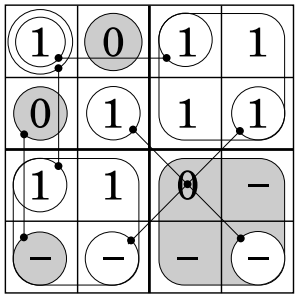




Tablice logiczne,
czyli kwadratura wielu kół

… diagram, pusty u góry i na końcu,
patrzył na nią ze swoją zimną logiką.[1]

(Robert L. Forward: *Dragon's Egg*, 1980)

Diagramy Venna, najpopularniejsze z diagramów logicznych w pierwszym okresie ich rozwoju, i ich proste odmiany[2] były przydatne do reprezentacji problemów i wnioskowań dla niewielkiej liczby zmiennych, zwykle najwyżej trzech. Klasyczny diagram Venna dla trzech zmiennych jest najbardziej znaną formą tego diagramu, a wiele osób wręcz uważa, że to jest jedyny diagram Venna. Diagramy Venna dla większej liczby zmiennych, których wymyślono z czasem wiele rodzajów,[3] bywają estetycznie atrakcyjne, ale do rozumowań nie najlepiej się nadają – są trudne do narysowania bez specjalistycznych narzędzi, układ kombinacji zbiorów jest w nich zawiły i nieczytelny – bardzo trudno nieraz określić, do których z n zbiorów należy dany obszar elementarny. Różne kształty i rozmiary obszarów elementarnych wprowadzają dodatkowy „szum informacyjny". Na szczęście (czy też nieszczęście), przez wiele dziesięcioleci od powstania diagramów Venna nie było potrzeby rozwiązywania zadań logicznych z większą liczbą zmiennych, a do elementarnych wnioskowań typu sylogistycznego czy dowodzenia tautologii logicznych z powodzeniem wystarczały trzy zmienne.

Sytuacja uległa zasadniczej zmianie około połowy zeszłego wieku, kiedy pojawiły się pierwsze cyfrowe układy elektroniczne, a w końcu oparte na tej technice cyfrowe układy sterujące i komputery. Okazało się nagle, że działanie takich układów, w których przetwarzane informacje najprościej było kodować dwoma stanami układu (np. „płynie prąd" – „nie płynie prąd"), znakomicie da się opisywać formułami algebry logiki Boole'a, w której też występują tylko dwie wartości – **prawda** i **fałsz**. Przypomniano sobie wtedy o diagramach logicznych, jako możliwym narzędziu operowania wyrażeniami dla funkcji algebry logiki. Jak jednak wskazaliśmy wyżej, klasyczne diagramy Venna nie najlepiej nadawały się do tego celu. Znacznie wygodniejszy okazał się bardziej regularny sposób ich konstrukcji w postaci prostokątnej *tablicy logicznej*. Zanim omówimy te tablice, sposób ich użycia i zastosowania, musimy wprowadzić kilka dodatkowych pojęć i oznaczeń stosowanych w tej dziedzinie.

Algebra Boole'a. Kilkakrotnie już wspomniana[4] algebra logiki, zwana też algebrą Boole'a,[5] powstała w połowie XIX w. jako próba zapisu problemów (pierwotnie z teorii zbiorów) w postaci analogicznej do formuł algebraicznych, tak, aby wnioskowania dało się przeprowadzać w ten sam sposób, jak rachunki algebraiczne. Próba się w zasadzie udała, dając przy okazji początek nowej dziedzinie matematyki, zwanej algebrą abstrakcyjną. Jednak zbytnie upodobnienie przez Boole'a tej notacji do arytmetyki, jak np. zapis operacji negacji **nie** x jako odejmowania $(1 - x)$, spowodowało, że rachunki w tej notacji były zawiłe i długie, co początkowo hamowało akceptację tej algebry. Z czasem notację uproszczono i algebra Boole'a stała się podstawą nowoczesnej logiki matematycznej i niektórych innych dziedzin, jak teoria zbiorów, kombinatoryka, teoria informacji, czy metamatematyka.

Z arytmetycznej notacji Boole'a pozostały jednak wciąż pewne elementy, używane zwłaszcza w jej zastosowaniach technicznych, takich jak projektowanie cyfrowych układów elektronicznych. Są to liczbowe oznaczenia wartości logicznych **prawda** → 1; **fałsz** → 0 (pierwotnie 1 oznaczało zbiór uniwersalny, zaś 0 – zbiór pusty) i arytmetyczne oznaczenia i nazwy

podstawowych operatorów logicznych: **i** → · (*iloczyn logiczny*, symbol często opuszczany, jak przy mnożeniu arytmetycznym); **lub** → + (*suma logiczna*); Boole pierwotnie używał tu *dysjunkcji wykluczającej* (*nierównoważności*) zamiast *alternatywy* (*dysjunkcji włączającej*) i na koniec **nie** $x \to \bar{x}$ (*negacja*). Tablica prawdy[6] dla operatorów logicznych w tej notacji wygląda jak następuje:

x_1	x_2	$\bar{x}_1$	$\bar{x}_2$	$x_1 + x_2$	$x_1 x_2$	$x_1 \Rightarrow y$	$x_1 \equiv x_2$	$x_1 \not\equiv x_2$	$x_1 \mid x_2$	$x_1 \downarrow x_2$
0	0	1	1	0	0	1	1	0	1	1
0	1	1	0	1	0	1	0	1	1	0
1	0	0	1	1	0	0	0	1	1	0
1	1	0	0	1	1	1	1	0	0	0

Podstawowe prawa algebry Boole'a, oprócz definicji operatorów logicznych (o czym więcej poniżej) wyglądają jak następuje:

Prawa wyłączonego środka: $x + \bar{x} = 1$; $x\,\bar{x} = 0$.

Prawa pochłaniania: $x + x = x$; $x\,x = x$; $x + 1 = 1$; $x \cdot 1 = x$; $x + 0 = x$; $x \cdot 0 = 0$.

Prawa rozdzielności: $x\,(y + z) = x\,y + x\,z$; $x + y\,z = (x + y)\,(x + z)$.

Zauważmy istotnie różne własności operacji algebry Boole'a w stosunku do odpowiednich operacji arytmetycznych (np. trzy z praw pochłaniania i drugie prawo rozdzielności są w arytmetyce fałszywe).

Funkcje przełączające. Jeśli mamy pewien ustalony zbiór wartości, możemy definiować funkcje, których argumenty, jak również wartości, mogą przybierać wartości z tego zbioru. Dla dwuelementowego zbioru wartości logicznych {0, 1} możemy więc definiować *funkcje logiczne* postaci f: $\{0, 1\}^n \to \{0, 1\}$, t.j. funkcje o wartościach 0 lub 1 z n argumentami przyjmującymi także wartości 0 lub 1. W technice cyfrowej nazywane są one *funkcjami przełączającymi*, gdyż służą do opisu układów włączających lub wyłączających różne podzespoły, z wartościami 0 i 1 interpretowanymi jako wyłączenie (prąd nie płynie) lub włączenie (prąd płynie) odpowiednio, a nie jako wartości logiczne fałszu i prawdy.

Przykład praktyczny. W wielu urządzeniach technicznych i powszechnego użytku (cyfrowe zegarki elektroniczne, tablice informacyjne wind, panele sterowania radia, magnetowidu, kuchenki mikrofalowej, itp.) zachodzi potrzeba wyświetlania w możliwie tani i prosty sposób cyfr od 0 do 9. Najprostszy i najczęściej stosowany wyświetlacz do tego celu składa się z siedmiu sektorów w postaci kresek – odpowiednie zapalenie tych kresek powoduje wyświetlenie obrazu pożądanej cyfry, jak na rysunku:

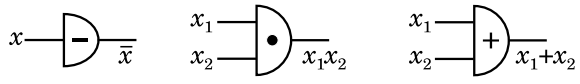
Potrzebny jest zatem układ, który dla każdej podanej cyfry ustali, które sektory mają zostać zapalone. Dla każdego sektora trzeba więc określić właściwą funkcję o wartościach 0 (nie zapalać sektora) lub 1 (zapalić sektor), której argumentem jest kod pożądanej cyfry. Jeśli na wejściu układu pojawiają się kody cyfr do wyświetlenia w postaci liczb binarnych (więc zerojedynkowych), to potrzebujemy siedmiu funkcji przełączających czterech zmiennych (do za-

kodowania cyfr 0 – 9 potrzeba czterech cyfr binarnych). Patrząc na rysunek powyżej, możemy łatwo napisać tablicę prawdy dla tych funkcji:

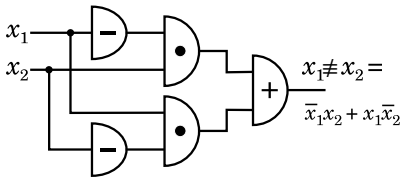
cyfra	x_1	x_2	x_3	x_4	s_1	s_2	s_3	s_4	s_5	s_6	s_7
0	0	0	0	0	1	1	1	0	1	1	1
1	0	0	0	1	0	0	1	0	0	1	0
2	0	0	1	0	1	0	1	1	1	0	1
3	0	0	1	1	1	0	1	1	0	1	1
4	0	1	0	0	0	1	1	1	0	1	0
5	0	1	0	1	1	1	0	1	0	1	1
6	0	1	1	0	1	1	0	1	1	1	1
7	0	1	1	1	1	0	1	0	0	1	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	0	0	0	0	0	0	0
11	1	0	1	1	-	-	-	-	-	-	-
12	1	1	0	0	-	-	-	-	-	-	-
13	1	1	0	1	-	-	-	-	-	-	-
14	1	1	1	0	-	-	-	-	-	-	-
15	1	1	1	1	-	-	-	-	-	-	-

Dla kodów binarnych odpowiadających liczbom spoza zakresu 0 – 9 wartości funkcji są w zasadzie nieokreślone – zakłada się, że takie kody nigdy się na wejściu nie pojawią. W praktycznym układzie jeden z nich można wykorzystać jako sygnał „nie wyświetlaj żadnej cyfry" – w tym przypadku wykorzystano do tego celu kod $10_X = 1010_{II}$. W praktyce bardzo często występują takie częściowo określone funkcje – ich wartości dla niektórych kombinacji wartości argumentów są nieokreślone lub nieistotne.

Funkcje przełączające (2). Z uwagi na skończoną liczbę możliwych wartości funkcji i argumentów, dla każdego n istnieje skończona liczba różnych funkcji przełączających, a mianowicie dokładnie 2^{2^n}. I tak np. dla dwu zmiennych mamy 16 różnych funkcji (ponumerowanych od 0 do 15 odpowiednio do ciągu wartości funkcji dla kolejnych kombinacji argumentów). Kilka z nich jest banalnych, jak np. funkcje stałe: $f_0(x_1, x_2) = 0$; $f_{15}(x_1, x_2) = 1$ i argumenty: $f_{12}(x_1, x_2) = x_1$; $f_{10}(x_1, x_2) = x_2$. Inne to np. podstawowe operacje logiczne: $f_8(x_1, x_2) = x_1 x_2$ lub $f_{14}(x_1, x_2) = x_1 + x_2$, itd., zob. tablicę prawdy powyżej. Funkcje te można opisywać za pomocą tablic prawdy lub diagramów Johnstona[7], lub też za pomocą formuł algebry Boole'a. Ten ostatni zapis ma tę zaletę, że zwykle bezpośrednio odpowiada strukturze układu cyfrowego realizującego daną funkcję.[8] W klasycznym systemie konstrukcji układów cyfrowych mamy do dyspozycji trzy standardowe układy (zwane *bramkami logicznymi*) realizujące negację, iloczyn logiczny (koniunkcję) i sumę logiczną (alternatywę):[9]



Jeśli przedstawimy np. *dysjunkcję wykluczającą* (*nierównoważność*) $x_1 \not\equiv x_2$ w postaci odpowiedniej formuły: $f_6(x_1, x_2) = \bar{x}_1 x_2 + x_1 \bar{x}_2$, to układ ją realizujący będzie wyglądał tak:



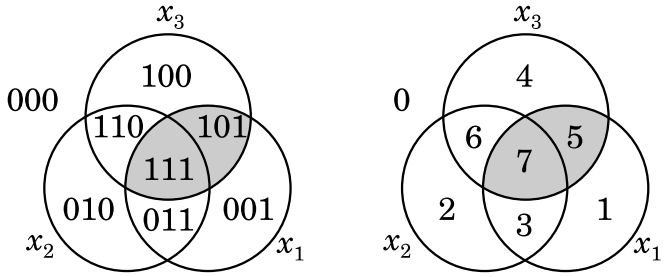
I tutaj dochodzimy do istoty problemu zastosowania tablic logicznych w projektowaniu układów cyfrowych. Otóż daną funkcję przełączającą, zwłaszcza bardziej złożoną (o większej liczbie zmiennych niż dwie), można przedstawić za pomocą wie-

lu różnych, ale równoważnych formuł logicznych, którym mogą odpowiadać układy o różnej złożoności (mierzonej np. liczbą potrzebnych bramek logicznych). Ponieważ każdy układ kosztuje (zwłaszcza w początkowym okresie rozwoju techniki cyfrowej ceny, a także i rozmiary bramek logicznych były dość spore), zachodzi potrzeba znajdowania dla funkcji logicznych wyrażeń o możliwie najmniejszej złożoności. Zagadnienie znajdowania takich wyrażeń, zwane problemem *minimalizacji funkcji przełączających*, okazało się trudne. Teoretycznie można to robić metodami algebraicznymi, przekształcając wyrażenie logiczne dla danej funkcji zgodnie z prawami algebry Boole'a, lecz metoda ta jest bardzo niepraktyczna ze względu na złożoność takich formuł, wielką liczbę potrzebnych przekształceń i brak jednoznacznych reguł określających, w jaki sposób należy przekształcać formułę, aby uległa ona uproszczeniu. Zwłaszcza w sytuacji, gdy często aby daną formułę bardziej uprościć, trzeba ją najpierw skomplikować…

Pokażmy to na elementarnym przykładzie. Weźmy funkcję „kreska Sheffera" $x_1 \mid x_2$ (w układach cyfrowych zwaną *bramką NAND*). Jak można odczytać z tablicy prawdy powyżej, tzw. kanoniczna postać alternatywna tej funkcji (co to jest i jak ją odczytać z tablicy – zob. poniżej) to $f_7(x_1, x_2) = \bar{x}_1\,\bar{x}_2 + \bar{x}_1\,x_2 + x_1\,\bar{x}_2$. Ta postać formuły wymaga do realizacji 6 bramek (2 negacje, 3 iloczyny i 1 sumę). Próbując ją uprościć, zastosujmy prawo rozdzielności do pierwszych dwóch składników, a potem prawa wyłączonego środka i pochłaniania: $f_7(x_1, x_2) = \bar{x}_1\,(\bar{x}_2 + x_2) + x_1\,\bar{x}_2 = \bar{x}_1 + x_1\,\bar{x}_2$. Uzyskana formuła jest prostsza, wymaga tylko 4 bramek (2 negacje, 1 iloczyn i 1 suma). Co więcej, nie da się jej dalej upraszczać, przynajmniej bezpośrednio. Albowiem, aby uzyskać jeszcze prostsze wyrażenie, w pierwszym kroku przekształcenia należy najpierw formułę bardziej skomplikować, używając jednego z praw pochłaniania (w drugą stronę): $f_7(x_1, x_2) = \bar{x}_1\,\bar{x}_2 + \bar{x}_1\,x_2 + \bar{x}_1\,\bar{x}_2 + x_1\,\bar{x}_2 = \bar{x}_1\,(\bar{x}_2 + x_2) + \bar{x}_2\,(\bar{x}_1 + x_1) = \bar{x}_1 + \bar{x}_2$. Ta formuła wymaga już tylko 3 bramek (2 negacje, 1 suma). Stosując do niej prawo de Morgana możemy dalej uzyskać postać $f_7(x_1, x_2) = \overline{x_1\,x_2}$, wymagającą tylko dwu bramek (iloczynu i negacji).

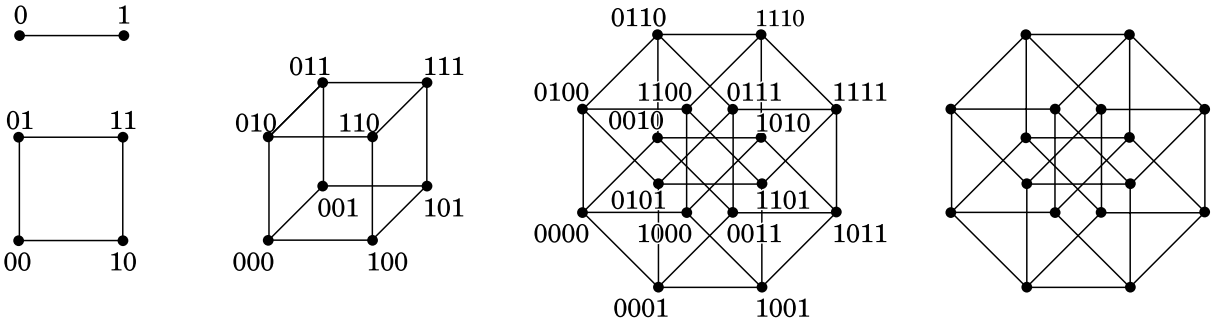
Okazało się jednak, że do tego celu można zaprząc odpowiedni diagram – tablicę logiczną.

Tablice logiczne. Każdy obszar elementarny na diagramie Venna odpowiada jednej kombinacji należenia i nienależenia punktów tego obszaru do każdego ze zbiorów tworzących diagram. W interpretacji zbiorów na diagramie Venna jako zmiennych logicznych (diagram Johnstona[10]) przynależność do zbioru oznacza, że dana zmienna logiczna ma wartość 1 (**prawda**), nienależenie do zbioru oznacza wartość 0 (**fałsz**) odpowiedniej zmiennej. Zaznaczając na diagramie Venna te wartości w odpowiednich obszarach elementarnych otrzymamy coś w rodzaju diagramowej tablicy prawdy, jak na przykładzie obok. Na drugim rysunku pokazane są wartości dziesiętne liczb binarnych opisujących obszary.

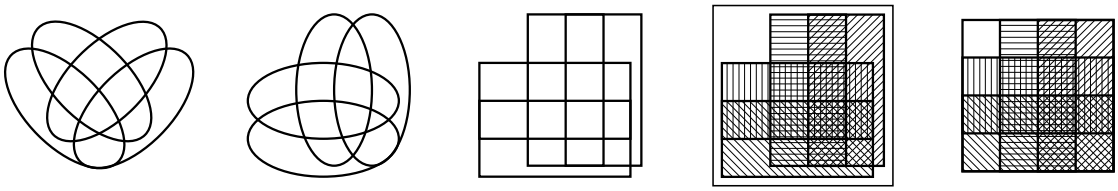


Zaznaczając (np. kolorem) obszary elementarne odpowiadające kombinacjom wartości zmiennych, dla których jakaś funkcja logiczna jest prawdziwa, otrzymamy jej obraz na diagramie Venna. Na przykład, jeśli zaznaczać będziemy te pola kolorem szarym, to diagram powyżej przedstawi nam funkcję trzech zmiennych $f_{160}(x_1, x_2, x_3) = x_1\,x_2\,x_3 + x_1\,\bar{x}_2\,x_3 = x_1\,x_3$. Jak już jednak wspomnieliśmy, działania na takich diagramach nie są wygodne, zwłaszcza dla większej liczby zmiennych, gdyż różne kształty i rozmiary pól i rosnące wraz z liczbą obszarów problemy z określeniem do jakich zbiorów dany obszar należy, a do jakich nie, mocno te

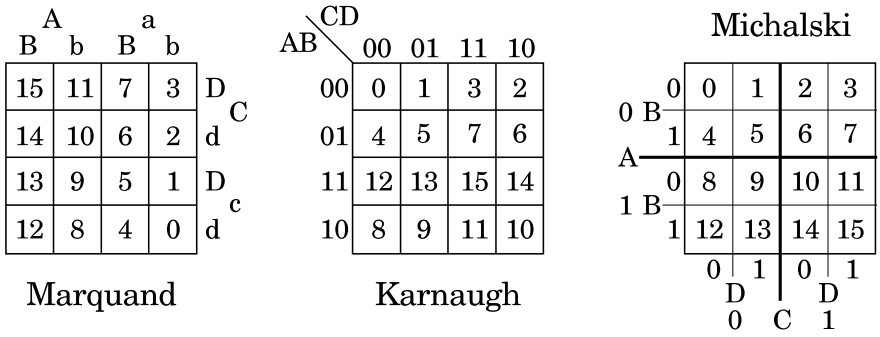
działania utrudniają. Należało więc znaleźć sposób bardziej regularnego rozmieszczenia tych 2^n kombinacji wartości zmiennych, niż na to pozwala klasyczny diagram Venna. Jedną z możliwości było wykorzystanie prostego i regularnego tworu geometrycznego – wielowymiarowej kostki o jednostkowej krawędzi. Jeśli jeden jej narożnik umieścić w początku układu współrzędnych (0, 0, …, 0), a przeciwległy narożnik w punkcie o wszystkich współrzędnych jedynkowych (1, 1, …, 1), to narożniki kostki będą miały współrzędne odpowiadające dokładnie wszystkim 2^n kombinacjom wartości zmiennych logicznych odpowiadających osiom układu współrzędnych. Konstrukcje dla n = 1, 2, 3, i 4 pokazuje rysunek:



Hiperkostka 4-wymiarowa pokazana jest także po prawej bez oznaczeń współrzędnych wierzchołków, aby unaocznić wyraźniej jej strukturę. Jak widać na rysunku, koncepcja ta (choć okazała się użyteczna teoretycznie), nie prowadzi do uzyskania diagramów wygodnych w użyciu. Należało poszukać sposobu regularnego „rozpłaszczenia" wielowymiarowej kostki na płaszczyźnie. Pomysł podsunęła struktura prostokątnego diagramu Carrolla.[11] Sposób uzyskania prostokątnej postaci diagramu (o nieco innej strukturze niż diagram Carrolla) przez stopniowe przekształcanie klasycznego diagramu Venna dla 4 zmiennych pokazuje rysunek. Różne kreskowanie prostokątów odpowiadających poszczególnym czterem zmiennym wprowadzono dla ich rozróżnienia w sytuacji pokrywania się części ich konturów.



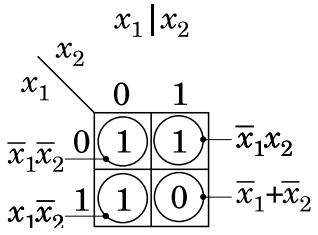
Prostokątną tablicę logiczną tego typu zaproponował w 1952 Veitch[12] (nazwano ją więc *kartą Veitcha*). Wkrótce potem Karnaugh[13] zmodyfikował ją w sposób ułatwiający jej zastosowanie do minimalizacji funkcji logicznych mniejszej liczby zmiennych (do 4). Jego tablice logiczne, zwane także *mapami Karnaugha*, były przez długi okres podstawowym narzędziem pracy projektantów układów przełączających. Jakiś czas potem okazało się zresztą, że tablicę logiczną (w formie jak u Veitcha) wynalazł dużo wcześniej amerykański logik Marquand.[14] Opisał ją już w 1881 r., a więc ledwie rok po publikacji artykułu Venna o jego diagramach, i to w tym samym angielskim czasopiśmie, co Venn swoje diagramy.[15] Tak jak to bywało z wieloma ważnymi innowacjami przed i po Marquandzie, pomysł Marquanda został całkowicie zapomniany i trzeba go było odkrywać na nowo po 70 latach. Podobne tablice proponowało na początku XX w. także dwóch innych autorów, ale też nie zostali zauważeni.[16] W latach 60-tych ubiegłego wieku inną modyfikację diagramu Marquanda-Veitcha, ułatwiającą jego używanie dla większej liczby zmiennych, zaproponował Michalski.[17] Ostatecznie najczęściej używanymi wersjami tablic logicznych stały się mapy Karnaugha i diagramy Michalskiego,[18] porównane z tablicami Marquanda na rysunku poniżej (dla n = 4):



Numery pól diagramów są dziesiątkowymi wersjami zero-jedynkowych kombinacji wartości odpowiednich zmiennych. Ich rozłożenie w diagramach pokazuje różnice struktury różnych typów tablic logicznych. Diagram Marquanda pokazano w formie, w jakiej Marquand użył go jako panelu odczytu wyników w swojej maszynie logicznej.[19] Marquand używał dużych liter do zmiennych, a małych liter do ich negacji. Diagram Michalskiego podano z pełnym opisem; zwykle używa się wersji uproszczonej, jak to robimy poniżej.

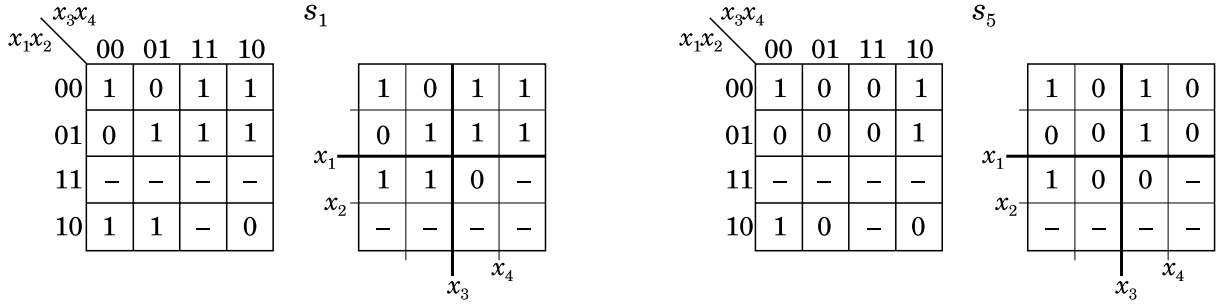
Minimalizacja funkcji przełączających. By móc używać tablic logicznych do minimalizacji funkcji logicznych, musimy umieć tłumaczyć formuły logiczne definiujące te funkcje na odpowiedni obraz na tablicy logicznej i odwrotnie. Do tego celu będzie nam potrzebne pojęcie *postaci normalnej* funkcji logicznej. Każdą funkcję logiczną można przekształcić do postaci sumy logicznej iloczynów logicznych zmiennych i ich negacji (*postać alternatywna*), a także do postaci iloczynu logicznego sum logicznych zmiennych i ich negacji (*postać koniunkcyjna*). Dana funkcja może mieć wiele równoważnych postaci normalnych, o różnej złożoności. Nasza przykładowa funkcja Sheffera ma m.in. takie dwie postacie alternatywne, jak podano wyżej: $f_7(x_1, x_2) = \bar{x}_1\,\bar{x}_2 + \bar{x}_1\,x_2 + x_1\,\bar{x}_2 = \bar{x}_1 + x_1\,\bar{x}_2$. Natomiast postać koniunkcyjną ma tylko jedną: $f_7(x_1, x_2) = \bar{x}_1 + \bar{x}_2$ (i jest ona jednocześnie jej jeszcze inną postacią alternatywną). Najbardziej złożone z postaci normalnych (osobno alternatywnych i koniunkcyjnych) nazwano *postaciami kanonicznymi* (odpowiednio alternatywną i koniunkcyjną). Odznaczają się one tym, że każdy iloczyn logiczny w kanonicznej postaci alternatywnej (lub suma logiczna w kanonicznej postaci koniunkcyjnej) zawiera wszystkie n zmiennych, niektóre ewentualnie zanegowane. Istnieje bezpośrednia odpowiedniość między postacią kanoniczną formuły funkcji a tablicą prawdy dla tej funkcji. Mianowicie każdemu wierszowi tablicy, w którym funkcja ma wartość 1 odpowiada jeden iloczyn jej kanonicznej alternatywnej postaci normalnej, a każdemu wierszowi z wartością 0 funkcji – jedna suma logiczna jej kanonicznej koniunkcyjnej postaci normalnej. To, które zmienne w tym iloczynie (lub sumie) są zanegowane, określa kombinacja wartości argumentów odpowiadająca temu wierszowi tablicy: w postaci alternatywnej negacji odpowiada wartość 0, a w postaci koniunkcyjnej – wartość 1. Tak to będzie wyglądało dla naszego przykładu funkcji Sheffera:

x_1	x_2	$x_1 \mid x_2$	iloczyny	sumy
0	0	1	$\bar{x}_1\,\bar{x}_2$	-
0	1	1	$\bar{x}_1\,x_2$	-
1	0	1	$x_1\,\bar{x}_2$	-
1	1	0	-	$\bar{x}_1 + \bar{x}_2$

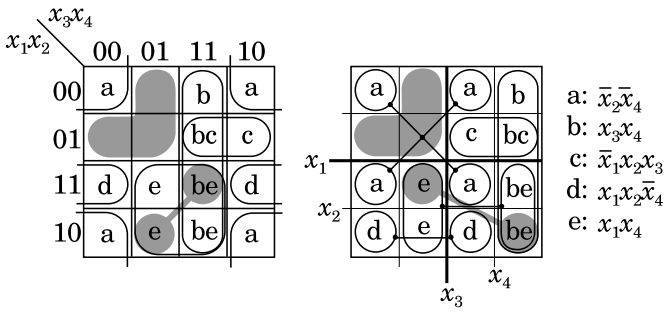


Ponieważ tablice logiczne są po prostu tablicami prawdy, w których rolę wierszy pełnią kratki tablicy, to samo można przedstawić na tablicy logicznej (Karnaugha), podanej obok tabeli.

Przykład praktyczny (2). Możemy teraz, mając pełną tablicę prawdy, przedstawić funkcje opisujące układ sterowania wyświetlacza. Oto, przykładowo, dwie z nich (s_1 i s_5; kreski „–” oznaczają nieokreślone wartości funkcji):

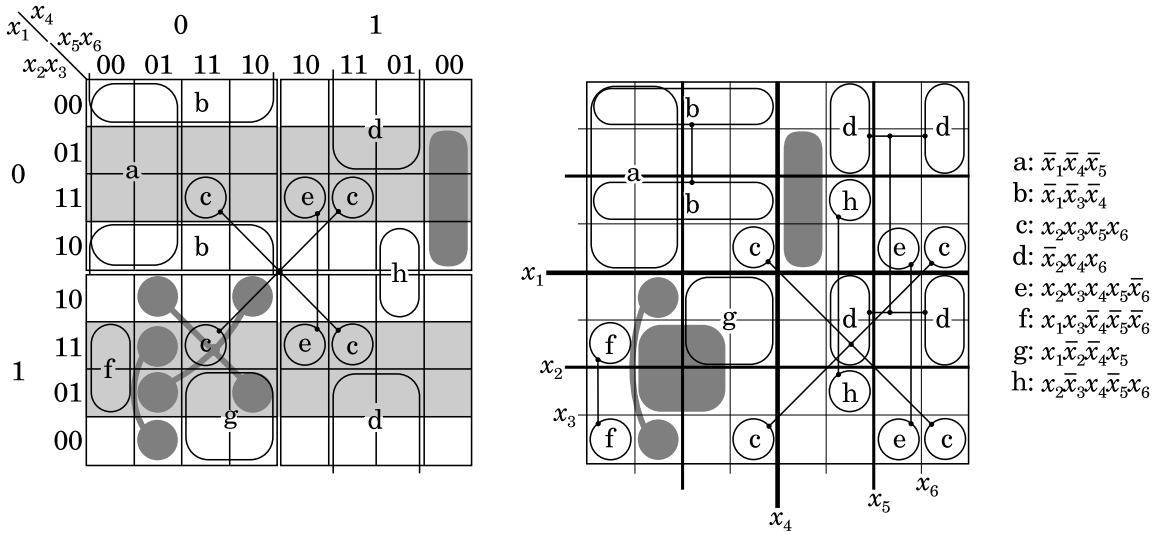


Minimalizacja funkcji przełączających (2). Pozostało nam jeszcze tylko jedno: jak przedstawiać na tablicy logicznej inne postacie normalne (nie kanoniczne) i jak znajdować na tablicy minimalna postać normalną. Okazuje się, że iloczynom zmiennych (prostych i zanegowanych) lub ich sumom odpowiadają pewne określone grupy kratek tablicy logicznej. Grupy te mają charakterystyczne kształty łatwe do rozpoznania na tablicy, a ich rozmiary (liczba kratek) nie są dowolne: jest to zawsze potęga dwójki – 2, 4, 8, 16, …) . Wygoda użycia mapy Karnaugha bierze się stąd, że na niej grupy te składają się zawsze z pól sąsiednich (przy czym przyjmuje się, że sąsiednie są także kratki na brzegu mapy umieszczone po przeciwległych jej stronach). Niestety, własność ta nie zachodzi dla map Karnaugha przy liczbie zmiennych większej niż 4. Dla innych tablic logicznych, np. dla diagramu Michalskiego, grupy niekoniecznie składają się z kratek sąsiednich, ale ich kształty także są na tyle charakterystyczne, że stosunkowo łatwo dają się rozpoznać. Dla porównania, przykłady grup odpowiadających kilku iloczynom (sumom) logicznym pokazano na rysunku na mapie Karnaugha i diagramie Michalskiego. Pokazano także (szarym kolorem) przykłady grup nieprawidłowych, którym nie odpowiadają żadne proste iloczyny czy sumy zmiennych. Algebraiczne formuły dla tych grup podano tu w formie koniunkcji, jakby to były jedynki funkcji.



Na diagramie Michalskiego różne grubości linii pomagają w rozpoznaniu właściwych grup. Staje się to szczególnie istotne dla większej liczby zmiennych, wtedy bowiem struktura mapy Karnaugha traci swoją prostotę. Mianowicie, część grup przestaje się składać z kratek sąsiednich, gdyż obszary odpowiadające niektórym zmiennym przestają być spójne. Np. mapa Karnaugha dla 6 zmiennych składa się z czterech map dla 4 zmiennych ułożonych jak 4 kratki mapy dla dwu zmiennych, przy czym niektóre z nich są jeszcze użyte w odbiciu zwierciadlanym, aby kratki dodatkowych dwóch zmiennych stały się sąsiednie. Mimo to obszary zmiennych x_3 i x_6 dalej pozostają niespójne, w rezultacie czego niektóre grupy zawierające te zmienne stają się również niespójne, podobnie, jak niektóre grupy na diagramie Michalskiego. Ta niejednolitość kształtów znacznie utrudnia używanie map Karnaugha dla większej liczby zmiennych, podczas gdy na diagramach Michalskiego diagramowe reguły tworzenia grup są zawsze takie same,[20] niezależnie od liczby zmiennych. Pokazuje to porównanie kształtu kilku przykładowych grup na obu tych diagramach dla 6 zmiennych poniżej. Kolorem ciemnoszarym zaznaczono przykłady grup nieprawidłowych, a jasnoszarym – roz-

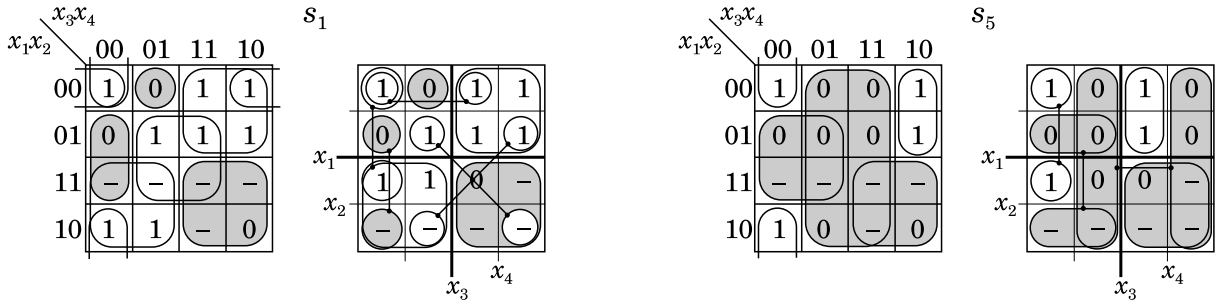
łączne obszary odpowiadające zmiennej x_3 na mapie Karnaugha. Dla zmiennej x_6 obszary takie mają ten sam kształt, tylko ustawione są pionowo.



Im większa grupa, tym krótsze (zawierające mniejszą liczbę zmiennych) wyrażenie (iloczyn lub suma) przez nią reprezentowane. Co za tym idzie, podstawowy sposób minimalizacji funkcji przełączających polega na tym, by znaleźć najmniejszy zestaw jak największych grup na tablicy logicznej, których suma (dla postaci alternatywnej) lub iloczyn (dla postaci koniunkcyjnej) jest poprawną postacią normalną danej funkcji. Dla postaci alternatywnej w grupy należy łączyć wyłącznie te kratki, w których funkcja ma wartość 1 (jedynki funkcji; grupy takie nazwano *prostymi implikantami* funkcji), a dla postaci koniunkcyjnej – kratki o wartości 0 (zera funkcji; grupy zer nazywają się tutaj *prostymi implicentami* funkcji). Jeśli implikanty pokryją wszystkie jedynki funkcji (i żadnego zera!), to ich suma równa się naszej funkcji. To samo zachodzi w postaci koniunkcyjnej – implicenty muszą pokryć wszystkie zera (i żadnej jedynki), a wtedy ich iloczyn daje nam naszą funkcję. Stąd procedura znalezienia takiej minimalnej postaci normalnej funkcji nazywa się często *problemem pokrycia*. Problemy tego typu występują także w wielu innych dziedzinach i są to w ogólnym przypadku problemy o dużej złożoności obliczeniowej (liczba potrzebnych operacji rożnie wykładniczo ze wzrostem liczby zmiennych n). Dla większej liczby zmiennych stosuje się zatem często algorytmy przybliżonej minimalizacji, dające rozwiązania znacznie mniejszym kosztem obliczeniowym, choć niekoniecznie ściśle minimalne. Jednym z lepszych algorytmów tego rodzaju jest klasyczny już algorytm *AQ* Michalskiego, który oprócz w przybliżeniu minimalnego pokrycia podaje także dosyć dokładne oszacowanie błędu (odległości od minimum).[21]

Oprócz przedstawionej minimalizacji postaci alternatywnej lub koniunkcyjnej, na tablicach logicznych można także minimalizować inne rodzaje układów przełączających, np. trójpoziomowe układy zbudowane za pomocą bramek NOR (negacja sumy, strzałka Peirce'a) lub NAND (negacja iloczynu, kreska Sheffera),[22] a także wykonywać różne inne operacje, jak np. analiza układów przełączających pod kątem możliwości wystąpienia w nich pewnych rodzajów błędów (np. tzw. „hazardu").

Przykład praktyczny (3). Wykonajmy więc minimalizację naszych przykładowych funkcji s_1 i s_5. Ich pokrycia minimalne pokazuje rysunek.



Przedstawiono zarówno postać alternatywną jak i koniunkcyjną (implicenty tej ostatniej zaznaczono szarym kolorem), na mapie Karnaugha i na diagramie Michalskiego. Fakt nieokreśloności funkcji pozwala często na większe uproszczenie wyrażenia dla tej funkcji, gdyż wybrane wartości nieokreślone możemy zaliczyć albo do zer, albo do jedynek funkcji, zależnie od tego, jak nam jest wygodniej przy tworzeniu grup komórek tablicy. Widać to dla funkcji s_1 powyżej – jej implikanty i implicenty zachodzą na siebie (oczywiście, mogą to robić bezkarnie tylko na polach z wartością nieokreśloną funkcji), a co za tym idzie postać alternatywna i koniunkcyjna przedstawiają różne funkcje ($s_{1a} \neq s_{1k}$). Uzyskane w ten sposób postacie funkcji są funkcjami określonymi – jednymi z wielu funkcji zgodnych z naszą pierwotną funkcją dla tych kombinacji wartości argumentów, dla których jest ona określona:

$$s_{1a}(x_1, x_2, x_3, x_4) = x_1\bar{x}_3 + x_2\,x_4 + \bar{x}_1\,x_3 + \bar{x}_2\,\bar{x}_3\,\bar{x}_4 \;\; (\text{lub} + \bar{x}_1\,\bar{x}_2\,\bar{x}_4);$$
$$s_{1k}(x_1, x_2, x_3, x_4) = (x_1 + x_2 + x_3 + \bar{x}_4)\,(\bar{x}_2 + x_3 + x_4)\,(\bar{x}_1 + \bar{x}_3).$$

Minimalna postać alternatywna s_{1a} ma dwa równoważne warianty różniące się ostatnim składnikiem. Pojawianie się wielu równoważnych wariantów wyrażenia funkcji zdarza się przy minimalizacji dosyć często. W tym przypadku, gdy minimalizujemy jednocześnie kilka (tu – siedem) funkcji tych samych argumentów, wybór wariantu podyktowany jest zwykle występowaniem takich samych składników w kilku funkcjach. Jeśli popatrzymy na funkcje s_5 (tutaj mamy $s_{5a} = s_{5k}$, gdyż implikanty i implicenty nie zachodzą na siebie):

$$s_{5a}(x_1, x_2, x_3, x_4) = \bar{x}_2\,\bar{x}_3\,\bar{x}_4 + \bar{x}_1\,x_3\,\bar{x}_4 =$$
$$s_{5k}(x_1, x_2, x_3, x_4) = \bar{x}_4\,(\bar{x}_2 + x_3)\,(\bar{x}_1 + \bar{x}_3),$$

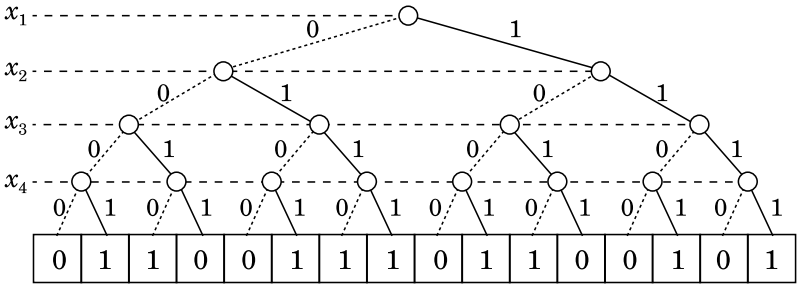
to zauważymy występowanie jednego ze składników pierwszego wariantu funkcji s_{1a} w wyrażeniu dla funkcji s_{5a}, zatem ten wariant funkcji s_{1a} bardziej opłaca się wybrać. W ten sposób jedna bramka realizująca iloczyn $\bar{x}_2\,\bar{x}_3\,\bar{x}_4$ będzie mogła zostać wykorzystana w obu funkcjach. Podobnie powtarza się suma $(\bar{x}_1 + \bar{x}_3)$, więc jej bramka też może być podwójnie wykorzystana.

Minimalizację pozostałych pięciu funkcji pozostawiamy jako ćwiczenie dla czytelnika. Ciekawy wynik uzyskamy dla funkcji s_6: postać alternatywna i koniunkcyjna są dla niej identyczne (jak dla funkcji Sheffera analizowanej na początku). Wyniki tej minimalizacji, zrealizowane łącznie jako jeden układ cyfrowy o siedmiu wyjściach, można zastosować do sterowania wyświetlaczem cyfr.

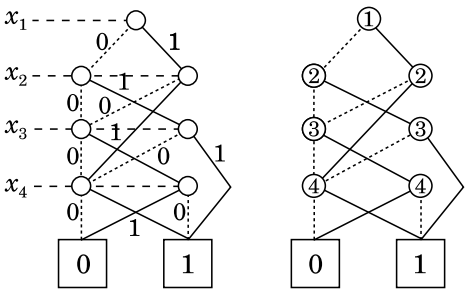
Diagram GLD. Michalski uogólnił swój diagram logiczny do postaci pozwalającej reprezentować funkcje o argumentach mających więcej niż dwie wartości.[23] Nosi on nazwę *diagramu GLD* (*Generalized Logic Diagram*) i znajduje zastosowania w rozpoznawaniu obrazów, maszynowym uczeniu się (gdzie służy do upraszczania opisów pojęć, których uczy się komputer), w wizualizacji wiedzy,[24] itp.

Drzewa decyzyjne. Tablice logiczne przedstawione powyżej to jeszcze nie ostatnie słowo w dziedzinie diagramów logicznych. W miarę rozwoju techniki cyfrowej i komputerowej w oparciu o układy scalone wielkiej skali integracji, minimalizacja liczby bramek logicznych przestała być kluczowym zagadnieniem, bo w układach tych mieszczą się z powodzeniem nawet setki tysięcy bramek, a ich jednostkowy koszt staje się niewielki. Ważniejsze stają się zagadnienia niezawodności, możliwości łatwego projektowania i realizacji procesu produkcji układu, budowa układów z regularnych struktur komórkowych pozwalających realizować dowolne funkcje za pomocą niewielkich modyfikacji połączeń, możliwość operowania funkcjami wielu zmiennych (złożoność tablic prawdy czy tablic logicznych rośnie wykładniczo względem liczby zmiennych), itp. Także coraz więcej układów zaczęto projektować komputerowo. Do tych celów tablice logiczne nie najlepiej się nadawały, dokonała się więc kolejna zmiana reprezentacji diagramowej: opracowano tzw. *drzewa decyzyjne.* W użyciu jest wiele rodzajów tych drzew. Nie będziemy ich tu szczegółowo omawiać z braku miejsca, przedstawimy tylko ogólną zasadę ich budowy.

Drzewa decyzyjne mają podobną historię, jak tablice Marquanda: zaproponował je Macfarlane już w roku 1885.[25] Zostały one potem zapomniane na dziesiątki lat, by zostać ponownie odkryte w 1959 roku,[26] ulepszone i szerzej rozpropagowane w drugiej połowie lat siedemdziesiątych ubiegłego wieku[27] i użyte dopiero zupełnie niedawno w komputerowym projektowaniu układów cyfrowych.[28] Zacznijmy od propozycji Macfarlane'a. „Wyprostował" on tablicę Marquanda do postaci liniowego ciągu kratek, natomiast opis przynależności kratek do odpowiednich zmiennych logicznych przedstawił w postaci drzewa, podobnie jak na rysunku poniżej dla czterech zmiennych:



To jest pełne drzewo dla przedstawionej funkcji. Ma ono złożoność wykładniczą względem liczby zmiennych (tak jak tablice prawdy czy zwykłe tablice logiczne). W praktyce stosuje się więc drzewa zredukowane (otrzymywane przez stosowanie dwóch prostych diagramowych reguł redukcji), jak podano obok dla tej samej funkcji, w dwu wariantach zapisu.



Przez odpowiedni dobór kolejności zmiennych w drzewie można w ten sposób uzyskać dla większości funkcji reprezentacje, które nie mają złożoności wykładniczej. Dalsze modyfikacje i ulepszenia dają jeszcze lepsze rezultaty.[29]

Kolejnym etapem rozwoju drzew decyzyjnych są najnowsze diagramy dla projektowania układów do obliczeń kwantowych.[30] Nie potrafimy co prawda jeszcze fizycznie budować komputerów kwantowych, ale metody ich projektowania i komputerowej symulacji są już nieźle zaawansowane.

Zenon Kulpa

[1] W oryginale "… the diagram, empty at the top and end, stared at her with its cold logic." Tłum. z ang. Zenon Kulpa.

[2] Zenon Kulpa: Koła Eulera i diagramy Venna, czyli jaka jest różnica między kanarkiem? *Tytuł Roboczy*, 2006.09/10 (015); Zenon Kulpa: Rozmaite diagramy logiczne, czyli idźcie i rozmnażajcie się. See `http://www.ippt.gov.pl/~zkulpa/diagrams/diagser.html`

[3] Zenon Kulpa: Koła Eulera i diagramy Venna, …, op. cit.

[4] Zenon Kulpa: Diagramy kontra predykaty, czyli jeśli nie możesz go pokonać, przyłącz się do niego. *Tytuł Roboczy*, 2006.07/08 (014); Zenon Kulpa: Rozmaite diagramy logiczne, …, op. cit.

[5] Jerzy [George] Boole (1815-1864), matematyk i filozof angielski, twórca *algebry logiki* (*algebry Boole'a*). Pierwszy wykład jego algebry ukazał się w połowie XIX w.: George Boole: *The Mathematical Analysis of Logic*. Barclay and Macmillan, Cambridge 1847.

[6] Patrz Zenon Kulpa: Rozmaite diagramy logiczne, …, op. cit.

[7] Op. cit.

[8] Pierwszy zauważył to w 1938 r. twórca teorii informacji Klaudiusz Shannon w swojej pracy doktorskiej opublikowanej jako: Claude E. Shannon: A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, **57**: 1938, pp. 713-723.

[9] Istnieje kilka systemów oznaczeń bramek logicznych. Tu dla ułatwienia czytelnikowi orientacji stosuję konwencję wpisywania w uniwersalną ikonę bramki symbolu realizowanej operacji logicznej używanego w tekście.

[10] Zenon Kulpa: Rozmaite diagramy logiczne, …, op. cit.

[11] Zenon Kulpa: Koła Eulera i diagramy Venna, …, op. cit. Diagram opisany był przez Lewisa Carrolla w *The Game of Logic*. Macmillan, London 1886.

[12] Edward W. Veitch: A chart method for simplifying truth functions. *Proc. of the 1952 ACM National Meeting* (Pittsburgh), 1952, pp. 127-133.

[13] Maurice Karnaugh: The map method for synthesis of combinational logic circuits, *Trans. AIEE,* pt I, **72**(9), November 1953, pp. 593-599.

[14] Allan W. Marquand (1853-1924), amerykański logik (uczeń Karola Peirce'a) oraz historyk sztuki. Skonstruował jedną z pierwszych mechanicznych „maszyn logicznych", opartą na strukturze tablicy logicznej swojego pomysłu. Jego nieortodoksyjne, matematyczne podejście do nauczania logiki (które zaczerpnął od Peirce'a) wywołało protesty władz uniwersytetu w Princeton, w rezultacie których w r. 1883 zmienił profesję i z logika matematycznego stał się poważanym historykiem sztuki i archeologiem.

[15] Allan Marquand: On logical diagrams for *n* terms. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **12**(75), 1881, pp. 266-270.

[16] William J. Newlin: A new logical diagram. *Journal of Philosophy, Psychology, and Scientific Methods,* **3**, Sept. 1906, p. 239; William E. Hocking: Two extensions of the use of graphs in elementary logic. *University of California Publications in Philosophy,* **2**(2), 1909, p. 31.

[17] Ryszard S. Michalski (1937-2007), polsko-amerykański informatyk pracujący w USA (wyjechał z Polski po doktoracie, w 1971 r.). Zajmował się algorytmami upraszczania funkcji logicznych, logicznymi reprezentacjami wiedzy i maszynowym uczeniem się. Twórca m.in. nowej wersji tablicy logicznej typu Marquanda-Veitcha oraz znanego algorytmu *AQ* przybliżonego rozwiązywania *problemu pokrycia*.

[18] Opis tego diagramu ukazał się po raz pierwszy w Polsce w: Ryszard S. Michalski: *Graficzna minimalizacja w klasie alternatywnych normalnych wyrażeń funkcji logicznych na podstawie tablic typu tablic Veitcha-Karnaugha*. Prace Instytutu Automatyki PAN, vol. **52**, Warszawa 1967; a na arenie międzynarodowej w: Ryszard S. Michalski: Recognition of total or partial symmetry in a completely or incompletely specified switching function. In: *Proc. IV Congress IFAC (Finite Automata and Switching Systems)*. Warsaw, June 16-21, 1969. Vol. **27**, pp. 109-129.

[19] Maszynę tę Marquand ukończył w 1882 r. Jej opis opublikował w Allan Marquand: A Machine for Producing Syllogistic Variation. In Charles Peirce, ed.: *Studies in Logic by Members of the Johns Hopkins University*. Little, Brown, and Company, Boston 1883; oraz w Allan Marquand: A New Logical Machine. *Proceedings of the American Academy of Arts and Sciences,* **21,** 1885, pp. 303-307. Maszyna zachowała się do naszych czasów.

Opis jej użycia wraz ze zdjęciem można znaleźć w Martin Gardner: *Logic Machines and Diagrams*. University of Chicago Press, Chicago 1982 [2nd ed.].

[20] Formalne zasady tworzenia właściwych grup opisał Michalski w: Ryszard S. Michalski: *Graficzna minimalizacja…, op. cit., oraz* Ryszard S. Michalski: *Synteza wyrażeń minimalnych i rozpoznawanie symetrii funkcji logicznych*. Prace Instytutu Automatyki PAN, vol. **92**, Warszawa 1971.

[21] Michalski opisał go po raz pierwszy w swojej pracy doktorskiej, wydanej jako: Ryszard S. Michalski: *Synteza wyrażeń minimalnych*…, op. cit.

[22] Zenon Kulpa: Synteza quasi-minimalnych układów logicznych wielu zmiennych z elementów NAND lub NOR. Praca magisterska, Wydział Elektroniki Politechniki Warszawskiej, Warszawa 1970; Ryszard S. Michalski, Zenon Kulpa: A system of programs for the synthesis of switching circuits using the method of disjoint stars. In: C.V. Freiman, ed.: *Information Processing 71* (Proc. IFIP Congress, Ljubljana 1971). North-Holland, Amsterdam 1972. Vol. **1**: pp. 61-65.

[23] Ryszard S. Michalski: *A geometric model for the synthesis of interval covers*. Report No. 461, Departament of Computer Science, Univeristy of Illinois, Urbana, IL 1971; Ryszard S. Michalski: A variable-valued logic system as applied to picture description and recognition. In: F. Nake, A. Rosenfeld, eds.: *Graphic Languages*. North-Holland Publishing Co., Amsterdam 1972.

[24] Zob. np. Ryszard S. Michalski: A variable-valued logic system as applied …, op. cit.; Janusz Wnęk, Ryszard S. Michalski: Comparing symbolic and subsymbolic learning: three studies. In: R.S. Michalski, G. Tecuci, eds.: *Machine Learning: A Multistrategy Approach*. Vol. IV, Morgan Kaufmann, San Mateo, CA 1993; Bartłomiej Śnieżyński, Robert Szymacha, Ryszard S. Michalski: Knowledge Visualization Using Optimized General Logic Diagrams. In: M.A. Klopotek, S.T. Wierzchon, K.Trojanowski, eds.: *Intelligent Information Processing and Web Mining*, (Proc. International IIS: IIPWM'05 Conference, Gdańsk, Poland, June 13-16, 2005. Springer-Verlag, Berlin 2005, pp. 137-146.

[25] Alexander Macfarlane: The logical spectrum. *Philosophical Magazine*, **19**, 1885, p. 286. Demonstracja użycia tych drzew w logice: Alexander Macfarlane: Adaptation of the method of the logical spectrum to Boole's problem. *Proceedings of the American Association for the Advancement of Science*, **39**, 1890, p. 57.

[26] C.Y. Lee: Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, **38**, 1959, pp. 985-999.

[27] Sheldon B. Akers: Binary decision diagrams. *IEEE Transactions on Computers*, **C-27(** 6), 1978, pp. 509-516.

[28] Randal E. Bryant: Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, **C-35**(8), 1986, pp. 677-691.

[29] Randal E. Bryant: Graph-based algorithms …, op. cit.; Rolf Drechsler, Andisheh Sarabi, Michael Theobald, Bernd Becker, Marek A. Perkowski: Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In: *Proc. 31st Annual Design Automation Conference (DAC),* San Diego, CA, June 1994. ACM Press, New York 1994, pp. 415-419; Marek A. Perkowski, Małgorzata Chrzanowska-Jeske, Yang Xu: Lattice diagrams using Reed-Muller logic. In: *Proc. WG 10.5 Workshop on Applications of the. Reed-Muller Expansion in Circuit Design (RM'97),* Oxford, UK, September 1997, pp. 85-102.

[30] George F. Viamontes, Manoj Rajagopalan, Igor L. Markov, John P. Hayes: Gate-level simulation of quantum circuits. In: *Proc. of the Asia South Pacific Design Automation Conference*, January 2003, pp. 295-301.