2018

Synteza logiczna
pojęcia podstawowe

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 Rola i znaczenie syntezy logicznej

Synteza logiczna jest gałęzią wiedzy, która w ostatnich latach rozwijała się niezwykle intensywnie, a jej zastosowania szybko przekroczyły granice tradycyjnej dziedziny układów cyfrowych, dochodząc do obszarów wiedzy zaliczanej do szeroko rozumianych technik informacyjnych, a nawet informatyki. Przyczyną tej sytuacji jest z jednej strony rozwój technologii mikroelektronicznych, a z drugiej coraz większe zapotrzebowanie na analizę i eksplorację danych.

Rozwój technologii mikroelektronicznych zwiększa możliwości techniki cyfrowej, ale ich pełne wykorzystanie wymaga rozwoju nowych metod syntezy logicznej. Powszechnie stosowane tradycyjne metody syntezy logicznej np. minimalizacja funkcji boolowskich są niedostosowane do zasobów układów FPGA, wyposażonych w komórki LUT oraz pamięci ROM. Próby zaradzenia tej sytuacji podejmowane były od dawna, ale szczególnej intensywności nabrały stosunkowo niedawno. Mocnym głosem okazała się książka T. Sasao „Memory based logic synthesis" [1.17], w której po raz pierwszy tak wyraźnie stwierdzono, że jedynymi skutecznymi metodami syntezy są redukcja argumentów i dekompozycja funkcjonalna.

Wpływ zaawansowanych procedur syntezy logicznej na jakość implementacji sprzętowych układów cyfrowych jest najbardziej znaczący w algorytmach wykorzystujących nowoczesne struktury programowalne. Struktury takie są powszechnie stosowane w układach przetwarzania informacji i sygnałów np. w realizacjach algorytmów kryptograficznych, w filtrach cyfrowych, układach transformacji falkowej oraz w syntezie funkcji generowania indeksów. Zatem stosowanie zaawansowanych procedur syntezy logicznej – dostępnych głównie w oprogramowaniu uniwersyteckim – niejednokrotnie może się przyczynić do sukcesu rynkowego wielu urządzeń cyfrowych, w szczególności tych realizowanych w technologii układów programowalnych przez użytkownika FPLD. Należy jednak podkreślić, że w przypadku generatorów adresu stosowanie procedur syntezy logicznej jest niezbędne [1.11], [1.15].

Skuteczność procedur syntezy logicznej można wykazać nawet na najprostszych przykładach. Na przykład prosta 10-argumentowa funkcja boolowska TL27 (specyfikację tej funkcji podano w podrozdz. 2.6) syntezowana programem ISE 14.7 może być zrealizowana na 21 4-wejściowych komórkach logicznych układu Spartan-III. Nie jest to sytuacja odosobniona, gdyż synteza Synteza programem Vivado 2015.4.2 w układzie Virtex-7 wymaga zastosowania pięciu 6-wejściowych, trzech 5-wejściowych komorek oraz jednej 4-wejściowej komórki. Niewiele lepiej jest dla systemu Altera Quartus II i układu Cyclone III: siedmiu 4-wejściowych oraz trzech 3-wejściowych komórek logicznych.

Ta sama funkcja poddana zaawansowanym procedurom syntezy logicznej może być zrealizowana na 2 (dwóch!) 4 wejściowych komórkach logicznych . Do uzyskania tak prostej struktury trzeba zastosować dwie procedury syntezy logicznej: procedurę redukcji argumentów oraz procedurę dekompozycji funkcjonalnej.

Sytuacja nie poprawia się nawet dla układów specjalnie przygotowywanych do realizacji na pamięciach. Charakterystycznym przykładem może być układ arytmetyki rozproszonej [1.13] filtru f5 [1.5]. Układ ten w reprezentacji za pomocą w pełni określonych funkcji boolowskich jest opisany tablicą o 11 zmiennych wejściowych i 11 wyjściach. Jego implementacja w strukturze Virtex-7 wykonana programem Vivado 2015.4.2 wymaga zastosowania 531 komórek (w tym 330 6-wejściowych, 86 5-wejściowych). Ponieważ układ ten, poddany procedurze redukcji argumentów jest funkcją zaledwie 7 argumentów, jego realizacja (wykonywana w tej samej strukturze programem Vivado) zajmuje 9 komórek 6-wejściowych oraz po jednej 5-, 4-, i 2-wejściowej. Tak wielka różnica w jakości rozwiązania nie może być spowodowana tylko jakością wykonania oprogramowania – u jej podstaw musi leżeć odmienna metodyka syntezy. Potwierdzeniem tego przypuszczenia są najnowsze prace w tej dziedzinie [1.1], [1.15] wykazujące, że potencjalne możliwości redukcji argumentów i dekompozycji funkcjonalnej nie są jeszcze w pełni wykorzystane.

Zaawansowane procedury syntezy logicznej mogą być również stosowane w zagadnieniach związanych z klasyfikacją danych, obejmowanych nazwą eksploracji danych [1.7], [1.8]. *Eksploracja danych* (ang. data mining), nazywana często *odkrywaniem wiedzy w bazach danych* (ang. *knowledge discovery in databases*), jest dynamicznie rozwijającą się dziedziną informatyki o szerokich zastosowaniach, m.in. w telekomunikacji, inżynierii biomedycznej, bankowości, itp.

Jednym z ważniejszych zastosowań tych algorytmów są systemy wykrywania anomalii w sieciach telekomunikacyjnych. Są to systemy pracujące wg typowego schematu maszynowego uczenia, gdyż kombinacja reguł oraz algorytmów klasyfikacji służy do wykrywania anomalii na podstawie analizy danych treningowych.

Innym typowym zastosowaniem jest wspomaganie decyzji podejmowanych przy diagnozie różnych chorób. Polega to na generowaniu reguł decyzyjnych obliczanych na podstawie baz danych zgromadzonych z badań wielu pacjentów. Obliczone w ten sposób reguły decyzyjne (tzw. klasyfikatory) pozwalają diagnozować nowych pacjentów.

W eksploracji danych typowe zadanie polega na tworzeniu reguł (wyrażeń boolowskich) reprezentujących pierwotne obiekty zapisane w tablicach danych. w przypadku układów logicznych procesy takie są określane mianem minimalizacji funkcji boolowskich. Uzyskiwane w wyniku takiego procesu reguły są wykorzystywane do klasyfikacji danych albo do kolejnych etapów optymalizacji logicznej (w przypadku układów cyfrowych). Oczywiście dane wejściowe algorytmów uogólniania reguł 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 [1.6], a w przypadku tablic funkcji logicznych nieokreśloność argumentu

wektora wejściowego oznacza występowanie w specyfikacji wektorów o wszystkich możliwych wartościach danego argumentu [1.3].

Na abstrakcyjnym poziomie algorytmów eksploracja danych polega m.in. na tzw. uogólnianiu/generacji reguł decyzyjnych, redukcji atrybutów, hierarchicznym podejmowaniu decyzji. Można wykazać, że algorytmy te są odpowiednikami algorytmów syntezy logicznej, a w szczególności tych które powstały w czasie ostatnich 30 lat. Na przykład generacja/uogólnianie reguł decyzyjnych – jest to typowa procedura stosowana w eksploracji danych i odpowiada minimalizacji funkcji boolowskiej, redukcja atrybutów odpowiada redukcji argumentów, natomiast hierarchiczne podejmowanie decyzji jest to dekompozycja funkcjonalna.

2 Algebra Boole'a i przekształcenia boolowskie

2.1 Aksjomaty algebry Boole'a

Algebra Boole'a jest modelem matematycznym operacji na sygnałach binarnych reprezentujących sygnały elektryczne o dwóch wartościach: 0 lub 1. Wartości te są przyporządkowane dwom poziomom napięcia wytwarzanego przez (elektroniczne) układy logiczne. Najczęściej przyjmuje się, że napięciu wysokiemu jest przyporządkowana wartość sygnału 1, natomiast napięciu niskiemu – wartość 0.

Algebra Boole'a jest algebrą z trzema operacjami na dwuwartościowych argumentach, które przyjmują wartości: 0 i 1. Rezultaty tych operacji są także dwuwartościowe. Te trzy operacje to:

- suma logiczna (suma boolowska, dysjunkcja),
- iloczyn logiczny (iloczyn boolowski, koniunkcja),
- negacja (inwersja).

Dwie pierwsze operacje są wieloargumentowe, a trzecia jest jednoargumentowa.

Operacja sumy logicznej (OR) jest zdefiniowana następująco: jeżeli co najmniej jeden z argumentów jest równy 1, to wynik jest równy 1, zatem suma logiczna jest równa 0 tylko dla przypadku, gdy wszystkie argumenty są równe 0. Działania te zapisujemy następująco:

0 + 0 = 0

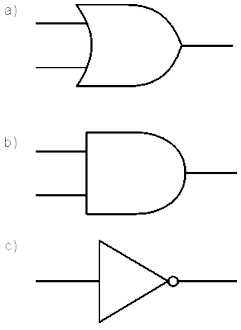
0 + 1 = 1

1 + 0 = 1

1 + 1 = 1

gdzie + oznacza operację OR. Operację OR realizuje bramka OR o symbolu graficznym jak na rys. 1.1a.

Operacja iloczynu logicznego (AND) jest zdefiniowana następująco: wynik iloczynu jest równy 1 wtedy i tylko wtedy, gdy wszystkie argumenty przyjmują wartość 1, co zapisujemy w następujący sposób:



Rys. 1.1. Bramki logiczne a) OR, b) AND, c) NOT

$0 \cdot 0 = 0$

$0 \cdot 1 = 0$

$1 \cdot 0 = 0$

$1 \cdot 1 = 1$

gdzie · oznacza operację AND. Operację AND realizuje bramka AND o symbolu graficznym podanym na rys. 1.1b.

Operacja negacji (NOT) zmienia wartość argumentu na przeciwny. Negacją 0 jest 1, a negacją 1 jest 0, co zapisujemy:

$\backslash bar\{1\} = 0$

$\backslash bar\{0\} = 1$

Operacja NOT zmiennej *x*, jest oznaczana $\bar{x}$, a jej symbol graficzny (bramka NOT) podany jest na rys. 1.1c.

Z punktu widzenia syntezy logicznej w technice cyfrowej specjalnemu zainteresowaniu podlega dwuelementowa algebra Boole'a, którą będziemy rozumieli jako system <*B*, +, · , 0, 1>, gdzie zbiorem jest *B* = {0, 1}. Dwuwartościowa (zwana również binarną) algebra Boole'a stanowi podstawę nowoczesnej syntezy logicznej formułując prawa jakim podlegają zmienne boolowskie tj. zmienne ze zbioru *B*. Prawa te (z uwzględnieniem praw De Morgana) podajemy zbiorczo w formie zestawienia, w którym każde prawo jest zaopatrzone w odpowiednią nazwę.

Prawa algebry Boole'a

Własności stałych

$\mathrm{a} + 0 = \mathrm{a} \qquad a \cdot 0 = 0$

$a + 1 = 1 \qquad a \cdot 1 = a$

Własności negacji

$a + \bar{a} = 1 \qquad a \cdot \bar{a} = 0$

Podwójna negacja

$\bar{\bar{a}} = a$

Idempotentność

$a + a = a \qquad a \cdot a = a$

Przemienność

$a + b = b + a \qquad a \cdot b = b \cdot a$

Łączność

$a + (b + c) = (a + b) + c \qquad a \cdot (b \cdot c) = (a \cdot b) \cdot c$

Rozdzielność

$$a + (b \cdot c) = (a + b) \cdot (a + c) \qquad a \cdot (b + c) = a \cdot b + a \cdot c$$

Prawa De Morgana

$$y = \overline{a \cdot b} = \bar{a} + \bar{b} \qquad y = \overline{a + b} = \bar{a} \cdot \bar{b}$$

W algebrze Boole'a, operacje „+" (dysjunkcja) i „·" (koniunkcja) nazywa się również przez analogię do arytmetyki odpowiednio dodawaniem i mnożeniem. Operacje dodawania i mnożenia są przemienne oraz rozdzielne względem siebie. Elementy binarne 0 oraz 1 spełniają rolę elementu neutralnego odpowiednio względem operacji dodawania i mnożenia. Dla każdego elementu *a* istnieje element$\bar{a}$, nazywany negacją, spełniający odpowiednie własności.

Starszeństwo działań w algebrze Boole'a jest takie same jak w zwykłej arytmetyce (np. wyrażenie *a* + *b*•*c* interpretujemy jako *a* + (*bc*), a nie jako (*a* + *b*)*c*, a nawiasy są opuszczane tam, gdzie nie prowadzi to do nieporozumień; opuszczamy także znak mnożenia „·", a zamiast symbolu „+", często jest używany symbol $\vee$.

Wyrażenie boolowskie to formuła, w której zmienne boolowskie połączone są operatorami: + (OR), · (AND), $\bar{x}$ (NOT).

Na przykład:

a + *b* + *c* · *d* + *e*

a + *b* + *cd* + *e*

a + *b*(*d* + *e*).

W zapisie wyrażeń boolowskich kropkę często pomija się, a kolejność operacji przyjmuje się następująco:

1. NOT
2. AND
3. OR

Kolejność ta może być zmieniona przez stosowanie nawiasów.

Typowym zastosowaniem algebry Boole'a jest uproszczenie wyrażeń boolowskich. Na przykład:

$$\bar{a}\mathrm{bc} + \mathrm{a}\bar{b}\bar{c} + \mathrm{a}\bar{b}\mathrm{c} + \mathrm{ab}\bar{c} + \mathrm{abc}$$
$$= \bar{a}\mathrm{bc} + \mathrm{a}\bar{b}\left(\bar{c} + c\right) + \mathrm{ab}\left(\bar{c} + c\right)$$
$$= \bar{a}\mathrm{bc} + \mathrm{a}\bar{b} + \mathrm{ab} = \bar{a}\mathrm{bc} + \mathrm{a}\left(\bar{b} + b\right)$$
$$= \bar{a}\mathrm{bc} + \mathrm{a} =$$

(korzystamy z własności *a* + *abc* = *a*(1 + *bc*) = *a*)

$$= \mathrm{a} + \bar{a}\mathrm{bc} + \mathrm{abc} = \mathrm{a} + \mathrm{bc}$$

2.2 Przekształcenia boolowskie

Innym typowym zastosowaniem algebry Boole'a jest przekształcanie wyrażeń boolowskich z postaci typu „iloczyn sum" (CNF – *Conjunctive Normal Form*) na postać typu „suma iloczynów" (DNF – *Disjunctive Normal Form*).

Korzysta się przy tym z zasad uproszczonego mnożenia, z których pierwsza jest omawianą już zasadą rozdzielności dodawania względem mnożenia, a drugą wykazujemy poniżej.

Wykazać, że: $(x + y)(\bar{x} + z) = xz + \bar{x}y$

$(x + y)(\bar{x} + z) = x\bar{x} + xz + \bar{x}y + yz = xz + \bar{x}y + yz = xz + \bar{x}y + 1yz = xz + \bar{x}y + (x + \bar{x})yz ==$
$xz + \bar{x}y + xyz + \bar{x}yz = xz + xyz + \bar{x}y + \bar{x}yz = xz(1 + y) + \bar{x}yy(1 + 1) = xz + \bar{x}y$

PRZYKŁAD 1.1

Podane wyrażenie typu „iloczyn sum":

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

przedstawić w postaci sumy iloczynów.

Rozwiązanie

$(A + B + \bar{C})(A + B + D)(A + B + E)(A + \bar{D} + E)(\bar{A} + C) = (A + B + \bar{C}DE)\big(AC + \bar{A}(\bar{D} + E)\big) =$
$(A + B + \bar{C}DE)(AC + \bar{A}\bar{D} + \bar{A}E) = AC + ABC + \bar{A}B\bar{D} + \bar{A}BE + \bar{A}\bar{C}DE = AC + \bar{A}B\bar{D} + \bar{A}BE + \bar{A}\bar{C}DE$

Ogromne znaczenie w syntezie logicznej mają przekształcenia wyrażeń boolowskich jednorodnych, tzn. takich w których zmienne występują wyłącznie w postaci prostej albo zanegowanej.

PRZYKŁAD 1.2

Wyrażenie typu CNF: $(x_2+x_4)(x_3+x_4)(x_3+x_5)(x_1+x_2+x_3)$ można przekształcić do postaci DNF w sposób następujący:

$(x_4+x_2)(x_4+x_3)(x_3+x_5)(x_3+x_1+x_2) = (x_4+x_2x_3)(x_3+x_1x_5+x_2x_5) = x_3x_4+x_1x_4x_5+x_2x_4x_5+x_2x_3$

Nietrudno przypuszczać, że w praktycznych zastosowaniach będziemy mieli do czynienia z bardziej rozbudowanymi przekształceniami CNF na DNF. W celu ułatwienia skomplikowanych obliczeń i zapisów zmienne boolowskie x_i, x_j, x_k będziemy reprezentowali ich indeksami *i*, *j*, *k*.

PRZYKŁAD 1.3

(3 + 7)(2 + 5 + 8)(3 + 6 + 8)(6 + 7 + 8)(3 + 5 + 6) = (3 + 7) (3 + 6 + 58)(8 + (2 + 5) (6 + 7)) = (3 + ~~36 + 358 + 37~~ + 67 + 578)(8 + 26 + 27 + 56 + 57) = 38 + 236 + 237 + + 356 + 357 + 678 + 267 + ~~267~~ + 567 + ~~567~~ + ~~578 + 25678 + 2578 + 5678~~ + 578

= 38 + 236 + 237 + 356 + 357 + 678 + 267 + 567 + 578

Transformacja CNF na DNF jest również wygodnym i często stosowanym narzędziem przy obliczaniu pokrycia kolumnowego binarnej macierzy M.

Pokryciem kolumnowym binarnej macierzy reprezentowanej tablicą *M*:

$$M = [m_{ij}], i \in \{1, \dots, w\}, j \in \{1, \dots, n\}$$

jest zbiór $L \in \{1, \dots, n\}$ taki, że dla każdego $i \in \{1, \dots, w\}$ istnieje $j \in L$, dla którego $m_{ij} = 1$. Jeżeli usunięcie którejkolwiek z kolumn skutkuje brakiem pokrycia, jest to minimalne pokrycie kolumnowe. Inaczej mówiąc elementami pokrywanymi są wiersze M, a pokrywającymi – kolumny tej macierzy. Jednak brak formalnych elementów pokrywanych i pokrywających skłania do wprowadzenia nazwy „pokrycie kolumnowe".

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.

W celu obliczenia wszystkich minimalnych pokryć kolumnowych należy zapisać zbiory kolumn wskazywane 1 (jedynkami) w postaci iloczynu sum, a następnie uzyskaną koniunkcję sum przekształcić do minimalnego wyrażenia boolowskiego typu suma iloczynów. Składniki tych iloczynów reprezentują wszystkie minimalne pokrycia kolumnowe.

PRZYKŁAD 1.4

Tabl. 1.1

L_1	L_2	L_3	L_4	L_5	L_6	L_7
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

Dla macierzy M (Tabl. 1.1) stosowne obliczenia są następujące:

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

$=(L_4 + L_2 L_3)(L_7 + L_6(L_2 + L_3)) = (L_4 + L_2 L_3)(L_7 + L_2 L_6 + L_3 L_6) = L_4 L_7 + L_2 L_4 L_6 + L_3 L_4 L_6 + + L_2 L_3 L_7 + L_2 L_3 L_6$

Na tej podstawie stwierdzamy, że wszystkie minimalne pokrycia kolumnowe macierzy z Tabl. 1.1 są reprezentowane zbiorami: {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}. Zauważmy, że dla każdego wiersza tej tablicy elementy wskazywane przez kolumny L_i należące do obliczonych podzbiorów zawierają zawsze co najmniej

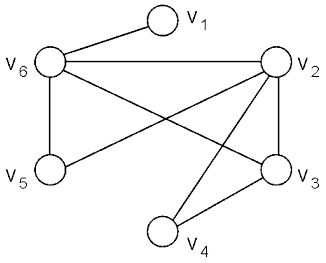
3 Elementy teorii grafów

3.1 Graf prosty, pojęcie relacji

Grafem prostym (niezorientowanym) nazywamy parę $G = (V, E)$, gdzie V jest niepustym skończonym zbiorem wierzchołków, a E jest skończonym zbiorem krawędzi – nieuporządkowanych par różnych elementów ze zbioru V [1.18]. Przykład grafu podany jest na rys. 1.2. Jest to graf, w którym zbiór wierzchołków $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ oraz zbiór krawędzi $E = \{(v_1,v_6), (v_2,v_3), (v_2,v_4), (v_2,v_5), (v_2,v_6), (v_3,v_4), (v_3,v_6), (v_5,v_6)\}$.

Dowolny podzbiór wierzchołków, w którym każde dwa wierzchołki są połączone krawędzią nazywamy *kliką*. Klikę, która nie jest zawarta w żadnej istotnie innej klice, nazywamy maksymalną. Najliczniejszą klikę w danym grafie nazywamy *największa kliką*. Klikami dla grafu z rys. 1.2 są następujące zbiory:

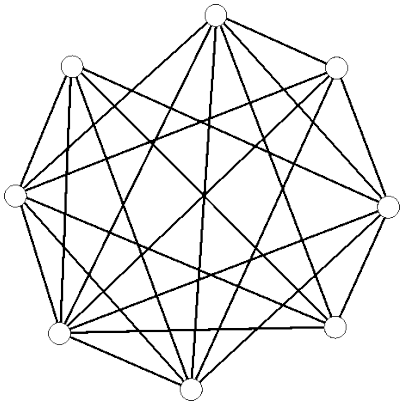
$V = \{v_2, v_3, v_4\}$, $V = \{v_2, v_3, v_6\}$, $V = \{v_2, v_5, v_6\}$.



Rys. 1.1. Przykład grafu

Zbiorem niezależnym nazywamy dowolny zbiór wierzchołków, które nie są sąsiednie w danym grafie. Analogicznie określamy pojęcie *maksymalnego zbioru niezależnego*. Przykłady zbiorów niezależnych dla grafu z rys. 1.2: $V = \{v_1,v_4, v_5\}$, $V = \{v_1, v_3, v_5\}$, $V = \{v_4, v_6\}$, $V = \{v_1, v_2\}$.

Obliczanie klik nie jest zadaniem łatwym. Można się zastanawiać jak obliczyć maksymalne kliki w grafie podanym na rys. 1.3. Dlatego problem obliczania maksymalnych klik warto sprowadzić do problemu obliczania maksymalnych klas zgodności definiowanych dla danej relacji zgodności. Najpierw zdefiniujemy ważne pojęcie iloczynu kartezjańskiego.



Rys. 1.2. Przykład grafu

Iloczynem kartezjańskim zbiorów A i B, oznaczanym $A \times B$ nazywamy zbiór wszystkich par uporządkowanych (a, b), takich że pierwszy element pary należy do zbioru A ($a \in A$), natomiast drugi do B ($b \in B$), czyli:

$A \times B = \{(a,b): a \in A, b \in B\}$

PRZYKŁAD 1.5

Niech $A = \{p,q\}$ oraz $B = \{r,s,t\}$,

wtedy $A \times B = \{(p,r), (p,s), (p,t), (q,r), (q,s), (q,t)\}$.

Relacją nazywamy dowolny podzbiór iloczynu kartezjańskiego zbiorów *A*, *B*. Typowe własności relacji na zbiorze *A* (czyli $A \times A$) są definiowane następująco:

- zwrotność: $\forall a \in A: aRa$
- symetria: $\forall a, b \in A: aRb \Rightarrow bRa$
- przechodniość: $\forall a, b, c \in A: aRb, bRc \Rightarrow bRc$

Najważniejszymi relacjami stosowanymi w syntezie logicznej są relacje równoważności i zgodności.

Relację, która jest zwrotna i symetryczna nazywamy *relacją zgodności*. Relacja zgodności jest również nazywana relacją nierozróżnialności. Relacja zgodności klasyfikuje elementy zbioru w nierozłączne podzbiory zwane *Systemem zbiorów* (więcej na ten temat w rozdz. 1.4).

Szczególnym przypadkiem relacji zgodności jest relacja równoważności. *Relacja równoważności* jest relacją zwrotną, symetryczną i przechodnią. Relacja równoważności dzieli zbiór A na rozłączne podzbiory, co w konsekwencji prowadzi do pojęcia podziału (zbioru) tworząc bardzo użyteczny w syntezie logicznej rachunek podziałów omawiany w następnym rozdziale.

Własności relacji zgodności pokrywają się z intuicyjnym rozumieniem zgodności:

a) każdy element jest zgodny z samym sobą,

b) jeśli element v_1 jest zgodny z v_2, to również v_2 jest zgodny z v_1,

c) jeśli v_1 jest zgodny z v_2 oraz v_2 jest zgodny z v_3, to z tego nie wynika, że v_1 jest zgodny z v_3.

Relacja zgodności umożliwia wprowadzenie pojęcia *Maksymalnych Klas Zgodności* (MKZ).

Zbiór par określających relację zgodności nazywa się zbiorem par zgodnych. Pary zgodne umożliwiają wyznaczenie maksymalnych zbiorów zgodnych.

Zbiór $V = \{v_1,\ldots,v_p\}$ nazywamy maksymalnym zbiorem zgodności (maksymalną klasą zgodności), jeżeli każda para v_i, v_j wzięta z tego zbioru jest zgodna oraz nie istnieje żaden inny zbiór elementów zgodnych *V'*, zawierający *V*. Maksymalne zbiory zgodne są odpowiednikiem klik w grafie prostym.

Z powyższego wynika, że w celu obliczania klik metodą maksymalnych klas zgodności należy:

1. Zapisać pary SPRZECZNE (v_i, v_j), (v_k, v_l), (v_p, v_q), w postaci koniunkcji dwuskładnikowych sum $(v_i + v_j)(v_k + + v_l)(v_p, + v_q) \ldots$
2. Koniunkcję dwuskładnikowych sum przekształcić do minimalnego wyrażenia boolowskiego typu suma iloczynów $v_iv_jv_k + v_pv_qv_rv_s + \ldots$

Wtedy MKZ są uzupełnieniami zbiorów reprezentowanych przez składniki iloczynowe tego wyrażenia.

PRZYKŁAD 1.6

Dla grafu z rys.1.2 pary zgodne są: (v_1, v_6), (v_2, v_3), (v_2, v_4), (v_2, v_5), (v_2, v_6), (v_3, v_4), (v_3, v_6), (v_5, v_6). Zatem pary sprzeczne będą:

$E = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_1, v_5), (v_3, v_5), (v_4, v_5), (v_4, v_6)\}$

Obliczamy wyrażenie boolowskie typu „koniunkcja sum”:

$(v_1 + v_2)(v_1 + v_3)(v_1 + v_4)(v_1 + v_5)(v_3 + v_5)(v_4 + v_5)(v_4 + v_6) =$

$= (v_1 + v_2)(v_1 + v_3)(v_1 + v_4)(v_1 + v_5)(v_4 + v_5)(v_4 + v_6)(v_3 + v_5) =$

(stosujemy zasadę $(a + b)(a + c) = a + bc$)

$= (v_1 + v_2v_3v_4v_5)(v_4 + v_5v_6)(v_3 + v_5) =$

(wymnażamy i redukujemy zbędne składniki)

$= (v_1v_4 + v_1v_5v_6 + v_2v_3v_4v_5 + \cancel{v_2v_3v_4v_5v_6})(v_3 + v_5) =$

$= (v_1v_4 + v_1v_5v_6 + v_2v_3v_4v_5)(v_3 + v_5) =$

$= v_1v_3v_4 + \cancel{v_1v_3v_5v_6} + v_2v_3v_4v_5 + v_1v_4v_5 + v_1v_5v_6 + \cancel{v_2v_3v_4v_5} =$

$= v_1v_3v_4 + v_1v_4v_5 + v_1v_5v_6 + v_2v_3v_4v_5$

Odejmując od zbioru $\{v_1, \ldots, v_6\}$ zbiory reprezentowane przez poszczególne składniki uzyskanego wyrażenia obliczamy wszystkie maksymalne klasy zgodne, czyli kliki:

$\{v_1, \ldots, v_6\} - \{v_1, v_3, v_4\} = \{v_2, v_5, v_6\}$

$\{v_1, \ldots, v_6\} - \{v_1, v_4, v_5\} = \{v_2, v_3, v_6\}$

$\{v_1, \ldots, v_6\} - \{v_1, v_5, v_6\} = \{v_2, v_3, v_4\}$

$\{v_1, \ldots, v_6\} - \{v_2, v_3, v_4, v_5\} = \{v_1, v_6\}$

W podobny sposób możemy sformułować algorytm obliczania *Maksymalnych Zbiorów Niezależnych* (MZN).

1. Zapisać pary zgodne (v_i, v_j), (v_k, v_l), (v_p, v_q), …w postaci koniunkcji dwuskładnikowych sum $(v_i, + v_j)(v_k, + v_l)(v_p, + v_q)$ …
2. Koniunkcję dwuskładnikowych sum przekształcić do minimalnego wyrażenia boolowskiego typu suma iloczynów $v_iv_jv_k + v_pv_qv_rv_s$ +…

Wtedy MZN są uzupełnieniami zbiorów reprezentowanych przez składniki iloczynowe tego wyrażenia.

PRZYKŁAD 1.7

Obliczmy wszystkie MZN dla grafu z rys. 1.2.

$E = \{(v_1, v_6), (v_2, v_3), (v_2, v_4), (v_2, v_5), (v_2, v_6), (v_3, v_4), (v_3, v_6), (v_5, v_6)\}$

Obliczamy wyrażenie boolowskie typu „koniunkcja sum”:

$(v_1 + v_6)(v_2 + v_3)(v_2 + v_4)(v_2 + v_5)(v_2 + v_6)(v_3 + v_4)(v_3 + v_6)(v_5 + v_6) =$

$= (v_2 + v_3)(v_2 + v_4)(v_2 + v_5)(v_2 + v_6)(v_6 + v_1)(v_6 + v_3)(v_6 + v_5)(v_3 + v_4) =$

$= (v_2 + v_3v_4v_5v_6)(v_6 + v_1v_3v_5)(v_3 + v_4) =$

$= (v_2v_6 + v_1v_2v_3v_5 + v_3v_4v_5v_6 + \cancel{v_1v_3v_4v_5v_6})(v_3 + v_4) =$

$= v_2v_3v_6 + v_1v_2v_3v_5 + v_3v_4v_5v_6 + v_2v_4v_6 + \cancel{v_1v_2v_3v_4v_5} + \cancel{v_3v_4v_5v_6} =$

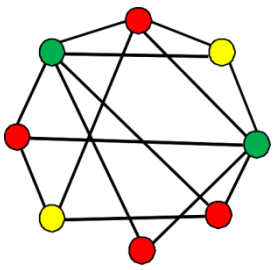
$= v_2v_3v_6 + v_1v_2v_3v_5 + v_3v_4v_5v_6 + v_2v_4v_6 =$

$= v_2v_3v_6 + v_2v_4v_6 + v_1v_2v_3v_5 + v_3v_4v_5v_6$

Stąd maksymalne zbiory niezależne MZN: $\{v_1, v_4, v_5\}$, $\{v_1, v_3, v_5\}$, $\{v_4, v_6\}$, $\{v_1, v_2\}$.

3.2 Kolorowanie grafu

Kolorowaniem grafu nazywamy przyporządkowanie kolorów do wierzchołków grafu w taki sposób, aby żadna para wierzchołków sąsiednich nie miała takiego samego koloru. Przykład prawidłowo pokolorowanego grafu pokazany jest na rys. 1.4.



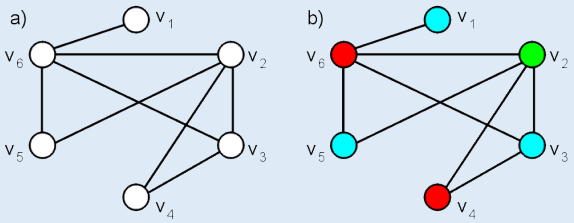
Rys. 1.3. kolorowanie grafu

Podstawowym problemem jest zadanie obliczenia najmniejszej liczby kolorów zapewniających prawidłowe pokolorowanie grafu. Jest to tzw. liczba chromatyczna grafu. Korzystając z pojęcia maksymalnych zbiorów niezależnych można algorytm kolorowania grafu sprowadzić do wykonania następujących obliczeń:

1. Obliczyć wszystkie maksymalne zbiory niezależne MZN. Uzyskujemy *Rodzinę Maksymalnych Zbiorów Niezależnych* (RMZN).
2. Obliczyć pokrycie zbioru wierzchołków *V* minimalną liczbą maksymalnych zbiorów niezależnych (tzw. minimalne pokrycie). Minimalne pokrycie reprezentuje minimalny zbiór kolorów.
3. W uzyskanym pokryciu usunąć elementy powtarzające się.

PRZYKŁAD 1.8

$E = \{(v_1, v_6), (v_2, v_3), (v_2, v_4), (v_2, v_5), (v_2, v_6), (v_3, v_4), (v_3, v_6), (v_5, v_6)\}$



Rys. 1.4. Graf i jego kolorowanie

Dla grafu z rys. 1.5a należy wyznaczyć kolorowanie z minimalną liczbą kolorów. W tym celu najpierw obliczamy maksymalne zbiory niezależne: $\{v_1, v_4, v_5\}$, $\{v_1, v_3, v_5\}$, $\{v_4, v_6\}$, $\{v_1, v_2\}$. Następnie tworzymy minimalną podrodzinę pokrywającą zbiór $V = \{v_1, \ldots, v_6\}$: $\{v_1, v_3, v_5\}$, $\{v_4, v_6\}$, $\{v_1, v_2\}$. Wreszcie po usunięciu wierzchołków powtarzających się uzyskujemy zbiory rozłączne reprezentują kolorowanie: $\{v_1,$

Na zakończenie podamy jeszcze jedną – wygodną do stosowania w praktyce – metodę obliczania maksymalnych klas zgodności.

Metoda wg par zgodnych

Niech E będzie relacją na zbiorze $V = \{v_1, ..., v_n\}$, tzn. zbiorem par zgodnych (sąsiednich) v_i, v_j: $i, j \in \{1, ..., n\}$ oraz $i \neq j$. Obliczenie (maksymalnych) klas zgodności w zbiorze V przy danej E sprowadza się do wykonania następujących czynności.

1. Zapisujemy elementy v w rodzinach S_j zgodnie z zasadą $v_i \in S_j$ tylko wtedy, gdy v_j, v_i jest parą zgodną oraz $i < j$.
2. Jeżeli RKZ_k jest rodziną klas zgodności uzyskaną w k-tym kroku algorytmu, to rodzinę RKZ_{k+1} uzyskujemy, obliczając przecięcie każdej $KZ \in RKZ_k$ ze zbiorem S_{k+1}. Wyróżniamy przy tym następujące przypadki:

   a) $S_{k+1} = \phi$ i wtedy RKZ_{k+1} jest powiększana o klasę $KZ = \{k+1\}$,

   b) $KZ \cap S_{k+1} = \phi$ wtedy klasa KZ nie ulega zmianie,

   c) $KZ \cap S_{k+1} \neq \phi$ wtedy powstaje nowa klasa $KZ' = KZ \cap S_{k+1} \cup \{k+1\}$.

PRZYKŁAD 1.9

W przykładzie obliczymy maksymalne klasy zgodności algorytmem korzystającym z par zgodnych. Następujące pary są zgodne: (v_1, v_2), (v_1, v_3), (v_1, v_5), (v_2, v_3), (v_2, v_5), (v_3, v_4), (v_3, v_5), (v_3, v_7), (v_4, v_6), (v_4, v_7), (v_5, v_6), (v_6, v_7). Na tej podstawie wyznaczamy zbiory S_j i tworzymy kolejne listy (rodziny RKZ_k) klas zgodnych, które dla uproszczenia zapisów podawać będziemy w postaci zbiorów indeksów:

$S_1 = \phi$	{1}
$S_2 = \{1\}$	{1, 2}
$S_3 = \{1, 2\}$	{1, 2, 3}
$S_4 = \{3\}$	{3, 4}, {1, 2, 3}
$S_5 = \{1, 2, 3\}$	~~{3, 5}~~, {1, 2, 3, 5}, {3, 4}
$S_6 = \{4, 5\}$	{5, 6}, {4, 6}, {1, 2, 3, 5}, {3, 4}
$S_7 = \{3, 4, 6\}$	~~{6, 7}~~, {4, 6, 7}, ~~{3, 7}~~, {3, 4, 7}, {5, 6}, {1, 2, 3, 5}

PRZYKŁAD 1.10

Dla relacji R, w której zbiór par zgodnych jest $E = \{(e_1,e_2), (e_1,e_3), (e_1,e_4), (e_1,e_5), (e_1,e_6), (e_1,e_7), (e_2,e_3), (e_2,e_5), (e_2,e_6), (e_2,e_7), (e_3,e_4), (e_3,e_5), (e_3,e_6), (e_3,e_8), (e_4,e_6), (e_4,e_7), (e_4,e_8), (e_5,e_6), (e_5,e_7), (e_5,e_8), (e_6,e_7), (e_6,e_8), (e_7,e_8)\}$ obliczyć maksymalne klasy zgodności:

a) metodą wg par zgodnych,

b) metodą wg par sprzecznych.

Rozwiązanie a)

$S_1 = \varnothing$	e_1
$S_2 = e_1$	e_1, e_2
$S_3 = e_1, e_2$	e_1, e_2, e_3
$S_4 = e_1, e_3$	$e_1, e_3, e_4 / e_1, e_2, e_3$ [1)]
$S_5 = e_1, e_2, e_3$	~~e_1, e_2, e_5~~$/e_1, e_2, e_3, e_5 / e_1, e_3, e_4$
$S_6 = e_1, e_2, e_3, e_4, e_5$	$e_1, e_2, e_3, e_5, e_6 / e_1, e_3, e_4, e_6 /$ ~~e_1, e_3, e_4~~
$S_7 = e_1, e_2, e_4, e_5, e_6$	$e_1, e_2, e_5, e_6, e_7 / e_1, e_4, e_6, e_7 / e_1, e_2, e_3, e_5, e_6 / e_1, e_3, e_4, e_6$
$S_8 = e_3, e_4, e_5, e_6, e_7$	$e_5, e_6, e_7, e_8 / e_4, e_6, e_7, e_8 / e_3, e_5, e_6, e_8 / e_3, e_4, e_6, e_8 / e_1, e_2, e_5, e_6, e_7 / e_1, e_4, e_6, e_7 / e_1, e_2, e_3, e_5, e_6 / e_1, e_3, e_4, e_6$

Rozwiązanie b)

Pary sprzeczne: (e_1, e_8) (e_2, e_4) (e_2, e_8) (e_3, e_7) (e_4, e_5)

$$(e_8 + e_1)(e_8 + e_2)(e_4 + e_2)(e_4 + e_5)(e_3 + e_7) = = (e_8 + e_1e_2)(e_4 + e_2e_5)(e_3 + e_7) =$$

$$= (e_4e_8 + e_2e_5e_8 + e_1e_2e_4 + e_1e_2e_5)(e_3 + e_7) = e_3e_4e_8 + e_2e_3e_5e_8 + e_1e_2e_3e_4 + e_1e_2e_3e_5 + e_4e_7e_8 + + e_2e_5e_7e_8 + e_1e_2e_4e_7 + e_1e_2e_5e_7$$

W obu przypadkach maksymalne klasy zgodności są następujące:

$\{e_1, e_2, e_5, e_6, e_7\}$	$\{e_1, e_2, e_3, e_5, e_6\}$
$\{e_1, e_4, e_6, e_7\}$	$\{e_5, e_6, e_7, e_8\}$
$\{e_4, e_6, e_7, e_8\}$	$\{e_1, e_2, e_3, e_5, e_6\}$
$\{e_1, e_3, e_4, e_6\}$	$\{e_3, e_5, e_6, e_8\}$
$\{e_3, e_4, e_6, e_8\}$	

[1)] Do rozdzielenia klas zgodności użyto znaku „/” (zamiast nawiasów „{” i „}”

4 Kostki, systemy zbiorów i podziały

Zajmiemy się rachunkiem kostek i rachunkiem podziałów, których podstawy wynikają z jednej strony z klasycznej algebry Boole'a, a z drugiej z algebry ternarnej [1.4] i algebry zbiorów. Oprócz typowych wartości logicznych 0 i 1, stosować będziemy wartość nieokreśloną, którą oznaczać będziemy, w zależności od interpretacji: $*$ lub –.

Na zbiorze {0, 1, $*$} definiujemy *częściowy porządek* $\subseteq$, w którym:

$0 \subseteq 0, * \subseteq * , 1 \subseteq 1, 0 \subseteq *, 1 \subseteq *,$

oraz żadna inna para nie występuje w relacji $\subseteq$.

Ciągi elementów ze zbioru {0, 1, $*$} nazywać będziemy kostkami. Kostki będziemy oznaczać nieindeksowanymi literami, na przykład k, a ich składowe będziemy indeksowali, na przykład k_i. Również, w celu uproszczenia oznaczeń kostki (i wektory) będziemy zapisywać bez nawiasów i przecinków. Na przykład ($*$, 1, $*$, 0, 1) zapiszemy jako $*1*01$.

Częściowy porządek $\subseteq$ rozszerzamy na zbiór $\{0, 1,*\}^n$:

$k \subseteq k'$ wtedy i tylko wtedy, gdy $k_i \subseteq k'_i$ dla każdego i, $1 \leq i \leq n$.

Na przykład $01* \subseteq *1*$.

W zbiorze częściowo uporządkowanym $\langle\{0, 1, *\}, \subseteq \rangle$, definiujemy typowe pojęcie najmniejszego górnego ograniczenia LUB (*Least Upper Bound*), a mianowicie LUB{0} = 0, LUB{1} = 1, a ponadto operacja LUB dla każdego niepustego podzbioru z {0,1,$*$} wynosi $*$. Operację LUB rozszerzamy na zbiór $\{0, 1, *\}^n$ jako najmniejsze górne ograniczenie obliczane dla poszczególnych składowych odpowiednich kostek. Na przykład:

LUB{$*001$, 1101, 0101} = $**01$.

Na zbiorze {0, 1,$*$} określamy operację binarną, $\cup$ – *sumowanie*, dla której związek z najmniejszym górnym ograniczeniem jest następujący:

$k \cup k'$ = LUB{k, k'}.

Mamy wtedy:

$k \subseteq k'$ wtedy i tylko wtedy, gdy $k \cup k' = k'$.

Operacja $\cup$-sumowania jest idempotentna, przemienna i łączna. Jej uogólnienie dla każdego niepustego podzbioru S z $\{0, 1, *\}^n$, jest następujące:

$$\text{LUB } S = \bigcup_{k \in S} k$$

Na zbiorze $\{0, 1, *\}$ definiujemy *relację zgodności* ~, gdzie:

$0 \sim 0, * \sim *, 1 \sim 1, 0 \sim * , 1 \sim *, * \sim 0 , * \sim 1,$

ale pary (0,1) i (1,0) nie podlegają relacji ~.

Zgodność jest zwrotna i symetryczna, to znaczy dla każdej $k, k' \in \{0, 1, *\}$, $k \sim k$ i $k \sim k'$ implikuje $k' \sim k$. Jednak nie jest to relacja przechodnia, gdyż $0 \sim *$ i $* \sim 1$, ale $0 \not\sim 1$.

Należy zauważyć, że:

$k \subseteq k'$ implikuje $k \sim k'$,

oraz

$k \sim k'$ implikuje albo $k \subseteq k'$ albo $k' \subseteq k$.

Relacja zgodności $\sim$ może być rozszerzona na obiekty ze zbioru $\{0, 1, *\}^n$:

$k \sim k'$ wtedy i tylko wtedy, gdy $k_i \sim k_i'$ dla każdego i, $1 \leq i \leq n$.

Na przykład, $01** \sim *10*$.

Największe dolne ograniczenie GLB (*Greatest Lower Bound*) istnieje wyłącznie dla pewnych podzbiorów zbioru $\{0, 1, *\}$. Na przykład GLB{0,1} nie istnieje, ale GLB{0,*} = 0. Wykorzystując GLB, definiujemy $\cap$-*iloczyn w* sposób następujący: $k \cap k'$ jest określony wtedy i tylko wtedy, gdy k i k' są zgodne. Zatem w $\cap$-iloczynie wartości *pewnej* przez *niepewną*, wartość pewna przeważa. Ponadto,

$k \subseteq k'$ wtedy i tylko wtedy, gdy $k \cap k' = k$.

Jeśli *k, k',* oraz *k"* są parami zgodne, to $\cap$-iloczyn $(k \cap k') \cap k''$ jest określony i wynosi $k \cap (k' \cap k'')$. Zatem operacja $\cap$ jest łączna. Operacja $\cap$-iloczynu może być rozszerzona na dowolny zbiór *C* elementów parami zgodnych:

$$\mathrm{GLB}\ C \bigcup_{k \in C} k$$

Dalsze uogólnienie $\cap$-iloczynu dotyczy kostek zgodnych. Na przykład, $01{*}{*} \cap {*}10{*} = 010{*}$. Z każdym $k \in \{0,1,*\}^n$ kojarzymy zbiór *bin k* binarnych kostek (mintermów):

$$bin\ k = \{b \in \{0,1\}^n \mid b \subseteq k\}.$$

Na przykład, jeśli $k = 01{*}{*}$, wtedy *bin k* = {0100, 0101, 0110, 0111}.

Systemem zbiorów na zbiorze *S* nazywamy rodzinę zbiorów $\sigma = \{B_1,...,B_k\}$, w której podzbiory (tzw. bloki) są maksymalne w tym sensie, że inkluzja $B_i \subseteq B_j$ implikuje $i = j$. Na przykład, jeśli $S = \{1,2,3\}$, to σ = {{1,2}, {2, 3}} jest systemem zbiorów na *S*. Zapis ten upraszczamy następująco: $\sigma = \{\overline{1,2}; \overline{2,3}\}$. Zauważmy, że systemy zbiorów nie są zamknięte ze względu na operację iloczynu. Na przykład $\sigma = \{\overline{1,2}; \overline{2,3}\} \bullet \sigma = \{\overline{1,2}; \overline{2,3}\}$ nie jest systemem zbiorów.

Jeśli $\beta = \{B_i\}$ jest dowolną rodziną zbiorów na *S*, to

$$max\ \beta = \{B \subseteq S \mid B = B_i \text{ dla pewnego } i \text{ oraz } B \subseteq B_j \text{ implikuje } B_j = B\}.$$

Warto zauważyć*,* że operacja *max* odwzorowuje rodziny zbiorów w systemy zbiorów. *Iloczyn* dwóch systemów zbiorów σ i σ' jest definiowany następująco:

$$\sigma \circ \sigma' = max(\sigma \bullet \sigma').$$

Na przykład: $\{\overline{1,2}; \overline{3,4,5}\}$

$$\{\overline{1,2,3}; \overline{3,4,5}\} \bullet \{\overline{1,3,4}; \overline{1,5}, \overline{2,3,4}\} = \{\overline{1,3}; \overline{2,3}; \overline{3,4}; \overline{5}\}$$

Łatwo sprawdzić, że iloczyn systemów zbiorów jest idempotentny, łączny i przemienny. Kończąc powyższe rozważania przypomnijmy, że system zbiorów jest konsekwencją działania relacji zgodności.

Mając na uwadze, że relacja zgodności jest szczególnym przypadkiem relacji równoważności nietrudno stwierdzić, że wynikiem działania relacji równoważności jest podział zbioru na rozłączne podzbiory.

Formalnie, podziałem π zbioru *S* nazywać będziemy rodzinę rozłącznych podzbiorów zbioru *S*, których suma równa się *S*.

Niech *S* = {1, 2, 3, 4, 5, 6, 7, 8, 9}, to

τ_1 = {{1, 2}, {3, 4}, {5, 6}, {7, 8, 9}},

τ_2 = {{1, 6}, {2, 3}, {4, 5}, {7, 8}, {9}}

są podziałami zbioru *S*. Dla uproszczenia zapisów stosujemy zapis:

$$\tau_1 = \left(\overline{1,2}; \overline{3,4}; \overline{5,6}; \overline{7,8,9};\right)$$
$$\tau_2 = \left(\overline{1,6}; \overline{2,3}; \overline{4,5}; \overline{7,8}; \overline{9};\right)$$
$$\tau_3 = \left(\overline{1,2}; \overline{3}; \overline{4}; \overline{5,6}; \overline{7,8}; \overline{9}\right)$$

Dla podziałów wprowadza się relację $\leq$ oraz działanie iloczynu i sumy:

$$\Pi_1 \leq \Pi_2 \Leftrightarrow (\forall B_i \in \Pi_1)\left(\exists B_j \in \Pi_2\right)\left(B_i \subseteq B_j\right)$$

Największym i najmniejszym podziałem zbioru *S* są odpowiednio:

$$\Pi(1) = (\overline{1,2,\ldots,n})$$
$$\Pi(0) = (\overline{1}; \overline{2}; \ldots)$$

Niech $\tau_3 = \left(\overline{1,2}; \overline{3}; \overline{4}; \overline{5,6}; \overline{7,8}; \overline{9}\right)$. Wówczas $\tau_3 \leq \tau_1$. Analogiczny związek nie zachodzi między τ_1 i τ_2, ani między τ_2 i τ_3.

Podział Π nazywamy iloczynem podziałów Π_1 i Π_2 ($\Pi = \Pi_1 \cdot \Pi_2$), jeżeli Π jest największym podziałem (tzn. podziałem mającym największe bloki) spełniającym warunki: $\Pi \leq \Pi_1$ oraz $\Pi \leq \Pi_2$. Podział $\Pi_1 \cdot \Pi_2$ można wyznaczyć bardzo prosto wyznaczając wszystkie możliwe iloczyny w sensie teorii mnogości każdego bloku Π_1, z każdym blokiem Π_2; zbiór tak otrzymanych zbiorów jest poszukiwanym podziałem.

Sumą podziałów $\Pi_1 + \Pi_2$ nazywamy najmniejszy podział, nie mniejszy od Π_1 oraz od Π_2. Wyznaczenie $\Pi_1 + \Pi_2$ jest nieco trudniejsze. Niech podziały Π_1 i Π_2 mają odpowiednio bloki B_i i B_j takie, że $B_i \cap B_j \neq \phi$. Tworzymy wtedy nowy blok $B_{ij} = B_i \cup B_j$ i sprawdzamy, czy Π_1 i Π_2 zawiera taki blok B_k, że

$B_k \cap B_{ij} \neq \phi$. Jeżeli tak, to tworzymy nowy blok $B_k \cup B_{ij}$ itd. w wyniku takiego postępowania otrzymamy stopniowo podział $\Pi_1 + \Pi_2$.

Dla wprowadzonych podziałów mamy:

$$\tau_1 \cdot \tau_2 = \left(\overline{1}; \overline{2}; \overline{3}; \overline{4}; \overline{5}; \overline{6}; \overline{7,8}; \overline{9}\right)$$
$$\tau_1 + \tau_2 = \left(\overline{1,2,3,4,5,6}; \overline{7,8,9}\right)$$
$$\tau_3 \cdot \tau_1 = \left(\overline{1,2}; \overline{3}; \overline{4}; \overline{5,6}; \overline{7,8}; \overline{9}\right) = \tau_3$$
$$\tau_3 + \tau_1 = \left(\overline{1,2}; \overline{3,4}; \overline{5,6}; \overline{7,8}; \overline{7,8,9}\right) = \tau_1$$

Wygodne w zastosowaniach jest również pojęcie ilorazu podziałów (podziału ilorazowego). Niech Π_1 i Π_2 są podziałami na *S*. Podział $\Pi_1|\Pi_2$ jest podziałem ilorazowym Π_1 i Π_2, jeżeli jego elementy są blokami iloczynu $\Pi_1 \cdot \Pi_2$, a bloki są blokami Π_1. Na przykład: dla *S* = {1,2,3,4,5,6}, $\Pi_1 = \left\{\overline{1,2,5}; \overline{3,4,6}\right\}$, $\Pi_2 = \left\{\overline{1,2}; \overline{3,6}; \overline{4,5}\right\}$ podział ilorazowy będzie:

$$\Pi_1|\Pi_2 = \left\{\overline{(1,2)(5)}; \overline{(3,6)(4)}\right\}.$$

5 Zadania

ZADANIE 1.1

W zbiorze S = {1, 2, 3, 4, 5, 6, 7, 8} następujące pary są zgodne: (1,3), (1,7), (2,5), (2,8), (3,4), (3,5), (3,6), (4,5), (4,6), (5,7), (5,8), (6,7), (6,8). Obliczyć (sensowną metodą) wszystkie maksymalne klasy zgodności.

ZADANIE 1.2

Dla grafu $G = (V,E)$, w którym $V = \{1,2,3,4,5,6,7,8\}$, $E = \{(1,8), (2,4), (2,8), (3,7), (4,5)\}$ obliczyć maksymalne kliki metodą: a) zbiorów niezależnych, b) maksymalnych klas zgodności.

ZADANIE 1.3

Dla relacji R, w której zbiór par zgodnych jest $E = \{(B_1,B_2), (B_1,B_3), (B_1,B_6), (B_1,B_7), (B_2,B_3), (B_2,B_5), (B_2,B_6), (B_2,B_7), (B_3,B_4), (B_3,B_5), (B_3,B_6), (B_4,B_6), (B_4,B_7), (B_4,B_8), (B_5,B_6), (B_5,B_8), (B_6,B_7), (B_6,B_8)\}$ obliczyć maksymalne klasy zgodności.

ZADANIE 1.4

Dla podziałów Π_1, Π_2, Π_3 określonych na zbiorze $\{a, b, c, d, e, f, g, h, i, j, k\}$:

$$\Pi_1 = (\overline{a,b,c};\overline{d,e};\overline{g,h,\imath};\overline{\jmath,k})$$

$$\Pi_2 = (\overline{a,b};\overline{c,g,h};\overline{d,e,f};\overline{\imath,\jmath,k})$$

$$\Pi_3 = (\overline{a,b,c,f};\overline{d,e};\overline{g,h,\imath,\jmath,k})$$

należy wyznaczyć:

a) $\Pi_1 + \Pi_2$ oraz $\Pi_1 \bullet \Pi_2$,

b) $\Pi_1 + \Pi_3$ oraz $\Pi_1 \bullet \Pi_3$.

Bibliografia

[1.1] Borowik G., Łuba T.: Fast Algorithm of Attribute Reduction Based on the complementation of Boolean Function, ch. 2, pp. 25-41, Springer International Publishing, 2014.

[1.2] Borowik, G., Łuba, T., Tomaszewicz, P.: On the Memory Capacity to Implement Logic Functions, Eds. F. Pichler, R. Moreno-Díaz, A. Quesada-Arencibia, Computer Aided Systems Theory – EUROCAST 2011, vol.6928: Springer-Verlag Berlin Heidelberg, Lecture Notes in Computer Science, 343-350, 2012.

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

[1.4] Brzozowski, J. A., Łuba, T.: Decomposition of Boolean Functions Specified by Cubes, in Journal of Multiple-Valued Logic and Soft Computing, Vol. 9, Old City Publishing Inc., Philadelphia, 377-417, 2003.

[1.5] Goodman D.J., Carey M.J.: Nine Digital Filters for Decimation and Interpolation, IEEE Trans. On Acoustics, Speech and Signal Processing 25(2), pp.121-126, 1977.

[1.6] Grzymala-Busse J. W.: *Data with Missing Attribute Values: Generalization of Indiscernibility Relation and Rule Induction*. In: Peters J.F. et al. (eds.): Transactions on Rough Sets, LNCS 3100, pp. 78–95, Springer-Verlag, Berlin, 2004.

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

[1.8] 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.

[1.9] Łuba T., Selvaraj H.: *A General Approach to Boolean Function Decomposition and its Applications in FPGA-based Synthesis*. VLSI Design, Special Issue on Decompositions in VLSI Design, Vol. 3, Nos. 3-4, pp. 289-300, 1995.

[1.10] Łuba T., Rybnik J.: Algorithmic Approach to Discernibility Function with Respect to Attributes and Objects Reduction, Foundations of Computing and Decision Sciences, Vol. 18, No. 3-4, 241–258, 1993.

[1.11] Łuba T., Poźniak K., Zbierzchowski B.: *Redukcja i kompresja zmiennych w syntezie funkcji generowania indeksów.* Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne, Nr. 10, 2016.

[1.12] Mańkowski, M., Łuba, T., Jankowski, C.: Evaluation of Decision Table Decomposition Using Dynamic Programming Classifiers, The 24th International Workshop on Concurrency, Specification and Programming, Sept. 2015.

[1.13] Meyer-Baese U.: Digital Signal Processing with Field Programmable Gate Arrays. Springer Verlag, Second Edition, Berlin, 2004.

[1.14] Rokach L., Maimon O.: *Data Mining using Decomposition Methods*. In Oded Maimon O., Lior Rokach R., Data Mining and Knowledge Discovery Handbook, pp. 981-998, Springer, New York 2010.

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

[1.16] Sasao, T.: A Reduction Method For the Number of Variables to Represent Index Generation Functions: s-Min Method, IEEE 45th International Symposium on Multiple-Valued Logic, 164–169, 2015.

[1.17] Sasao, T.: Memory-Based Logic Synthesis, Springer New York Dordrecht Heidelberg London, 2011.

[1.18] Sysło M. M., Deo N., Kowalik J. S.: *Algorytmy optymalizacji dyskretnej.* PWN, 1993.