Exemplos de aplicação de álgebra booleana

Como já vimos, o conjunto $\mathcal{P}(S)$ juntamente com as operações de união, intersecção e complemento de conjuntos forma uma álgebra booleana. Cálculo proposicional é outro exemplo de álgebra booleana. Em ambos os casos vimos, por exemplo, a possibilidade de se simplificar expressões ou então verificar a equivalência de duas expressões. Embora os símbolos utilizados nas expressões sejam diferentes, as principais operações definidas em um têm similar no outro e as regras ou propriedades que valem em um valem também no outro e vice-versa. A álgebra booleana captura, de forma abstrata, a estrutura em comum entre ambos.

Quando falamos de expressões, elas envolvem elementos (geralmente denotados por símbolos do nosso alfabeto). Se pensarmos esses símbolos como incógnitas (variáveis que podem tomar como valor quaisquer dos elementos da álgebra booleana), podemos interpretar essas expressões como representação de funções. Se a expressão envolve n símbolos que tomam valor em uma álgebra booleana cujo conjunto de suporte é A, então a função é um mapeamento de A^n em A.

A seguir serão introduzidos dois exemplos que ilustram o uso da álgebra booleana em problemas práticos. Em particular, a ênfase nessas aplicações está justamente nos mapeamentos de A^n em A.

1.1 Circuitos Lógicos

Computadores representam e manipulam dados em binário. Um processamento qualquer no computador pode ser visto como uma enorme caixa preta na qual as entradas e saídas são bits (valores 0 ou 1). Imagine, por exemplo, a operação de adição em um computador. Suponha que $\mathbf{a} = a_3\, a_2\, a_1\, a_0$ e $\mathbf{b} = b_3\, b_2\, b_1\, b_0$ são dois números de 4 bits a serem somados, nos quais o índice 0 e 3 representam, respectivamente, os bits menos e mais significativo.

Podemos realizar a adição desses números da mesma forma que realizamos a adição na aritmética usual onde os números são representados na base 10. Devemos apenas tomar cuidado com o fato de que no sistema binário temos apenas dois dígitos possíveis, o 0 e o 1.

Se soubermos como proceder com a adição para uma coluna de bits (dígitos a_i, b_i e um eventual vai-um c_i vindo da coluna anterior), então a operação de adição pode ser realizada conforme especificada no diagrama da figura 1.1. Cada caixa no diagrama corresponde a um somador de bits. Ao se encadear quatro deles, obtém-se um somador de números de quatro bits.

Visto como uma caixa-preta, o somador apresentado na figura 1.1 possui como entradas os 8 bits (4 de cada um dos números a serem somados) e um zero que corresponde ao vai-um inicial, e possui como

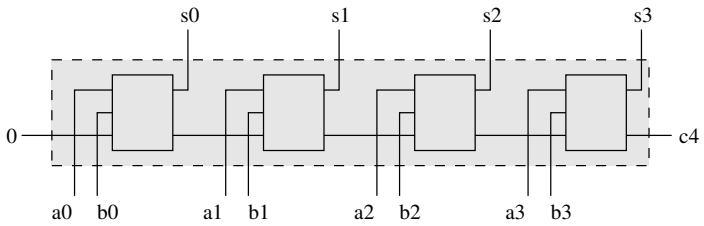

Figura 1.1: Esquema de um somador de 4 bits.

saídas os bits soma do resultado $\mathbf{s} = s_3\, s_2\, s_1\, s_0$ e um eventual vai-um para a quinta coluna. Não é difícil interpretar essa caixa-preta como uma função; para cada possível combinação de valores de entrada, ela gera uma saída bem definida. Similarmente, as caixas-pretas menores (ou seja, os somadores de bits) podem também ser interpretados como funções. No caso, eles possuem três entradas e duas saídas. O seu funcionamento, baseado em nosso conhecimento de aritmética binária, pode ser descrito pela seguinte tabela-verdade.

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

É possível expressarmos s_i e c_{i+1} em termos de a_i, b_i e c_i ? Observe que, no caso do cálculo proposicional, nós aprendemos como gerar tabelas-verdade a partir de uma expressão. Aqui temos um problema inverso: dada uma tabela-verdade, qual é a expressão que a gerou ? Sempre existe tal expressão? Ela é única?

Uma vez que os únicos valores que as variáveis podem tomar são 0 ou 1, a igualdade $\overline{a}_i\, \overline{b}_i\, c_i = 1$, por exemplo, só ocorre se e somente se $a_i = 0$, $b_i = 0$ e $c_i = 1$. Assim, podemos escrever

$$s_i = \overline{a}_i\, \overline{b}_i\, c_i + \overline{a}_i\, b_i\, \overline{c}_i + a_i\, \overline{b}_i\, \overline{c}_i + a_i\, b_i\, c_i$$

e

$$c_{i+1} = \overline{a}_i\, b_i\, c_i + a_i\, \overline{b}_i\, c_i + a_i\, b_i\, \overline{c}_i + a_i\, b_i\, c_i$$

Em problemas reais, desejamos realizar (implementar) essas funções fisicamente. Considere dispositivos como os mostrados na figura 1.2, que recebem sinais de entrada à esquerda e produzem sinais de saída à direita.

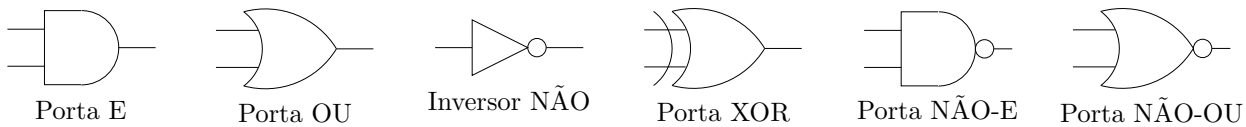

Figura 1.2: Representação gráfica de algumas portas lógicas.

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 1.3 mostra um circuito com três inversores e uma porta E.

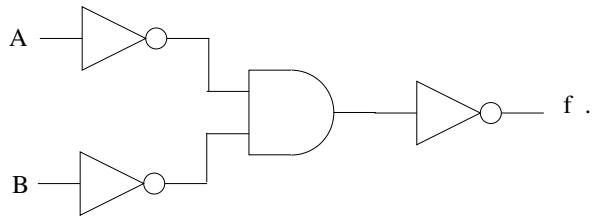

Figura 1.3: Um circuito simples.

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.

Retomando o exemplo do somador, como seria a realização das funções s_i e c_i? Note que, se utilizarmos dispositivos como os mostrados na figura 1.2, então cada termo produto pode ser realizado com duas portas E (pois todos eles envolvem duas operações de produto) e cada operação de adição pode ser realizado por uma porta OU. Além disso, são necessários vários inversores.

Uma pergunta natural é se existe uma realização "mais econômica", ou seja um circuito equivalente que utiliza um menor número de dispositivos. Uma abordagem para se obter tal realização pode ser baseada na simplificação das expressões originais; simplificação no sentido de obter expressões equivalentes com menor número de operações e símbolos que aparecem na expressão.

Uma possível solução que utiliza apenas 4 dispositivos é mostrada na figura 1.4. A verificação de que o circuito é realmente uma realização das funções s_i e c_{i+1} será deixada como exercício.

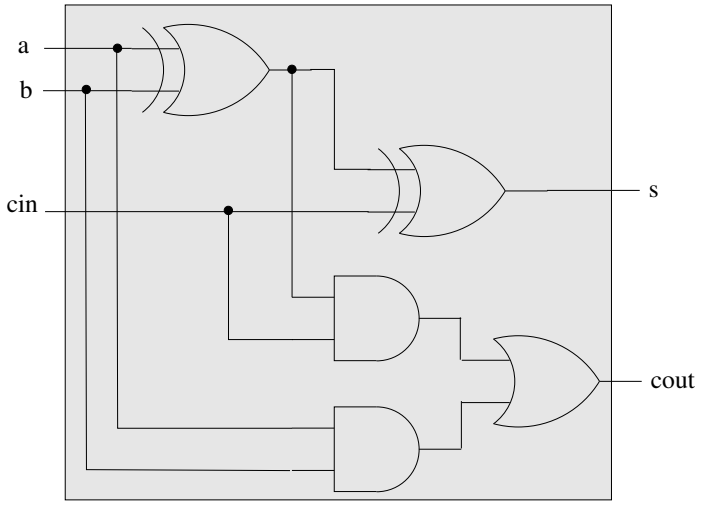

Figura 1.4: Esquema de um somador de bits.

Podemos agora listar algumas questões interessantes relacionadas ao projeto de circuitos lógicos:

- 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?

Espera-se que essas perguntas sejam respondidas ao longo do semestre.

1.2 Processamento de imagens binárias

Seja $E = Z^2$ o plano dos números inteiros. Uma função $f : E \to \{0, 1, 2, \ldots, k-1\}$ 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).

1.2.1 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, $\forall x \in E$,

$$1_S(x) = 1 \iff x \in S\,. \tag{1.1}$$

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.

1.2.2 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.

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 1.5 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.

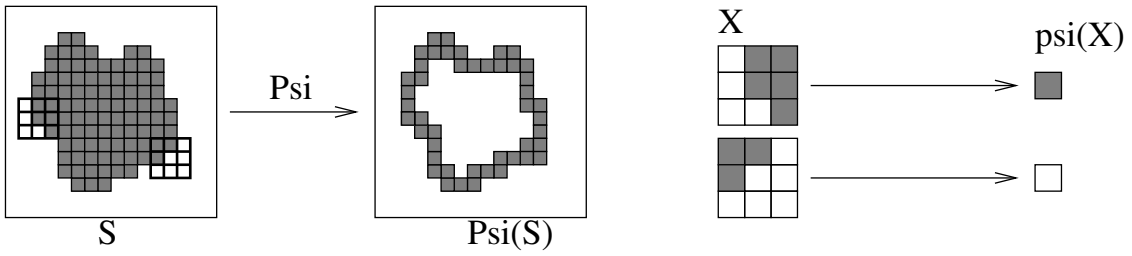

Figura 1.5: Ilustração de um operador localmente definido.

1.2.3 Como projetar um operador de imagens binárias

Em problemas de processamento de imagens desejamos transformar imagens de forma a obter efeitos de interesse como eliminação de ruído, correção de distorções, segmentação de certos objetos. etc.

Muitos desses efeitos podem ser obtidos com operadores localmente defindos, descritos acima. Por exemplo, suponha que desejamos encontrar os contornos (internos) de um objeto. Na figura 1.6 mostra um exemplo.

Figura 1.6: Imagem de entrada e imagem com o respectivo contorno.

Para determinar o operador que realiza tal mapeamento, podemos construir uma tabela-verdade, cuja coluna da esquerda contém todos os possíveis padrões observáveis dentro de uma janela 3×3 e a coluna da direita contém o valor desejado como saída (esse valor será 1 se o ponto correspondente ao centro da janela é um ponto que corresponde a contorno e 0 caso contrário). Ao se aplicar a função correspondente a essa tabela em todos os pontos da imagem, obtém-se como resultado a imagem com os contornos.

A abordagem descrita pressupõe que o projetista conhece o comportamento do operador. Na maioria dos casos, no entanto, a realidade é outra. Por exemplo, o problema de reconhecer caracteres "a" em uma imagem de uma página de livro demandaria um trabalho demasiadamente árduo se fosse realizada da forma descrita acima. Isso pode ser contornado utilizando-se técnicas de aprendizado computacional: exemplos de imagens entrada-saída são preparados manualmente e alimentam um sistema computacional que se encarregará da tarefa de construir a tal tabela-verdade. O desafio do sistema computacional, no entanto, não termina aqui. Em geral, a tabela-verdade construída assim é incompleta e também pode conter contradições (ou seja, um padrão sendo mapeado para 0 e para 1). Resolver essas contradições e também definir os valores de saída para os padrões que não foram observados constituem alvo de estudo de técnicas de aprendizado computacional.

No contexto de processamento de imagens binárias, algumas questões interessantes são:

- qual deve ser o tamanho da janela W (ou seja, quantas variáveis devem ser consideradas na função booleana)?

- uma vez determinado o operador (função booleana), qual é a representação que permite avaliação do seu valor para uma determinada entrada?
- uma seqüência de processamentos pode ser reduzida a um único processamento e vice-versa?

Essas perguntas não serão respondidas durante o semestre. Aqueles que tiverem interesse pelo assunto podem falar com a autora deste texto.

Referências Bibliográficas

[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.

[Dougherty and Lotufo, 2003] Dougherty, E. R. and Lotufo, R. A. (2003). *Hands-on Morphological Image Processing*. SPIE Press.

[Soille, 1999] Soille, P. (1999). *Morphological Image Analysis*. Springer-Verlag, Berlin.

Última revisão em 22 de março de 2007.