie kolumny x_i, gdzie T_f jest tablicą funkcji f.

Zmienną x_i nazywamy *zmienną niezbędną* [2.6], jeżeli tablica specyfikacji $T_f(X - \{x_i\})$ jest sprzeczna.

Zmiennymi niezbędnymi funkcji f z tablicy 2.26a są x_4 oraz x_6, gdyż tablice $T_f(X - \{x_4\})$, $T_f(X - \{x_6\})$ są sprzeczne.

Można wykazać, że zmienna x_i jest *zmienną niezbędną* funkcji $f(x_1,\ldots,x_n)$, jeśli istnieją wektory a, b: $f(a) \neq f(b)$ takie, że $a_i \neq b_i$ oraz $a_j = b_j$ dla każdego $j \neq i$.

Przyjmując do opisu funkcji f: $\boldsymbol{B}^n \rightarrow \{0,1,-\}$ podziały na zbiorze $\boldsymbol{K}$ ponumerowanych wektorów z $\boldsymbol{B}^n$, możemy podane wyżej spostrzeżenia wyrazić analitycznie. Oznaczmy przez $P(x_i)$ podział na $\boldsymbol{K}$ realizowany zmienną x_i, a przez P_f, podział wyjściowy funkcji f. Wtedy warunek na minimalno-argumentową realizację funkcji f można sprawdzić za pomocą nierówności:

$$P\left(x_{i_1}\right) \cdot P\left(x_{i_2}\right) \cdot \ldots \cdot P\left(x_{i_t}\right) \leq P_f$$

PRZYKŁAD 2.17

Obliczymy wszystkie realizacje minimalno-argumentowe funkcji $f = f(x_1,\ldots, x_7)$, której tablica prawdy podana jest w tabl. 2.26a, z odpowiednią numeracją wektorów.

Wektory prawdziwe i fałszywe tej funkcji ponumerowano liczbami naturalnymi 1,...,9, czyli $\boldsymbol{K} = \{1,\ldots,9\}$. Mamy wtedy (przy oznaczeniach $P(x_i) = P_i$) następujące podziały na $\boldsymbol{K}$:

$$P_1 = \{\overline{5}; \overline{1,2,3,4,6,7,8,9}\}$$

$$P_2 = \{\overline{1,2,6,7,8}; \overline{3,4,5,9}\}$$

$$P_3 = \{\overline{1,3,5,6}; \overline{2,4,7,8,9}\}$$

$$P_4 = \{\overline{1,4,5,6,7,8,9}; \overline{2,3}\}$$

$$P_5 = \{\overline{7}; \overline{1,2,3,4,5,6,8,9}\}$$

$$P_6 = \{\overline{1,5,7,9}; \overline{2,3,4,6,8}\}$$

$$P_7 = \{\overline{2,3,6,7,8}; \overline{1,4,5,9}\}$$

$$P_f = \{\overline{1,2,3,4}; \overline{5,6,7,8,9}\}$$

Jak już stwierdziliśmy, zmiennymi niezbędnymi tej funkcji są x_4, x_6. Dlatego najpierw wyznaczymy iloczyn $P = P_4 \cdot P_6$:

$$P_4 \cdot P_6 = \{\overline{1,5,7,9}; \overline{4,6,8}; \overline{2,3}\}$$

Blokami iloczynu $P = (B_1,\ldots, B_3)$ są $B_1 = \{1,5,7,9\}$, $B_2 = \{4,6,8\}$, $B_3 = \{2,3\}$. W dalszych obliczeniach pomijamy blok B_3, gdyż B_3 jest zawarty w jednym bloku podziału P_f.

Kolejną czynnością jest obliczenie podziału ilorazowego $P_4 \cdot P_6 | P_f$:

$$P_4 \cdot P_6 | P_f = \{\overline{(1)(5,7,9)}; \overline{(4)(6,8)}; \overline{(2,3)}\} \qquad (2\text{-}8)$$

Obliczony podział ilorazowy pozwala wyznaczyć czynniki wyrażenia boolowskiego RA_f. Wyznaczane są one dla każdej pary mintermów p, q, takiej, że p i q należą do jednego bloku iloczynu $P_4 \cdot P_6$, ale do różnych bloków podziału P_f, co łatwo odczytać z podziału ilorazowego biorąc pary p, q z różnych elementów tego podziału. Przez elementy rozumiemy tu podzbiory ograniczone różnymi nawiasami. Dla każdej tak wyznaczonej pary p, q sprawdzamy na jakich pozycjach, czyli dla jakich zmiennych różnią się odpowiadające im wektory w tablicy prawdy. Na przykład, dla pary $p = 1$, $q = 5$ mamy, że odpowiednie wektory w tablicy prawdy funkcji f (tabl. 4.17a) różnią się na pozycjach x_1, x_2, a więc odpowiadający tej parze czynnik RA będzie $C_1 = x_1 \vee x_2$. Zestawienie zmiennych, dla których różnią się pary p, q nazywać będziemy tablicą porównań. Odczytując z podziału ilorazowego (2-8) pary p, q otrzymujemy następująca listę porównań:

PRZYKŁAD 2.17 c.d.

1,5: x_1, x_2

1,7: x_3, x_5, x_7

1,9: x_2, x_3

4,6: x_2, x_3, x_7

4,6: x_2, x_7

Redukując w zbiorach C każde C_i, dla którego istnieje $C_j: C_j \subseteq C_i$, otrzymujemy równanie $RA = \mathbf{1}$ o postaci:

$$(x_1 + x_2)(x_3 + x_5 + x_7)(x_2 + x_3)(x_2 + x_7) = \mathbf{1}$$

które przekształcamy do postaci sumo-iloczynowej:

$x_2x_3 + x_2x_5 + x_2x_7 + x_1x_3x_7$

Na tej podstawie wyznaczamy minimalne rozwiązania o najmniejszej liczności: {x_2, x_3, x_4, x_6}, {x_2, x_4, x_5, x_6}, {x_2, x_4, x_6, x_7}. Stąd wniosek, że siedmioargumentowa funkcja f w rzeczywistości jest zależna wyłącznie od czterech argumentów. Zauważmy też, że ostatnie z podanych rozwiązań oznacza usunięcie z tablicy funkcji f kolumn odpowiadających zmiennym x_1, x_3, x_5. Sytuacja taka była już prezentowana w tablicy 2.26c.

Skuteczność redukcji argumentów, a także jej wpływ na inne procedury logicznej pokażemy na bardziej skomplikowanym przykładzie.

PRZYKŁAD 2.28

Dla funkcji *F* podanej w tablicy 2.27 obliczyć wszystkie minimalne zbiory argumentów, od których ta funkcja zależy. Zmienne niezbędne tej funkcji to: x_1, x_3, x_7.

Kolejno obliczamy podziały: według zmiennych niezbędnych, ilorazowy oraz listę porównań.

$$P_1 \cdot P_3 \cdot P_7 = \{\overline{1,4,8}; \overline{2,7}; \overline{3}; \overline{5,6,10}; \overline{9}\}$$

$$P_1 \cdot P_3 \cdot P_7 | P_F = \{\overline{(1)(4)(8)}; \overline{(2)(7)}; \overline{(3)}; \overline{(5)(6)(10)}; \overline{(9)}\}$$

Tablica 2.27

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_1	y_2
1	0	0	0	1	0	0	1	1	0	0	1
2	0	1	1	0	0	0	0	1	0	0	1
3	1	1	1	0	1	1	0	0	0	1	0
4	0	1	0	0	0	1	1	0	1	0	0
5	0	0	1	0	1	1	1	0	0	0	1
6	0	1	1	0	0	0	1	1	0	0	0
7	0	1	1	0	1	1	0	1	1	1	1
8	0	0	0	1	0	0	1	0	1	1	1
9	1	1	1	0	0	0	1	1	0	1	0
10	0	0	1	1	0	0	1	0	1	1	0

Lista porównań (zapis wg indeksów zmiennych):

1|4 : ~~2,4,6,8,9~~
1|8 : 8,9
4|8 : 2,4,6
2|7 : 5,6,9
5|6 : 2,5,6,8
5|10 : ~~4,5,6,9~~
6|10 : ~~2,4,8,9~~

Na podstawie tablicy porównań tworzymy wyrażenie boolowskie, które w zapisie według indeksów zmiennych x_i jest następujące:

$(8 + 9)(2 + 4 + 6)(5 + 6 + 9)(2 + 5 + 6 + 8) = (9 + 8)(9 + 5 + 6)(2 + 6 + 4)(2 + 6 + 5 + 8) =$
$= [9 + 8(5 + 6)][2 + 6 + 4(5 + 8)] = (9 + 5 \cdot 8 + 6 \cdot 8)(2 + 6 + 4 \cdot 5 + 4 \cdot 8) =$
$= 2 \cdot 9 + 6 \cdot 9 + 4 \cdot 5 \cdot 9 + 4 \cdot 8 \cdot 9 + 2 \cdot 5 \cdot 8 + \sout{5 \cdot 6 \cdot 8} + 4 \cdot 5 \cdot 8 + \sout{4 \cdot 5 \cdot 8} + \sout{2 \cdot 6 \cdot 8} +$
$+ 6 \cdot 8 + \sout{4 \cdot 5 \cdot 6 \cdot 8} + \sout{4 \cdot 6 \cdot 8}$

Stąd wszystkie minimalne rozwiazania:

x_1, x_2, x_3, x_7, x_9
x_1, x_3, x_6, x_7, x_9
x_1, x_3, x_6, x_7, x_8
$x_1, x_3, x_4, x_5, x_7, x_9$
$x_1, x_3, x_4, x_7, x_8, x_9$
$x_1, x_2, x_3, x_5, x_7, x_8$
$x_1, x_3, x_4, x_5, x_7, x_8$

W bardziej skomplikowanych przykładach można zaobserwować wpływ redukcji argumentów na proces realizacji funkcji w postaci minimalnych wyrażeń boolowskich. Na przykład dla funkcji TL27 z tabl. 2.28 można obliczyć następujące redukty.

Tablica 2.28

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	y
1	0	0	1	0	1	1	1	0	1	0	0
2	1	0	1	0	0	1	0	1	0	0	0
3	0	1	0	0	0	1	1	1	1	0	0
4	1	0	1	1	1	0	1	0	1	1	0
5	1	1	0	0	0	1	0	0	1	1	0
6	0	1	0	0	0	1	0	1	1	0	0
7	1	1	1	0	1	0	0	1	1	0	0
8	0	1	0	0	1	1	0	0	0	0	0
9	0	1	0	1	0	0	0	0	1	0	0
10	0	1	1	1	1	1	1	0	1	1	1
11	0	0	0	0	0	1	0	1	0	0	1
12	1	1	0	1	1	1	0	0	1	1	1
13	0	1	0	0	1	0	0	0	0	0	1
14	0	1	0	0	0	1	1	1	1	1	1
15	0	0	1	0	0	0	0	1	1	0	1
16	1	1	1	1	0	1	0	0	0	1	1
17	1	1	1	1	1	0	1	0	0	1	1
18	1	1	1	1	1	1	1	1	1	1	1
19	0	0	1	0	0	0	0	0	0	0	1
20	1	1	0	1	1	0	0	1	1	1	1
21	0	0	1	0	0	0	1	1	1	1	1
22	1	1	1	1	1	0	0	0	1	0	1
23	1	0	1	0	1	1	1	1	0	1	1
24	0	1	1	0	0	0	0	1	1	0	1
25	0	1	0	0	1	1	1	0	0	0	1

$R_1 = \{x_1, x_2, x_4, x_5, x_6, x_7, x_{10}\}$

$R_2 = \{x_1, x_2, x_4, x_6, x_7, x_8, x_{10}\}$

$R_3 = \{x_1, x_2, x_4, x_6, x_7, x_9, x_{10}\}$

$R_4 = \{x_1, x_4, x_5, x_6, x_7, x_9, x_{10}\}$

$R_5 = \{x_1, x_4, x_6, x_7, x_8, x_9, x_{10}\}$

$R_6 = \{x_1, x_5, x_6, x_7, x_8, x_9, x_{10}\}$

$R_7 = \{x_2, x_3, x_4, x_5, x_6, x_7, x_{10}\}$

$R_8 = \{x_2, x_3, x_5, x_6, x_7, x_8, x_{10}\}$

$R_9 = \{x_3, x_4, x_5, x_6, x_7, x_9, x_{10}\}$

$R_{10} = \{x_3, x_5, x_6, x_7, x_8, x_9, x_{10}\}$

Wybierając jeden z tych zbiorów (np. R_3) możemy dla zredukowanej w ten sposób funkcji zastosować inne procedury optymalizacji np. minimalizację. Uzyskamy wtedy następujące wyrażenie:

$$f = \bar{x}_1\bar{x}_2\bar{x}_7 + x_1x_2x_4 + \bar{x}_1x_{10} + \bar{x}_1\bar{x}_4\bar{x}_6 + x_7\bar{x}_9$$

Warto podkreślić, że rozwiązanie wyznaczone za pomocą programu ESPRESSO [2.1] prowadzi do wyrażenia

$$f = \bar{x}_5\bar{x}_6x_8 + \bar{x}_1\bar{x}_2\bar{x}_5 + x_5\bar{x}_6\bar{x}_8\bar{x}_{10} + +x_4\bar{x}_7\bar{x}_{10} + x_7\bar{x}_9 + x_6x_7x_{10}$$

zawierającego 9 argumentów. Jak widać, nie prowadzi ono do realizacji minimalno-argumentowej. Jeśli jednak procedurę minimalizacji zastosujemy do funkcji o zredukowanych argumentach, to uzyskamy wyrażenie o mniejszej liczbie składników iloczynowych.

1.6.2 Redukcja argumentów metodą uzupełniania

Wyjątkowo skuteczna w redukcji argumentów okazała się metoda zaproponowana w [2.6], w której klasyczne rozwiązanie problemu transformacji CNF na DNF zostało zredukowane do obliczenia uzupełnienia

CNF tej funkcji, gdzie uzupełnienie redukuje się do obliczenia pokrycia kolumnowego macierzy binarnej [??]. Zaproponowane podejście bardzo przyspiesza obliczenia, a wydajna reprezentacja algorytmu w pamięci operacyjnej maszyny obliczeniowej pozwala na osiągnięcie wyników, które nie mogą być osiągnięte przy użyciu innych publikowanych metod i systemów [2.1]. Z tych powodów, jak też ze względu na niezbędność stosowania algorytmu redukcji w realizacjach sprzętowych układów dystrybucji adresów IP, skanowania wirusów, itp., wykonano praktycznie użyteczne oprogramowanie Lightning, wykorzystujące zasadę uzupełniania. Zasadę tę wyjaśnimy posługując się funkcją z Przykładu 2.17 (tabl. 2.26a).

PRZYKŁAD 2.19

Obliczoną w Przykładzie 2.17 zredukowaną tablicę porównań:

1,5: x_1, x_2

1,7: x_3, x_5, x_7

1,9: x_2, x_3

4,6: x_2, x_7

zapisujemy w postaci binarnej macierzy M:

$$\boldsymbol{M} = \begin{matrix} 1\,2\,3\,4\,5\,6\,7 \\ \begin{bmatrix} 1\,1\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,1\,0\,1 \\ 0\,1\,1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,1 \end{bmatrix} \end{matrix}$$

Obliczanie uzupełnienia funkcji jednorodnej polega na rozkładzie macierzy *M* na podzbiory kostek, których uzupełnienie można łatwo obliczyć. Taką postacią jest kofaktor o jednym wierszu. Uzupełnienie takiego kofaktora oblicza się według następujących zasad:

1) Jeżeli kofaktor zawiera tylko jedną jedynkę, jego uzupełnienie jest identyczne jak kofaktor.
2) Jeżeli kofaktor zawiera więcej niż jedną jedynkę, jego uzupełnienie zawiera tyle kostek (wierszy), ile jest jedynek w kofaktorze, przy czym wszystkie wiersze uzupełnienia mają jedynkę tylko na pozycjach odpowiadających kolejnym jedynkom kofaktora. Takie postępowanie wynika z prawa de Morgana. Przykładowo, dla kofaktora [0 0 1 0 1 0 1] uzupełnieniem będzie macierz:

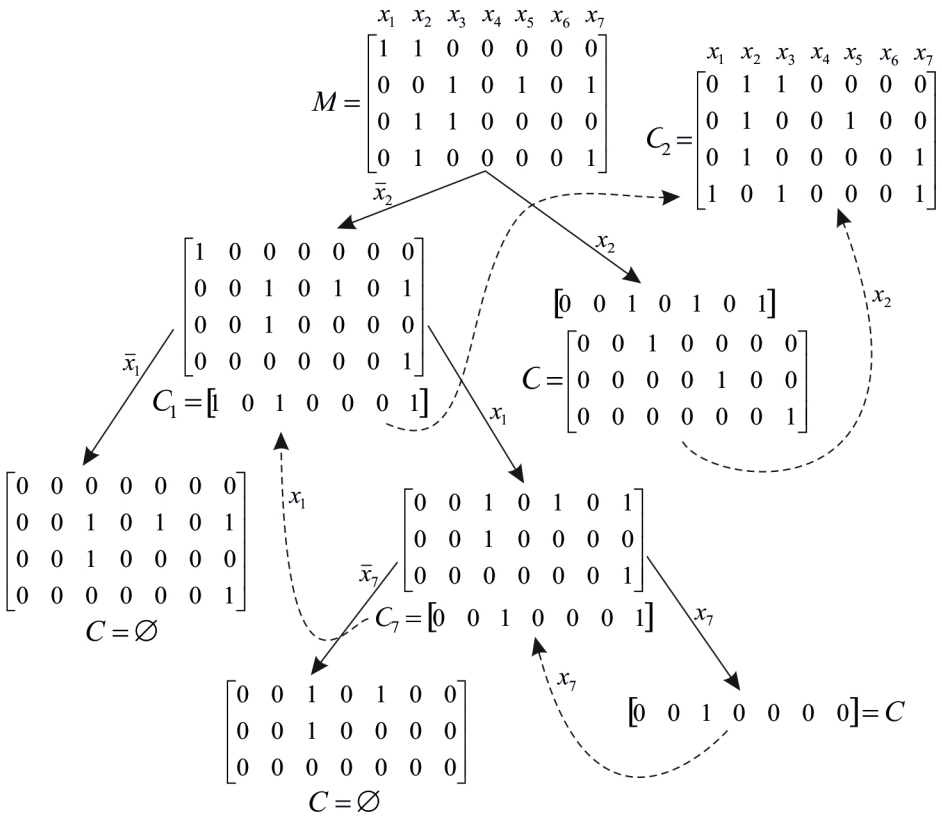
   $$\begin{bmatrix} 0010000 \\ 0000100 \\ 0000001 \end{bmatrix}$$

3) Jeżeli kofaktor zawiera wiersz samych zer (tautologia), to jego uzupełnieniem jest zbiór pusty.
4) Jeżeli ko faktor jest zbiorem pustym, to jego uzupełnieniem jest wiersz samych zer (tautologia).

Po obliczeniu uzupełnień na poszczególnych poziomach rozkładu należy scalić wyniki cząstkowe zgodnie ze wzorem:

$$\bar{f} = x_j \bar{f}_{x_j} + \bar{f}_{\bar{x}_j}$$

Realizowany według powyższych zasad rozkład macierzy *M* podany jest na rys. 2.14 (linia ciągła). Linią przerywaną oznaczono kolejne etapy scalania wyników dopełnienia.



Rys. 2.14. Przykład obliczania uzupełnienia funkcji monotonicznej

Na rys. 2.14 podano również wyniki scalania macierzy cząstkowych uzyskanych na poszczególnych etapach:

$C_7 = x_7\,[0\,0\,1\,0\,0\,0\,0] + \varnothing = [0\,0\,1\,0\,0\,0\,1]$

$C_1 = x_1\,[0\,0\,1\,0\,0\,0\,1] + \varnothing = [1\,0\,1\,0\,0\,0\,1]$

$$C_2 = x_2 \cdot \begin{bmatrix} 0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1 \end{bmatrix} + [1\,0\,1\,0\,0\,0\,1] = \begin{matrix} 1\,2\,3\,4\,5\,6\,7 \\ \begin{bmatrix} 0\,1\,1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,1\,0\,0 \\ 0\,1\,0\,0\,0\,0\,1 \\ 1\,0\,1\,0\,0\,0\,1 \end{bmatrix} \end{matrix}$$

Macierz uzyskana na najwyższym piętrze rozkładu, która została utworzona ze scalenia macierzy cząstkowych dla zmiennej x_2, reprezentuje – pozycją jedynek w poszczególnych wierszach – redukty funkcji z Przykładu 2.19. Obliczone uzupełnienie należy interpretować w następujący sposób: jedynka w *j*-tej kolumnie oznacza, że minimalny redukt uwzględnia zmienną x_j. Do uzupełnienia należy dodać zmienne niezbędne x_4 oraz x_6. Minimalne redukty funkcji *F* są następujące i takie same jak w rozwiązaniu metodą klasyczną.

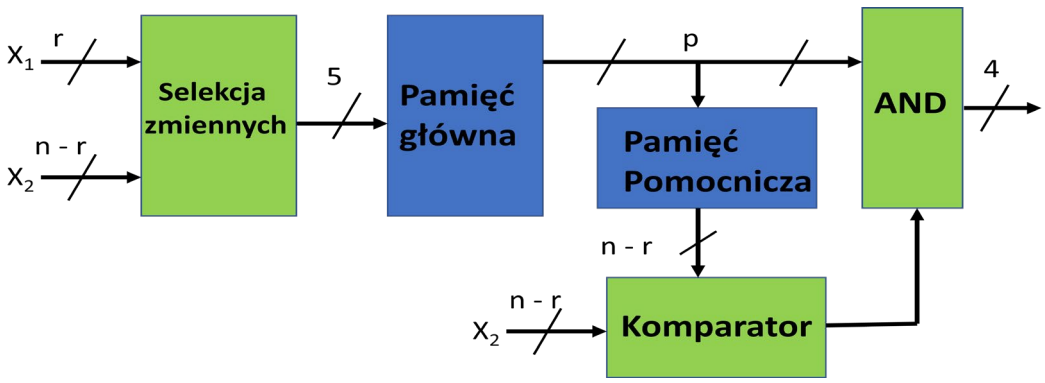
$\{x_2,x_3,x_4,x_6\}$
$\{x_2,x_4,x_5,x_6\}$
$\{x_2,x_4,x_6,x_7\}$
$\{x_1,x_3,x_4,x_6,x_7\}$

Redukcja argumentów oraz dekompozycja, to – zgodnie z opinią prof. Sasao wyrażoną w referacie [2.14] prezentowanym na konferencji EPFL Workshop on Logic Synthesis & Verification, Dec. 2015 – jedno z najpilniejszych zadań syntezy logicznej.

W przypadku implementacji funkcji boolowskich w układach FPGA (*Field Progammable Gate Array*) redukcja argumentów, umożliwia realizację funkcji w układzie kombinacyjnym o mniejszej liczbie wejść. Jest to szczególnie istotne dla struktur FPGA z wbudowanymi pamięciami, dla których liczba zmiennych wejściowych jest głównym czynnikiem decydującym o złożoności implementacji. Jeszcze większe znaczenie redukcji argumentów dotyczy syntezy funkcji generowania indeksów. Funkcje generowania indeksów to silnie nieokreślone funkcje boolowskie:

$f : D^n \rightarrow \{1,2,\ldots,k\}$, gdzie $D^n \subseteq \{0,1\}^n$, oraz $|D^n| = k$.

Wektory należące do zbioru D^n nazywa się wektorami rejestrowanymi. Cechą charakterystyczną tych funkcji jest duża liczba zmiennych wejściowych, przy stosunkowo małej liczności zbioru D^n . Konsekwencją tej własności jest skuteczne redukowanie zmiennych wejściowych, umożliwiające realizację generatora indeksów w strukturze zaproponowanej przez Sasao (rys. 2.15). W tak zbudowanym generatorze indeksów wyróżnia się dwie pamięci: pamięć główną i pamięć pomocniczą, komparator i bramę AND.



Rys. 2.15. Generator indeksów

Blok selekcji zmiennych wybiera spośród wszystkich zmiennych podzbiór X_1, reprezentujący minimalną liczbę zmiennych (tzw. redukt) realizowanej Funkcji Generowania Indeksów (FGI). Liczność reduktu jest r. Jest to jednocześnie liczba wejść do pamięci głównej. Na wyjściach pamięci głównej pojawia się indeks wektora wejściowego $X = X_1 + X_2$. Jest on reprezentowany wektorem binarnym o liczbie bitów $p = \lceil \log_2(k + 1) \rceil$. Indeks ten jest poprawnym indeksem wektora wejściowego wtedy, jeżeli jest to wektor rejestrowany. I tylko wtedy indeks ten pojawi się na wyjściach generatora. W przypadku, gdy wektor wejściowy nie jest wektorem rejestrowanym na wyjściach generator pojawi się 0. Sprawdzenie, czy wyjścia pamięci głównej reprezentują poprawny indeks jest realizowane w pamięci pomocniczej i komparatorze.

W pamięci pomocniczej są zapisane wektory reprezentowane zmiennymi należącymi do zbioru X_2. Wektory te są podawane na wejście komparatora i porównywane z rzeczywistym wektorem X_2 podawanym na drugie wejście komparatora. Oczywiście wektor pobrany z pamięci pomocniczej będzie różny od rzeczywistego wektora X_2 w przypadku, gdy nie należy on do wektora rejestrowanego. Wtedy komparator sygnałem wyjściowym o 0 zablokuje pojawienie się na wyjściach bramy AND wektora wytworzonego w pamięci głównej.

Działanie tak skonstruowanego generatora indeksów można zaprezentować na przykładzie funkcji generowania indeksów analizowanej w referacie Sasao – tab. 2.29a. Dla funkcji tej wyznaczono 5-argumentowy redukt – tab. 2.29b.

Tablica 2.29

a)

		b)	c)
01100001000100001010011000010001000010 10	1	01100	0000
01011111011010100011010111110110100011	2	01011	1011
11110101011101110000111101010111011 10001	3	11110	1111
00011110000100010111000111100001000 10111	4	00011	1110
00111100000001000101001111000000010001 01	5	00111	1010
01110010010001001001011100100100010010 01	6	01110	0010
00100011100011110010001000111000111100 10	7	00100	0100
11111111101000111100010001110001111001 0	8	11111	1100
11101110001100010110111011100011000101 10	9	11101	1101
10100001101001000011101000011010010000 11	10	10100	0001

Nietrudno zauważyć, że o skuteczności tak skonstruowanego generatora indeksów decyduje skuteczność obliczania reduktów. Stosując program Lightning można obliczyć całą serię reduktów 4-argumentowych. Jedno z tych rozwiązań podano w tab. 2.29c.

1.7 Analiza i eksploracja danych

1.7.1 Wprowadzenie

W praktyce, metody syntezy logicznej są wykorzystywane głównie do optymalizacji systemów cyfrowych, które przetwarzają sygnały binarne. Podstawowym zadaniem tych metod jest poprawa implementacji oraz odwzorowania systemów w różnej technologii. Jednakże można wykazać, że wiele metod syntezy logicznej, a w szczególności te wykorzystywane do optymalizacji kombinacyjnych układów logicznych, może być z powodzeniem zastosowanych w typowych zadaniach przetwarzania i wyszukiwania informacji, eksploracji danych, a także w dziedzinie systemów ekspertowych, maszynowego uczenia, czy sztucznej inteligencji [2.21], [2.22].

Przez eksplorację danych, znaną również pod nazwą odkrywania wiedzy w bazach danych, jest rozumiany proces automatycznego pozyskiwania znaczących, ale dotychczas nieznanych informacji z baz danych. Dlatego te informacje określa się jako „ukryte", a celem jest te informacje wyekstrahować. W wyniku eksploracji danych można na pewnym poziomie abstrakcji: zdiagnozować pacjenta, przeprowadzić sondaż, np. przed wyborami prezydenckimi, klasyfikować dane internetowe, czy podjąć decyzję o przyznaniu bądź odrzuceniu kredytu.

W większości zastosowań głównym zadaniem eksploracji danych jest indukcja reguł decyzyjnych, które są obliczane na podstawie wyników badań i pomiarów zgromadzonych w bazie danych. Wygenerowane reguły (zwane również klasyfikatorami) umożliwiają podjęcie decyzji. Typowym przykładem bazy danych oraz jej analizy jest Wisconsin Breast Cancer Database (źródło: Dr William H. Wolberg, University of Wisconsin Hospital, Madison, Wisconsin, USA), w której diagnoza raka piersi jest realizowana za pomocą bazy danych o dziewięciu atrybutach, zgromadzonej dla 699 pacjentek [2.29].

Systemy decyzyjne i kombinacyjne układy logiczne są bardzo podobne. System decyzyjny jest zwykle opisany przez tablicę decyzyjną, natomiast kombinacyjny układ logiczny przez tablicę prawdy. Atrybuty warunkowe systemu decyzyjnego odpowiadają zmiennym wejściowym układu logicznego, a atrybuty decyzyjne – zmiennym wyjściowym. Stąd wiele pojęć z tych obydwu obszarów może być wzajemnie na siebie odwzorowanych, a podobieństwo systemów decyzyjnych oraz układów logicznych pozwala na wykorzystanie specjalistycznych metod syntezy logicznej w dziedzinie eksploracji danych.

Na przykład zadanie redukcji danych w systemach informacyjnych jest rozwiązywane przez minimalizację liczby cech (atrybutów/parametrów), a następnie usunięcie nadmiarowych obiektów. Podobnym zadaniem w dziedzinie syntezy logicznej jest redukcja argumentów.

Innym zagadnieniem eksploracji danych jest podejmowanie decyzji na podstawie wcześniej zgromadzonych danych. Polega ono na uogólnianiu wiedzy oraz indukowaniu reguł decyzyjnych. W wyniku indukcji otrzymuje się zbiór reguł logicznych, który pozwala podejmować decyzje nie tylko dla obiektów należących do bazy pierwotnej, dla której przeprowadzono obliczenia, ale przede wszystkim dla nowych

obiektów do niej nie należących. Jest to bardzo ważne w przypadku zadań maszynowego uczenia. Analogicznym zagadnieniem do indukcji reguł z dziedziny eksploracji danych jest zagadnienie minimalizacji funkcji logicznych z dziedziny syntezy logicznej. Z uwagi na inne interpretacje i aplikacje te zagadnienia wydają się być zupełnie różne, aczkolwiek jest to stwierdzenie błędne.

Celem rozdziału jest wskazanie i omówienie możliwości zastosowania zaawansowanych algorytmów syntezy logicznej w typowych zadaniach eksploracji danych, takich jak: ekstrakcja cech, indukcja reguł decyzyjnych i wielu innych.

1.7.2 Systemy informacyjne oraz systemy decyzyjne

Wiele rzeczywistych problemów oraz zdarzeń może zostać opisanych przy użyciu baz danych (tablic danych), czyli tak zwanych systemów informacyjnych. Na przykład, za pomocą tych tablic możemy opisywać wybrane parametry i stan pacjentów w czasie badań medycznych. Wtedy poszczególne instancje zapisane w wierszach tablicy charakteryzują pacjenta przez odpowiednie wartości parametrów (atrybutów). Na przykład, jeżeli obiektem jest KOWALSKI, atrybutem zaś wiek, to wartością tego atrybutu dla obiektu KOWALSKI może być np. MŁODY. Rozważania dotyczące algorytmów eksploracji danych ograniczymy w większości przypadków do systemów informacyjnych o specyficznej strukturze, a mianowicie do tablic decyzyjnych, których zastosowania w systemach podejmowania i wspomagania decyzji, a także w wielu zadaniach maszynowego uczenia, są coraz powszechniejsze.

Formalnie, parę $A = (U, A)$ nazywamy *systemem informacyjnym,* gdzie U – jest niepustym, skończonym zbiorem *obiektów*, A – jest niepustym, skończonym zbiorem *atrybutów,* tj. każdy element $a \in A$ jest funkcją z U w V_a, gdzie zbiór V_a jest dziedziną parametru a i jest nazywany *zbiorem wartości* dla parametru a. Wtedy funkcja ρ odwzorowuje produkt U oraz A w zbiór wszystkich wartości. Przez $\rho(u_t, a_i)$, gdzie $u_t \in U$, $a_i \in A$, oznaczamy wartość atrybutu dla danego obiektu.

Często zbiór atrybutów A ma jeden lub więcej atrybutów wyróżnionych lub jest o taki atrybut rozbudowywany. Celem tych atrybutów jest podejmowanie decyzji na podstawie informacji zawartej w bazie danych. Formalnie, *systemem decyzyjnym* jest system informacyjny postaci $A = (U, A \cup D)$, gdzie $A \cap D = \varnothing$. Atrybuty w zbiorze A nazywamy *atrybutami warunkowymi,* natomiast atrybuty w zbiorze D nazywamy *atrybutami decyzyjnymi.* Systemy decyzyjne są z reguły opisywane za pomocą tablic decyzyjnych. Wtedy, funkcja ρ odwzorowuje $U \times (A \cup D)$ w zbiór wszystkich wartości atrybutów. Przykładowy system decyzyjny dany jest w tab. 2.30. Można zauważyć, że tablica decyzyjna klasyfikuje obiekty $\{u_1, ..., u_8\}$ do czterech różnych klas, tj. $d \in \{1, 2, 3, 4\}$.

Tablica 2.30

A	a_1	a_2	a_3	a_4	a_5	a_6	*d*
1	0	0	1	1	0	0	1
2	1	*	2	0	1	1	2
3	0	1	1	0	0	1	2
4	1	2	2	*	2	0	2
5	*	2	2	2	0	1	1
6	0	0	1	1	0	1	3
7	0	1	0	3	2	?	4
8	2	2	2	3	2	0	4

W obydwu przypadkach, tj. kiedy tablica opisująca system informacyjny oraz system decyzyjny ma w pełni określoną funkcję ρ, system nazywamy *w pełni określonym.* Jednakże w praktyce, dane wejściowe algorytmów eksploracji danych są często zaburzone przez brakujące wartości atrybutów [2.18]. Wtedy odpowiadająca funkcja ρ jest nie w pełni zdefiniowana, a systemy nazywamy *nie w pełni określonymi.* W [2.8] definiuje się cztery przypadki wartości atrybutów dla nie w pełni określonych tablic decyzyjnych, tj. brakujące wartości oznacza przez „?", wartości „do not care" (bez znaczenia) przez „*", zastrzeżone wartości „do not care" przez „+", koncepcyjne wartości atrybutów przez „–". Dodatkowo zakłada się, że dla każdego obiektu przynajmniej jedna wartość atrybutu jest określona [2.24], [2.25]. W naszych rozważaniach dla uproszczenia przyjmujemy, że będziemy uwzględniali tablice tylko z ewentualnymi wartościami „do not care" dla niektórych atrybutów.

1.7.3 Relacja zgodności i relacja nierozróżnialności

W podrozdziale przedstawiamy notację relacji nierozróżnialności, która jest podstawowym pojęciem w dziedzinie eksploracji danych. Podobne pojęcie nazywane relacją zgodności (tolerancji) jest stosowane w dziedzinie układów logicznych. Jest używane głównie do celów dekompozycji układów kombinacyjnych [2.17], oraz logiki sekwencyjnej [2.22]. Na bazie tej relacji skonstruowano pojęcie *r*-podziału, które znajduje zastosowanie w syntezie logiki wielowartościowej, a tym samym może być zastosowane do reprezentacji zarówno systemów informacyjnych jak i systemów decyzyjnych.

Niech $A = (U, A \cup D)$, będzie systemem decyzyjnym, wtedy z każdym podzbiorem $B \subseteq A$ kojarzymy relację równoważności $\mathrm{IND}_A(B)$:

$$\mathrm{IND}_A(B) = \{(u_p, u_q) \in U^2 : \forall\, a_i \in B,\ \rho(u_p, a_i) = \rho(u_q, a_i)\},$$

$\mathrm{IND}_A(B)$ nazywamy *B*-relacją nierozróżnialności [2.24].

Jednakże, dla systemu decyzyjnego przedstawionego w tablicy 2.30, symbol „*" dla obiektu może wyrażać wartość 0 lub 1. W rezultacie, obiekt przedstawia wiele wierszy tablicy. Stąd, klasyfikacja z użyciem relacji nierozróżnialności nie znajduje zastosowania. Aby rozwiązać ten problem wprowadzamy relację zgodności.

Wartości atrybutów a_i, tj. $\rho_{pi} = \rho(u_p, a_i)$ oraz $\rho_{qi} = \rho(u_q, a_i)$ nazywamy *zgodnymi* ($\rho_{pi} \sim \rho_{qi}$) wtedy i tylko wtedy, gdy $\rho_{pi} = \rho_{qi}$ lub $\rho_{pi} = *$ lub $\rho_{qi} = *$, gdzie „*" oznacza przypadek, dla którego wartość

atrybutu jest „do not care”. Z drugiej strony, jeżeli ρ_{pi} oraz ρ_{qi} są zdefiniowane i są różne mówimy, że ρ_{pi} jest nie zgodne z ρ_{qi} i tę relację oznaczamy przez $\rho_p \not\sim \rho_{qi}$.

Na podstawie tej definicji mówimy o *relacji zgodności* $\mathrm{COM}_A(B)$ określonej dla każdego zbioru $B \subseteq A$:

$$\mathrm{COM}_A(B) = \{(u_p, u_q) \in U^2 : \forall\, a_i \in B,\ \rho(u_p, a_i) \sim \rho(u_q, a_i)\},$$

gdzie o obiektach *p* oraz *q*, które spełniają relację $\mathrm{COM}_A(B)$ mówimy, że są zgodne w zbiorze *B.* Obiekty zgodne w zbiorze *B = A* nazywamy po prostu zgodnymi.

Relacja zgodności, nazywana również relacją tolerancji, jest zwrotna i symetryczna, a stąd generuje klasy zgodności na zbiorze obiektów U. Relacja zgodności pozwala na klasyfikowanie obiektów, ale klasy obiektów nie tworzą podziałów na zbiorze U, jak to jest w przypadku relacji nierozróżnialności. $\mathrm{COM}_A(B)$ klasyfikuje obiekty grupując je w klasy zgodności, tj. $U/\mathrm{COM}_A(B)$ gdzie $B \subseteq A$.

Stąd, dowolny zbiór obiektów U reprezentujący system informacyjny lub system decyzyjny może zostać przypisany wielokrotnie i do różnych klas dla danego podzbioru atrybutów *A*. Taką rodzinę klas nazywamy *r*-podziałem i oznaczamy przez Π(B), gdzie B jest wybranym podzbiorem zbioru $A = \{a_1, \ldots, a_m\}$.

r-podział na zbiorze U może być postrzegany jako zbiór nierozłącznych podzbiorów U, których suma jest równa U. Wszystkie symbole i operacje dla algebry podziałów [2.22], mogą zostać bezpośrednio zastosowane do algebry r-podziałów. W szczególności relacja mniejszy lub równy dla dwóch *r*-podziałów Π_1 oraz Π_2 ($\Pi_1 \leq \Pi_2$) zachodzi wtedy i tylko wtedy, gdy dla każdej klasy z *r*-podziału Π_1, oznaczanej w skrócie przez $K_i(\Pi_1)$ istnieje $K_j(\Pi_2)$ taka, że $K_i(\Pi_1) \subseteq K_j(\Pi_2)$.

r-podział na jednoelementowym zbiorze $B = \{a_i\}$ oznaczamy przez $\Pi(a_i)$ lub po prostu $\boldsymbol{\Pi_{a_i}}$.

r-podział indukowany przez zbiór B jest iloczynem *r*-podziałów indukowanych przez pojedyncze atrybuty $a_i \in B$.

1.8 Redukcja atrybutów

1.8.1 Wprowadzenie

Redukcja atrybutów (argumentów) jest algorytmem stosowanym w dwóch odrębnych dziedzinach wiedzy, a mianowicie w zagadnieniach związanych z klasyfikacją danych – maszynowe uczenie, eksploracja danych itd. [2.23], [2.24] oraz w zagadnieniach związanych z optymalizacją i syntezą logiczną układów cyfrowych (por. rozdz. 2.6). W obu przypadkach jest to problem polegający na redukowaniu nadmiarowych specyfikacji w tablicach danych (tablicach prawdy układów logicznych) za pośrednictwem usuwania zbędnych atrybutów (argumentów). Uzyskiwane w wyniku takiego procesu tablice są wykorzystywane do generowania uogólnionych reguł decyzyjnych albo do kolejnych etapów optymalizacji logicznej (w przypadku układów cyfrowych). Oczywiście dane wejściowe algorytmów redukcji są w obu przypadkach zasadniczo różne. Dla

tablic danych mamy do czynienia z wielowartościowymi atrybutami warunkowymi i wielowartościowymi atrybutami decyzyjnymi. Dla tablic prawdy wartości argumentów i wartości funkcji są binarne. Jednocześnie całkowicie inaczej są interpretowane tzw. wartości nieokreślone. W tablicy danych wartość nieokreślona atrybutu warunkowego oznacza, że wartość tego atrybutu nie została ustalona [2.18], a w przypadku tablic funkcji logicznych nieokreśloność argumentu wektora wejściowego oznacza występowanie w specyfikacji wektorów o wszystkich możliwych wartościach danego argumentu.

W rezultacie stosowane w praktyce algorytmy redukcji argumentów oraz atrybutów znacznie się różnią, a jak pokazują przeprowadzone eksperymenty – stosowane w syntezie logicznej metody i algorytmy redukcji argumentów okazują się skuteczniejsze niż analogiczne algorytmy redukcji atrybutów stosowane w analizie danych.

W układach logicznych i ich realizacjach sprzętowych istotnym zagadnieniem jest obliczenie minimalnych zbiorów argumentów o najmniejszej liczności. W przypadku redukcji atrybutów ważniejsze jest obliczenie wszystkich minimalnych zbiorów atrybutów. Taka potrzeba może się zdarzyć, gdy wyliczony zbiór atrybutów mniejszy pod względem liczności niż inny może być trudny do realizacji lub bardziej kosztowny. Na przykład, gdy rozważamy obliczenia redukcji atrybutów dla potrzeb diagnozy medycznej dany parametr może wyrażać skomplikowane lub kosztowne badanie, lub wyrażać badanie, które ma negatywny wpływ na zdrowie pacjenta, lub badanie, które nie jest możliwe do przeprowadzenia. Natomiast inne rozwiązanie – o większej liczności – może być łatwiejsze do realizacji w praktyce. Dlatego opracowanie odpowiednich do zastosowań, skutecznych i szybkich algorytmów redukcji atrybutów jest szczególnie istotne w systemach eksploracji danych.

1.8.2 Algorytm redukcji atrybutów

Systemy informacyjne i systemy decyzyjne są redukowalne co najmniej na dwa sposoby, tj. te same lub nierozróżnialne obiekty mogą być reprezentowane przez tablice wielokrotnie lub – niektóre atrybuty mogą być nadmiarowe [2.19]. Omówione w poprzednim podrozdziale relacje nierozróżnialności oraz zgodności mogą zostać wykorzystane do realizacji zadania redukcji. Nie należy ich traktować jako konkurencyjne techniki, lecz wręcz przeciwnie – jak pokazano w tym rozdziale – techniki, które wzajemnie się uzupełniają.

Niech $A = (U, A \cup D)$, gdzie $A \cap D = \varnothing$, będzie systemem decyzyjnym, gdzie zbiory A, D są odpowiednio zbiorami atrybutów warunkowych oraz decyzyjnych.

Każdy system informacyjny A generuje r-podziały $\Pi(A)$, $\Pi(D)$, gdzie atrybuty a_i generują poszczególne r-podziały Π_{a_i}. r-podziały reprezentują klasy zgodności COM na zbiorze U. Niech $B \subseteq A$ będzie podzbiorem atrybutów oraz u_p, $u_q \in U$. Para $(u_p, u_q) \in \mathrm{COM}(B)$ wtedy i tylko wtedy gdy $\rho(u_p, a_i) \sim \rho(u_q, a_i)$ dla każdego $a_i \in B$. Jeśli zapiszemy klasy zgodności COM(B) jako $\Pi(B)$,wtedy:

$$\Pi(B) = \bigcap_{i:a_i \epsilon B} \Pi(a_i)$$

Używając *r*-podziałów generowanych zbiorem atrybutów możemy wprowadzić pojęcie zależności funkcjonalnej pomiędzy rozłącznymi podzbiorami *A* oraz *D*. Mówimy, że *D* jest funkcjonalnie zależny od *A* (symbolicznie $A \Rightarrow D$), wtedy i tylko wtedy gdy $\Pi(A)$ jest nie większy niż $\Pi(D)$, (tzn. $\Pi(A) \leq \Pi(D)$). W terminach systemów informacyjnych znaczy to, że zbiór atrybutów *D* zależy od zbioru atrybutów *A*. Inaczej mówiąc, jeśli tylko para obiektów nie może być rozróżniona atrybutami ze zbioru *A*, to również nie można tych obiektów rozróżnić atrybutami ze zbioru *D*.

Uproszczenie systemu decyzyjnego z punktu widzenia minimalnego zbioru atrybutów zachowujących zdolności klasyfikacyjne systemu (lub zależność funkcjonalną $A \Rightarrow D$) należy do zadań określanych mianem redukcji wiedzy. Redukcja wiedzy w systemach decyzyjnych polega na redukcji liczby atrybutów warunkowych, czyli wyznaczaniu tak zwanych reduktów oraz usuwaniu nadmiarowych obiektów/reguł.

Zbiór $B \subseteq A$ jest reduktem systemu decyzyjnego $A = (U, A \cup D)$ wtedy i tylko wtedy, gdy $\Pi(B) \leq \Pi(D)$ oraz nie istnieje podzbiór właściwy *B'* zbioru *B* taki, że $\Pi(B') \leq \Pi(D)$. Inaczej mówiąc, reduktem systemu decyzyjnego $\boldsymbol{A} = (U, A \cup D)$, w którym $A \Rightarrow D$, jest zbiór $B \subseteq A$ taki, że $B \Rightarrow D$ oraz nie istnieje podzbiór właściwy *B'* zbioru *B*, gdzie $B' \Rightarrow D$. Atrybut $a \in A$ nazywamy atrybutem zbędnym w systemie decyzyjnym *A* wtedy i tylko wtedy, gdy $\Pi(A \setminus \{a\}) \leq \Pi(D)$, w przeciwnym przypadku *a* jest atrybutem niezbędnym. Zbiór wszystkich atrybutów niezbędnych nazywamy rdzeniem systemu decyzyjnego lub po prostu rdzeniem.

Tablica 2.31

A	a_1	a_2	a_3	a_4	a_5	a_6	*D*
1	0	1	0	1	0	0	1
2	1	0	0	0	1	3	2
3	1	1	0	2	2	3	3
4	1	1	0	2	3	3	2
5	1	1	1	0	2	3	4
6	0	0	2	0	2	3	1
7	1	1	2	0	2	2	5
8	1	1	2	0	2	3	6
9	1	0	2	2	1	3	6
10	1	1	2	2	3	1	7

Obliczymy niezbędność atrybutów dla tablicy decyzyjnej z tabl. 2.31. Skoro $\Pi(A \setminus \{a_1\}) \leq \Pi(D)$, a_1 jest zbędny dla zależności funkcjonalnej $A \Rightarrow D$. W przeciwieństwie, a_3 jest atrybutem niezbędnym, gdyż $\Pi(A \setminus \{a_3\}) \not\leq \Pi(D)$ i wtedy po usunięciu atrybutu a_3 fakt ten manifestuje się sprzecznością wierszy w tablicy. Łatwo również wykazać, że zbiory $B_1 = \{a_1,a_3,a_5,a_6\}$ oraz $B_2 = \{a_2,a_3,a_5,a_6\}$, są reduktami, gdyż $\Pi(a_1,a_3,a_5,a_6) \leq \Pi(D)$, $\Pi(a_2,a_3,a_5,a_6) \leq \Pi(D)$, a po usunięciu dowolnego a_i ze zbioru B_1 lub B_2, odpowiednia nierówność nie będzie spełniona. Uproszczona tablica systemu dla B_1 jest podana w tabl. 2.32.

Tablica 2.32

A'	a_1	a_3	a_5	a_6	*D*
1	0	0	0	0	1
2	1	0	1	3	2
3	1	0	2	3	3
4	1	0	3	3	2
5	1	1	2	3	4
6	0	2	2	3	1
7	1	2	2	2	5
8	1	2	2	3	6
9	1	2	1	3	6
10	1	2	3	1	7

Pojęciami podstawowymi w algorytmie redukcji atrybutów są: macierz porównań oraz funkcja rozróżnialności. Można wtedy wykazać, że między reduktami systemu decyzyjnego $\boldsymbol{A}$, a implikantami funkcji rozróżnialności $f_{\boldsymbol{A}}$, która w rzeczywistości jest monotoniczną funkcją boolowską, zachodzi następujący związek:

$\{a_{i_1}, \ldots, a_{i_p}\} \in \mathrm{RED}_A$, wtedy i tylko wtedy, gdy $a_{i_1} \cdot \ldots \cdot a_{i_p}$ jest implikantem prostym f_A.

Natomiast, wyznaczenie implikantów prostych odbywa się – na przykład – za pomocą przekształcenia monotonicznej funkcji rozróżnialności w postaci iloczynu sum boolowskich do postaci sumy iloczynów. Zgodnie z podrozdziałem 2.6.2, można do tych obliczeń zastosować efektywny algorytm uzupełnienia funkcji boolowskiej.

Funkcja rozróżnialności dla systemu decyzyjnego z tabl. 2.31 jest następująca:

$f_A = a_3a_5a_6(a_1 + a_2)$.

PRZYKŁAD 2.20.c.d.

Wtedy macierz porównań powstaje w wyniku rozróżnienia par obiektów wskazanych w *r*-podziale ilorazowym (tabl. 2.34).

Tablica 2.34

p, *q*	atrybuty rozróżniające
1,7	$\{a_2,a_3,a_4,a_5\}$
3,5	$\{a_2,a_3,a_4\}$
3,6	$\{a_2,a_4\}$
3,7	$\{a_3,a_4,a_5\}$
5,6	$\{a_2,a_3,a_4\}$
5,7	$\{a_2,a_3,a_4,a_5\}$
6,7	$\{a_2,a_3,a_4,a_5\}$
2,5	$\{a_4,a_5\}$

Zapisując macierz porównań w postaci funkcji rozróżnialności otrzymujemy: $f_A = (a_2+a_4)(a_4+a_5)$, ponieważ wiele czynników tego wyrażenia upraszcza się ze względu na własność pochłaniania. Ostatecznie po przekształceniu iloczynu sum do sumy iloczynów otrzymujemy: $f_A = a_4 + a_2a_5$.

Stąd, łącząc poszczególne składniki f_A z rdzeniem $R = \{a_1, a_6\}$ otrzymujemy wszystkie redukty systemu decyzyjnego z tabl. 5.4: $\{a_1,a_4,a_6\}$, $\{a_1,a_2,a_5,a_6\}$.

$$\Pi(R) = \Pi(a_1) \cdot \Pi(a_6) = \{\overline{1,7};\ \overline{3,5,6,7};\ \overline{4};\ \overline{2,5};\ \overline{8};\ \overline{5}\}$$

oraz $\Pi(d) = \{\overline{1,5};\ \overline{2,3,4};\ \overline{6};\ \overline{7,8}\}$, następujący *r*-podział ilorazowy:

$$\Pi(R) \mid \Pi(d) = \{\overline{(1)(7)};\ \overline{(3)(5)(6)(7)};\ \overline{(4)};\ \overline{(2)(5)};\ \overline{(8)};\ \overline{(5)}\}$$

PRZYKŁAD 2.21

Zgodnie z metodą omawianą w podrozdz. 2.6.1 w celu obliczenia reduktów należy obliczyć zmienne niezbędne, a następnie obliczyć podział ilorazowy $P_N|P_D$, gdzie P_N reprezentuje iloczyn podziałów indukowanych zmiennymi niezbędnymi. Podział $P_N|P_D$ stanowi punkt wyjścia do obliczenia tablicy porównań.

Zgodnie z metodą omawianą w podrozdz. 2.6.1 w celu obliczenia reduktów należy obliczyć zmienne niezbędne, a następnie obliczyć podział ilorazowy $P_N|P_D$, gdzie P_N reprezentuje iloczyn podziałów indukowanych zmiennymi niezbędnymi. Podział $P_N|P_D$ stanowi punkt wyjścia do obliczenia tablicy porównań.

Dla funkcji z tabl. 2.35 zmiennymi niezbędnymi są a_4 i a_6.

Tablica 2.35

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	d
1	1	0	0	0	1	0	1	0
2	1	0	1	1	1	1	0	0
3	1	1	0	1	1	1	0	0
4	1	1	1	0	1	1	1	0
5	0	1	0	0	1	0	1	1
6	1	0	0	0	1	1	0	1
7	1	0	1	0	0	0	0	1
8	1	0	1	0	1	1	0	1
9	1	1	1	0	1	0	1	1

PRZYKŁAD 2.21 c.d.

Odpowiednie obliczenia prowadzące do uzyskania podziału ilorazowego są następujące:

$P_4 = \{\overline{1,4,5,6,7,8,9}; \overline{2,3}\}$

$P_6 = \{\overline{1,5,7,9}; \overline{2,3,4,6,8}\}$

$P_4 \cdot P_6 = \{\overline{1,5,7,9}; \overline{4,6,8}; \overline{2,3}\}$

$P_D = \{\overline{1,2,3,4}; \overline{5,6,7,8}\}$

$P_4 \cdot P_6 | P_D = \{\overline{(1),(5,6,7,9)}; \overline{(4),(6,8)}; \overline{(2,3)}\}$

Na tej podstawie tworzymy tablicę porównań (tabl. 2.36), której ostateczna postać nie zawiera nadmiarowego wiersza oznaczonego 4,6.

Tablica 2.36

p, *q*	Zmienne rozróżniające
1,5	$\{a_1,a_2\}$
1,7	$\{a_3,a_5,a_7\}$
1,9	$\{a_2,a_3\}$
4,6	$\{a_2,a_3,a_7\}$
4,8	$\{a_2,a_7\}$

Zbiory atrybutów w poszczególnych wierszach tablicy porównań reprezentują czynniki wyrażenia boolowskiego typu iloczyn sum, które następnie jest przekształcane do postaci typu suma iloczynów. Po uwzględnieniu atrybutów niezbędnych z wyrażenia typu suma iloczynów uzyskujemy wszystkie redukty funkcji z tabl. 2.35:

$\{a_2,a_3,a_4,a_6\}$

$\{a_2,a_4,a_5,a_6\}$

$\{a_2,a_4,a_6,a_7\}$

$\{a_1,a_3,a_4,a_6,a_7\}$

Nietrudno zauważyć, że redukcja atrybutów jest takim samym zadaniem z jakim mieliśmy do czynienia w zagadnieniach redukcji argumentów. Można zatem skorzystać z doświadczeń i metod syntezy logicznej wypracowanych w technice cyfrowej. Godnym polecenia wydaje się algorytm uzupełniania funkcji boolowskich zastosowany do obliczania transformacji CNF na DNF. Jest to istotne o tyle, że transformacja ta jest wykorzystywana w modelowym systemie eksploracji danych jakim jest RSES [5.25]. Równoważność tych dwóch procesów najłatwiej wyjaśnić na przykładzie tablicy funkcji boolowskiej (tabl. 2.35) omawianej w rozdz. 2.6. Interpretując czynniki wyrażenia CNF tej funkcji

$$(a_1 + a_2)(a_3 + a_5 + a_7)(a_2 + a_3)(a_2 + a_7)$$

jako kostki funkcji boolowskiej, nietrudno zauważyć, że uzupełnienie tej funkcji reprezentowane jest kostkami: a_2a_3; a_2a_5; a_2a_7; $a_1a_3a_7$.

Taki sam wynik uzyskuje się dokonując transformacji CNF na DNF (por. rozdz.2.6).

Z dokonanego przeglądu systemów klasyfikacji danych wynika, że stosowane w nich algorytmy redukcji atrybutów są mało skuteczne. Potwierdzeniem tego przypuszczenia są eksperymenty przeprowadzone ze znanym i powszechnie stosowanym do tych obliczeń systemem RSES [2.28]. Wykazano, że system ten nie radzi sobie z tablicami danych o dużej liczbie nieokreśloności. Wymownym przykładem jest często cytowana logistyczna baza danych *trains* (tabl. 2.37).

Tablica 2.37

```
.type fr
.i 32
.o 1
.p 10
23016320081311611006100100010010 0
12009130071200020-----0101000000 0
11006100041311013-----0000101000 0
21007130011300212006121100100000 0
12001131101200010-----0101000000 0
010103000613----------0000001000 1
11001100091301 50------0000001000 1
011101200910----------0001000000 1
21007100151200612007101001000000 1
000091201622----------1000000000 1
.end
```

Dla bazy *trains* program opracowany z zastosowaniem metody uzupełnienia funkcji boolowskiej generuje 689 reduktów, natomiast RSES zaledwie 333 (z pominięciem opcji *Don't discern with missing values*). Natomiast uruchomienie RSES dla opcji *Don't discern with missing values* kończy się (po kilku godzinach obliczeń) komunikatem *Not enough memory*. Istotnym jest, że pominięcie opcji *Don't discern with missing values* skutkuje uwzględnianiem brakujących wartości podczas tworzenia macierzy porównań. Inaczej mówiąc, w czasie wyznaczania zbioru atrybutów na których różnią się dwa wybrane obiekty, odróżniane są te o wartości NULL. W rezultacie takiego postępowania otrzymujemy wynik, który jest różny od zbioru wszystkich minimalnych zbiorów atrybutów.

Innym przykładem potwierdzającym bezwzględną przewagę metody uzupełnienia funkcji boolowskiej w metodzie redukcji atrybutów jest funkcja KAZ (tabl. 2.38). Jest to funkcja 21 argumentów binarnych, często stosowana w testowaniu zaawansowanych narzędzi syntezy logicznej. Otóż program RSES oblicza wszystkie 5574 redukty tej funkcji w ciągu 39 minut, natomiast opracowana i zaimplementowana procedura z zastosowaniem algorytmu uzupełnienia tworzy zbiór wszystkich 5574 reduktów w czasie 2.5 s.

Tablica 2.38

```
type fr
.i 21
.o 1
.p 31
100110010110011111101 1
111011111011110111100 1
001010101000111100000 1
001001101100110110001 1
100110010011011001101 1
100101100100110110011 1
001100100111010011011 1
001101100011011011001 1
110110010011001001101 1
100110110011010010011 1
110011011011010001100 1
010001010000001100111 0
100110101011111110100 0
111001111011110011000 0
101101011100010111100 0
110110000001010100000 0
110110110111100010111 0
110001000111100100001 0
001001000101111101101 0
100100011111100110110 0
100011000110011011110 0
110101000110101100001 0
110110001101101100111 0
010000111001000000001 0
001001100101111110000 0
100100111111001110010 0
000010001110001101101 0
101000010100001110000 0
101000110101010011111 0
101010000001100011001 0
011100111110111101111 0
.end
```

1.9 Indukcja reguł decyzyjnych

Celem indukcji reguł decyzyjnych jest wygenerowanie z danych zbioru reguł, które będą użyte do klasyfikowania nowych obiektów. Należy podkreślić, że zarówno rozwój algorytmów indukcji reguł, jak i sposób ich oceny ukierunkowany jest przede wszystkim na perspektywę klasyfikowania nowych obiektów. Ponieważ zbiór reguł traktowany jest wtedy jako klasyfikator, poprawność klasyfikowania jak największej liczby obiektów jest główną miarą oceny.

Ze względu na znaczenie algorytmów redukcji reguł w klasyfikacji danych rozważa się różne strategie obliczeniowe. W strategiach tych wyróżnikiem jest stopień uogólniania reguł, co wpływa na precyzję klasyfikacji obiektów spoza zbioru treningowego.

1.9.1 Strategia dwustopniowej selekcji reguł

Naturalną strategią indukcji reguł jest strategia dwustopniowej selekcji reguł. W strategii tej dla każdego obiektu u_i klasy K tworzy się zbiór wszystkich minimalnych reguł pozytywnych (ozn. R(u_i)). Suma zbiorów R(u_i) dla wszystkich obiektów klasy K tworzy rodzinę minimalnych reguł tej klasy. Rodzina ta jest następnie poddawana procesowi selekcji, którego celem jest wyznaczenie minimalnej rodziny minimalnych reguł. Zarówno w procesie uogólniania pojedynczego obiektu, jak też w procesie selekcji korzysta się z pojęcia pokrycia kolumnowego binarnej macierzy M, omawianym już w rozdz. 1.2.

Obliczanie pokrycia kolumnowego jest standardowym zadaniem polegającym na transformacji wyrażenia boolowskiego typu CNF (*Conjunctive Normal Form*) na wyrażenie DNF (*Disjunctive Normal Form*). W wyrażeniu CNF czynniki koniunkcji są dysjunkcjami zmiennych boolowskich etykietujących te kolumny M dla których w danym wierszu $m_{ij} = 1$. Liczba czynników jest równa liczbie wierszy macierzy M. Istotnym problemem jest transformacja CNF na DNF, gdyż składniki wyrażenia DNF są koniunkcjami zmiennych reprezentujących kolumny macierzy M.

Zastosowanie pokrycia kolumnowego do uogólnienia reguły decyzyjnej obiektu u_i dotyczy tzw. macierzy rozróżnialności.

Macierz rozróżnialności tworzymy przez porównanie obiektu u_i z każdym obiektem u_j należącym do innej klasy decyzyjnej. Porównanie polega na utworzeniu binarnego wektora *w*, w którym na pozycja *k* jest wartość zero, jeśli wartość atrybut $A_k(u_i)$ obiektu u_i jest taka sama jak wartość atrybutu $A_k(u_j)$ obiektu u_j lub co najmniej jedna z tych wartości jest nieokreślona. W przeciwnym przypadku wartość składowej k wektora *w* jest równa jeden. Zbiór wektorów *w* tworzy macierz rozróżnialności.

Z definicji macierzy rozróżnialności wynika, że aby uogólniona reguła obiektu u_i nie pokrywała zdanego obiektu innej klasy decyzyjnej, to w odpowiedniej R(u_i) należy zostawić atrybuty, które odróżniają R(u_i) od każdego obiektu innej klasy decyzyjnej.

Istotę zjawiska możemy zaobserwować na hipotetycznym przykładzie reprezentującym wyniki sondażu przed wyborami prezydenckimi w pewnej republice.

PRZYKŁAD 2.22

Sondaż przed wyborami prezydenckimi w pewnej republice przeprowadzono wg reguł decyzyjnych uzyskanych z danych (tabl. 2.39) reprezentujących odpowiedzi na pytania (tak, nie) uzyskane od 9 respondentów. W tablicy tej odpowiedziom *tak* przyporządkowano wartość 1, odp. *nie* – 0, przy jednoczesnym wyróżnieniu zwolenników ocenianego kandydata na prezydenta atrybutem TAK, a przeciwników, atrybutem NIE. Celem jest obliczenie uogólnionych reguł decyzyjnych określających zwolenników tego kandydata dla respondentów o dowolnych odpowiedziach na pytania sondażowe.

Tablica 2.39

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	d
1	1	0	0	0	1	1	0	1
2	0	1	0	0	1	0	1	1
3	1	0	1	0	0	0	0	1
4	1	0	1	0	1	1	0	1
5	1	1	1	0	1	0	1	1
6	1	0	0	0	1	0	1	0
7	1	0	1	1	1	1	0	0
8	1	1	0	1	1	1	0	0
9	1	1	1	0	1	1	1	0

Na przykład dla obiektu u_1 systemu decyzyjnego z tablicy 2.39 odpowiednia macierz rozróżnialności będzie:

$$\boldsymbol{M} = \begin{matrix} \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \begin{matrix} 1\,2\,3\,4\,5\,6\,7 \\ \end{matrix} \begin{bmatrix} 0\,0\,0\,0\,0\,1\,1 \\ 0\,0\,1\,1\,0\,0\,0 \\ 0\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,0\,0\,0\,1 \end{bmatrix},$$

czyli dla zapewnienia „pełnego rozróżnienia” w regule R(u_1) należy zostawić atrybuty: a_6 lub a_7 i a_3 lub a_4 i a_2 lub a_4 i a_2 lub a_3 lub a_6. Zapisując te warunki w postaci wyrażenia CNF:

$$(a_6 + a_7)\,(a_3 + a_4)\,(a_2 + a_4)\,(a_2 + a_3 + a_7)$$

dochodzimy do wniosku, że w celu obliczenia wszystkich uogólnionych reguł obiektu u_2 (o minimalnej liczbie atrybutów) należy dokonać transformacji wyrażenia CNF do postaci DNF. Wtedy składniki DNF reprezentują minimalne zbiory atrybutów R(u_2). Wynik takiej transformacji jest następujący:

$$a_4a_7 + a_2a_4a_6 + a_3a_4a_6 + a_2a_3a_7 + a_2a_3a_6$$

Oczywiście najlepsza reguła będzie dla a_4a_7, czyli $R(u_2) = (a_4,0)$ & $(a_7,0)$ i jak łatwo sprawdzić żaden obiekt klasy $d = 0$ nie spełnia tej reguły.

Podstawowe znaczenie w metodzie ma procedura obliczająca pokrycie kolumnowe macierzy *M*. Ze względu na złożoność obliczeniową, zadanie generowania reguł nie jest zwykle możliwe do rozwiązania w czasie wielomianowym. W szczególności zbiór pokryć macierzy *M* może zawierać zbyt wiele elementów, aby program mógł znaleźć rozwiązanie w rozsądnym czasie. W związku z tym zaproponowano zastosowanie algorytmu uzupełnienia funkcji boolowskiej.

Możliwość zastosowania uzupełnienia funkcji boolowskiej do obliczenia wszystkich minimalnych pokryć kolumnowych macierzy binarnej M omówiona była w rozdz. 2.6.2. Na tej podstawie stwierdzamy: zamiast stosować transformację CNF na DNF wystarczy obliczyć uzupełnienie funkcji boolowskiej reprezentowanej macierzą M. Metoda jest dokładnie przedstawiona w rozdz. 2.6.2, dlatego ograniczymy jej omówienie jedynie do obliczenia uzupełnienia macierzy M. Funkcja boolowska macierzy M oraz jej uzupełnienie podane są w tablicy 2.40.

Tablica 2.40

		x_1	x_2	x_3	x_4	x_5	x_6	x_7	f
1		–	–	–	–	–	1	1	1
2		–	–	1	1	–	–	–	1
3		–	1	–	1	–	–	–	1
4		–	1	1	–	–	–	1	1
5		–	0	–	0	–	0	–	0
6		–	0	0	–	–	0	–	0
7		–	–	0	0	–	0	–	0
8		–	0	0	–	–	–	0	0
9		–	–	–	0	–	–	0	0

Procedura uzupełniania – jak wykazano w [2.17],[2.18] – jest bardzo szybka, zatem jej zastosowanie do indukcji reguł jest wskazane. Dysponując szybkim algorytmem indukcji reguł dla pojedynczych obiektów możemy pokusić się o indukcję dla wszystkich reguł danej klasy decyzyjnej i wybrać z nich reguły najogólniejsze, przeznaczone do następnego etapu selekcji. Przy takiej organizacji trzeba będzie jednak wprowadzić heurystyczne algorytmy selekcji.

PRZYKŁAD 2.22. c.d.

Dla binarnego systemu decyzyjnego podanego w tabl. 2.39 obliczamy minimalne reguły decyzyjne dla obiektów u_1 do u_5. Oznaczając przez R_i reguły generowane przez obiekt u_i uzyskujemy kolejno:

$r_1 = (a_4,0)\ \&\ (a_7,0)$ $\quad r_2 = (a_1,0)$ $\quad r_3 = (a_5,0)$

$r_4 = (a_4,1)\ \&\ (a_7,0)$ $\quad r_5 = (a_2,1)\ \&\ (a_6,0)$ oraz $(a_3,1)\ \&\ (a_6,0)$

Usuwając reguły powtarzające się ostateczną listę reguł minimalnych zapisujemy jak następuje:

$R_1 = (a_1,0)$ $\quad R_2 = (a_4,0)\ \&\ (a_7,0)$ $\quad R_3 = (a_5,0)$

$R_4 = (a_2,1)\ \&\ (a_6,0)$ $\quad R_5 = (a_3,1)\ \&\ (a_6,0)$

Na tej podstawie dla każdej obliczonej wyżej reguły R_i wyznaczamy wszystkie obiekty decyzji $d = 1$ pokrywane przez R_i.

PRZYKŁAD 2.22 c.d.

Przykładowo:

$R_1 \supseteq u_1$

$R_2 \supseteq u_2, u_3, u_4$

$R_4 \supseteq u_1, u_5$

Tabelę pokryć pokazano w tabl. 2.41.

Tablica 2.41

	R_1	R_2	R_3	R_4	R_5
u_1	1	0	0	1	0
u_2	0	1	0	0	0
u_3	0	1	1	0	1
u_4	0	1	0	0	0
u_5	0	0	0	1	1

Tablica pokryć umożliwia wybór (selekcję) takiego minimalnego zbioru reguł, który pokrywa wszystkie obiekty ustalonej klasy decyzyjnej. Minimalny zbiór reguł klasy decyzyjnej $d = 1$ można wyznaczyć obliczając minimalne pokrycie kolumnowe tablicy pokryć. W tym celu zapisujemy wiersze tablicy w postaci zbiorów kolumn wskazywanych przez pozycje jedynek w danym wierszu. Metoda selekcji pokryć zastosowana w proponowanym algorytmie indukcji oblicza wszystkie minimalne pokrycia kolumnowe metodą uzupełnienia funkcji boolowskiej, której specyfikacja (również jej uzupełnienia) jest podana w tabl. 2.42

Tablica 2.42

	x_1	x_2	x_3	x_4	x_5	f
1	1	–	–	1	–	1
2	–	1	–	–	–	1
3	–	1	1	–	1	1
4	–	–	–	1	1	1
5	0	0	–	–	0	0
9	–	0	–	0	–	0

Z zapisu uzupełnienia wynika, że wszystkie minimalne pokrycia kolumnowe są: R_2, R_4 oraz R_1, R_2, R_5. Oczywiście taki sam wyniki uzyskamy dokonując transformacji wyrażenia typu CNF na DNF:

$$r_2(r_1 + r_4)\,(r_2 + r_3 + r_5)\,(r_4 + r_5) = r_2r_4 + r_1\,r_2r_5$$

Z powyższych rozważań wynika, że zadanie uogólnienia reguł decyzyjnych ustalonej klasy D_k jest analogiczne do zadania minimalizacji funkcji boolowskiej $f = (F, R)$, w której wektory zbioru F odpowiadają obiektom klasy D_k, a zbiór R jest macierzą rozróżniającą. Złożoność obliczeniową tego problemu można oszacować złożonością obliczeniową zadania minimalizacji funkcji boolowskiej. Obliczeniem decydującym o eksplozji kombinatorycznej tego problemu jest zatem obliczenie wszystkich pokryć kolumnowych Tablicy

Pokryć. O złożoności tego problemu decyduje szybko rosnąca (ze wzrostem liczby atrybutów) liczność rodziny minimalnych reguł klasy D_k.

Otóż w przypadku systemu decyzyjnego z tablicy 2.39 liczba wszystkich minimalnych reguł jest 5: tym samym odpowiednia tablica pokryć (tabl. 2.41) ma 5 kolumn. W rezultacie obliczenie minimalnych pokryć kolumnowych tej tablicy można wykonać „ręcznie" – jest widoczne „gołym okiem". Zjawisko występującej w tym problemie eksplozji kombinatorycznej dobrze wyjaśnia przykład tablicy decyzyjnej podanej w tablicy 2.43. W tym przypadku liczba wszystkich minimalnych reguł jest 68. Zatem odpowiednia tablica pokryć ma aż 68 kolumn, co znacznie utrudnia obliczenie uzupełnienia.

Tablica 2.43

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	y
1	0	0	1	0	1	1	1	0	1	0	0
2	1	0	1	0	0	1	0	1	0	0	0
3	0	1	0	0	0	1	1	1	1	0	0
4	1	0	1	1	1	0	1	0	1	1	0
5	1	1	0	0	0	1	0	0	1	1	0
6	0	1	0	0	0	1	0	1	1	0	0
7	1	1	1	0	1	0	0	1	1	0	0
8	0	1	0	0	1	1	0	0	0	0	0
9	0	1	0	1	0	0	0	0	1	0	0
10	0	1	1	1	1	1	1	0	1	1	1
11	0	0	0	0	0	1	0	1	0	0	1
12	1	1	0	1	1	1	0	0	1	1	1
13	0	1	0	0	1	0	0	0	0	0	1
14	0	1	0	0	0	1	1	1	1	1	1
15	0	0	1	0	0	0	0	1	1	0	1
16	1	1	1	1	0	1	0	0	0	1	1
17	1	1	1	1	1	0	1	0	0	1	1
18	1	1	1	1	1	1	1	1	1	1	1
19	0	0	1	0	0	0	0	0	0	0	1
20	1	1	0	1	1	0	0	1	1	1	1
21	0	0	1	0	0	0	1	1	1	1	1
22	1	1	1	1	1	0	0	0	1	0	1
23	1	0	1	0	1	1	1	1	0	1	1
24	0	1	1	0	0	0	0	1	1	0	1
25	0	1	0	0	1	1	1	0	0	0	1

Z powyższych rozważań wynika, że obliczenie uogólnionych reguł decyzyjnych dla rzeczywistych baz danych muszą być – przynajmniej dla tablicy pokrycia – realizowane algorytmami heurystycznymi. Natomiast rewelacyjna procedura uzupełniania (*Complement*) może być zastosowana wyłącznie do obliczania minimalnych pokryć kolumnowych macierzy rozróżnialności.

Należy pamiętać, że algorytm uzupełnienia funkcji boolowskiej (*complement*) jest algorytmem systematycznym. Uzyskuje najlepsze wyniki pokrycia, a co ważne dla generacji reguł, tworzy wszystkie rozwiązania problemu. Zalet tego algorytmu niestety nie da się zawsze wykorzystać, szczególnie dla bardziej złożonych problemów. Algorytm uzupełniania jest szczególnie wrażliwy na ilość kolumn w tablicy rozróżnialności. Praktyka pokazuje, że tablice rozróżnialności podczas uogólniania reguł zawierają zazwyczaj

więcej obiektów (wierszy), niż kolumn (atrybutów), dlatego dla małych i średnich baz w rozumieniu ilości atrybutów (od 1 do 30 atrybutów) z powodzeniem używamy uzupełnienia. Niestety odwrotną strukturę posiadają tablice rozróżnialności reguł. Ilość kolumn jest zawsze niemniejsza niż ilość wierszy. Wynika to z faktu, że z każdego obiektu (wiersza) indukowana jest minimalnie jedna reguła.

Ograniczenia w stosowaniu algorytmu uzupełniania funkcji boolowskich spowodowały potrzebę użycia szybkich, heurystycznych algorytmów minimalizacji tablic boolowskich. Możliwe do zastosowania są dwa algorytmy iteracyjne: MaxCol i MinRow, omawiane poprzednio w rozdz. 2.4.2.

Algorytmy Complement, MaxCol, MinRow stanowią strategię indukcji reguł decyzyjnych, którą nazywać będziemy strategią dwustopniowej selekcji.

Regułowy system decyzyjny

Procedura dwustopniowej selekcji reguł składa się z następujących etapów.

1. Wyznaczenie macierzy rozróżnialności dla obiektu u_i ustalonej klasy decyzyjnej
2. Obliczenie wszystkich uogólnionych reguł obiektu u_i

   Macierz rozróżnialności jest wykorzystywana do obliczenia wszystkich uogólnionych reguł obiektu u_i. Obliczenie zbiory wszystkich minimalnych reguł sprowadzić można do problemu obliczenia pokryć kolumnowych macierzy rozróżnialności.

   Załóżmy, że kolumny macierzy rozróżnialności są indeksowane kolejnymi atrybutami warunkowymi tablicy decyzyjnej. Wtedy każde minimalne pokrycie kolumnowe tej macierzy jest zbiorem atrybutów warunkowych uogólnionej i minimalnej reguły decyzyjnej obiektu u_i. W celu utworzenia tej reguły atrybutom zbioru reprezentującego pokrycie kolumnowe przyporządkowuje się odpowiednie wartości obiektu u_i, czyli jeśli minimalny zbiór atrybutów jest $\{a_i, a_j,\ldots,a_k\}$ oraz obiekt u_i ma wartości atrybutów odpowiednio $\{w_i, w_j,\ldots,w_k\}$ to ugólnioną regułą jest:

   $$(a_{i=} w_i) \,\&\, (a_{j=} w_j) \,\&\, (a_{k=} w_k) = d_k$$

3. Obliczenie rodziny minimalnych ugólnionych reguł klasy decyzyjnej D_k

   Powtarzając obliczenia z punków 1 i 2 dla każdego obiektu u_i ustalonej klasy decyzyjnej D_k uzyskuje się rodzinę minimalnych uogólnionych reguł klasy D_k:

   $$R(D_k) = (r1,\ldots, r_{max})$$

   Reguły wchodzące w skład tej rodziny stanowią zbiór najlepszych reguł reprezentujących wszystkie obiekty klasy D_k.
4. Wyznaczenie tablicy pokryć klasy D_k

Chcąc uzyskać minimalny zbiór reguł (niekoniecznie o najmniejszej liczności) reprezentujących klasę D_k należy utworzyć tablicę pokryć (TP). Tablicą pokryć jest binarna tablica o liczbie kolumn n (n jest licznością rodziny $R(D_k)$) i liczbie wierszy równej k (k – liczba obiektów klasy klasy D_k). Element TP(i,j) tej tablicy przyjmuje wartość 1, gdy reguła r_j pokrywa obiekt u_i, w przeciwnym przypadku 0.

5. Obliczenie minimalnego zbioru uogólnionych reguł klasy D_k.

   Minimalny zbiór uogólnionych reguł reprezentujących (pokrywających) klasę D_k można wyznaczyć obliczając minimalne pokrycie kolumnowe TP.

Ze względu na złożoność obliczeniową, zadanie generowania reguł nie jest zwykle możliwe do rozwiązania w czasie wielomianowym. W szczególności zbiór pokryć macierzy M_p może zawierać zbyt wiele elementów, aby program mógł znaleźć rozwiązanie w rozsądnym czasie.

1.9.2 Sekwencyjne pokrywanie

Najprostszym algorytmem heurystycznym indukcji reguł jest sekwencyjne pokrywanie (*sequential covering*). W algorytmie tym tworzona jest reguła pokrywająca część obiektów bazowego zbioru treningowego danej klasy decyzyjnej, a obiekty spełniające tę regułę są usuwane. Proces jest powtarzany dla pozostałych obiektów, aż do pokrycia wszystkich obiektów tej klasy. Taką samą procedurę stosuje się dla wszystkich pozostałych klas decyzyjnych.

Tablica 2.44

A	a	b	c	d	e
1	1	0	0	1	1
2	1	0	0	0	1
3	0	0	0	0	0
4	1	1	0	1	0
5	1	1	0	2	2
6	2	2	0	2	2
7	2	2	2	2	2

Dla systemu decyzyjnego A podanego w tabl. 2.44, w którym a, b, c, d są atrybutami warunkowymi, natomiast e jest atrybutem decyzyjnym, obliczymy reguły decyzyjne dla klasy $e = 1$.

Najpierw obliczymy macierz M_1 generowaną obiektem u_1 (pierwszy wiersz tablicy). Porównując u_1 z każdym obiektem z klasy nie należącym do klasy decyzyjnej $e = 1$, otrzymujemy następującą macierz porównań M_1:

$$\begin{matrix} & a & b & c & d \end{matrix}$$
$$\boldsymbol{M_1} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Łatwo wykazać, że minimalne pokrycia kolumnowe dla M_1, to: {a, b} oraz {b, d}, tzn.

$$(a + d)(b)(b + d)(a + b + d)(a + b + c + d) = ab + bd$$

Pokrycia te można obliczyć wykorzystując również algorytm uzupełnienia funkcji boolowskiej przedstawiony w podrozdziale 2.5. Wyznaczone w ten sposób minimalne reguły wynoszą odpowiednio:

$(a,1)$ & $(b,0) \rightarrow (e,1)$

$(b,0)$ & $(d,1) \rightarrow (e,1)$

Podobnie dla obiektu u_2 mamy:

$$\begin{matrix} a & b & c & d \end{matrix}$$
$$\boldsymbol{M_2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

czyli minimalne reguły to:

$(a,1)$ & $(b,0) \rightarrow (e,1)$

$(a,1)$ & $(d,0) \rightarrow (e,1)$

Łatwo zauważyć, że reguła $(a,1)$ & $(b,0) \rightarrow (e,1)$, podobnie jak w metodzie ekspansji funkcji boolowskiej, pokrywa wszystkie reguły pierwotne systemu decyzyjnego A i klasy $e = 1$.

Postępując analogicznie dla pozostałych klas decyzyjnych, tzn. $e = 0$ oraz dla $e = 2$, otrzymujemy następujące reguły uogólnione (jedno z możliwych rozwiązań):

$(a,1)$ & $(b,0) \rightarrow (e,1)$

$(a,0) \rightarrow (e,0)$

$(b,1)$ & $(d,1) \rightarrow (e,0)$

$(d,2) \rightarrow (e,2)$

lub w innym zapisie:

$(a,1)$ & $(b,0) \rightarrow (e,1)$

$(a,0) \vee (b,1)$ & $(d,1) \rightarrow (e,0)$

$(d,2) \rightarrow (e,2)$

Tablica 2.45

a	*b*	*c*	*d*	*e*
1	0	–	–	1
0	–	–	–	0
–	1	–	1	0
–	–	–	2	2

Tablicowy zapis powyższych reguł podano w tablicy 2.45. Można również zauważyć, że w wyniku indukcji reguł uległ redukcji argument *c*, co oznacza, że jest on atrybutem zbędnym.

1.10 Zadania

ZADANIE 2.1

Zminimalizować metodą tablic Karnaugha funkcje boolowskie reprezentujące wyjścia *c*, *e*, *f* dekodera wskaźnika siedmiosegmentowego z tablicy 2.2.

ZADANIE 2.2

Dla funkcji *F* opisanej podziałami P_1 do P_8 oraz P_F zmienne niezbędne są x_6 oraz x_8. Należy wyznaczyć wszystkie realizacje minimalno argumentowe tej funkcji.

$P_1 = (\overline{1,6,11,12}; \overline{2,3,4,5,7,8,9,10})$

$P_2 = (\overline{1,11,12}; \overline{2,3,4,5,6,7,8,9,10})$

$P_3 = (\overline{2,5,7,10}; \overline{1,3,4,6,7,8,9,11,12})$

$P_4 = (\overline{2,4,7,8,9,10}; \overline{1,3,5,6,11,12})$

$P_5 = (\overline{2,3,5,6,7,10,12}; \overline{1,4,8,9,11})$

$P_6 = (\overline{1,3,5,7,8,10,11,12}; \overline{2,4,6,9})$

$P_7 = (\overline{1,2,4,6,7,9,11,12}; \overline{3,5,8,10})$

$P_8 = (\overline{1,4,6,8,10}; \overline{2,3,5,7,9,11,12})$

$P_F = (\overline{1,2,3,6,8,9,11,12}; \overline{4,7,10})$

ZADANIE 2.3

Dla podanej tablicy decyzyjnej (tabl. 2.46) obliczyć wszystkie minimalne zbiory atrybutów. Przyjąć, że atrybutem „niezbędnym” jest *f*

Tablica 2.46

U	*a*	*b*	*d*	*c*	*e*	*f*	*g*
1	2	2	1	2	1	1	3
2	2	2	1	1	1	1	3
3	2	1	1	3	1	2	1
4	3	3	2	4	3	1	2
5	3	3	2	3	1	2	3
6	2	2	1	1	1	2	1
7	2	1	1	4	3	1	2
8	1	3	2	4	3	1	2

ZADANIE 2.4

Dla danych zapisanych w tabl. 2.47 obliczyć zbiór minimalnych reguł decyzyjnych dla decyzji *e*.

Tablica 2.47

U	*a*	*b*	*c*	*d*	*e*
1	2	2	2	2	0
2	2	2	0	2	0
3	1	1	0	2	0
4	1	1	0	1	1
5	0	0	0	0	1
6	1	1	1	1	2
7	0	1	0	0	2
8	0	1	0	1	2

1.11 Bibliografia

[2.1] Brayton R.K., Hachtel G.D., McMullen C.T., Sangiovanni-Vincentelli A.: *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[2.2] De Micheli G.: Synteza *i optymalizacja układów cyfrowych*. WNT, Warszawa 1998.

[2.3] Kamionka-Mikuła H., Małysiak H., Pochopień B.: *Praktyczna teoria układów cyfrowych*, Wydawnictwo Politechniki Śląskiej, Gliwice 2011.

[2.4] *Kamionka-Mikuła H., Małysiak H., Pochopień B.: Synteza i analiza układów cyfrowych*. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, 2011.

[2.5] Lala P. K.: *Practical Digital Logic Desing and Testing*. Prentice-Hall, 1996.

[2.6] Łuba T., Rybnik J.: *Rough Sets and Some Aspects in Logic synthesis*. In: Słowiński R. (ed.): Intelligent Decision Support – Handbook of Application and Advances of the Rough Sets Theory, Kluwer Academic Publishers, 1992.

[2.7] Łuba T., Ojrzeńska-Wójter D.: *Układy logiczne w zadaniach*. Oficyna Wydawnicza Politechniki Warszawskiej, 2011.

[2.8] Łuba T., Borowik G.: *Synteza logic*zna. Oficyna Wydawnicza Politechniki Warszawskiej, 2015.

[2.9] Łuba T.(red.), Rawski M., Tomaszewicz P., Zbierzchowski B.: *Programowalne układy przetwarzania sygnałów i informacji,* WKŁ, Warszawa 2008.

[2.10] Pawlak Z.: *Rough Sets: Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, 1991.

[2.11] Pochopień B.: *Podstawy techniki cyfrowej*. Wydawnictwo Wyższej Szkoły Biznesu w Dąbrowie Górniczej, 2011.

[2.12] Stańczyk U., Cyran K., Pochopień B.: *Theory of logic Circuits. Vol. 1. – Fundamental issues*. Publishers of the Silesian University of Technology, Gliwice 2007.

[2.13] Stańczyk U., Cyran K., Pochopień B.: *Theory of logic circuits – Vol. 2 – Fundamental issues*. Publishers of the Silesian University of Technology, Gliwice 2007.

[2.14] Sasao T.: *Index Generation Functions, Logic Synthesis for Pattern Matching*, EPFL Workshop on Logic Synthesis & Verification, Dec. 2015.

[2.15] Zieliński C.: *Podstawy projektowania układów cyfrowych*. Wydawnictwo Naukowe PWN, Warszawa 2003.

[2.16] Borowik G., Łuba T., *Fast Algorithm of Attribute Reduction Based on the Complementation of Boolean Function*, in Advanced Methods and Applications in Computational Intelligence, s. Topics in Intelligent Engineering and Informatics, Eds. R. Klempous, J. Nikodem, W. Jacak, Z. Chaczko, Ch. 2, pp. 25-41, Springer International Publishing, 2014, DOI: 10.1007/978-3-319-01436-4_2.

[2.17] Brayton R.K., Hachtel G.D., McMullen C.T., Sangiovanni-Vincentelli A.: *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[2.18] Grzymala-Busse J.W., Hu M.: *A Comparison of Several Approaches to Missing Attribute Values in Data Mining*. In: Ziarko W., Yao Y. (eds.): Rough Sets and Current Trends in Computing 2000, LNAI 2005, pp. 378−385, Springer-Verlag, Berlin, 2001.

[2.19] Komorowski J., Pawlak Z., Polkowski L., Skowron A.: *Rough sets: A tutorial* (1999).

[2.20] Kryszkiewicz M., Lasek P.: *FUN: Fast Discovery of Minimal Sets of Attributes Functionally Determining a Decision Attribute*. In: Peters J.F. et al. (eds.): Transactions on Rough Sets IX, LNCS 5390, pp. 76–95, Springer-Verlag, Berlin, 2008.

[2.21] Łuba T. (et al.): *Rola i znaczenie syntezy logicznej w eksploracji danych dla potrzeb telekomunikacji i medycyny*. Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne, Nr. 5, 2014.

[2.22] Łuba T., Borowik G.: *Synteza logiczna*, Oficyna Wydawnicza PW, Warszawa 2015.

[2.23] Pawlak Z.: *Rough Sets: Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, 1991.

[2.24] Skowron A., Rauszer C.: *The Discernibility Matrices and Functions in Information Systems*. In: Słowiński R. (ed.): Intelligent Decision Support – Handbook of Application and Advances of the Rough Sets Theory, Kluwer Academic Publishers, 1992.

[2.25] Stefanowski J.: *Algorytmy indukcji reguł decyzyjnych w odkrywaniu wiedzy*. Rozprawa habilitacyjna, wersja z 8 lutego 2001, Wydawnictwo Politechniki Poznańskiej, Seria Rozprawy nr 361, 2001.

[2.26] 26 ROSE2 – *Rough Sets Data Explorer*, http://idss.cs.put.poznan.pl/site/rose.html

[2.27] ROSETTA – *A Rough Set Toolkit for Analysis of Data*, http://www.lcb.uu.se/tools/rosetta/

[2.28] RSES – *Rough Set Exploration System*, http://logic.mimuw.edu.pl/~rses/

[2.29] *UC Irvine Machine Learning Repository*, http://archive.ics.uci.edu/ml/