

Introdução a funções lógicas e circuitos lógicos

Última revisão em 04 de março de 2016

Este documento é parte das notas de aula da disciplina MAC0329 (Álgebra Booleana e Circuitos Digitais) e apresenta uma introdução aos conceitos e terminologias importantes no estudo de circuitos digitais.

Para a leitura deste texto, alguma noção sobre representação de dados em computador, especialmente a representação binária de números, é importante. Dentre os tópicos relacionados e previamente vistos, citamos:

- sistemas de representação numérica: representações de números em diferentes bases (especialmente a base 2); conversão de representação de uma base para outra;
- representação de números no computador: palavras “binárias” com número fixo de bits (tipicamente 64 ou 32); representação de números sem sinal e com sinal; representação complemento de 2; intervalo dos números que podem ser representados em n bits;
- algoritmo de adição de números binários; adição e subtração de números sem sinal e com sinal (considerando a representação na forma complemento de 2); detecção de overflow nas operações de adição e subtração.

1 Funções lógicas

No estudo de circuitos digitais, comumente deparamos com os termos **função lógica**, **função binária**, **função booleana**, **função de chaveamento**. Todos eles referem-se a funções que mapeiam entradas binárias em saídas binárias. O termo lógica está predominantemente associado aos valores **verdadeiro** e **falso**, o binário aos valores **1** e **0**, e chaveamento com os estados **ligado** e **desligado**. Ao se remover o “aspecto semântico” dessas funções, temos uma estrutura matemática abstrata denominada **álgebra booleana** (nome derivado de George Boole, um matemático inglês), útil no estudo dessas funções. Uma definição formal de álgebra booleana será apresentada em outro documento, mas o conteúdo aqui descrito já utiliza algumas notações algébricas a serem vistas posteriormente.

Consideremos um computador. A menor unidade de armazenamento de informação no computador é o bit (contração de *Binary Digit*), capaz de representar dois estados que

podem ser interpretados como 0 e 1. Assim, computadores adotam a base binária para representar dados. Grupos de k bits são usados para armazenar dados no computador (os computadores populares modernos usam $k = 64$ bits). Portanto, podemos pensar que qualquer processamento de dados no computador é uma transformação de entradas em binário para saídas também em binário.

Transformações de dados podem ser descritas por uma função. No caso de dados binários, estamos considerando funções que mapeiam um certo número n de entradas binárias, que denotaremos x_1, x_2, \dots, x_n , para m saídas binárias y_1, \dots, y_m . Matematicamente temos uma função $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$. Podemos pensar também que $f = (f_1, f_2, \dots, f_m)$, com $f_i : \{0, 1\}^n \rightarrow \{0, 1\}$ e $y_i = f_i(x_1, x_2, \dots, x_n) \forall i$.

Breve pausa: $\{0, 1\}^n$ denota o produto cartesiano de conjuntos. Por exemplo, $\{0, 1\}^3$ corresponde ao produto cartesiano $\{0, 1\} \times \{0, 1\} \times \{0, 1\}$. Um elemento de $\{0, 1\} \times \{0, 1\} \times \{0, 1\}$ é uma trinca (a, b, c) , tal que $a, b, c \in \{0, 1\}$. De forma geral, dados por exemplo dois conjuntos A e B , temos que o produto cartesiano de A e B é o conjunto definido por $A \times B = \{(a, b) : a \in A \text{ e } b \in B\}$ (isto é, pares (a, b) tais que o primeiro elemento do par pertence a A e o segundo elemento do par pertence a B).

Pergunta: Se supormos que A contém n_a elementos e B contém n_b elementos, qual é o número de elementos (pares) no conjunto $A \times B$?

Exemplo: um exemplo de transformação, ou processamento, de dados é a adição de dois números. Supondo que o computador é de 8 bits, cada número a ser somado corresponderia a um conjunto de 8 bits. Considerando os dois números a serem somados, teríamos portanto 16 entradas. O resultado da adição deverá ser um número armazenado também em 8 bits. Assim, a função de adição é da forma $f : \{0, 1\}^{16} \rightarrow \{0, 1\}^8$.

O algoritmo de adição de binários é similar à adição usual de números em base 10 (ou seja, os números são somados coluna a coluna, da direita para a esquerda) com a diferença de que há apenas dois possíveis dígitos (0 e 1). Em cada coluna, pode haver um *carry-in* c_{in} (vai-um que veio da coluna anterior) e um *carry-out* c_{out} (vai-um para a próxima coluna). Usando essas notações, podemos escrever o comportamento da soma referente a uma coluna, conforme a tabela 1.

Na tabela, a e b indicam os dígitos em uma certa coluna dos dois números a serem somados e s o dígito da soma nesta mesma coluna. Pode-se ver que esta tabela define, para todas as possíveis atribuições de valores às variáveis c_{in} , a e b , o valor do bit soma s e do carry-out c_{out} . Isto é, temos duas funções binárias, $s(c_{in}, a, b)$ e $c_{out}(c_{in}, a, b)$, ambas com 3 entradas. Usando a notação anteriormente mencionada, poderíamos também considerar

Entrada			Saída	
c_{in}	a	b	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabela 1: Tabela que define o somador de bits.

que temos uma função $f(c_{in}, a, b) = (s(c_{in}, a, b), c_{out}(c_{in}, a, b))$.

A “transformação” de entrada para saída da operação de adição de dois números de 8 bits pode também ser descrito em uma tabela como a acima (porém é melhor não tentar fazer isso, pois o número de linhas da tabela é muito grande. Quantas linhas terá a tabela?) De forma similar, qualquer mapeamento de entradas binárias para saídas binárias pode ser descrita por uma tabela.

Se supormos que qualquer processamento de dados em computadores pode ser descrita por uma função binária, então podemos construir computadores usando dispositivos que implementam essas funções. Vamos então estudar quais seriam esses dispositivos e quais funções podem ser implementadas.

2 Circuitos lógicos

Visando entender um pouco como funções tais como as funções s e c_{out} definidas na tabela 1 poderiam ser implementadas, vamos “escrever” a função s de uma forma especial:

$$s(c_{in}, a, b) = 1 \iff (c_{in} = 0 \text{ e } a = 0 \text{ e } b = 1) \text{ ou } (c_{in} = 0 \text{ e } a = 1 \text{ e } b = 0) \text{ ou } \\ (c_{in} = 1 \text{ e } a = 0 \text{ e } b = 0) \text{ ou } (c_{in} = 1 \text{ e } a = 1 \text{ e } b = 1)$$

Note que no lado direito da equivalência, há uma expressão que inclui os **conectivos lógicos** E e OU, que podem ser “interpretados” da forma que estamos acostumados. A expressão “enumera” as entradas para as quais a função toma valor 1. Não é preciso muito esforço para verificar que para as demais entradas a função s toma valor 0.

Esses conectivos, que correspondem a operações lógicas básicas, definem funções lógicas, que podem ser descritas por tabelas conforme apresentadas a seguir, juntamente com a definição de outras funções lógicas básicas.

		Função lógica					
		E	OU	NÃO	XOR	NÃO-E	NÃO-OU
x_1	x_2	$x_1 x_2$	$x_1 + x_2$	\bar{x}_1	$x_1 \oplus x_2$	$\bar{x}_1 x_2$	$\bar{x}_1 + x_2$
0	0	0	0	1	0	1	1
0	1	0	1	1	1	1	0
1	0	0	1	0	1	1	0
1	1	1	1	0	0	0	0

Tabela 2: Tabela-verdade das funções lógicas básicas.

As colunas x_1, x_2 denotam as entradas e as demais colunas definem funções lógicas. XOR corresponde ao OU EXCLUSIVO. Por exemplo, a terceira coluna define o E lógico que toma valor 1 se e somente se $x_1 = 1$ e $x_2 = 1$, e isto pode ser expresso algebraicamente por $x_1 x_2$. A expressão algébrica das demais funções aparece também no cabeçalho de cada coluna.

Tabela-verdade é o nome que se dá à definição de uma função lógica escrita na forma de uma tabela como a tabela acima. Os valores são denotados pelos binários 0 e 1, mas poderiam ser equivalentemente denotados pelos valores lógicos V (verdadeiro) e F (falso).

A expressão da função s , apresentada acima, em termos de conectivos lógicos E e OU, sugere que se tivéssemos dispositivos físicos que implementassem o comportamento desses conectivos, seria possível implementar fisicamente a função s .

No caso de computadores digitais, os dispositivos que implementam as funções lógicas básicas acima são denominados **portas lógicas**. A figura 1 mostra as **portas lógicas** mais comumente usadas. Essas portas recebem sinais de entrada à esquerda e produzem um sinal de saída à direita.

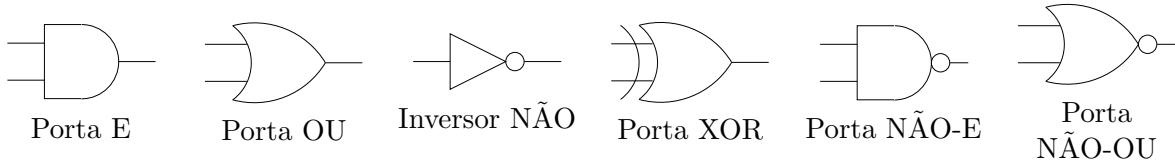


Figura 1: Representação gráfica de algumas portas lógicas.

Ao conectarmos a saída de um dispositivo nas entradas de outros, podemos construir uma rede interconexa de dispositivos. No caso de computadores digitais, a interconexão define o que é chamado de **círcuito digital**. Sem nos atermos à característica física da

implementação, podemos usar genericamente o termo **círcuito lógico** para nos referirmos a essas redes interconexas.

Seja um circuito, como por exemplo o mostrado na figura 2, que usa três inversores e uma porta E. A saída do circuito, expressa como $f(A, B)$, indica que o valor da saída depende das duas entradas A e B , que podem ser vistas como variáveis da função.

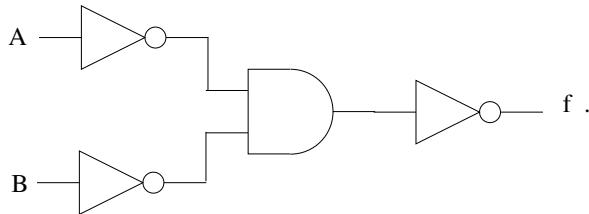


Figura 2: Um circuito simples.

Dizemos que **um circuito realiza uma função**. Podemos descrever seu funcionamento em uma tabela-verdade. Cada linha da tabela corresponde a uma das possíveis atribuições de valor às variáveis de entrada do circuito. No caso do circuito da figura 2, a tabela-verdade é mostrada a seguir, juntamente com os valores do circuito em pontos intermediários entre a entrada e a saída. Na forma algébrica, temos $f(A, B) = \overline{\overline{A} \overline{B}}$.

A	B	\overline{A}	\overline{B}	$\overline{A} \overline{B}$	$f(A, B) = \overline{\overline{A} \overline{B}}$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

Note que $f(A, B) = \overline{\overline{A} \overline{B}} = A + B$. Isto mostra que uma mesma função pode ser realizada por diferentes circuitos. Quando dois circuitos realizam uma mesma função, eles são ditos equivalentes. O circuito da figura 2 é equivalente à porta OR.

Obter a tabela-verdade correspondente a um circuito é uma tarefa mecânica. Obviamente, as tabelas-verdade (e portanto as funções realizadas) por circuitos equivalentes são exatamente iguais. Por outro lado, o problema inverso de desenhar (projetar) um circuito que realiza uma dada função parece ser muito complexa. Porém, há uma forma sistemática que permite desenhar circuitos correspondentes a uma função binária qualquer (porém, em geral o circuito assim obtido não é econômico nem eficiente).

Voltamos para o exemplo do somador de bits, e em especial à função s . Abaixo reescrevemos a expressão por conveniência:

$$s(c_{in}, a, b) = 1 \iff (c_{in} = 0 \text{ e } a = 0 \text{ e } b = 1) \text{ ou } (c_{in} = 0 \text{ e } a = 1 \text{ e } b = 0) \text{ ou } \\ (c_{in} = 1 \text{ e } a = 0 \text{ e } b = 0) \text{ ou } (c_{in} = 1 \text{ e } a = 1 \text{ e } b = 1)$$

Qual seria um circuito que realiza a função s ? Note que a expressão s é verdade (i.e., vale 1) se ao menos uma das quatro conjunções ($(c_{in} = 0 \text{ e } a = 0 \text{ e } b = 1)$, $(c_{in} = 0 \text{ e } a = 1 \text{ e } b = 0)$, $(c_{in} = 1 \text{ e } a = 0 \text{ e } b = 0)$ ou $(c_{in} = 1 \text{ e } a = 1 \text{ e } b = 1)$) for verdade. Na forma algébrica, a conjunção $(c_{in} = 0 \text{ e } a = 0 \text{ e } b = 1)$ pode ser expressa de forma compacta como $\bar{c}_{in} \bar{a} b$ (por quê?). Similarmente, os demais termos. Assim, a função s pode ser escrita pela expressão:

$$s(c_{in}, a, b) = \bar{c}_{in} \bar{a} b + \bar{c}_{in} a \bar{b} + c_{in} \bar{a} \bar{b} + c_{in} a b$$

Admitindo-se portas lógicas com múltiplas entradas, precisamos então de quatro portas E, uma porta OU e três inversores.

[incluir a figura do circuito aqui ...]

De forma similar, podemos chegar a um circuito que realiza a função c_{out} . Com isso teremos um circuito somador de bits, cujas entradas são os bits c_{in} , a e b , e as saídas são os bits s e c_{out} .

2.1 Circuito somador

Sejam $A = a_3 a_2 a_1 a_0$ e $B = b_3 b_2 b_1 b_0$ dois números binários de 4 bits (onde o subscrito 0 e 3 representam, respectivamente, o bit menos e mais significativo). Podemos projetar um circuito somador, isto é, um circuito que calcula a soma dos dois números, compondo-se

somadores de bits em série. Denotando as entradas dos somadores de bits por a_i , b_i e c_i (vai-um da coluna anterior) e as saídas por s_i e c_{i+1} , a ligação em série pode ser feita de forma que a saída vai-um de uma coluna i alimente a entrada vai-um da coluna $i + 1$, como mostrado na figura 3.

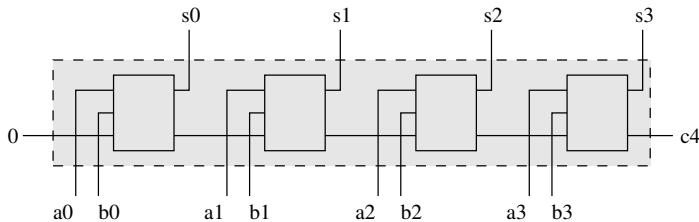


Figura 3: Esquema de um somador de 4 bits.

Cada uma das “caixas” corresponde a um somador de bits, que conforme já discutido acima, trata-se de um circuito de duas saídas (e, portanto, realiza duas funções). Do ponto de vista de função, podemos enxergar o somador de bits como uma função da forma $f : \{0, 1\}^3 \rightarrow \{0, 1\}^2$ (ou duas funções da forma $f : \{0, 1\}^3 \rightarrow \{0, 1\}$).

No diagrama, a entrada c_0 , isto é, o *carry-in* na coluna do bit menos significativo, é zero. Este valor é apropriado para a realização das operações de adição e subtração na representação complemento de dois. No caso da representação sem sinal, a adição pode ser efetuada por esse circuito sem alteração alguma; já a subtração pode ser realizada complementando-se o termo a ser subtraído e fazendo $c_0 = 1$ (complementar e somar 1 é a forma de se obter o negativo de um número na representação complemento de dois). Conforme visto antes, embora o mesmo circuito possa ser usado para as operações de adição e subtração no caso de representação sem sinal e no caso de representação com sinal (na forma complemento de dois), a forma de detecção de *overflow* difere de uma representação para a outra.

Exercício: Para ambas as representações, escreva uma função f que toma valor 1 no caso de *overflow* e toma valor zero caso contrário. Como poderia ser a implementação dessas funções ?

Utilizamos acima o circuito somador de bits discutido anteriormente, no qual cada função (isto é, s e c_{out}) é realizada por um circuito independente do outro. Na figura 4 é mostrado um outro circuito que também realiza as duas funções, porém com compartilhamento de subcircuito.

Exercício: Escreva a tabela-verdade correspondente ao circuito da figura 4, incluindo uma coluna para a saída de cada uma das cinco portas lógicas (não apenas para as portas que geram as saídas do circuito). Verifique que as saídas s e c_{out} são conforme esperadas.

Resumo: Neste documento foram introduzidos terminologias e conceitos relaciona-

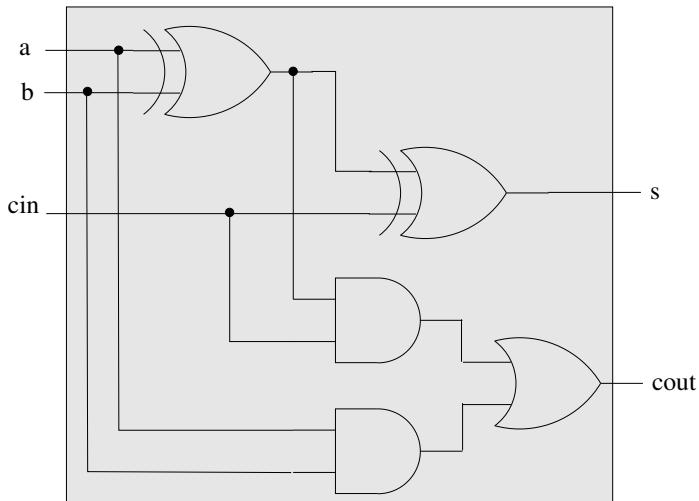


Figura 4: Esquema de um somador de bits, com compartilhamento de subcircuito.

dos a funções lógicas e circuitos lógicos. Considerando computadores como dispositivos que transformam dados binários em dados binários, podemos descrever o comportamento funcional do sistema computacional por meio de funções binárias (lógicas). Uma forma simples de especificação dessas funções são as tabelas-verdade. A partir da tabela-verdade pode-se deduzir uma expressão correspondente à função definida por ela. Conforme visto acima, qualquer expressão escrita em termos dos conectivos lógicos E, OU e a negação NÃO pode ser realizada por meio de um circuito (conectando-se as portas lógicas e inversores conforme a expressão). Ou não ?

Na sequência, serão consideradas questões como:

- Quais funções podem ser realizadas por um circuito?
- Como verificar se dois circuitos são equivalentes?
- Dada uma função realizável, qual a realização que utiliza o menor número de dispositivos?
- Dado um circuito, existe circuito equivalente e mais simples?
- Se nos restringirmos ao uso de determinados tipos de porta apenas (e não todos), quais tipos de funções são realizáveis?
- Como otimizar o compartilhamento de subcircuitos para a realização de várias funções ?

Não percam o próximo capítulo !

Minimização de funções booleanas

Última revisão em 15 de março de 2016

Este documento é parte das notas de aula da disciplina MAC0329 (Álgebra Booleana e Circuitos Digitais). Nesta parte abordaremos a manipulação de expressões na forma soma de produtos (SOP) e na forma produto de somas (POS), principalmente a simplificação delas.

Já vimos que uma função lógica pode ser descrita por meio de uma tabela-verdade (apesar de essa representação só ser praticável manualmente quando o número de entradas é pequeno ...) e, reciprocamente, que uma tabela-verdade define uma função.

1 Algumas definições

Antes de mais nada, vamos recordar as operações lógicas básicas E, OU e NÃO. Elas são definidas conforme a tabela 1.

		Função lógica		
x_1	x_2	E $x_1 x_2$	OU $x_1 + x_2$	NÃO \bar{x}_1
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Tabela 1: Tabela-verdade das operações lógicas E, OU e NÃO.

Algumas outras definições serão úteis ao longo do texto e são apresentadas aqui.

- iremos denominar por **variável booleana** qualquer variável que toma valores em $B = \{0, 1\}$;
- o **complemento** de uma variável booleana x , denotado \bar{x} , é uma variável booleana tal que $\bar{x} = \bar{a}$ sempre que $x = a$ para qualquer $a \in B$;
- um **literal** é uma variável booleana x ou o seu complemento \bar{x} .

2 Expressões na forma SOP e POS

Dada a definição de uma função por meio de sua tabela-verdade, podemos escrever a função por meio de uma expressão na forma soma de produtos (SOP). Para tanto, devemos considerar as entradas (atribuição de valores às variáveis da função) para as quais a função toma valor 1. Similarmente, podemos escrever a função por meio de uma expressão na forma produto de somas (POS), conforme mostrado no próximo exemplo.

Exemplo: Seja a tabela-verdade (**OBS.:** Lembrem-se de sempre escrever as entradas da tabela verdade em ordem lexicográfica) :

Entrada			Saída
a	b	c	$f(a, b, c)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A função definida na tabela-verdade acima toma valor 1 para as entradas 010, 100, 101, 110 e 111. O produto $\bar{a}b\bar{c}$ toma valor 1 quando a entrada é 010 (pois, $\bar{a}b\bar{c} = \bar{0}1\bar{0} = 1 \cdot 1 \cdot 1 = 1$). Para qualquer outra entrada esse produto toma valor 0. Assim, podemos associar um único produto para cada entrada. Portanto, uma função pode ser escrita como a soma dos produtos associados às entradas para as quais ela toma valor 1. A função da tabela acima pode ser escrita como:

$$f(a, b, c) = \bar{a}b\bar{c} + a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c} + abc \quad (1)$$

Trata-se de uma expressão na forma soma de produtos (SOP). Os produtos na equação acima são canônicos pois envolvem todas as três variáveis, sendo cada uma delas ou na forma não barrada ou na forma barrada.

De forma dual, podemos também escrever uma função na forma produto de somas (POS). Observe que a função da tabela toma valor zero para as entradas 000, 001 e 011. A soma $a + b + c$ toma valor 0 para a entrada 000 e valor 1 para as demais entradas.

Assim, similarmente as caso dos produtos, podemos associar uma soma para cada entrada. Portanto, uma função pode também ser escrita como o produto de somas associadas às entradas para as quais ela toma valor 0. No caso da função f acima, temos que:

$$f(a, b, c) = (a + b + c)(a + b + \bar{c})(a + \bar{b} + \bar{c}) \quad (2)$$

Trata-se de uma expressão na forma produto de somas (POS), com somas canônicas.

Como o conjunto de todas as sequências de n bits corresponde à representação binária dos números entre 0 e $2^n - 1$, o produto canônico (também chamado de **mintermo**) associado a uma entrada de n bits pode ser caracterizado por um índice decimal. A tabela 2 apresenta todos os mintermos em três variáveis e a notação com índice decimal associada a cada um deles.

Entrada binária	mintermo	notação
0 0 0	$\bar{x}_1 \bar{x}_2 \bar{x}_3$	m_0
0 0 1	$\bar{x}_1 \bar{x}_2 x_3$	m_1
0 1 0	$\bar{x}_1 x_2 \bar{x}_3$	m_2
0 1 1	$\bar{x}_1 x_2 x_3$	m_3
1 0 0	$x_1 \bar{x}_2 \bar{x}_3$	m_4
1 0 1	$x_1 \bar{x}_2 x_3$	m_5
1 1 0	$x_1 x_2 \bar{x}_3$	m_6
1 1 1	$x_1 x_2 x_3$	m_7

Tabela 2: Tabela de mintermos em 3 variáveis.

Usando-se o índice decimal associado a cada produto canônico, podemos escrever a expressão de forma mais compacta como:

$$f(a, b, c) = m_2 + m_4 + m_5 + m_6 + m_7 = \sum m(2, 4, 5, 6, 7)$$

Maxtermo (ou **soma canônica**) em n variáveis x_1, x_2, \dots, x_n tem definição similar ao mintermo: em vez de produto, consiste de soma de n literais, cada um correspondendo a uma variável. As expressões $\bar{x}_1 + \bar{x}_2 + \bar{x}_3$ e $\bar{x}_1 + x_2 + x_3$ são exemplos de maxtermos em três variáveis. A tabela 3 lista todos os maxtermos de 3 variáveis.

Na forma produto de somas compacta temos:

$$f(a, b, c) = M_0 \cdot M_1 \cdot M_3 = \prod M(0, 1, 3)$$

Entrada binária	maxtermo	notação
0 0 0	$x_1 + x_2 + x_3$	M_0
0 0 1	$x_1 + x_2 + \bar{x}_3$	M_1
0 1 0	$x_1 + \bar{x}_2 + x_3$	M_2
0 1 1	$x_1 + \bar{x}_2 + \bar{x}_3$	M_3
1 0 0	$\bar{x}_1 + x_2 + x_3$	M_4
1 0 1	$\bar{x}_1 + x_2 + \bar{x}_3$	M_5
1 1 0	$\bar{x}_1 + \bar{x}_2 + x_3$	M_6
1 1 1	$\bar{x}_1 + \bar{x}_2 + \bar{x}_3$	M_7

Tabela 3: Tabela de maxtermos com 3 variáveis.

3 Minimização de funções booleanas

Vimos que uma mesma função pode ser expressa por mais de uma expressão. Dependendo do contexto no qual essas expressões são utilizadas, pode-se desejar encontrar, dentre todas as expressões que representam uma mesma função booleana, aquela que satisfaz algum critério. Por exemplo, pode ser do interesse obter uma expressão “mais curta”, ou então, uma expressão que não envolve um termo produto com mais de um determinado número de literais.

Uma das simplificações bastante estudadas no contexto de circuitos digitais é a minimização lógica dois-níveis, isto é, a expressão de uma função na forma soma de produtos, envolvendo o menor número possível de termos produto e, em cada produto, o menor número possível de literais. O número de produtos e o número de literais em cada produto definem, respectivamente, o número de portas lógicas E e o número de entradas na correspondente porta E, em uma implementação direta da expressão.

Algoritmos de minimização lógica dois-níveis partem da expressão na forma SOP.

Exemplo: Sejam três variáveis a , b e c . Dados os produtos abc e $\bar{a}bc$, seja a disjunção (soma) $abc + \bar{a}bc$. Observe que podemos usar a mesma regra da álgebra elementar e colocar bc em evidência: $abc + \bar{a}bc = (a + \bar{a})bc = bc$ ($a + \bar{a} = 1$ sempre e, portanto, $(a + \bar{a})bc = 1 \cdot bc = bc$). O termo resultante bc é também um produto, porém sem a variável a . Note que o produto bc toma valor 1 para as entradas 011 e 111; o valor da variável a não afeta o valor desse produto. Veja uma compilação na tabela a seguir:

Expressão	Entradas para as quais a expressão toma valor 1
$\bar{a}bc$	{011}
abc	{111}
$\bar{a}bc + abc = bc$	{011, 111}

Em termos de circuitos, a simplificação acima significa que duas portas lógicas E de três entradas cada podem ser substituídas por uma porta E de duas entradas! Além disso, não será mais necessária uma porta OU.

Um produto pode ser associado a um intervalo. No caso do produto bc , as entradas para as quais o produto toma valor 1 é dado pelo conjunto $\{011, 111\}$. Esse conjunto tem um extremo inferior, 011, e um extremo superior, 111, definindo o intervalo $[011, 111]$. Um intervalo pode ser representado compactamente trocando-se as coordenadas não fixas por X . Assim $X11$ corresponde ao intervalo $[011, 111]$. Se o produto considerado fosse a , teríamos que ele toma valor 1 para as entradas $\{100, 101, 110, 111\}$ que pode ser entendido como o intervalo $[100, 111]$. Em notação compacta, esse intervalo pode ser denotado por $1XX$. Similarmente, um intervalo pode ser associado a um produto. No caso de três variáveis a, b e c , o intervalo $[000, 100] = \{000, 100\}$ (ou simplesmente $X00$) corresponde ao produto $\bar{b}\bar{c}$. O intervalo $X0X = [000, 101] = \{000, 001, 100, 101\}$ corresponde ao produto \bar{b} . Um intervalo contém necessariamente 2^k elementos, onde $0 \leq k \leq n$.

Esses conceitos/terminologias estão resumidos no quadro a seguir:

Produto	elementos cobertos	intervalo	notação compacta (cubo/intervalo)	dimensão (k)	tamanho (2^k)
ab	$\{110, 111\}$	$[110, 111]$	$11X$	1	$2^1 = 2$
c	$\{001, 011, 101, 111\}$	$[001, 111]$	$XX1$	2	$2^2 = 4$

Observação: Daqui em diante utilizaremos equivalentemente os termos **produto**, **cubo** ou **intervalo** quando nos referirmos a um produto.

3.1 Simplificação algébrica

Usando regras algébricas, como por exemplo em $abc + \bar{a}bc = (a + \bar{a})bc = bc$, podemos realizar a simplificação de uma expressão.

Retomando a expressão do início, podemos simplificar como segue:

$$\begin{aligned}
 f(a, b, c) &= \bar{a}b\bar{c} + a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c} + abc \\
 &= \bar{a}b\bar{c} + a\bar{b}\bar{c} + a\bar{b}c + a\bar{b}c + ab\bar{c} + abc \\
 &= (\bar{a} + a)b\bar{c} + a\bar{b}(\bar{c} + c) + ab(\bar{c} + c) \\
 &= b\bar{c} + a\bar{b} + ab \\
 &= b\bar{c} + a(\bar{b} + b)
 \end{aligned}$$

$$\begin{aligned}
 &= b\bar{c} + a \\
 &= a + b\bar{c}
 \end{aligned}$$

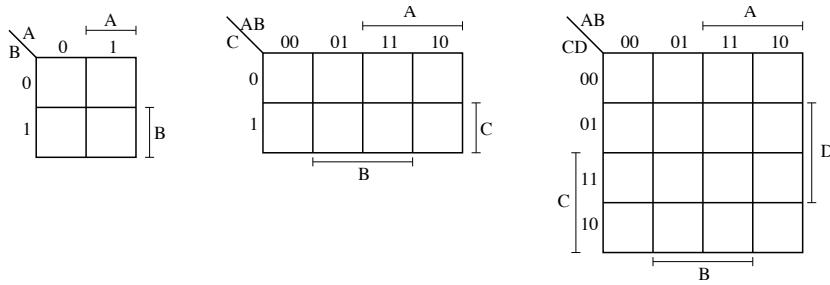
A simplificação algébrica, porém, pode ser complicada pois requer o conhecimento das regras e, além disso, dependendo da ordem na qual as regras são aplicadas pode ser necessário um grande número de passos para se chegar à simplificação (sem contar que às vezes podemos, depois de vários passos, voltar à situação inicial). Se a expressão envolver muitas variáveis, fazer a simplificação na mão traz adicionalmente uma grande chance de se cometer erros bobos (trocar o nome de uma variável sem querer, esquecer o barra sobre uma variável de um passo para o outro, etc).

Iremos ver as regras algébricas e exercitar a simplificação e manipulação algébrica de expressões oportunamente, mas não neste momento.

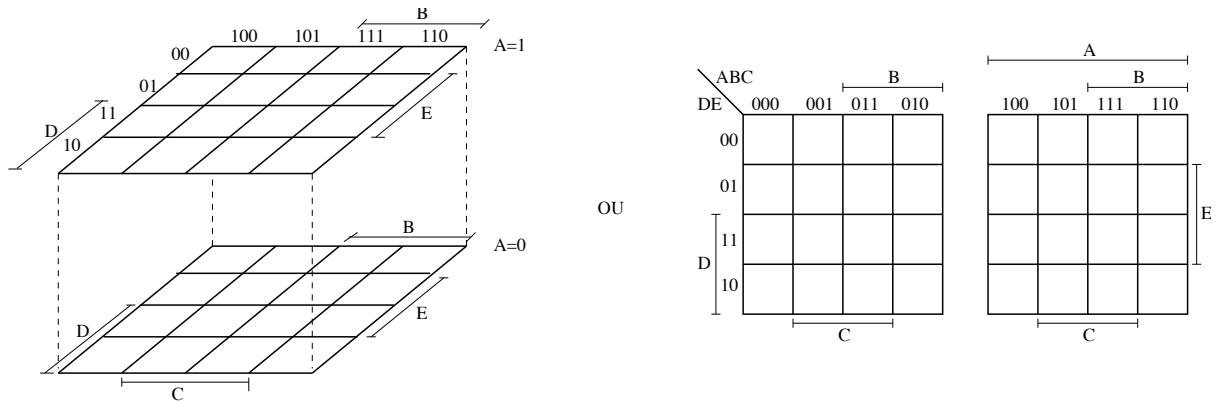
3.2 Mapas de Karnaugh

Mapas de Karnaugh são diagramas que são utilizados para auxiliar o processo de minimização lógica dois-níveis, quando o número de variáveis não passa de 6. Embora não seja usado na prática, como ferramenta pedagógica é interessante e será explorado a seguir.

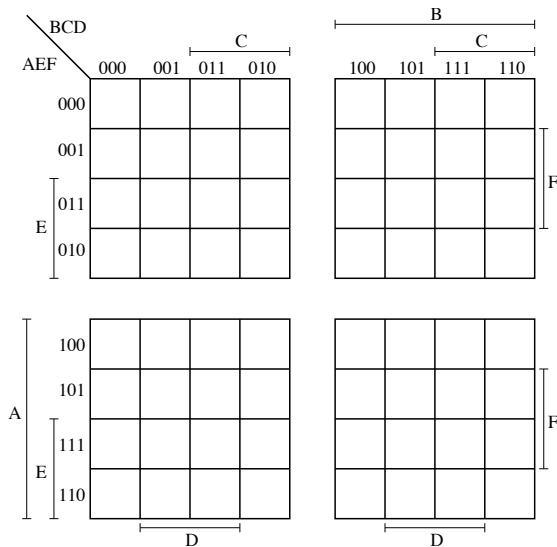
Mapa de Karnaugh de 2, 3 e 4 variáveis:



Mapa de Karnaugh de 5 variáveis:



Mapa de Karnaugh de 6 variáveis:



Cada célula dos mapas corresponde a um elemento de $\{0, 1\}^n$ (no caso $n = 2, 3, 4, 5$ ou 6). A concatenação do cabeçalho da coluna com o cabeçalho da linha de uma célula dá o elemento correspondente àquela célula. No caso de 3 variáveis, o mapa à esquerda na figura 1 mostra em cada célula a entrada correspondente, enquanto o mapa à direita mostra em cada célula o valor decimal das respectivas entradas. Observe que o cabeçalho está disposto em uma sequência não-usual. Por exemplo, para duas variáveis, a sequência natural seria 00, 01, 10, 11. Porém, a sequência utilizada é 00, 01, 11, 10, que possui a característica de dois elementos adjacentes (na sequência) diferirem em apenas 1 bit (essa propriedade vale se juntarmos os extremos esquerdo e direito do mapa e, similarmente, os extremos superior e inferior).

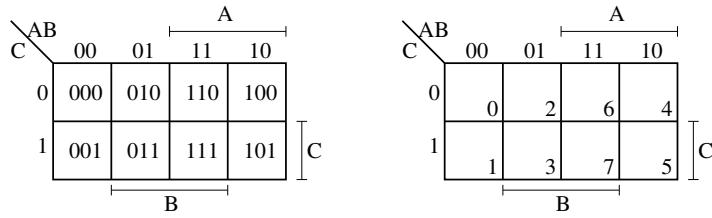


Figura 1: As entradas correspondentes a cada célula do mapa de Karnaugh de 3 variáveis em notação binária e decimal, respectivamente.

Vejamos, por meio de um exemplo, como pode ser realizada a minimização utilizando o mapa de Karnaugh. Seja $f(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 x_2 x_3 + x_1 x_2 x_3$. Algebricamente, podemos proceder como segue:

$$\begin{aligned}
 f(x_1, x_2, x_3) &= \bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_1 x_2 x_3 + x_1 x_2 x_3 \\
 &= \bar{x}_1 \bar{x}_2 (\bar{x}_3 + x_3) + \bar{x}_1 x_2 (\bar{x}_3 + x_3) + x_1 x_2 x_3 \\
 &= \bar{x}_1 \bar{x}_2 + \bar{x}_1 x_2 + x_1 x_2 x_3 \\
 &= \bar{x}_1 (\bar{x}_2 + x_2) + x_1 x_2 x_3 \\
 &= \bar{x}_1 + x_1 x_2 x_3 \\
 &= \bar{x}_1 + x_2 x_3
 \end{aligned}$$

Aparentemente a expressão acima é minimal. Para utilizarmos o mapa de Karnaugh, precisamos primeiramente transformar os mintermos da função para a notação cúbica. Assim,

Mintermo	notação cúbica
$\bar{x}_1 \bar{x}_2 \bar{x}_3$	000
$\bar{x}_1 \bar{x}_2 x_3$	001
$\bar{x}_1 x_2 \bar{x}_3$	010
$\bar{x}_1 x_2 x_3$	011
$x_1 x_2 x_3$	111

Em seguida, as células correspondentes a esses mintermos devem ser marcados com 1 no mapa, conforme mostrado no mapa da esquerda na figura 2.

O processo consiste, então, em procurar, para cada 1 no mapa, o maior agrupamento retangular de 1's que inclui o primeiro 1, e que tenha 2^k elementos ($k \geq 1$), chamados cubos maximais. No exemplo da figura 2, o maior agrupamento que cobre 000 é o cubo

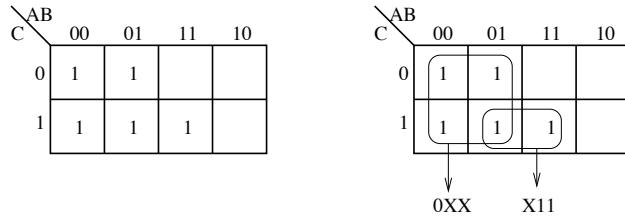


Figura 2: Exemplo do uso do mapa de Karnaugh: minimização da função $f = \sum m(0, 1, 2, 3, 7)$.

0XX. Este cubo não cobre o elemento 111. Assim, tomamos também o maior cubo que cobre 111, que no caso é o cubo X11. Depois desse procedimento, todos os mintermos da função encontram-se cobertos por algum cubo. Assim, podemos dizer que uma solução SOP minimal corresponde aos cubos 0XX e X11. O produto correspondente ao cubo 0XX é \bar{x}_1 e o correspondente a X11 é $x_2 x_3$. Portanto, uma forma SOP minimal é $f(x_1, x_2, x_3) = \bar{x}_1 + x_2 x_3$.

Exemplo: Minimize a função $f(a, b, c, d) = \sum m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15)$. A resposta é $f(a, b, c, d) = c + \bar{a} b d + \bar{b} \bar{d}$. Veja o mapa da figura 3.

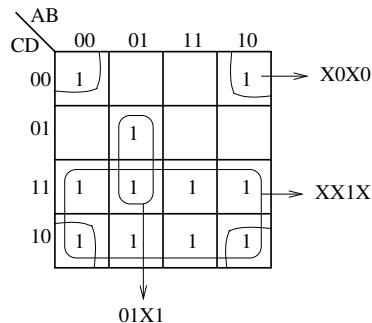


Figura 3: Minimização da função $f(a, b, c) = \sum m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15)$.

Exemplo: Minimize a função $f(a, b, c, d) = \sum m(0, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15)$. Neste caso, há mais de uma solução. A figura 4 mostra todos os cubos maximais de f . As possíveis soluções são:

$$\begin{aligned} f(a, b, c, d) &= \bar{a} b + a \bar{b} + \bar{a} \bar{c} \bar{d} + b c \\ f(a, b, c, d) &= \bar{a} b + a \bar{b} + \bar{a} \bar{c} \bar{d} + a c \end{aligned}$$

$$\begin{aligned} f(a, b, c, d) &= \bar{a} b + a \bar{b} + \bar{b} \bar{c} \bar{d} + b c \\ f(a, b, c, d) &= \bar{a} b + a \bar{b} + \bar{b} \bar{c} \bar{d} + a c \end{aligned}$$

Este exemplo mostra que a solução não é única (ou seja, podem existir mais de uma forma SOP minimal de mesmo custo) e que nem todos os implicantes primos fazem parte da forma SOP minimal.

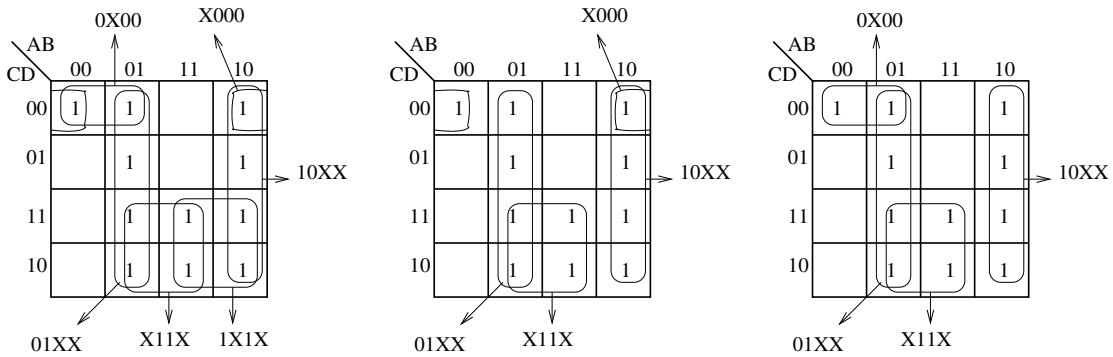


Figura 4: Minimização da função $f(a, b, c) = \sum m(0, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15)$. Esquerda: todos os implicants primos (ou cubos maximais). Centro: uma solução. Direita: outra solução.

Mapa de Karnaugh para encontrar a forma POS minimal: o mapa de Karnaugh pode ser utilizado também para encontrar a forma POS (produto de somas) minimal de uma função booleana? A resposta é sim. Considere a função $f(a, b, c) = \sum m(0, 4, 5, 7)$. A minimização SOP de f por mapa de Karnaugh é mostrada na figura 5. A forma SOP minimal é $f(a, b, c) = \bar{b}\bar{c} + a.c$.

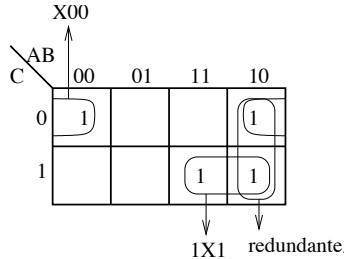


Figura 5: Exemplo do uso do mapa de Karnaugh: minimização da função $f(a, b, c) = \sum m(0, 4, 5, 7)$.

Usando o fato de que $f = \bar{\bar{f}}$, podemos calcular inicialmente a minimização SOP de $\bar{f}(a, b, c) = \sum m(1, 2, 3, 6)$ por mapa de Karnaugh (figura 6). O resultado obtido é $\bar{f}(a, b, c) = b\bar{c} + \bar{a}c$.

Em seguida, temos $f = \bar{b}\bar{c} + \bar{a}c = (\bar{b}\bar{c})(\bar{a}c) = (\bar{b} + c)(a + \bar{c})$.

Tudo isto pode ser diretamente realizado no mapa de Karnaugh conforme mostrado na figura 7. Em vez de marcar os 0-cubos da função no mapa, marcamos os 0-cubos do complemento de f (ou, equivalentemente, os zeros da função). Aplica-se o processo de encontrar os cubos maximais. Para escrever a função na forma POS minimal, basta escrevermos o termo soma correspondente a cada cubo. No exemplo, o cubo $0X1$ corresponde ao termo soma $a + \bar{c}$ e o cubo $X10$ ao termo soma $\bar{b} + c$. Assim, temos que a forma POS

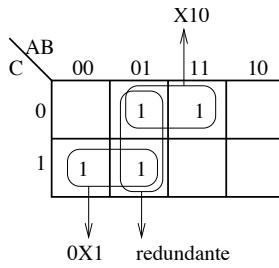


Figura 6: Minimização da função $\bar{f}(a, b, c) = \sum m(1, 2, 3, 6)$.

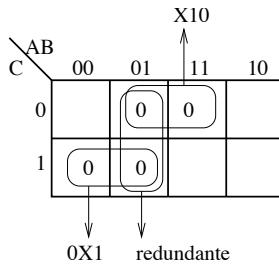


Figura 7: Minimização POS da função $f(a, b, c) = \prod M(1, 2, 3, 6)$.

minimal é $f(a, b, c) = (a + \bar{c})(\bar{b} + c)$.

3.3 Minimização na presença de don't cares

Em algumas situações, o valor de uma função para algumas entradas não são relevantes (tipicamente porque tais entradas nunca ocorrerão na prática). Em tais situações, tanto faz se a função toma valor 0 ou 1 nessas entradas, que serão denominadas de **don't cares**.

Para minimizar uma função incompletamente especificada, os don't cares podem ser utilizados para formar cubos maximais junto com os 1's da função. Don't cares agrupados serão mapeados para 1 pela função simplificada resultante, enquanto os não utilizados serão mapeados para zero.

3.4 Minimização de múltiplas funções

Exemplo 1: Considere as funções f_1 e f_2 dadas pelas seguintes tabelas:

$x_1 x_2 x_3$	$f_1(x_1, x_2, x_3)$	$x_1 x_2 x_3$	$f_2(x_1, x_2, x_3)$
000	1	000	0
001	1	001	0
010	0	010	0
011	0	011	0
100	0	100	0
101	1	101	1
110	0	110	1
111	0	111	1

A minimização individual destas duas funções resulta em $f_1(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 + \bar{x}_2 x_3$ e $f_2(x_1, x_2, x_3) = x_1 x_3 + x_1 x_2$. Para implementá-las, são necessárias 4 portas E.

Note, porém, que podemos escrever $f_1(x_1 x_2 x_3) = \bar{x}_1 \bar{x}_2 + \bar{x}_2 x_3 = \bar{x}_1 \bar{x}_2 + x_1 \bar{x}_2 x_3$ e $f_2(x_1 x_2 x_3) = x_1 x_3 + x_1 x_2 = x_1 x_2 + x_1 \bar{x}_2 x_3$. Neste caso, há um produto comum às duas funções e portanto para implementá-las são necessárias 3 portas E, pois uma das portas E é compartilhada pelas duas funções.

Exemplo 2: Considere as funções (já minimizadas individualmente) :

$$f_0 = a$$

$$f_1 = \bar{b}$$

$$f_2 = bc + ab$$

$$f_3 = \bar{a} \bar{b} + \bar{a} c$$

Se expressas desta forma, para implementá-las, são necessárias 6 portas E. No entanto, note que elas podem ser reescritas como:

$$f_0 = a \bar{b} + ab$$

$$f_1 = \bar{a} b + a \bar{b}$$

$$f_2 = \bar{a} bc + ab$$

$$f_3 = \bar{a} \bar{b} + \bar{a} b c$$

e neste caso são necessárias apenas 4 portas E.

Estes exemplos mostram que minimizar individualmente as funções não necessariamente representa a melhor solução para a minimização conjunta. Observe também que, no

segundo exemplo, na minimização conjunta reduzimos o número total de produtos, mas o número de somas aumentou. Isto é uma desvantagem? Se o objetivo é a realização em um PLA (*Programmable logic array*) não faz diferença se a função é formada, por exemplo, por 2 ou 3 termos produto. Porém, se o objetivo for minimizar também o número de entradas das portas, então é preciso tomar cuidado para não aumentar demaisadamente o número de termos (pois isso implica aumento no número de entradas da porta OU).

3.5 PLA

A minimização lógica dois-níveis ganhou impulso na década de 1980 devido aos dispositivos conhecidos como **PLA** (*Programmable Logic Arrays*). Eles consistem de um conjunto de entradas, com uma malha programável de conexões para um conjunto de portas E, e uma malha programável de conexões entre as saídas das portas E para um conjunto de portas OU. Por malha programável entende-se que os cruzamentos podem ser conectados (programados) para conduzir o sinal. No estado inicial, nenhum cruzamento está conectado nas malhas de um PLA.

A figura 8 mostra um modelo lógico básico de um PLA típico, com 3 variáveis de entrada e três saídas. Os círculos (pontos pretos) sobre o cruzamento das linhas indicam onde há conexão. No exemplo, as portas lógicas E realizam, respectivamente de cima para baixo, as funções (produtos) $a\bar{b}$, $\bar{a}b\bar{c}$, $\bar{b}c$ e $a\bar{c}$; as portas lógicas OU realizam, respectivamente da esquerda para a direita, as funções $f_1(a, b, c) = \bar{a}b\bar{c} + a\bar{b}$, $f_2(a, b, c) = \bar{a}b\bar{c} + \bar{b}c$ e $f_3(a, b, c) = \bar{a}b\bar{c} + a\bar{c}$.

Para não sobrecarregar o diagrama, em geral desenha-se de forma simplificada como o mostrado na figura 9.

PLAs comerciais têm tipicamente¹ entre 10 e 20 entradas, entre 30 e 60 portas E (produtos) e entre 10 e 20 portas OU (saídas). Em um PLA com 16 entradas, 48 produtos e 8 saídas, existem $2 \times 16 \times 48 = 1536$ cruzamentos na malha E e $8 \times 48 = 384$ cruzamentos na malha OU. Um número considerável de funções relativamente complexas podem ser realizadas via um PLA. Claramente, quanto menor o número de variáveis e termos produtos utilizados na expressão de uma função, menor será o “tamanho” do PLA necessário para a realização da função.

Exercícios:

¹Informação colhida em torno de 2010 ...

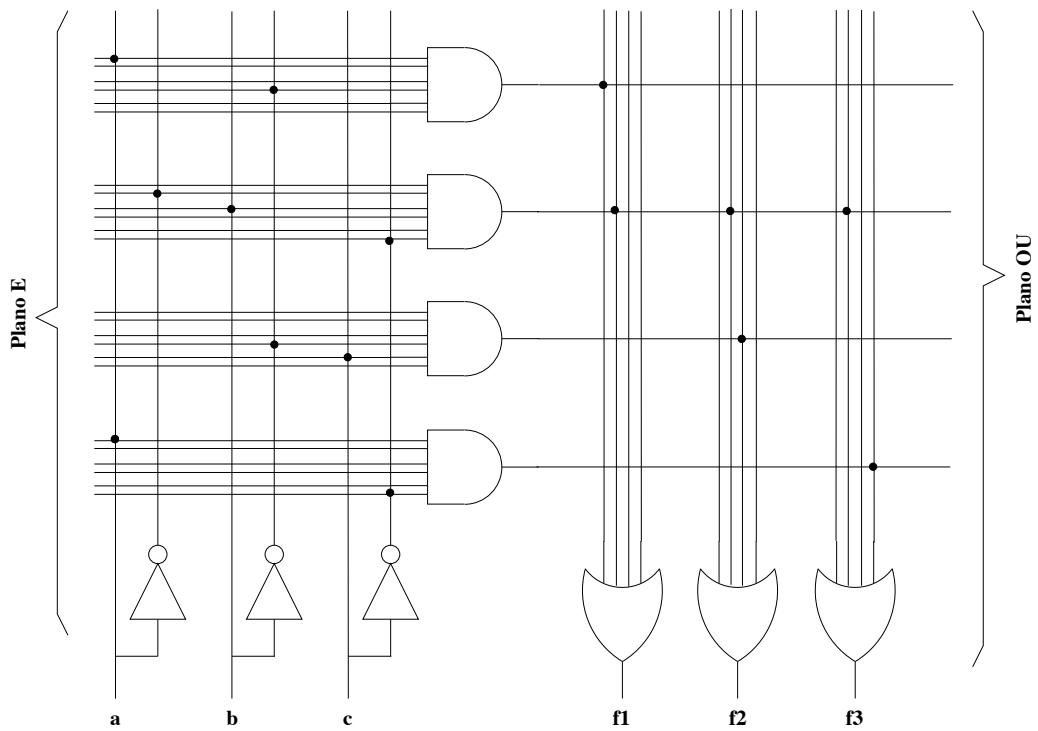


Figura 8: Esquema lógico de um PLA.

1. Minimize as funções dos exemplos 1 e 2 usando o mapa de Karnaugh e compare os resultados com os que foram apresentados acima.
2. Mostre como fica a realização das quatro funções do Exemplo 2, utilizando 4 portas E, em um PLA.

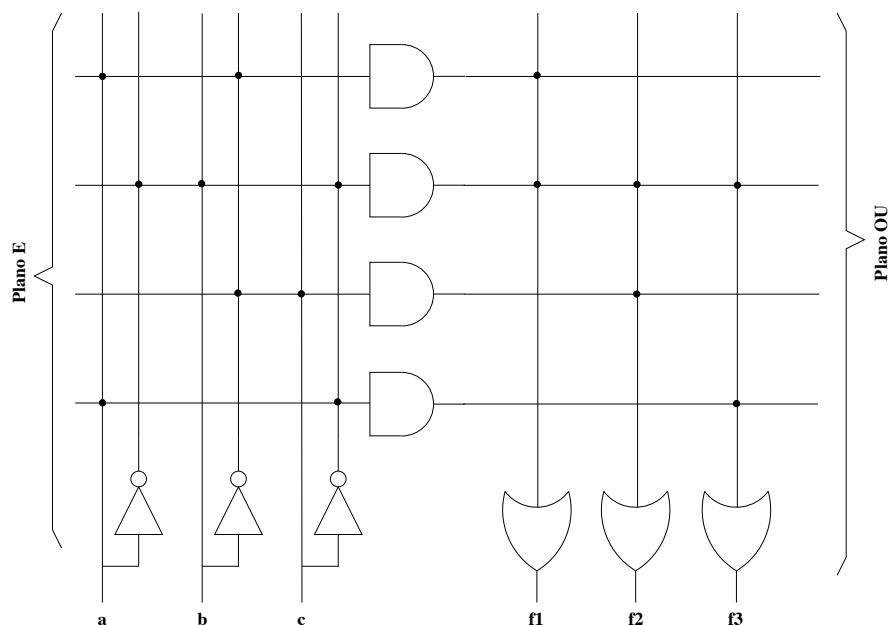


Figura 9: Esquema simplificado de um PLA.

Minimização de funções booleanas - parte II

Última revisão em 04 de abril de 2016

Este documento é parte das notas de aula da disciplina MAC0329 (Álgebra Booleana e Circuitos Digitais). Nesta parte continuaremos a abordar a minimização de funções booleanas e, mais especificamente, a minimização lógica dois-níveis.

1 Revisão de alguns conceitos

Para a leitura deste documento é conveniente relembrarmos alguns conceitos:

Função lógica de n variáveis: qualquer função binária definida em n variáveis binárias, isto é, uma função da forma $f : \{0, 1\}^n \rightarrow \{0, 1\}$

Produtos. Um produto é uma expressão booleana que consiste de uma conjunção de literais, não envolvendo uma mesma variável mais de uma vez. Em particular, o produto canônico (ou mintermo) é um produto em que cada variável ocorre exatamente uma vez, ou na forma barrada ou na forma não-barrada. Um produto canônico em n variáveis toma valor 1 em apenas um elemento do conjunto $\{0, 1\}^n$.

Por exemplo, dadas três variáveis a , b e c , os produtos abc e $\bar{a}bc$ são canônicos. A disjunção deles, $abc + \bar{a}bc = (a + \bar{a})bc = bc$, é também um produto, porém não é canônico. Observe que o produto bc toma valor 1 para os elementos 011 e 111; o valor da variável a não afeta o valor desse produto.

Somas. Dual aos produtos. A soma $a + \bar{b} + c$ toma valor zero apenas quando $a = 0$, $b = 1$ e $c = 0$.

Expressão na forma SOP e POS: qualquer função lógica pode ser expressa como soma de produtos canônicos ou como produto de somas canônicas. As expressões nessas formas podem ser obtidas, por exemplo, diretamente da tabela-verdade da função.

Mintermo e maxtermo: correspondem a, respectivamente, produto e soma canônicos

Implicantes. Dadas duas funções f e g , dizemos que f implica g se $f(\mathbf{x}) = 1 \implies g(\mathbf{x}) = 1, \forall \mathbf{x} \in \{0, 1\}^n$. Em particular, qualquer produto p que faz parte da forma SOP de uma função f implica f e é denominado “implicante de f ”.

Cubos. Na função $f(a, b, c) = abc + \bar{a}bc$, os mintermos na notação compacta são m_7 (pois $111_{(2)} = 7_{(10)}$) e m_3 (pois $011_{(2)} = 3_{(10)}$). Assim, é usual escrevermos $f(a, b, c) = \sum m(3, 7)$. Uma vez que existe uma correspondência um-para-um entre os mintermos em n variáveis e os elementos de $\{0, 1\}^n$, uma função expressa como soma de mintermos pode ser vista como um subconjunto de $\{0, 1\}^n$.

Soma de mintermos	subconjunto de $\{0, 1\}^3$
$\bar{a}bc$	$\{011\}$
abc	$\{111\}$
$\bar{a}bc + abc$	$\{011, 111\}$

Os elementos de $\{0, 1\}^n$ (aqui denotados como sequências ou *strings* de n dígitos binários) podem ser vistos como pontos no n -espaço. A coleção dos 2^n elementos de $\{0, 1\}^n$ forma os vértices de um hipercubo. A figura 1 mostra um hipercubo no espaço de dimensão 3.

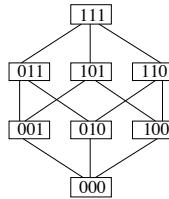


Figura 1: Um hipercubo de dimensão 3. (que feio, preciso melhorar isso!)

No contexto de circuitos lógicos, hipercubos são denominados **n -cubos**. Os vértices de um n -cubo são denominados 0-cubos. Dois 0-cubos formam um 1-cubo se eles diferem em apenas uma coordenada. Quatro 0-cubos formam um 2-cubo se eles são iguais a menos de duas coordenadas. De modo geral, 2^k 0-cubos formam um k -cubo se eles são exatamente iguais a menos de k coordenadas.

Intervalo. Observe que cubos não são subconjuntos arbitrários de $\{0, 1\}^n$. Um cubo é um conjunto de elementos em $\{0, 1\}^n$ para os quais um produto toma valor 1, i.e., se p é um produto então o cubo correspondente a p é o conjunto $p\langle 1 \rangle = \{\mathbf{x} \in \{0, 1\}^n : p(\mathbf{x}) = 1\}$.

Cubos podem também ser vistos como **intervalos**. Um intervalo no n -cubo é caracterizado por dois extremos: o menor e o maior elementos contidos nele. Assim, no cubo $\{0, 1\}^3$, o intervalo de extremo inferior 100 e extremo superior 101 é denotado $[100, 101]$ e definido¹ por $[100, 101] = \{\mathbf{x} \in \{0, 1\}^3 : 100 \leq \mathbf{x} \leq 101\}$.

¹A relação \leq usada na definição de intervalos não é a relação usual entre números. Por exemplo, “100” \leq “101”. Mas “011” não é menor que “101”; estes dois não são comparáveis. Formalmente, definimos que $(a_1, a_2, \dots, a_n) \leq (b_1, b_2, \dots, b_n)$ se, e somente se, $a_i \leq b_i, \forall i = 1, 2, \dots, n$.

Denotamos um k -cubo ou intervalo de dimensão k colocando um X nas coordenadas que não são iguais. Assim, no caso de três variáveis a , b e c , o 1-cubo $\{000, 100\}$ (ou, equivalentemente, o intervalo $[000, 100]$), que corresponde ao produto $\bar{b}\bar{c}$, é representado por $X00$. O 2-cubo $\{000, 001, 100, 101\}$ (ou, equivalentemente, o intervalo $[000, 101]$), que corresponde ao produto \bar{b} , é representado por $X0X$. Um intervalo contém necessariamente 2^k elementos, onde $0 \leq k \leq n$. Quanto maior a dimensão de um cubo, menor o número de literais presentes no correspondente produto.

Exemplo: A figura 2 mostra alguns cubos. Dizemos que o 0-cubo 000 está *contido* no (ou é coberto pelo) 1-cubo $X00$, ou ainda, que o 1-cubo $X00$ *cobre* o 0-cubo 000. Analogamente, dizemos que o 1-cubo $X00$ está contido no 2-cubo $X0X$ ou que o 2-cubo $X0X$ cobre o cubo $X00$.

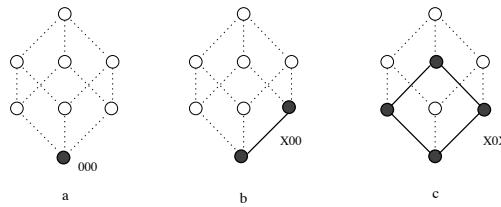


Figura 2: Os cubos 000, $X00$ e $X0X$.

Produto, implicante, cubo, e intervalo. Esses termos são equivalentes. Um produto canônico (também chamado mintermo), é equivalente a 0-cubo ou intervalo trivial. Por exemplo, dadas três variáveis lógicas, a , b e c , o produto canônico $a\bar{b}c$ corresponde ao 0-cubo 101, ou ao intervalo $[101, 101]$. Já o produto (não canônico) $a\bar{b}$ corresponde ao 1-cubo $10X$ ou ao intervalo $[100, 101]$.

Implicantes primos. Um produto ou implicante p de f é maximal se não existe outro produto ou implicante p' de f tal que p seja também implicante de p' . Implicantes maximais são denominados implicantes primos.

No caso do mapa de Karnaugh, os agrupamentos de 1's que construímos no mapa são implicantes primos (produtos maximais).

2 Minimização Tabular de Quine-McCluskey

Definição: Uma expressão lógica escrita na forma soma de produtos é **minimal** se

1. não existe nenhuma outra expressão equivalente² na forma soma de produtos com um número menor de termos, e
2. não existe nenhuma outra expressão equivalente na forma soma de produtos com igual número de termos mas com menor número de literais.

Denominaremos tal expressão de **SOP minimal**³. Ao se remover, de uma forma SOP minimal, um produto ou um literal de qualquer um dos produtos, a expressão resultante não mais representa a mesma função.

O problema pode ser analogamente definido para a forma POS. O problema de encontrar a forma SOP ou POS minimal de uma função é denominada **minimização lógica dois-níveis** devido ao fato de funções na forma SOP ou POS poderem ser realizadas em circuitos de dois níveis (isto é, no caso da forma SOP, o primeiro nível é formado pelas portas E, uma para cada produto, e o segundo nível consiste de uma porta OU que recebe como entradas as saídas das portas E).

Mapas de Karnaugh, vistos no outro documento, representam uma maneira visual e intuitiva de se minimizar funções booleanas. No entanto, eles só se aplicam a funções com até 6 variáveis e não são sistemáticos (adequados para programação). O algoritmo tabular de Quine-McCluskey para minimização de funções Booleanas é um método clássico que sistematiza este processo de minimização para um número arbitrário de variáveis.

Tanto os mapas de Karnaugh como o algoritmo de Quine-McCluskey (QM), em sua formulação clássica, requerem que a função booleana a ser minimizada esteja na forma SOP canônica. A idéia básica do algoritmo QM consiste em encarar os mintermos da SOP canônica como pontos no n -espaço, ou seja, como vértices de um n -cubo. A partir do conjunto desses vértices (ou 0-cubos) procura-se gerar todos os 1-cubos possíveis combinando-se dois deles (equivale a gerar as arestas do cubo que ligam dois 0-cubos da função). A partir da combinação de dois 1-cubos procura-se gerar todos os possíveis 2-cubos e assim por diante, até que nenhum cubo de dimensão maior possa ser gerado a partir da combinação de dois cubos de dimensão menor. Os cubos resultantes (aqueles que não foram combinados com nenhum outro) ao final de todo o processo são os **implicantes primos** (ou seja, cubos maximais) da função.

Esse processo de combinar dois cubos pode ser facilmente associado ao processo algébrico de simplificação. Os mintermos da expressão na forma canônica inicial corres-

²Duas expressões são equivalentes se definem uma mesma função.

³SOP é contração de *Sum of products*

pondem aos 0-cubos. Combinar dois 0-cubos para gerar um 1-cubo corresponde a combinar dois mintermos para eliminar uma variável e gerar um termo com menos literais para substituí-los, como mostramos no seguinte exemplo :

$$x_1x_2x_3 + x_1x_2\bar{x}_3 = x_1x_2(x_3 + \bar{x}_3) = x_1x_2 \cdot 1 = x_1x_2$$

Quando considerados no 3-espacão, o processo mostrado na expressão algébrica acima corresponde ao processo de agruparmos os 0-cubos 111 e 110 para geração do 1-cubo 11X, como ilustra a figura 3.

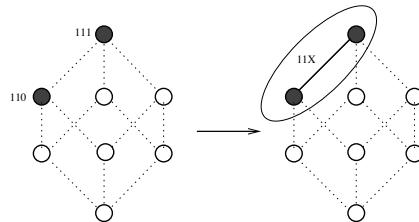


Figura 3: Passo elementar do algoritmo de Quine-McCluskey

À primeira vista, poderíamos afirmar que a soma de todos os implicantes primos corresponde à expressão minimal da função Booleana. No entanto, existem casos em que a soma de dois ou mais implicantes primos cobre um outro. Neste caso, este último termo é redundante, no sentido de que ele pode ser eliminado do conjunto de implicantes primos, sem que a expressão resultante deixe de ser equivalente à expressão original. Podemos ilustrar esta situação no seguinte exemplo.

Exemplo: Considere a expressão Booleana $f(a, b, c) = \sum m(0, 1, 3, 7)$. Os implicantes primos dessa função são 00X, 0X1 e X11 (calcule usando o mapa de Karnaugh). Graficamente, estes implicantes primos (ou cubos) correspondem respectivamente aos intervalos [000, 001], [001, 011] e [011, 111] ilustrados na figura 4(a). Note, porém, que o

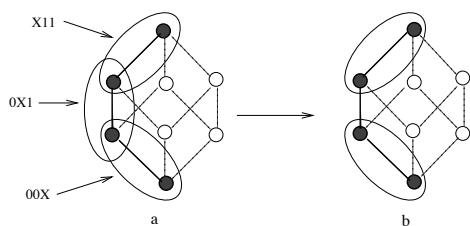


Figura 4: Os (a) implicantes primos e uma (b) cobertura mínima .

intervalo $[001, 011]$ é redundante, ou seja, a mesma expressão pode ser expressa apenas pelos implicantes primos $00X$ e $X11$ (figura 4(b)).

Teorema: Qualquer produto em uma expressão minimal na forma soma de produtos é um implicantе primo.

Dem.: A prova deste teorema é simples. Suponha que exista algum produto p na expressão que não seja um implicantе primo. Por definição, existe um produto p' tal que $p < p'$ e tal que p' implica a função. Então, ao substituirmos p por p' na expressão, obtemos uma expressão equivalente, porém com custo menor. Isto contradiz com o fato de que a expressão era minimal. \square

Este teorema diz, em outras palavras, que para encontrarmos uma expressão minimal de uma função, basta considerarmos apenas os produtos que são implicantes primos da função. Porém, como nem todos os implicantes primos de uma função f fazem parte da forma SOP minimal de f , é ainda necessário selecionar o “menor subconjunto” de implicantes primos suficientes para expressar f . Portanto, um procedimento para obter a forma SOP minimal de uma função pode ser:

1. Calcular todos os implicantes primos da função
2. Calcular uma cobertura mínima dos 1's da função (menor subconjunto de implicantes primos cuja soma representa a função)

O ponto central da segunda etapa é o cálculo de um menor subconjunto do conjunto de implicantes primos suficientes para cobrir⁴ todos os mintermos da função. Tal conjunto é denominado uma **cobertura mínima**.

No caso de mapas de Karnaugh, estas duas etapas são realizadas conjuntamente de forma um tanto “intuitiva”. No caso do algoritmo QM, estas etapas são realizadas explícita e separadamente. As etapas são descritas a seguir, por meio de um exemplo.

2.1 Cálculo de implicantes primos

A primeira etapa do algoritmo QM consiste de um processo para determinação de todos os implicantes primos. A seguir descrevemos os passos que constituem esta etapa, mostrando

⁴Um conjunto de implicantes primos (cubos máximos) cobre um mintermo (0-cubo) se este é coberto por pelo menos um dos implicantes primos.

como exemplo o cálculo dos implicantes primos da função $f(x_1, x_2, x_3) = \sum m(0, 1, 4, 5, 6)$.

- Primeiro passo : converter os mintermos para a notação binária.

000, 001, 100, 101, 110

- Segundo passo : Separar os mintermos em grupos de acordo com o número de 1's em sua representação binária e ordená-los em ordem crescente, em uma coluna, separando os grupos com uma linha horizontal.

000
001
100
101
110

- Terceiro passo : combinar todos os elementos de um grupo com todos os elementos do grupo adjacente inferior para geração de cubos de dimensão maior. Para cada 2 grupos comparados entre si, gerar um novo grupo na próxima coluna e colocar os novos cubos. Marcar com \checkmark os cubos que foram usados para gerar novos cubos.

\checkmark	000
\checkmark	001
\checkmark	100
\checkmark	101
\checkmark	110

 \Rightarrow

00X	
X00	
\checkmark	X01
10X	
1X0	

Observação : o novo cubo gerado será inserido no novo conjunto se e somente se ele ainda não estiver contido nele.

Repetir o processo para cada nova coluna formada, até que nenhuma combinação mais seja possível.

\checkmark	000
\checkmark	001
\checkmark	100
\checkmark	101
\checkmark	110

 \Rightarrow

\checkmark	00X
\checkmark	X00
\checkmark	X01
\checkmark	10X
	1X0

 \Rightarrow

X0X

- Quarto passo : Listar os implicantes primos. Os implicantes primos são os cubos que não foram combinados com nenhum outro, ou seja, aqueles que não estão com a marca \checkmark .

1X0 e X0X

2.2 Cálculo de uma cobertura mínima

Uma cobertura mínima pode ser calculada com a ajuda de uma tabela denominada **Tabela de Implicantes Primos**. Este processo é mostrado a seguir, usando como exemplo a minimização da função $f(x_1, x_2, x_3, x_4, x_5) = \sum(1, 2, 3, 5, 9, 10, 11, 18, 19, 20, 21, 23, 25, 26, 27)$.

1. Construir a Tabela de Implicantes Primos: No topo das colunas desta tabela deve-se colocar os mintermos de f e, à esquerda de cada linha, os implicantes primos. Os implicantes primos devem ser listados em ordem decrescente de acordo com a sua dimensão, isto é, em ordem crescente de acordo com o número de literais. Deve-se acrescentar uma coluna à esquerda e uma linha na parte inferior da tabela.

Em cada linha, marcar as colunas com \checkmark quando o implicante primo da linha cobre o mintermo da coluna.

		1	2	3	5	9	10	11	18	19	20	21	23	25	26	27
	XX01X		\checkmark	\checkmark			\checkmark	\checkmark	\checkmark	\checkmark					\checkmark	\checkmark
	X10X1					\checkmark		\checkmark							\checkmark	\checkmark
	0X0X1	\checkmark		\checkmark		\checkmark		\checkmark								
	00X01	\checkmark			\checkmark											
	X0101				\checkmark								\checkmark			
	1010X										\checkmark	\checkmark				
	10X11								\checkmark				\checkmark			
	101X1										\checkmark	\checkmark				

2. Selecionar os implicantes primos essenciais: deve-se procurar na tabela as colunas que contém apenas uma marca \checkmark . A linha na qual uma dessas colunas contém a marca \checkmark corresponde a um implicante primo essencial. Em outras palavras, este implicante primo é o único que cobre o mintermo da coluna e, portanto, não pode ser descartado. Então, deve-se marcar com um asterisco (*) esta linha na coluna mais à esquerda, para indicar que este é um implicante primo essencial. A seguir, deve-se marcar, na última linha da tabela, todas as colunas cujo mintermo é coberto pelo implicante primo selecionado.

No exemplo, o mintermo 2 é coberto apenas pelo implicante primo $XX01X$. Logo $XX01X$ é essencial.

		1	2	3	5	9	10	11	18	19	20	21	23	25	26	27
*	XX01X		✓	✓			✓	✓	✓	✓					✓	✓
	X10X1					✓		✓						✓		✓
	0X0X1	✓		✓		✓		✓								
	00X01	✓			✓											
	X0101				✓							✓				
	1010X										✓	✓				
	10X11								✓			✓				
	101X1										✓	✓				
			✓	✓			✓	✓	✓	✓					✓	✓

A linha correspondente a um implicante primo essencial, bem como as colunas cujos mintermos são cobertos por esse implicante primo, devem ser descondirados no prosseguimento do processo.

Deve-se repetir o processo enquanto existir, na tabela restante, algum implicante primo essencial.

No exemplo, vemos que o mintermo 25 é coberto apenas pelo implicante primo $X10X1$ e que o mintermo 20 é coberto apenas pelo implicante primo $1010X$. Logo, esses dois implicantes primos também são essenciais.

		1	2	3	5	9	10	11	18	19	20	21	23	25	26	27
*	XX01X		✓	✓			✓	✓	✓	✓					✓	✓
*	X10X1					✓		✓					✓		✓	
	0X0X1	✓		✓		✓		✓								
	00X01	✓			✓											
	X0101				✓							✓				
*	1010X										✓	✓				
	10X11								✓			✓				
	101X1										✓	✓				
			✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

3. Reducir a tabela: eliminar as colunas cujos mintermos já foram cobertos (ou seja, manter apenas as colunas correspondentes aos mintermos não cobertos pelos implicantes primos essenciais). Eliminar as linhas correspondentes aos implicantes primos essenciais e as linhas que não cobrem nenhum dos mintermos restantes na tabela.

No exemplo, após a redução, temos a seguinte tabela:

		1	5	23
	0X0X1	✓		
	00X01	✓	✓	
	X0101		✓	
	10X11			✓
	101X1			✓

4. Selecionar os implicantes primos secundariamente essenciais: eliminar as linhas dominadas e as colunas dominantes e, em seguida, selecionar os essenciais.

Colunas dominantes: Diz-se que uma coluna β na Tabela de Implicantes Primos domina uma coluna α se e somente se todos os implicantes que cobrem o mintermo da coluna α cobrem também o mintermo da coluna β . Se β domina α , então a coluna β pode ser removida da tabela. (Por quê?)

Linhas dominadas ou equivalentes: Sejam A e B duas linhas na Tabela de Implicantes Primos reduzida. Então dizemos que a linha A domina B se o implicante da linha A cobre, ao menos, todos os mintermos cobertos pelo implicante da linha B . Dizemos que as linhas A e B são equivalentes se os respectivos implicantes primos cobrem exatamente os mesmos mintermos na tabela. Se A domina B , ou se A e B são equivalentes, e **se o custo do implicante da linha A é menor ou igual ao da linha B** (ou seja, o implicante da linha A possui não mais literais que o implicante da linha B), então a linha B pode ser eliminada da tabela (Por quê?).

Após a eliminação de colunas dominantes e linhas dominadas (ou equivalentes), deve-se repetir o mesmo processo do passo 2, porém os implicantes primos essenciais serão chamados secundariamente essenciais e marcados com dois asteriscos (**).

No exemplo, a linha do implicante primo $X0101$ pode ser eliminada pois é dominada pela linha do implicante $00X01$. A linha do implicante $101X1$ pode ser eliminada pois é equivalente a do implicante $10X11$. Neste último caso, note que, alternativamente, podemos eliminar a linha do implicante $10X11$ em vez da linha do implicante $101X1$.

		1	5	23
	0X0X1	✓		
**	00X01	✓	✓	
**	10X11			✓
		✓	✓	✓

Deve-se repetir o processo descrito neste passo até que não seja mais possível fazer qualquer eliminação ou até que a tabela fique vazia. Se a tabela não ficar vazia, a tabela restante é chamada **tabela cíclica**.

5. Resolver a tabela cíclica: Para isso, uma possível abordagem é o método de Petrick, um método de busca exaustiva. Ele fornece todas as possíveis combinações dos implicantes primos restantes que são suficientes para cobrir os mintermos ainda não cobertos pelos implicantes primos já selecionados. Deve-se escolher, dentre essas combinações, uma que envolve o menor número de termos. Caso existam mais de uma nestas condições, deve-se escolher a de custo mínimo (aquele que envolve menor número de literais).

Exemplo: Considere a tabela cíclica a seguir:

		0	4	13	15	10	26	16
a	0X10X		✓	✓				
b	011XX			✓	✓			
c	01X1X				✓	✓		
d	1X0X0						✓	✓
e	00X00	✓	✓					
f	X1010					✓	✓	
g	X0000	✓						✓

Para que todos os mintermos da tabela cíclica sejam cobertos, a seguinte expressão deve ser verdadeira.

$$(e + g)(a + e)(a + b)(b + c)(c + f)(d + f)(d + g) = 1$$

Transformando esta expressão em soma de produtos, obtemos todas as possíveis soluções (cada produto é uma solução viável). Dentre os produtos, deve-se escolher aquele(s) de menor custo (menor número de implicantes primos e implicantes com menor número de literais). (Se eu não errei nos cálculos, as soluções são $\{a, c, d, e\}$, $\{b, c, d, e\}$ e $\{a, c, d, g\}$ pois os outros tem custo maior).

Outro exemplo: Considere a tabela cíclica a seguir e suponha que o custo de A é menor que o de B e que o custo de C é menor que o de D .

	m_1	m_2	m_3
A	✓		
B	✓	✓	
C			✓
D		✓	✓

Então as possíveis soluções são $(A + B)(B + D)(C + D) = (AB + AD + B + BD)(C + D) = (B + AD)(C + D) = BC + BD + ACD + AD$. Dos que envolvem dois implicantes, certamente o custos de BC e de AD são menores que o custo de BD . Então a escolha final fica entre BC e AD .

Resumo do Procedimento para cálculo de cobertura mínima:

1. Montar a tabela de implicantes primos
2. Identificar todos os implicantes primos essenciais e eliminar as linhas correspondentes, bem como as colunas dos mintermos cobertos por esses implicantes.
3. Eliminar colunas dominantes: Se uma coluna β tem \checkmark em todas as linhas que uma outra coluna α tem \checkmark , a coluna β é dominante e pode ser eliminada (pois se escolhermos um implicante primo que cobre α , β será necessariamente coberto também).
4. Eliminar linhas dominadas ou equivalentes: se uma linha A tem \checkmark em todas as colunas em que a linha B tem \checkmark , então a linha A domina a linha B . Se elas tem \checkmark exatamente nas mesmas colunas, então elas são equivalentes. Se, além disso, o número de literais de A é menor que o de B , então a linha B pode ser eliminada (pois se tivéssemos uma cobertura envolvendo B , ao trocarmos B por A na cobertura teríamos uma cobertura de menor custo).

Observação: Se o objetivo da minimização é encontrar apenas UMA solução minimal (e NÃO TODAS), então podemos eliminar uma linha B se existe uma linha A tal que A domina B , ou A é equivalente a B , e ambos têm um mesmo custo.

5. Identificar os implicantes essenciais secundários e eliminar as linhas correspondentes, bem como as colunas dos mintermos cobertos por esses implicantes.
6. Repetir 3, 4 e 5 enquanto possível
7. Se a tabela não estiver vazia, aplicar o método de Petrick (que lista todas as possíveis soluções para o restante da tabela) e escolher uma solução de custo mínimo.

Exemplo: Considere a função $f(a, b, c) = a\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + ab\bar{c} = \sum m(2, 5, 6, 7)$.

Podemos realizar a simplificação algébrica da seguinte forma:

$$\begin{aligned}
 f(a, b, c) &= a\bar{b}c + \bar{a}b\bar{c} + ab\bar{c} + abc \\
 &= a\bar{b}c + abc + \bar{a}b\bar{c} + ab\bar{c} + a\bar{b}\bar{c} + abc \\
 &= ac + b\bar{c} + ab \\
 &= ac + b\bar{c}
 \end{aligned}$$

Por QM temos

✓	010
✓	101
✓	110
✓	111

⇒

X10
1X1
11X

Os implicants primos são $X10$ ($b\bar{c}$), $1X1$ (ac) e $11X$ (ab). Uma cobertura mínima pode ser calculada usando-se a Tabela de Implicantes Primos.

		2	5	6	7
*	X10	✓		✓	
*	1X1		✓		✓
	11X			✓	✓
		✓	✓	✓	✓

Os implicants primos $X10$ e $1X1$ são essenciais e cobrem todos os mintermos da função. Logo formam uma cobertura mínima.

2.3 Funções incompletamente especificadas

Em algumas situações, o valor de uma função para algumas entradas não são relevantes (tipicamente porque tais entradas nunca ocorrerão na prática). Em tais situações, tanto faz se a função toma valor 0 ou 1 nessas entradas, que serão denominadas de **don't cares**.

Para minimizar uma função incompletamente especificada pelo algoritmo QM, é interessante considerarmos todos os don't cares na etapa de cálculo dos implicants primos, pois isto aumenta a chance de obtermos cubos maiores (portanto produtos com menos literais). Observe que, durante as iterações para a geração dos implicants primos, um

cubo que cobre apenas don't cares não deve ser eliminado pois ele pode, eventualmente em iterações futuras, se juntar a outro cubo para formar outro cubo maior.

De forma mais genérica do que a vista anteriormente, podemos caracterizar uma função booleana f através dos seus conjuntos um, zero e dc (de don't care), definidos respectivamente por $f\langle 1 \rangle = \{\mathbf{x} \in \{0, 1\}^n : f(\mathbf{x}) = 1\}$, $f\langle 0 \rangle = \{\mathbf{x} \in \{0, 1\}^n : f(\mathbf{x}) = 0\}$ e $f\langle * \rangle = \{0, 1\}^n \setminus (f\langle 1 \rangle \cup f\langle 0 \rangle)$.

Na parte de cálculo dos implicantes primos devem ser utilizados todos os elementos de $f\langle 1 \rangle \cup f\langle * \rangle$. Na parte de cálculo de uma cobertura mínima devem ser considerados apenas os elementos de $f\langle 1 \rangle$.

2.4 Cálculo da forma POS minimal

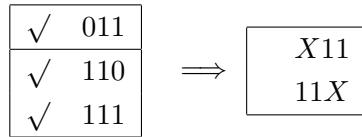
Similarmente ao que já vimos na minimização por mapas de Karnaugh, para se obter a forma POS minimal de uma função por QM procede-se da seguinte forma. No cálculo dos implicantes primos, em vez de listar os mintermos, listamos os 0s da função e aplicamos o método tabular (a primeira parte do algoritmo QM). Em seguida, realizamos o cálculo da cobertura mínima utilizando-se nas colunas os 0s da função. Ao final, expressa-se o resultado como produto dos implicantes primos selecionados complementados.

A explicação é a seguinte: ao tomarmos os 0s da função em vez dos 1s, estamos considerando a minimização SOP de \bar{f} . Agora, uma vez que $f = \bar{\bar{f}}$, ao complementarmos a forma SOP minimal de \bar{f} , obtemos a forma POS minimal de f .

Exemplo: Minimizar na forma POS a função $f(a, b, c) = \prod M(3, 6, 7)$. Algebricamente temos:

$$\begin{aligned} f(a, b, c) &= (a + \bar{b} + \bar{c})(\bar{a} + \bar{b} + c)(\bar{a} + \bar{b} + \bar{c}) \\ &= (a + \bar{b} + \bar{c})(\bar{a} + \bar{b} + \bar{c})(\bar{a} + \bar{b} + c)(\bar{a} + \bar{b} + \bar{c}) \\ &= (a\bar{a} + \bar{b} + \bar{c})(\bar{a} + \bar{b} + c\bar{c}) \\ &= (\bar{b} + \bar{c})(\bar{a} + \bar{b}) \end{aligned}$$

Por QM temos:



Os implicantes são $X11$ e $11X$, que escritos na forma de produtos correspondem respectivamente a bc e ab . Complementando estes produtos temos: $\bar{b} + \bar{c}$ e $\bar{a} + \bar{b}$, que são as somas que aparecem na forma POS minimal.

3 Minimização dois-níveis de múltiplas funções*

No outro documento vimos brevemente a ideia de compartilhar portas E quando existem duas ou mais funções a serem realizadas, definidas sobre as mesmas variáveis. Vimos também que, nessas condições, não necessariamente a minimização individual das funções leva a uma melhor solução. Isto é, há situações nas quais minimizar individualmente as funções impede o compartilhamento de portas E, embora existam produtos que sejam implicantes de mais de uma função.

Quando pensamos em minimizar um conjunto de funções, diferentes critérios podem ser considerados. Entre eles, alguns são intuitivos:

- reduzir o número total de produtos (ou seja, portas E).
- reduzir o número de entradas para as portas E (tentar usar produtos com o menor número possível de literais)
- reduzir o número de entradas para as portas OU (ou seja, utilizar o menor número possível de produtos para cada função)

Quando consideramos a minimização de múltiplas funções, podemos aplicar um processo análogo ao do algoritmo QM. A noção de implicante primo é agora definida com relação às múltiplas funções. Devemos considerar não apenas os implicantes primos de cada uma das funções, mas também todos os implicantes maximais que podem ser compartilhados por 2 ou mais funções. Vamos esclarecer isso analisando um exemplo concreto.

Exemplo 3: Considere as funções, escritas na forma SOP canônica

$$f_1(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 x_3 + x_1 \bar{x}_2 x_3 = \sum m(0, 1, 5)$$

$$f_2(x_1, x_2, x_3) = x_1 \bar{x}_2 x_3 + x_1 x_2 \bar{x}_3 + x_1 x_2 x_3 = \sum m(5, 6, 7)$$

Escritos na notação cúbica, os mintermos de f_1 são 000, 001 e 101 enquanto os de f_2 são 101, 110 e 111. Os implicantes primos (com respeito a f_1 e f_2) são:

- a) os implicantes primos de f_1 : $00X$ e $X01$
- b) os implicantes primos de f_2 : $1X1$ e $11X$
- c) os implicantes de f_1 e de f_2 que não são cobertos por outro implicant de f_1 e de f_2 : 101 (ou seja, 101 é implicant (não é primo!) de ambas as funções e não existe nenhum outro implicant de f_1 e de f_2 que o cobre).

Na minimização de múltiplas funções, compartilhar produtos é uma forma de se minimizar o número de portas E no circuito. Os implicantes do tipo (c) são, em outras palavras, os maiores subcubos compartilhados entre duas ou mais funções. Portanto, o critério de minimização deve levar em consideração não apenas os implicantes primos de cada função, mas também todos os implicantes do tipo do item (c).

O algoritmo QM adaptado para o cálculo de implicantes primos de múltiplas funções

Considere novamente as funções do exemplo 1: $f_1(x_1, x_2, x_3) = \sum m(0, 1, 5)$ e $f_2(x_1, x_2, x_3) = \sum m(5, 6, 7)$. A tabela 1 mostra o cálculo dos implicantes primos de $\mathbf{f} = (f_1, f_2)$. Inicialmente, listam-se todos os mintermos (independente da função a qual eles pertencem), um em cada linha, agrupados de acordo com o número de ocorrência de 1s. Ao lado, cria-se uma coluna para cada uma das funções e, para cada coluna, marcam-se as linhas correspondentes aos mintermos da função. Assim, por exemplo, na coluna correspondente à função f_1 aparece 1 nas linhas dos mintermos 000, 001 e 101.

Para gerar os implicantes primos, procede-se de forma análoga ao da minimização de uma única função, tomando-se cuidado para (1) combinar somente os pares de subcubos que fazem parte de pelo menos uma mesma função e (2) quando dois cubos C e C' são combinados e um novo cubo C'' é gerado, marcar o subcubo C para descarte somente se o cubo gerado C'' é também cubo das mesmas funções das quais C é cubo (idem para C'). Por exemplo, 001 pode ser combinado com 101 para gerar o cubo $X01$. Neste momento, o cubo 001 (da função f_1) pode ser descartado pois ele é coberto por $X01$ (que também é cubo da função f_1). No entanto, o cubo 101 (das funções f_1 e f_2) NÃO pode ser descartado pois o cubo $X01$ que o cobre não é cubo da função f_2 . Quando dois cubos

são combinados e um novo cubo é gerado, ele deve ser colocado em uma outra tabela (mais a direita na figura 1), indicando-se a qual das funções ele pertence. O processo deve ser repetido até que nenhuma combinação seja mais possível. Ao final do processo, os implicantes candidatos são aqueles que não foram marcados para descarte ao longo do processo. No exemplo da figura 1 são aqueles marcados por a, b, c, d e e .

$x_1x_2x_3$	f_1	f_2
000	1	✓
001	1	✓
101	1	e
110	1	✓
111	1	✓

$x_1x_2x_3$	f_1	f_2
00X	1	a
X01	1	b
1X1		c
11X		d

Tabela 1: Método tabular para cálculo dos implicantes primos de $\mathbf{f} = (f_1, f_2)$.

O segundo passo do algoritmo é a seleção de uma cobertura mínima. A tabela 2 mostra a **tabela de implicantes primos** de $\mathbf{f} = (f_1, f_2)$, utilizada para a seleção de uma cobertura mínima. A tabela contém colunas correspondentes a cada um dos mintermos de cada uma das funções, e linhas correspondentes aos implicantes primos calculados no passo anterior. Os números ao lado esquerdo de um implicante primo indicam que o implicante é das funções com os respectivos números. Por exemplo, na coluna ao lado esquerdo de $00X$ aparece (1) para indicar que $00X$ é um implicantе da função f_1 ; ao lado de 101 aparece (1, 2) para indicar que 101 é implicantе de f_1 e de f_2 . Para cada linha, marcam-se com \checkmark as colunas correspondentes aos mintermos cobertos pelo implicantе primo, **desde que o implicantе primo seja implicantе da função correspondente ao mintermo** (por exemplo, $X01$ cobre 101 , mas não se marca na coluna 101 da função f_2 pois $X01$ não é implicantе de f_2).

Após a construção da tabela, deve-se primeiramente selecionar os implicantes primos essenciais, que são marcados com * ($00X$ e $11X$). Procede-se com a redução da tabela, eliminando-se a linha dos essenciais, bem como as colunas dos mintermos cobertos por eles. Após a redução da tabela, obtemos a tabela da direita da figura 2.

Se levarmos em consideração apenas a minimização do número de produtos, podemos dizer que o implicantе (e) domina os implicantes (b) e (c). Portanto, as linhas (b) e (c) podem ser eliminadas, resultando apenas a linha (e). Temos então o resultado $00X, 11X$ e 101 .

No entanto, se levarmos em conta também, além da minimização do número de produtos, o número de literais em cada produto, não podemos dizer que a linha (e)

		f_1			f_2		
		000	001	101	101	110	111
*	1	(a) 00X	✓	✓			
1		(b) X01		✓	✓		
2		(c) 1X1			✓		✓
*	2	(d) 11X				✓	✓
1,2		(e) 101			✓	✓	
			✓	✓		✓	✓

		f_1	f_2
		101	101
1	(b) X01	✓	
2	(c) 1X1		✓
1,2	(e) 101	✓	✓

Tabela 2: Tabela de implicantes primos de $\mathbf{f} = (f_1, f_2)$.

domina as outras duas. Neste caso, temos uma tabela cíclica e utilizaremos o método de Petrick para resolvê-lo.

Para cobrir ambas as colunas, a seguinte igualdade deve ser verdadeira:

$$(b + e)(c + e) = 1$$

Escrevendo a expressão acima na forma SOP, temos

$$bc + be + ce + e = 1$$

Daqui podemos concluir que a solução de menor custo é escolher (e). Assim, temos o resultado 00X, 11X e 101 (por coincidência, o mesmo obtido considerando o custo mais simples).

Cálculo de implicantes primos de múltiplas funções usando mapas de Karnaugh

Exemplo 5: Considere as seguintes funções.

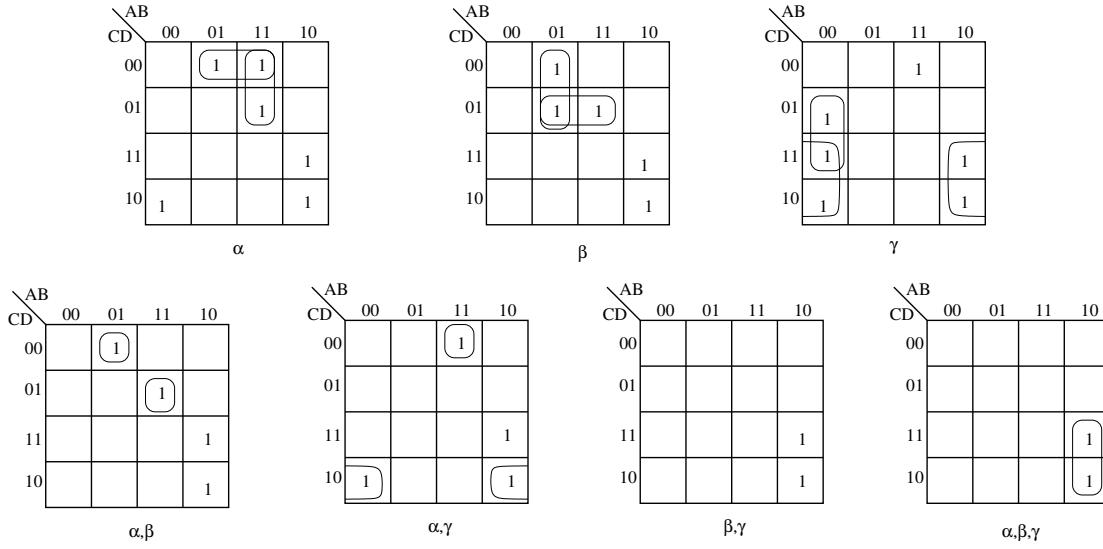
$$f_\alpha(a, b, c, d) = \sum m(2, 4, 10, 11, 12, 13)$$

$$f_\beta(a, b, c, d) = \sum m(4, 5, 10, 11, 13)$$

$$f_\gamma(a, b, c, d) = \sum m(1, 2, 3, 10, 11, 12)$$

O processo de cálculo dos implicantes primos pode ser realizado usando os mapas de Karnaugh. Na figura 5 aparecem 7 mapas de Karnaugh, das funções f_α , f_β , f_γ , $f_\alpha \cdot f_\beta$, $f_\alpha \cdot f_\gamma$, $f_\beta \cdot f_\gamma$ e $f_\alpha \cdot f_\beta \cdot f_\gamma$, respectivamente.

Começa-se marcando os implicantes primos (cubos maximais) no mapa da função $f_\alpha \cdot f_\beta \cdot f_\gamma$. O único implicante primo de $f_\alpha \cdot f_\beta \cdot f_\gamma$ é 101X. Em seguida, marcam-se os implicantes primos da interseção de duas funções, desde que os mesmos não sejam

Figura 5: Implicantes primos da função $f = (f_\alpha, f_\beta, f_\gamma)$.

cobertos pelos implicantes de um produto de funções de ordem maior. Por exemplo, em $f_\alpha \cdot f_\gamma$, $101X$ é um implicante primo mas ele é coberto pelo implicante primo $101X$ de $f_\alpha \cdot f_\beta \cdot f_\gamma$. Portanto, este implicante primo não é marcado. Repete-se o processo para cada uma das funções, marcando-se apenas os implicantes primos que não aparecem em nenhuma das funções $f_\alpha \cdot f_\beta$, $f_\alpha \cdot f_\gamma$, $f_\beta \cdot f_\gamma$ e $f_\alpha \cdot f_\beta \cdot f_\gamma$.

Assim, os implicantes obtidos são os mostrados na tabela a seguir. A coluna da esquerda indica as funções implicadas pelo implicante.

$101X$	$\alpha \beta \gamma$
$X010$	$\alpha \gamma$
1100	$\alpha \gamma$
0100	$\alpha \beta$
1101	$\alpha \beta$
$X01X$	γ
$00X1$	γ
$X101$	β
$010X$	β
$110X$	α
$X100$	α

O método tabular para determinação dos implicantes primos é mostrado na tabela 3.

	f_α	f_β	f_γ	IP		f_α	f_β	f_γ	IP	
0001			1	✓	00X1			1	f	
0010	1		1	✓	001X			1	✓	
0100	1	1		i	X010	1		1	g	
0011			1	✓	010X			1	d	
0101		1		✓	X100	1			b	
1010	1	1	1	✓	X011			1	✓	
1100	1		1	j	X101			1	e	
1011	1	1	1	✓	101X	1	1	1	h	
1101	1	1		k	110X	1			c	

X01X	f_α	f_β	f_γ	IP
				a

Tabela 3: Método tabular para cálculo dos implicantes primos de $\mathbf{f} = (f_\alpha, f_\beta, f_\gamma)$.

Compare os implicantes primos obtidos pelos mapas de Karnaugh e pelo método QM adaptado.

A tabela de implicantes primos é mostrada na figura 4. Primeiramente são identificados os implicantes essenciais.

			f_α						f_β					f_γ					
			2	4	10	11	12	13	4	5	10	11	13	1	2	3	10	11	12
	γ	(a) X01X													✓	✓	✓	✓	✓
	α	(b) X100		✓			✓												
	α	(c) 110X				✓	✓												
	β	(d) 010X							✓	✓									
	β	(e) X101								✓			✓						
*	γ	(f) 00X1												✓	✓				
*	$\alpha\gamma$	(g) X010	✓		✓										✓		✓		
*	$\alpha\beta\gamma$	(h) 101X			✓	✓					✓	✓					✓	✓	
	$\alpha\beta$	(i) 0100		✓						✓									
*	$\alpha\gamma$	(j) 1100					✓												✓
	$\alpha\beta$	(k) 1101						✓					✓						
			✓	✓	✓	✓	✓				✓	✓		✓	✓	✓	✓	✓	✓

Tabela 4: Tabela dos implicantes primos e seleção de uma cobertura mínima de $\mathbf{f} = (f_\alpha, f_\beta, f_\gamma)$.

Obtemos os **essenciais**: f, g, h, j

Caso 1: Minimizar APENAS o número de produtos (implementação em PLA)

			f_α		f_β		
			4	13	4	5	13
	α	(b) X100	✓				
	α	(c) 110X		✓			
	β	(d) 010X			✓	✓	
	β	(e) X101				✓	✓
**	$\alpha\beta$	(i) 0100	✓		✓		
**	$\alpha\beta$	(k) 1101		✓			✓
			✓	✓	✓	✓	✓

- o número de literais presentes nos implicantes não é importante: b e c podem ser eliminados pois são dominados por i e k , respectivamente.
- i e k são essenciais secundários.
- Para cobrir 5 podemos selecionar d ou e .

RESULTADO (ao selecionarmos d)

$$f_\alpha : g + h + i + j + k = X010 + 101X + 0100 + 1100 + 1101$$

$$f_\beta : d + h + i + k = X101 + 101X + 0100 + 1101$$

$$f_\gamma : f + g + h + j = 00X1 + X010 + 1100 + 101X$$

Note que na expressão de f_β , o termo 1101 é redundante e portanto pode ser eliminado. Assim temos $f_\beta = d + h + i = X101 + 101X + 0100$.

Caso 2: Minimizar número de produtos E número de literais nos produtos

			f_α		f_β		
			4	13	4	5	13
	α	(b) X100	✓				
	α	(c) 110X		✓			
	β	(d) 010X			✓	✓	
	β	(e) X101				✓	✓
	$\alpha\beta$	(i) 0100	✓		✓		
	$\alpha\beta$	(k) 1101		✓			✓

- a tabela acima é cíclica. Não podemos eliminar a linha (b) nem a (c) pois, embora elas sejam dominadas respectivamente pelas linhas (i) e (k), o custo dessas últimas é maior.

- devemos aplicar o método de Petrick

$$(b+i)(c+k)(d+i)(d+e)(e+k) = 1$$

que resulta em

$$cei + bcde + eik + dik + bdk$$

Desses, os de menor custo são cei e bdk

- Para cada função, selecionar o menor subconjunto que a cobre.

RESULTADO (ao selecionarmos cei)

$$f_\alpha : c + g + h + i = 110X + X010 + 101X + 0100$$

$$f_\beta : e + h + i = X101 + 101X + 0100$$

$$f_\gamma : f + g + h + j = 00X1 + X010 + 101X + 1100$$

Comparando os casos 1 e 2 acima, em ambos precisamos de 7 produtos. Há, no entanto, diferença no número de produtos na função f_α . Em PLA, a porta OU dessa função terá uma entrada a mais que no caso não-PLA.

Exemplo 6: minimizar as funções

$$f_1(a, b, c, d) = \sum m(3, 4, 5, 7, 9, 13, 15) + d(11, 14)$$

$$f_2(a, b, c, d) = \sum m(3, 4, 7, 9, 13, 14) + d(0, 1, 5, 15)$$

Os implicantes primos são mostrados na figura 6.

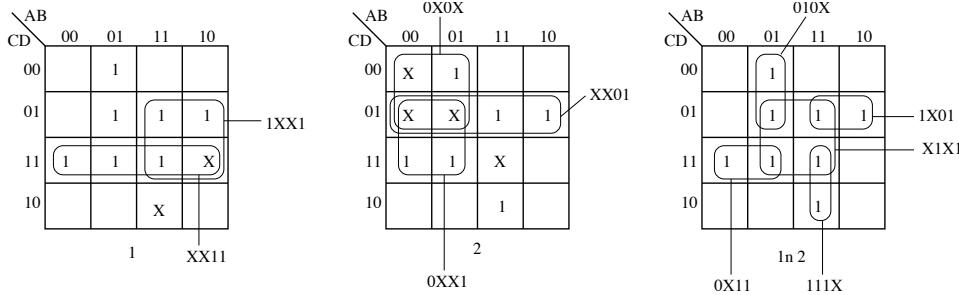


Figura 6: Implicantes primos da função $\mathbf{f} = (f_1, f_2)$.

Uma cobertura mínima pode ser encontrada com o auxílio da Tabela de Implicantes Primos. Os implicantes essenciais são $111X$ e $010X$.

			f_1							f_2						
			3	4	5	7	9	13	15	3	4	7	9	13	14	
	1	1XX1					✓	✓	✓							
	1	XX11	✓			✓			✓							
	2	XX01												✓	✓	
	2	0XX1								✓			✓			
	2	0X0X									✓					
*	1,2	111X							✓							✓
	1,2	0X11	✓			✓				✓			✓			
	1,2	1X01					✓	✓						✓	✓	
	1,2	X1X1			✓	✓		✓	✓				✓		✓	
*	1,2	010X		✓	✓						✓					
				✓	✓				✓		✓				✓	

Eliminando os essenciais e colunas, e as linhas vazias, temos a seguinte tabela reduzida. Se considerarmos implementação em PLA, podemos eliminar a linha dos implicants $XX11$ e $0XX1$ pois estes são dominados pela linha do implicant $0X11$. Similarmente, podemos eliminar as linhas dos implicants $1XX1$ e $XX01$.

A tabela resultante é a seguinte. Escolhendo-se os secundariamente essenciais obtém-se uma cobertura para todos os mintermos restantes na tabela.

Assim, a solução final é:

$$f_1 : 0X11, 1X01, 111X, 010X$$

$$f_2 : 0X11, 1X01, 111X, 010X$$

Ou seja, $f_1 = f_2$! (Por que isso aconteceu !??)

Exemplo 7: minimizar as funções

$$f_1(a, b, c, d) = \sum m(0, 2, 7, 10) + d(12, 15)$$

$$f_2(a, b, c, d) = \sum m(2, 4, 5) + d(6, 7, 8, 10)$$

$$f_3(a, b, c, d) = \sum m(2, 7, 8) + d(0, 5, 13)$$

O cálculo dos implicants pelo método tabular é mostrado na tabela 5. Os impli-

	f_1	f_2	f_3	IP		f_1	f_2	f_3	IP	
0000	1		1	✓		00X0	1	1	i	
0010	1	1	1	m		X000		1	h	
0100		1		✓		0X10			g	
1000	1	1	1			X010	1	1	f	
0101		1	1	✓		010X		1	✓	
0110		1		✓		01X0		1	✓	
1010	1	1		✓		10X0		1	e	
1100	1			k		01X1		1	d	
0111	1	1	1	j		X101			c	
1101			1	✓		011X		1	✓	
1111	1			✓		X111	1		b	

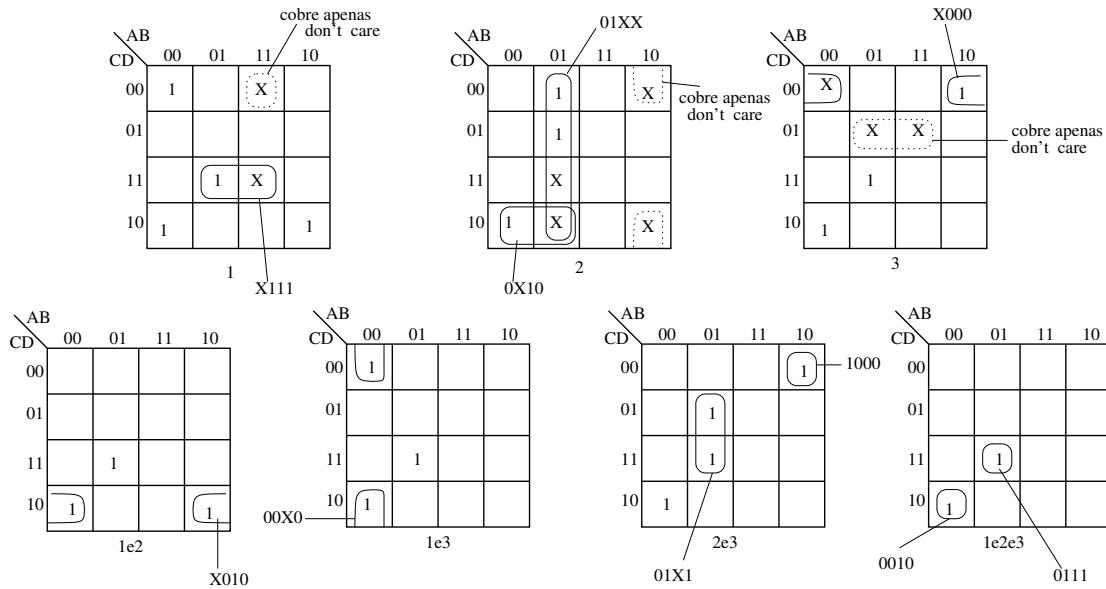
	f_1	f_2	f_3	IP
01XX		1		a

Tabela 5: Método tabular para cálculo dos implicants primos de $\mathbf{f} = (f_1, f_2, f_3)$.

cantes 1100, X101 e 10X0 cobrem apenas don't cares e podem ser desconsiderados na etapa de cálculo de cobertura mínima.

Os implicants podem também ser obtidos pelo mapa de Karnaugh. Veja a figura 7.

Uma cobertura mínima pode ser encontrada com o auxílio da Tabela de Implicants Primos, que é mostrada na seguinte tabela. Nesta mesma tabela, os implicants essenciais aparecem marcados por *.

Figura 7: Implicantes primos da função $f = (f_1, f_2, f_3)$.

			f_1				f_2			f_3		
			0	2	7	10	2	4	5	2	7	8
*	2	(a) 01XX						✓	✓			
	1	(b) X111			✓							
	23	(d) 01X1							✓		✓	
*	12	(f) X010		✓		✓	✓					
	2	(g) 0X10					✓					
	3	(h) X000										✓
*	13	(i) 00X0	✓	✓						✓		
	123	(j) 0111			✓						✓	
	23	(l) 1000					✓					✓
	123	(m) 0010		✓			✓			✓		
			✓	✓	✓	✓	✓	✓	✓	✓		

Eliminando-se a linha dos essenciais e as colunas dos mintermos cobertos por eles, obtém-se a seguinte tabela, onde a linha do implicante 1000 é dominada pela linha do implicante X000. Ao se eliminar a linha do implicante 1000, o implicante X000 torna-se essencial. Das três linhas que restam, é imediato verificar que a escolha que minimiza o custo é 0111, que será marcada com ***.

			f_1	f_3	
			7	7	8
	1	X111	✓		
	23	01X1		✓	
**	3	X000			✓
***	123	0111	✓	✓	
	23	1000			✓
			✓	✓	✓

← dominada pela linha do X000

Portanto obtemos:

$$f_1: X010, 00X0, 0111$$

$$f_2: 01XX, X010, 0111 \implies 01XX, X010 \text{ (pois } 0111 \text{ é coberto por } 01XX\text{)}$$

$$f_3: 00X0, X000, 0111.$$

4 Comentários

Algoritmos de minimização tabular (como o Quine-McCluskey) têm algumas desvantagens do ponto de vista computacional:

- eles listam explicitamente todos os implicantes primos, cuja quantidade pode ser de ordem exponencial no número de variáveis n
- requerem que a função a ser minimizada esteja na forma SOP canônica. Não é raro que, juntamente com os don't cares, o número de produtos canônicos seja da ordem de 2^{n-1}
- a tabela-cíclica pode ser bem grande. O cálculo da cobertura mínima é um problema computacionalmente difícil, no sentido de não haver solução eficiente.

Devido a isso, implementações existentes baseadas no algoritmo QM fazem uso de heurísticas (que produzem resultados sub-ótimos, mas muitas vezes bastante próximos aos resultados ótimos). Um dos algoritmos heurísticos bastante conhecidos é o ESPRESSO, desenvolvido na década de 1990 por um grupo da University of California – Berkeley [Brayton et al., 1984, McGreer et al., 1993]. ESPRESSO não calcula os implicantes primos explicitamente. Além disso, ele também realiza minimização de múltiplas funções e de funções multi-valoradas (lógica multi-valores).

Referências

- [Brayton et al., 1984] Brayton, R. K., Hachtel, G. D., McMullen, C. T., and Sangiovanni-Vincentelli, A. L. (1984). *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers.
- [McGreer et al., 1993] McGreer, P. C., Sanghavi, J., Brayton, R. K., and Sangiovanni-Vincentelli, A. L. (1993). Espresso-Signature : A New Exact Minimizer for Logic Functions. *IEEE trans. on VLSI*, 1(4):432–440.

Álgebra booleana

Última revisão em 06 de abril de 2016

As funções lógicas podem ser formalizadas pela Álgebra Booleana. Como qualquer sistema algébrico, a álgebra booleana considera um conjunto de elementos e operações definidas sobre esses elementos. O termo “booleana” é derivado do nome do matemático inglês George Boole (1815-1864), a quem é creditado a origem da álgebra booleana. A álgebra booleana é o resultado dos esforços de Boole para sistematizar o raciocínio lógico.

Neste documento veremos uma definição formal de álgebra booleana, que é baseada em um conjunto de axiomas (ou postulados). Veremos também algumas leis ou propriedades de álgebras booleanas e que todas essas leis podem ser derivadas algebricamente a partir dos postulados da definição. Vários exemplos de álgebras booleanas são listados. Dentro eles, merecem destaque a álgebra dos conjuntos e o cálculo proposicional.

Referências para esta parte do curso: [Hill and Peterson, 1981], [Garnier and Taylor, 1992], [Whitesitt, 1961] entre outros.

1 Definição axiomática de álgebra booleana

Seja uma sétupla $\langle A, +, \cdot, \bar{}, 0, 1 \rangle$ na qual A é um conjunto, $+$ e \cdot são operações binárias sobre A , $\bar{}$ é uma operação unária em A e 0 e 1 são dois elementos distintos em A . O sistema algébrico $\langle A, +, \cdot, \bar{}, 0, 1 \rangle$ é uma **álgebra booleana** se os seguintes axiomas são satisfeitos:

A1. As operações $+$ e \cdot são **comutativas**, ou seja, para todo x e y em A ,

$$x + y = y + x \quad \text{e} \quad x \cdot y = y \cdot x$$

A2. Cada operação é **distributiva** sobre a outra, isto é, para todo x , y e z em A ,

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z) \quad \text{e} \quad x + (y \cdot z) = (x + y) \cdot (x + z)$$

A3. Os elementos 0 e 1 são os **elementos identidades**, ou seja, para todo $x \in A$,

$$x + 0 = x \quad \text{e} \quad x \cdot 1 = x$$

A4. Todo elemento $x \in A$ possui um complemento, ou seja, existe um elemento \bar{x} tal que

$$x + \bar{x} = 1 \quad \text{e} \quad x \cdot \bar{x} = 0.$$

Observação 1: Na literatura encontramos outras definições para álgebra booleana. Em geral, as definições incorporam um maior número de propriedades. Vale registrar que os postulados acima apresentados, elaborados por Huntington em 1904, correspondem a um conjunto minimal de postulados, isto é, nenhum deles pode ser derivado a partir dos demais. Mais ainda, é um conjunto completo no sentido de que qualquer outra propriedade de uma álgebra booleana pode ser derivada a partir desses postulados. Desta forma, qualquer sistema algébrico que satisfaz os 4 axiomas acima é uma álgebra Booleana. Mais adiante mostraremos como a propriedade associativa (frequentemente incorporada à definição de álgebra booleana) e várias outras podem ser derivadas a partir dos postulados acima.

Observação 2: Pode-se fazer um paralelo com a álgebra elementar dos números. Por exemplo, sobre o conjunto dos números reais, define-se as operações de adição, subtração, etc. Essas operações satisfazem alguma propriedades (por exemplo, a adição é comutativa). Enquanto na álgebra booleana temos a noção de complemento, na álgebra elementar temos a noção de oposto (em relação à adição) e de inverso (em relação à multiplicação).

2 Exemplos de álgebra booleana

Exemplo 1: O conjunto $B = \{0, 1\}$ para o qual definimos

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

$$1 \cdot 1 = 1 + 1 = 1 + 0 = 0 + 1 = 1$$

$$0 + 0 = 0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$$

é uma álgebra booleana.

Os axiomas A1, A3 e A4 são satisfeitos por definição. Para verificar o axioma A2, dados três elementos quaisquer x, y e z em B , podemos construir uma tabela verdade para todas as possíveis combinações de valores para x, y e z . Vejamos, nas colunas indicadas com * na parte inferior da tabela, a validade da distributividade em relação a \cdot , ou seja, que $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

x	y	z	$(y + z)$	$x \cdot (y + z)$	$(x \cdot y)$	$(x \cdot z)$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1
			*				*

Denotamos esta álgebra booleana por $\langle B, +, \cdot, \bar{\cdot}, 0, 1 \rangle$. Esta é a álgebra que está por trás dos circuitos lógicos.

Exemplo 2: Dado um conjunto S , $\mathcal{P}(S)$ denota o conjunto das partes de S , isto é, $\mathcal{P}(S) = \{X : X \subseteq S\}$. Conforme já estamos familiarizados, as propriedades da álgebra de conjuntos equivalentes aos 4 postulados da definição de álgebra booleana são:

$$\text{A1. } X \cup Y = Y \cup X \quad \text{e} \quad X \cap Y = Y \cap X$$

$$\text{A2. } X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z) \quad \text{e} \quad X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$$

$$\text{A3. } \emptyset \cup X = X \quad \text{e} \quad U \cap X = X$$

$$\text{A4. } X \cap X^c = \emptyset \quad \text{e} \quad X \cup X^c = U$$

Então, $\langle \mathcal{P}(S), \cup, \cap, {}^c, \emptyset, S \rangle$ é uma álgebra booleana. Dessas propriedades, a única que pode não ser trivial é a A2. Para se convencer da validade dessas propriedades, pode-se recorrer aos diagramas de Venn.

No diagrama de Venn o conjunto universo é representado por um retângulo, mais precisamente, pelos pontos interiores ao retângulo. Qualquer conjunto é desenhado como sendo uma curva fechada, inteiramente contida no retângulo. Pontos interiores à curva correspondem aos elementos do conjunto. No exemplo da figura 1, a união e interseção de dois conjuntos genéricos estão representadas pelas regiões hachuradas das figuras 1a e 1b, respectivamente. O complemento de um conjunto é representado no diagrama da figura 1c.

Como exemplo, vamos verificar a propriedade $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$. O conjunto $X \cap (Y \cup Z)$ corresponde à região hachurada pelas linhas verticais e pelas

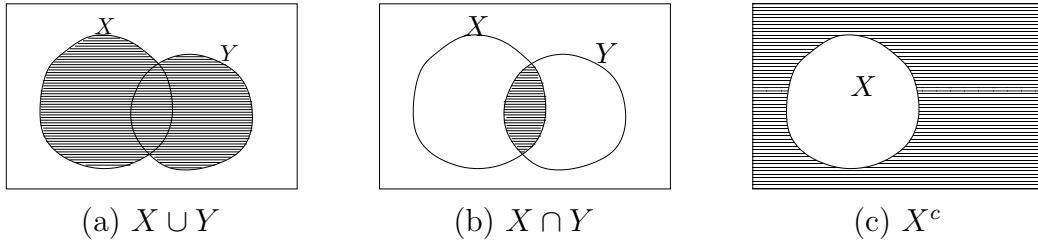


Figura 1: Diagramas de Venn (a) União de dois conjuntos. (b) Interseção de dois conjuntos. (c) Complemento de um conjunto.

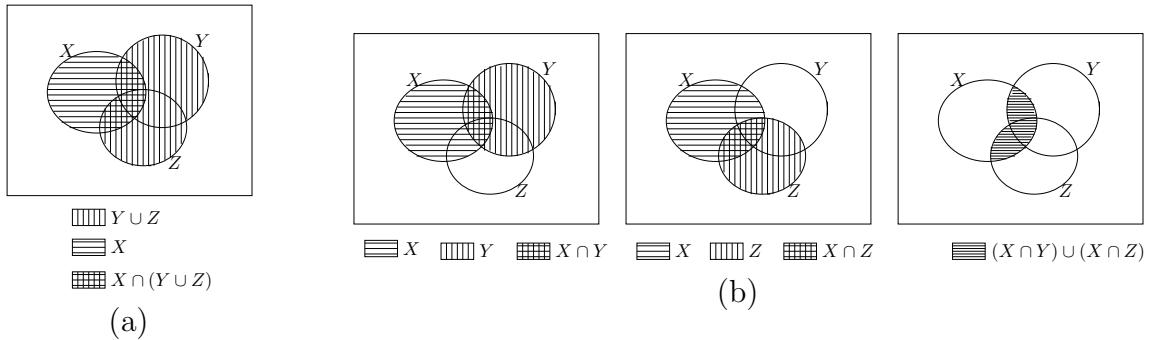


Figura 2: (a) $X \cap (Y \cup Z)$ e (b) $(X \cap Y) \cup (X \cap Z)$.

linhas horizontais na figura 2a. Esta coincide com a região hachurada no diagrama mais à direita da figura 2b, que representa o conjunto $(X \cap Y) \cup (X \cap Z)$.

Exemplo 3: O cálculo proposicional é um campo da lógica matemática que estuda proposições, ou seja, afirmações que ou são verdadeiras (V) ou são falsas (F), mas não ambas. As proposições podem ser conectadas usando-se os conectivos lógicos E, OU e NÃO, dando origem a novas proposições. Os conectivos lógicos podem ser representados pelos símbolos conforme tabela a seguir.

Conectivo	símbolo
E	\wedge
OU	\vee
NÃO	\neg

Supondo que x e y são duas proposições quaisquer, define-se essas operações conforme a tabela-verdade a seguir:

		x	y	$x \wedge y$			x	y	$x \vee y$
		F	F	F			F	F	F
x	$\neg x$	F	V	F			F	V	V
		V	F	F			V	F	V
		V	V	V			V	V	V

Qualquer semelhança com as operações lógicas vistas no contexto de circuitos lógicos não é mera coincidência. De fato, a lógica (ou cálculo) proposicional é uma álgebra booleana e ela tem uma correspondência um-para-um com $\langle B, +, \cdot, \bar{\cdot}, 0, 1 \rangle$, visto acima, conforme apontado a seguir.

Lógica proposicional	álgebra booleana B
\vee	$+$
\wedge	\cdot
F	0
V	1
$\neg x$	\bar{x}

Como consequência, temos também a correspondência entre as tabelas-verdade das operações \neg , \vee , \wedge com as tabelas-verdade das operações : $\bar{\cdot}$, $+$ e \cdot .

x	y	$\neg x$	$x \vee y$	$x \wedge y$	x	y	\bar{x}	$x + y$	$x \cdot y$
F	F	V	F	F	0	0	1	0	0
F	V	V	V	F	0	1	1	1	0
V	F	F	V	F	1	0	0	1	0
V	V	F	V	V	1	1	0	1	1

Exemplo 4: O conjunto $B^n = B \times B \times \dots \times B$, com as operações $+$, \cdot e $\bar{\cdot}$ herdadas de B e definidas, para quaisquer $(x_1, x_2, \dots, x_n), (y_1, y_2, \dots, y_n) \in B^n$, da seguinte forma

$$(x_1, x_2, \dots, x_n) + (y_1, y_2, \dots, y_n) = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$$

$$(x_1, x_2, \dots, x_n) \cdot (y_1, y_2, \dots, y_n) = (x_1 \cdot y_1, x_2 \cdot y_2, \dots, x_n \cdot y_n)$$

$$\overline{(x_1, x_2, \dots, x_n)} = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$$

é uma álgebra booleana. Verifique que de fato os 4 postulados são válidos.

3 Princípio da dualidade

Vimos que podemos escrever expressões envolvendo variáveis lógicas. Por exemplo, $a b c + a b \bar{c}$. Essa expressão pode ser manipulada algebraicamente (por exemplo, $a b c + a b \bar{c} = a b(c + \bar{c}) = a b(1) = a b$). Se a cada derivação algébrica apenas alguma manipulação algébrica válida é aplicada, garante-se que a expressão final resultante é equivalente à expressão original (isto é, o valor de ambas é igual um ao outro para quaisquer atribuições de valores para as variáveis).

O princípio de dualidade da álgebra booleana afirma que cada expressão ou identidade algébrica dedutível a partir dos postulados em uma álgebra booleana continua válida se todas as ocorrências dos operadores $+$ e \cdot e os elementos identidade 0 e 1 são trocados um pelo outro.

De fato, isso faz sentido pois o dual de cada um dos axiomas é também um axioma. Observe:

$$\begin{array}{rcl}
 & x \cdot y = y \cdot x & \\
 \text{Axioma A1} & \downarrow & \downarrow \\
 & x + y = y + x & \\
 \\[10pt]
 & x \cdot (y + z) = (x \cdot y) + (x \cdot z) & \\
 \text{Axioma A2} & \downarrow & \downarrow & \downarrow & \downarrow \\
 & x + (y \cdot z) = (x + y) \cdot (x + z) & & & \\
 \\[10pt]
 & x + 0 = x & \\
 \text{Axioma A3} & \downarrow & \downarrow \\
 & x \cdot 1 = x & \\
 \\[10pt]
 & x + \bar{x} = 1 & \\
 \text{Axioma A4} & \downarrow & \downarrow \\
 & x \cdot \bar{x} = 0 & \\
 \end{array}$$

Portanto, dada uma expressão E , ao se substituir sucessivamente subexpressões de E e das expressões derivadas de E por subexpressões equivalentes, obtém-se uma derivação que resulta em uma expressão equivalente a E . Se partirmos do dual da expressão original E e trocarmos todas as derivações realizadas a partir de E pelos correspondentes duais, obteremos uma expressão que é equivalente ao dual de E . Isto significa, em particular, que se conseguimos provar algebraicamente que $E_1 = E_2$, automaticamente teremos a prova de que $dual(E_1) = dual(E_2)$.

4 Leis fundamentais da álgebra booleana

Desta parte em diante omitiremos o símbolo \cdot na maioria das vezes; em vez de $x \cdot y$, escreveremos simplesmente xy . Suponha que $\langle A, +, \cdot, \bar{}, 0, 1 \rangle$ é uma álgebra booleana. Então, os seguintes resultados são válidos.

[Unicidade do 0 e 1] Os elementos 0 e 1 são únicos.

PROVA: Sejam dois elementos zero, 0_1 e 0_2 . Por A3, temos que para todo x_1 e x_2 em A ,

$$x_1 + 0_1 = x_1 \quad \text{e} \quad x_2 + 0_2 = x_2$$

Logo, se tomarmos as igualdades para $x_1 = 0_2$ e $x_2 = 0_1$, temos que

$$0_2 + 0_1 = 0_2 \quad \text{e} \quad 0_1 + 0_2 = 0_1$$

Por A1 e a transitividade de $=$, resulta que $0_1 = 0_2$.

A unicidade de 1 pode ser provada usando o princípio da dualidade.

[Idempotência] Para todo elemento $x \in A$, $x + x = x$ e $xx = x$.

PROVA:

$$\begin{array}{lll} x + x &= (x + x) \cdot 1 & (A3) \\ &= (x + x)(x + \bar{x}) & (A4) \\ &= x + x\bar{x} & (A2) \\ &= x + 0 & (A4) \\ &= x & (A3) \end{array} \qquad \begin{array}{lll} xx &= xx + 0 & (A3) \\ &= xx + x\bar{x} & (A4) \\ &= x(x + \bar{x}) & (A2) \\ &= x \cdot 1 & (A4) \\ &= x & (A3) \end{array}$$

[Identidade] Para todo $x \in A$, $x + 1 = 1$ e $x0 = 0$.

$$\begin{array}{lll} x + 1 &= 1 \cdot (x + 1) & (A3) \\ &= (x + \bar{x})(x + 1) & (A4) \\ &= x + \bar{x} \cdot 1 & (A2) \\ &= x + \bar{x} & (A3) \\ &= 1 & (A4) \end{array}$$

[Complemento do um (zero)] $\bar{1} = 0$ e $\bar{0} = 1$.

$$\begin{aligned}\bar{1} &= \bar{1} \cdot 1 & (A3) \\ &= 0 & (A4)\end{aligned}$$

[Absorção] Para todo $x, y \in A$, $x + xy = x$ e $x(x + y) = x$.

$$\begin{aligned}x + xy &= x \cdot 1 + xy & (A3) \\ &= x(1 + y) & (A2) \\ &= x \cdot 1 & (\text{Identidade}) \\ &= x & (A3)\end{aligned}$$

[Unicidade de \bar{x}] O inverso de qualquer elemento $x \in A$ é único.

PROVA: Sejam \bar{x}_1 e \bar{x}_2 em A tais que

$$x + \bar{x}_1 = 1 \quad \text{e} \quad x + \bar{x}_2 = 1 \quad \text{e} \quad x\bar{x}_1 = 0 \quad \text{e} \quad x\bar{x}_2 = 0$$

Então, temos que

$$\begin{aligned}\bar{x}_2 &= 1 \cdot \bar{x}_2 & (A3) \\ &= (x + \bar{x}_1)\bar{x}_2 & (\text{hipótese}) \\ &= x\bar{x}_2 + \bar{x}_1\bar{x}_2 & (A2) \\ &= 0 + \bar{x}_1\bar{x}_2 & (\text{hipótese}) \\ &= x\bar{x}_1 + \bar{x}_1\bar{x}_2 & (\text{hipótese}) \\ &= (x + \bar{x}_2)\bar{x}_1 & (A2) \\ &= 1 \cdot \bar{x}_1 & (\text{hipótese}) \\ &= \bar{x}_1 & (A3)\end{aligned}$$

[Involução] Para todo $x \in A$, $\bar{\bar{x}} = x$.

PROVA: Seja $\bar{\bar{x}} = y$. Então, por A4 temos que $\bar{x}y = 0$ e $\bar{x} + y = 1$. Mas por A4, $\bar{x}x = 0$ e $\bar{x} + x = 1$. Por causa da unicidade do complemento, $\bar{\bar{x}} = y = x$.

[Associatividade] Para quaisquer $x, y, z \in A$, $x + (y + z) = (x + y) + z$ e $x(yz) = (xy)z$.

[Lema] Para quaisquer $x, y, z \in A$, $x[(x + y) + z] = [(x + y) + z]x = x$.

$$\begin{aligned}x[(x + y) + z] &= [(x + y) + z]x & (A1) \\ x[(x + y) + z] &= x(x + y) + xz & (A2) \\ &= x + xz & (\text{absorção}) \\ &= x & (\text{absorção})\end{aligned}$$

Usando o lema acima, provaremos a propriedade associativa. Seja

$$\begin{aligned}
 Z &= [(x+y)+z][x+(y+z)] \\
 &= [(x+y)+z]x + [(x+y)+z](y+z) \quad (A2) \\
 &= x + [(x+y)+z](y+z) \quad (\text{lema}) \\
 &= x + \{(x+y)+z\}y + \{(x+y)+z\}z \quad (A2) \\
 &= x + \{(y+x)+z\}y + \{(x+y)+z\}z \quad (A1) \\
 &= x + \{y + [(x+y)+z]z\} \quad (\text{lema}) \\
 &= x + (y+z)
 \end{aligned}$$

De forma similar,

$$\begin{aligned}
 Z &= (x+y)[x+(y+z)] + z[x+(y+z)] \quad (A2) \\
 &= (x+y)[x+(y+z)] + z \quad (\text{lema}) \\
 &= \{x[x+(y+z)] + y[x+(y+z)]\} + z \quad (A2) \\
 &= \{x[x+(y+z)] + y\} + z \quad (\text{lema}) \\
 &= (x+y) + z \quad (\text{lema})
 \end{aligned}$$

Logo, $x + (y+z) = (x+y) + z$

[Teorema de DeMorgan] Para quaisquer $x, y \in A$, $\overline{(x+y)} = \overline{x}\overline{y}$ e $\overline{xy} = \overline{x} + \overline{y}$.

Vamos mostrar que $(x+y) + \overline{x}\overline{y} = 1$ e que $(x+y)(\overline{x}\overline{y}) = 0$.

$$\begin{aligned}
 (x+y) + \overline{x}\overline{y} &= [(x+y) + \overline{x}][(x+y) + \overline{y}] \quad (A2) \\
 &= [\overline{x} + (x+y)][\overline{y} + (x+y)] \quad (A1) \\
 &= [(\overline{x}+x) + y][x + (\overline{y}+y)] \quad (\text{Associativa} + A1) \\
 &= 1 \cdot 1 \quad (A4 + \text{Identidade}) \\
 &= 1 \quad (A3)
 \end{aligned}$$

$$\begin{aligned}
 (x+y) \cdot \overline{x}\overline{y} &= x(\overline{x}\overline{y}) + y(\overline{y}\overline{x}) \quad (A2 + A1) \\
 &= (x\overline{x})\overline{y} + (y\overline{y})\overline{x} \quad (\text{associativa}) \\
 &= 0 + 0 \quad (A4 + \text{Identidade}) \\
 &= 0 \quad (A3)
 \end{aligned}$$

Portanto, pela unicidade do complemento, podemos concluir que $\overline{(x+y)} = \overline{x}\overline{y}$.

A igualdade dual pode ser demonstrada pelo princípio da dualidade, ou usando o fato de que as igualdades acima valem também para \overline{x} e \overline{y} no lugar de x e y . \square

5 Relações de Ordem Parciais

5.1 Conjuntos parcialmente ordenados (posets)

Seja A um conjunto não vazio. Uma **relação binária** R sobre A é um subconjunto de $A \times A$, isto é, $R \subseteq A \times A$. Se $(x, y) \in R$, denotamos a relação de x por y como sendo xRy (lê-se x -erre- y).

Relação de ordem parcial: Uma relação binária \leq sobre A é uma **ordem parcial** se ela é

1. (reflexiva) $x \leq x$, para todo $x \in A$
2. (anti-simétrica) Se $x \leq y$ e $y \leq x$, então $x = y$, para todo $x, y \in A$
3. (transitiva) Se $x \leq y$ e $y \leq z$ então $x \leq z$, para todo $x, y, z \in A$

Se \leq é uma ordem parcial em A , então a relação \geq definida por, para quaisquer $x, y \in A$, $x \geq y$ se e somente se $y \leq x$, é também uma ordem parcial em A .

Observação: Apenas uma curiosidade: uma relação de equivalência é bem parecida com uma relação de ordem parcial. A diferença está na segunda propriedade: ordens parciais satisfazem anti-simetria, enquanto relações de equivalência satisfazem simetria (i.e., se $x \sim y$ então $y \sim x$, para todo $x, y \in A$).

Conjuntos parcialmente ordenados (poset): Um conjunto A munido de uma relação de ordem parcial \leq é denominado um conjunto parcialmente ordenado (ou *poset*) e denotado por (A, \leq) . Se (A, \leq) é um poset, então (A, \geq) também é um poset.

Exemplo 1: A relação de ordem \leq usual definida no conjunto dos números reais é uma ordem parcial (na verdade, ela é mais que uma ordem parcial; é uma **ordem total**, pois todos os elementos são comparáveis dois a dois). A relação $<$ não é uma ordem parcial pois ela não é reflexiva.

Exemplo 2: A relação de inclusão de conjuntos \subseteq é uma ordem parcial.

Diagrama de Hasse: Escrevemos $x < y$ quando $x \leq y$ e $x \neq y$. Dado um poset (A, \leq) e $x, y \in A$, dizemos que y cobre x se, e somente se, $x < y$ e não há outro elemento $z \in A$

tal que $x < z < y$. Um diagrama de Hasse do poset (A, \leq) é uma representação gráfica onde vértices representam os elementos de A e dois elementos x e y são ligados por uma aresta se e somente se y cobre x . Em um diagrama de Hasse, os elementos menores (com relação a ordem parcial) são em geral desenhados abaixo dos elementos maiores.

Exemplo: O diagrama de Hasse do poset $(\{a, b, c\}, \subseteq)$ é mostrado na figura 3. Trata-se do cubo, visto no contexto de circuitos lógicos.

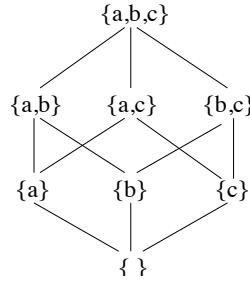


Figura 3: Diagrama de Hasse de $(\{a, b, c\}, \subseteq)$.

6 Relação de ordem e álgebra booleana

Seja $\langle A, +, \cdot, \neg, 0, 1 \rangle$ uma álgebra booleana. Definimos uma relação binária \leq em A da seguinte forma:

$$\forall x, y \in A, \quad x \leq y \quad \text{se e somente se} \quad x + y = y \quad (1)$$

A relação \leq definida pela equação 1 é uma relação de ordem parcial. De fato, a relação \leq é (1) reflexiva pois pela lei de idempotência ($x + x = x$) temos que $x \leq x$ para todo $x \in A$; é (2) anti-simétrica pois se $x \leq y$ e $y \leq x$, então $x + y = y$ e $y + x = x$ e, portanto, pela comutatividade de $+$, segue que $x = y$; e é (3) transitiva pois se $x \leq y$ e $y \leq z$, então

$$\begin{aligned} z &= y + z && (\text{pois } y \leq z) \\ &= (x + y) + z && (\text{pois } x \leq y) \\ &= x + (y + z) && (\text{associatividade de } +) \\ &= x + z && (\text{pois } y \leq z) \end{aligned}$$

Logo, $x \leq z$.

Referências

- [Garnier and Taylor, 1992] Garnier, R. and Taylor, J. (1992). *Discrete Mathematics for New Technology*. Adam Hilger.
- [Hill and Peterson, 1981] Hill, F. J. and Peterson, G. R. (1981). *Introduction to Switching Theory and Logical Design*. John Wiley, 3rd edition.
- [Whitesitt, 1961] Whitesitt, J. E. (1961). *Boolean Algebra and its Applications*. Addison-Wesley.