Álgebra Booleana e Aplicações

Notas de aula (2005)

Nina S. T. Hirata

Depto. de Ciência da Computação
Instituto de Matemática e Estatística
USP - Universidade de São Paulo

Última revisão: 22 de fevereiro de 2006

Notas de aula de MAC0329 (2005)

Nina S. T. Hirata

Depto. de Ciência da Computação
Instituto de Matemática e Estatística
USP - Universidade de São Paulo

Correções, sugestões e comentários são bem-vindos.

Capítulo 1

Introdução

Esta disciplina tem como objetivos (1) introduzir a álgebra booleana, um sistema algébrico cujo desenvolvimento inicial dos fundamentos é devido a George Boole (1815-1864), e (2) mostrar sua aplicação, principalmente no projeto lógico de circuitos digitais.

O curso inicialmente abordará algumas aplicaçãoes da álgebra booleana de forma bastante breve. Em seguida, elementos comuns dessas aplicações serão abstraídas para se chegar às definições e fundamentos teóricos da álgebra booleana. Alguns conceitos, resultados e algoritmos importantes no contexto da álgebra booleana (tais com propriedades, relação com ordens parciais, simplificação de expressões booleanas) serão estudados. Paralelamente, serão estudados também a relação e a aplicação dos mesmos no projeto de circuitos lógicos.

Este capítulo contém notas de aula relativas a parte inicial do curso onde serão explorados tópicos que ilustram aplicações da álgebra booleana. Precisamente, serão apresentados uma breve introdução ao problema de projeto de circuitos lógicos, a álgebra dos conjuntos, a lógica proposicional e operadores de imagens binárias.

1.1 Circuitos Lógicos

Considere dispositivos como os mostrados na figura 1.1, que recebem sinais de entrada à esquerda e produzem sinais de saída à direita.

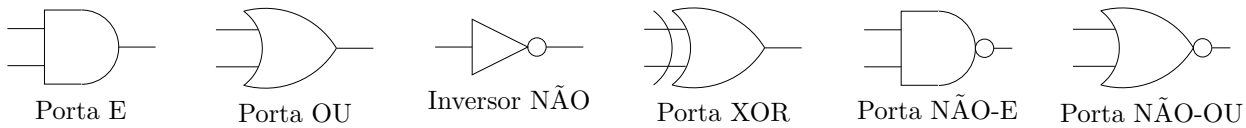


Figura 1.1: Representação gráfica de algumas portas lógicas.

Suponha que nestes dispositivos tanto as entradas como as saídas tomam apenas o valor 0 (zero) ou 1 (um). A relação entrada-saída desses dispositivos está descrita a seguir, onde $x_1, x_2 \in \{0, 1\}$ denotam entradas. Por exemplo, a saída da porta E é denotada por $x_1\, x_2$ e toma valor 1 se e somente se $x_1 = 1$ e $x_2 = 1$.

		E	OU	NÃO	XOR	NÃO-E	NÃO-OU
x_1	x_2	$x_1\,x_2$	$x_1 + x_2$	$\overline{x}_1$	$x_1 \oplus x_2$	$\overline{x_1\,x_2}$	$\overline{x_1 + x_2}$
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

Estes dispositivos (portas e inversores) podem ser interconectados. A rede resultante é denominada **circuito**. Por exemplo, a figura 1.2 mostra um circuito com três inversores e uma porta E.

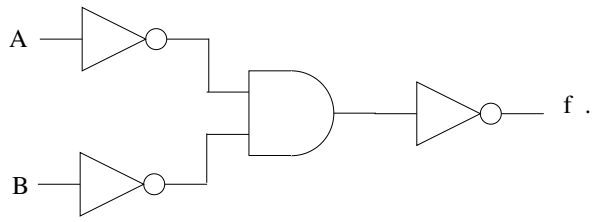


Figura 1.2: Um circuito simples.

A relação entrada-saída deste circuito é mostrada na tabela a seguir:

					saída
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

A esta altura já é possível percebermos que a relação entrada-saída de um circuito descreve uma função. Por exemplo, no caso do circuto acima, a saída pode ser expressa como $f(A, B)$ para indicar que o valor da saída depende das duas entradas A e B que podem ser vistas como variáveis. Neste sentido, cada linha das tabelas acima corresponde a uma das possíveis atribuições de valor às variáveis de entrada do circuito. Além disso, a saída pode ser caracterizada por uma expressão, justamente aquela que aparece no cabeçalho das colunas. Assim no circuito acima, $f(A, B) = \overline{\overline{A}\,\overline{B}}$. Dizemos que um circuito realiza uma função.

Se você for um leitor atento, já deve ter percebido que $f(A, B) = \overline{\overline{A}\,\overline{B}} = A + B$. Em outras palavras, uma mesma função pode ser representada por diferentes expressões. Ou ainda, visto por outro ângulo, diferentes circuitos têm um mesmo comportamento. Quando dois circuitos realizam uma mesma função eles são ditos equivalentes. Esta observação leva-nos a uma questão natural: **Como verificar se dois circuitos são equivalentes?** e leva-nos também à seguinte questão interessante: **Dada uma função (supondo-se que a mesma pode ser realizada por circuitos), qual é o "menor" circuito que a realiza?** Aqui, o termo menor pode ser associado ao número de dispositivos utilizados no circuito.

Já vimos que um circuito realiza uma função. Será que a inversa é válida também? Ou seja, **será que qualquer função de $\{0,1\}^n$ em $\{0,1\}$ pode ser realizada por um circuito?**

Deixemos esta pergunta temporariamente de lado e vamos resolver um exercício. 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). Deseja-se projetar um circuito que realiza a adição de A e B. Deseja-se também, supondo que os números estão na forma complemento de dois, detectar *overflow*.

Uma possível solução consiste em projetarmos um circuito para realizar a adição dos bits de uma coluna onde as entradas são os bits a_i, b_i e o vai-um c_i da coluna anterior e as saídas são o bit da soma s_i e o vai-um c_{i+1} para a próxima coluna. Esses circuitos podem ser concatenados em série de forma que a saída vai-um de uma coluna i alimenta a entrada vai-um da coluna $i+1$, conforme mostrado na figura 1.3.

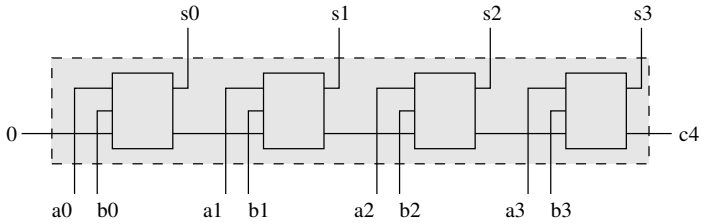


Figura 1.3: Esquema de um somador de 4 bits.

Cada uma das "caixas" corresponde a um somador de bits, cujo exemplo de uma possível realização em circuito pode ser visto na figura 1.4. Observe que este trata-se de um circuito de duas saídas (e, portanto, realiza duas funções). As funções poderiam ser realizadas por um circuito cada, mas na solução apresentada há compartilhamento de subcircuitos.

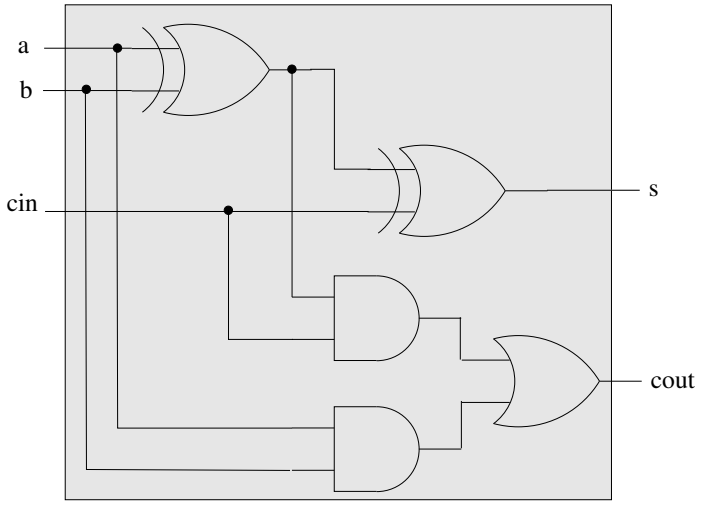


Figura 1.4: Esquema de um somador de bits.

Em geral, para projetar circuitos, o primeiro passo é a descrição funcional do circuito que desejamos projetar. Isto pode ser feito através da construção de uma tabela como as vistas acima (tabelas-verdade), especificando-se quais são as entradas, quais as saídas e a relação entrada-saída. Fica como exercício a construção da tabela para o somador de bits visto acima. A partir da tabela-verdade, pode-se projetar diretamente um circuito (veremos isso posteriormente, na segunda metade do curso). No entanto, circuitos projetados desta forma nem sempre são as "melhores". Há várias questões que podem ser consideradas quando se deseja projetar o "melhor"circuito. Algumas dessas questões (tais como minimizar o número de portas, restringir-se ao uso de um subconjunto de tipos de portas, compartilhar subcircuitos, etc) serão vistas mais tarde.

Resumindo, algumas questões interessantes que procuraremos responder posteriormente são:

- 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?

1.2 Álgebra dos conjuntos

A álgebra dos conjuntos é um exemplo de álgebra booleana que intuitivamente é relativamente simples de ser entendida. Por esta razão, ela será introduzida antes da introdução formal de álgebra booleana, com o objetivo de ajudar o entendimento de uma definição formal a ser apresentada mais adiante no curso.

Referências para esta parte do curso: capítulos 1 a 6 de [Filho, 1980], capítulo 2 de [Mendelson, 1977], capítulo 1 de [Whitesitt, 1961], capítulo sobre **conjuntos** de qualquer livro sobre Matemática Discreta (Discrete Mathematics) [Ross and Wright, 1992, Garnier and Taylor, 1992].

Conjuntos e elementos

Conjuntos são coleções de objetos, denominados elementos[1]

Exemplos de conjuntos

O conjunto de todos os números inteiros, o conjunto de todos os alunos de MAC0329 do semestre corrente, o conjunto de todos os seres humanos vivos atualmente, o conjunto de todos os números reais maiores que zero e menores que 1, o conjunto de todos os jogadores da seleção brasileira de futebol, o conjunto de todas as letras do alfabeto romano, etc.

Notação

Conjuntos serão representados por letras maiúsculas: A, B, C, S, etc. Elementos de um conjunto serão representados por letras minúsculas: a, b, x, y, etc.

Em geral podemos especificar um conjunto descrevendo os elementos, ou então enumerando os elementos. Por exemplo, o conjunto A de todos os números inteiros pares pode ser expresso por:

$$A = \{x \in Z : x \text{ é par}\}$$

e o conjunto B das cores da bandeira brasileira pode ser expresso por:

$$B = \{verde, amarelo, azul, branco\}$$

[1] Não é objetivo fazermos uma definição formal de conjunto. Basta utilizaremos a noção intuitiva que temos sobre conjuntos.

Conjuntos universo e vazio

Dois conjuntos especiais são o **conjunto universo**, isto é, o conjunto de todos os objetos em questão, e o **conjunto vazio**, isto é, o conjunto que não contém nenhum elemento. Os conjuntos universo e vazio são denotados, respectivamente, por U e $\emptyset$.

Conjunto unitário

Em álgebra de conjuntos, os objetos de interesse são os conjuntos e não os elementos que os formam. Assim, as operações devem ser definidas sobre ou entre conjuntos, mas nunca sobre elementos isolados. Para tratar elementos, devemos considerar conjuntos unitários. Por exemplo, se a é um elemento de U então $\{a\}$ denota o **conjunto unitário** que contém apenas um único elemento, o elemento a.

Relação elemento $\times$ conjunto

Se um elemento x **pertence** a um conjunto A, escrevemos $x \in A$. Diremos, alternativamente, que x é **membro** de A. Se x não pertence ao conjunto A, escrevemos $x \notin A$.

Relação conjunto $\times$ conjunto

Um conjunto A é **igual** a um conjunto B, denotado $A = B$, se eles contém exatamente os mesmos elementos. Se não forem iguais, eles são **diferentes**, e denotado por $A \neq B$.

Um conjunto A está **contido** num conjunto B se todos os elementos de A pertencem também ao conjunto B. Escrevemos $A \subseteq B$ e dizemos também que A é um **subconjunto** de B. Se, além disso, B possui pelo menos um elemento que não pertence a A, então dizemos que A está **propriamente contido** em B, ou que A é um subconjunto próprio de B, e denotamos $A \subset B$.

Propriedades da relação $\subseteq$

A relação de inclusão de conjuntos $\subseteq$ obedece às seguintes propriedades. Para quaisquer X, Y e Z,

I1. (reflexiva) $X \subseteq X$

I2. (transitiva) $X \subseteq Y$ e $Y \subseteq Z \Longrightarrow X \subseteq Z$

I3. (anti-simétrica) $X \subseteq Y$ e $Y \subseteq X \Longrightarrow X = Y$

I4. (a) $\emptyset \subseteq X$
(b) $X \subseteq U$

Conjunto potência (power set) ou conjunto das partes de um conjunto

Dado um conjunto A, o **conjunto potência de** A é denotado por $\mathcal{P}(A)$ e definido por $\mathcal{P}(A) = \{X \subseteq U : X \subseteq A\}$, ou seja, $\mathcal{P}(A)$ é o conjunto de todos os subconjuntos de A.

Exercício: Mostre que se A contém n elementos então $\mathcal{P}(A)$ contém 2^n elementos.

Prova: Para $n = 0$ o resultado é óbvio. Suponha $n > 0$. Na escolha de um subconjunto X de A, existem duas possibilidades para cada elemento $x \in A$: ou $x \in X$ ou $x \notin X$. Como o fato de x estar ou não em X independe do fato de qualquer outro elemento y de A estar ou não em X, então existem 2^n formas de se escolher um subconjunto de A.

Exercício: Seja $A = \{a, b, c\}$. Liste todos os elementos de $\mathcal{P}(A)$.

Complemento, união e interseção

O **complemento** de um conjunto X, denotado X^c, consiste de todos os elementos em U que não estão em X, ou seja, $X^c = \{x \in U : x \notin X\}$.

Conjuntos podem ser combinados para gerar outros conjuntos. Para isso, podemos considerar duas regras (operações) que definem formas pelas quais conjuntos podem ser combinados: a **união** e a **interseção**.

Dados dois conjuntos X e Y quaisquer, a **união** de X e Y é denotada $X \cup Y$ e definida como sendo o conjunto de elementos que pertencem ou a X, ou a Y ou a ambos, ou seja, $X \cup Y = \{x \in U : x \in X \text{ ou } x \in Y\}$. A **interseção** de X e Y é denotada $X \cap Y$ e definida como sendo o conjunto de elementos que pertencem tanto a X como a Y, ou seja, $X \cap Y = \{x \in U : x \in X \text{ e } x \in Y\}$.

Se $X \cap Y = \emptyset$ (conjunto vazio) então dizemos que X e Y são **disjuntos**.

Exemplos:

$\{1, 2, 3\} \cup \{2, 4, 6\} = \{1, 2, 3, 4, 6\}$ $\quad\quad$ $\{1, 2, 3\} \cap \{2, 4, 6\} = \{2\}$

$\{a\} \cup \{b\} = \{a, b\}$ $\quad\quad$ $\{a\} \cap \{b\} = \emptyset$

Diagramas de Venn

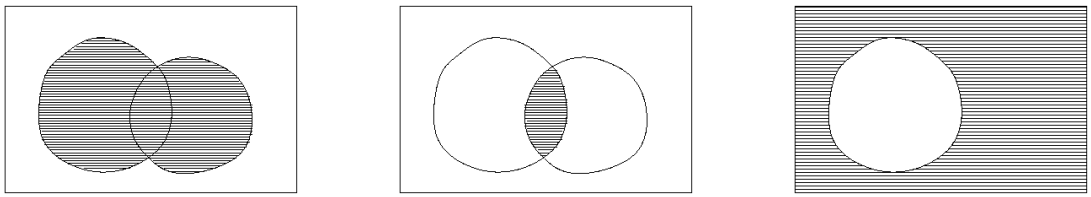
Os diagramas de Venn são úteis para reforçar a noção intuitiva sobre conjuntos, principalmente para analisar relações entre os conjuntos e também seus membros. Para demonstrar propriedades dos conjuntos, uma prova estritamente algébrica seria necessária. No entanto, para entender uma propriedade e, mais do que isso, para nos convencermos de sua validade, os diagramas de Venn são bastante úteis.

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 1.5, a união e interseção de dois conjuntos genéricos estão representadas pelas regiões hachuradas das figuras 1.5a e 1.5b, respectivamente. O complemento de um conjunto é representado no diagrama da figura 1.5c.

Exercício: Seja x um elemento no conjunto universo U e X e Y dois subconjuntos quaisquer de U. Mostre que x é membro de apenas um dos conjuntos $X \cap Y$, $X \cap Y^c$, $X^c \cap Y$ e $X^c \cap Y^c$.

Dica: Desenhe o diagrama de Venn e argumente.

Leis fundamentais



(a) $X \cup Y$ (b) $X \cap Y$ (c) X^c

Figura 1.5: Diagramas de Venn (a) União de dois conjuntos. (b) Interseção de dois conjuntos. (c) Complemento de um conjunto.

Dados conjuntos X, Y, Z quaisquer, utilize diagramas de Venn para convencer-se da validade das seguintes leis.

L1. Comutativa

(a) $X \cap Y = Y \cap X$
(b) $X \cup Y = Y \cup X$

L2. Associativa

(a) $X \cap (Y \cap Z) = (X \cap Y) \cap Z$
(b) $X \cup (Y \cup Z) = (X \cup Y) \cup Z$

L3. Distributiva

(a) $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
(b) $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$

L4. Idempotência

(a) $X \cap X = X$
(b) $X \cup X = X$

L5. Absorção

(a) $X \cap (X \cup Y) = X$
(b) $X \cup (X \cap Y) = X$

L6. Complementação

(a) $X \cap X^c = \emptyset$
(b) $X \cup X^c = U$

L7. Complementação dupla

$$(X^c)^c = X$$

L8. De Morgan

(a) $(X \cap Y)^c = X^c \cup Y^c$
(b) $(X \cup Y)^c = X^c \cap Y^c$

L9. Operações com $\emptyset$ e U

(a) (Elemento neutro) $U \cap X = X$ e $\emptyset \cup X = X$

(b) $\emptyset \cap X = \emptyset$ e $U \cup X = U$

(c) $\emptyset^c = U$ e $U^c = \emptyset$

As igualdades das leis acima podem ser entendidas com o auxílio de diagramas de Venn. Para provar as igualdades podemos mostrar que o conjunto do lado esquerdo está contido no do lado direito e vice-versa (propriedade de anti-simetria de $\subseteq$), ou ainda via transformações lógicas (ver exemplo mais adiante).

Note que $X \cup Y = (X^c \cap Y^c)^c$. Isto implica que o operador $\cup$ poderia ser dispensado. Maiores detalhes sobre isso serão vistos oportunamente. Enquanto isso, vale a pena mencionarmos que embora não necessário, o uso dos três operadores é conveniente.

Algumas leis são semelhantes aos da álgebra dos números. No entanto, na álgebra dos conjuntos não existem, como na álgebra usual, expressões do tipo $2X$ ou X^2 e algumas leis como as de número 3b, 4 e 5 não são válidas na álgebra dos números.

Observe também que a maior parte das leis aparece aos pares. Iremos ver mais adiante que isso está ligado ao princípio da dualidade.

Exercício: Prove ou mostre via diagramas de Venn a validade das leis L3, L5 e L8 acima.

Como exemplo, vamos mostrar a validade da lei L3(a), isto é, $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$. Primeiramente utilizaremos o diagrama de Venn para nos convencermos da validade. O conjunto $X \cap (Y \cup Z)$ corresponde à região hachurada pelas linhas verticais e pelas linhas horizontais na figura 1.6a. Esta coincide com a região hachurada no diagrama mais à direita da figura 1.6b, que representa o conjunto $(X \cap Y) \cup (X \cap Z)$.

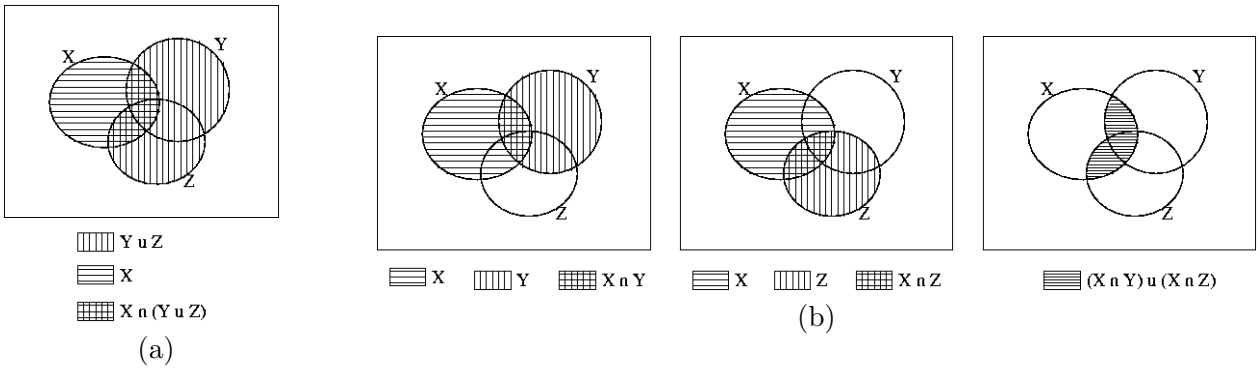


Figura 1.6: (a) $X \cap (Y \cup Z)$. (b) $(X \cap Y) \cup (X \cap Z)$.

Para provar a igualdade, devemos mostrar que $X \cap (Y \cup Z) \subseteq (X \cap Y) \cup (X \cap Z)$ e que $(X \cap Y) \cup (X \cap Z) \subseteq X \cap (Y \cup Z)$.

Prova: Considere $x \in X \cap (Y \cup Z)$. Então $x \in X$. Além disso, $x \in Y \cup Z$. Isso significa que ou $x \in Y$, e neste caso $x \in X \cap Y$, ou $x \in Z$, e neste caso $x \in X \cap Z$. Logo, $x \in (X \cap Y) \cup (X \cap Z)$.

Por outro lado, considere $y \in (X \cap Y) \cup (X \cap Z)$. Então, ou $y \in (X \cap Y)$ ou $y \in (X \cap Z)$. Se $y \in (X \cap Y)$, então $y \in X$ e $y \in Y$. Se $y \in Y$ então $y \in Y \cup Z$ e portanto, $y \in X \cap (Y \cup Z)$. De forma similar, se $y \in (X \cap Z)$, então $y \in X$ e $y \in Z$, de modo que $y \in Y \cup Z$ e portanto, $y \in X \cap (Y \cup Z)$. □

Podemos utilizar o mesmo raciocínio acima, porém expressando os conjuntos explicitamente, conforme a seguir:

$$\begin{aligned} X \cap (Y \cup Z) &= \{x : x \in X \text{ e } x \in Y \cup Z\} \\ &= \{x : x \in X \text{ e } (x \in Y \text{ ou } x \in Z)\} \\ &= \{x : (x \in X \text{ e } x \in Y) \text{ ou } (x \in X \text{ e } x \in Z)\} \\ &= \{x : x \in X \cap Y \text{ ou } x \in X \cap Z\} \\ &= (X \cap Y) \cup (X \cap Z) \end{aligned}$$

Exercício: A seguintes generalizações das leis de De Morgan são válidas ? Explique sua resposta.

$$(A_1 \cup A_2 \cup \cdots \cup A_n)^c = A_1^c \cap A_2^c \cap \cdots \cap A_n^c$$

$$(A_1 \cap A_2 \cap \cdots \cap A_n)^c = A_1^c \cup A_2^c \cup \cdots \cup A_n^c$$

Exercício: Desenhe a relação $X \subseteq Y$ num diagrama de Venn. Quais igualdades envolvendo os conjuntos X e Y são verdadeiras quando $X \subseteq Y$? Liste pelo menos três.

Outras propriedades

Para quaisquer conjuntos X, Y e Z, as seguintes propriedades são verdadeiras:

P1. (a) $X \cap Y \subseteq X$ e $X \cap Y \subseteq Y$
(b) $X \subseteq X \cup Y$ e $Y \subseteq X \cup Y$

P2. (a) $X \cap Y = X$ sse $X \subseteq Y$
(b) $X \cup Y = Y$ sse $X \subseteq Y$

P3. (a) $X = Y$ sse ($X \subseteq Y$ e $Y \subseteq X$)
(b) $X = Y$ sse $X^c = Y^c$

Exercício: Mostre que $A \cap (A \cup B) = A$.

Por P1(b), sabemos que $A \subseteq A \cup B$. Mas então, por P2(a) $A \subseteq A \cup B$ implica que $A \cap (A \cup B) = A$.

Exercício: Dados dois conjuntos X e Y a **diferença** deles é definida por $X \setminus Y = \{x \in U : x \in X \text{ e } x \notin Y\}$ e a **diferença simétrica** entre eles é definida por $X \Delta Y = (X \setminus Y) \cup (Y \setminus X)$. Expresse estes conjuntos em termos das operações de complementação, união e interseção (deduza a partir do diagrama de Venn).

Obs.: Na presença dos operadores $\cup$, $\cap$ e ^{c}, não há necessidade dos operadores $\setminus$ e Δ. No entanto, estes operadores podem ser práticos.

Simplificação de expressões

As operações $\cup$, $\cap$ e ^{c} podem ser utilizadas para combinar conjuntos de várias formas. A combinação pode ser representada por uma expressão que descreve como os conjuntos foram combinados. Assim como a combinação de conjuntos resulta em um conjunto, uma expressão que descreve uma combinação de conjuntos representa um conjunto (aquele que resulta após as combinações serem executadas).

Como vimos no caso de algumas leis, existem diferentes formas para se expressar um mesmo conjunto. Por exemplo, vimos que $X = X \cup X$. Ou ainda, $(X \cup Y)^c = X^c \cap Y^c$. Assim sendo, surge a possibilidade de estudarmos diferentes formas de expressão de conjuntos. Expressões podem ser expandidas, fatoradas ou simplificadas aplicando-se as leis fundamentais.

Exemplo: Mostramos a simplificação da expressão $[(A \cap B) \cup (A \cap B^c)] \cap (A^c \cup B)$.

$$\begin{aligned}
[(A \cap B) \cup (A \cap B^c)] \cap (A^c \cup B) &= [A \cap (B \cup B^c)] \cap (A^c \cup B) \\
&= (A \cap U) \cap (A^c \cup B) \\
&= A \cap (A^c \cup B) \\
&= (A \cap A^c) \cup (A \cap B) \\
&= \emptyset \cup (A \cap B) \\
&= A \cap B
\end{aligned}$$

Exercício: Simplifique as seguintes expressões:
a) $(A \cap B^c)^c \cup (B \cap C)$
b) $[(A \cup B) \cap (A \cup B^c)] \cap (A \cup B)$
c) $(A \cap B \cap C) \cup (A \cap B \cap C^c) \cup (A^c \cap B \cap C^c) \cup (A^c \cap B^c \cap C^c)$
d) $(A \cup B) \cap (A \cup B^c) \cap (A^c \cup B)$

Exercício: Verifique se as seguintes igualdades / afirmações são válidas. Justifique (pode ser via diagrama de Venn) ou mostre um contra-exemplo

a) $(A \cap B) \cup B = B$
b) $(A \cap C) \cap (B \cup C) = A \cap C$
c) Se $A \cup B = A \cup C$ então $B = C$
d) $A \cap (B \cup C) = (A \cap B) \cup C$
e) $A \cup B = (A^c \cap B^c)^c$
f) $(A \cup B^c) \cap (A^c \cup B) \cap (A^c \cup B^c) = A^c \cup B^c$
g) $A \cap (B \setminus C) = (A \cap B) \setminus (A \cap C)$
h) $A \cap B = A \setminus (A \setminus B)$
i) $X \setminus X = \emptyset$
j) $X \setminus \emptyset = X$
k) $\emptyset \setminus X = \emptyset$

l) $(X \setminus Y) \setminus Z = X \setminus (Y \cup Z)$
m) $(X \setminus Y) \setminus Z = (X \setminus Z) \setminus Y$
n) $X \setminus Y = X \cap Y^c$
o) $(A \setminus B)^c = B \cup A^c$
p) $(A \setminus B) \cap C = (A \cap C) \setminus B$
q) $X \Delta X = \emptyset$
r) $X \Delta Y = Y \Delta X$
s) $X \Delta \emptyset = X$
t) $X \Delta Y = (X \cap Y^c) \cup (X^c \cap Y)$
u) $X \cap (Y \Delta Z) = (X \cap Y) \Delta (X \cap Z)$
v) $X \cup (Y \Delta Z) = (X \cup Y) \Delta (X \cup Z)$
x) Se $A \subseteq B$ e $A \subseteq C$ então $A \subseteq B \cap C$

Nos seguintes exemplos ilustramos como podemos utilizar a álgebra dos conjuntos para analisar afirmações ou conjunto de afirmações.

Exemplo:

Dado que Sócrates é um homem e que todos os homens são mortais, deseja-se mostrar que Sócrates é mortal.

Vamos usar a propriedade de que $X \subseteq Y$ e $Y \subseteq Z$ implica $X \subseteq Z$.

Sejam
U: conjunto de todos os seres vivos
X: conjunto de todos os seres vivos humanos
Y: conjunto de todos os mortais
S: conjunto unitário cujo único elemento é Sócrates

Utilizando esta notação, temos que $S \subseteq X$ (Sócrates é um homem) e que $X \subseteq Y$ (todos os homens são mortais). Logo, $S \subseteq Y$ (ou seja, Sócrates é mortal).

Exemplo:

Considere as quatro afirmações a seguir:
a) Um homem infeliz não é dono do seu próprio nariz.
b) Todos os homens casados têm responsabilidades
c) Todo homem ou é casado ou é dono do seu próprio nariz (ou ambos).
d) Nenhum homem com responsabilidades pode pescar todos os dias.

Sejam
U: conjunto de todos os homens
H: conjunto dos homens felizes
B: conjunto dos homens donos dos próprios narizes
M: conjunto dos homens casados
R: conjunto dos homens com responsabilidades
F: conjunto dos homens que pescam todo dia

Que tipo de conclusões podemos derivar a partir das afirmações acima?

a) $H^c \subseteq B^c \iff B \subseteq H$
b) $M \subseteq R \iff R^c \subseteq M^c$
c) $M \cup B = U \iff M^c \subseteq B$ (ou $B^c \subseteq M$)
d) $R \cap F = \emptyset \iff F \subseteq R^c$

Combinando (d) e (b) temos que $F \subseteq R^c \subseteq M^c$ (Todo homem que pesca todos os dias não são casados).

Combinando $F \subseteq M^c$ e (c) temos que $F \subseteq B$ (Todo homem que pesca todos os dias é dono do seu próprio nariz)

Combinando $F \subseteq B$ e (a) temos que $F \subseteq H$ (Todo homem que pesca todos os dias é feliz).

Exemplo: Três colecionadores A, B e C de obras literárias antigas têm interesse pelas seguintes obras:

A obras sobre política em inglês e ficção em língua estrangeira.

B obras sobre política, exceto ficção em inglês, e obras em inglês que não sejam ficção

C obras que não são ficção, que sejam em inglês ou sobre política em língua estrangeira.

Pergunta-se quais são as obras pelas quais mais de um colecionador têm interesse?

Defina os conjuntos

A: todas as obras pelos quais A se interessa
B: todas as obras pelos quais B se interessa
C: todas as obras pelos quais C se interessa
E: todas as obras em inglês
F: todas as obras que são ficção
P: todas as obras sobre política

Podemos então expressar o conjunto Z de obras pelos quais pelo menos dois colecionadores possuem interesse por:

$$Z = (A \cap B) \cup (A \cap C) \cup (B \cap C) \tag{1.1}$$

Analogamente, podemos expressar os conjuntos A, B e C em termos dos conjuntos E, N e P da seguinte forma:

$$\begin{aligned} A &= (P \cap E) \cup (F \cap \overline{E}) \\ B &= (P \cap \overline{(F \cap E)}) \cup (E \cap \overline{F}) \\ C &= \overline{F} \cap (E \cup (P \cap \overline{E})) \end{aligned}$$

Simplificando Z, após substituírmos A, B e C, temos que

$$Z = (E \cap \overline{F}) \cup (P \cap \overline{E}) \tag{1.2}$$

ou seja, que há pelo menos dois interessados em obras não-ficção em inglês e obras sobre política em língua estrangeira.

ATÉ AQUI, foram vistos os principais conceitos relacionados a conjuntos. Em particular, note que conjuntos juntamente com as operações de união, interseção e complementação podem ser vistos como um sistema algébrico, onde expressões podem ser escritas para representar uma série de operações sobre conjuntos e as mesmas podem ser, por exemplo, simplificadas aplicando-se manipulações algébricas baseadas nas leis básicas. E já que estamos falando de conjuntos, vamos relembrar alguns outros conceitos que poderão ser úteis ao longo do curso.

Produto cartesiano

Sejam A e B dois conjuntos não vazios. O produto cartesiano de A e B, denotado $A \times B$, é o conjunto de todos os pares ordenados (x, y) tais que o primeiro elemento x pertence a A e o segundo elemento y pertence a B.

$$A \times B = \{(x, y) : x \in A \text{ e } y \in B\}$$

Generalizando, dados n conjuntos A_1, A_2,..., A_n, o produto cartesiano destes n conjuntos é dado por

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) : a_1 \in A_1 \text{ e } a_2 \in A_2 \text{ e } \ldots \text{ e } a_n \in A_n\}$$

Quando $A_i = A_j$ para quaisquer i e j, denota-se o produto cartesiano acima também por A^n.

Exercício: Seja $B = \{0, 1\}$. Liste todos os elementos do produto cartesiano $B \times B \times B$.

Relações binárias e funções

Sejam A e B dois conjuntos não vazios. Uma relação binária R sobre A e B é um subconjunto de $A \times B$, isto é, $R \subseteq A \times B$.

Dizemos que y é correspondente de x pela relação R se $(x, y) \in R$, e denotamos xRy (lê-se x-erre-y).

Uma relação binária $f \subseteq A \times B$ é uma função de A em B se para todo $x \in A$ existe um único $y \in B$ tal que $(x, y) \in f$. A função é denotada $f : A \to B$ e em vez de xfy denotamos $f(x) = y$.

Exercício: Explique o que são funções sobrejetoras, injetoras e bijetoras.

1.3 Cálculo proposicional

Referências para este assunto: [Ross and Wright, 1992], [Garnier and Taylor, 1992], capítulo 1 de [Mendelson, 1977], capítulo 3 de [Whitesitt, 1961], etc.

Proposição

Proposições são sentenças afirmativas declarativas que não sejam ambígüas e que possuem a propriedade de serem ou verdadeiras ou falsas, mas não ambas.

Exemplos:
. "Gatos têm quatro patas"
· "$1 + 2 = 3$"
. "A Terra é quadrada"
. "3 é um número primo"

Exemplos de sentenças que não são proposições:
. "O que estou dizendo agora é mentira"
. "Irá chover amanhã"
. "Onde está a chave ?"

Cálculo proposicional

É uma sub-área da álgebra da lógica que estuda um conjunto formal de regras que permitem a análise e manipulação de proposições.

Conectivos lógicos

Proposições simples podem ser concatenadas através de conectivos lógicos E, OU, NÃO para formar novas proposições compostas.

Exemplos: Das proposições "Fulano está cansado" e "Ciclano está cozinhando", pode-se formas as proposições "Fulano está cansado E Ciclano está cozinhando", ou "Fulano está cansado OU Ciclano está cozinhando", ou "Fulano NÃO está cansado".

Notações

Proposições serão representadas por letras como x, y, z, p, q, etc. Em geral, as letras que representam proposições simples são denominadas **variáveis** (lógicas).

Proposições têm valor lógico ou V (VERDADEIRO) ou F (FALSO).

Utilizaremos os seguintes símbolos para representar os conectivos lógicos:

Conectivo	símbolo
E	$\wedge$
OU	$\vee$
NÃO	$\neg$

Os conectivos implicação condicional ($\rightarrow$) e bicondicional $\leftrightarrow$

Em adição aos três conectivos vistos acima, é comum também a utilização dos condicionais SE-ENTÃO ($\rightarrow$) e SE-E-SOMENTE-SE ($\leftrightarrow$).

Para proposições x e y quaiquer, expressões do tipo "SE x ENTÃO y" são relativamente comuns, especialmente na matemática. No contexto de cálculo proposicional devemos nos limitar aos valores V e F. Nosso interesse é saber o valor da expressão $x \rightarrow y$. Parece razoável pensar que se x é V e y é V, então a expressão $x \rightarrow y$ é também V. Similarmente, se x é V e y é F, então $x \rightarrow y$ é F. Para completar a definição, associa-se V para $x \rightarrow y$ quando x é F.

Uma outra forma de encarar este condicional é pensar que partindo de uma verdade chega-se a uma verdade. Então "partir de uma verdade e chegar a uma verdade" é verdadeiro enquanto "partir de uma verdade e chegar a uma falsidade" é falso. Já quando se parte de uma falsidade pode-se chegar tanto a uma verdade quanto a uma falsidade.

Representamos expressões do tipo "x se, e somente se, y" por $x \leftrightarrow y$. A expressão $x \leftrightarrow y$ é verdadeira quando x e y tomam o mesmo valor e é equivalente à expressão $(x \rightarrow y) \wedge (y \rightarrow x)$.

Expressão lógica

As proposições podem ser representadas por expressões envolvendo várias variáveis como em $x \wedge y$, $(x \wedge y) \vee \neg z$, etc. As regras para a formação de expressões são:

(1) Qualquer variável (letra) representando uma proposição é uma expressão lógica
(2) Se p e q são expressões lógicas, então $(\neg p)$, $(p \wedge q)$, $(p \vee q)$, $(p \rightarrow q)$ e $(p \leftrightarrow q)$ são expressões lógicas.

Exemplos: Alguns exemplos de expressões lógicas

$(x \rightarrow (y \vee (z \wedge (\neg x))))$

$(x \wedge y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z)$

Os parênteses servem para explicitar as precedências (da mesma forma com que estamos acostumados em relação às expressões aritméticas usuais).

Tabela-verdade

Da mesma forma que proposições simples podem ser ou verdadeiras ou falsas, proposições compostas podem também ser ou verdadeiras ou falsas. O valor-verdade de uma expressão que representa uma proposição composta depende dos valores-verdade das sub-expressões que a compõem e também a forma pela qual elas foram compostas.

Tabelas-verdade são diagramas que explicitam a relação entre os valores-verdade de uma expressão composta em termos dos valores-verdade das subexpressões e variáveis que a compõem. Mostramos a seguir as tabelas-verdade para os conectivos lógicos $\neg$, $\wedge$, e $\vee$. Suponha que x e y são duas variáveis lógicas.

x	$\neg x$
F	V
V	F

x	y	$x \wedge y$
F	F	F
F	V	F
V	F	F
V	V	V

x	y	$x \vee y$
F	F	F
F	V	V
V	F	V
V	V	V

A tabela-verdade lista todas as possíveis combinações de valores-verdade V e F para as variáveis envolvidas na expressão cujo valor lógico deseja-se deduzir. Assim, quando a expressão possui duas variáveis, sua tabela-verdade contém 4 linhas. Em geral, se uma expressão possui n variáveis, sua tabela-verdade contém 2^n linhas.

As tabelas-verdade dos condicionais SE-ENTÃO e SE-E-SOMENTE-SE são mostradas a seguir.

x	y	$x \rightarrow y$
F	F	V
F	V	V
V	F	F
V	V	V

x	y	$x \leftrightarrow y$
F	F	V
F	V	F
V	F	F
V	V	V

Tanto $\rightarrow$ como $\leftrightarrow$ podem ser expressos em termos dos demais conectivos. Por isso, eles poderiam ser considerados não necessários. Porém, a sua utilização é comum devido a conveniência para expressar certas proposições.

Exemplos de tabela-verdade

A tabela verdade da expressão $(x \vee (y \wedge z)) \rightarrow y$ é mostrada a seguir

x	y	z	$y \wedge z$	$x \vee (y \wedge z)$	$(x \vee (y \wedge z)) \rightarrow y$
F	F	F	F	F	V
F	F	V	F	F	V
F	V	F	F	F	V
F	V	V	V	V	V
V	F	F	F	V	F
V	F	V	F	V	F
V	V	F	F	V	V
V	V	V	V	V	V

A mesma tabela pode ser expressa em formas mais concisas, como as mostradas a seguir. Os números na última linha da tabela indicam a ordem na qual as respectivas colunas devem ser preenchidas.

$(x$	$\vee$	$(y$	$\wedge$	$z))$	$\rightarrow$	y
F	F	F	F	F	V	F
F	F	F	F	V	V	F
F	F	V	F	F	V	V
F	V	V	V	V	V	V
V	V	F	F	F	F	F
V	V	F	F	V	F	F
V	V	V	F	F	V	V
V	V	V	F	V	V	V
1	3	1	2	1	4	1

x	y	z	$(x$	$\vee$	$(y$	$\wedge$	$z))$	$\rightarrow$	y
F	F	F		F		F		V	
F	F	V		F		F		V	
F	V	F		F		F		V	
F	V	V		V		V		V	
V	F	F		V		F		F	
V	F	V		V		F		F	
V	V	F		V		F		V	
V	V	V		V		F		V	
1	1	1		3		2		4	

Exercício: Faça a tabela-verdade para as expressões:

a) $\neg(x \wedge y)$
b) $\neg(x \vee y) \rightarrow z$
c) $\neg((x \vee y) \rightarrow z)$
d) $y \wedge \neg(x \vee y)$

Tautologias e contradições

Uma expressão é uma **tautologia** se ela toma valor V para todas as possíveis atribuições de valor V e/ou F para as variáveis presentes nela.

Exemplo: As expressões $x \to x$ e $x \vee \neg x$ são tautologias.

Uma expressão é uma **contradição** se ela toma valor F para todas as possíveis atribuições de valor V e/ou F para as variáveis presentes nela.

Exemplo: Se a expressão x é uma tautologia, então $\neg x$ é uma contradição. Similarmente, se x é uma contradição, então $\neg x$ é uma tautologia.

Exercício: Para cada expressão abaixo, responda se ela é uma tautologia, uma contradição ou nenhuma das duas.

a) $x \wedge \neg x$
b) $(x \to y) \to y) \to y$
c) $(x \wedge \neg y) \vee (\neg x \wedge y)$
d) $(x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y)$
e) $(x \to (y \to z)) \leftrightarrow ((x \wedge y) \to z)$
f) $((x \to y) \vee (y \to z)) \to (x \to (y \vee z))$

Implicação e equivalência lógica

Dizemos que uma expressão x **implica logicamente** uma expressão y se, e somente se, cada atribuição de valor às variáveis que torna x verdadeira torna y verdadeira também. Utilizamos a notação $x \Rightarrow y$ para dizer que x implica logicamente y.

Teorema: Uma expressão x implica logicamente y se, e somente se, $x \to y$ é uma tautologia.

Prova: x implica logicamente y se, e somente se, sempre que x for verdadeira, y também o for. Portanto, x implica logicamente y se, e somente se, nunca se dá o caso em que x é verdadeira e y é falsa. Mas isto significa que a expressão $x \to y$ nunca é falsa, ou seja, que $x \to y$ é uma tautologia.

Duas expressões são **logicamente equivalentes** se a tabela-verdade delas forem iguais. Utilizamos a notação $\Leftrightarrow$.

Teorema: x e y são logicamente equivalentes se, e somente se, $x \leftrightarrow y$ é uma tautologia.

Equivalências lógicas

E1. Comutatividade

(a) $x \vee y \Leftrightarrow y \vee x$

(b) $x \wedge y \Leftrightarrow y \wedge x$

E2. Associatividade

(a) $(x \vee y) \vee z \Leftrightarrow x \vee (y \vee z)$

(b) $(x \wedge y) \wedge z \Leftrightarrow x \wedge (y \wedge z)$

E3. Distributividade

(a) $x \wedge (y \vee z) \Leftrightarrow (x \wedge y) \vee (x \wedge z)$

(b) $x \vee (y \wedge z) \Leftrightarrow (x \vee y) \wedge (x \vee z)$

E4. Idempotência

(a) $x \vee x \Leftrightarrow x$

(b) $x \wedge x \Leftrightarrow x$

E5. Leis de absorção

(a) $x \vee (x \wedge y) \Leftrightarrow x$

(b) $x \wedge (x \vee y) \Leftrightarrow x$

(c) $(x \wedge y) \vee \neg y \Leftrightarrow x \vee \neg y$

(d) $(x \vee y) \wedge \neg y \Leftrightarrow x \wedge \neg y$

E6. Dupla negação

(a) $\neg\neg x \Leftrightarrow x$

E7. Leis de De Morgan

(a) $\neg(x \vee y) \Leftrightarrow (\neg x \wedge \neg y)$

(b) $\neg(x \wedge y) \Leftrightarrow (\neg x \vee \neg y)$

E8. Tautologias e contradições

(a) $(V \wedge x) \Leftrightarrow x$

(b) $(V \vee x) \Leftrightarrow V$

(c) $(F \wedge x) \Leftrightarrow F$

(d) $(F \vee x) \Leftrightarrow x$

Exemplo: Vamos verificar a equivalência E7(a). Para isso montamos a tabela-verdade:

x	y	$\neg$	$(x$	$\vee$	$y)$	$\leftrightarrow$	$(\neg x$	$\wedge$	$\neg y)$
F	F	V		F		V	V	V	V
F	V	F		V		V	V	F	F
V	F	F		V		V	F	F	V
V	V	F		V		V	F	F	F
1	1	3		2		4	2	3	2

Podemos ver que $\neg(x \vee y) \leftrightarrow (\neg x \wedge \neg y)$ é uma tatutologia. Ou ainda, podemos ver que o valor-verdade de $\neg(x \vee y)$ e $(\neg x \wedge \neg y)$ são iguais para todas as linhas da tabela. Logo, $\neg(x \vee y) \Leftrightarrow (\neg x \wedge \neg y)$.

Exercício: Mostre as equivalências E3(a), E5(a), E5(d), E8(a) e E8(c).

Outras equivalências

E9. Contrapositivo

- $x \to y \Leftrightarrow \neg y \to \neg x$

E10. Eliminação de condicionais

(a) $x \to y \Leftrightarrow \neg x \vee y$

(b) $x \to y \Leftrightarrow \neg(x \wedge \neg y)$

E11. Eliminação de bicondicionais

(a) $x \leftrightarrow y \Leftrightarrow (x \wedge y) \vee (\neg x \wedge \neg y)$

(b) $x \leftrightarrow y \Leftrightarrow (\neg x \vee y) \wedge (\neg y \vee x)$

Exercício: Mostre as equivalências E9, E10(a), E10(b), E11(a) e E11(b).

Exercício: Mostre que

a) $(x \wedge y) \vee (x \wedge \neg y) \leftrightarrow x$

b) $(x \to y) \leftrightarrow (\neg y \to \neg x)$ (Prova por contradição)

Algumas implicações lógicas

I1. $p \Rightarrow (p \vee q)$

I2. $(p \wedge q) \Rightarrow p$

I3. $(p \to C) \Rightarrow \neg p$ (C denota uma contradição)

I4. $[p \wedge (p \to q)] \Rightarrow q$

I5. $[(p \to q) \wedge \neg q] \Rightarrow \neg p$

I6. $[(p \vee q) \wedge \neg p] \Rightarrow q$

I7. $p \Rightarrow [q \to (p \wedge q)]$

I8. $[(p \leftrightarrow q) \wedge (q \leftrightarrow r)] \Rightarrow (p \leftrightarrow r)$

I9. $[(p \to q) \wedge (q \to r)] \Rightarrow (p \to r)$

Exercício: Mostre as implicações I1, I3, I4, I6, I8 e I9.

Mais dois conectivos

Barra de Sheffer (Sheffer's stroke): Significando "não ambos verdadeiro", é definido pela seguinte tabela-verdade

x	y	$x\|y$
F	F	V
F	V	V
V	F	V
V	V	F

Negação conjunta (joint denial): Significando "nem um e nem outro", é definido pela seguinte tabela-verdade

x	y	$x \downarrow y$
F	F	V
F	V	F
V	F	F
V	V	F

Exercício: Mostre que $\neg x \Leftrightarrow x|x$ e $\neg x \Leftrightarrow x \downarrow x$.

Exercício: Mostre que $x \vee y \Leftrightarrow (x|x)|(y|y)$ e $x \wedge y \Leftrightarrow (x \downarrow x) \downarrow (y \downarrow y)$.

Redundâncias ou Sistemas adequados de conectivos

Toda expressão determina uma função-verdade que pode ser expressa via tabelas-verdade. Existem $2^{(2^n)}$ funções-verdade de n variáveis já que existem 2^n possíveis atribuições de valor-verdade para essas n variáveis e para cada uma dessas atribuições a função pode tomar valor V ou F.

Teorema: Toda função-verdade pode ser expressa por uma expressão envolvendo apenas os conectivos $\vee$, $\wedge$ e $\neg$.

Um conjunto de conectivos é dito formar um **sistema adequado de conectivos** se toda função-verdade pode ser expressa por expressões que envolvem apenas conectivos do conjunto.

Os seguintes conjuntos são sistemas adequados de conectivos:

a) $\{\vee, \wedge, \neg\}$
b) $\{\vee, \neg\}$
c) $\{\wedge, \neg\}$
d) $\{\neg, \rightarrow\}$
e) $\{|\}$
f) $\{\downarrow\}$

Exemplo: As quatro funções-verdade de uma variável são :

	f_0	f_1	f_2	f_3
x	x	$\neg x$	$x \vee \neg x$	$x \wedge \neg x$
F	F	V	V	F
V	V	F	V	F

Exercício: Liste todas as funções-verdade com duas variáveis.

Métodos de prova

As provas matemáticas com as quais lidamos todos os dias (?) são muito baseadas em elementos da lógica proposicional.

Não é objetivo estudarmos métodos de prova neste curso, mas apenas para dar uma idéia, alguns métodos de prova são apresentados a seguir de forma informal.

Prova direta: É a situação típica em que temos um conjunto de hipóteses $h_1, h_2, \ldots, h_n$ e queremos derivar uma conclusão c. Ou seja, queremos mostrar

$$h_1 \wedge h_2 \wedge \ldots \wedge h_n \Rightarrow c$$

Prova indireta: Temos a prova **contrapositiva**

$$\neg c \Rightarrow \neg(h_1 \wedge h_2 \wedge \ldots \wedge h_n)$$

e a **prova por contradição**

$$h_1 \wedge h_2 \wedge \ldots \wedge h_n \wedge \neg c \Rightarrow \text{uma contradição}$$

Observe ainda que

$$h_1 \wedge h_2 \wedge \ldots \wedge h_n \Rightarrow c$$

é equivalente a

$$(h_1 \Rightarrow c) \text{ e } (h_2 \Rightarrow c) \text{ e } \ldots \text{ e } (h_n \Rightarrow c)$$

que leva-nos à prova por casos.

A idéia de **prova formal** pode ser expressa no contexto da lógica proposicional. Maiores detalhes podem ser obtidos, por exemplo, em [Ross and Wright, 1992].

1.4 Morfologia Matemática Binária

Seja $E = Z^2$ o plano dos números inteiros. Uma função $f : E \rightarrow \{0, 1, 2, \ldots, k-1\}$ define uma **imagem digital monocromática** (em tons de cinza) com k níveis (ou tons) de cinza. Se o contradomínio da função f possui apenas dois valores, então f define uma **imagem binária**. Cada elemento $(x, y) \in E$ corresponde a um ponto (ou pixel) da imagem e o valor da função $f(x, y)$ é o nível de cinza da imagem nesse ponto. Tipicamente, em uma imagem em níveis de cinza utilizam-se 256 tons (de 0 a 255, justamente os números que podem ser armazenados em um *byte*). Em geral, o 0 corresponde ao preto e o 255 ao branco. Numa imagem binária é comum a utilização dos valores 0 e 1 (ou, às vezes, 0 e 255).

Neste texto, estamos interessados apenas em imagens binárias e vamos supor que os valores das imagens são 0 ou 1. Pontos com valor 0 serão denominados *background* (fundo) enquanto os de valor 1 serão denominados *foreground* (objeto, componente).

1.4.1 Imagens Binárias e Conjuntos

Seja $f : E \to \{0,1\}$ uma imagem binária. Então, podemos definir um conjunto $S_f = \{x \in E : f(x) = 1\}$. Obviamente $S_f \subseteq E$. Por outro lado, dado um conjunto $S \subseteq E$, a função indicadora de S é definida por, $\forall x \in E$,

$$1_S(x) = 1 \iff x \in S\,. \tag{1.3}$$

Pode-se mostrar que $1_{S_f} = f$ e $S = S_{1_S}$. Em outras palavras, uma função binária define um conjunto e a função indicadora deste conjunto é a própria função.

1.4.2 Operadores de Imagens Binárias

Seja $\{0,1\}^E$ o conjunto de todas as imagens binárias definidas em E. Um mapeamento $\Psi : \{0,1\}^E \to \{0,1\}^E$ é um operador de imagens binárias, ou seja, um mapeamento que leva imagens binárias em imagens binárias.

Uma vez que imagens binárias em E podem ser entendidos como subconjuntos de E, operadores de imagens binárias podem ser pensadas como mapeamentos do tipo $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$, ou seja, mapeamentos que levam subconjuntos de E em subconjuntos de E.

Do ponto de vista prático, tratar mapeamentos do tipo $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$, mesmo que E seja finito, não é viável. Na prática, utilizam-se mapeamentos que podem ser caracterizados por uma função local. Para descrever tais mapeamentos, precisamos introduzir algumas notações.

Algumas notações e definições básicas

A *origem* de E é denotada por o e a operação usual de adição em E por $+$. O *translado* de um conjunto $S \subseteq E$ por $z \in E$ é denotado S_z e definido por $S_z = \{x + z : x \in S\}$. O *transposto* de S é denotado $\check{S}$ e definido por $\check{S} = \{x \in E : -x \in S\}$.

Seja $W \subseteq E$, um subconjunto especial a ser denominado de *janela*, tal que $o \in W$. Agora imagine a janela sendo deslizada sobre uma imagem binária S. A cada ponto $x \in E$, podemos considerar o subconjunto $S \cap W_x$. Se este subconjunto for transladado para a origem, temos o subconjunto $(S \cap W_x)_{-x} = (S_{-x} \cap W) \subseteq W$. Portanto, podemos considerar uma função binária do tipo $\psi : \mathcal{P}(W) \to \{0,1\}$ e em seguida definir:

$$\Psi(S) = \{x \in E : \psi(S_{-x} \cap W) = 1\}$$

Como os elementos do domínio da função ψ são subconjuntos de W, podemos associar uma variável binária x_i a cada ponto de $w_i \in W$, $i = 1, 2, \ldots, n$ (onde n é o tamanho da janela W). Para cada $x \in E$, podemos então considerar $x_i = 1 \iff w_i \in (S_{-x} \cap W)$.

Resumindo, funções binárias do tipo $f : \{0,1\}^n \to \{0,1\}$ definem um conjunto de mapeamentos de imagens binárias, localmente definidos por uma janela W com n pontos. A figura 1.7 mostra a imagem S, a imagem transformada I, e os subconjuntos (padrões) observados em dois pontos de S e os seus respectivos valores de saída.

Algumas perguntas a serem respondidas são: (1) quais tipos de mapeamentos podems ser localmente definidos? (2) quais propriedades são válidas para essa classe de operadores ? (3) dado um operador, qual a melhor forma de representá-lo (por exemplo, com respeito à eficiência computacional), etc.

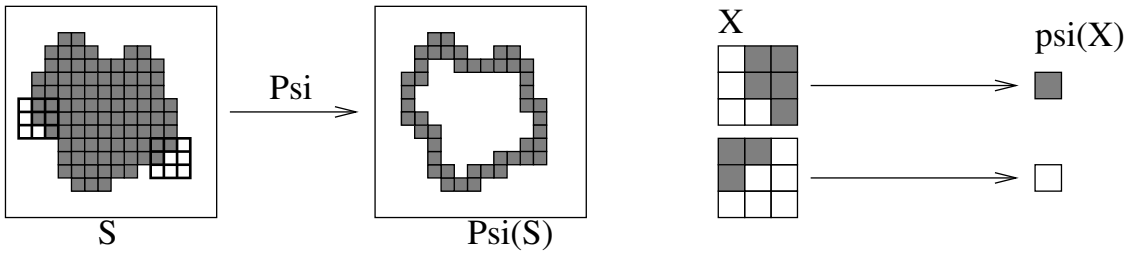


Figura 1.7: Ilustração de um operador localmente definido.

1.4.3 Morfologia Matemática

A morfologia matemática é uma abordagem para processamento de imagens surgida na década de 1960, na França, baseada na exploração das formas (geométricas) presentes em imagens. Ela foi inicialmente desenvolvida para tratar imagens binárias, no contexto de estudo de porosidade de rochas minerais. Mais tarde, ela foi extendida para imagens em níveis de cinza e atualmente há também estudos no contexto de imagens coloridas.

A formalização da morfologia matemática apóia-se na teoria dos reticulados (uma estrutura álgebrica da qual a álgebra booleana é um caso particular). Por enquanto não vamos nos preocupar com os formalismos matemáticos.

O restante deste texto é dedicado à apresentação de alguns operadores morfológicos bastante utilizados, que são do tipo localmente definidos descrito acima. Referências bibliográficas: [**?**] e [**?**].

Os operadores da morfologia matemática são caracterizados por subconjuntos denominados **elementos estruturantes**. A idéia básica é explorar a relação entre este subconjunto e o conjunto que define uma imagem binária. Vamos supor que os elementos estruturantes contém a origem. A seguir, imagens binárias são denotadas por X e elementos estruturantes por B e, a não ser que seja explicitamente mencionado, eles são subconjuntos de E.

Erosão e dilatação

A **erosão** de uma imagem X por um elemento estruturante B é definida por :

$$\varepsilon_B(X) = \{x \in E : B_x \subseteq X\}$$

onde B_x é o conjunto B transladado por x.

A **dilatação** de uma imagem X por um elemento estruturante B é definida por :

$$\delta_B(X) = \{x \in E : \check{B}_x \cap X \neq \emptyset\}$$

Em geral, os elementos estruturantes utilizados são tais que $\check{B} = B$. Portanto, podem ser encontradas definição que não utiliza $\check{B}$.

De uma forma geral, a erosão diminui o tamanho dos objetos presentes na imagem, enquanto a dilatação aumenta. Veja os exemplos na figura 1.8. Note também que a erosão elimina objetos menores que o elemento estruturante, enquanto a dilatação elimina buracos e espaços menores que o elemento estruturante.

Características da erosão:

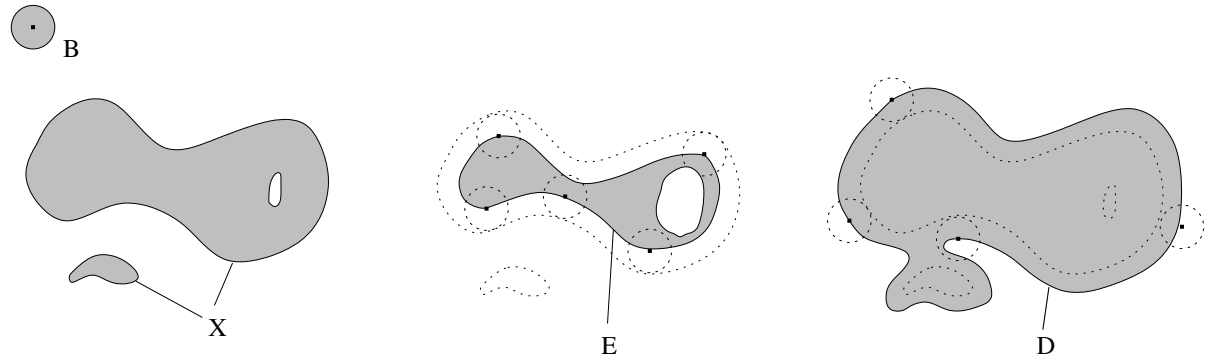


Figura 1.8: Efeitos da erosão e dilatação binárias (E=$\varepsilon_B(X)$ e D=$\delta_B(X)$.

- tamanho dos objetos são reduzidos
- objetos menores que o elemento estruturante são eliminados
- número de componentes pode aumentar

Características da dilatação :

- tamanho dos objetos são aumentados
- buracos menores que o elemento estruturante são eliminados
- número de componentes pode diminuir

Esta dualidade não é apenas coincidência: de fato, há uma relação de dualidade entre estes dois operadores, dada por:

$$\varepsilon_B(X) = (\delta_B(X^c))^c$$

onde $\cdot^c$ representa a operação de complementação usual de conjuntos.

Note também que, se a origem está contida em B, então $\varepsilon_B(X) \subseteq X \subseteq \delta_B(X)$.

Gradiente morfológico e bordas

O operador $\rho_B(X) = \delta_B(X) - \varepsilon_B(X)$ onde $-$ é a operação de diferença entre conjuntos, é denominado **gradiente morfológico**.

O operador $\rho_B^+(X) = \delta_B(X) - X$ é a borda ou **gradiente externo**, e o operador $\rho_B^-(X) = X - \varepsilon_B(X)$ é a borda ou **gradiente interno**.

A figura 1.9[2] ilustra os três tipos de gradiente, pelo elemento estruturante 3×3.

A figura 1.10 ilustra os três tipos de gradiente, para o elemento estruturante 3×3 e cruz, respectivamente.

Note que o gradiente morfológico é a união da borda externa com a borda interna, e portanto pode ser entendida como uma borda "mais grossa".

[2]Nas figuras 1.9, 1.10 e 1.12, os objetos aparecem em branco e o fundo em preto, enquanto que nas demais figuras a região escura corresponde aos objetos.

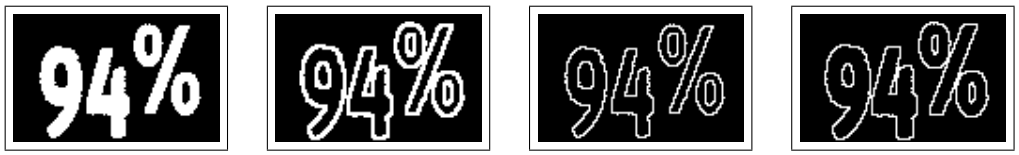


Figura 1.9: Da esquerda para a direita: imagem inicial, gradiente morfológico, gradiente interno e gradiente externo.

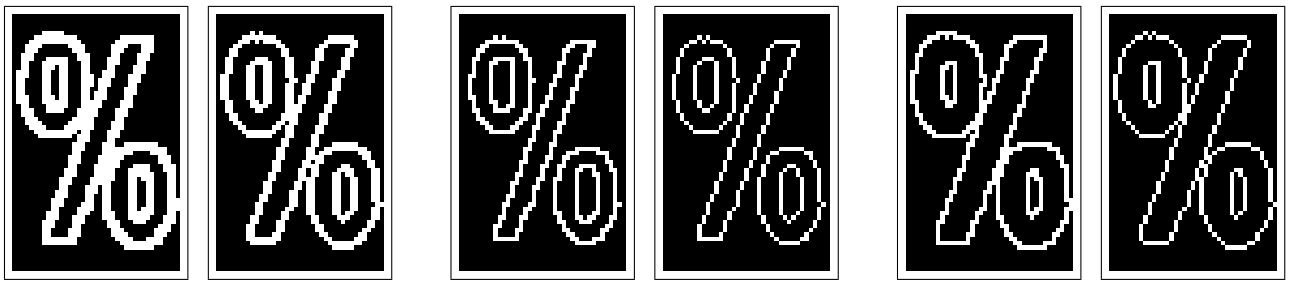


Figura 1.10: Da esquerda para a direita: gradiente morfológico, gradiente interno e gradiente externo (com elemento estruturante 3×3 e cruz, respectivamente).

Em geral, os elementos estruturantes utilizados para extração de borda são a cruz ou o quadrado elementar (quadrado 3×3). Dependendo do elemento estruturante utilizado, as bordas podem apresentar pequenas diferenças. Observe também que a cruz está relacionada com a 4-adjacência, enquanto o quadrado está associado com a 8-adjacência.

Abertura e fechamento

O operador $\gamma_B(X) = \delta_B(\varepsilon_B(X))$ é denominado **abertura** de X por B, e o operador $\varphi_B(X) = \varepsilon_B(\delta_B(X))$ é denominado **fechamento** de X por B.

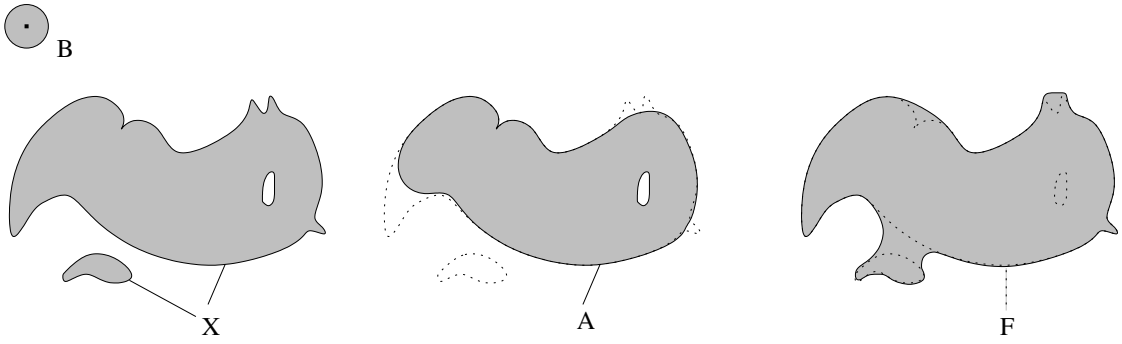


Figura 1.11: Efeitos da aberturta e fechamento binários (A=$\gamma_B(X)$ e F=$\varphi_B(X)$).

Características da abertura:

- objetos menores que o elemento estruturante são eliminados
- partes pequenas do objeto são eliminadas
- partes pequenas do fundo não são eliminadas

Características do fechamento :

- preenche pequenos espaços entre objetos
- preenche pequenos buracos
- preenche reentrâncias estreitas

Estes operadores também possuem uma propriedade de dualidade, isto é: $\gamma_B(X) = [\varphi_B(X^c)]^c$. Além disso, $\gamma_B(X) \subseteq X \subseteq \varphi_B(X)$, $\gamma\gamma = \gamma$ e $\varphi\varphi = \varphi$.

Exemplo ilustrativo

A figura 1.12 mostra o resultado dos quatro operadores (erosão, dilatação, abertura e fechamento) pelo quadrado elementar.

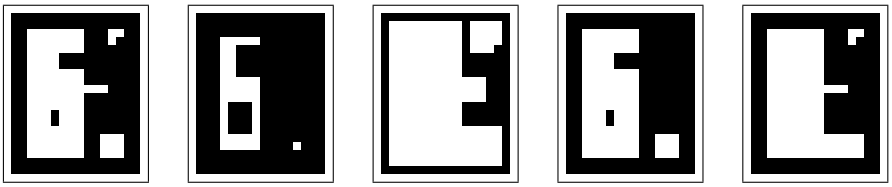


Figura 1.12: Da esquerda para a direita: imagem inicial, erosão, dilatação, abertura e fechamento.

Operador Hit-or-Miss

Operadores **hit-or-miss** são caracterizados por um par (B_1, B_2) de elementos estruturantes. A idéia é que um certo ponto fará parte da imagem resultante se e somente se o elemento B_1 "atinge" totalmente a imagem entrada nesse pixel (está totalmente contido na imagem entrada) e se o elemento B_2 perde totalmente a imagem entrada nesse pixel (está totalmente contido no complemento da imagem entrada).

$$H_{(B_1,B_2)}(X) = \{x \in E : (B_1)_x \subseteq X \text{ e } (B_2)_x \subseteq X^c\}$$

Este operador pode ser escrito também como

$$H_{(B_1,B_2)}(X) = \varepsilon_{B_1}(X) \cap \varepsilon_{B_2}(X^c)$$

Um operador equivalente ao operador hit-or-miss são as **sup-geradoras**. Dados dois elementos estruturantes A e B, tais que $A \subseteq B \subseteq W$, o operador sup-gerador é definido como:

$$\Lambda_{(A,B)}(X) = \{x \in E : A_x \subseteq (X \cap W_x) \subseteq B_x\}$$

Exemplo: detecção de ponto extremo esquerdo em segmentos de reta horizontais, considerando-se a 4-vizinhança. Considere os elementos estruturantes da figura 1.13.

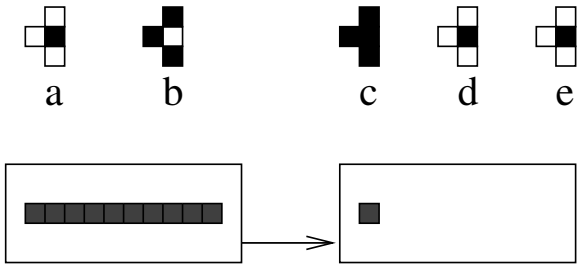


Figura 1.13: Detecção de ponto extremo esquerdo em segmento de reta horizontal. Temos $H_{(B_1,B_2)}(X) = \Lambda_{(A,B)}(X)$. (a=$B_1$, b=$B_2$, c=$W$, d=$A$, e=$B$)

1.4.4 Decomposição canônica

O núcleo de um W-operador $\Psi : \mathcal{P}(E) \to \mathcal{P}(E)$ é dado por:

$$\mathcal{K}_W(\Psi) = \{X \in \mathcal{P}(W) : o \in \Psi(X)\} \tag{1.4}$$

Uma vez que $o \in \Psi(X) \iff \psi(X) = 1$, para todo $X \in \mathcal{P}(W)$, podemos reescrever

$$\mathcal{K}_W(\Psi) = \{X \in \mathcal{P}(W) : \psi(X) = 1\}.$$

O conjunto $[A, B] = \{X \in \mathcal{P}(W) : A \subseteq X \subseteq B\}$ é denominado *intervalo* com extremidades A e B. Se $A = B$ então $[A, B]$ é um *intervalo trivial* (i.e., contém um único elemento) e se $A \not\subseteq B$ então $[A, B]$ é um *intervalo degenerado* (i.e., $[A, B] = \emptyset$).

A *base* de um W-operador Ψ, denotado $\mathcal{B}_W(\Psi)$, é o conjunto de todos os intervalos maximais de $\mathcal{K}_W(\Psi)$. Isto é, $[A, B] \in \mathcal{B}_W(\Psi)$ implica que $\forall [A', B'] \subseteq \mathcal{K}_W(\Psi)$, se $[A, B] \subseteq [A', B']$ então $[A, B] = [A', B']$.

Qualquer W-operador pode ser expresso unicamente em termos de seu núcleo, isto é, como a união de operadores sup-geradores caracterizados pelos intervalos do núcleo [**?**]:

$$\Psi = \bigcup \left\{ \Lambda_{(A,B)} : [A, B] \subseteq \mathcal{K}_W(\Psi) \right\}. \tag{1.5}$$

Em termos de sua base, Ψ pode ser expresso unicamente por :

$$\Psi = \bigcup \left\{ \Lambda_{(A,B)} : [A, B] \in \mathcal{B}(\Psi) \right\}. \tag{1.6}$$

As representações da equação 1.5 e 1.6 são denomindas, respectivamente, por decomposição canônica e decomposição canônica minimal. Como os operadores sup-geradores são expressos em termos de erosões e dilatações, segue que qualquer operador localmente definido pode ser expresso em termos de erosões e dilatações.

Observe que as decomposições acima possuem uma estrutura paralela. Há situações em que expressar um operador em sua estrutura seqüencial (como aberturas e fechamentos, por exemplo) é muito mais conveniente. Porém, o problema de se determinar uma estrutura seqüencial de um operador não é trivial.

1.4.5 Comentários finais

A teoria de Morfologia Matemática binária é formulada sobre a álgebra booleana (ou, equivalentemente, sobre o reticulado booleano). Com um pouco de esforço, não é difícil perceber que as funções de $\mathcal{P}(W)$ em $\{0,1\}$ que caracterizam os operadores localmente definidos são similares às funções dos circuitos digitais. Portanto, todas as perguntas que se aplicam no contexto de projeto de circuitos digitais pode também ser aplicado no contexto de operadores de imagens binárias.

1.5 Discussão

Existe alguma coisa comum entre os tópicos discutidos acima ? Em todos os casos há elementos (sinais 0 ou 1, conjuntos, sentenças, padrões de pixels) que podem ser representados por símbolos. Há operadores (portas lógicas, intersecção, união e complementação, conjunção, disjunção e negação) que se aplicam a estes elementos. Os elementos operados pelos operadores constituem expressões, que correspodem a elementos também. Por exemplo, dois conjuntos X e Y, operados pela operação de união $\cup$ e expresso pela expressão $X \cup Y$, corresponde a um conjunto (ao conjunto $X \cup Y$). Circuitos podem ser simplificados, expressões envolvendo operação entre conjuntos e sentenças lógicas podem ser simplificadas. Há os elementos especiais: 0 1, $\emptyset$ U, F V.

Se analisarmos com cuidado, é possível fazermos analogias entre conceitos vistos em um tópico com os vistos em outro. A abstração dos conceitos comuns aos assuntos apresentados nas seções acima corresponde à álgebra booleana, que será apresentada no próximo capítulo.

Capítulo 2

Álgebra Booleana

Nesta parte 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. Todas essas leis podem ser derivadas algebricamente a partir dos postulados.

Para as formalizações apresentadas aqui, procure associar os equivalentes vistos na parte de álgebra dos conjuntos. Recomenda-se também que o leitor faça o inverso: prestar atenção como os conceitos apresentados via álgebra de conjunto podem ser formalizados (tratados de forma abstrata).

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

2.1 Definição axiomática

Uma forma utilizada para definir a álgebra booleana é através de um conjunto de axiomas (postulados). Os axiomas apresentados a seguir foram elaborados por Huntington em 1904.

Axioma 1: Existe um conjunto A, sujeito a uma relação de equivalência denotada $=$, que satisfaz o princípio da substituição. Por substituição, entendemos que se $x = y$ então x pode ser substituído por y em qualquer expressão que envolve x, sem alterar o valor da expressão.

Axioma 2: Há duas operações binárias, $+$ e $\cdot$, definidas em A, tais que $x + y$ e $x \cdot y$ estão em A sempre que x e y estão em A.

Outros quatro axiomas são as seguintes propriedades:

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. Existem em A **elementos identidade** 0 e 1, distintos, com relação às operações $+$ e $\cdot$, respectivamente. Ou seja, para todo $x \in A$,

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

A partir disto podemos dizer que há pelo menos dois elementos distintos em A.

A4. Para cada elemento $x \in A$ existe um elemento $\overline{x}$ em A tal que

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

O elemento $\overline{x}$ será chamado **complemento** de x.

Denotaremos uma álgebra booleana por uma sextupla ordenada. No caso da definição acima, temos a álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$.

Observação: Alguns autores incorporam outras propriedades como parte da definição de uma álgebra booleana. Vale registrar que os postulados de Huntington 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 propriedade de uma álgebra booleana pode ser derivada/provada a partir desses postulados. 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.

2.2 Exemplos de álgebra booleana

Exemplo 1: O conjunto $B = \{0, 1\}$ onde definimos

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

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

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

é uma álgebra booleana.

Os axiomas A1, A3 e A4 são satisfeitos por definição. Para verificar o axioma A2 podemos construir uma tabela verdade com todas as possíveis combinações de valores para x, y e z. Vejamos a validade da distributividade em relaçao 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 $\langle B, +, \cdot, \bar{}, 0, 1\rangle$. 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, $\mathcal{P}(S) = \{X : X \subseteq S\}$. Então, $\langle \mathcal{P}(S), \cup, \cap, ^c, \emptyset, S\rangle$ é uma álgebra booleana.

Como já vimos na parte de álgebra dos conjuntos (seção 1.2), os equivalentes aos 4 postulados são:

A1. $X \cup Y = Y \cup X \quad$ e $\quad X \cap Y = Y \cap X$

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

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

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

Exemplo 3: A lógica (ou cálculo) proposicional (veja seção 1.3 para maiores detalhes) é uma álgebra booleana. De fato, ela tem uma correspondência um-para-um com $\langle B, +, \cdot, \bar{}, 0, 1\rangle$, conforme mostrado a seguir:

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

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

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

x	y	$\overline{x}$	$x + y$	$x \cdot y$
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

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

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

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

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

é uma álgebra booleana.

2.3 Leis fundamentais da álgebra booleana

Princípio da dualidade: Cada expressão ou identidade algébrica dedutível a partir dos postulados em uma álgebra booleana continua válida se todas as ocorrências dos operadores + e · e os elementos identidade 0 e 1 são trocados um pelo outro.

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

$$\begin{array}{lccccccc} & x & \cdot & y & = & y & \cdot & x \\ \text{Axioma A1} & & \downarrow & & & & \downarrow & \\ & x & + & y & = & y & + & x \end{array}$$

$$\begin{array}{lccccccccccccc} & x & \cdot & (y & + & z) & = & (x & \cdot & y) & + & (x & \cdot & z) \\ \text{Axioma A2} & & \downarrow & & \downarrow & & & & \downarrow & & \downarrow & & \downarrow & \\ & x & + & (y & \cdot & z) & = & (x & + & y) & \cdot & (x & + & z) \end{array}$$

$$\begin{array}{lccccc} & x & + & 0 & = & x \\ \text{Axioma A3} & & \downarrow & \downarrow & & \\ & x & \cdot & 1 & = & x \end{array}$$

$$\begin{array}{lccccc} & x & + & \overline{x} & = & 1 \\ \text{Axioma A4} & & \downarrow & & & \downarrow \\ & x & \cdot & \overline{x} & = & 0 \end{array}$$

Assim, se na prova de uma proposição E trocarmos cada derivação pela sua dual obtemos uma outra prova (válida, pois axiomas são trocadas por axiomas). Esta nova prova é uma prova da dual de E.

Desta parte em diante omitiremos o símbolo · 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: Suponha que existem dois elementos zero, 0_1 e 0_2. Sejam x_1 e x_2 dois elementos quaisquer em A. Por A3, temos que

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

Tome, em particular, $x_1 = 0_2$ e $x_2 = 0_1$. Assim temos

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

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 $x \in A$, $\quad x + x = x$ e $\quad xx = x$.

PROVA:

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

[**Identidade**] Para todo $x \in A$, $x+1=1 \quad \text{e} \quad x0=0$.

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

[**Complemento do um (zero)**] $\overline{1}=0 \quad \text{e} \quad \overline{0}=1$.

$$\begin{array}{rcll} \overline{1} & = & \overline{1}\cdot 1 & (A3) \\ & = & 0 & (A4) \end{array}$$

[**Absorção**] Para todo $x, y \in A$, $x+xy=x \quad \text{e} \quad x(x+y)=x$.

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

[**Unicidade de $\overline{x}$**] O inverso de qualquer elemento $x \in A$ é único, isto é, se $x+y=1 \;\text{ e }\; x\,y=0$ para algum $y \in A$, então $y=\overline{x}$.

PROVA: Por contradição. Suponha que existem dois elementos distintos $\overline{x}_1$ e $\overline{x}_2$ em A tais que

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

$$\begin{array}{rcll} \overline{x}_2 & = & 1\cdot\overline{x}_2 & (A3) \\ & = & (x+\overline{x}_1)\,\overline{x}_2 & \text{(hipótese)} \\ & = & x\,\overline{x}_2+\overline{x}_1\,\overline{x}_2 & (A2) \\ & = & 0+\overline{x}_1\,\overline{x}_2 & \text{(hipótese)} \\ & = & x\,\overline{x}_1+\overline{x}_1\,\overline{x}_2 & \text{(hipótese)} \\ & = & (x+\overline{x}_2)\,\overline{x}_1 & (A2) \\ & = & 1\cdot\overline{x}_1 & \text{(hipótese)} \\ & = & \overline{x}_1 & (A3) \end{array}$$

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

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

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

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

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

Usando o lema acima, provaremos a propriedade associativa. Seja

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

De forma similar,

$$\begin{array}{rcll} Z & = & (x+y)[x+(y+z)]+z[x+(y+z)] & (A2) \\ & = & (x+y)[x+(y+z)]+z & (lema) \\ & = & \{x[x+(y+z)]+y[x+(y+z)]\}+z & (A2) \\ & = & \{x[x+(y+z)]+y\}+z & (lema) \\ & = & (x+y)+z & (lema) \end{array}$$

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

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

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

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

$$\begin{array}{rcll} (x+y)\cdot\overline{x}\,\overline{y} & = & x(\overline{x}\,\overline{y})+y(\overline{y}\,\overline{x}) & (A2+A1) \\ & = & (x\overline{x})\,\overline{y}+(y\overline{y})\,\overline{x} & (associativa) \\ & = & 0+0 & (A4+Identidade) \\ & = & 0 & (A3) \end{array}$$

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. □

Note a similaridade destas propriedades com as propriedades dos conjuntos (e também com as da lógica proposicional). Enquanto lá fizemos uso dos diagramas de Venn e das tabelas-verdade, respectivamente, para nos convencermos da validade das propriedades, aqui as demonstrações são algébricas.

2.4 Relações de Ordem Parciais

2.4.1 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 2.1.

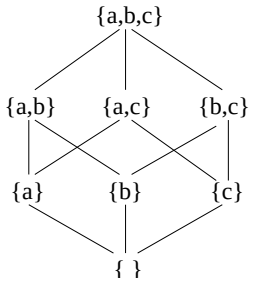


Figura 2.1: Diagrama de Hasse de $(\{a, b, c\}, \subseteq)$.

2.4.2 Elementos notáveis de um poset

Menor e maior elementos: Seja $(A, \leq)$ um poset.

- Chama-se **menor elemento** de A um elemento $0 \in A$ tal que $0 \leq x$ para todo $x \in A$ (também denotado $m(A)$).
- Chama-se **maior elemento** de A um elemento $1 \in A$ tal que $x \leq 1$ para todo $x \in A$ (também denotado $M(A)$).

Teorema: Em um poset $(A, \leq)$, se existe um menor elemento, ele é único. Similarmente, se existe um maior elemento, ele é único.

PROVA: Suponha que 0_1 e 0_2 são menores elementos de $(A, \leq)$. Então, teríamos $0_1 \leq 0_2$ (pois 0_1 é menor elemento) e $0_2 \leq 0_1$ (pois 0_2 é menor elemento). Pela propriedade anti-simétrica da relação $\leq$, concluímos que $0_1 = 0_2$. A unicidade do maior elemento segue de forma análoga. □

Exemplo: Dado um conjunto não vazio S, no poset $(\mathcal{P}(S), \subseteq)$ o menor elemento é o $\emptyset$ (conjunto vazio) e o maior elemento é o S (conjunto universo).

Exemplo: No poset do diagrama abaixo, o maior elemento é a e não há um menor elemento.

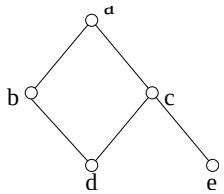


Figura 2.2: Diagrama de um poset.

Elementos minimais e maximais:

- Chama-se **elemento minimal** de A um elemento $a \in A$ tal que elemento algum de A precede estritamente a, isto é, para qualquer $x \in A$, se $x \leq a$ então $x = a$. O conjunto dos elementos minimais de A é denotado $min(A)$.
- Chama-se **elemento maximal** de A um elemento $a \in A$ tal que elemento algum de A sucede estritamente a, isto é, para qualquer $x \in A$, se $x \geq a$ então $x = a$. O conjunto dos elementos maximais de A é denotado $max(A)$.

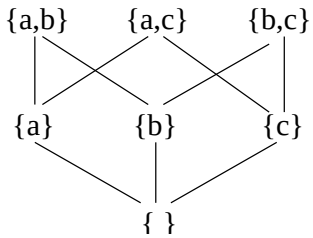
Exemplo: No poset da figura 2.2, os elementos minimais são d e e enquanto o único elemento maximal é a.

Teorema: Se 0 é o menor elemento de $(A, \leq)$, então 0 é o único elemento minimal de $(A, \leq)$.

PROVA: Seja 0 o menor elemento de A e suponha que $a \leq 0$ onde $a \in A$. Como 0 é o menor elemento em A, sabemos também que $0 \leq a$. Portanto, pela anti-simetria de $\leq$, temos que $a = 0$. Logo, 0 é minimal.

Suponha que x é um elemento minimal em A. Como 0 é o menor elemento, temos que $0 \leq x$. Por x ser minimal, temos $x = 0$ e, portanto, 0 é o único elemento minimal. □

Exemplo: Seja $S = \{a, b, c\}$ e considere o conjunto de todos os subconjuntos próprios de S, conforme ilustrado no diagrama abaixo.



Este conjunto tem três elementos maximais, $\{a, b\}$, $\{a, c\}$ e $\{b, c\}$. O menor elemento deste conjunto é $\{\}$ e, conforme teorema acima, é o único elemento minimal. O conjunto não possui maior elemento.

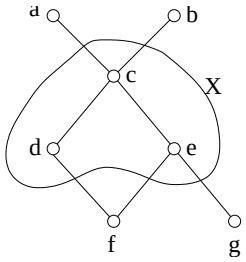
Limitantes inferiores e superiores; Ínfimo e supremo:

Seja $(A, \leq)$ um poset e seja $X \subset A$.

- Chama-se **limitante inferior** de X em A a todo elemento $a \in A$ tal que $a \leq x$ para todo $x \in X$. O conjunto dos limitantes inferiores de X em A é denotado por $li(X)$.
- Chama-se **limitante superior** de X em A a todo elemento $a \in A$ tal que $x \leq a$ para todo $x \in X$. O conjunto dos limitantes superiores de X em A é denotado por $ls(X)$.
- Chama-se **ínfimo** de X em A o maior elemento dos limitantes inferiores de X em A. O ínfimo de X é denotado $\bigwedge X$. O ínfimo de $\{x, y\}$ é denotado $x \wedge y$. O operador $\wedge$ é chamado *conjunção*.

- Chama-se **supremo** de X em A o menor elemento dos limitantes superiores de X em A. O supremo de X é denotado $\bigvee X$. O supremo de $\{x, y\}$ é denotado $x \vee y$. O operador $\vee$ é chamado *união*.

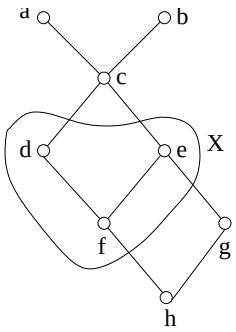
Exemplo:



Elementos minimais: $min(A) = \{f, g\}$
Menor elemento de A: não existe
Elementos maximais: $max(A) = \{a, b\}$
Maior elemento de A: não existe

$X = \{c, d, e\}$
Limitantes inferiores de X: $li(X) = \{f\}$
ínfimo de X: $\bigwedge X = f$
Limitantes superiores de X: $ls(X) = \{a, b, c\}$
Supremo de S: $\bigvee X = c$

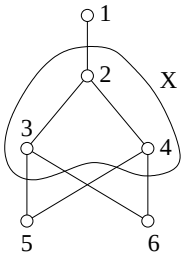
Exemplo:



Elementos minimais: $min(A) = \{h\}$
Menor elemento de A: $m(A) = h$
Elementos maximais: $max(A) = \{a, b\}$
Maior elemento de A: não existe

$X = \{d, e, f\}$
Limitantes inferiores de X: $li(X) = \{f, h\}$
ínfimo de X: $\bigwedge X = f$
Limitantes superiores de X: $ls(X) = \{a, b, c\}$
Supremo de S: $\bigvee X = c$

Exemplo:



Elementos minimais: $min(A) = \{5, 6\}$
Menor elemento de A: não existe
Elementos maximais: $max(A) = \{1\}$
Maior elemento de A: $M(A) = 1$

$X = \{2, 3, 4\}$
Limitantes inferiores de X: $li(X) = \{5, 6\}$
ínfimo de X: não existe
Limitantes superiores de X: $ls(X) = \{1, 2\}$
Supremo de S: $sup(X) = 2$

2.4.3 Reticulados

Um poset $(A, \leq)$ no qual cada par de elementos possui um ínfimo e um supremo em A (ou seja, para quaisquer $x, y \in A$, $x \wedge y$ e $x \vee y$ estão em A) é um **reticulado**.

Um reticulado $(A, \leq)$ é **completo** se todo subconjunto não vazio X de A possui um ínfimo e um supremo em A. Qualquer reticulado completo possui menor elemento 0 e maior elemento 1. Qualquer reticulado finito é completo.

Um reticulado $(A, \leq)$ é **distributivo** se as operações de união ($\vee$) e conjunção ($\wedge$) são distributivas, isto é, para quaisquer x, y e z em A,

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \quad \text{e} \quad x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z).$$

Um reticulado $(A, \leq)$, com menor elemento 0 e maior elemento 1, é **complementado** se todo elemento em A possui um complemento, isto é, para todo $x \in A$ existe $\overline{x}$ tal que $x \vee \overline{x} = 1$ e $x \wedge \overline{x} = 0$.

Teorema: Um reticulado $(A, \leq)$, distributivo e complementado, é uma álgebra booleana (também chamado de reticulado booleano).

PROVA: Considere um reticulado distributivo e complementado $(A, \leq)$ com as operações de união $\vee$ e conjunção $\wedge$ como definidos acima, e complementação $\bar{}$. Claramente $\vee$ e $\wedge$ definem operações binárias em A. Sejam ainda 0 e 1 o menor e o maior elementos de $(A, \leq)$.

Vamos verificar que os axiomas de Huntington são satisfeitos. As operações $\vee$ e $\wedge$ são comutativas por definição; são distributivas pois o reticulado é distributivo; o axioma 4 é satisfeito pois o reticulado é complementado, e o axioma 3 é satisfeito pois $x \vee 0 = x$ e $x \wedge 1 = x$ para qualquer $x \in A$. $\square$

2.5 Relação de ordem e álgebra booleana

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

$$\forall x, y \in A, \quad x \leq y \ \text{ se e somente se } \ x + y = y \tag{2.1}$$

A relação $\leq$ definida pela equação 2.1 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

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

Logo, $x \leq z$.

2.5.1 Toda álgebra booleana é um reticulado

Teorema: Sejam x, y elementos de uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$. Então os elementos $x + y$ e xy são respectivamente o supremo e o ínfimo de $\{x, y\}$.

Teorema: Toda álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$ é um reticulado.

PROVA: Qualquer subconjunto $\{x, y\}$ de A tem supremo $x + y$ e ínfimo xy (teorema anterior). Logo $\langle A, +, \cdot, \bar{}, 0, 1\rangle$ é um reticulado. $\square$

2.5.2 Átomos

Um **átomo** de uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$ é um elemento não nulo x que não pode ser expresso na forma $x = y + z$ com $y \neq x$ e $z \neq x$.

Exemplo 1: Os átomos de $\langle \mathcal{P}(S), \cup, \cap, ^c, \emptyset, S\rangle$ são todos os conjuntos unitários.

Exemplo 2: A álgebra booleana relativa ao conjunto B^n de todas as n-uplas binárias tem como átomos as n-uplas com exatamente uma coordenada igual a 1.

Teorema: Um elemento não nulo x de uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$ é um **átomo** se e somente se não há elemento y em A tal que $0 < y < x$.

PROVA:

(se x é um átomo então não há elemento y em A tal que $0 < y < x$) Suponha que x é um átomo e que $y < x$. Então, $x = x\,1 = (y+x)\cdot(y+\overline{y}) = y+(x\,\overline{y})$. Como x é um átomo, então ou $y = x$ ou $(x\,\overline{y}) = x$. Como $x \neq y$ por hipótese, então $(x\,\overline{y}) = x$. Consequentemente $y = x\,y = (x\,\overline{y})\,y = x\,(\overline{y}\,y) = x\,0 = 0$.

(se não há elemento y em A tal que $0 < y < x$ então x é um átomo) Agora suponha que não há elemento y em A tal que $0 < y < x$ e (suponha por absurdo) que x não é um átomo. Então, $x = y + z$ para y e z diferentes de x e, como $y \leq y + z = x$, temos que $y < x$. Além disso, $y = 0$ (pois caso contrário teríamos $0 < y$, uma contradição). Logo, $x = 0 + z = z \neq x$ (absurdo). □

Teorema: Seja $\langle A, +, \cdot, \bar{}, 0, 1\rangle$ uma álgebra booleana finita com conjunto de átomos $\{a_1, a_2, \ldots, a_n\}$. Cada elemento não nulo x de A pode ser escrito como o supremo de um conjunto de átomos

$$x = a_{i_1} + a_{i_2} + \cdots + a_{i_k}\,.$$

Mais ainda, tal expressão é única, a menos da ordem dos átomos.

PROVA: A demonstração é por contradição. Suponha que existem elementos que não são expressos como supremo de átomos. Seja x um desses elementos. Então como x não é atomo, temos que $x = y + z$ para algum $y < x$ e $z < x$, e mais ainda, pelo menos y ou z não são átomos. Supondo que y não é átomo, temos que ele é supremo de dois elementos menores que ele, dos quais pelo menos um não é átomo. Assim, há uma seqüência de elementos não-átomos $x_0 = x > x_1 = y > x_2 > \ldots$. Mas, como A é finito, haverão índices k e m, com $k < m$ tal que $x_k = x_m$. Pela transitividade de $<$ em $x_k > x_{k+1} > \ldots > x_m$ segue que $x_k > x_m$, contradizendo $x_k = x_m$. Portanto, podemos concluir que todos os elementos não-nulos em A podem ser expressos como supremo de átomos.

Para mostrar a unicidade, primeiro devemos mostrar que

$$x = \bigvee \{a \in A : a \text{ é átomo e } a \leq x\} \tag{2.2}$$

isto é, x pode ser expresso como o supremo de todos os átomos menores ou igual a ele. (A demonstração fica como exercício).

Agora, suponha que $x = a_{i_1} + a_{i_2} + \cdots + a_{i_k}$ é uma expressão de x como supremo de átomos. Então temos que $a_{i_j} \leq x$, e, portanto, $a_{i_j} \in \{a \in A : a \text{ é átomo}, a \leq x\}$. Agora, seja a um átomo tal que $a \in \{a \in A : a \text{ é átomo e } a \leq x\}$. Então, como a é átomo e $a \leq x$,

$$0 \neq a = a\,x = a\,(a_{i_1} + a_{i_2} + \cdots + a_{i_k}) = a\,a_{i_1} + a\,a_{i_2} + \cdots + a\,a_{i_k}$$

Pelo menos um $a\,a_{i_j}$ precisa ser diferente de zero. Logo, $a\,a_{i_j} = a = a_{i_j}$ uma vez que ambos são átomos. Ou seja, a é um dos átomos em $\{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$. □

2.5.3 Isomorfismo de álgebras booleanas

Uma função bijetora ϕ entre duas álgebras booleanas $\langle A_1, +_1, \cdot_1, \bar{}_1, 0_1, 1_1\rangle$ e $\langle A_2, +_2, \cdot_2, \bar{}_2, 0_2, 1_2\rangle$ que satisfaz

$$\phi(x +_1 y) = \phi(x) +_2 \phi(y)$$

$$\phi(x \cdot_1 y) = \phi(x) \cdot_2 \phi(y)$$

e

$$\phi(\overline{x}) = \overline{\phi(x)}$$

é um **isomorfismo** de álgebra booleana.

Duas álgebras booleanas são isomorfas se existe um isomorfismo entre elas.

Teorema: Sejam duas álgebras booleanas finitas com o mesmo número de átomos e sejam $\{a_1, a_2, \ldots, a_n\}$ e $\{b_1, b_2, \ldots, b_n\}$ respectivamente os seus conjuntos de átomos. Então, existe um isomorfismo ϕ entre eles tal que $\phi(a_i) = b_i$, para todo $i \in \{1, 2, \ldots, n\}$.

Teorema: Qualquer álgebra booleana finita com n átomos é isomorfa à álgebra booleana $\langle \mathcal{P}(S), \cup, \cap, {}^c, \emptyset, S\rangle$ onde S é um conjunto com n elementos.

2.5.4 Comentários finais sobre ordens parciais

Os principais conceitos vistos nesta seção são sumarizados através da especialização para os dois seguintes exemplos.

1) Dado um conjunto não vazio S, considere a álgebra booleana $\langle \mathcal{P}(S), \cup, \cap, ^c, \emptyset, S\rangle$.

- A seguinte relação binária é definida em $\mathcal{P}(S)$:

$$\forall X, Y \in \mathcal{P}(S), \quad X \subseteq Y \iff X \cup Y = Y$$

- a relação $\subseteq$ é uma ordem parcial. Logo, $(\mathcal{P}(S), \subseteq)$ é um poset
- o maior elemento de $\mathcal{P}(S)$ é S

- o menor elemento de $\mathcal{P}(S)$ é $\emptyset$
- o supremo de dois conjuntos $X, Y \in \mathcal{P}(S)$ é dado por $X \cup Y$
- o ínfimo de dois conjuntos $X, Y \in \mathcal{P}(S)$ é dado por $X \cap Y$
- $(\mathcal{P}(S), \subseteq)$ é um reticulado booleano (distributivo e complementado)
- os átomos de $\langle \mathcal{P}(S), \cup, \cap, ^c, \emptyset, S \rangle$ são os conjuntos unitários.
- qualquer elemento de $\mathcal{P}(S)$ pode ser expresso como união de conjuntos unitários

2) Considere a álgebra booleana $\langle B, +, \cdot, \bar{}, 0, 1 \rangle$ e o conjunto das funções booleanas $\mathcal{B}$ de n variáveis. Seja $\preceq$ uma relação em $\mathcal{B}$ definida para quaisquer $f, g \in \mathcal{B}$ da seguinte forma:

$$f \preceq g \iff f(x_1, x_2, \ldots, x_n) \leq g(x_1, x_2, \ldots, x_n) \text{ para todo } (x_1, x_2, \ldots, x_n) \in B^n$$

- Verifique que $(\mathcal{B}, \preceq)$ é um conjunto parcialmente ordenado.
- A função constante zero $f_0(x_1, x_2, \ldots, x_n) = \mathbf{0} \in \mathcal{B}$ é o menor elemento de $(\mathcal{B}, \preceq)$.
- A função constante um $f_1(x_1, x_2, \ldots, x_n) = \mathbf{1} \in \mathcal{B}$ é o maior elemento de $(\mathcal{B}, \preceq)$.
- O supremo de duas funções $f, g \in \mathcal{B}$ é dado por

$$(f + g)(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n) + g(x_1, x_2, \ldots, x_n)$$

- O ínfimo de duas funções $f, g \in \mathcal{B}$ é dado por

$$(f \cdot g)(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n) \cdot g(x_1, x_2, \ldots, x_n)$$

- O complemento de uma função $f \in \mathcal{B}$ é dado por

$$\overline{f}(x_1, x_2, \ldots, x_n) = \overline{f(x_1, x_2, \ldots, x_n)}$$

- Quem são os átomos de $\mathcal{B}$?

Exercícios:

1. Mostre que o conjunto B^n mais as operações definidas no exemplo 4 da página 31 é uma álgebra booleana.
2. Considere o conjunto dos números reais R, juntamente com as operações usuais de adição e multiplicação. Quais dos axiomas A1, A2, A3 não são satisfeitos ? É possível definir uma operação unária em R tal que o axioma A4 seja satisfeito ?

3. Seja $A = \{1, 2, 3, 5, 6, 10, 15, 30\}$, ou seja, o conjunto de divisores de 30. Defina operações binárias $+$ e $\cdot$ e uma operação unária $^-$ da seguinte forma: para quaiquer $a_1, a_2 \in A$,

$$a_1 + a_2 = \text{o mínimo múltiplo comum entre } a_1 \text{ e } a_2$$

$$a_1 \cdot a_2 = \text{o máximo divisor comum entre } a_1 \text{ e } a_2$$

$$\overline{a}_1 = 30/a_1$$

Quais são os elementos identidade com respeito a $+$ e $\cdot$? Mostre que A, com as três operações acima, é uma álgebra booleana.

4. Prove as seguintes igualdades

a) $x + \overline{x}y = x + y$ (e sua dual $x(\overline{x} + y) = xy$)

b) $x + y = \overline{\overline{x}\,\overline{y}}$ (e sua dual $xy = \overline{\overline{x} + \overline{y}}$)

c) $(x + y)(x + \overline{y}) = x$ (e sua dual $xy + x\overline{y} = x$)

d) (Teorema do consenso) $xy+yz+\overline{x}z = xy+\overline{x}z$ (ou o dual $(x+y)(y+z)(\overline{x}+z) = (x+y)(\overline{x}+z)$)

e) $yx = zx$ e $y\overline{x} = z\overline{x}$ implica que $y = z$

f) $(x + y + z)(x + y) = x + y$

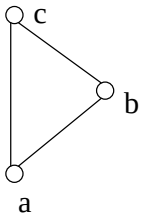
5. Simplifique as seguintes expressões

a) $y\overline{z}(\overline{z} + \overline{z}x) + (\overline{x} + \overline{y})(\overline{x}y + \overline{x}z)$

b) $x + xyz + yz\overline{x} + wx + \overline{w}x + \overline{x}y$

6. Mostre que em qualquer álgebra booleana $\langle A, +, \cdot, ^-, 0, 1\rangle$, $x\,\overline{y} = 0$ se, e somente se, $x\,y = x$.

7. Mostre que a configuração abaixo não pode ocorrer no diagrama de Hasse de nenhum poset.



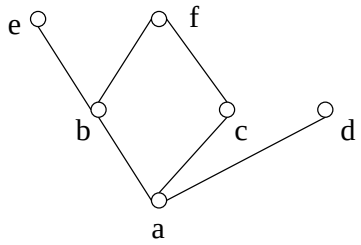
8. Seja R uma ordem parcial sobre A e seja $X \subseteq A$. Mostre que $S = R \cap (X \times X)$ é uma relação de ordem parcial (em outras palavras, qualquer subconjunto de um poset é também um poset).

9. Seja a relação de divisibilidade $|$ definida sobre os inteiros positivos da seguinte forma: para quaisquer x, y inteiros positivos, $x|y$ se, e somente se, x divide y.

a) Mostre que $|$ é uma relação de ordem parcial

b) Desenhe o diagram de Hasse de $\{1, 2, 3, 4, 6, 12\}$ com respeito à relação parcial $|$.

10. Liste os elementos da relação de ordem cujo diagrama de Hasse é o seguinte:



11. Mostre que se 1 é o maior elemento de $(A, \leq)$, então 1 é o único elemento maximal de $(A, \leq)$.

12. Mostre por indução que todo subconjunto finito de um reticulado tem um ínfimo e um supremo.

13. Sejam x, y elementos de uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$. Mostre que os elementos $x + y$ e xy são respectivamente o supremo e o ínfimo de $\{x, y\}$.

14. Mostre que em qualquer álgebra booleana, $x + y = y$ se,e somente se, $xy = x$. Expresse essa relação na álgebra dos conjuntos.

15. Mostre que em uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$, se $x \leq y$ então $x + y = y$ e $x\,y = x$, para todo $x, y \in A$.

16. Mostre que em uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$, se $y \leq z$ então $x\,y \leq x\,z$ e $x + y \leq x + z$ para todo $x \in A$.

17. Mostre que em qualquer álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$

    a) $xy \leq x \leq x + y$, para todo x e y em A,

    b) $0 \leq x \leq 1$, para todo x em A.

18. Seja $\langle A, +, \cdot, \bar{}, 0, 1\rangle$ uma álgebra booleana finita com conjunto de átomos $\{a_1, a_2, \ldots, a_n\}$. Sabendo que cada elemento não nulo x de A pode ser escrito como o supremo de um conjunto de átomos

$$x = a_{i_1} + a_{i_2} + \cdots + a_{i_k}$$

    mostre que, mais especificamente,

$$x = \bigvee \{a \in A : a \text{ é átomo e } a \leq x\}$$

    isto é, x é o supremo de todos os átomos menores ou igual a ele.

    Dica: Comece expressando 1 como um supremo de átomos e use o fato de que $x = x\,1$.

Capítulo 3

Expressões e Funções Booleanas

3.1 Expressões Booleanas

Variáveis e literais: Dada uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$, uma **variável booleana** é uma variável que toma valores em A.

Dada uma variável booleana x, o **complemento** de x, denotado $\overline{x}$, é uma variável booleana tal que $\overline{x} = \overline{a}$ sempre que $x = a$ para qualquer $a \in A$.

Um **literal** é uma variável booleana x ou o seu complemento $\overline{x}$.

Expressões booleanas: Dada uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$, as seguintes são expressões booleanas em n váriáveis $x_1, x_2, \ldots, x_n$:

- os elementos 0 e 1,
- as variáveis booleanas $x_1, x_2, \ldots, x_n$,
- $x + y$, $x \cdot y$, $\overline{x}$, onde x e y são expressões booleanas nas variáveis $x_1, x_2, \ldots, x_n$.

Observe que uma expressão booleana em n váriáveis $x_1, x_2, \ldots, x_n$ não necessariamente precisa conter todas as n variáveis.

Se uma expressão pode ser derivada a partir de outra aplicando-se um número finito de vezes as regras (leis/propriedades) da álgebra booleana, então elas são ditas **equivalentes**. O valor de expressões equivalentes, para cada atribuição de valores às variáveis booleanas, é o mesmo.

Expressões booleanas definem uma função. Expressões equivalentes definem uma mesma função.

3.2 Funções booleanas

Dada uma álgebra booleana $\langle A, +, \cdot, \bar{}, 0, 1\rangle$, uma expressão booleana em n variáveis x_1, x_2, ..., x_n define uma **função booleana** $f : A^n \to A$. O valor da função f para um elemento $a = (a_1, a_2, \ldots, a_n) \in A^n$ é calculado substituindo-se cada ocorrência de x_i na expressão por a_i, para $i = 1, 2, \ldots, n$, e calculando-se o valor da expressão.

Seja $A(n)$ o conjunto de todas as funções booleanas em A com n variáveis e seja $\leq$ uma relação definida em $A(n)$ da seguinte forma:

$$f \leq g \iff f(\mathbf{a}) \leq g(\mathbf{a}),\ \forall \mathbf{a} \in A^n\,. \tag{3.1}$$

Seja $(f \cdot g)(\mathbf{a}) = f(\mathbf{a}) \cdot g(\mathbf{a})$, $(f+g)(\mathbf{a}) = f(\mathbf{a}) + g(\mathbf{a})$, e $\overline{f}(\mathbf{a}) = \overline{f(\mathbf{a})}$, $\forall \mathbf{a} \in A^n$. Fazendo $\mathbf{0}(\mathbf{a}) = 0$ e $\mathbf{1}(\mathbf{a}) = 1$ para todo $\mathbf{a} \in A^n$, o conjunto $(A(n), +, \cdot, \bar{}, \mathbf{0}, \mathbf{1})$ também é uma álgebra booleana.

Note que nem todas as funções $f : A^n \to A$ podem ser definidas por uma expressão booleana; funções booleanas são aquelas que podem ser definidas por uma expressão booleana.

Exemplo: A função $f : B^2 \to B$, definida pela expressão $f(x_1, x_2) = x_1 + x_2$ pode ser representada pela tabela-verdade a seguir, à esquerda. Note que ela é igual a tabela-verdade da expressão $x_1 + \overline{x}_1 x_2$, à direita. Logo, as expressões $x_1 + x_2$ e $x_1 + \overline{x}_1 x_2$ são equivalentes (ou seja, definem uma mesma função).

x_1	x_2	$x_1 + x_2$
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	$\overline{x}_1$	$\overline{x}_1\, x_2$	$x_1 + \overline{x}_1\, x_2$
0	0	1	0	0
0	1	1	1	1
1	0	0	0	1
1	1	0	0	1

Há $2^{(2^2)} = 16$ funções de 2 variáveis para $\langle B, +, \cdot, \bar{}, 0, 1\rangle$, conforme mostrados a seguir:

x_1	x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

3.3 Formas canônicas

3.3.1 Soma de produtos

Produto: Um produto é uma expressão booleana que é ou uma literal, ou uma conjunção de duas ou mais literais, duas das quais nunca envolvem a mesma variável. Em outras palavras, um produto é uma conjunção em que uma variável aparece no máximo uma vez (na forma barrada ou não barrada). Produtos podem ser expressos como $p = \prod_{i=1}^{n} \sigma_i$, $\sigma_i \in \{x_i, \overline{x}_i, \text{' '}\}$, com ' ' denotando o caractere vazio. Por exemplo, para $n = 4$, $x_1 x_3$ e $x_2 \overline{x}_3 \overline{x}_4$ são exemplos de produtos.

Mintermos: Mintermo (ou **produto canônico**) em n variáveis $x_1, x_2, \ldots, x_n$ é uma expressão booleana formada pelo produto de cada uma das n variáveis ou dos respectivos complementos (mas não ambas). Ou seja, consiste do produto de n literais, cada um correspondendo a uma variável (se x_i está presente no produto, então $\overline{x}_i$ não está, e vice-versa).

Exemplo: Supondo 3 variáveis x_1, x_2, x_3, $x_1 \overline{x}_2 x_3$ e $x_1 x_2 x_3$ são exemplos de mintermos.

Notação: Denote x por x^1 e $\overline{x}$ por x^0. Assim, qualquer mintermo pode ser expresso por $m_{e_1,e_2,\ldots,e_n} = x_1^{e_1}\, x_2^{e_2}\, \ldots\, x_n^{e_n}$ onde $e_i \in \{0,1\}$. Por exemplo, se considerarmos $n = 3$, então $m_{001} = x_1^0\, x_2^0\, x_3^1 = \overline{x}_1\, \overline{x}_2\, x_3$.

O conjunto de todas as seqüências de n bits corresponde à representação binária dos números entre 0 e $2^n - 1$. Com base neste fato, podemos caracterizar os mintermos $m_{e_1,e_2,\ldots,e_n}$ através da correspondente notação decimal $m_{\sum 2^i\, e_i}$. A tabela 3.1 apresenta todos os mintermos em três variáveis e a notação associada a cada um deles.

$e_1\, e_2\, e_3$	$m_{e_1,e_2,\ldots,e_n}$
0 0 0	$\overline{x}_1\overline{x}_2\overline{x}_3 = m_0$
0 0 1	$\overline{x}_1\overline{x}_2 x_3 = m_1$
0 1 0	$\overline{x}_1 x_2\overline{x}_3 = m_2$
0 1 1	$\overline{x}_1 x_2 x_3 = m_3$
1 0 0	$x_1\overline{x}_2\overline{x}_3 = m_4$
1 0 1	$x_1\overline{x}_2 x_3 = m_5$
1 1 0	$x_1 x_2\overline{x}_3 = m_6$
1 1 1	$x_1 x_2 x_3 = m_7$

Tabela 3.1: Tabela de mintermos com 3 variáveis.

Teorema: Há 2^n mintermos em n variáveis e não há dois mintermos equivalentes.

PROVA: Como um mintermo consiste de n literais, cada um podendo ser uma variável x ou o seu complemento $\overline{x}$, há no total 2^n possíveis formas de se combinar as literais.

Para mostrar que não há dois mintermos equivalentes, seja $m_{e_1,e_2,\ldots,e_n} = x_1^{e_1}\, x_2^{e_2}\, \ldots\, x_n^{e_n}$ um mintermo e considere

$$x_i = \begin{cases} 1, & \text{se } e_i = 1, \\ 0, & \text{se } e_i = 0. \end{cases}$$

Então, $m_{e_1,e_2,\ldots,e_n}(x_1, x_2, \ldots, x_n) = 1$, pois todos os literais em m tem valor 1.

Qualquer outro mintermo m' tem pelo menos um literal x^{e_j} que é complemento do correspondente literal em m. Portanto, substituindo os valores de x_i definidos acima neste mintermo, haverá pelo menos um zero no produto. Isto quer dizer que este mintermo vale zero para estes valores em particular. Portanto, para quaisquer dois mintermos, há sempre uma atribuição de valores às variáveis que torna um deles 1 e o outro 0. □

Soma de produtos: Dizemos que uma expressão está na forma **soma de produtos** (SOP) se ela é um produto ou se é uma disjunção de dois ou mais produtos e se nenhum par de produtos p e p' é tal que $p \leq p'$ (a relação $\leq$ é a definida pela equivalência 3.1).

As expressões xy, $x + yz$ e $xyw + \overline{x}z + yz$ estão na forma SOP, enquanto que $x(y + z)$ e $xy + yzx$ não estão.

Os átomos de $\langle A(n), +, \cdot, \bar{}, \mathbf{0}, \mathbf{1}\rangle$ são os 2^n mintermos (novamente, considerando a relação $\leq$ definida pela equivalência 3.1). Portanto, toda função booleana pode ser escrita como uma disjunção (soma)

de mintermos distintos. Mais ainda, tal representação é única a menos da ordem dos mintermos, conforme teorema visto na parte de ordens parciais. Este mesmo resultado pode ser mostrado de uma outra forma. Veja a seguir.

Soma canônica de produtos (SOP canônica): Dizemos que uma expressão está na forma **soma canônica de produtos** (SOP canônica) se ela é um mintermo ou se é uma disjunção de dois ou mais mintermos distintos.

Teorema: Qualquer função booleana que não seja identicamente 0 (nulo) pode ser expressa unicamente na forma **soma canônica de produtos** (soma de mintermos ou SOP canônica). Mais precisamente, se f é uma função booleana em n variáveis então sua forma SOP canônica é dada por

$$f(x_1, x_2, \ldots, x_n) = \bigvee_{\mathbf{e}\in\{0,1\}^n} f(e_1, e_2, \ldots, e_n)\, x_1^{e_1}\, x_2^{e_2}\, \ldots\, x_n^{e_n}$$

PROVA: Uma demonstração pode ser encontrada em [Garnier and Taylor, 1992], página 408. Essa demonstração será discutida em sala de aula, mas nãos será transcrita aqui. □

Exemplos:

a) Expressar a função $f(x_1, x_2) = x_1 + x_2$ na forma SOP canônica.

De acordo com o teorema acima, f pode ser escrito como

$$f(x_1, x_2) = f(0,0)\overline{x}_1\,\overline{x}_2 + f(0,1)\overline{x}_1\,x_2 + f(1,0)x_1\,\overline{x}_2 + f(1,1)x_1\,x_2$$

Se calculamos o valor de f para todos os elementos $\mathbf{e} \in \{0,1\}^2$ temos $f(0,0) = 0$ e $f(0,1) = f(1,0) = f(1,1) = 1$. Portanto,

$$\begin{aligned} f(x_1, x_2) &= 0 \cdot \overline{x}_1\,\overline{x}_2 + 1 \cdot \overline{x}_1\,x_2 + 1 \cdot x_1\,\overline{x}_2 + 1 \cdot x_1\,x_2 \\ &= \overline{x}_1\,x_2 + x_1\,\overline{x}_2 + x_1\,x_2 \end{aligned}$$

b) Expressar $f(x, y, z, w) = (xz+y)(zw+\overline{w})$ na forma SOP. Neste caso, basta aplicarmos a distributiva para eliminar os parênteses.

$$\begin{aligned} f(x, y, z, w) &= (xz + y)(zw + \overline{w}) \\ &= (xz + y)zw + (xz + y)\overline{w} \quad (\textit{distributiva}) \\ &= xzw + yzw + xz\overline{w} + y\overline{w} \quad (\textit{distributiva}) \end{aligned}$$

c) Expressar $f(x, y, z) = [(x + \overline{y}) + z](x + \overline{\overline{y}})\overline{x}$ na forma SOP. Idem anterior.

$$\begin{aligned} f(x, y, z) &= [(x + \overline{y}) + z](x + \overline{\overline{y}})\overline{x} \\ &= [(x + \overline{y}) + z](x + y)\overline{x} \\ &= [(x + \overline{y}) + z](x\overline{x} + y\overline{x}) \\ &= [(x + \overline{y}) + z]y\overline{x} \\ &= xy\overline{x} + \overline{y}y\overline{x} + zy\overline{x} \\ &= 0 + 0 + zy\overline{x} \\ &= \overline{x}yz \end{aligned}$$

d) Escrever $f(x,y,z,w) = (xz+y)(zw+\overline{w})$ na forma SOP canônica. Aqui poderíamos utilizar uma abordagem similar ao do exemplo (a). É possível, no entanto, utilizarmos manipulações algébricas (eliminar os parênteses e em seguida "introduzir", em cada produto, as variáveis que não aparecem).

$$\begin{aligned} f(x,y,z,w) &= (xz+y)zw + (xz+y)\overline{w} \\ &= xzw + yzw + xz\overline{w} + y\overline{w} \\ &= xzw(y+\overline{y}) + (x+\overline{x})yzw + x(y+\overline{y})z\overline{w} + (x+\overline{x})y(z+\overline{z})\overline{w} \\ &= xyzw + x\overline{y}zw + xyzw + \overline{x}yzw + xyz\overline{w} + x\overline{y}z\overline{w} + xy(z+\overline{z})\overline{w} + \overline{x}y(z+\overline{z})\overline{w} \\ &= xyzw + x\overline{y}zw + \overline{x}yzw + xyz\overline{w} + x\overline{y}z\overline{w} + xyz\overline{w} + xy\overline{z}\overline{w} + \overline{x}yz\overline{w} + \overline{x}y\overline{z}\,\overline{w} \\ &= xyzw + xyz\overline{w} + xy\overline{z}\,\overline{w} + x\overline{y}zw + x\overline{y}z\overline{w} + \overline{x}yzw + \overline{x}yz\overline{w} + \overline{x}y\overline{z}\,\overline{w} \end{aligned}$$

Observações: Em vez de **produto**, alguns autores utilizam também os nomes **termo produto**, **produto fundamental**, **conjunção fundamental** ou **produto normal**.

Em vez de **soma de produtos**, utilizam-se também os nomes **soma de produtos normais** e **forma normal disjuntiva**.

Em vez de **soma canônica de produtos** (SOP canônica), utilizam-se também os nomes **soma padrão de produtos**, **forma normal disjuntiva completa** ou **forma mintermo**. Note, porém, que alguns autores usam o nome **forma normal disjuntiva** em vez de **forma normal disjuntiva completa**.

Nós usaremos **soma de produtos** e **soma canônica de produtos**.

3.3.2 Produto de somas

Todos os conceitos definidos com respeito a expressões do tipo produto podem também ser definidos com respeito a expressões do tipo soma.

Uma **soma** define-se de forma análoga ao produto: soma é ou um literal ou a disjunção de dois ou mais literais, duas das quais nunca envolvem a mesma variável. Dizemos que uma expressão booleana está na forma **produto de somas** (POS) se ela é uma soma ou é uma conjunção de duas ou mais somas.

Maxtermo (ou **soma canônica**) em n variáveis $x_1, x_2, \ldots, x_n$ tem definição similar ao mintermo: em vez de produto, consiste de soma de n literais, cada um correspondendo a uma variável. As expressões $\overline{x}_1 + \overline{x}_2 + \overline{x}_3$ e $\overline{x}_1 + x_2 + x_3$ são exemplos de maxtermos. A tabela 3.2 lista todos os matexrmos de 3 variáveis.

$e_1e_2e_3$	maxtermos
0 0 0	$x_1 + x_2 + x_3 = M_0$
0 0 1	$x_1 + x_2 + \overline{x}_3 = M_1$
0 1 0	$x_1 + \overline{x}_2 + x_3 = M_2$
0 1 1	$x_1 + \overline{x}_2 + \overline{x}_3 = M_3$
1 0 0	$\overline{x}_1 + x_2 + x_3 = M_4$
1 0 1	$\overline{x}_1 + x_2 + \overline{x}_3 = M_5$
1 1 0	$\overline{x}_1 + \overline{x}_2 + x_3 = M_6$
1 1 1	$\overline{x}_1 + \overline{x}_2 + \overline{x}_3 = M_7$

Tabela 3.2: Tabela de maxtermos com 3 variáveis.

Teorema: Há 2^n maxtermos e não há dois maxtermos equivalentes.

Produto canônico de somas (POS canônica): Dizemos que uma expressão booleana está na forma **produto canônico de somas** (POS canônica) se ela é um maxtermo ou é uma conjunção de dois ou mais maxtermos distintos.

Teorema: Qualquer função booleana que não seja identicamente 1 pode ser expressa unicamente na forma **produto canônico de somas** (produto de maxtermos ou POS canônica).

Exemplo: Escrever $x + z + \overline{y}\,\overline{w}$ na forma POS canônica.

$$\begin{aligned} f(x,y,z,w) &= x + z + \overline{y}\,\overline{w} \\ &= x + (z + \overline{y}\,\overline{w}) \\ &= x + (z + \overline{y})(z + \overline{w}) \\ &= (x + z + \overline{y})(x + z + \overline{w}) \\ &= (x + \overline{y} + z + w\overline{w})(x + y\overline{y} + z + \overline{w}) \\ &= (x + \overline{y} + z + w)(x + \overline{y} + z + \overline{w})(x + y + z + \overline{w})(x + \overline{y} + z + \overline{w}) \\ &= (x + \overline{y} + z + w)(x + \overline{y} + z + \overline{w})(x + y + z + \overline{w}) \end{aligned}$$

3.4 Representação de funções booleanas

A mais elementar das álgebras booleanas é aquela correspondente ao conjunto $B = \{0, 1\}$. Neste curso, a álgebra booleana de maior interesse é a álgebra booleana das funções booleanas associadas a B, ou seja, as funções do tipo $f : B^n \to B$.

Daqui em diante consideremos a álgebra booleana $\langle B(n), +, \cdot, \bar{}, \mathbf{0}, \mathbf{1}\rangle$. No caso da álgebra booleana $\langle B(n), +, \cdot, \bar{}, \mathbf{0}, \mathbf{1}\rangle$, há exatamente $(2)^{2^n}$ elementos em $B(n)$, ou seja, todas as funções de B^n em B podem ser definidas por expressões booleanas.

Algumas formas para descrever/representar funções booleanas são:

- expressões booleanas
- tabelas-verdade
- diagramas de decisão binária
- circuitos

3.4.1 Expressões booleanas

Além das formas soma de produtos e produto de somas (canônica ou não), expressões booleanas podem ter uma forma seqüencial ou mixta. Entre elas, merece destaque a decomposição de Shannon.

O teorema de **Expansão de Shannon** afirma que qualquer função f de n variáveis pode ser escrita em termos de funções de $n - 1$ variáveis da seguinte forma:

$$f(x_1, \ldots, x_k, \ldots, x_n) = \overline{x}_i f(x_1, \ldots, 0, \ldots, x_n) + x_i f(x_1, \ldots, 1, \ldots, x_n)$$

As funções $f(x_1, \ldots, 0, \ldots, x_n)$ e $f(x_1, \ldots, 1, \ldots, x_n)$ são funções de $n-1$ variáveis (não dependem da variável x_i).

3.4.2 Tabelas-verdade

Esta é a forma mais trivial e direta possível para se representar uma função booleana. Uma função de n variáveis pode ser representada por um vetor de 2^n posições, tal que a posição i do vetor armazena o valor da função para a entrada correspondente ao valor decimal i. No entanto, para um valor grande de n, esta representação torna-se impraticável.

Uma função booleana $f : \{0,1\}^n \to \{0,1\}$ pode ser completamente caracterizada em termos de seu conjunto-um, $f\langle 1\rangle = \{\mathbf{b} \in \{0,1\}^n : f(\mathbf{b}) = 1\}$, bem como em termos do seu conjunto-zero, $f\langle 0\rangle = \{\mathbf{b} \in \{0,1\}^n : f(\mathbf{b}) = 0\}$. Portanto, alternativamente à tabela-verdade, pode-se representar uma função através do seu conjunto-um ou do conjunto-zero. Nestes casos, encontrar uma representação eficiente desses conjuntos é a questão principal.

Observe que $x_1\overline{x}_2x_3 = 1$ se e somente se $x_1 = 1, x_2 = 0$ e $x_3 = 1$. Portanto, a SOP canônica de uma expressão booleana pode ser diretamente obtida através da soma dos mintermos correspondentes aos 1's da sua tabela-verdade. Analogamente, $(x_1 + \overline{x}_2 + x_3) = 0$ se e somente se $x_1 = 0, x_2 = 1$ e $x_3 = 0$, e portanto, a POS canônica pode ser obtida pelo produto dos maxtermos correspondentes aos 0's da tabela-verdade.

Exemplo: Dada a tabela-verdade

$x_1x_2x_3$	f_1
000	1
001	1
010	0
011	0
100	1
101	1
110	1
111	0

a forma SOP canônica da função é

$$f(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2x_3 + x_1\overline{x}_2x_3$$

e sua notação simplificada é dada por :

$$f(x_1, x_2, x_3) = m_0 + m_1 + m_4 + m_5 + m_6.$$

De forma mais compacta, é usual escrevermos

$$\sum m(0, 1, 4, 5, 6)$$

A forma POS canônica é $f(x_1, x_2, x_3) = (x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3) = \prod M(2, 3, 7)$.

3.4.3 Diagramas de decisão binária

Diagramas de decisão binária (ou BDD, do inglês *Binary Decision Diagram*) são grafos orientados utilizados para representar funções booleanas. Para mais detalhes, consulte [Akers, 1978, Bryant, 1986, Brace et al., 1990].

Eles estão diretamente relacionados com a expansão de Shannon descrita acima.

3.4.4 Circuitos (hardware)

Utilizando-se componentes lógicos (portas, inversores, etc) pode-se criar um circuito que realiza uma determinada função booleana. Conforme já vimos, dado um circuito lógico, pode-se determinar a correspondente expressão booleana. Analogamente, dada uma expressão booleana, pode-se construir o circuito correspondente. Além disso, da mesma forma que existem várias expressões booleanas que definem uma mesma função, existem vários circuitos que realizam uma mesma função.

No contexto de circuitos, uma das questões mais importantes é saber determinar o melhor circuito (em termos de custo, eficiência, etc) que realiza uma dada função. Existem alguns critérios de simplificação de funções booleanas que visam responder essa questão. A simplificação mais estudada é conhecida como **minimização lógica dois-níveis** e consiste em encontrar uma menor expressão minimal na forma SOP (o número de produtos corresponde ao número de portas E, o número de literais em um produto corresponde ao número de entradas da respectiva porta E). O termo dois-níveis está associado ao número máximo de portas que um sinal de entrada deve percorrer até a saída. Claramente, de uma forma grosseira, quanto menor o número de níveis de um circuito, mais eficiente ele será.

Por outro lado, uma realização dois níveis pode utilizar muitas portas lógicas (e pode haver limitações físicas quanto ao número de entradas de uma porta lógica, por exemplo). Assim, estudam-se também realizações multi-níveis de funções booleanas. Neste caso, há grande interesse em se reduzir o número de portas lógicas necessárias. Este problema é, em geral, conhecido por **decomposição funcional**.

Esses tópicos, relacionados a circuitos, serão vistos em detalhes mais adiante.

Exercícios:

1. Liste todos os mintermos em 3 variáveis.
2. Escreva $f(a,b,c,d,e) = (\overline{ac} + \overline{d})(\overline{b + ce})$ na forma SOP.
3. Escreva $f(a,b,c,d) = (a+b)c\overline{d} + (a+b)\overline{c}d$ na forma SOP canônica.
4. Escreva $f(x,y,z,w) = x + z + \overline{yw}$ na forma SOP canônica. Existe relação entre a forma SOP canônica e a forma POS canônica? Qual?
5. Lembrando que $x \leftrightarrow y \Leftrightarrow [(x \rightarrow y) \wedge (y \rightarrow x)]$, e que $x \rightarrow y \Leftrightarrow (\neg x \vee y)$, escreva $(a \vee b) \leftrightarrow \neg c$ na forma SOP canônica.
6. Ache a expressão na forma SOP canônica que define a função dada pela tabela-verdade abaixo. Você consegue simplificar esta expressão e obter uma outra equivalente e mais curta ?

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

7. Prove que qualquer função booleana que não seja identicamente 0 (nulo) pode ser expressa na forma SOP canônica.

8. Prove que a forma SOP canônica de qualquer função booleana que não seja identicamente 0 (nulo) é única, a menos da ordem dos produtos canônicos.

Capítulo 4

Lógica combinacional dois-níveis

Computadores digitais processam dados em formato binário. Esses processamentos podem ser encarados como mapeamentos do tipo $f : \{0,1\}^n \rightarrow \{0,1\}^m$. Vimos que qualquer mapeamento $f : \{0,1\}^n \rightarrow \{0,1\}$ pode ser expresso como soma de produtos canônicos (soma de mintermos) ou como produto de somas canônicas (produto de maxtermos). As expressões do tipo soma e produto podem ser realizadas em circuito pelas portas OU e E, respectivamente. Então, em príncípio, qualquer mapeamento $f : \{0,1\}^n \rightarrow \{0,1\}^m$ pode ser realizado por circuitos que utilizam apenas os inversores e as portas E e OU.

Deste capítulo em diante trataremos apenas de funções booleanas do tipo $f : \{0,1\}^n \rightarrow \{0,1\}$. Essas funções são, por alguns autores, chamadas de **funções de chaveamento**.

4.1 Lógica combinacional e seqüencial

Circuitos lógicos são classificados em dois tipos: combinacionais e seqüenciais. Os **circuitos combinacionais** são aqueles nos quais as saídas são determinadas em função apenas das entradas atuais. Os **circuitos seqüenciais** são aqueles nos quais as saídas dependem não apenas das entradas atuais mas também de dados prévios nos instantes anteriores. Pode-se dizer que circuitos seqüenciais envolvem realimentação, ou seja, eles possuem "memória".

O número de níveis de um circuito é definido como o número máximo de portas lógicas que um sinal de entrada deve atravessar para chegar até a saída. No caso de expressões do tipo soma de produtos, uma realização direta em circuito consiste de um conjunto de portas E (que são alimentados pelos sinais de entrada, invertidos ou não) no primeiro nível e, no segundo nível, uma porta OU (que recebe como entradas as saídas das portas E do primeiro nível). A saída da porta OU no segundo nível é o valor da função. Assim, um circuito desses é um circuito dois-níveis.

Lógica combinacional dois-níveis relaciona-se com o estudo de expressões que culminem em uma realização por circuito combinacional dois-níveis. Em particular, um problema muito estudado no contexto de circuitos lógicos é o problema de minimização lógica dois níveis, ou seja, o de encontrar uma menor expressão na forma soma de produtos que seja equivalente a uma função $f : \{0,1\}^n \rightarrow \{0,1\}$. Do ponto de vista de circuito, o número de níveis igual a dois implica que o circuito é eficiente em termos de tempo de processamento e a minimização do número de produtos na expressão implica minimização do número de portas lógicas (e, portanto, do tamanho do circuito e seu custo).

4.2 Minimização lógica dois-níveis

Ao falarmos em minimização lógica dois-níveis, estamos pensando em expressões na forma soma de produtos. Inicialmente definiremos o que é uma expressão minimal.

Definição: Uma expressão booleana escrita como soma de produtos é **minimal** se (1) não existe nenhuma outra expressão equivalente 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.

Dada uma expressão minimal na forma soma de produtos, ao se remover um produto ou um literal de qualquer um dos produtos, a expressão resultante não mais representa a mesma função.

Dada uma função qualquer, como pode ser calculada uma expressão minimal dessa função na forma soma de produtos? Antes de prosseguirmos em direção à resposta, introduzimos alguns termos e conceitos.

4.2.1 Produtos, cubos e intervalos

Produtos como expressão:

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

Considere três variáveis a, b e c e os mintermos abc e $\overline{a}bc$. Sabemos que $abc + \overline{a}bc = (a + \overline{a})bc = bc$. O termo bc é também um produto, porém ele não é canônico. O produto bc toma valor 1 para os elementos 011 e 111 de $\{0,1\}^3$. Em outras palavras, o valor da variável a não afeta o valor desse produto.

Produtos como subconjuntos de $\{0,1\}^n$ ou sub-cubos:

Lembramos que na função $f(a,b,c) = abc + \overline{a}bc$, 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 $\{0,1\}^3$
$\overline{a}bc$	$\{011\}$
abc	$\{111\}$
$\overline{a}bc + abc$	$\{011, 111\}$

Os elementos de $\{0,1\}^n$ (aqui denotados como seqüências ou strings de n números binários) podem ser encarados como pontos no n-espaço. A coleção de todos os 2^n pontos desse espaço forma os vértices de um hipercubo. A figura 4.1 mostra um hipercubo no espaço de dimensão 3.

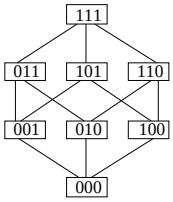


Figura 4.1: Um hipercubo de dimensão 3.

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.

Produtos como intervalos do poset $(\{0,1\}^n, \leq)$:

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 $p\langle 1\rangle = \{\mathbf{b} \in \{0,1\}^n : p(\mathbf{b}) = 1\}$.

Cubos, no contexto de reticulados, são sub-reticulados do reticulado $\{0,1\}^n$, denominados **intervalos**. Um intervalo num poset é caracterizado por dois extremos: o menor e o maior elementos contidos nele. Assim, no poset $(\{0,1\}^3, \leq)$, o intervalo de extremo inferior 100 e extremo superior 101 é denotado $[100, 101]$ e definido por $[100, 101] = \{\mathbf{x} \in \{0,1\}^3 : 100 \leq \mathbf{x} \leq 101\}$.

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 correponde ao produto $\overline{b}\,\overline{c}$, é representado por $X00$. O 2-cubo $\{000, 001, 100, 101\}$ (ou, equivalentemente, o intervalo $[000, 101]$), que corresponde ao produto $\overline{b}$, 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 4.2 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 $X00$.

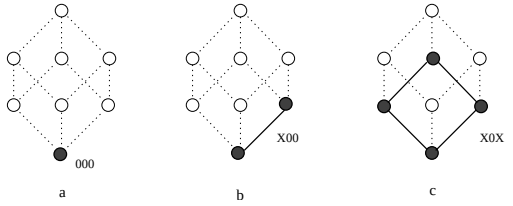


Figura 4.2: Os cubos 000, $X00$ e $X0X$.

Resumo:

Produtos, cubos e intervalos referem-se a mesma coisa. A tabela a seguir resume esses conceitos:

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	$2^2 = 4$

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

Uma função booleana com poucas variáveis (tipicamente 3 ou 4) pode ser graficamante ilustrada através do diagrama de Hasse. Usaremos a convensão de desenhar elementos de $f\langle 1\rangle = \{\mathbf{b} \in \{0,1\}^n : f(\mathbf{b}) = 1\}$ como círculos preenchidos, enquanto os elementos de $f\langle 0\rangle = \{\mathbf{b} \in \{0,1\}^n : f(\mathbf{b}) = 0\}$ serão representados por círculos não preenchidos. A representação via diagrama de Hasse da função $f_1(x_1, x_2, x_3) = \overline{x}_1\,\overline{x}_2\,\overline{x}_3 + \overline{x}_1\,\overline{x}_2\,x_3 + \overline{x}_1\,x_2\,x_3 + x_1\,x_2\,x_3$ é mostrada na figura 4.3.

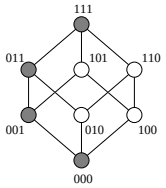


Figura 4.3: Representação via diagrama de Hasse da função $f_1(x_1, x_2, x_3) = \overline{x}_1\,\overline{x}_2\,\overline{x}_3 + \overline{x}_1\,\overline{x}_2\,x_3 + \overline{x}_1\,x_2\,x_3 + x_1\,x_2\,x_3$.

Sempre que um dos produtos de uma soma de produtos toma valor 1, a soma também toma valor 1. No caso da função $f(a, b, c) = abc + \overline{a}bc$, se $abc = 1$ então $f(a, b, c) = 1$. Isto pode ser expresso como $\overline{a}bc \leq f$. Analogamente temos $abc \leq f$ e $bc \leq f$.

Dada uma função f e um produto p, dizemos que o conjunto $p\langle 1\rangle$ **é um cubo de** f se $p \leq f$ (ou, equivalentemente, se $p\langle 1\rangle \subseteq f\langle 1\rangle$). Logo, $\overline{a}bc$ e abc, por exemplo, são cubos de $f(a, b, c) = abc + \overline{a}bc$.

Neste sentido, a forma SOP canônica de f pode ser vista como a coleção de todos os 0-cubos de f, aqueles que correspondem aos elementos em $f\langle 1\rangle$.

Os dois primeiros cubos da Fig. 4.4 (em negrito) são cubos da função f_1 mas os dois últimos não são.

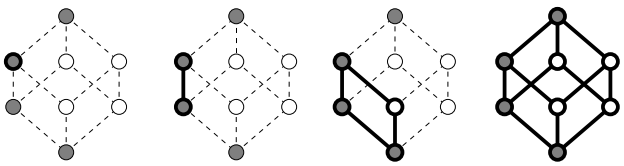


Figura 4.4: Exemplos de um 0-cubo (intervalo 011), um 1-cubo (intervalo $0X1$), um 2-cubo (intervalo $0XX$) e um 3-cubo (intervalo XXX), respectivamente (em negrito).

Definição: Um **implicante primo** (ou simplesmente **primo**) de uma função booleana f é um produto p tal que $p \leq f$, e não há outro produto p', $p < p'$, tal que $p' \leq f$.

Os implicantes primos são cubos ou intervalos maximais contidos em $f\langle 1\rangle$ (i.e., um cubo de f que não é totalmente contido em outro cubo de f). Por exemplo, na função $f(a,b,c) = abc + \overline{a}bc$, abc é um cubo de f, mas não é implicante primo de f pois $abc < bc \leq f$. Já bc é implicante primo de f.

Exemplo: A função Booleana $f = \sum m(0,1,4,5,6)$ é representada pelos vértices 000, 001, 100, 101 e 110. Os mintermos de f e os implicantes primos de f (cubos $X0X$ e $1X0$) são mostrados respectivamente nas figuras 4.5a e 4.5b.

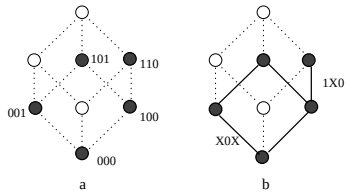


Figura 4.5: (a) Mintermos e (b) implicantes primos de $f = \sum m(0,1,4,5,6)$.

Voltemos agora à questão do início desta seção: dada uma função qualquer, qual é a expressão equivalente minimal na forma soma de produtos?

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

A prova deste teorema é simples. Suponha que exista algum produto p na expressão que não seja um implicante 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.

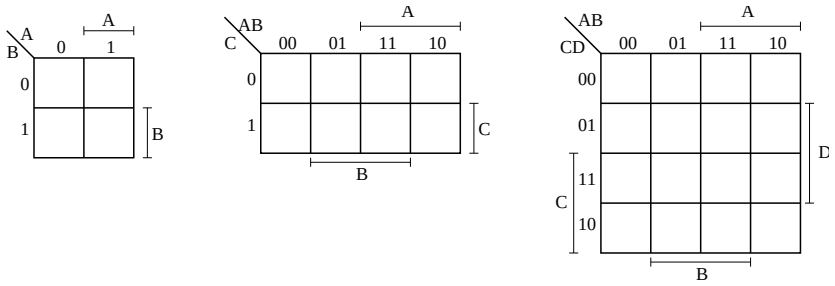
Este teorema diz, em outra palavras, que para encontrarmos uma expressão minimal de uma função, basta considerarmos apenas os implicantes primos da função.

Nas próximas seções serão apresentadas algumas técnicas utilizadas para o cálculo de uma expressão minimal na forma soma de produtos.

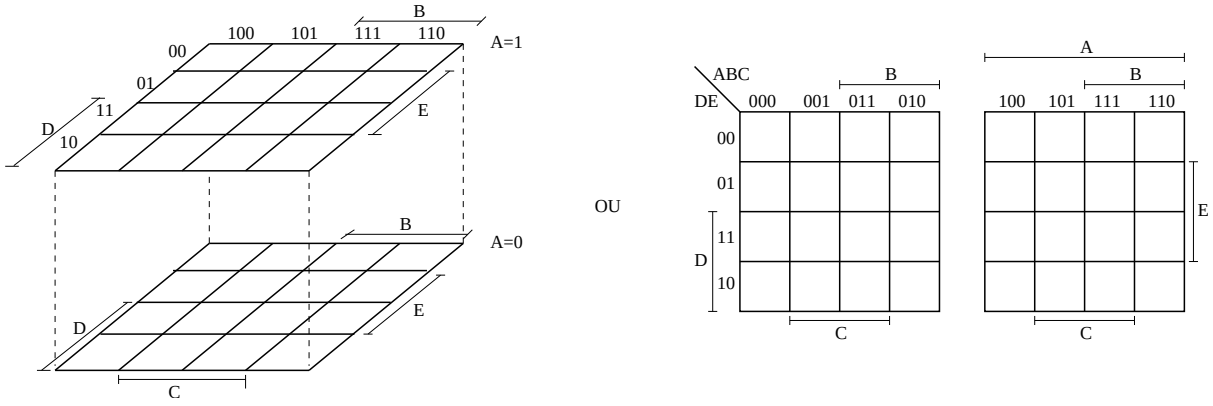
4.2.2 Mapas de Karnaugh

A minimização de uma expressão Booleana pode ser realizada algebricamente aplicando-se os axiomas e leis da álgebra Booleana. Entretanto, a manipulação algébrica, além de ser uma tarefa cansativa, pode facilmente induzir uma pessoa a cometer erros, principalmente quando o número de variáveis envolvidas é grande. Além disso, muitas vezes é difícil ter certeza de que o resultado obtido é minimal. Mapas de Karnaugh são diagramas que são utilizados para auxiliar este processo.

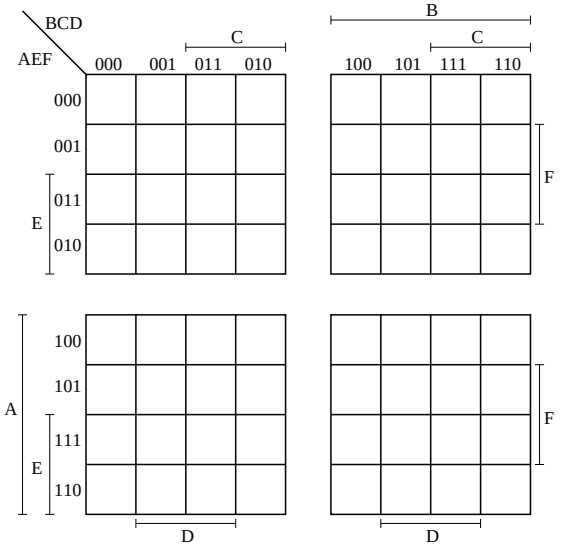
Mapa de Karnaugh de 2, 3 e 4 variáveis:



Mapa de Karnaugh de 5 variáveis:



Mapa de Karnaugh de 6 variáveis:



Cada célula dos mapas corresponde a um elemento de $\{0,1\}^n$. 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 4.6 mostra em cada célula o 0-cubo correspondente, enquanto o mapa à direita mostra em cada célula o valor decimal dos respectivos 0-cubos. Observe que o cabeçalho está disposto em uma seqüência não-usual. Por exemplo, para duas variáveis, a seqüência natural seria 00, 01, 10, 11. Porém, a seqüência utilizada é 00, 01, 11, 10, que possui a característica de dois elementos adjacentes (na seqüência) diferirem em apenas 1 bit.

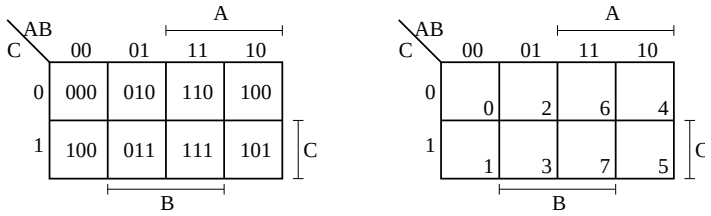


Figura 4.6: Os 0-cubos correspondentes a cada célula da mapa de Karnaugh de 3 variáveis em notação binária e decimal, respectivamente.

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

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

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
$\overline{x}_1\,\overline{x}_2\,\overline{x}_3$	000
$\overline{x}_1\,\overline{x}_2\,x_3$	001
$\overline{x}_1\,x_2\,\overline{x}_3$	010
$\overline{x}_1\,x_2\,x_3$	011
$x_1\,x_2\,x_3$	111

Em seguida, as células correspondentes a esses 0-cubos devem ser marcados com 1 no mapa, conforme mostrado no mapa da esquerda na figura 4.7. O processo consiste, então, em procurar, para cada 0-cubo da função, o maior cubo da função que o cobre. Isto, no mapa de Karnaugh, corresponde a juntar o 0-cubo em questão com 0-cubos adjacentes a ele de forma a sempre formar um retângulo (ou quadrado) com 2^k elementos ($k >= 1$). No exemplo da figura 4.7, o maior cubo que cobre 000 é o cubo 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 $\overline{x}_1$ e o correspondente a X11 é $x_2\,x_3$. Portanto, uma forma SOP minimal é $f(x_1, x_2, x_3) = \overline{x}_1 + x_2\,x_3$.

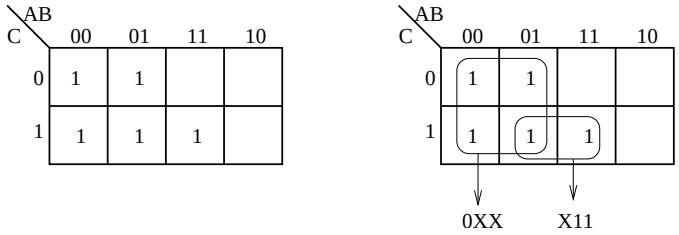


Figura 4.7: Exemplo do uso do mapa de Karnaugh: minimização da função $f = \sum m(0,1,2,3,7)$.

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 $f(a,b,c,d) = c + \overline{a}\,b\,d + \overline{b}\,\overline{d}$. Veja o mapa da figura 4.8.

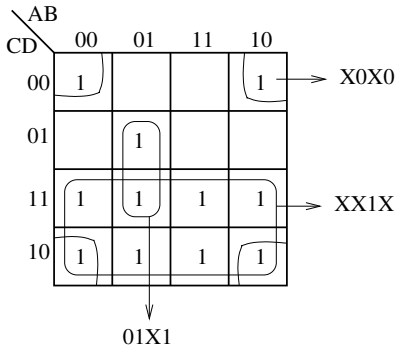


Figura 4.8: Minimização da função $f(a,b,c) = \sum m(0,2,3,5,6,7,8,10,11,14,15)$.

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 4.9 mostra todos os cubos maximais de f. As possíveis soluções são:

$f(a,b,c,d) = \overline{a}\,b + a\,\overline{b} + \overline{a}\,\overline{c}\,\overline{d} + b\,c$
$f(a,b,c,d) = \overline{a}\,b + a\,\overline{b} + \overline{a}\,\overline{c}\,\overline{d} + a\,c$
$f(a,b,c,d) = \overline{a}\,b + a\,\overline{b} + \overline{b}\,\overline{c}\,\overline{d} + b\,c$
$f(a,b,c,d) = \overline{a}\,b + a\,\overline{b} + \overline{b}\,\overline{c}\,\overline{d} + a\,c$

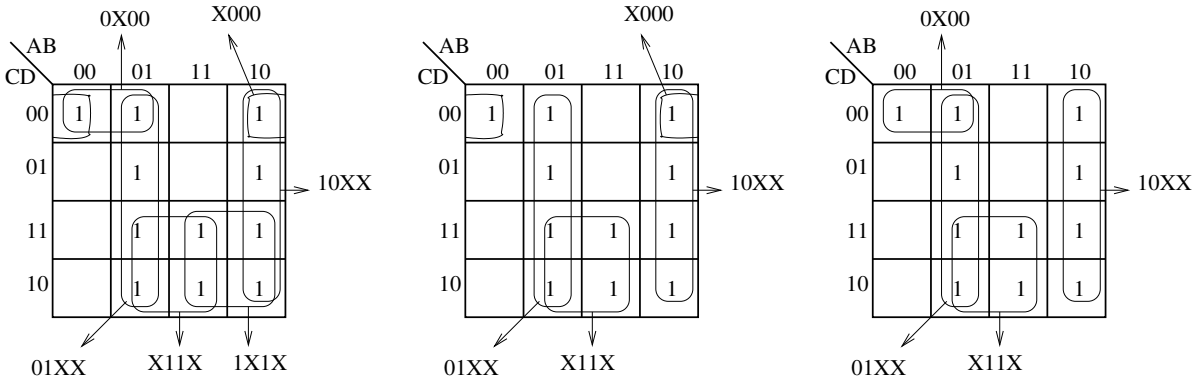


Figura 4.9: Minimização da função $f(a,b,c) = \sum m(0,4,5,6,7,8,9,10,11,14,15)$. Esquerda: todos os implicantes primos (ou cubos maximais). Centro: uma solução. Direita: outra solução.

Mapa de Karnaugh para encontrar a forma POS minimal: Será que o mapa de Karnaugh pode ser utilizado também para se 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 4.10. A forma SOP minimal é $f(a,b,c) = \overline{b}\,\overline{c} + a\,c$.

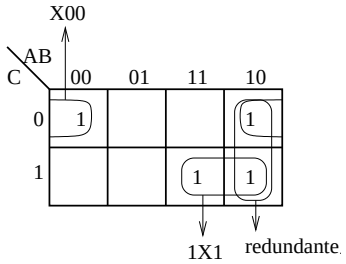


Figura 4.10: Exemplo do uso do mapa de Karnaugh: minimização da função $f(a,b,c) = \sum m(0,4,5,7)$.

A minimização SOP de $\overline{f}(a,b,c) = \sum m(1,2,3,6)$ por mapa de Karnaugh é mostrada na figura 4.11. O resultado obtido é $\overline{f}(a,b,c) = b\,\overline{c} + \overline{a}\,c$.

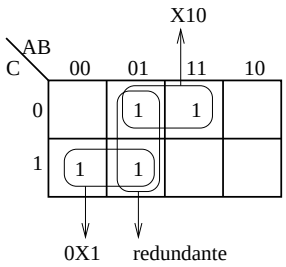


Figura 4.11: Minimização da função $\overline{f}(a,b,c) = \sum m(1,2,3,6)$.

Agora, observe que $f = \overline{\overline{f}}$. Portanto, posso escrever $f = \overline{b\,\overline{c} + \overline{a}\,c} = (\overline{b\,\overline{c}})(\overline{\overline{a}\,c}) = (\overline{b} + c)(a + \overline{c})$.

Tudo isto pode ser diretamente realizado no mapa de Karnaugh conforme mostrado na figura 4.12. Em vez de marcar os 0-cubos da função no mapa, marcamos os 0-cubos do complemento de f (ou,

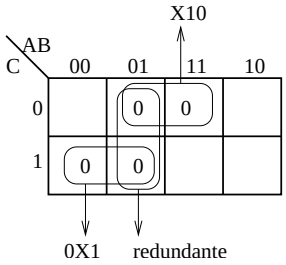


Figura 4.12: Minimização POS da função $f(a,b,c) = \prod M(1,2,3,6)$.

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 $a + \overline{c}$ e o cubo X10 ao termo soma $\overline{b} + c$. Assim, temos que a forma POS minimal é $f(a,b,c) = (a + \overline{c})(\overline{b} + c)$.

4.2.3 Minimização Tabular de Quine-McCluskey

Mapas de Karnaugh 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) 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 destes 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.

Este processo de combinar dois cubos pode ser facilmente associado ao processo algébrico de simplificação. Os mintermos da expressão na forma canônica inicial correspondem 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 :

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

Quando considerados no 3-espaço, 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 4.13.

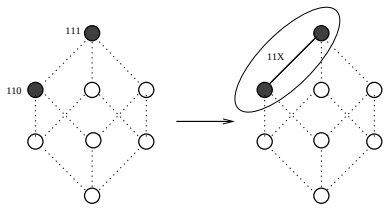


Figura 4.13: Passo elementar do algoritmo de Quine-McCluskey

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 4.14(a). Note, porém, que o intervalo $[001, 011]$ é redundante, ou seja, a mesma expressão pode ser expressa apenas pelos implicantes primos $00X$ e $X11$ (figura 4.14(b)).

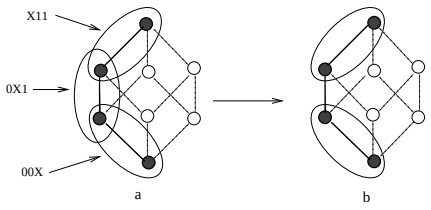


Figura 4.14: Os (a) implicantes primos e uma (b) cobertura mínima .

Conforme teorema apresentado algumas páginas atrás, sabemos que os produtos que aparecem na forma SOP minimal de uma função booleana são implicantes primos da função. Mas nem todos os implicantes primos aparecem na forma SOP minimal, como mostrado no exemplo acima. 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

O ponto central da segunda etapa é o cálculo de um menor subconjunto do conjunto de implicantes primos suficientes para cobrir[1] todos os mintermos da função Booleana. 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.

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 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

[1] Um conjunto de implicantes primos (cubos maximais) cobre um mintermo (0-cubo) se este é coberto por pelo menos um dos implicantes primos.

- 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 $\surd$ os cubos que foram usados para gerar novos cubos.

$\surd$	000		$00X$
$\surd$	001		$X00$
$\surd$	100	$\Longrightarrow$	$X01$
$\surd$	101		$10X$
$\surd$	110		$1X0$

  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.

$\surd$	000		$\surd$	$00X$		
$\surd$	001		$\surd$	$X00$		
$\surd$	100	$\Longrightarrow$	$\surd$	$X01$	$\Longrightarrow$	$X0X$
$\surd$	101		$\surd$	$10X$		
$\surd$	110			$1X0$		

- 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 $\surd$.

  $1X0$ e $X0X$

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 $\surd$ 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		$\surd$	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$					$\surd$	$\surd$
	X10X1					$\surd$		$\surd$						$\surd$		$\surd$
	0X0X1	$\surd$		$\surd$		$\surd$		$\surd$								
	00X01	$\surd$			$\surd$											
	X0101				$\surd$							$\surd$				
	1010X										$\surd$	$\surd$				
	10X11									$\surd$			$\surd$			
	101X1											$\surd$	$\surd$			

2. Selecionar os implicantes primos essenciais: deve-se procurar na tabela as colunas que contém apenas uma marca $\surd$. A linha na qual uma dessas colunas contém a marca $\surd$ 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		$\surd$	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$					$\surd$	$\surd$
	X10X1					$\surd$		$\surd$						$\surd$		$\surd$
	0X0X1	$\surd$		$\surd$		$\surd$		$\surd$								
	00X01	$\surd$			$\surd$											
	X0101				$\surd$							$\surd$				
	1010X										$\surd$	$\surd$				
	10X11									$\surd$			$\surd$			
	101X1											$\surd$	$\surd$			
			$\surd$	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$					$\surd$	$\surd$

   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		$\surd$	$\surd$			$\surd$	$\surd$	$\surd$	$\surd$					$\surd$	$\surd$
*	X10X1					$\surd$		$\surd$						$\surd$		$\surd$
	0X0X1	$\surd$		$\surd$		$\surd$		$\surd$								
	00X01	$\surd$			$\surd$											
	X0101				$\surd$							$\surd$				
*	1010X										$\surd$	$\surd$				
	10X11									$\surd$			$\surd$			
	101X1											$\surd$	$\surd$			
			$\surd$	$\surd$		$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$		$\surd$	$\surd$	$\surd$

3. Reduzir 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	$\surd$		
	00X01	$\surd$	$\surd$	
	X0101		$\surd$	
	10X11			$\surd$
	101X1			$\surd$

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.

   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 a dimensão do implicante da linha A é maior ou igual ao do implicante da linha B, então a linha B pode ser eliminada da tabela.

   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 restantes. Deve-se escolher dentre elas, uma que envolve o menor número de termos. Caso existam mais de uma nestas condições, deve-se escolher a de custo mínimo (aquela 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	$\surd$		
B	$\surd$	$\surd$	
C			$\surd$
D		$\surd$	$\surd$

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 $\surd$ em todas as linhas que uma outra coluna *alpha* tem $\surd$, 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 $\surd$ em todas as colunas em que a linha B tem $\surd$, então a linha A domina a linha B. Se elas tem $\surd$ 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 $f(a,b,c) = a\,\overline{b}\,c + \overline{a}\,b\,\overline{c} + a\,b\,\overline{c} + a\,b\,c = \sum m(2,5,6,7)$.

Podemos realizar a simplificação algébrica da seguinte forma:

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

Por QM temos

√	010
√	101
√	110
√	111

$\Longrightarrow$

$X10$
$1X1$
$11X$

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

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

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

4.2.4 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 implicantes primos, pois isto aumenta a chance de obter cubos maiores (portanto produtos com menos literais). Observe que, durante as iterações para a geração dos implicantes primos, um cubo que cobre apenas don't cares não pode 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 então caracterizar uma função booleana f através dos seus conjuntos um, zero e dc (de don't care), definidos respectivamente por $f\langle 1\rangle = \{\mathbf{b} \in \{0,1\}^n : f(\mathbf{b}) = 1\}$, $f\langle 0\rangle = \{\mathbf{b} \in \{0,1\}^n : f(\mathbf{b}) = 0\}$ 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 $f\langle 1\rangle$.

4.2.5 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 procede-se da seguinte forma. No cálculo dos implicantes primos, em vez de listar os mintermos, lista-se os 0s da função e aplica-se o método tabular. Realiza-se 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: aos tomarmos os 0s da função em vez dos 1s, estamos considerando a minimização SOP de $\overline{f}$. Agora, uma vez que $f = \overline{\overline{f}}$, ao complementarmos a forma SOP minimal de $\overline{f}$, 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:

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

Por QM temos:

√	011
√	110
√	111

$\Longrightarrow$

$X11$
$11X$

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

4.2.6 O algoritmo ISI

Podemos dizer que o algoritmo QM utiliza uma abordagem *bottom-up* para gerar todos os implicantes primos. Outra possível abordagem é a *top-down*, utilizada pelo ISI (*Inremental splitting of intervals*). O ISI inicia o processo a partir do n-cubo e, sucessivamente, elimina os zeros da função, tomando cuidado em representar os elementos que restam após uma eliminação através do conjunto de seus intervalos maximais. Assim, depois de eliminar todos os zeros, os elementos restantes correspondem aos uns (mintermos) da função, os quais estarão representados pelos intervalos maximais, ou seja, pelos implicantes primos da função.

Para fazer paralelo com algo mais concreto, podemos pensar que o QM é aquele processo em que o fulano constrói uma parede juntando os tijolos, enquanto o ISI é aquele em que o fulano esculpe uma estátua removendo partes de uma pedra (ou madeira).

Passo básico do ISI

Remover um elemento de um cubo (intervalo) e representar os elementos restantes pelos subintervalos maximais contidos neles é a chave do algoritmo ISI.

Sejam $[A, B]$ e $[C, D]$ dois intervalos em $\{0,1\}^n$. A diferença de $[A, B]$ e $[C, D]$ é o conjunto

$$[A, B] \setminus [C, D] = \{\mathbf{x} \in \{0,1\}^n : \mathbf{x} \in [A, B] \text{ e } \mathbf{x} \notin [C, D]\} \tag{4.1}$$

que pode também ser expresso por $[A, B] \setminus [C, D] = [A, B] \cap [C, D]^c$.

Proposição: Sejam $[A, B]$ e $[C, D]$ dois intervalos de $\{0,1\}^n$ cuja interseção é não-vazia. Então,

$$\begin{aligned}[A, B] \setminus [C, D] = \\ \Big\{[A, B'] : B' \text{ é elemento maximal daqueles que não contém nenhum elemento de } [C, D]\Big\} \cup \\ \Big\{[A', B] : A' \text{ é elemento minimal daqueles que não estão contidos em nenhum elemento de } [C, D]\Big\}.\end{aligned} \tag{4.2}$$

A equação acima mostra como a diferença de intervalos pode ser expressa em termos da coleção de intervalos maximais contidos na diferença.

Se $dim([A, B]) = k$, qualquer intervalo maximal contido na diferença tem dimensão $k - 1$. O número de intervalos maximais contidos na diferença é dado por $dim([A, B]) - dim([A, B] \cap [C, D])$.

Exemplo: Mostramos a seguir algumas diferenças representadas pelos intervalos maximais contidos na diferença.

Sejam $[A, B] = [000, 111]$ e $[C, D] = [001, 111]$. O número de intervalos em $[A, B] \setminus [C, D]$ é $dim([A, B]) - dim([A, B] \cap [C, D]) = 3 - 2 = 1$ e a dimensão dos intervalos resultantes é $dim([A, B]) - 1 = 3 - 1 = 2$. Veja Fig. 4.15.

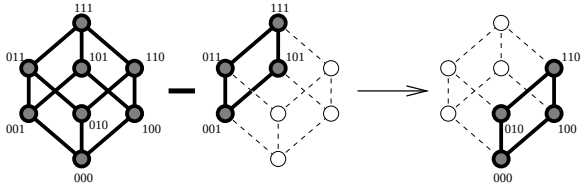


Figura 4.15: $[000, 111] \setminus [001, 111] = \{[000, 110]\}$.

Sejam $[A, B] = [000, 111]$ e $[C, D] = [001, 011]$. O número de intervalos em $[A, B] \setminus [C, D]$ é $dim([A, B]) - dim([A, B] \cap [C, D]) = 3 - 1 = 2$ e a dimensão dos intervalos resultantes é $dim([A, B]) - 1 = 3 - 1 = 2$. Veja Fig. 4.16.

Sejam $[A, B] = [000, 111]$ e $[C, D] = [011, 011]$. Este é o caso em que apenas um ponto do cubo é removido. O número de intervalos em $[A, B] \setminus [C, D]$ é $dim([A, B]) - dim([A, B] \cap [C, D]) = 3 - 0 = 3$ e a dimensão dos intervalos resultantes é $dim([A, B]) - 1 = 3 - 1 = 2$. Veja Fig. 4.17.

Um exemplo completo

Consideremos a minimização da função $f(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2x_3 + x_1x_2\overline{x}_3$. Temos $f\langle 0\rangle = \{111, 011, 010\}$. A figura 4.18 mostra os elementos que permanecem após as sucessivas remoções dos elementos em $f\langle 0\rangle$. Cada seta indica um passo de remoção (note que não mostramos os intervalos maximais resultantes individualmente, mostramos os elementos restantes em negrito). A figura 4.19 mostra o mesmo processo em uma estrutura de árvore. Cada nível da árvore corresponde ao resultado após um passo de remoção. Note que alguns intervalos podem ser descartados por estarem contidos em outros.

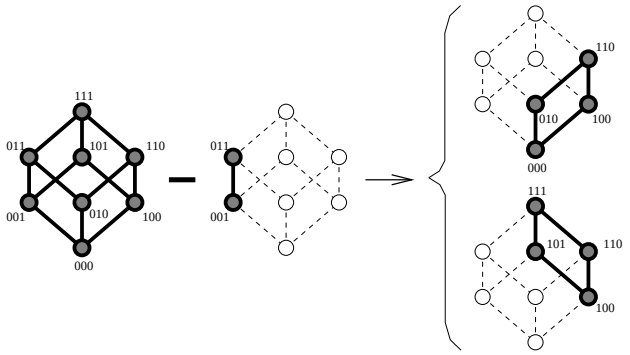


Figura 4.16: $[000, 111] \setminus [001, 011] = \{[000, 110], [100, 111]\}$.

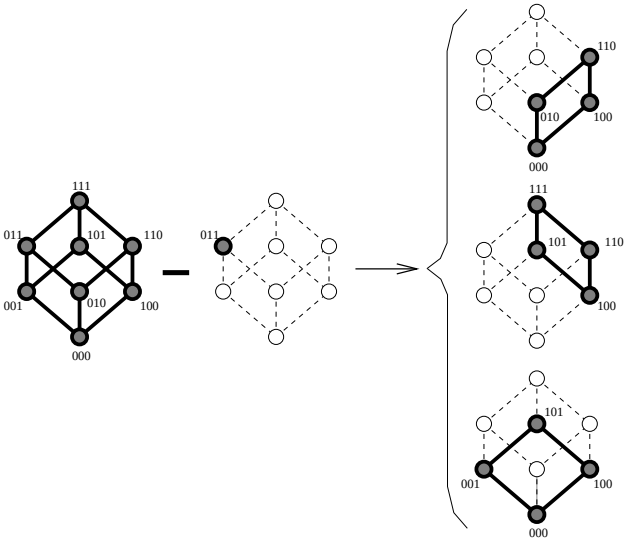


Figura 4.17: $[000, 111] \setminus [011, 011] = \{[000, 110], [100, 111], [000, 101]\}$.

ISI: Minimização de funções incompletamente especificadas

O algoritmo ISI remove sucessivamente os zeros do cubo inicial de forma a obter, ao final do processo, o conjunto de todos os intervalos maximais de $f\langle 1\rangle$ (i.e., implicantes primos). Se a função possui don't cares, então ao final do processo serão obtidos os intervalos maximais de $f\langle 1\rangle \cup f\langle *\rangle$. No entanto, intervalos que são formados inteiramente por elementos de $f\langle *\rangle$ não serão necessários na etapa de cálculo de uma cobertura mínima. Portanto, estes podem ser descartados, assim que são detectados durante o processo de remoção dos zeros.

Mostramos a idéia através da sua aplicação na minimização de f, com $f\langle 0\rangle = \{010, 111\}$, $f\langle 1\rangle = \{000, 001, 101\}$ e $f\langle *\rangle = \{100, 011, 110\}$, cujo processo é ilustrado nas figuras 4.20 and 4.21.

Os elementos que são don't cares são representados no diagrama de Hasse como vértices sem os círculos; círculos preenchidos representam elementos de $f\langle 1\rangle$ enquanto os não preenchidos representam os de $f\langle 0\rangle$. As arestas em negrito indicam que os elementos adjacentes a ela fazem parte de um intervalo.

No início do processo, todos os elementos fazem parte do intervalo XXX. Após a remoção de 111, três intervalos são gerados: $0XX$, $X0X$ and $XX0$. Nenhum deles pode ser descartado pois todos eles

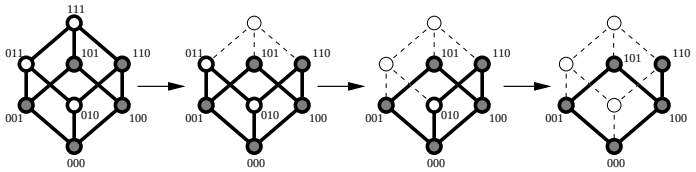


Figura 4.18: Cálculo de implicantes primos de f ($f\langle 0\rangle = \{010, 011, 111\}$ e $f\langle 1\rangle = \{000, 001, 100, 101, 110\}$). Ordem de remoção: 111, 011 e 010.

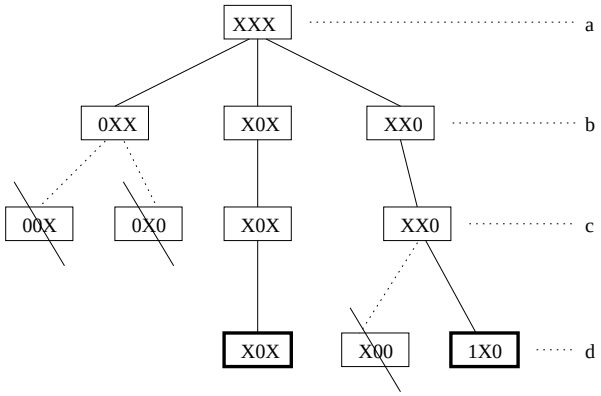


Figura 4.19: Cálculo de implicantes primos de f ($f\langle 0\rangle = \{010, 011, 111\}$ e $f\langle 1\rangle = \{000, 001, 100, 101, 110\}$). (a) intervalo inicial; (b) após remoção de 111 ; (c) após remoção de 011; (d) resultado final, após remoção de 010.

contém pelo menos um elemento de $f\langle 1\rangle$. Em seguida, o elemento 010 é removido dos intervalos $0XX$, and $XX0$, produzindo os intervalos $00X, 0X1$ and $X00, 1X0$, respectivamente. Os intervalos $00X$ and $X00$ podem ser descartados pois estão contidos em $X0X$. O intervalo $1X0$ pode ser descartado pois não cobre nenhum elemento de $f\langle 1\rangle$. Assim, os intervalos que restam são $X0X$ and $0X1$. O segundo será eliminado no processo de cálculo de cobertura mínima.

Note que a função resutante é consistente com a inicial; dos elementos em $f\langle *\rangle$, 011 e 110 são mapeados para 0, e somente 100 é mapeado para 1.

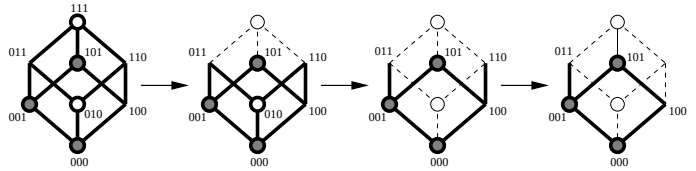


Figura 4.20: Cálculo de implicantes primos de f ($f\langle 0\rangle = \{010, 111\}$, $f\langle 1\rangle = \{000, 001, 101\}$ e $f\langle *\rangle = \{100, 011, 110\}$).

Exercícios

1. Minimize a função $f(a, b, c) = \sum m(1, 2, 3, 4, 5, 6)$.
2. Minimize a função $f(a, b, c, d) = \sum m(0, 2, 3, 6, 7, 8, 9, 10, 13)$.
3. Minimize a função $f(a, b, c, d,) = \sum m(0, 4, 5, 6, 7, 8, 9, 10, 11, 14, 15)$ por QM.

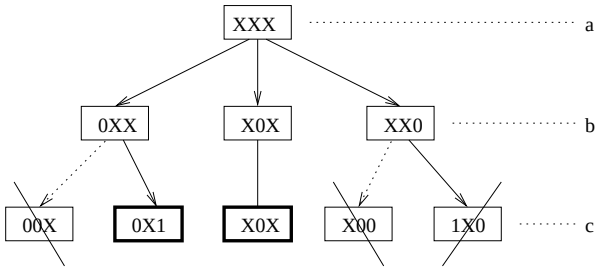


Figura 4.21: Cálculo de implicantes primos de f ($f\langle 0\rangle = \{010, 111\}$, $f\langle 1\rangle = \{000, 001, 101\}$ e $f\langle *\rangle = \{100, 011, 110\}$).

Confira os resultados: os implicantes primos são 01XX, 10XX, X11X, 1X1X, 0X00, X000 e os essenciais são 01XX, 10XX. Chega-se a uma tabela cíclica constituída dos implicantes 0X00 (1), X000 (2), X11X (3), 1X1X (4) e pelo métdo de Petrick conclui-se que as possíveis soluções são dadas por $(1+2)(3+4) = 1, 3+1, 4+2, 3+2, 4$. Ou seja, há quatro possíveis soluções de custo equivalente. Compare coma solução por mapa de Karnaugh (figura 4.9).

4. Minimizar $f(a, b, c, d) = \sum m(0, 2, 8, 12, 13) = \prod M(1, 3, 4, 5, 6, 7, 9, 10, 11, 14, 15)$.

   Resposta (conferir): $f = 00X0 + 110X + 1X00$ ou $f = 00X0 + 110X + X000$, $f = (\overline{a} + \overline{c})(a + \overline{b})(b + \overline{d})$.

5. O que se pode dizer sobre a forma SOP e POS minimal de uma função? Tem uma que sempre utiliza menor número de portas ou o número de portas nas ambas as formas podem ser iguais?

6. Minimizar $f(w, x, y, z) = \sum m(0, 7, 8, 10, 12) + d(2, 6, 11)$.

   Resposta (conferir): $\overline{x}\,\overline{z} + w\,\overline{y}\,\overline{z} + \overline{w}\,x\,y$ ($X0X0$, $1X00$ e $011X$)

7. Minimizar na forma POS a função $f(x, y, z) = \prod M(0, 1, 6, 7)$

8. Minimizar na forma SOP a função $f(a, b, c, d) = \sum m(0, 1, 4, 5, 12, 13)$

9. Minimizar na forma SOP a função $f(a, b, c, d) = \sum m(0, 2, 8, 9) + d(1, 13)$

10. Minimize a função $f(a, b, c, d, e) = \sum m(1, 2, 3, 5, 9, 10, 11, 18, 19, 20, 21, 23, 25, 26, 27)$.

11. Minimizar a função $f(a, b, c, d, e) = \sum m(0, 4, 10, 13, 15, 16, 22, 23, 26) + d(5, 11, 12, 14, 18, 21, 24)$.

12. Minimizar na forma SOP a função $f(a, b, c, d, e) = \sum m(0, 1, 2, 9, 13, 16, 18, 24, 25) + d(8, 10, 17, 19)$

4.3 Minimização dois-níveis de múltiplas funções

4.3.1 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 4.22 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) $a\,\overline{b}$, $\overline{a}\,b\,\overline{c}$, $\overline{b}\,c$ e $a\,\overline{c}$; as portas lógicas OU realizam, respectivamente da esquerda para a direita, as funções $f_1(a,b,c) = \overline{a}\,b\,\overline{c} + a\,\overline{b}$, $f_2(a,b,c) = \overline{a}\,b\,\overline{c} + \overline{b}\,c$ e $f_3(a,b,c) = \overline{a}\,b\,\overline{c} + a\,\overline{c}$.

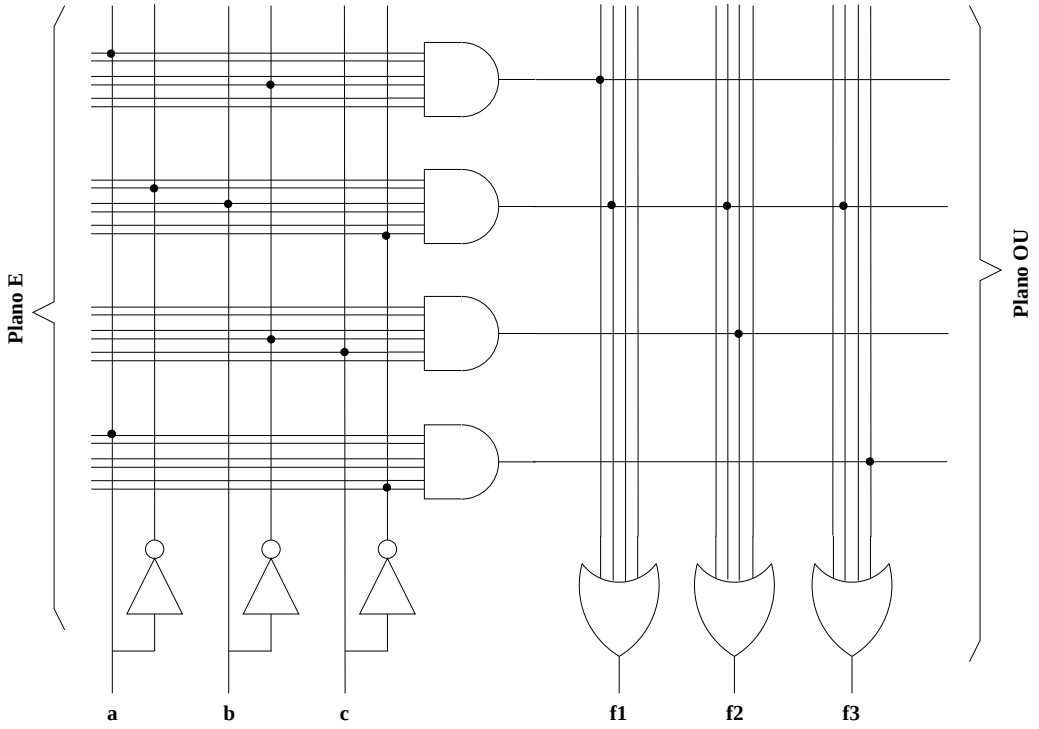


Figura 4.22: Esquema lógico de um PLA.

Para não sobrecarregar o diagrama, em geral desenha-se de forma simplificada como o mostrado na figura 4.23.

PLAs comerciais têm tipicamente 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.

4.3.2 Minimização conjunta

Uma vez que em um PLA podem ser realizadas várias funções, a minimização de um conjunto de funções (e não apenas de uma única função) torna-se o alvo de interesse. Vamos analisar dois exemplos:

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

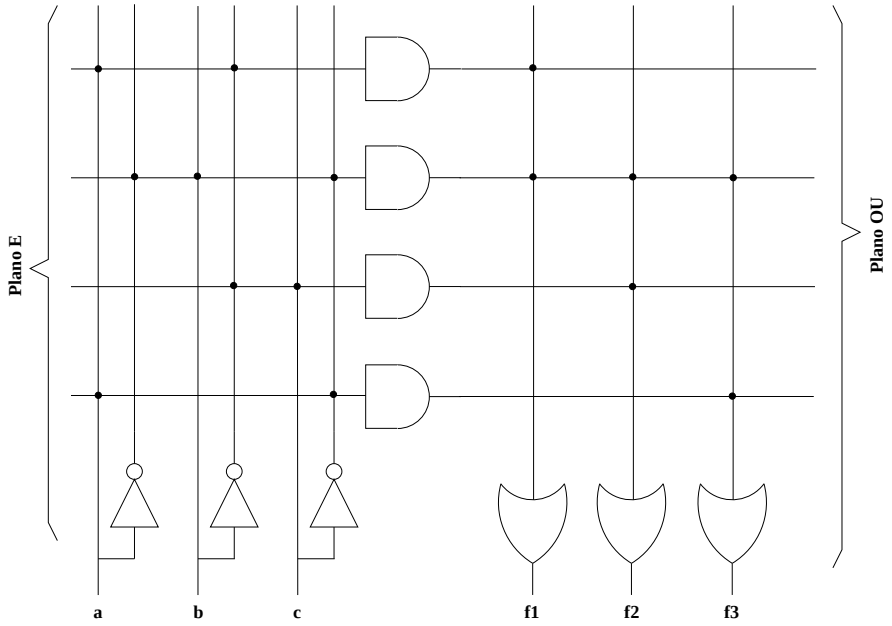


Figura 4.23: Esquema simplificado de um PLA.

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

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

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

Note, porém, que podemos escrever $f_1(x_1\,x_2\,x_3) = \overline{x}_1\,\overline{x}_2 + \overline{x}_2\,x_3 = \overline{x}_1\,\overline{x}_2 + x_1\,\overline{x}_2\,x_3$ e $f_2(x_1\,x_2\,x_3) = x_1\,x_3 + x_1\,x_2 = x_1\,x_2 + x_1\,\overline{x}_2\,x_3$. 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 funcões.

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

$f_0 = a$

$f_1 = \overline{b}$

$f_2 = bc + ab$

$f_3 = \overline{a}\,\overline{b} + \overline{a}\,c$

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

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

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

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

$f_3 = \overline{a}\,\overline{b} + \overline{a}\,b\,c$

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 pôde-se reduzir o número total de produtos, mas o número de somas aumentou. Existe alguma vantagem nisso ? Que impacto isto tem no custo de implementação ? Por essas e outras, minimização de múltiplas funções é um problema mais complexo que o da minimização individual de funções.

Exercícios:

1. Desenhe UM circuito que implementa as duas funções do Exemplo 1, utilizando apenas portas NÃO-E.

2. Desenhe o PLA que realiza as quatro funções do Exemplo 2, utilizando 4 portas E.

Considerações sobre o custo a ser minimizado

Quando pensamos em minimizar um conjunto de funções, reduzir o número de certos elementos necessários para a sua implementação é a principal preocupação. Entre este elementos, alguns são bastante intuitivos:

- reduzir o número total de produtos. Em um PLA, significa reduzir o número de linhas no plano 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)

Estas noções podem ser equacionadas da seguinte forma :

> CUSTO = número total de portas E + a (número total de entradas para as portas E) + b(número total de entradas para as portas OU).

com $0 \leq a, b < 1$ (i.e., minimizar o número total de produtos é o principal critério; os outros são considerados secundários ou irrelevantes).

Em um PLA com certo número fixo de entradas, de portas E e de portas OU, a área do chip também é fixa. Em particular, a área ocupada por um implicante primo (produto) independe do seu número de literais. Portanto, podemos dizer que o custo de todos os implicantes primos é o mesmo e é razoável assumirmos $a = 0$. Em termos de seleção de uma cobertura mínima, isto implica que o número de literais nos implicantes primos não é relevante. Similarmente, utilizar um produto a mais do que o mínimo necessário para a realização de uma função (ou seja, aumentar o número de entradas em uma porta OU) não afeta o custo; ou seja, $b = 0$. Isto significa que uma vez que um implicante primo é identificado como essencial para uma das funções, ele pode ser utilizado por qualquer uma das outras funções (mesmo que não seja implicante primo dela), sem custo adicional. Assim, na

minimização de funções com o objetivo de implementação em PLA é (de um modo geral) razoável considerarmos $a = b = 0$, isto é, preocuparmo-nos em apenas minimizar o número total de produtos necessários para realizar todas as funções.

A realização das funções de outra forma, não utilizando PLA, pode requerer a e b não nulos. No entanto, ainda é razoável supormos $a \ll 1$ e $b \ll 1$ (isto é, o principal critério é a minimização do número total de produtos). Se utilizarmos $a \ll 1$ e $b = 0$, a chance de termos uma tabela cíclica complicada é maior do que quando utilizamos $a = b = 0$.

Quando consideramos a minimização de múltiplas funções, podemos aplicar um processo análogo ao do algoritmo QM. Numa primeira etapa são calculados todos os implicantes primos e numa segunda etapa é escolhida uma cobertura mínima. O problema da escolha de uma cobertura mínima pode ficar mais simples se considerarmos $a = b = 0$ na equação do custo acima (veremos isso mais adiante através de exemplos).

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 isto analisando um exemplo concreto.

Exemplo 3: Considere as funções do exemplo 1, escritas na forma SOP canônica

$$f_1(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3 + x_1\overline{x}_2x_3 = \sum m(0, 1, 5)$$

$$f_2(x_1, x_2, x_3) = x_1\overline{x}_2x_3 + x_1x_2\overline{x}_3 + x_1x_2x_3 = \sum m(5, 6, 7)$$

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 implicante de f_1 e de f_2: 101 (ou seja, 101 é implicante (não é primo!) de ambas as funções e não existe nenhum outro implicante 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

Exemplo 4: 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 4.1 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 4.1), indicando-se a qual das funções ela 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 4.1 são aqueles marcados por a, b, c, d e e.

$x_1x_2x_3$	f_1	f_2	
000	1		$\surd$
001	1		$\surd$
101	1	1	e
110		1	$\surd$
111		1	$\surd$

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

Tabela 4.1: 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 4.2 mostra a **tabela de implicantes primos** de $\mathbf{f} = (f_1, f_2)$, utilizada para a seleção de uma cobertuta mínima. A tabela contém colunas correpondentes 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 indica que ele é implicante daquelas funções. Por exemplo, na coluna ao lado esquerdo de $00X$ aparece (1) para indicar que $00X$ é um implicante da função f_1; ao lado de 101 aparece $(1, 2)$ para indicar que 101 é implicante de f_1 e de f_2. Para cada linha, marcam-se com $\surd$ as colunas correspondentes aos mintermos cobertos pelo implicante primo, desde que o implicante primo seja implicante 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 implicante de f_2).

Após a construção da tabela, deve-se primeiramente selecionar os implicantes primos essencias, 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 4.2.

Se levarmos em consideração apenas a minimização do número de produtos, podemos dizer que o implicante (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) domina as outras duas. Neste caso, temos uma table cíclica e utilizaremos o método de Petrick para resolvê-lo.

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

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).

Exercício: Desenhe o PLA que realiza as duas funções minimizadas do exemplo 4.

Nem sempre as soluções relativas ao custo mais simples serão as mesmas às relativas ao custo mais complexo, como veremos no seguinte exemplo.

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 através dos mapas de Karnaugh. Na figura 4.24 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 cobertos pelos implicantes de um produto de funções de ordem maior. Por xemplo, 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.

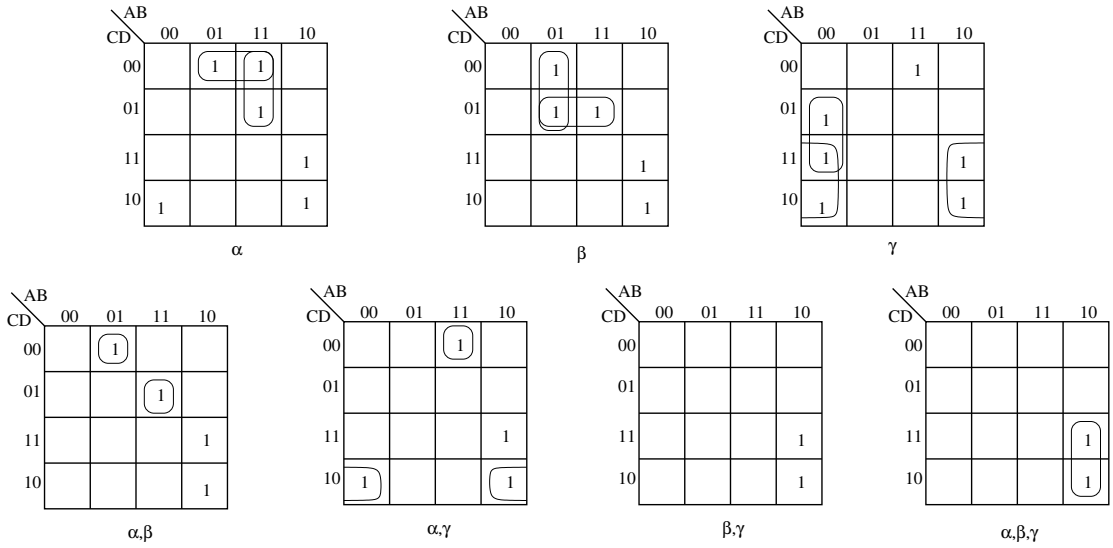


Figura 4.24: Implicantes primos da função $\mathbf{f} = (f_\alpha, f_\beta, f_\gamma)$.

$101X$	$\alpha\ \beta\ \gamma$
$X010$	$\alpha\ \gamma$
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 4.3.

	f_α	f_β	f_γ	IP
0001			1	√
0010	1		1	√
0100	1	1		i
0011			1	√
0101		1		√
1010	1	1	1	√
1100	1		1	j
1011	1	1	1	√
1101	1	1		k

	f_α	f_β	f_γ	IP
00X1			1	f
001X			1	√
X010	1		1	g
010X		1		d
X100	1			b
X011			1	√
X101		1		e
101X	1	1	1	h
110X	1			c

	f_α	f_β	f_γ	IP
X01X			1	a

Tabela 4.3: 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 4.4. 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 4.4: 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	$\surd$				
	α	(c) 110X		$\surd$			
	β	(d) 010X			$\surd$	$\surd$	
	β	(e) X101				$\surd$	$\surd$
	$\alpha\beta$	(i) 0100	$\surd$		$\surd$		
	$\alpha\beta$	(k) 1101		$\surd$			$\surd$

- 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 4.25.

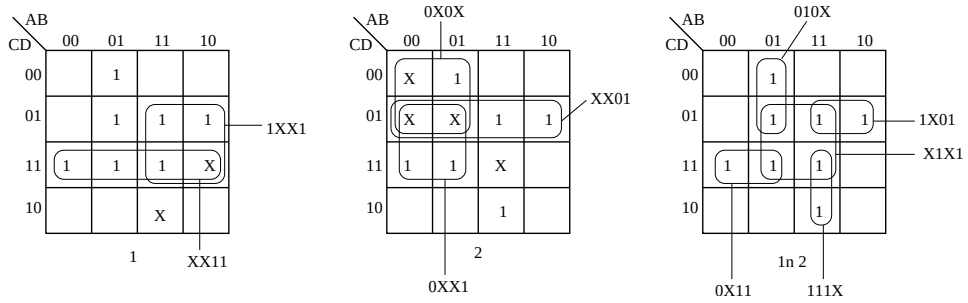


Figura 4.25: Implicantes primos da função $\mathbf{f} = (f_1, f_2)$.

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, pdemos eliminar a linha dos implicantes $XX11$ e $0XX1$ pois estes são dominados pela linha do implicante $0X11$. Similarmente, podemos eliminar as linhas dos implicantes $1XX1$ e $XX01$.

			f_1				f_2			
			3	7	9	13	3	7	9	13
	1	1XX1			√	√				
	1	XX11	√	√						
	2	XX01							√	√
	2	0XX1					√	√		
	1,2	0X11	√	√			√	√		
	1,2	1X01			√	√			√	√
	1,2	X1X1		√		√		√		√

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

			f_1				f_2			
			3	7	9	13	3	7	9	13
**	1,2	0X11	√	√			√	√		
**	1,2	1X01			√	√			√	√
	1,2	X1X1		√		√		√		√
			√	√	√	√	√	√	√	√

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
$f_1(a,b,c,d) = \sum m(0,2,7,10) + d(12,15)$
$f_2(a,b,c,d) = \sum m(2,4,5) + d(6,7,8,10)$
$f_3(a,b,c,d) = \sum m(2,7,8) + d(0,5,13)$

	f_1	f_2	f_3	IP
0000	1		1	√
0010	1	1	1	m
0100		1		√
1000		1	1	l
0101		1	1	√
0110		1		√
1010	1	1		√
1100	1			k
0111	1	1	1	j
1101			1	√
1111	1			√

	f_1	f_2	f_3	IP
00X0	1		1	i
X000			1	h
0X10		1		g
X010	1	1		f
010X		1		√
01X0		1		√
10X0		1		e
01X1		1	1	d
X101			1	c
011X		1		√
X111	1			b

	f_1	f_2	f_3	IP
01XX		1		a

Tabela 4.5: Método tabular para cálculo dos implicantes primos de $\mathbf{f} = (f_1, f_2, f_3)$.

O cálculo dos implicantes pelo método tabular é mostrado na tabela 4.5. Os implicantes 1100, $X101$ e $10X0$ cobrem apenas don't cares e podem ser desconsiderados na etapa de cálculo de cobertura mínima.

Os implicantes podem também ser obtidos pelo mapa de Karnaugh. Veja a figura 4.26.

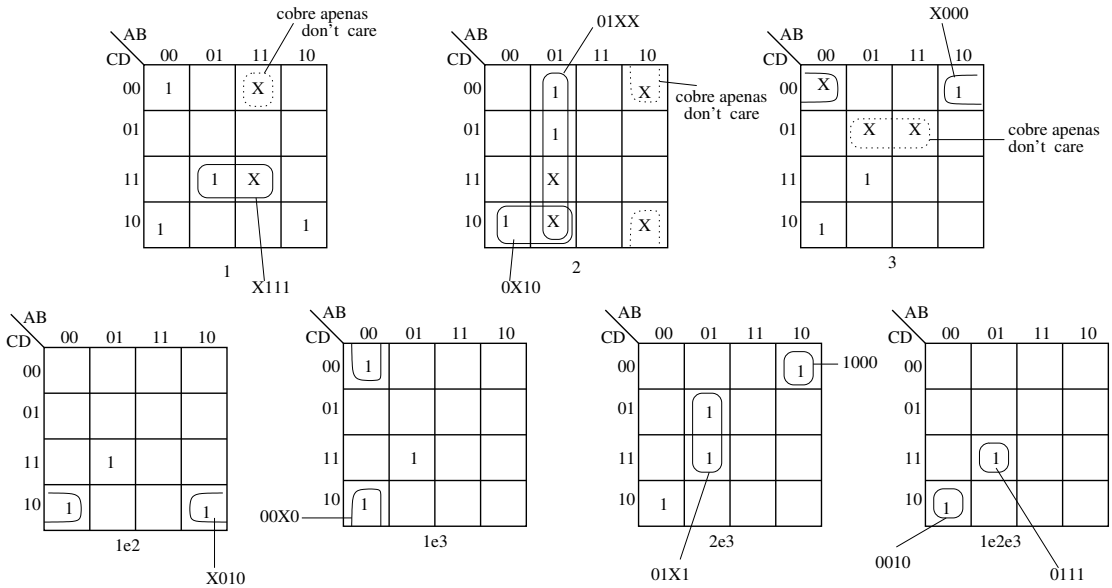


Figura 4.26: Implicantes primos da função $\mathbf{f} = (f_1, f_2, f_3)$.

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

			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 essencias 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 $\Longrightarrow$ $01XX$, $X010$ (pois 0111 é coberto por $01XX$)
f_3: $00X0$, $X000$, 0111.

Exercícios:

1. Para cada uma das soluções do exemplo 5, desenhe o PLA correspondente. Comente as diferenças.

2. Deseja-se projetar um circuito com quatro entradas e duas saídas e que realiza a adição módulo 4. Por exemplo, $(3+3) mod\, 4 = 2$, etc. Os números a serem adicionados são dados em binário respectivamente por $x_2\, x_1$ e $y_2\, y_1$. A saída também deve ser dada em binário ($z_2\, z_1 = 00$ se a soma é 0, $z_2\, z_1 = 01$ se a soma é 1, etc).

   a) determine uma função na forma SOP canônica para z_1 e para z_2

   b) Simplifique-as individualmente

   c) Simplifique-as em conjunto

3. Código BCD refere-se à codificação de dígitos decimais de 0 a 9 pela respectiva representação binária. Para tanto são necessários 4 bits. As combinações binárias de 0000 a 1001 são utilizadas para codificação e as demais não são utilizadas.

O incremento por 1 do código BCD pode ser definido pela seguinte tabela-verdade:

$abcd$	$wxyz$
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	0000
1010	XXXX
...	...
1111	XXXX

Em outras palavras, pode-se pensar esta tabela-verdade como representando 4 funções (w, x, y, e z) com 4 entradas.

Utilize o programa ESPRESSO para obter a minimização individual e conjunta dessas funções. Compare o custo para implementação desses dois casos.

4. Considere um subtrator para números de dois bits. As entradas $a\,b$ e $c\,d$ definem dois números binários N_1 e N_2 (i.e., $N_1 = ab$ e $N_2 = cd$). Suponha que $N_1 \geq N_2$. As saídas $f\,g$ do circuito corsespondem à diferença $N_1 - N_2$ (i.e., $f\,g = N_1 - N_2$).

   a) Escreva a tabela-verdade para $f\,g$

   b) Encontre a forma SOP minimal de f e de g

   c) Encontre a forma POS minimal de f e de g

   d) Qual das duas formas ((b) ou (c)) é mais econômica?

   e) Minimize f e g em conjunto (forma SOP) para implementação em PLA e compare o resultado obtido com os resultados do item (b).

5. Minimizar para implementar em PLA as funções
   $f_1(a,b,c,d) = \sum m(0,2,6,7,15) + d(8,10,14)$
   $f_2(a,b,c,d) = \sum m(0,1,3,7,15) + d(8,10,14)$

4.4 Outros algoritmos de minimização lógica 2-níveis

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 algoritmo QM é um processo demorado e além disso utiliza muita memória, o que limita sua utilidade na prática para funções com relativamente poucas variáveis (poucas dezenas).

Muito esforço ocorreu nas décadas de 1980 e início da década de 1990 no sentido de se desenvolver programas para minimização de funções com várias variáveis. Um dos motivadores deste esforço são os PLAs, dispositivos programáveis, que permitem a realização dois níveis de múltiplas funções. Os esforços realizados foram no sentido de achar alternativas que não requeressem o cálculo explícito de todos os implicantes primos, formas eficientes para o cálculo de coberturas mínimas e heurísticas eficientes nesses processos.

Hoje em dia, ESPRESSO (desenvolvido por um grupo da Universidade de Berckley [Brayton et al., 1984, McGreer et al., 1993]) é a referência para minimização lógica dois-níveis. ESPRESSO é o resultado de uma série de esforços, realizados principalmente na década de 1980, com o objetivo de contornar as limitações do algoritmo QM. Para maiores detalhes veja [Brayton et al., 1984], capítulo 7 de [Micheli, 1994], seção 6.10 de [Hill and Peterson, 1993]. ESPRESSO não calcula os implicantes primos explicitamente; ele utiliza uma série de heurísticas. Além disso, ele também realiza minimização de múltiplas funções e de funções multi-valoradas (lógica multi-valores). Depois do ESPRESSO, foram propostas outras melhorias (como o uso de BDD — Binary Decision Diagrams) por Coudert e outros [Coudert, 1994, Coudert, 1995], e mais recentemente o BOOM [Hlavicka and Fiser, 2001, Fišer and Hlavicka, 2003].

Capítulo 5

Lógica combinacional modular e multi-níveis

Uma possível estruturação de um programa de computador é a sua decomposição em módulos/funções. Do ponto de vista global, apenas interessam as entradas e saídas dos módulos para que os mesmos possam ser compostos de forma adequada no programa.

Este mesmo princípio pode ser observado também no projeto de circuitos lógicos. Alguns circuitos comumente utilizados podem ser modularizados e utilizados na realização de circuitos complexos. Exemplos de módulos bastante usados são os decodificadores, codificadores, multiplexadores e demultiplexadores.

Quando módulos são utilizados, em geral a realização de circuitos tende a ser multi-níveis (mais de dois níveis de portas lógicas). Uma conseqüência imediata disso é que esses circuitos serão mais lentos que os circuitos dois-níveis. No entanto, há casos em que uma realização multi-níveis utiliza menos portas lógicas do que uma realização dois níveis. Além disso, a modularidade é um aspecto altamente atraente no projeto de circuitos complexos (por possibilitar, por exemplo, testes por partes e uma melhor visão de sua estrutura global).

Esta seção apresenta circuitos conhecidos por decodificadores, codificadores, multiplexadores e demultiplexadores. Apresenta também alguns exemplos de sua utilização em circuitos específicos e na realização de funções booleanas quaisquer. Inclui também uma breve introdução à lógica multi-níveis. Referências para este capítulo: [Nelson et al., 1995, Hill and Peterson, 1993].

5.1 Decodificadores

Decodificadores são circuitos combinacionais com n entradas e 2^n saídas. Para cada uma das 2^n possíveis combinações de valores para a entrada, apenas uma saída toma valor 1. A figura 5.1 mostra um esquema genérico de um decodificador $n : 2^n$. As entradas $x_{n-1} \ldots x_1\, x_0$ podem ser interpretadas como um número binário entre 0 e $2^n - 1$ e tem-se $z_i = 1 \Longleftrightarrow \sum_{k=0}^{n-1} 2^k\, x_k = i$.

Exemplo: Seja o decodificador $2 : 4$ e suponha que as entradas são $B\,A$ (i.e., $x_0 = A$ e $x_1 = B$). Um circuito correpondente ao decodificador é ilustrado na figura 5.2. No caso temos $z_0 = \overline{B}\,\overline{A}$, $z_1 = \overline{B}\,A$, $z_2 = B\,\overline{A}$ e $z_3 = BA$.

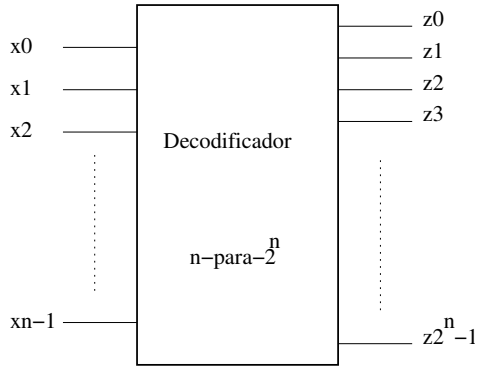


Figura 5.1: Esquema de um decodificador $n : 2^n$.

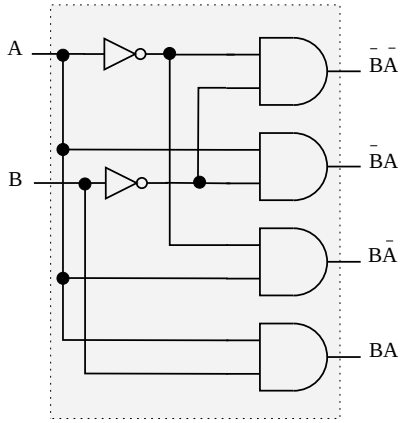


Figura 5.2: Circuito de um decodificador $2 : 4$.

Conceitualmente, o circuito acima poderia ser extendido para realizar decodificadores $n : 2^n$, para um valor de n arbitrariamente grande. No entanto, na prática existem limitações tecnológicas (físicas) conhecidas como *fan-in* (número máximo de entradas possíveis em uma porta lógica) que restringem este valor n. Para valores grandes de n, decodificadores são realizados por circuitos multi-níveis, conforme veremos mais adiante.

5.1.1 Memórias ROM

Um exemplo de uso de decodificadores são as memórias do tipo ROM (*Read-Only Memory*). Suponha por exemplo uma memória com 4 posições. O endereço destas posições pode ser codificado em dois bits $x_1\, x_0$. Decodificadores podem ser utilizados para se endereçar uma certa posição na memória, gerando-se um sinal na linha de saída que corresponde à posição a ser endereçada.

A cada endereço $(x_1\, x_0)$ fonecido como entrada do decodificador, apenas uma saída do decodificador ficará ativa. A palavra na posição de memória endereçada (isto é, o dado armazenado na linha de saída ativada) será transmitida para a saída $(z_3\, z_2\, z_1\, z_0)$ (note que o esquema da figura está simplificado; a rigor, cada porta OU possui exatamente 4 entradas que poderão ou não estar conectadas a cada uma das linhas de saída do decodificador).

Generalizando o esquema acima, para 2^n posições de memória com palavras de m bits, precisaríamos de um decodificador $n : 2^n$ e um plano OU com m portas (um para cada bit da palavra).

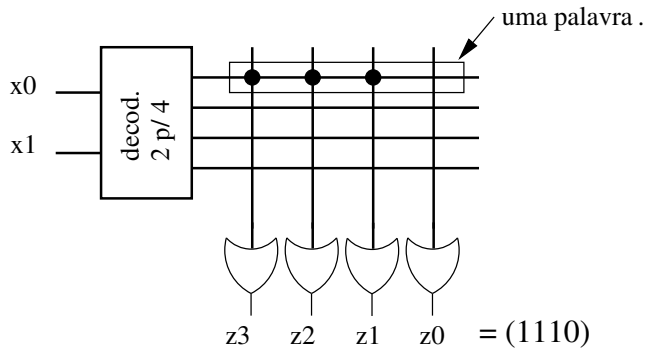


Figura 5.3: Esquema de uma memória ROM com 4 posições e palavras de 4 bits.

Memórias ROM possuem uma estrutura semelhante aos PLAs (i.e., um plano de portas E e um plano de portas OU). As diferenças em relação a um PLA são o fato de que ROMs possuem necessariamente 2^n portas E (todos os produtos são canônicos) e apenas o plano OU é programável. Se o plano OU tem conexões fixas, trata-se de uma ROM. Se o plano OU pode ser programado, então trata-se de uma PROM (*Programmable ROM*), e se o plano OU pode ser reprogramado trata-se de uma EPROM (*Erasable Programmable ROM*).

5.1.2 Realização de funções com decodificadores

Uma vez que um decodificador $n : 2^n$ realiza todos os produtos canônicos de n variáveis, qualquer função com n variáveis pode ser realizada com um decodificador $n : 2^n$ e uma porta OU (com um número de entradas maior ou igual ao número de 1s da função) ou uma porta NÃO-OU (com um número de entradas maior ou igual ao número de 0s da função).

O custo da realização de uma função com decodifcadores é, em termos de portas lógicas, (muito provavelmente) maior que o da realização SOP minimal. No entanto, a simplicidade de projeto torna-o atraente. Além disso, quando múltiplas funções precisam ser realizadas, menor tende a ser a diferença dos custos entre a realização SOP minimal e a realização com decodificadores.

Exemplo: A função $f(a,b,c) = \sum m(0,1,4,6,7) = \prod M(2,3,5)$ pode ser realizada usando decodificadores conforme ilustrado na figura 5.4. No caso da realização com porta NÃO-OU, observe que $f(a,b,c) = \prod M(2,3,5) = \overline{\overline{M_2 \cdot M_3 \cdot M_5}} = \overline{\overline{M_2} + \overline{M_3} + \overline{M_5}} = \overline{m_2 + m_3 + m_5}$.

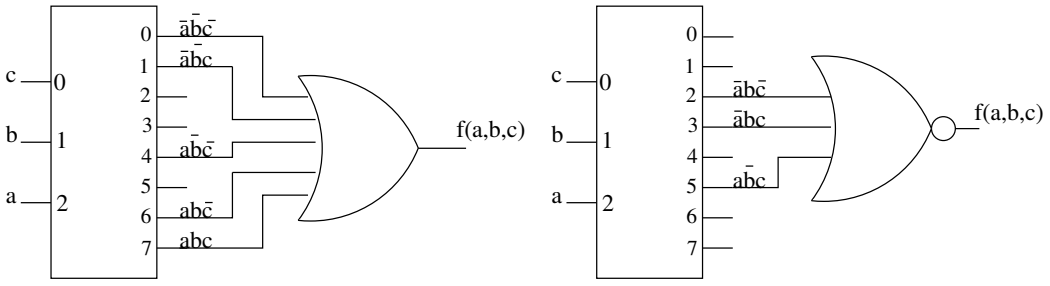


Figura 5.4: Realização de $f(a,b,c) = \sum m(0,1,4,6,7)$ com decodificador $3 : 8$ e uma porta OU (esquerda) ou uma porta NÃO-OU (direita).

5.1.3 Realização multi-níveis de decodificadores

Vimos que decodificadores possuem n entradas e 2^n saídas e que sua realização trivial utiliza 2^n portas E, com n entradas cada. Para contornar o problema de *fan-in* (número máximo de entradas possíveis em uma porta lógica), decodificadores com grande número de entradas podem ser realizados por circuitos com múltiplos níveis. A figura 5.5 mostra como pode ser realizado um decodificador $12 : 2^{12}$ em três níveis.

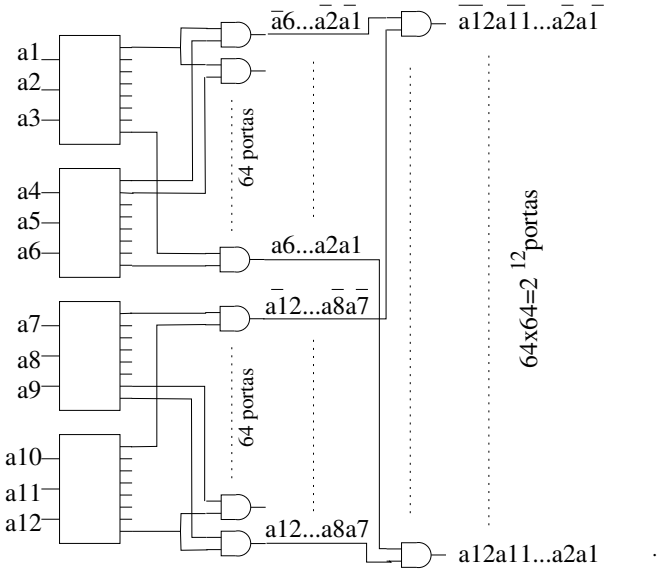


Figura 5.5: Realização três-níveis de um decodificador $12 : 2^{12}$.

No primeiro nível são usados 4 decodificadores $3 : 8$. No segundo nível, 64 portas E de duas entradas são usadas para combinar cada uma das 8 saídas do primeiro decodificador com cada uma das 8 saídas do segundo decodificador. A mesma coisa para as saídas do terceiro com as do quarto decodificador. Cada uma das saídas das primeiras 64 portas E do segundo nível são combinadas com cada uma das saídas das últimas 64 portas E no mesmo nível, resultando em um total de $64 \times 64 = 2^{12}$ portas E no terceiro nível. As saídas dessas 2^{12} portas E correspondem aos produtos canônicos de 12 variáveis.

A solução acima utiliza portas E com três entradas no primeiro nível e portas E com duas entradas nos demais níveis. Se o circuito fosse realizado em apenas um nível, as portas E teriam 12 entradas.

Em uma outra possível realização, poderíamos substituir as 128 portas E de duas entradas no segundo nível acima por 2^{12} portas E de quatro entradas e eliminar as portas do terceiro nível. Isto aparentemente reduziria o número total de portas, mas uma vez que 2^{12} domina de longe 128 e uma vez que as portas agora teriam quatro entradas em vez de duas, não se pode dizer que há economia no custo total.

Um outro problema devido às limitações tecnológicas é o conhecido por *fan-out* (número máximo de portas que podem ser alimentadas por uma saída de uma porta lógica). No caso da realização três-níveis do decodificador $12 : 2^{12}$ visto acima, as saídas das portas no segundo nível alimentam 64 portas no terceiro nível.

Para contornar o *fan-out*, uma possível solução são as realizações em estruturas de árvore. A figura 5.6 mostra a realização de um decodificador 3 : 8 em uma estrutura de árvore. Em vez de termos todas as variáveis alimentando portas no primeiro nível, temos variáveis que alimentam portas nos outros níveis, como o exemplo mostrado na figura 5.2. Usando este esquema, pode-se reduzir o número de portas no próximo nível que precisam ser alimentadas pela saída de uma porta no nível anterior. Mesmo assim, o problema de *fan-out* não é totalmente eliminado pois as variáveis introduzidas nos níveis posteriores do circuito precisam alimentar muitas portas. No entanto, é mais fácil controlar o sinal de algumas poucas entradas (variáveis) para que eles sejam capazes de alimentar um maior número de portas do que fazer o mesmo com as saídas das portas lógicas. Esta solução possui um maior número de níveis e um maior número de portas lógicas do que o esquema mostrado na figura 5.5, mas para decodificadores de muitas entradas pode ser a única solução.

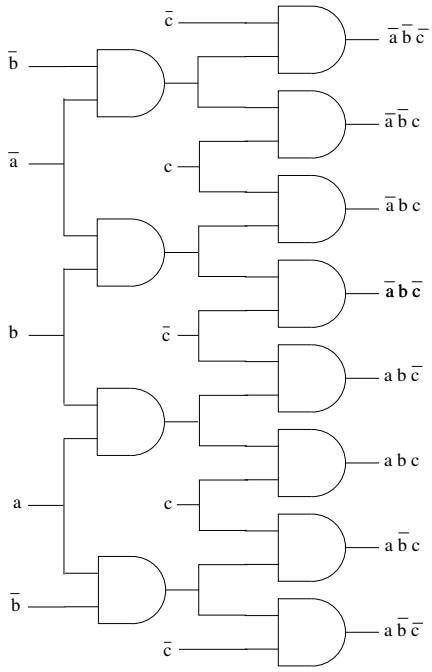


Figura 5.6: Decodificador em estrutura de árvore.

Na prática, as realizações de decodificadores para um número grande de entradas é baseada em uma combinação das estruturas da figura 5.5 e da figura 5.6.

5.2 Codificadores

Codificadores são circuitos combinacionais que são o inverso de decodificadores. Um codificador de n entradas deve possuir s saídas satisfazendo

$$2^s \geq n \quad \text{ou } s \geq \log 2n$$

Usualmente deve-se ter apenas uma entrada ativa e a saída será o código binário correspondente à entrada. Isto é, se a i-ésima entrada estiver ativa, a saída será o código binário de i. A figura 5.7 mostra o esquema de um codificador de n entradas.

Embora usualmente os codificadores sejam definidos como um circuito no qual apenas uma entrada

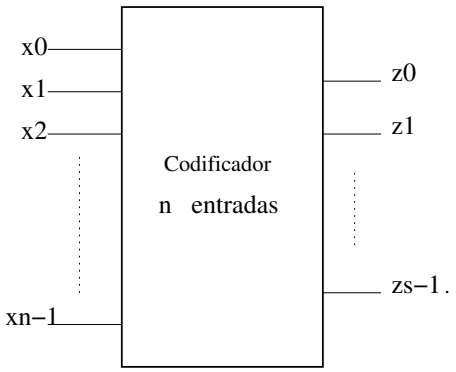


Figura 5.7: Esquema de um codificador.

encontra-se ativa, é possível termos codificadores com propósitos específicos que, por exemplo, para certos tipos de combinação de entradas gera um dado código de saída e para outras combinações de entradas gera outro código de saída.

5.2.1 Teclado

Decodificadores e codificadores podem ser usados, por exemplo, em teclados. Suponha por exemplo que um teclado simplificado possui 70 teclas. Em vez de se ter 70 fios conectando cada uma das teclas a um gerador de código ASCII, podemos ter um esquema como o ilustrado na figura 5.8.

O cruzamento das saídas do decodificador 3-8 com as entradas do codificador 16×4 corresponde a uma tecla. Quando houver sinal na linha de saída correspondente à tecla pressionada, o sinal entrará no codificador. A saída do codificador indica em qual das 14 colunas está a tecla pressionada, enquanto as linhas que ligam a saída do decodificador ao gerador de código ASCII indicam em qual das 5 linhas a tecla está. O contador à esquerda da figura alimenta as entradas do decodificador (varia ciclicamente de 0 a 7), tendo o efeito de gerar saída em uma das 5 linhas, ciclicamente. Obviamente há várias questões que precisam ser consideradas tais como garantir que o contador realiza um ciclo completo durante o período de tempo em que uma tecla está pressionada (para "não perder" a tecla pressionada), mas também não mais que um ciclo (para não produzir o efeito de duas pressões), ou então tratar as combinações de teclas que usualmente são pressionadas simultaneamente (como SHIFT+outra, CTRL+outra). Essas questões não serão consideradas neste curso.

5.3 Multiplexadores (Seletores de dados)

Multiplexadores são circuitos combinacionais com n entradas e uma saída. Apenas uma entrada é selecionada para ser enviada à saída. A entrada selecionada é justamente aquela que é especificada pelos seletores, que consistem de k sinais, satisfazendo $k \geq \log_2 n$. A figura 5.9 mostra um multiplexador 4×1, isto é, um multiplexador de 4 entradas.

Se os seletores forem $S_1S_0 = 00$, então teremos $Z = D_0$; se forem $S_1S_0 = 01$, então teremos $Z = D_1$; $S_1S_0 = 10$, então teremos $Z = D_2$; $S_1S_0 = 11$, então teremos $Z = D_3$.

Podemos observar que Z é uma função das variáveis S_0, S_1 e D_0, D_1, D_2, D_4. Assim, podemos escrever Z como

$$Z(D_0, D_1, D_2, D_4, S_1, S_0) = D_0\,\overline{S}_0\,\overline{S}_1 + D_1\,\overline{S}_0\,S_1 + D_2\,S_0\,\overline{S}_1 + D_3\,S_0\,S_1$$

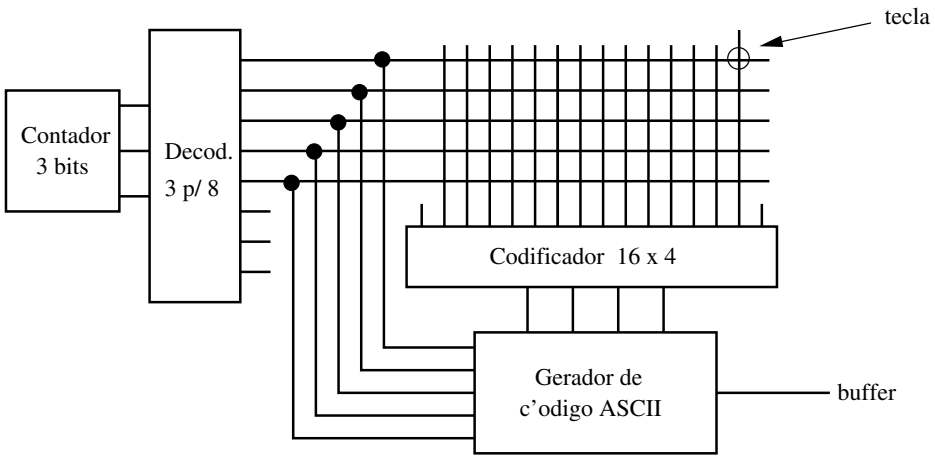


Figura 5.8: Esquema de um teclado.

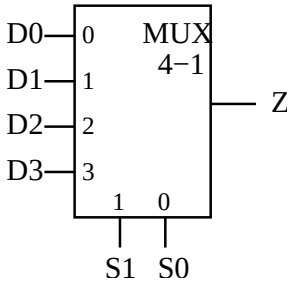


Figura 5.9: Multiplexador de 4 entradas.

e portanto multiplexadores podem ser realizados com circuitos dois-níveis, consistindo de n portas E no primeiro nível e uma porta OU no segundo nível, conforme mostrado na figura 5.10. Note que para cada possível combinação de valores de S_1S_0, apenas um dos produtos toma valor 1 na equação acima.

5.3.1 Realização de funções com MUX

Funções podem ser realizadas utilizando-se MUX. Para tanto, deve-se escolher as variáveis que funcionarão como seletores e, em seguida, as n entradas que poderão ou não depender das demais variáveis.

Exemplo: Realizar a função $f(a,b,c) = \overline{a}\,\overline{b} + a\,c$ usando um MUX 4×1, com as variáveis a e b como seletores.

Uma possível forma para fazer isso é expandir a expressão da função de forma que os literais correspondentes às variáveis a e b apareçam em todos os produtos da expressão resultante. No caso da função dada, fazemos

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

Assim, no MUX 4×1 basta fazermos $s_1 = a$, $s_0 = b$, $D_0 = 1$, $D_1 = 0$, $D_2 = c$ e $D_3 = c$.

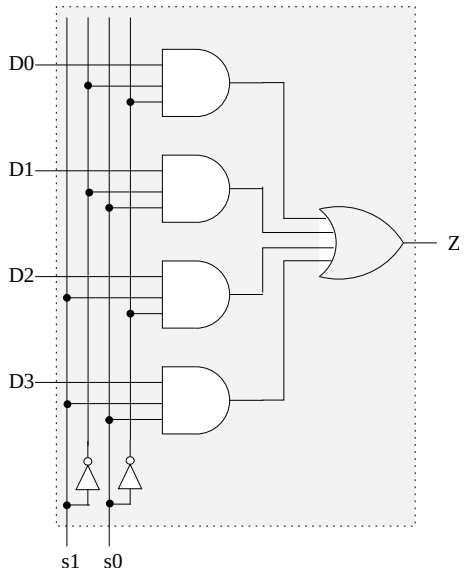


Figura 5.10: Circuito de um multiplexador de 4 entradas.

Exercício 1: Escreva a realização da função $f(a,b,c) = \overline{a}\,\overline{b} + a\,c$ usando um MUX 8×1, com as variáveis a, b e c como seletores.

Exercício 2: Escreva a realização da função $f(a,b,c,d) = \sum m(0,1,3,6,7,8,11,12,14)$ usando um MUX 8×1, com as variáveis a, b e c como seletores.

Exercício 3: Escreva a realização da função $f(a,b,c,d) = \sum m(0,1,3,6,7,8,11,12,14)$ usando um MUX 4×1, com as variáveis a e b (neste caso, as entradas possivelmente dependerão das variáveis c e d e serão necessárias portas adicionais para a realização de f).

5.3.2 Realização de MUX como composição de MUX

Um MUX pode ser realizado como composição de MUXes com um menor número de entradas. Por exemplo, um MUX 4×1 pode ser realizado através de 3 MUX 2×1, conforme ilustrado na figura 5.11.

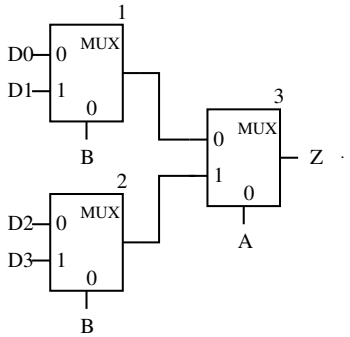


Figura 5.11: Realização de um MUX 4×1 como composição de três MUX 2×1.

Note que se $AB = 00$, então como $B = 0$ a saída do MUX 1 é D_0 e a do MUX 2 é D_2 e, como $A = 0$, a saída do MUX 3 é D_0. Se $AB = 01$ então as saídas são, respectivamente, D_1, D_3 e D_1.

Se $AB = 10$, então as saídas são, respectivamente, D_0, D_2 e D_2. Finalmente, se $AB = 11$, então as saídas são, respectivamente, D_1, D_2 e D_3. Em todos os casos, a terceira saída é a mesma de um MUX 4×1 com AB como entrada para seletores.

Pergunta: E se trocarmos as entradas para os seletores na figura acima? Se colocarmos A no seletor dos MUX 1 e 2 e B na do MUX 3, ainda é possível realizar um MUX 4×1 com AB como entrada para seletores ?

5.3.3 Realização multi-níveis de funções com MUX

Uma função pode ser realizada com múltiplos níveis de multiplexadores. Para cada nível deve-se definir as variáveis que alimentarão os seletores. Em função disso fica definida as variáveis que alimentarão as entradas dos multiplexadores no primeiro nível.

Considere a função $f(a, b, c, d) = \sum m(2, 5, 8, 9, 11, 12, 14, 15)$. Se for utilizado um MUX 8×1, então serão necessários três variáveis para alimentar os seletores. A quarta variável pode ser diretamente alimentada nas entradas, conforme mostrado na figura 5.12.

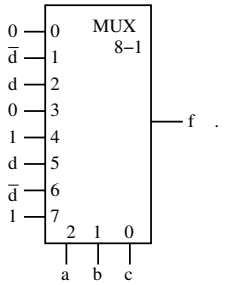


Figura 5.12: Realização de $f(a, b, c, d) = \sum m(2, 5, 8, 9, 11, 12, 14, 15)$ com um MUX 8×1.

Se pensarmos em utilizar MUX 4×1 e MUX 2×1 na realização de f, duas possíveis soluções, mostradas na figura 5.13, são:

1. dois MUX 4×1 no primeiro nível, alimentando um MUX 2×1 no segundo nível
2. 4 MUX 2×1 no primeiro nível e 1 MUX 4×1 no segundo nível.

Estas estruturas podem ser obtidas a partir da análise dos mintermos arranjados em forma tabular, conforme mostrado a seguir. A tabela da esquerda considera o uso das entradas b e c como seletores dos MUXes 4×1 no primeiro nível e o uso da varíável a como seletor do MUX 2×1 do segundo nível. A tabela da direita faz o análogo para a implementação com 4 MUX 2×1 no primeiro nível e 1 MUX 4×1 no segundo nível.

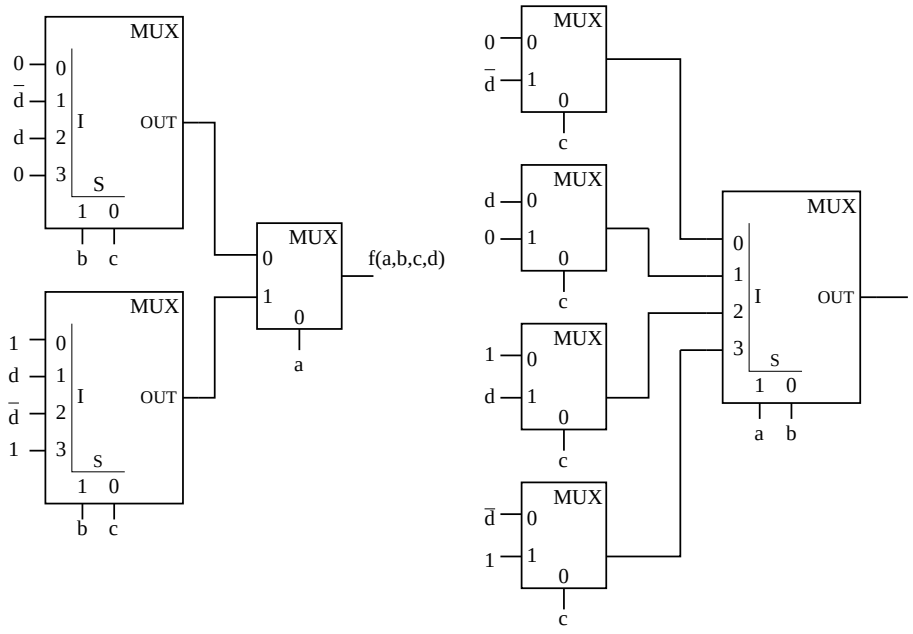


Figura 5.13: Realização de $f(a,b,c,d) = \sum m(2,5,8,9,11,12,14,15)$ com (a) dois MUX 4×1 no primeiro nível e um MUX 2×1 no segundo nível e (b) 4 MUX 2×1 no primeiro nível e 1 MUX 4×1 no segundo nível.

$abcd$	a	bc	d	input
0010	0	01	0	$\overline{d}$
0101	0	10	1	d
1000	1	00	0	$\overline{d}+d=1$
1001	1	00	1	
1100	1	10	0	$\overline{d}$
1110	1	11	0	$\overline{d}+d=1$
1111	1	11	1	

$abcd$	ab	c	d	input
0010	00	1	0	$\overline{d}$
0101	01	0	1	d
1000	10	0	0	$\overline{d}+d=1$
1001	10	0	1	
1100	11	0	0	$\overline{d}$
1110	11	1	0	$\overline{d}+d=1$
1111	11	1	1	

Outra abordagem para determinar a estrutura hierárquica dos MUXes na realizações de funções com múltiplos níveis de MUXes é baseada na aplicação sucessiva da expansão de Shannon. Dependendo da seqüência de variáveis em torno das quais a expansão é aplicada, pode-se chegar a diferentes estruturas.

O teorema de **Expansão de Shannon** afirma que qualquer função f de n variáveis pode ser escrita em termos de funções de $n-1$ variáveis da seguinte forma:

$$f(x_1,\ldots,x_k,\ldots,x_n) = \overline{x}_i f(x_1,\ldots,0,\ldots,x_n) + x_i f(x_1,\ldots,1,\ldots,x_n)$$

As funções $f(x_1,\ldots,0,\ldots,x_n)$ e $f(x_1,\ldots,1,\ldots,x_n)$ são funções de $n-1$ variáveis. O teorema pode ser aplicado recursivamente nessas duas funções.

Exemplo: Consideremos novamente a função $f(a,b,c,d) = \sum m(2,5,8,9,11,12,14,15)$. Sua realização com um MUX 8×1 está mostrada na figura 5.12 acima. Vamos mostrar agora que aplicando-se sucessivamente a expansão de Shannon é possível obter diferentes estruturas de realizações de f

usando MUX.

$$\begin{aligned}
f &= \overline{a}\,\overline{b}\,c\,\overline{d} + \overline{a}\,b\,\overline{c}\,d + a\,\overline{b}\,\overline{c}\,\overline{d} + a\,\overline{b}\,\overline{c}\,d + a\,\overline{b}\,c\,d + a\,b\,\overline{c}\,\overline{d} + a\,b\,c\,\overline{d} + a\,b\,c\,d \\
&= \overline{a}\,\overline{b}\,c\,\overline{d} + \overline{a}\,b\,\overline{c}\,d + a\,\overline{b}\,\overline{c}\,\overline{d} + a\,b\,\overline{c}\,\overline{d} + a\,\overline{b}\,\overline{c}\,d + a\,\overline{b}\,c\,d + a\,b\,c\,\overline{d} + a\,b\,c\,d \quad \text{(rearranjo)} \\
&= \overline{a}\,\overline{b}\,c\,\overline{d} + \overline{a}\,b\,\overline{c}\,d + a\,\overline{c}\,\overline{d} + a\,\overline{b}\,d + a\,b\,c \quad \text{(algumas simplificações)} \\
&= \overline{a}\,(\overline{b}\,c\,\overline{d} + b\,\overline{c}\,d) + a(\overline{c}\,\overline{d} + \overline{b}\,d + b\,c) \quad \text{(expansão em torno de } a) \\
&= \overline{a}\,(\overline{b}(c\,\overline{d}) + b(\overline{c}\,d)) + a\,(\overline{b}(\overline{c}\,\overline{d} + d) + b\,(\overline{c}\,\overline{d} + c)) \quad \text{(expansão em torno de } b) \\
&= \overline{a}\,\overline{b}(c\,\overline{d}) + \overline{a}\,b(\overline{c}\,d)) + a\,\overline{b}(\overline{c}\,\overline{d} + d) + a\,b\,(\overline{c}\,\overline{d} + c) \quad \text{(distribuição com respeito a } a) \\
&= \overline{a}\,\overline{b}(\overline{c}(0) + c\,(\overline{d})) + \overline{a}\,b(\overline{c}\,(d) + c(0)) + a\,\overline{b}(\overline{c}\,(1) + c(d)) + a\,b\,(\overline{c}\,(\overline{d}) + c(1)) \quad \text{(expansão em torno de } c)
\end{aligned}$$

A última expressão acima pode ser realizada com 4 MUX 2×1 no primeiro nível, com c como entrada para os seletores, mais um MUX 4×1 no segundo nível, com a e b como entrada para os seletores.

Nas equações acima, logo após a expansão em torno de b, poderíamos ter prosseguido da seguinte forma:

$$\begin{aligned}
f &= \overline{a}\,(\overline{b}(c\,\overline{d}) + b(\overline{c}\,d)) + a\,(\overline{b}(\overline{c}\,\overline{d} + d) + b\,(\overline{c}\,\overline{d} + c)) \quad \text{(expansão em torno de } b) \\
&= \overline{a}\,(\overline{b}\,c\,(\overline{d}) + \overline{b}\,\overline{c}(0) + b\,\overline{c}\,(d) + b\,c\,(0)) + a\,(\overline{b}\overline{c}\,(1) + \overline{b}\,c(d) + b\,\overline{c}\,(\overline{d}) + b\,c\,(1)))
\end{aligned}$$

Esta última expressão pode ser realizada com 2 MUX 4×1 no primeiro nível, com b e c como entrada para os seletores, mais um MUX 2×1 no segundo nível, com a como entrada para os seletores.

5.4 Demultiplexadores (Distribuidores de dados)

Demultiplexadores são circuitos combinacionais inversos aos multiplexadores, isto é, possuem apenas uma entrada que é transmitida a apenas uma das n saídas. A saída a ser escolhida é selecionada pelos valores dos seletores. Se o demultiplexador possui n saídas, então são necessários k seletores, com $2^k \geq n$.

Pergunta: Para que serve um demultiplexador? Suponha que há dois sistemas A e B e que A possui n saídas que geram ciclicamente sinais que devem ser transmitidos para os respectivos n receptores de B. A transmissão seria direta se houvesse um canal de comunicação entre cada saída de A para a respectiva entrada de B. Se houver apenas um canal de transmissão entre A e B, pode-se utilizar um multiplexador na saída de A e um demultiplexador na entrada de B. Os seletores devem ser ajustados para que, no multiplexador, seja selecionado o sinal que se deseja transmitir a B e, no demultiplexador, seja selecionada a saída conectada à entrada de B para o qual se deseja direcionar o sinal transmitido.

5.5 Lógica combinacional multi-níveis

Existem funções cuja realização por circuitos com mais de dois níveis é natural e simples enquanto que sua realização por circuitos de dois níveis é proibitivamente ineficiente ou caro.

Um desses exemplos é o circuito verificador de paridade. Suponha que em um sistema de transmissão os dados são codificados em 7 bits. O oitavo bit é utilizado para guardar a informação sobre a paridade desses 7 bits. Se o número de bits 1 no dado é impar, o oitavo bit é 1; se é par, o oitavo bit é

0. Assim, a cada grupo de oito bits transmitidos deve-se ter necessariamente um número par de bits 1. Após a transmissão dos dados, verificar se a paridade a cada 8 bits é par é um teste simples que pode detectar erros (não todos) na transmissão. De fato, pode-se mostrar através de uns cálculos que, por exemplo, quando a probabilidade de ocorrer erros na linha de transmissão é de 1 em cada 10^4 bits transmitidos, as chances de erro diminuem de 7×10^{-4} para 28×10^{-8} caso seja feita a verificação de paridade.

Como poderia ser realizado o circuito para verificar a paridade? Suponha inicialmente uma situação com 4 bits (3 de dados e um de paridade). Então, o seguinte circuito produz saída 1 se, e somente se, se a paridade é impar.

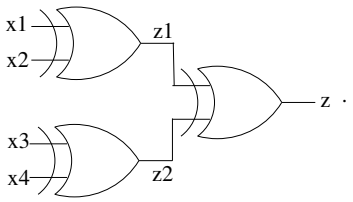


Figura 5.14: Circuito para verificação de paridade para uma entrada de 4 bits.

Isto pode ser verificado analisando-se a seguinte tabela.

# de x_is iguais a 1	# de z_is iguais a 1	z
0	0	0
1	1	1
2	0 ou 2	0
3	1	1
4	0	0

Note que o número de saídas $z_i = 1$ é impar se, e somente se, o número de entradas $x_i = 1$ é também ímpar. Se o número de $z_i = 1$ é impar, então $z = 1$. Assim, se a paridade é par, temos $z = 0$ e se é ímpar temos $z = 1$.

O circuito acima pode ser extendido para verificar a paridade de 8 bits, simplesmente conectando-se a saída de dois circuitos daqueles a uma porta XOR, conforme mostrado na figura 5.15. Para realizar tal circuito são necessários 7 portas XOR ou então 21 portas NÃO-E e 14 inversores.

Se considerarmos a forma SOP, é fácil constatarmos que não é possível fazer nenhuma minimização a partir da forma SOP canônica. Portanto, no caso de 4 variáveis, como são 7 mintermos correspondentes a entradas de paridade ímpar, seriam necessárias 7 portas E com 4 entradas e uma porta OU com 7 entradas (se pensarmos somente em portas com duas entradas, claramente a forma SOP é muito pior que a apresentada acima).

De uma forma geral, podemos dizer que a lógica multi-níveis é mais flexível que a lógica 2-níveis, isto é, circuitos multi-níveis oferecem maiores possibilidades para reduzir a quantidade de portas lógicas necessárias. No entanto, o problema de otimização associado à lógica multi-níveis pode ser

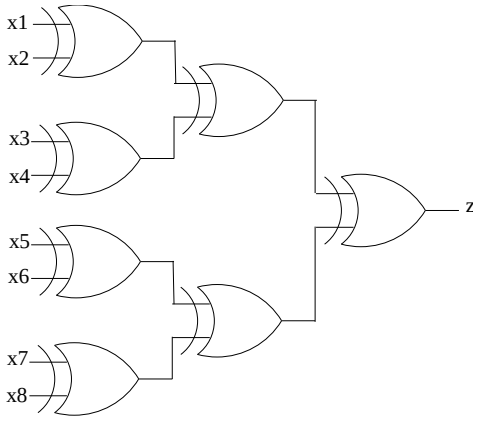


Figura 5.15: Circuito para verificação de paridade para entradas de 8 bits.

muito mais complexo. Não existem técnicas para, dada uma função f qualquer, encontrar a realização ótima multi-níveis de f, para um número de níveis fixo qualquer (exceto para 2 níveis).

Algumas das abordagens existentes para minimização lógica multi-níveis resumem-se à composição seqüencial de circuitos de subfunções (módulos) mais simples. Este assunto não será tratado em detalhes neste curso. A seguir serão apresentadas duas abordagens utilizadas para projeto de circuitos multi-níveis.

5.5.1 Decomposição funcional e estrutural de funções

Não é objetivo deste curso aprofundar-se nestes tópicos. Aqui apresentaremos apenas uma breve introdução.

Na **decomposição funcional** [Perkowski et al., 1995], o objetivo é, dada uma função f de n variáveis, escrevê-la como composição de duas ou mais funções com menor número de variáveis.

Exemplo: seja f uma função com n variáveis $x_1, x_2, \ldots, x_n$. Uma possível decomposição de f pode ser dada por

$$f(x_1, x_2, \ldots, x_n) = g(h(x_1, x_2, \ldots, x_k), x_{k+1}, \ldots, x_n)$$

Tanto g como h são funções que dependem de menos variáveis. Consequentemente, pode ser mais fácil encontrar uma realização eficiente e barata de g e h do que uma realização eficiente e barata de f.

Os circuitos resultantes da decomposição funcional possuem uma estrutura hierárquica, ou seja, de múltiplos níveis, onde cada uma das subfunções pode ser vista como um módulo. Veja a figura 5.16. Esta estrutura de decomposição é apenas um exemplo de uma possível forma de decomposição. Dada uma função qualquer, verificar se ela pode ser expressa em uma certa estrutura de decomposição não é tarefa fácil.

Na **decomposição estrutural** [Jacobi, 1996], o objetivo é, dada uma função f de n variáveis, encontrar subfunções comuns que possam ser compartilhadas. Essa idéia é mostrada através de um exemplo. Consideremos a função f dada por

$$f = x_1\, x_3\, \overline{x}_4 + x_1\, x_5 + x_2\, x_3\, \overline{x}_4 + x_2\, x_5 + x_3\, \overline{x}_4\, x_7 + x_5\, x_7 + x_6$$

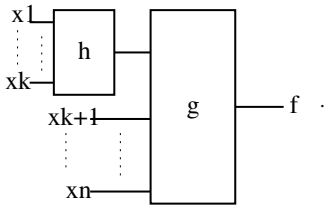


Figura 5.16: Exemplo de decomposição funcional.

Rearranjando os produtos e efetuando algumas manipulações,

$$\begin{aligned} f &= \underbrace{x_1\, x_3\, \overline{x}_4 + x_2\, x_3\, \overline{x}_4}_{\underbrace{(x_1 + x_2)}_{y_1}\, x_3\, \overline{x}_4} + \underbrace{x_1\, x_5 + x_2\, x_5}_{\underbrace{(x_1 + x_2)}_{y_1}\, x_5} + \underbrace{x_3\, \overline{x}_4\, x_7 + x_5\, x_7}_{x_7\, \underbrace{(x_3\, \overline{x}_4 + x_5)}_{y_2}} + x_6 \\ &= \underbrace{y_1\, x_3\, \overline{x}_4 + y_1\, x_5}_{y_1\, \underbrace{(x_3\, \overline{x}_4 + x_5)}_{y_2}} + \underbrace{y_2\, x_7 + x_6}_{y_3} \\ &= y_1\, y_2 + y_3 \end{aligned}$$

O compartilhamento de subfunções resulta em uma estrutura de rede de funções, conforme mostrada na figura 5.17. Este tipo de decomposição pode ser realizado levando-se em conta a implementação

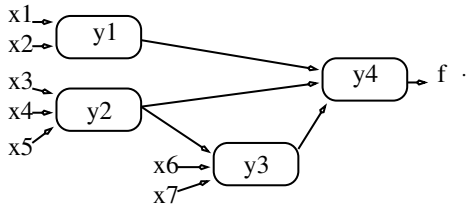


Figura 5.17: Exemplo de decomposição estrutural.

de mais de uma função booleana.

O projeto multi-níveis é um problema computacionalmente difícil. Para abordá-lo, é necessário estabelecer as condições de contorno que caracterizam o tipo de decomposição desejada. Entre estas condições, podemos citar o número máximo de níveis, o número máximo de variáveis de entrada em cada subfunção, a utilização de determinados componentes, etc. A solução do problema, mesmo em condições de contorno bem limitadas, não é trivial ou então nem existe ainda. O projeto multi-níveis pode também ser aplicado no contexto de aprendizado computacional (indução de classificadores) [Zupan, 1997] e de operadores morfológicos para processamento de imagens.

Capítulo 6

Lógica Seqüencial

As notas de aula referentes a este tópico estão resumidas. Alguns detalhes discutidos e exemplos vistos em sala de aula não aparecem aqui. Referências para esta parte de curso: capítulos 6 a 8 de [Nelson et al., 1995], capítulos 9 e 10 de [Hill and Peterson, 1993].

Lógica combinacional: saída depende apenas dos valores correntes da entrada

Lógica seqüencial: saída depende também dos valores das entradas passadas (existe realimentação no circuito). Neste caso, deve existir algum mecanismo que permita "armazenar" os valores passados. Em sistemas digitais, os dispositivos que armazenam dados são denominados **memória**. Um tipo de circuito que tem a capacidade para "armazenar" dados são os chamados flip-flops. Os flip-flops, além de armazenar os dados, tem a capacidade de alterar o valor armazenado mediante sinal de entrada adequado.

Exemplo: Pegue uma porta OU com ambas as entradas em 0 e conecte a saída em uma das entradas. Em seguida, mude o valor da outra entrada para 1. O que acontece se colocarmos o valor dessa mesma entrada de volta para 0?

Pegue uma porta NÃO-OU com ambas as entradas em 0 e conecte o negado da saída em uma das entradas. Em seguida, mude o valor da outra entrada para 1. O que acontece se colocarmos o valor dessa mesma entrada de volta para 0?

6.1 Flip-flops

6.1.1 Flip-flop RS

S (set)

R (reset)

Q (saída ou estado do flip-flop)

S_i denota o valor do sinal que alimenta a entrada S num certo instante de tempo t_i.

R_i denota o valor do sinal que alimenta a entrada R num certo instante de tempo t_i.

Q_i denota a saída ou estado do flip-flop num certo instante de tempo t_i.

Q_{i+1} denota a saída (ou próximo estado) do flip-flop em decorrência de termos, no instante de tempo t_i, os valores S_i e R_i nas entradas S e R.

A figura a seguir mostra o circuito de um flip-flop RS baseado em portas NÃO-OU e a respectiva representação diagramática.

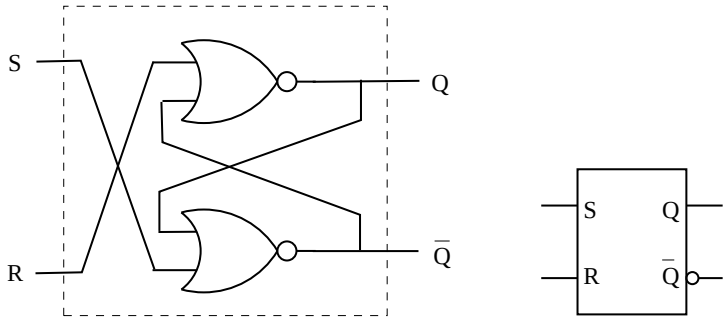


Tabela que descreve o comportamento do flip-flop RS:

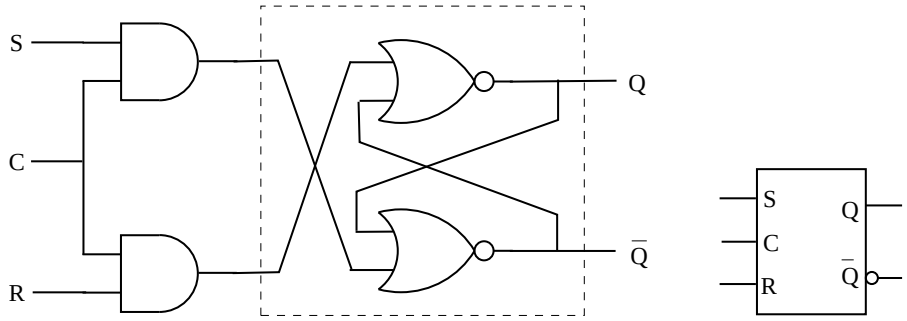
S_i	R_i	Q_i	Q_{i+1}	
0	0	0	0	Nenhuma mudança
0	0	0	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	?	proibido
1	1	1	?	

A condição $R = S = 1$ não é permitida pois, numa situação em que ambas passam simultaneamente para 1, pode ocorrer oscilação da saída (saída não se estabiliza) ou a nova saída Q depende de qual das portas NÃO-E reage primeiro.

Considerando $R_i = S_i = 1$ como don't care, a tabela acima pode também ser expressa pela equação

$$Q_{i+1} = S_i + Q_i \, \overline{R}_i$$

Em geral a mudança de estado dos flip-flops precisa ser coordenado. Para isso, existem os flip-flops controlados como o da figura a seguir. Um sinal de controle usado para coordenar a mudança de estado dos flip-flops é o sinal do *clock*. A figura a seguir mostra um flip-flop RS controlado. Note que existe um terceiro sinal de entrada C. Quando $C = 0$, a saída de ambas as portas E é 0 e portanto mudanças no valor de R e S não tem efeito nenhum sobre o estado do flip-flop. Quando $C = 1$, temos o mesmo comportamento descrito na tabela acima.

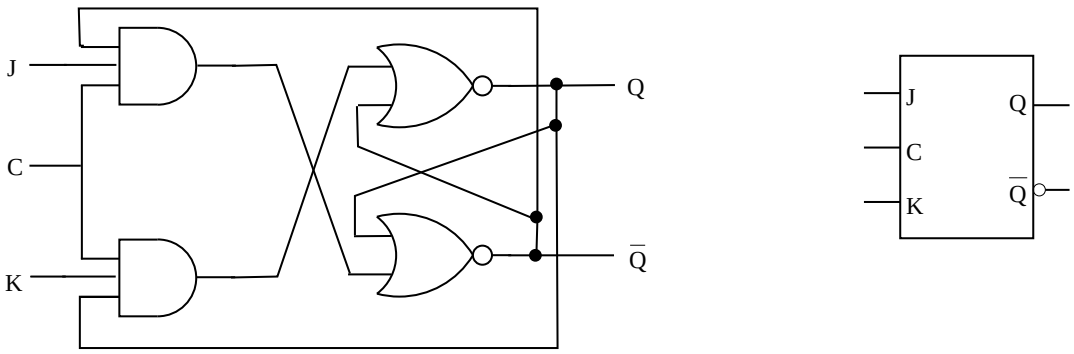


6.1.2 Flip-flop J-K

Vimos que, após ambas as entradas serem ativadas simultaneamente, o flip-flop RS pode ter comportamento imprevisível. Isso, do ponto de vista prático, é desagradável pois o projetista do circuito teria de garantir que as duas entradas nunca ficarão ativas simultaneamente.

Os flip-flops JK são uma variante do RS e não possuem esse problema. No JK, quando ambas as entradas passam para um (1), o circuito tem o efeito de mudar de estado, isto é, se a saída era 1, então passa a ser 0 e vice-versa.

A figura a seguir mostra o circuito de um flip-flop JK e a respectiva representação diagramática.



Vejamos o que acontece quando $J = K = 1$. Suponha inicialmente que $Q^i = 0$ e J_i e K_i passam a 1. Com $C^i = 1$, a porta E de cima fica com saída 1 e em conseqüência temos $Q_{i+1} = 1$ e $\overline{Q}_{i+1} = 0$. Similarmente, se tivéssemos $Q^i = 1$, e J_i e K_i passassem para 1, com $C^i = 1$ teríamos $Q_{i+1} = 0$ e $\overline{Q}_{i+1} = 1$. Ou seja, $J = K = 1$ tem o efeito de inverter o estado do flip-flop.

Tabela de estados do flip-flop JK:

J^i	K^i	C^i	Q^{i+1}	
$\times$	$\times$	0	Q^i	não muda
0	0	1	Q_i	mantém
0	1	1	0	reset
1	0	1	1	set
1	1	1	$\overline{Q}_i$	inverte

Para $C_i = 1$ a equação do próximo estado é dada por

$$Q_{i+1} = J_i\overline{Q}_i + \overline{K}_i Q_i\,.$$

6.1.3 Flip-flop D

É um flip-flop que, quando o controle está ativo, copia a entrada D para a saída Q. Uma possível implementação é alimentar J com o sinal D e K com o seu inverso.

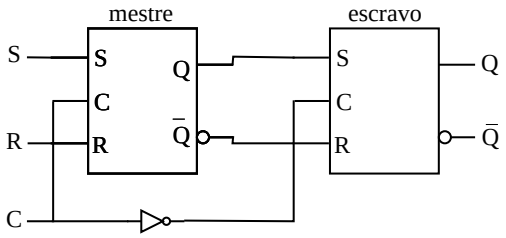
6.1.4 Flip-flop T

É um flip-flop que, quando o controle está ativo, inverte o valor da saída. Uma possível implementação é alimentar J e K de um flip-flop JK com 1.

6.1.5 Flip-flops mestre-escravo

Os flip-flops controlados vistos acima podem mudar de estado várias vezes enquanto o sinal que alimenta o controle C estiver ativo. Em geral, o sinal que alimenta a entrada C é o sinal de *clock*. Em condições ideais, poderíamos fazer com que a duração de um pulso de um sinal de clock seja menor que o tempo necessário para que ocorram mais de uma mudança de estado nos flip-flops. No entanto, na prática, este tipo de controle é difícil e pode nem ser possível. Portanto, existem os flip-flops denominados *edge-triggered* que são aqueles que mudam de estado somente na transição 1 para 0 (descida) ou na transição 0 para 1 (subida) do sinal de controle. Um exemplo desse tipo de flip-flop são os **flip-flops mestre-escravo**.

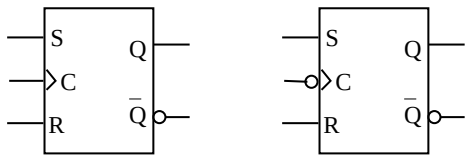
A figura a seguir mostra o esquema de um flip-flop RS mestre-escravo.



Quando o clock está baixo, mudanças nas entradas R e S não tem efeito no flip-flop mestre. O flip-flop escravo está habilitado para mudanças (sua entrada C é 1), mas nenhuma mudança ocorre nas suas entradas R e S (que vem das saídas do flip-flop mestre).

Quando o clock sobe, o mestre muda de estados de acordo com as entradas R e S e o escravo fica desabilitado (deve-se apenas garantir que o escravo fique desabilitado antes que ocorra qualquer mudança na saída do mestre). Quando o clock desce, o mestre fica desabilitado (e portanto "congela" a saída dele) e o escravo se habilita (ou seja, a saída do mestre é copiada para a saída do escravo). Desde que o clock subiu, a saída do mestre pode ter oscilado algumas vezes, porém a saída do flip-flop como um todo só muda quando o clock baixa. Este é, portanto, um exemplo de flip-flop que muda de estado na descida do sinal de clock.

A figura a seguir mostra a representação diagramática de flip-flops RS mestre-escravo com mudança de estado respectivamente na subida e na descida do sinal de controle.



6.2 Exemplos de circuitos seqüenciais

6.2.1 Síncronos × Assíncronos

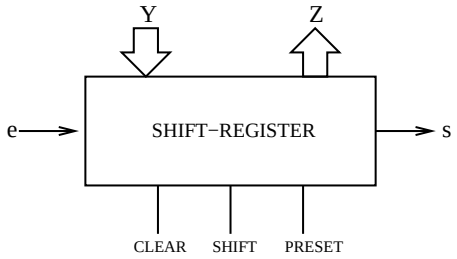
Em **circuitos síncronos**, a mudança de estado de todos os elementos de memória do circuito (flip-flops) ocorre em sincronia ("ao mesmo tempo") e é controlada pelo sinal de um *clock*.

Em **circuitos assíncronos**, a mudança de estado não é sincronizada, ou seja, não se utiliza o sinal de um *clock* para se promover a mudança de estado de todos os flip-flops. Há dois modos de promover a mudança de estado em circuitos assíncronos. No modo pulso, o controle do flip-flop pode ser alimentado por qualquer outro sinal. No modo fundamental, o retardo intrínseco ou propositadamente colocado no circuito é utilizado para funcionar como "memória". Em ambos os casos há restricões que devem ser satisfeitas para o circuito funcionar propriamente.

Neste texto, a maior parte dos exemplos que veremos serão de circuitos síncronos. Quando não for o caso, isso será explicitamente mencionado.

6.2.2 Registradores

A figura a seguir mostra um esquema de um registrador-deslocador (*shift-register*) genérico.



Entrada paralela: $Y = y_n\, y_{n-1} \ldots y_2\, y_1$ são os n bits a serem armazenados no registrador, num pulso do sinal PRESET.

Saída paralela: $Z = z_n\, z_{n-1} \ldots z_2\, z_1$ são os n bits que estão armazenados no registrador.

Entrada serial: e é 1 bit de entrada que ocupará a posição mais à esquerda, num pulso do sinal SHIFT.

Saída serial: s é 1 bit de saída num pulso do sinal SHIFT (é o bit que ocupava a posição mais à direita no registrador quando o sinal SHIFT subiu).

Os seguintes são sinais de controle:

CLEAR (Limpa): zera o registardor

PRESET (Carrega): armazena Y no registrador

SHIFT (desloca): desloca, uma posição, todos os bits para a direita

Podemos pensar em diferentes modos de operação para este tipo de registrador:

- entrada e saída seriais

  A entrada serial (bit e) deve estar sincronizada com o sinal SHIFT.

- entrada paralela e saída serial

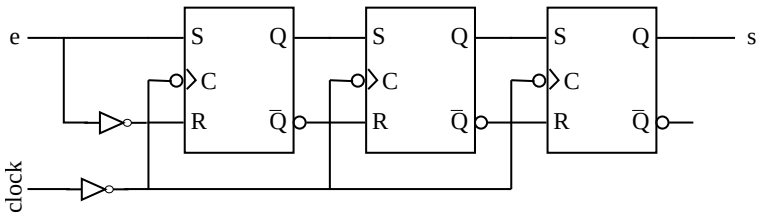
  - CLEAR para zerar o registrador
  - alimentar Y (para que esteja com os valores a serem armazenados no registrador)
  - PRESET para armazenar Y (supondo que em Y estão os valores que se deseja armazenar)
  - n SHIFTS, para produzir n bits (saídas) em série

- entrada serial e saída paralela
  - CLEAR para zerar o registrador
  - entrada de n bits em série, juntamente com n pulsos no sinal SHIFT
  - os n bits ficam disponíveis em Z

- entrada e saída paralelas
  - alimentar Y com os bits que se deseja armazenar
  - PRESET para armazenar Y (supondo que em Y estão os valores que se deseja armazenar)
  - os n bits ficam disponíveis em Z

Um registrador como o esquematizado acima pode ser realizado por um conjunto de n flip-flops. Cada flip-flop armazenaria 1 bit.

Exemplo: Uma realização de um registrador com entrada e saída seriais é mostrada na figura a seguir. Os flip-flops utilizados são do tipo RS *edge-triggered* na descida (ou seja um RS cujo valor de saída muda na descida do sinal de controle).



Note que o sinal que alimenta a entrada R é o complemento do sinal que alimenta S em todos os flip-flops. Portanto, esses correspondem aos flip-flops D que armazenam o valor de entrada ao pulso do sinal de controle. Quando há um pulso do clock, o bit e é armazenado no flip-flop mais à esquerda, o valor de cada flip-flop é armazenado no flip-flop a sua direita e o valor do flip-flop mais à direita é o bit de saída s.

Note que a mudança de estado dos flip-flops ocorre na transição de 1 para 0 do sinal que alimenta a entrada C. Portanto, a mudança de estado ocorre na transição de 0 para 1 do clock (uma vez que o sinal do clock é negado antes de alimentar C).

Registradores em outros modos de operação podem ser implementados usando flip-flops que possuem sinais de controle adicionais. Tipicamente, os flip-flops comerciais possuem, além da entrada para o sinal do *clock*, entradas para os sinais de controle CLEAR e PRESET. O primeiro faz $Q = 0$ enquanto o segundo faz $Q = 1$, independente dos outros sinais (obviamente parece não ter sentido ativar CLEAR e PRESET simultaneamente ...).

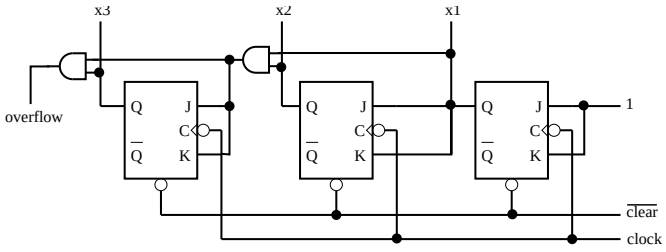
6.2.3 Contadores

O objetivo de um circuito contador é, a cada pulso do sinal de *clock*, incrementar (ou decrementar) o valor armazenado em alguma memória.

Um contador incremental módulo 2^n, com valor inicial 0, apresenta a seguinte seqüência de transição de valores:

$$0, 1, 2, 3, \ldots, 2^n - 1, 0, 1, 2, 3, \ldots$$

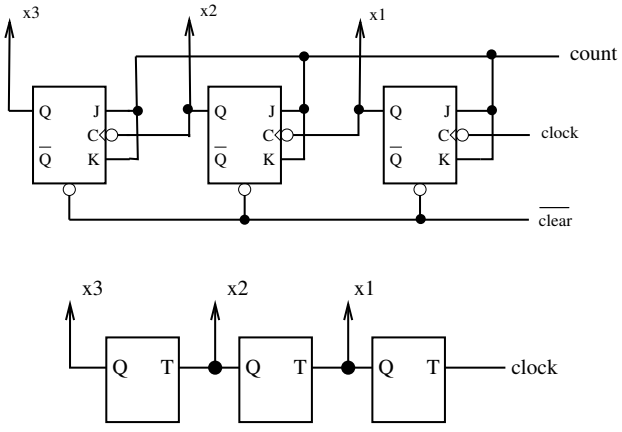
Uma possível implementação de um contador módulo 2^3 é mostrado na figura a seguir:



Suponha que incialmente todos os flip-flops estão em 0. A partir daí, a cada pulso no sinal do *clock*, o valor é incrementado em 1. Quando o circuito encontra-se no estado 111, o próximo estado será 000.

Como pode ser modificado o circuito acima para que ele seja um circuito contador incremental/decremental?

As seguintes figuras mostram circuitos contadores incrementais módulo 2^n assíncronos.

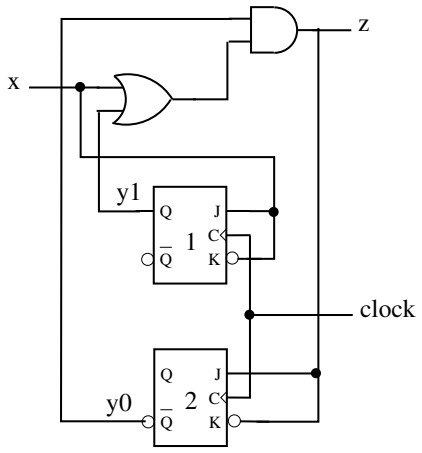


Em ambos os casos, apenas o primeiro flip-flop é alimentado pelo sinal do *clock*. Os demais flip-flops são alimentados pelas saídas dos flip-flops anteriores. Em ambos os casos, ocorrem estados transitórios da passagem de um estado para outro. Por exemplo, no segundo circuito, do estado 011 o circuito passa transitoriamente pelos estados 010 e 000 até ficar em 100 (que seria o estado seguinte ao estado 011. Portanto, circuitos combinacionais que possam fazer uso dos valores $x_3x_2x_1$ devem ser projetados de forma a evitar esses estados transitórios.

6.3 Análise de circuitos seqüenciais

A análise de circuitos seqüenciais consiste em, a partir do circuito, obter uma descrição funcional do mesmo. Tal descrição funcional pode ser obtida com o auxílio das equações de estado, as tabelas de estado e diagramas de estado. Estes conceitos serão apresentados a seguir através de um exemplo.

Considere o seguinte circuito:



a) Equação das entradas dos flip-flops

$J_1 = x$

$K_1 = \overline{x}$

$J_2 = z = (x + y_1)\,\overline{y}_0$

$K_2 = \overline{z} = \overline{(x + y_1)\,\overline{y}_0}$

b) Equação para os próximos estados:

(Lembre que $Q_{i+1} = J_i\,\overline{Q}_i + \overline{K}_i\,Q_i$)

$y_1^* = x\,\overline{y}_1 + x\,y_1 = x$

$y_0^* = [(x + y_1)\,\overline{y}_0]\,\overline{y}_0 + [(x + y_1)\,\overline{y}_0]y_0 = (x + y_1)\,\overline{y}_0$

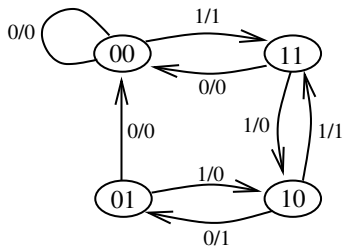
c) Tabela de estados

Cada célula da tabela representa o próximo estado ($y_1^*\,y_0^*$) e a saída (z). Ou seja, leia-se cada célula como $y_1^*\,y_0^*\,/\,z$.

Estado atual ($y_1\,y_0$)	x	
	0	1
00	00/1	11/1
01	00/0	10/0
10	01/1	11/1
11	00/0	10/0

d) Diagrama de estados

Cada nó representa um estado do sistema. Há uma aresta de um estado para outro se é possível uma transição de um para o outro. O rótulo nas arestas indica entrada x e saída z (leia-se $x\,/\,z$). Como há apenas uma variável de entrada, que pode tomar os valores ou 0 ou 1, então há exatamente 2 arestas que saem de cada nó.



Diagramas de estado são uma representação equivalente à tabela de estados.

6.4 Projeto de circuitos seqüenciais

Projeto de circuitos seqüenciais é um processo inverso ao da análise. No entanto, o ponto de partida em geral não é uma tabela ou diagrama de estados, e sim uma descrição funcional do circuito.

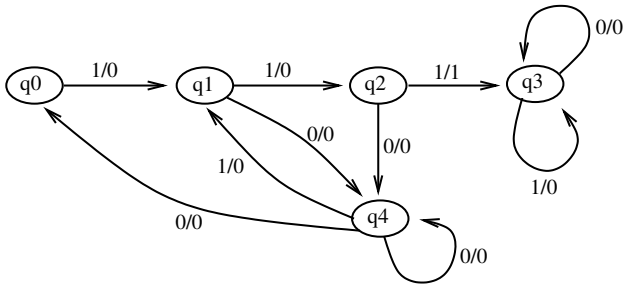
As etapas que fazem parte de projeto de circuitos seqüenciais são:

- Descrição funcional
- Tabela de estados (que pode ser obtida a partir do diagrama de estados ou não)
- Tabela minimal de estados
- Tabela de transição
- Equação das entradas dos flip-flops
- Circuito

Novamente, introduzimos esses conceitos através de um exemplo.

Exemplo: Detector de início de mensagem. Suponha uma linha x, sincronizada com o clock, que transmite sinais. Uma ocorrência de 3 bits 1 consecutivos é considerado início de mensagem. Desejamos projetar um circuito síncrono que detecta início de mensagem. Suponha que existe algum mecanismo que coloca o sistema detector de início de mensagem em um estado q_0 a cada final de mensagem e suponha que incialmente o sistema encontra-se no estado q_0.

a) Diagrama de estados



b) Tabela de estados

Estado	Entrada	
	0	1
q_0	$q_4/0$	$q_1/0$
q_1	$q_4/0$	$q_2/0$
q_2	$q_4/0$	$q_3/1$
q_3	$q_3/0$	$q_3/0$
q_4	$q_4/0$	$q_1/0$

c) Tabela minimal de estados Na tabela de estados acima, o estado q_0 é equivalente ao estado q_4. Isso poderia ser percebido até no próprio diagrama de estados. No entanto, em um caso genérico, nem sempre o diagrama de estados é gerado e, além disso, a equivalência de estados pode não ser tão óbvia. De qualquer forma, nesta etapa reduz-se a tabela de estados a uma tabela minimal, ou seja, eliminam-se os estados equivalentes. Para não gerar confusão na identificação dos estados, na tabela minimal de estados é aconselhável a utilização de outros nomes para os estados.

Estado	Entrada	
	0	1
a	$a/0$	$c/0$
b	$a/0$	$d/0$
c	$a/0$	$b/1$
d	$d/0$	$d/0$

d) Associação de estados

Se o número de estados na tabela minimal de estados é m, então serão necessários r flip-flops para armazenar qualquer um desses estados, onde r é tal que $2^{r-1} < m \leq 2^r$.

O problema de associação de estados consiste em definir qual das 2^r combinações de valores binários será utilizado para representar cada um dos estados do sistema. No exemplo que estamos considerando, como são 4 estados então são necessários $r = 2$ flip-flops e existem as três seguintes associações:

Estados	Associação		
	1	2	3
a	00	00	00
b	01	11	10
c	11	01	01
d	10	10	11

As demais associações são equivalentes a um desses três no sentido de que correspondem a uma rotação vertical ou à complementação de uma ou ambas as variáveis e, portanto, em termos de circuito resultante teriam o mesmo custo.

e) Tabelas de transição Para cada uma das associações consideradas, pode-se gerar uma tabela de transição. Em cada tabela de transição, as atribuições de estado estão listadas seguindo a ordem do gray-code $(00 - 01 - 11 - 10)$. Uma tabela de transição mostra qual será o próximo estado em função do estado e entrada atuais.

Associação 1

Estado	$y_1\, y_0$	$y_1^*\, y_0^*$	
		$x = 0$	$x = 1$
a	00	00	11
b	01	00	10
c	11	00	01
d	10	10	10

Associação 2

Estado	$y_1\, y_0$	$y_1^*\, y_0^*$	
		$x = 0$	$x = 1$
a	00	00	01
c	01	00	11
b	11	00	10
d	10	10	10

Associação 3

Estado	$y_1\, y_0$	$y_1^*\, y_0^*$	
		$x = 0$	$x = 1$
a	00	00	01
c	01	00	10
d	11	11	11
b	10	00	11

f) Equação das entradas dos flip-flops

A equação das entradas dos flip-flops pode ser gerada a partir da análise das tabelas de transições. Vejamos como se realiza esse processo para a associação 3 do item anterior.

Primeiramente, observe que sabemos que do estado $y_1\, y_0$ o circuito irá para o estado $y_1^*\, y_0^*$ e que cada variável de estado (no caso, y_1 e y_0) corresponde a um flip-flop. Assim, o que queremos descobrir é a expressão que descreve o sinal de entrada desses flip-flops para que a transição (mudança de estado) desejada ocorra.

Vamos analisar inicialmente o estado y_1. Restringindo a tabela de transição da associação 3 à variável y_1, temos:

y_1	y_1^*	
	$x=0$	$x=1$
a 0	0	0
c 0	0	1
d 1	1	1
b 1	0	1

Suponha que usaremos flip-flops JK neste circuito. Então, qual deve ser o valor de J e K para que a transição $y_1 \rightarrow y_1^*$ ocorra ? Para isso, recordemos a tabela do flip-flop JK. A tabela abaixo à esquerda descreve o comportamento do flip-flop JK e a tabela da direita mostra em quais condições ocorre a transição $Q \rightarrow Q^*$.

JK	Q^*	
	$Q=0$	$Q=1$
00	0	1
01	0	0
10	1	1
11	1	0

$Q \rightarrow Q$	J	K
$0 \rightarrow 0$	0	$\times$
$0 \rightarrow 1$	0	$\times$
$1 \rightarrow 0$	$\times$	1
$1 \rightarrow 1$	$\times$	0

Agora então podemos montar uma tabela para J e K, indicando os valores que produzirão a transição $y_1 \rightarrow y_1^*$.

y_1y_0	J_1	
	$x=0$	$x=1$
00	0	0
01	0	1
10	$\times$	$\times$
11	$\times$	$\times$

y_1y_0	K_1	
	$x=0$	$x=1$
00	$\times$	$\times$
01	$\times$	$\times$
11	0	0
10	1	0

Usando procedimento similar aos mapas de Karnaugh, da tabela à esquerda obtemos $J_1 = x\,y_0$ e da tabela à direita obtemos $K_1 = \overline{x}\,\overline{y}_0$.

De forma análoga, repetimos o processo para y_0. Restringindo a tabela de transição da associação 3 à variável y_0, temos:

y_0	y_0^*	
	$x=0$	$x=1$
a 0	0	1
c 1	0	0
d 1	1	1
b 0	0	1

Portanto, as tabelas para J_0 e K_0 serão respectivamente

y_1y_0	J_0	
	$x=0$	$x=1$
00	0	
01	$\times$	$\times$
11	$\times$	$\times$
10	0	1

y_1y_0	K_0	
	$x=0$	$x=1$
00	$\times$	$\times$
01	1	1
11	0	0
10	$\times$	$\times$

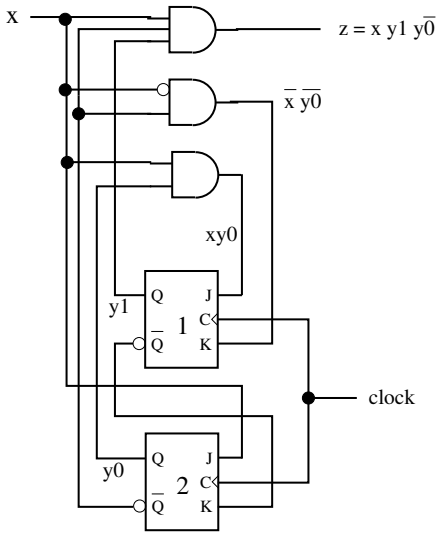
De onde obtemos que $J_0 = x$ e $K_0 = \overline{y}_1$.

Finalmente, a expressão para a saída z é dada por $x\, y_1\, \overline{y}_0$ (pois existe uma única situação em que a saída do circuito é 1; justamente quando ele se encontra no estado b e a entrada x é $= 1$. A expressão segue do fato de termos associado ao estado b o par $y_1 y_0 = 10$).

Procedimento similar pode ser aplicado para as associações 1 e 2. A associação 1 resulta em um circuito de custo (em termos de número total de portas lógicas) equivalente ao da associação 3 e a associação 2 resulta em um circuito de custi ligeiramente maior.

g) O circuito!

As equações obtidas para a associação 3 correspondem ao seguinte circuito.



Referências Bibliográficas

[Akers, 1978] Akers, S. B. (1978). Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6):509–516.

[Barrera and Salas, 1996] Barrera, J. and Salas, G. P. (1996). Set Operations on Closed Intervals and Their Applications to the Automatic Programming of Morphological Machines. *Electronic Imaging*, 5(3):335–352.

[Brace et al., 1990] Brace, K. S., Rudell, R. L., and Bryant, R. E. (1990). Efficient Implementation of a BDD Package. In *Proceedings of 27th ACM/IEEE Design Automation conference*, pages 40–45.

[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.

[Bryant, 1986] Bryant, R. E. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

[Coudert, 1994] Coudert, O. (1994). Two-level Logic Minimization: an Overview. *Integration, the VLSI Journal*, 17(2):97–140.

[Coudert, 1995] Coudert, O. (1995). Doing Two-level Minimization 100 Times Faster. In *Proc. of Symposium on Discrete Algorithms (SODA)*, San Francisco CA.

[Dougherty and Lotufo, 2003] Dougherty, E. R. and Lotufo, R. A. (2003). *Hands-on Morphological Image Processing*. SPIE Press.

[Filho, 1980] Filho, E. A. (1980). *Teoria Elementar dos Conjuntos*. Livraria Nobel S.A., São Paulo.

[Fišer and Hlavicka, 2003] Fišer, P. and Hlavicka, J. (2003). Boom - a heuristic boolean minimizer. *Computers and Informatics*, 22(1):19–51.

[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.

[Hill and Peterson, 1993] Hill, F. J. and Peterson, G. R. (1993). *Computer Aided Logical Design with Emphasis on VLSI*. John Wiley & Sons, fourth edition.

[Hlavicka and Fiser, 2001] Hlavicka, J. and Fiser, P. (2001). BOOM - A Heuristic Boolean Minimizer. In *Proc. of ICCAD*, pages 439–442.

[Jacobi, 1996] Jacobi, R. (1996). Síntese de Circuitos Lógicos Combinacionais. Décima Escola de Computação, Campinas.

[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.

[Mendelson, 1977] Mendelson, E. (1977). *Álgebra Booleana e Circuitos de Chaveamento*. Mcgraw-Hill.

[Micheli, 1994] Micheli, G. D. (1994). *Synthesis and Optimization of Digital Circuits*. McGraw-Hill.

[Nelson et al., 1995] Nelson, V. P., Nagle, H. T., Carroll, B. D., and Irwin, J. D. (1995). *Digital Logic Circuit Analysis and Design*. Prentice-Hall.

[Perkowski et al., 1995] Perkowski, M. A., Grygiel, S., and the Functional Decomposition Group (1995). A Survey of Literature on Function Decomposition - Version IV. Technical report, PSU Electrical Engineering Department.

[Ross and Wright, 1992] Ross, K. A. and Wright, C. R. B. (1992). *Discrete Mathematics*. Prentice Hall, 3rd edition.

[Soille, 1999] Soille, P. (1999). *Morphological Image Analysis*. Springer-Verlag, Berlin.

[Whitesitt, 1961] Whitesitt, J. E. (1961). *Boolean Algebra and its Applications*. Addison-Wesley.

[Zupan, 1997] Zupan, B. (1997). *Machine Learning Based on Function Decomposition*. PhD thesis, University of Ljubljana.