ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2000
Seria: INFORMATYKA z. 38 Nr kol.1451

Urszula STAŃCZYK
Politechnika Śląska, Instytut Informatyki

# DODATNIE FUNKCJE BOOLOWSKIE

**Streszczenie**. Dodatnie funkcje boolowskie charakteryzują się tym, że wszystkie argumenty funkcji występujące w postaci sumy implikantów lub iloczynu implicentów są w postaci prostej (nie pojawiają się żadne negacje). Niniejsza praca zawiera podstawowe definicje niezbędne do rozpatrzenia niektórych właściwości funkcji, istniejące ograniczenia nakładane na liczbę funkcji i ich modyfikację oraz algorytm generowania takich funkcji.

# POSITIVE BOOLEAN FUNCTIONS

**Summary**. Positive Boolean functions are those, which in their conjunctive or disjunctive form have all arguments positive. This paper contains some basic definitions necessary for dealing with functions' properties, the existing bounds on the number of positive Boolean functions and their modifications as well as the algorithm for generating these functions.

## 1. Wprowadzenie

Algebra Boole'a, której rozwój został zapoczątkowany przez angielskiego matematyka George'a Boole'a w 1854[1], odgrywa obecnie dużą rolę w matematyce, gdyż niektóre ważne systemy (np. szeroko stosowana algebra zbiorów) należą właśnie do algebry Boole'a. Szczególne znaczenie ma algebra dwuelementowa, będąca podstawą teorii układów przełączających i ich projektowania.

Ponieważ z funkcjami i logiką boolowską mamy ponadto do czynienia w tak popularnych dziedzinach, jak maszyny liczące, komunikacja cyfrowa, systemy sterowania itd., doczekały się

[1] George Boole, „An Investigation of the Laws of Thought".

one wielu znaczących publikacji, jednakże jak do tej pory niewiele uwagi poświęcono pewnej grupie funkcji logicznych, nazywanych dodatnimi funkcjami boolowskimi (*ang. Positive Boolean Functions*), których szczególne właściwości znalazły zastosowanie np. w przetwarzaniu obrazów przy filtracji stosowej [7]. W filtracji tego rodzaju funkcje te odpowiadają wprost filtrom operującym na binarnych ciągach danych wejściowych.

Niniejsza praca przedstawia zbiór podstawowych pojęć i definicji, rozważania teoretyczne pozwalające na określenie rzędu wielkości liczby dodatnich funkcji boolowskich (DFB) dla danej liczby zmiennych wejściowych, wreszcie algorytm generowania tych funkcji.

## 2. Podstawowe pojęcia i definicje

Przechodząc do rozważań dotyczących DFB i ich właściwości, musimy najpierw podać definicje pewnych podstawowych pojęć, którymi będziemy się posługiwać.

Zacznijmy od przyjęcia oznaczenia $f(x_1, x_2, \ldots, x_n)$ dla funkcji boolowskiej operującej na $n$ zmiennych wejściowych.

**Definicja 2.1**

Literałem nazywamy zmienną lub jej odwrotność (negację).

**Przykład 2.1**

Przykładowe literały to: $a$, $\bar{a}$, $\overline{x_1}$, $x_3$ itp.

**Definicja 2.2**

Implikantem funkcji przełączającej $f$ nazywamy taki iloczyn logiczny (koniunkcję) $g(x_1, x_2, \ldots, x_n)$ literałów, z których każdy występuje co najwyżej raz, że dla wszystkich możliwych kombinacji wartości argumentów, dla których $g(x_1, x_2, \ldots, x_n) = 1$ także i $f(x_1, x_2, \ldots, x_n) = 1$.

Odwołując się do terminologii logiki i rachunku zdań, implikantem funkcji $f$ jest taki iloczyn logiczny $g$ literałów, który implikuje funkcję $f$ (jeżeli $g = 1$, to $f = 1$).

Implicentem funkcji przełączającej $f$ jest taka suma logiczna $g(x_1, x_2, \ldots, x_n)$ (alternatywa lub dysjunkcja) literałów, z których każdy występuje co najwyżej raz, że dla wszystkich możliwych kombinacji wartości argumentów, dla których $g(x_1, x_2, \ldots, x_n) = 0$, także $f(x_1, x_2, \ldots, x_n) = 0$.

**Przykład 2.2**

Dla funkcji $F = \Sigma(0,2,6,7,8,10,14,15)_{abcd}$ przykładowe implikanty to: $b \cdot c$, $\overline{b} \cdot \overline{d}$, zaś przykładowe implicenty to: $\overline{b} + c$, $b + \overline{d}$.

**Definicja 2.3**

Implikant (implicent) $g(x_1, x_2, ..., x_n)$ jest pokrywany przez inny implikant (implicent) $h(x_1, x_2, ..., x_n)$, jeżeli wszystkie literały $h$ są literałami $g$.

**Przykład 2.3**

Implikant $g_1 = x_1 \cdot \overline{x_2} \cdot x_3$ jest pokrywany przez $h_1 = x_1 \cdot \overline{x_2}$, a implicent $g_2 = x_1 + \overline{x_2} + x_3$ jest pokrywany przez $h_2 = x_1 + \overline{x_2}$.

**Definicja 2.4**

Elementarnym implikantem (implicentem) funkcji $f$ jest taki implikant (implicent), który jest iloczynem logicznym (sumą logiczną) wszystkich argumentów funkcji.

Inaczej, elementarnym implikantem (implicentem) funkcji f jest taki implikant (implicent), który nie pokrywa żadnego innego implikanta (implicenta) funkcji.

Implikanty (implicenty) elementarne nazywane są również pełnymi iloczynami (pełnymi sumami), składnikami jedynki (czynnikami zera) lub mintermami (maxtermami).

**Przykład 2.4**

Przykładowy elementarny implikant funkcji $f$ to $g = x_1 \cdot \overline{x_2} \cdot x_3 ... x_n$. Przykładowy elementarny implicent to $k = x_1 + \overline{x_2} + x_3 + \cdots + \overline{x_n}$.

**Definicja 2.5**

Suma logiczna implikantów funkcji nazywana jest normalną postacią sumy, podczas gdy iloczyn logiczny implicentów funkcji nazywany jest normalną postacią iloczynu.

**Przykład 2.5**

Przykładowa normalna postać sumy funkcji to $f = x_1 \cdot x_3 \cdot x_5 + \overline{x_2} \cdot x_4$. Przykładowa normalna postać iloczynu funkcji to $f = (x_1 + \overline{x_2} + \overline{x_5})(\overline{x_3} + x_4 + \overline{x_1})$.

**Definicja 2.6**

Kanoniczną postacią sumy (iloczynu) nazywamy taką normalną postać sumy (iloczynu), w której wszystkie implikanty (implicenty) wchodzące w jej skład są elementarne.

**Przykład 2.6**

Przykładowa kanoniczna postać sumy to $f = x_1 \cdot \overline{x_2} \ldots x_n + \overline{x_1} \cdot x_2 \ldots \overline{x_n}$. Przykładowa kanoniczna postać iloczynu to $f = (x_1 + \overline{x_2} + \cdots + \overline{x_n})(\overline{x_1} + x_2 + \cdots + x_n)$.

**Definicja 2.7**

Postać normalna sumy (iloczynu), w której nie wszystkie implikanty (implicenty) są elementarne lub ich liczba jest mniejsza niż w postaci kanonicznej funkcji, nazywana jest postacią skróconą (sumy albo iloczynu).

**Definicja 2.8**

Implikanatmi (implicentami) prostymi nazywa się implikanty (implicenty), które pozbawione chociaż jednego argumentu przestają być implikanatmi (implicentami) funkcji.

**Definicja 2.9**

Postacią nieskracalną (nieredundancyjną) funkcji $f$ nazywa się taką skróconą postać sumy (iloczynu), w której usunięcie któregokolwiek z implikantów (implicentów) prostych pozostawia wyrażenie, które nie jest już odpowiednikiem oryginalnej funkcji $f$.

**Definicja 2.10**

Funkcję przełączającą $f$ nazywamy dodatnią funkcją boolowską, jeżeli we wszystkich implikantach (implicentach) wchodzących w skład nieskracalnej[1] postaci funkcji wszystkie literały odpowiadają postaciom prostym argumentów tej funkcji.

[1] Mowa o postaci nieskracalnej, gdyż tylko jedna funkcja $f$, $f = x_1 \cdot x_2 \ldots x_n$, jest jednocześnie w postaci kanonicznej i nieskracalnej dodatnią funkcją boolowską. Dla innych przypadków zakładamy, że będziemy je rozważać dopiero po usunięciu zbędnych implikantów (implicentów). Ponieważ zaś DFB są podzbiorem funkcji unitarnych (*ang. unate*), oznacza to, że każda z nich ma unikalną nieskracalną postać sumy (Muroga [2]).

## 3. Podstawowe właściwości dodatnich funkcji boolowskich

Jeśli chodzi o właściwości DFB, to warto zwrócić uwagę na fakt, że nie występuje dla nich problem hazardu. Zjawisko hazardu, będące przyczyną chwilowych przekłamań na wyjściu funkcji, jest rezultatem opóźnień spowodowanych czasem propagacji elementów i występuje, gdy sygnał i jego negacja (która z definicji nie może wystąpić dla dodatnich funkcji boolowskich) sterują jednocześnie jakimś elementem przechodząc poprzez układ kombinacyjny (patrz [8]).

DFB należą do klasy funkcji logicznych zawierających jedynkę, czyli dla wartości argumentów $(x_1, x_2, \ldots, x_n) = (1,1,\ldots,1)$ funkcja $f(x_1, x_2, \ldots, x_n) = f(1,1,\ldots,1) = 1$, co wynika z samej definicji DFB. Jeżeli przyjmiemy do rozważań postać sumy, to dowolna DFB przedstawia się jako alternatywa iloczynów, w których wszystkie argumenty mają postać prostą. Stąd, jeżeli wszystkie argumenty są równe jeden, to i wszystkie iloczyny muszą być równe jeden, a co za tym idzie i funkcja (będąc wówczas sumą jedynek logicznych) jest równa jeden.

DFB należą również do klasy funkcji monotonicznych, czyli takich, w których przy zwiększaniu stanu argumentów[1] wartość funkcji nie zmniejsza się. Właściwość ta, nazwana „układaniem w stos", została udowodniona przez E.N. Gilberta (patrz [1]) i jest wykorzystywana w filtracji stosowej, która jest jedną z metod przetwarzania obrazów (patrz [7]).

## 4. Określenie liczby DFB

### 4.1. Ograniczenia na liczbę DFB

Jeżeli utożsamimy dowolną funkcję przełączającą z odpowiadającą jej siatką Karnaugha, to dla funkcji $n$ zmiennych rozmiar siatki da się określić jako wariację z powtórzeniami, gdzie liczba przyjmowanych wartości jest równa 2 (bo każda zmienna może przyjmować wartość zero albo jeden), a liczba powtórzeń to liczba zmiennych, czyli $n$. Stąd rozmiar siatki to $2^n$.

Z kolei liczba wszystkich funkcji boolowskich znów jest wariacją z powtórzeniami, gdyż każda z $2^n$ kratek siatki może przyjmować wartość zero albo jeden. Dlatego liczba wszystkich

[1] Dla szczegółowego rozpatrzenia monotoniczności funkcji przełączających konieczne jest podanie definicji relacji większości, mniejszości itd. argumentów funkcji, a ponieważ nie jest to zasadniczym tematem niniejszej pracy, zainteresowanych odsyłamy do literatury - pozycje [3], [6], [7] prezentują zarówno definicję,jak i przykłady.

funkcji przełączających dla $n$ zmiennych to $2^{2^n}$. Z tej ogromnej liczby tylko niewielka część stanowi zbiór dodatnich funkcji boolowskich.

Ograniczenia na liczbę DBF jako pierwszy zaproponował Dedekind [10], a następnie zostały one opracowane przez E.N. Gilberta i to jemu zawdzięczamy poniższe twierdzenie, którego pełny dowód zawarty jest w [1].

**Lemat 4.1.1 [1]**

Liczba $\varphi(n)$ DBF dla $n$ zmiennych spełnia nierówności:

$$2^{C_{n,n/2}} \le \varphi(n) \le n^{C_{n,n/2}} + 2 \qquad (1)$$

gdzie $C_{n,n/2}$ [1] odpowiada współczynnikowi dwumiennemu, nazywanemu symbolem Newtona i jest równe

$$C_{n,n/2} = \binom{n}{n/2} = \frac{n!}{(n/2)!(n-n/2)!}$$

Powyższe granice już jednak przez samego Gilberta zostały uznane za zbyt szerokie, a ich modyfikacji, podanej w poniższym twierdzeniu, dokonali Baugh [4] (dolna granica) i Hensel [5] (górna granica).

**Lemat 4.1.2**

Liczba $\varphi(n)$ DBF dla $n$ zmiennych spełnia nierówności:

$$1 + \sum_{i=1}^{n-1} \{2^{C_{n,i}} - (n+1)\} \le \varphi(n) \le 3^{C_{n,n/2}} \qquad (2)$$

Wydaje się jednak, że także to ograniczenie, a przynajmniej dolną granicę, można jeszcze poprawić, czym zajmiemy się w następnym punkcie tej pracy.

### 4.2. Uściślenie dolnego ograniczenia liczby DBF

Spójrzmy na DBF z punktu widzenia ich generowania. Najprościej jest przeprowadzić rozważania na przykładzie z małą liczbą zmiennych, gdyż wtedy można wprost wypisać wszystkie rozwiązania, a potem pozostaje tylko uogólnić wzory. Rozpatrzmy więc przypadek funkcji 3 zmiennych i załóżmy stosowanie postaci sumy DBF.

---

[1] Ponieważ operujemy tutaj tylko na liczbach naturalnych, oznaczenie $n/2$ może mylnie sugerować pojawienie się liczby rzeczywistej dla nieparzystego $n$. Szczegółowe rozpatrzenie tego przypadku znajduje się w dalszej części tej pracy.

Każda DFB, oprócz tożsamościowego zera i tożsamościowej jedynki (opisanych później), przedstawia się jako suma $m$ implikantów prostych, gdzie $m = 1,2,\ldots k_{max}$, zanim więc zaczniemy generować funkcje, stwórzmy zbiór wszystkich implikantów prostych. Implikanty proste będą iloczynami od 1- do $n$-argumentowymi, jak następuje:

$x_1$
$x_2$ implikanty 1-argumentowe
$x_3$

$x_1x_2$
$x_1x_3$ implikanty 2-argumentowe
$x_2x_3$

$x_1x_2x_3$ ] implikant 3-argumentowy

podane w porządku[1] grupującym je pod względem liczby argumentów funkcji wchodzących w ich skład.

Następnym etapem jest określenie liczby implikantów w ramach każdej podgrupy, co jest bardzo proste przy korzystaniu z terminologii kombinatoryki:

• liczba implikantów 1-argumentowych odpowiada kombinacji 1-elementowej z $n$ $(n = 3)$ elementów, czyli

$$l_1 = \binom{n}{1} = \binom{3}{1} = n = 3$$

• liczba implikantów 2-argumentowych odpowiada kombinacji 2-elementowej z $n$ $(n = 3)$ elementów, czyli

$$l_2 = \binom{n}{2} = \binom{3}{2} = \frac{n!}{2!(n-2)!} = 3$$

• liczba implikantów 3-argumentowych odpowiada kombinacji 3-elementowej z $n$ $(n = 3)$ elementów, czyli

$$l_3 = \binom{n}{3} = \binom{3}{3} = \binom{n}{n} = 1$$

Stąd liczba wszystkich implikantów prostych

$$L = \sum_{i=1}^{n} l_i = \sum_{i=1}^{3} l_i = 7$$

[1] Algorytm generowania DBF, opisany w następnym punkcie tej pracy, zapewnia inne uporządkowanie implikantów prostych, co zostanie odpowiednio uzasadnione przy opisie tego algorytmu.

Liczbę wszystkich implikantów prostych dla dowolnej funkcji o $n$ zmiennych da się również wyrazić poniższym ogólnym wzorem.

**Twierdzenie 4.2.1**

Liczba wszystkich implikantów prostych dla funkcji $n$ zmiennych wyraża się wzorem:

$$L = 2^n - 1 \tag{3}$$

**Dowód**

Przedstawmy liczby implikantów prostych w ramach podgrup jako:

$l_1 = \binom{n}{1} = C_{n,1}$ liczba implikantów 1-argumentowych

$l_2 = \binom{n}{2} = C_{n,2}$ liczba implikantów 2-argumentowych

$l_k = \binom{n}{k} = C_{n,k}$ liczba implikantów $k$-argumentowych

.

.

.

$l_n = \binom{n}{n} = C_{n,n}$ liczba implikantów $n$-argumentowych

Stąd

$$L = \sum_{i=1}^{n} l_i = \sum_{i=1}^{n} \binom{n}{i} = \sum_{i=1}^{n} C_{n,i}$$

Przypomnijmy teraz dwumian Newtona

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} b^i$$

Jeżeli założymy $a = b = 1$, wówczas otrzymamy

$$2^n = \sum_{i=0}^{n} \binom{n}{i} = \binom{n}{0} + \sum_{i=1}^{n} \binom{n}{i}$$

Ponieważ z definicji $\binom{n}{0} = 1$, stąd

$2^n = 1 + \sum_{i=1}^{n} \binom{n}{i} \Rightarrow \sum_{i=1}^{n} \binom{n}{i} = 2^n - 1$, co było do udowodnienia.

Wróćmy teraz do problemu generowania DFB jako sum implikantów prostych. Jak było to już powiedziane wcześniej, każda taka suma może zawierać $m$ implikantów, gdzie $m = 1,2,\ldots k_{\max}$. Naturalne jest tutaj pytanie, jaka jest maksymalna liczba implikantów (czyli $k_{\max}$), które możemy zsumować logicznie i wciąż jeszcze stworzyć unikalną DFB.

Zgodnie z twierdzeniem dowiedzionym przez Spencera[9], $k_{\max}$ wyraża się następująco:

**Lemat 4.2.2**

Maksymalna liczba implikantów prostych wykorzystanych do stworzenia unikalnej DFB

$$k_{\max} = C_{n,n/2} \tag{4}$$

gdzie, jak było powiedziane wcześniej, $C_{n,n/2} = \binom{n}{n/2} = \frac{n!}{(n/2)!(n-n/2)!}$

Zastanówmy się teraz nad znaczeniem liczby $n/2$. Jeżeli $n$ jest parzyste, to $n/2$ pozostaje liczbą naturalną, na jakich tutaj operujemy. Jeżeli jednak $n$ jest nieparzyste, to musimy rozpatrywać przypadek $(n-1)/2$ lub $(n+1)/2$. Dla obu tych sytuacji $k_{\max}$ pozostaje takie samo, gdyż $\binom{n}{i} = \binom{n}{n-i}$, stąd też

$$\binom{n}{(n-1)/2} = \binom{n}{n-(n-1)/2} = \binom{n}{(2n-n+1)/2} = \binom{n}{(n+1)/2}$$

W dalszych rozważaniach dotyczących określenia dolnej granicy liczby DFB przyjmiemy pewne upraszczające założenie. Otóż generowane przez nas funkcje będą sumami implikantów z tej samej podgrupy, czyli o takiej samej liczbie zmiennych. Przy tak przyjętym założeniu nie musimy sprawdzać, czy implikanty nie pokrywają się wzajemnie, gdyż pokrywanie nie może wystąpić w ramach jednej podgrupy, a tylko pomiędzy podgrupami, gdy implikanty mają różną liczbę argumentów.

Wtedy dla danej podgrupy $l_i$ implikantów prostych możemy generować funkcje będące sumami od 1 do $l_i$ iloczynów, co dla danego $n$ będzie się wyrażać następująco:

Podgrupa $l_1 = \binom{n}{1} = C_{n,1} = n$ implikantów 1-argumentowych

Liczba funkcji o 1 implikancie 1-argumentowym $f_{1,1} = C_{l_1,1} = C_{n,1} = \binom{n}{1} = n$

Liczba funkcji o $j$ implikantach 1-argumentowych $f_{1,j} = C_{l_1,j} = C_{n,j} = \binom{l_1}{j} = \binom{n}{j}$

itd. aż do liczby funkcji o $l_1$ implikantach 1-argumentowych

$$f_{1,l_1} = C_{l_1,l_1} = C_{n,n} = \binom{l_1}{l_1} = \binom{n}{n} = 1$$

Stąd liczba wszystkich funkcji utworzonych z implikantów 1-argumentowych wyraża się następującym wzorem:

$$f_1 = \sum_{j=1}^{l_1} C_{l_1,j} = \sum_{j=1}^{l_1} \binom{l_1}{j} = 2^{l_1} - 1 = 2^n - 1$$

Podgrupa $l_k = \binom{n}{k} = C_{n,k}$ implikantów $k$-argumentowych

Liczba funkcji o 1 implikancie $k$-argumentowym $f_{k,1} = C_{l_k,1} = \binom{l_k}{1}$

Liczba funkcji o $j$ implikantach $k$-argumentowych $f_{k,j} = C_{l_k,j} = \binom{l_k}{j}$

itd. aż do liczby funkcji o $l_k$ implikantach $k$-argumentowych $f_{k,l_k} = C_{l_k,l_k} = \binom{l_k}{l_k} = 1$

Skąd liczba wszystkich funkcji utworzonych z implikantów $k$-argumentowych wyraża się następującym wzorem:

$$f_k = \sum_{j=1}^{l_k} C_{l_k,j} = \sum_{j=1}^{l_k} \binom{l_k}{j} = 2^{l_k} - 1 \tag{5}$$

Dlatego ogólnie możemy stwierdzić, że w ramach podgrupy liczba funkcji, które możemy stworzyć to 2 podniesione do potęgi odpowiadającej liczbie implikantów prostych w podgrupie i od tego należy odjąć 1.

Ponieważ podgrupa iloczynów wszystkich $n$ argumentów jest oczywiście 1-elementowa, więc w jej ramach możliwe jest stworzenie tylko jednej DFB (która w tym szczególnym przypadku, jak to już wskazaliśmy wcześniej, jest w postaci kanonicznej i końcowej jednocześnie, a implikant prosty jest jednocześnie implikantem elementarnym).

Szczególnymi przypadkami są również tożsamościowe zero i tożsamościowa jedynka, które zwyczajowo traktuje się jako funkcje o jednym implikancie. Te trzy przypadki uwzględnimy dodając liczbę 3 do końcowego wzoru określającego dolną granicę liczby DFB.

Na podstawie powyżej podanych wyprowadzeń otrzymujemy poniższe twierdzenie.

**Twierdzenie 4.2.3**

Liczba $\varphi(n)$ dodatnich funkcji boolowskich dla $n$ zmiennych spełnia warunek:

$$2\sum_{i=1}^{(n-1)/2}(2^{C_{n,i}}-1)+3<\varphi(n) \tag{6}$$

**Dowód**

Rozpatrzmy sumę liczb funkcji wygenerowanych w ramach każdej (z wyjątkiem ostatniej) podgrupy implikantów prostych $f'=\sum_{k=1}^{n-1} f_k$ gdzie $k$ wskazuje nam liczbę argumentów wchodzących w skład implikantów danej podgrupy. Jak pokazuje wzór (5), $f_k = 2^{l_k}-1$, gdzie $l_k$ jest liczbą implikantów w danej podgrupie. Stąd

$$f'=\sum_{k=1}^{n-1}(2^{l_k}-1) \tag{7}$$

Ponieważ $l_k = C_{n,k}$, uzyskujemy $f'=\sum_{k=1}^{n-1}(2^{C_{n,k}}-1)$. Przypominając zaś, że $C_{n,k}=C_{n,n-k}$ możemy zaobserwować we wzorze (7) symetrię, która sprowadzi nas do następującej modyfikacji: $f'=2\sum_{k=1}^{(n-1)/2}(2^{C_{n,k}}-1)$

Górna granica sumy w tym wzorze oczywiście wynika z przyjętej nieparzystości $n$[1]. Dla $n$ parzystego cały wzór należałoby zmodyfikować[2].

Po dodaniu do tej liczby wartości 3, odpowiadającej funkcji z 1 implikantem $n$-argumentowym, tożsamościowemu zeru i tożsamościowej jedynce otrzymamy wzór (6) z twierdzenia 4.2.3, który z jeszcze większą dokładnością niż podana wcześniej granica Baugha ogranicza od dołu liczbę DFB dla $n$ zmiennych, oraz pozwala na szybsze uzyskanie wyniku obliczeń poprzez znaczne zmniejszenie liczby składników w sumie.

[1] Z punktu widzenia filtracji stosowej wykorzystuje się głównie funkcje o nieparzystej liczbie argumentów (patrz [7]).

[2] Dla $n$ parzystego warunek na liczbę DBF jest następujący:

$$2\sum_{i=1}^{n/2-1}(2^{C_{n,i}}-1)+2^{C_{n,n/2}}+2<\varphi(n)$$

## 5. Algorytm generowania DFB

### 5.1. Problemy wynikające przy implementacji algorytmu

Zanim przejdziemy do opisu samego algorytmu generowania DFB, zwróćmy uwagę na pewne zasadnicze problemy, które pojawiają się przy jego projektowaniu, implementacji i wykonywaniu programu, realizującego tego typu algorytm.

Największym problemem jest bardzo duża liczba funkcji dla rosnącego $n$. Dla 3 zmiennych jest ich tylko 20, dla 5 zmiennych już niecałe 8 tysięcy, a dla 7 zmiennych powyżej $2^{36}$ itd. Tak duża liczba funkcji nakłada realne ograniczenia na czas ich generowania oraz możliwości ich przechowywania, gdyż popularne języki programowania operują na zmiennych o wielkości do $2^{32}$, wreszcie trudno sobie wyobrazić sens utworzenia i korzystania ze zbioru, którego rozmiar przekroczyłby kilkaset gigabajtów (tak byłoby dla 7 i więcej zmiennych).

Następnym elementem wydłużającym czas generowania funkcji jest przyjęcie założenia, że w wygenerowanym zbiorze nie będzie funkcji powtarzających się przy możliwym sumowaniu implikantów prostych o różnej liczbie argumentów. Aby tego uniknąć, konieczne jest sprawdzanie implikantów pod względem wzajemnego pokrywania się. Korzystamy tutaj z poniższego lematu.

**Lemat 5.1**

DFB stworzona jak następuje $f_{k+1} = f_k + P_{k+1}$ jest w nieskracalnej postaci sumy (tzn. nie może być zredukowana do innej odpowiadającej jej postaci sumy), jeśli tylko $P_{k+1}$ jest implikantem prostym nie pochłanianym i nie pochłaniającym żadnego innego implikanta $P_i, i = 1,2,\ldots,k$ w funkcji $f_k$.

Dowód powyższego twierdzenia jest uzyskiwany wprost przez zaprzeczenie.

### 5.2. Generowanie DFB

Opisywany w tym punkcie pracy algorytm został zaimplementowany i uruchomiony z wykorzystaniem języka programowania Turbo Pascal w wersji 7.0, toteż podawany kod odpowiada notacji charakterystycznej dla tego języka. Ze względu na ograniczenia narzucane przez ten język, jak również czas wykonywania, testowanie odbyło się dla trzech, czterech oraz pięciu zmiennych, dając oczekiwane rezultaty. Przy takich założeniach na przechowywanie

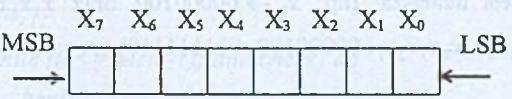
implikantów wystarczają zmienne typu bajt (8 bitów), tabela do ich przechowywania musi mieć 255 pozycji, a tabela do przechowywania tworzonej funkcji 70 pozycji[1].

Program wykonywany był na komputerze wyposażonym w procesor z serii Intel 486DX o częstotliwości zegara systemowego 50MHz i pamięć operacyjną 20MB, przy czym zaniechano zapisywania wygenerowanych funkcji na dysku, aby poznać czas samej generacji, który dla wszystkich trzech badanych przypadków wyniósł ułamek sekundy.

*5.2.1. Etap wstępny*

Ponieważ DFB tworzone są jako $m$-elementowe sumy implikantów prostych, gdzie $m = 1,2,\ldots k_{max}$, pierwszy etap musi obejmować wygenerowanie zbioru wszystkich implikantów prostych. Poniższy algorytm przechowuje je następnie w tablicy, której kolejne elementy interpretowane są w ten sposób, że ustawiony bit oznacza występowanie w implikancie argumentu o odpowiedniej wadze i indeksie, podczas gdy bity równe zero odpowiadają zmiennym nie pojawiającym się w danym implikancie. Przy tak przyjętym założeniu tworzenie zbioru implikantów prostych można sprowadzić do wygenerowania wszystkich liczb binarnych począwszy od 1 do wartości maksymalnej, równej liczbie implikantów prostych i odpowiadającej jedynkom na wszystkich bitach.

Poniższy rysunek przedstawia przyporządkowanie zmiennych funkcji bitom zgodnie z wagami i indeksami,




Rys. 1. Interpretowanie bitów jako zmiennych

Fig. 1. Interpreting bits as variables

gdzie MSB oznacza najbardziej znaczący bit, a LSB najmniej znaczący bit. Natomiast sposób uporządkowania implikantów prostych przy podanych założeniach pokazuje rys. 2, co odpowiada fragmentowi kodu:

```
for i:= 1 to licz_im do
tab_im_pr[i] := i;
```

w którym *licz_im* to zmienna określająca liczbę implikantów prostych dla danej liczby argumentów funkcji, a *tab_im_pr* jest tablicą przechowującą wszystkie implikanty.

[1] Podane wartości wynikają oczywiście z zależności podanych w poprzednich punktach tej pracy.

$x_0 \longrightarrow 00000001$

$x_1 \longrightarrow 00000010$

$x_1x_0 \longrightarrow 00000011$

$x_2 \longrightarrow 00000100$

$x_2x_0 \longrightarrow 00000101$

$\vdots \qquad\qquad \vdots$

$x_7x_6x_5x_4x_3x_2x_1 \longrightarrow 11111110$

$x_7x_6x_5x_4x_3x_2x_1x_0 \longrightarrow 11111111$

Rys. 2. Uporządkowanie implikantów

Fig. 2. The order of implicants

Zaimplementowane uporządkowanie bardzo upraszcza proces generowania zbioru implikantów, ale z kolei nie grupuje ich pod względem ilości argumentów wchodzących w skład. Jednakże ponieważ zadaniem algorytmu jest tworzenie wszystkich DFB, taki podział nie jest już potrzebny, gdyż i tak będziemy musieli sprawdzać, czy implikanty się nie pochłaniają wzajemnie.

Zwróćmy tutaj uwagę na fakt, że pokrywanie pomiędzy implikantami może odbywać się tylko „w przód", tzn. dany implikant może pokrywać inny tylko wtedy, gdy odpowiadająca mu liczba binarna jest mniejsza[1] (np. $x_2 \rightarrow 00000100$ oraz $x_7x_6x_5x_4x_3x_2x_1 \rightarrow 11111110$, więc $x_2$ pokrywa $x_7x_6x_5x_4x_3x_2x_1$ i $00000100 < 11111110$).

### *5.2.2. Procedura generowania funkcji*

Istota algorytmu zawiera się w rekurencyjnej procedurze generowania funkcji o $k$ składnikach, gdzie chcąc wygenerować wszystkie DFB konieczne jest zmienianie $k$ od 1 do $k_{max}$ (równe $C_{n,\lfloor n/2 \rfloor}$ jak podano wcześniej).

Działanie procedury można porównać do tworzenia kombinacji $k$-elementowej z $(2^n - 1)$ implikantów prostych, przy czym należy zapewnić, aby implikanty nie pokrywały się wzajemnie.

W poniższym algorytmie przyjęto następujące oznaczenia:

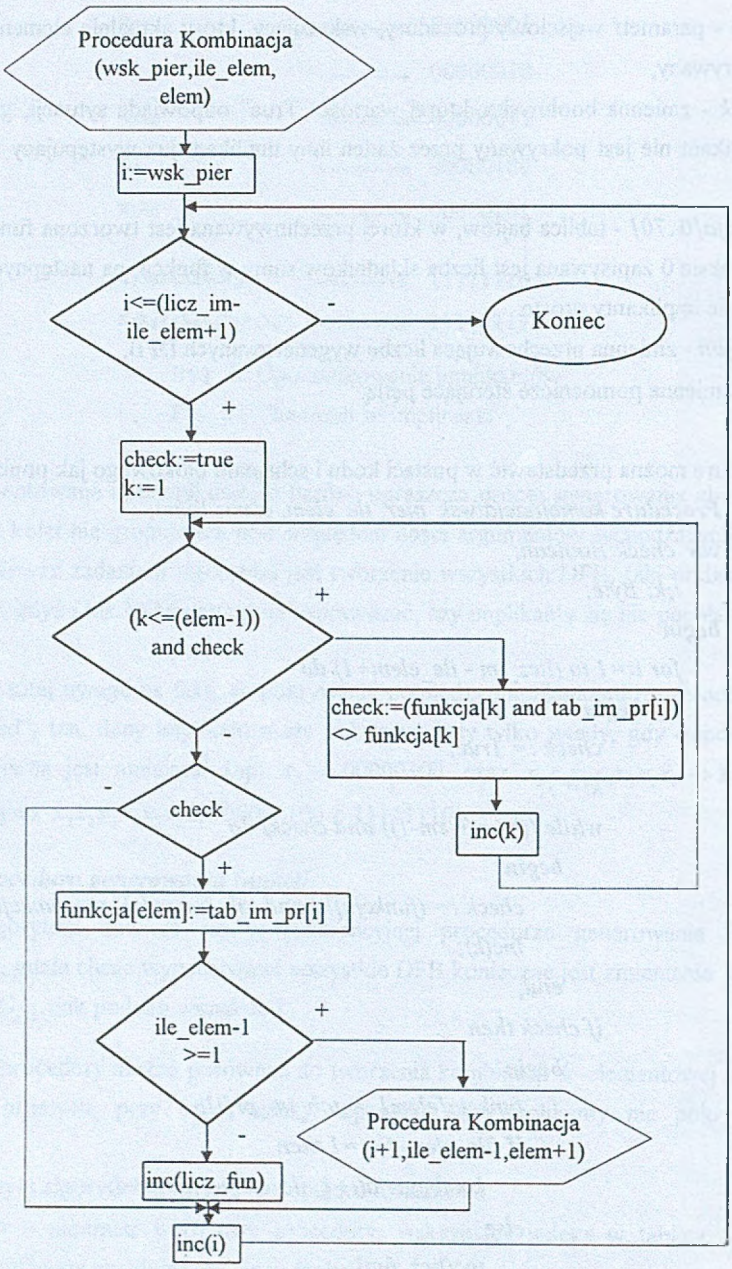
- *wsk_pier* - parametr wejściowy procedury, wskazujący indeks w tablicy *tab_im_pr* pierwszego implikanta znajdującego się w funkcji,
- *ile_elem* - parametr wejściowy procedury, określający, ile implikantów musimy jeszcze wybrać z tablicy *tab_im_pr*,aby stworzyć funkcję,

[1] Oczywiście jest to dodatkowy warunek do tego, który podaje definicja 2.3.

- *elem* - parametr wejściowy procedury, wskazujący, który aktualnie element kombinacji jest rozpatrywany,
- *check* - zmienna boolowska, której wartość „True" odpowiada sytuacji, gdy rozpatrywany implikant nie jest pokrywany przez żaden inny implikant już występujący w tworzonej funkcji,
- *funkcja[0..70]* - tablica bajtów, w której przechowywana jest tworzona funkcja. Na pozycji o indeksie 0 zapisywana jest liczba składników sumy w funkcji, na następnych pozycjach odpowiednie implikanty proste,
- *licz_fun* - zmienna przechowująca liczbę wygenerowanych DFB,
- *i,k* - zmienne pomocnicze sterujące pętlą.

Procedurę można przedstawić w postaci kodu i schematu blokowego jak poniżej.

```
Procedure kombinacja(wsk_pier, ile_elem, elem: Byte);
var check:Boolean;
    i;k: Byte;
begin
    for i:=1 to (licz_im - ile_elem+1) do
        begin
            check := True;
            k:= 1;
            while ((k<=(elem-1)) and check) do
                begin
                    check := (funkcja[k] and tab_im_pr[i]) <> funkcja[k];
                    inc(k);
                end;
            if check then
                begin
                    funkcja[elem] := tab_im_pr[i];
                    if (ile_elem-1)>=1 then
                        kombinacja(i+1, ile_elem-1,elem+1
                    else
                        inc(licz_fun);
                end;
        end;
end;
```




Rys. 3. Schemat blokowy procedury generowania DBF

Fig. 3. The block diagram of the procedure for genarating PBF

### *5.2.3. Możliwości modyfikacji algorytmu*

Efektywność podanego algorytmu wyznaczana jest poprzez dwa czynniki. Pierwszy z nich to rekurencja, na której oparty jest algorytm. Stopień zagnieżdżenia procedury odpowiada wprost liczbie składników generowanej aktualnie funkcji, a ta liczba zmienia się od 1 do $k_{max}$. Drugi czynnik to konieczność zapewnienia,aby implikanty nie pokrywały się wzajemnie. Przy dodawaniu nowego implikanta prostego do tworzonej funkcji jest on porównywany kolejno z wszystkimi pozostałymi, już tworzącymi funkcję składnikami i jeżeli okazuje się, że jest on pokrywany, to następuje jego pominięcie i pod uwagę brany jest następny implikant.

Rozważając optymalizację podanego algorytmu pod względem szybkości działania, należałoby zwrócić uwagę na oba wymienione wyżej czynniki. Rekurencja, która zapewnia tutaj „eleganckie" rozwiązanie, mogłaby być zastąpiona poprzez iterację, a unikalność tworzonych funkcji dałoby się osiągnąć poprzez przeprowadzenie porównania wszystkich implikantów przed wygenerowaniem funkcji. To jednakże wiązałoby się z koniecznością zachowania uzyskanych wyników, co znacznie zwiększyłoby rozmiar tablicy przechowującej implikanty, a i tak trzeba byłoby się do tych danych odwoływać podczas tworzenia funkcji. Dlatego też taka modyfikacja została zaniechana.

## 6. Zakończenie

DFB stanowią interesujący, a przede wszystkim stosunkowo jeszcze słabo zbadany i wykorzystany podzbiór funkcji logicznych, dlatego też są bardzo obiecującym obiektem dla przyszłych badań.

Weźmy przykładowo filtrację stosową - tutaj liczba zmiennych funkcji boolowskiej odpowiada szerokości apertury, czyli liczbie punktów obrazu, na której operuje filtr. Oczywiste wydaje się, że interesujące efekty można uzyskać dopiero po zastosowaniu apertury o szerokości 9, 25 czy jeszcze więcej, bo wówczas rozważamy całe otoczenie danego punktu. Jednakże dla takiej liczby zmiennych liczba funkcji możliwych do wygenerowania jest niewyobrażalnie duża. Z tego względu trudno określić, którą z nich należy wybrać, jeżeli chcemy uzyskać jakiś konkretny efekt jej działania. Dlatego też wskazane byłoby podzielenie całego tego zbioru na jakieś klasy i podklasy o wyszczególnionych charakterystykach. Taki podział może być przeprowadzony w oparciu o już znane, jak i nowo utworzone kryteria, dlatego w najbliższych latach można się spodziewać ciągle nowych publikacji prac, które zajmować się będą właśnie tym tematem.

Wszystkie próby kontaktu ze strony zainteresowanych tematem DFB będą mile widziane. Adres kontaktowy: ulas@zeus.polsl.gliwice.pl.

## LITERATURA

1. Gilbert E. N.: Lattice-theoretic properties of frontal switching functions. J. Math. Phys., vol. 33. pp. 57-67, April 1954.
2. Muroga S.: Threshold logic and its application. John Wiley and Sons, Inc., 1971.
3. Gabbouj M.: Optimal stack filter examples and Positive Boolean Functions. M.S.E.E. of Purdue Univ., December 1986.
4. Baugh C. R.: Bounds on the number of pseudothreshold functions. IEEE Trans. on Computers, C-20, pp. 1602-1605, 1971.
5. Hensel G.: Sur le nomble de functions booleannes montones de n variables. C. R. Acad. Sci. Paris, 262, pp. 1088-1090, 1966.
6. Siwiński J.: Układy przełączające w automatyce. WNT, Warszawa 1980.
7. Wojciechowski K., Kuś Z.: Przetwarzanie stosowe. ZN Pol. Śl. s. Automatyka z.113, Gliwice 1994.
8. Pr. zb. pod red. Małysiaka H.: Teoria automatów cyfrowych - laboratorium. Skrypt Pol. Śl. nr 2086, Gliwice 1997.
9. Spencer R. F., Jr.: Complex gates in digital systems design. IEEE Compt. Group News, pp. 47-56, September 1959.
10. Dedekind R.: Uber Zerlegungen von Zahlen durch ihre grossten gemeisamen Teiler. Festschrift Hotch Braunschweig, u. Ges. Werke, bd II, pp. 103-148, 1897.
11. Traczyk T.: Wstęp do teorii algebr Boole'a. PWN, Warszawa 1970.
12. Majewski W.: Układy logiczne. WNT, Warszawa 1974.

Recenzent: Prof. dr hab. inż. Marek Kubala

Wpłynęło do Redakcji 12 stycznia 1999 r.

**Abstract**

Boolean logic and Boolean functions come up in several broad fields such as digital computers, digital communication, control systems and are the subject matter of the switching circuits theory. Therefore, several books and papers have been written about them, but only little attention has been focused on positive Boolean functions.

As it is said in Section 2, positive Boolean functions are those of switching functions, which have no inversion (all arguments are positive) in their irredundant normal forms. PBF

have, among others, the stacking property, which is used in stack filtration of images, because at the binary level all stack filters are represented by PBF.

After giving some basic definitions from the switching circuit theory, this paper deals with the existing bounds on the number of PBF, due to Baugh [4] and Hensel [5] and their possible improvements. In this approach we assume that generated functions have all implicants with the same number of arguments, which significantly simplifies the formula for the lower bound and leads to the formula (6).

Finally, this work presents an algorithm for generating PBF i.e. the key procedure, which works recursively. The block diagram, shown in Fig. 3, is accompanied by the code in TurboPascal7.0 to illustrate its idea. As the total number of PBF drastically increases with the number of input variables, it has been possible to test the algorithm only for three up to five arguments and in these cases the results appeared plausible.

Anybody interested in the topic of Positive Boolean Functions is welcome to contact the author of this paper through e-mail. My address is: ulas@zeus.polsl.gliwice.pl.