USP/IME/Doutorado em Ciência da Computação
MAC-5701 Tópicos em Ciência da Computação

Diagramas de Decisão Binária

Levantamento Bibliográfico

RELATÓRIO DE TRABALHO

Autor: Heraldo Maciel França Madeira
Orientador: Prof. Dr. Junior Barrera

July 16, 2000

Chapter 1

Introdução

O objetivo deste estudo é produzir um levantamento das principais estruturas de dados utilizadas para representar funções booleanas, com especial atenção aos diagramas de decisão binária (BDD).

As estruturas do tipo BDD, especialmente o ROBDD (BDD ordenado e reduzido) têm-se mostrado muito interessantes para representar uma importante classe de operadores de conjuntos no contexto de Morfologia Matemática. Por conseguinte, BDDs têm aplicação direta em Processamento de Imagens Binárias: operadores invariantes por translação e localmente definidos podem ser descritos como uma função booleana cujas variáveis são os elementos (pixels) de uma janela de interesse.

Para a construção automática de operadores (e.g. por aprendizado PAC) tem-se uma situação onde apenas uma pequena fração do espaço booleano é conhecida, sendo o restante considerado como *não-especificações* (ou *dont't cares*). Neste estudo veremos também como se comportam as BDDs neste ambiente (das funções parciais).

Outra meta deste estudo é apresentar introdutoriamente a extensão dos diagramas de decisão para funções discretas.

A aplicação de diagramas de decisão em Morfologia Matemática é a motivação deste estudo. No entanto esta teoria não será abordada neste trabalho. Para uma introdução em língua portuguesa vide [1].

Chapter 2

Representação de Funções Booleanas

O comportamento dos circuitos digitais, das operações entre subconjuntos e da lógica básica do pensamento podem ser razoavelmente modelados pela Álgebra Booleana, teoria introduzida em 1849 pelo pensador inglês George Boole. Augustus De Morgan, contemporâneo de Boole, contribuiu muito para o desenvolvimento da teoria. Em 1938, Claude E. Shannon, pesquisador da Bell Labs, pioneiramente utilizou a álgebra booleana para analisar e prever o comportamento de circuitos de chaveamento por relés [2]. Neste capítulo estudaremos as estruturas clássicas utilizadas para representar funções booleanas.

2.1 Álgebra Booleana

Uma *álgebra booleana* (ou *reticulado booleano*) é um *reticulado limitado, distributivo e complementado* [3]. É usualmente denotada por $\mathcal{B} = (S, \vee, \wedge, \bar{\cdot}, O, I)$, onde S é um conjunto, $\vee$ e $\wedge$ são operações binárias em S, $\bar{\cdot}$ é uma operação unária em S, e O e I são os limitantes universais inferior e superior de S. Assim, um conjunto S forma uma álgebra booleana se as operações $\wedge$, $\vee$ e $\bar{\cdot}$ em S satisfazem as propriedades expostas na Tabela 2.1, para quaisquer $a, b, c \in S$. As operações são chamdas de disjunção ($\vee$), conjunção ($\wedge$) e complementação ($\bar{\cdot}$).

As propriedades de dualização são usualmente chamadas de *leis de De Morgan*. O princípio da dualidade estabelece que qualquer proposição válida da álgebra booleana assim permanece se trocarmos todas as ocorrências de

$a \wedge a = a$	$a \vee a = a$	(idempotência)
$a \wedge b = b \wedge a$	$a \vee b = b \vee a$	(comutatividade)
$a \wedge (b \wedge c) = (a \wedge b) \wedge c$	$a \vee (b \vee c) = (a \vee b) \vee c$	(associatividade)
$a \vee (a \wedge b) = a$	$a \wedge (a \vee b) = a$	(absorção)
$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$	(distributividade)
$a \wedge O = O$	$a \vee O = a$	(lim. univ. inferior)
$a \wedge I = a$	$a \vee I = I$	(lim. univ. superior)
$a \wedge \overline{a} = O$	$a \vee \overline{a} = I$	(complementariedade)
$\overline{I} = O$		
$\overline{(\overline{a})} = a$		(involução)
$\overline{(a \wedge b)} = \overline{a} \vee \overline{b}$	$\overline{(a \vee b)} = \overline{a} \wedge \overline{b}$	(dualização)

Table 2.1: Propriedades da Álgebra Booleana

$\wedge$, $\vee$, O, I, $\leq$, $\geq$ por $\vee$, $\wedge$, I, O, $\geq$, $\leq$, respectivamente. A ordem parcial $\leq$ que existe entre elementos de S satisfaz a propriedade $a \leq b \Leftrightarrow \overline{a} \vee b = I \Leftrightarrow a \wedge \overline{b} = O \Leftrightarrow a \vee b = b \Leftrightarrow a \wedge b = a$. (Na realidade, as operações $\vee$ e $\wedge$ são definidas a partir de $\leq$).

Os sucessores imediatos de O são chamados de *átomos* ou *mintermos* de $\mathcal{B}$. Os n mintermos são suficientes para se determinar, com o uso das operações, qualquer elemento de S. Por este motivo também denota-se por $\mathcal{B}_n$ uma álgebra booleana [3]. (Note, porém, que outros autores denotam por $\mathcal{B}_n$ o conjunto das funções $f : \{0,1\}^n \rightarrow \{0,1\}$). Os n predecessores imediatos de I são os *maxtermos* de $\mathcal{B}_n$. Por dualidade, eles são suficientes para se obter, por aplicações sucessivas das operações, todo o conjunto S. Os mintermos não são mutuamente comparáveis por $\leq$, tampouco os maxtermos.

Uma álgebra booleana $\mathcal{B}_n$ possui 2^n elementos, e o diagrama de Hasse que a representa é um n-cubo (hipercubo de dimensão n). A figura 2.1 mostra o diagrama para reticulados booleanos de dimensões 2, 3 e 4, e seus mintermos m_i e maxtermos M_i.

Na seqüência temos exemplos de álgebras booleanas muito estudadas:

- A *álgebra de chaveamento* $\mathcal{C} =(\{0,1\}, +, \cdot, \bar{\cdot}, 0, 1)$, onde as operações são definidas por: $\begin{matrix} 0+0=0 & 1+0=1 \\ 0+1=1 & 1+1=1 \end{matrix}$, $\begin{matrix} 0\cdot 0=0 & 1\cdot 0=0 \\ 0\cdot 1=0 & 1\cdot 1=1 \end{matrix}$ e $\begin{matrix} \overline{0}=1 \\ \overline{1}=0 \end{matrix}$, é uma álgebra booleana, e a relação $\leq$ é a usual em {0,1}.

- A extensão da álgebra de chaveamento para n-uplas (pontos no *espaço booleano* n-dimensional) é tambem uma álgebra booleana. Ela é deno-

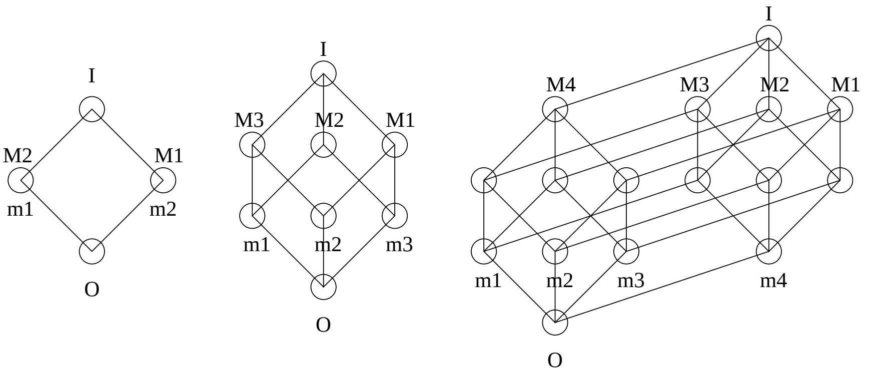


Figure 2.1: Álgebras booleanas de dimensões 2, 3 e 4

tada por $\mathcal{C}^n = (\{0,1\}^n, +, \cdot, \bar{\cdot}, \mathbf{0}, \mathbf{1})$, onde as operações são as aplicações bit-a-bit (componente-a-componente) das operações definidas acima. Do mesmo modo, $\leq$ é estendida para n bits: $a \leq b \Leftrightarrow a \cdot b = a, \forall a, b \in \{0,1\}^n$.

- A coleção $\mathcal{P}(W)$ de todos os subconjuntos de um conjunto W, associada às operações de união, interseção e complementação também forma uma álgebra booleana, denotada por $(\mathcal{P}(W), \cup, \cap, \cdot^c, \emptyset, W)$. A ordem parcial entre elementos de $\mathcal{P}(W)$ é a relação de inclusão de conjuntos usual ($\subseteq$).

Dois reticulados $(L_1, \leq)$ e $(L_2, \prec)$ são ditos *isomórficos* se há uma relação biunívoca $\Phi : L_1 \rightarrow L_2$ entre os elementos de L_1 e os de L_2 que preserva a relação de ordem parcial entre eles, isto é, $\forall x, y \in L_1$, $\Phi(x) \prec \Phi(y) \Leftrightarrow x \leq y$. Tal mapeamento Φ é um *isomorfismo de reticulado* e preserva também as operações $\wedge$ e $\vee$, isto é, $\Phi(x) \wedge \Phi(y) = \Phi(x \wedge y)$, $\Phi(x) \vee \Phi(y) = \Phi(x \vee y)$. Se aplicável (reticulado complementado), Φ preserva também o complemento: $\Phi(\overline{x}) = \overline{\Phi(x)}$.

O seguinte teorema relaciona uma álgebra booleana finita qualquer com a álgebra booleana do conjunto $\mathcal{P}(W)$ das partes de um conjunto W.

Teorema da representação, de Stone: *Seja W o conjunto dos átomos de uma álgebra booleana finita $\mathcal{B}$. Então $\mathcal{B}$ é isomórfica à algebra $(\mathcal{P}(W)$, $\cup$, $\cap$, $\cdot^c$, $\emptyset$, $W)$.*

Como qualquer conjunto finito é univocamente descrito por uma reunião de elementos, pelo isomorfismo mencionado acima chega-se ao *teorema da representação canônica disjuntiva*: *Cada elemento a de uma álgebra booleana $\mathcal{B}_n$ possui uma única representação por uma disjunção* ($\vee$) *de mintermos m_j de $\mathcal{B}_n$. Tais mintermos são os que satisfazem $m_j \leq a$.*

Por dualidade, temos o *teorema da representação canônica conjuntiva*: *Cada elemento a de uma álgebra booleana $\mathcal{B}_n$ possui uma única representação por uma conjunção* ($\wedge$) *de maxtermos M_j de $\mathcal{B}_n$. Tais maxtermos são os que satisfazem $a \leq M_j$.*

2.2 Expressões Booleanas

As operaçẽs $\wedge$, $\vee$ e $\bar{\cdot}$ permitem-nos obter um elemento de $\mathcal{B}_n$ a partir de outros elementos de $\mathcal{B}_n$. Ademais, as propriedades da Tabela 2.1 mostram que há diversas formas de se aplicar as operações para se chegar a um elemento. Se indicarmos por $x_1, x_2, \ldots, x_k$ os elementos de $\mathcal{B}_n$ que serão usados para obter um elemento y, então podemos descrever y por uma seqüência de operações aplicadas sobre os x_i. Chamamos esta descrição de *expressão booleana*, e a denotamos por $E(x_1, x_2, \ldots, x_k)$, onde os x_i são chamados de *literais*.

Formalmente, uma expressão booleana é definida indutivamente como se segue:

- Literais são expressões booleanas.
- Disjunção ($\vee$), conjunção ($\wedge$), e complementação ($\bar{\cdot}$) de expressões booleanas são expressões booleanas.
- Toda expressão booleana pode ser gerada por um número finito de aplicações dos itens anteriores.

A avaliação da expressão é feita identificando-se cada literal com um elemento de $\mathcal{B}_n$ e calculando passo-a-passo as conjunções, disjunções e complementações indicadas. No entanto, antes de se avaliar a expressão, que também representa um elemento de $\mathcal{B}_n$, esta pode ser manipulada através das propriedades da Tabela 2.1 a fim de se chegar a expressões mais simples. Uma forma simples é a *forma normal disjuntiva*, também chamada de *soma de produtos*. Ela é obtida (1) aplicando-se as leis de De Morgan a fim de transferir as complementações para as literais; (2) aplicando-se em seguida a propriedade distributiva ($a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$) sempre que cabível; e, finalmente, (3) aplicando-se as propriedades de idempotência para eliminar subexpressões repetidas.

Um *produto fundamental* de k literais é uma conjunção na qual cada literal x_i aparece exatamente uma vez, complementada ou não, para $i = 1, 2, \ldots, k$.

Uma conjunção em que faltam as literais $x_{j_1}, \ldots, x_{j_m}$ pode ser transformada em uma disjunção de produtos fundamentais multiplicando-a (*i.e.*, aplicando $\wedge$) por $(x_{j_1} \vee \overline{x_{j_1}}) \wedge \ldots \wedge (x_{j_m} \vee \overline{x_{j_m}})$, e em seguida aplicando-se as propriedades distributiva e de idempotência exatamente como descrito acima. Logo, uma expressão $E(x_1, \ldots, x_k)$ qualquer pode ser transformada em uma disjunção de produtos fundamentais das literais $x_1, \ldots, x_k$.

Há 2^k produtos fundamentais de k literais. Ademais, eles são os átomos de uma álgebra booleana $\mathcal{B}_{2^k}$, que contém 2^{2^k} elementos. $\mathcal{B}_{2^k}$ é chamada de *álgebra booleana livre* com k geradores $x_1, \ldots, x_k$. A álgebra $\mathcal{B}_n$, à qual pertencem os elementos representados pelas literais $x_1, \ldots, x_k$ e o elemento descrito por $E(x_1, \ldots, x_k)$, é chamada de álgebra base (ground algebra) [3].

Pelo teorema da representação canônica disjuntiva (Seção 2.1) e pelo fato de toda disjunção de produtos fundamentais das literais $x_1, \ldots, x_k$ ser uma expressão booleana em $x_1, \ldots, x_k$, tem-se que os elementos de $\mathcal{B}_{2^k}$ representam todas as expressões booleanas nessas literais, e que a correspondente disjunção de produtos fundamentais é única. Tal forma é chamada de *forma canônica disjuntiva*.

Duas expressões booleanas são ditas *equivalentes* se correspondem a um mesmo elemento de $\mathcal{B}_{2^k}$, ou seja, se elas podem ser transformadas na mesma expressão canônica. Logo, há 2^{2^k} classes de equivalência distintas de expressões booleanas nas literais $x_1, \ldots, x_k$.

2.3 Funções Booleanas

Dada uma álgebra booleana $\mathcal{B}_n = (A_n, \vee, \wedge, \bar{\cdot}, O, I)$, uma expressão booleana $E(x_1, \ldots, x_k)$ define uma *função booleana de k variáveis sobre $\mathcal{B}_n$*, $\varphi : A_n^k \to A_n$, de acordo com a regra:

$$\varphi(a_1, \ldots, a_k) = E(a_1, \ldots, a_k), \quad \forall (a_1, \ldots, a_k) \in A_n^k.$$

Dizemos que a expressão E descreve a função φ. Então uma função booleana de k variáveis é aquela que é descrita por uma expressão booleana em k literais.

Nem todas funções $\varphi : A_n^k \to A_n$ podem ser descritas por expressões $E(x_1, \ldots, x_k)$. Por exemplo, seja $A_2 = \{O, a_1, a_2, I\}$. Nenhuma das quatro expressões em uma literal $(O, x, \overline{x}, I)$ descreve a função $\varphi : A_2^1 \to A_2$ dada por $[\varphi(O), \varphi(a_1), \varphi(a_2), \varphi(I)] = [O, O, I, a_1]$. De fato, apenas 2^{2^k} funções $\varphi : A_n^k \to A_n$, dentre as $2^{n2^{nk}}$ possíveis são funções booleanas de k variáveis. Para o exemplo acima, as únicas funções booleanas de 1 variável sobre $\mathcal{B}_2$ são: $[O, O, O, O]$, $[O, a_1, a_2, I]$, $[I, a_2, a_1, O]$ e $[I, I, I, I]$.

Toda função $\varphi : \{0,1\}^N \to \{0,1\}$ é uma função booleana em N variáveis. Devido à larga aplicabilidade destas funções na teoria de redes de chaveamento (circuitos digitais) elas são chamadas de *funções de chaveamento* de N argumentos.

Seja $a \in \{0,1\}$ e x uma literal. Define-se a *exponenciação booleana* pela operação dada por $x^a = x$, se $a = 1$, e $x^a = \overline{x}$, se $a = 0$. Dada uma N-upla $(a_1, a_2, \ldots, a_N)$, o produto fundamental $x_1^{a_1} \wedge x_2^{a_2} \wedge \ldots \wedge x_N^{a_N}$ aparece em uma expressão booleana canônica que representa uma função $\varphi : \{0,1\}^N \to \{0,1\}$ se e somente se $\varphi(a_1, a_2, \ldots, a_N) = 1$.

Prova-se [3] que a álgebra booleana $\mathcal{F}_n^{(N)}$ de todas as funções booleanas de N variáveis sobre uma álgebra $\mathcal{B}_n$ é isomórfica à álgebra booleana livre com N geradores, que por sua vez é isomórfica à álgebra das funções de chaveamento de N variáveis. Por este motivo os termos função booleana e função de chaveamento são usados indistintamente.

Daqui em diante adotaremos a terminologia usada em Álgebra de Chaveamento: O termo *função booleana φ de N variáveis* designará uma função de chaveamento $\varphi : \{0,1\}^N \to \{0,1\}$. O termo *literal* designará uma variável (uma coordenada do espaço booleano N-dimensional) complementada ou não. Expressões booleanas serão formadas de acordo com a álgebra $\mathcal{C} = (\{0,1\}, +, \cdot, \bar{}, 0, 1)$. Mintermos passam a ser entendidos como átomos de $\mathcal{F}_1^{(N)}$, ou seja, como produtos (conjunções) de N literais (de variáveis distintas), e maxtermos como somas (disjunções) dessas literais. A forma canônica disjuntiva é então uma *soma de mintermos* e a sua dual um *produto de maxtermos*.

2.4 Representações

Vimos na seção anterior que as funções booleanas são representáveis por expressões booleanas. Em diversas aplicações esta é a forma mais usual de se fornecer uma descrição de tais funções, pois é a forma natural de expressarmos o comportamento lógico das coisas. Felizmente, grande parte das funções úteis têm uma expressão compacta. Por outro lado, há inúmeras expressões equivalentes para cada função, o que significa que tarefas como a verificação de igualdade entre duas funções booleanas não são fáceis. Expressões canônicas resolvem esta questão, mas não são compactas. Estas questões abrem duas áreas de estudo dentro da teoria booleana: (1) buscar novas representações para funções booleanas e (2) encontrar expressões equivalentes que sejam compactas (minimização de expressões).

Trazendo o paradigma de modelagem apresentado em [4] para o contexto das funções booleanas, os esquemas de representação que buscamos para elas devem, na medida do possível, ser

- (a) *válidos*, isto é, uma representação não pode descrever algo que não seja uma função booleana;
- (b) *não-ambíguos*: duas funções diferentes deverão ter representações diferentes. Um esquema não-ambíguo é também chamado de *completo*, pois qualquer propriedade da função pode ser extraída de sua representação;
- (c) *únicos*, ou seja, duas representações diferentes devem descrever funções diferentes.
- (d) *expressivos*: todas as funções de interesse devem ser representáveis.
- (e) *concisos*: as representações de interesse devem ser compactas, a fim de facilitar o armazenamento e o processamento, ou sua implementação em *hardware*. Redundâncias podem ser usadas para facilitar a verificação de erros ou representar informações difíceis de computar, mas por outro lado aumentam a chance de produzir inconsistências.
- (f) *eficazes*: o esquema de representação deve permitir o uso de algoritmos eficazes para computar as propriedades das funções, tais como o conjunto característico, a cobertura irredundante ou o próprio valor da função para uma N-upla de entrada.
- (g) *precisos*: as representações devem descrever com exatidão a função. A representação compacta de funções de muitas variáveis só é viável relaxando-se este requisito, usando para isto métodos probabilísticos.
- (h) *fechados sob as operações*: algoritmos que efetuam operações entre representações (por exemplo, complementação, composição de funções) devem produzir representações válidas.

- (i) *fáceis de usar*: A interação homem-máquina exigida tem que ser facilitada: tabelas, expressões e grafos extensos ou criptográficos são indesejáveis. Se esta é a forma de representação adotada, então ferramentas de manipulação, tais como programas para construção incremental da função, devem estar disponíveis.

Um desafio imediato que surge destas considerações é o da conversão entre esquemas diferentes de representação. Representações em esquemas válidos, não ambíguos e únicos são chamadas de *canônicas*.

Além da representação por expressões booleanas, que podem ser genéricas ou estarem em formas normais, ou ainda em formas canônicas, funções booleanas podem ser representadas de diversas outras maneiras. Assumiremos que a função a ser representada possui N variáveis.

A representação por *tabela-verdade* consiste em uma tabela de 2^N linhas por $N+1$ colunas de *bits* (dígitos binários). As primeiras N colunas representam as variáveis de entrada (argumentos) da função. Em cada linha está uma combinação distinta de variáveis. A última coluna representa o valor da função correspondente à combinação de entradas. Comumente, assume-se que as entradas estão ordenadas pela contagem crescente dos números binários descritos pelas N colunas de entrada. Neste caso a coluna de saída é suficiente para descrever a função, e a representação passa a ser chamada de *vetor-verdade*. A Figura 2.2-a ilustra a tabela-verdade para a função booleana $f(A,B,C) = A\overline{B} + BC$. O vetor-verdade correspondente é $(0,0,0,1,1,1,0,1)$.

A representação por *mapas de Karnaugh* (Figura 2.2-b) consiste em uma tabela de $2^{[\frac{N}{2}]}$ linhas por $2^{[\frac{N}{2}]}$ colunas. As linhas representam as combinações de $[\frac{N}{2}]$ variáveis, a as colunas as combinações das demais. Cada posição (*célula*) da tabela está, portanto, associada a uma combinação distinta das N variáveis de entrada, e contém o valor correspondente da função. Tanto linhas quanto colunas são rotuladas de acordo com o código de Gray das respectivas variáveis. Isto significa que células vizinhas correspondem a pontos no espaço booleano que diferem apenas em uma coordenada. Quando a quantidade de variáveis for pequena (≤ 6), Mapas de Karnaugh são eficazes para se encontrar visualmente expressões booleanas na forma de soma de produtos que sejam mínimas.

Tabelas-verdade, vetores-verdade e mapas de Karnaugh são canônicos, mas apresentam como principal desvantagem o tamanho, que é exponencial em N. Cada elemento valendo 1 em tais tabelas corresponde a um mintermo na expressão canônica disjuntiva da função, e cada elemento em 0 a um

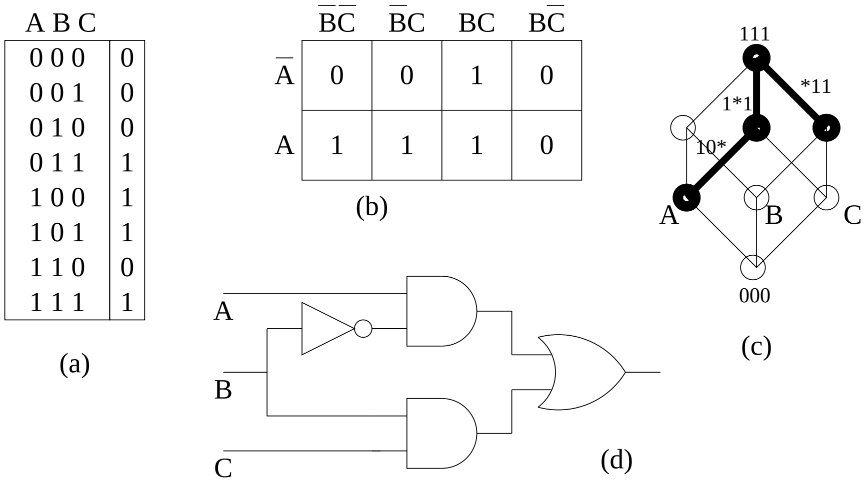


Figure 2.2: Representações para $f(A,B,C) = A\overline{B} + BC$.

maxtermo na expressão canônica conjuntiva.

Listas de mintermos (e listas de maxtermos) consistem em simplesmente listar as N-uplas que correspondem a 1's (e 0's) no vetor-verdade. São formas canônicas de representação que podem ser adequadas para certos problemas em que o vetor-verdade é esparso. Têm, entretanto, tamanho $N \cdot 2^N$ nos piores casos (funções 1 e 0, respectivamente). Por exemplo, a lista de mintermos de $f(A,B,C) = A\overline{B} + BC$ é dada por $(m_{011}, m_{100}, m_{101}, m_{111})$. A lista de maxtermos correspondente é dada por $(M_{000}, M_{001}, M_{010}, M_{110})$.

As representações por tabelas-verdade, vetores-verdade, mapas de Karnaugh, listas de mintermos e listas de maxtermos são facilmente traduzidas de e para expressões na forma canônica. Expressões na forma normal, por sua vez, estão diretamente relacionadas às representações por cubos, descritas a seguir.

Um *k-cubo* no espaço N-dimensional é uma função booleana que pode ser expressa por uma conjunção de $N-k$ literais em variáveis distintas (um mintermo, portanto, é um 0-cubo). k é a dimensão do cubo. Um cubo é completamente especificado por dois mintermos: o menor, em que as k variáveis ausentes na conjunção mencionada acima aparecem complementadas; e o maior, onde elas aparecem não complementadas. Um cubo C é *implicante* de uma função f de N variáveis se $C \leq f$. Uma *cobertura* de f

é um conjunto de implicantes tais que sua disjunção é f.

Um cubo C é um *implicante primo* de f se não existe outro implicante c de f tal que $C \leq c$. A coleção de todos os implicantes primos de f é chamada de *cobertura prima* de f. A *representação cúbica minimal* [5] de uma função booleana f consiste na sua cobertura prima, que é canônica. Contudo, o número de implicantes primos é $O(3^n/n)$ [6], e muitos deles podem ser redundantes, isto é, f pode ser coberta por um subconjunto da cobertura prima. Portanto, podem-se definir outras representações mais compactas com implicantes primos. As arestas enfatizadas na Figura 2.2-c, dada pelos cubos (10*) = [100,101], (1*1) = [101,111] e (*11) = [011,111], formam a cobertura prima da função $f(A, B, C) = A\overline{B} + BC$.

Representações cúbicas disjuntas de uma função booleana consistem em coberturas tais que a conjunção de quaisquer dois de seus cubos seja 0 (ou seja, eles não têm mintermos em comum). Sob certas condições podem ser canônicas. A lista de mintermos é uma representação desse tipo.

Redes booleanas ou *circuitos lógicos combinacionais* são representações gráficas de funções booleanas bastante adequados para projetar dispositivos físicos que as implemente. Cada operação booleana básica $(+, \cdot, \bar{\cdot})$ é representada por uma *porta lógica*. O circuito é um grafo acíclico dirigido construído pela interligação de portas lógicas. São representações não canônicas, mas trivialmente conversíveis em expressões booleanas, e *vice-versa*. A Figura 2.2-d ilustra um circuito lógico.

Os próximos capítulos abordam outras representações gráficas, baseadas em diagramas de decisão.

Chapter 3

Diagramas de Decisão Binária

Vimos que uma grande variedade de métodos de representação para funções booleanas foi desenvolvido. Nos clássicos, tais como tabelas-verdade e forma canônica, a representação de funções de n variáveis possuem tamanho $O(2^n)$. Outras abordagens, tais como listas de mintermos e cobertura prima resultam em representações compactas para muitas funções, mas operações simples como a complementação podem produzir representações de tamanho intratável. Descrevemos aqui um esquema de representação (BDD) baseado em grafo que é muito útil para uma grande classe de funções booleanas, muitas das quais não tratáveis por outros métodos.

3.1 Diagramas de Decisão Binária

A determinação do valor de uma função booleana através da execução de programas baseados apenas em comandos de ramificação (estruturas tipo IF-THEN-ELSE) foi introduzida por C. Y. Lee [7], ao que ele chamou de *programas de decisão binária*. Posteriormente Akers [8] popularizou esta forma de representação sob o nome de *diagrama de decisão binária* (BDD – *binary decision diagram*).

Algoritmos para manipulação de BDD são descritos em [9]. Indicações para implementação eficiente são encontradas em [10],[11],[12] e [13]. Um implementação paralela em ambiente distribuído é apresentada em [14]. Asahi *et al.* [15] propõe a implementação física de *dispositivos BDD* em diversas tecnologias avançadas (dispositivo óptico, dispositivo de *onda eletrônica* e estruturas de tunelamento de elétron). Aplicações recentes de BDDs em Processamento Digital de Imagens têm sido desenvolvidas [16], [17], [18], [19].

Definition 3.1.1 *Um Diagrama de Decisão Binária de uma função booleana* $f : \{0,1\}^n \to \{0,1\}$ *é um grafo acíclico direcionado (DAG) e enraizado, com dois tipos de vértices: terminais e não-terminais. Cada vértice terminal* v *tem como atributo um valor* $valor(v) \in \{0,1\}$*. Dos vértices não-terminais* v *saem dois arcos para dois vértices-filhos,* $zero(v)$ *e* $um(v)$*. Cada vértice* v *não-terminal é rotulado pelo índice de uma variável booleana,* $indice(v) \in \{1, 2, \ldots, n\}$*).*

Para uma dada atribuição de valores ao vetor de variáveis de entrada $\mathbf{x} = (x_1, \ldots, x_n)$*, o valor da função é determinado percorrendo-se o grafo desde a raiz até alcançar um vértice terminal. Assim procedendo, em cada vértice não terminal* v *com* $indice(v) = i$*, se* $x_i = 0$*, então o arco para* $zero(v)$ *é seguido. Caso contrário (se* $x_i = 1$*), o arco para* $um(v)$ *é seguido. O valor da função será dado pelo valor do vértice terminal alcançado.*

Cada vértice v em um BDD representa completamente uma função booleana f_v, a qual é dada recursivamente por:

- se v é um vértice terminal, então $f_v = valor(v)$;
- se v é um vértice não-terminal, então $f_v = \overline{x_{indice(v)}} \cdot f_{zero(v)} + x_{indice(v)} \cdot f_{um(v)}$.

Uma dada função booleana pode ser representada por BDDs distintos. Por exemplo, a árvore de decisão binária é apenas uma dentre as possíveis formas de representação neste esquema. Note que, pela definição de BDD, um caminho da raiz a um terminal pode corresponder a qualquer seqüência de variáveis de entrada. Nas próximas seções algumas restrições são postas nos BDDs (*e.g.* ordem) ou deles removidas (*e.g.* raiz) de modo a torná-los um esquema com unicidade de representação ou então para torná-los mais interessantes em certas aplicações.

A Figura 3.1 mostra diversos BDDs para a função $f(A, B, C) = A\overline{B} + BC$. Note que uns são mais compactos que outros. Por convenção, no desenho do BDD vértices não-terminais são representados por círculos contendo o nome ou o índice da variável associada. Vértices terminais são representados por um quadrado contendo o seu valor. Os arcos $zero(v)$ e $um(v)$ são representados respectivamente por linhas tracejadas e linhas contínuas, ou então são rotulados com 0 ou 1.

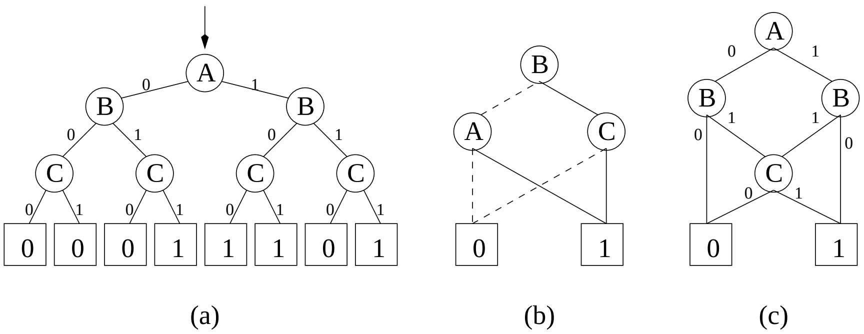


Figure 3.1: Alguns diagramas de decisão de $f(A, B, C) = A\overline{B} + BC$.

3.2 BDDs Compartilhados

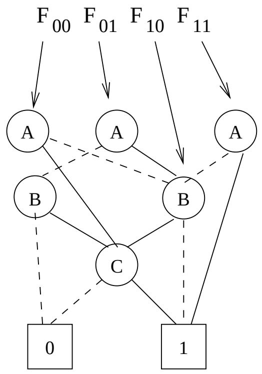
Um número arbitrário de funções (ou equivalentemente, uma função com um número arbitrário de saídas) podem ser representadas num único diagrama, como já era sugerido por Akers [8]. Isto se dá removendo-se o requisito de uma única raiz para o BDD. Um diagrama com múltiplas raízes é obtido pela reunião de BDDs de forma que vértices que representam a mesma função podem ser compartilhados entre os BDDs que deles necessitem, como ilustrado na Figura 3.2. Muitos sistemas que utilizam BDDs para representar funções booleanas trabalham desta forma para economizar memória.

De acordo com [20], um *BDD compartilhado*, ou *SBDD – Shared BDD*, é um conjunto de BDDs combinados por meio de uma árvore para a seleção da saída. Novas variáveis de entrada são criadas a fim de selecionar uma função (saída) dentre as diversas existentes. A Figura 3.2 mostra um SBDD para a função de múltiplas saídas definida pela tabela verdade indicada.

3.3 BDDs genéricos, IBDDs e Programas de Decisão

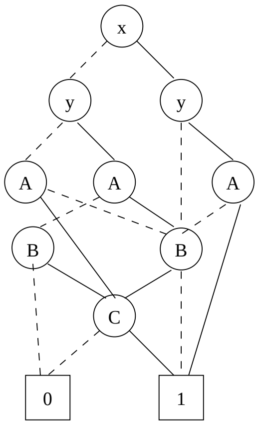
Diagramas genéricos, em que uma variável pode ocorrer mais de uma vez num caminho, em qualquer ordem, são pouco usados, pois os algoritmos para muitas operações de manipulação com tais BDDs têm complexidade exponencial [21]. Em [22] os BDDs estudados são genéricos. Jain *et al.* [23] usam BDDs com repetições de variáveis (IBDDs - BDDs Indexadas) para verificar com sucesso diversos circuitos de multiplicação binária.

A primeira restrição importante posta num BDD é a quantidade de vezes



(a)

A	B	C	F_{00}	F_{01}	F_{10}	F_{11}
0	0	0	1	0	1	1
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	0	0	1
1	1	1	1	1	1	1



(b)

Figure 3.2: BDD multi-raiz e SBDD das funções F_{xy} indicadas.

que uma variável pode ser visitada ao longo de um caminho do diagrama. BDDs em que cada variável aparece ("é lida") no máximo k vezes em um caminho, $k \in I\!N$, são chamados de *programas de decisão de k-leituras* ou BPk – *read-k-times-only branching programs* [24]. Tais estruturas são de interesse em Teoria de Complexidade. Neste contexto os BDDs são também chamados de *grafos de decisão.* [25], [24].

3.4 BDDs Livres

BDDs livres, ou *FBDD (Free BDDs)* [26] ou ainda *read-once branching programs* (BP1) [27] são BDDs onde cada variável pode ocorrer no máximo uma vez em cada caminho da raiz até um vértice terminal.

Gergov e Meinel desenvolveram algoritmos eficientes para operações básicas entre FBDDs [28],[26].

Cada vértice v de uma FBDD tem como filhos vértices que representam funções que não dependem da variável que rotula v. Então, uma forma de se construir o diagrama a partir da expressão booleana da função $f(x_1, \ldots, x_n)$ consiste em aplicar recursivamente a *Expansão de Shannon* das sub-funções em torno de alguma variável disponível x_i:

$f = x_i \cdot f|_{x_i} + \overline{x_i} \cdot f|_{\overline{x_i}}$.

As restrições $f|_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ e $f|_{\overline{x_i}} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ são chamadas de *cofatores* de f em relação às literais

x_i e $\overline{x_i}$, e são funções que não dependem de x_i.

Como uma variável aparece no máximo uma vez em cada caminho, uma função é avaliada em tempo $O(n)$ em um FBDD.

As restrições a seguir foram o que possibilitou o uso em larga escala dos BDDs em síntese lógica.

3.5 BDDs Ordenados

Em seu artigo pioneiro para a comunidade de síntese de circuitos, Akers [8] constrói o BDD de forma que as variáveis ocorrem sempre na mesma ordem ao longo dos caminhos da raiz ao terminal. Entretanto esta restrição não era por ele imposta, a fim de que o desenvolvimento de outras técnicas pudesse levar a diagramas mais compactos. R. E. Bryant [9] formalizou o conceito de BDD e seus variantes (ordenado e reduzido). Apesar de rebatizá-los de *grafos de função*, contrariando o jargão estabelecido, seu artigo é considerado um marco pois apresentou pela primeira vez algoritmos eficientes que efetuam operações lógicas entre BDDs, difundindo mais rapidamente a utilização desta estrutura.

Um *BDD ordenado* (*OBDD – ordered BDD*) é um BDD tal que qualquer caminho no grafo que vai da raiz a um vértice terminal visita as variáveis na ordem ascendente dos seus índices, *i.e.*, $indice(v) < indice(zero(v))$ sempre que v e $zero(v)$ são não-terminais, e $indice(v) < indice(um(v))$ sempre que v e $um(v)$ são não-terminais. Como OBDD é um caso particular de FBDD, isto é, cada variável aparece no máximo uma vez em cada caminho, uma função é avaliada em tempo $O(n)$ em um OBDD.

Renomeando as variáveis de forma a permutar seus índices, o novo diagrama obtido pode ter suas características muito modificadas, como por exemplo o número mínimo de vértices possível. As Figuras 3.1-b e 3.1-c mostram o efeito da mudança de ordem das variáveis.

Encontrar a ordem de variáveis que resulta em um OBDD mais compacto é um problema NP-completo [29] [30]. A técnica de peneiramento (*sifting*) consiste em iterativamente efetuar trocas de variáveis adjacentes segundo certo critério [31]. Tal heurística tem se mostrado eficiente e é utilizada em [32] para otimizar OBDDs.

Denota-se por πOBDD um OBDD que utiliza como ordem das variáveis uma permutação π do domínio de f. Meinel e Slobodová [33] propuseram a representação de f por meio de um OBDD que representa f', onde f' é obtida de f por uma transformação de domínio genérica $\tau : \{0,1\}^n \rightarrow \{0,1\}^m$, com

$m > n$ e τ injetora. Tal transformação é chamada de *cube transformation*. O esquema é batizado de TBDD, e uma representação neste esquema é denotada por τTBDD.

Em [34] é proposto um método eficiente para combinar incrementalmente dois OBDDs ordenados por permutações diferentes π_1 e π_2 sem que seja necessário convertê-los previamente a uma ordem comum π.

Um estudo de complexidade de importantes problemas em grafos são estudados em [35], analisando-os quando a entrada para tais problemas é representada por OBDDs.

3.6 OBDD Reduzido

Dois vértices v_1 e v_2 são *equivalentes* se são raízes de subgrafos isomórficos, ou seja, que representam a mesma subfunção. Um vértice v é dito *degenerado* se $zero(v) = um(v)$ [36].

Se um OBDD não contém pares de vértices equivalentes nem degenerados ele é chamada de *OBDD reduzido* (ROBDD). Um OBDD de N vértices pode ser transformado um ROBDD equivalente em tempo $O(N \cdot \log N)$ pelo algoritmo REDUCE apresentado em [9]. O seguinte teorema estabelece a canonicidade da representação por ROBDD.

Theorem 3.6.1 *Para qualquer função Booleana f, existe um único ROBDD representando f. Ademais, qualquer outro OBDD representando f contém mais vértices.*

Este teorema e sua prova são apresentados em [9]. Este artigo de R. Bryant é um marco no desenvolvimento da área, pois além desta estrutura de dados traz também algoritmos eficientes que resolvem importantes problemas em análise e síntese lógicas:

- *composição lógica*: calcular a função booleana resultante de uma operação lógica entre duas outras;
- *equivalência* ou *verificação formal*: verificar se duas representações correspondem a uma mesma função;
- *satisfatibilidade*: verificar se uma função é satisfatível, *i.e.*, se $f = 1$ para algum vetor de entrada;
- *tautologia*: verificar se f é uma tautologia, *i.e.*, se $f = 1$ para qualquer vetor de entrada.

3.7 QOBDD

Um OBDD obtido da árvore de decisão correspondente apenas eliminando-se os vértices equivalentes repetidos é chamado de *OBDD quase reduzido* (*QOBDD – quasi-reduced OBDD*) [36]. Em [37] o mesmo é denominado *diagrama de decisão simples*, e a operação de eliminação de subgrafos idênticos é chamada de *reticulação*.

Assim como ROBDDs, os QOBDDs são representações canônicas. Segundo [36], o tamanho de um QOBDD é da ordem do tamanho de sua ROBDD vezes o número de variáveis. Aparentemente os QOBDDs são de interesse apenas teórico.

Em um QOBDD, todos os caminhos da raiz a um vértice qualquer v possuem o mesmo comprimento. Um BDD com esta característica é chamado de *síncrono* [24].

Num esquema variante do ROBDD, chamado ROBDD particionado, ou (*POBDD - Partitioned OBDD*), o diagrama é ordenado por partes [38].

Chapter 4

Outras Representações Gráficas para Funções Booleanas

Em todos os diagramas vistos até agora, os vértices não terminais representam uma decisão, isto é, uma variável de entrada é testada e o arco correspondente ao seu valor é seguido. Como vimos, um vértice de decisão representa uma função de três entradas, sendo uma delas a variável de controle que serve para selecionar as outras duas, representando as subfunções que "chegam" de outros vértices do diagrama (por este motivo, costuma-se implementar um BDD por meio de uma rede booleana consistituída apenas por multiplexadores do tipo $2 \to 1$).

No entanto, outras formas interessantes de representação por diagramas semelhantes foram propostas. Em geral, estes novos diagramas usam, além de vértices de decisão, outros tipos de vértices, representando outras funções. Normalmente estas outras funções são uma das 16 funções lógicas possíveis entre as duas subfunções ou outras funções mais complexas. Obviamente estas generalizações têm de ser criteriosas para não cair no caso geral da simples representação por redes booleanas.

Meinel [27] define formalmente a classe dos diagramas que batizou de Ω-*BP* (Ω - *Branching Program*). Ω-BPs generalizam o conceito de programas (diagramas) de decisão equipando-se alguns vértices com funções booleanas $f \in \Omega \subseteq \mathcal{B}_{2^2}$. Neste contexto é permitida a repetição de variáveis no mesmo caminho. Um extenso estudo teórico da complexidade de tais diagramas é apresentado em [27].

4.1 Arcos Complementados

O primeiro caso de generalização consiste em incluir na BDD um vértice especial com apenas um arco de sáıda, que representa a função de negação. Com este esquema é possível representar f e $\overline{f}$ num mesmo diagrama, economizando metade dos vértices. Na verdade, para que todos os vértices (não terminais) continuem com exatamente dois arcos de saída ($um(v)$ e $dois(v)$), o que se faz é guardar a informação (*flag*) de negação no arco, ao invés de se criar o vértice especial.

Este esquema se chama *BDD com arcos complementados* e já havia sido sugerido em [8]. No desenho do BDD costuma-se indicar o arco complementado por um pequeno círculo no meio dele.

Para manter a canonicidade de representação, fixa-se o local onde arcos complementados podem ser usados. A saber, para cada vértice v o arco $(v, um(v))$ não deve ser complementado.

4.2 ZBDD

Um vértice v é dito *desnecessário* se $um(v)$ é o vértice terminal tal que $valor(um(v)) = 0$ [36]. Um diagrama obtido de um QOBDD removendo-se os vértices desnecessários é chamado de ZBDD ou *0-Sup-BDD* (*Zero-Suppressed BDD*) [39].

Num ZBDD, a determinação do valor da função para um certo vetor de entrada é feita de forma diversa à da definição 3.1.1. Por exemplo, suponhamos que $indice(v_a) = x_2$, $indice(v_b) = x_6$, e $um(v_a) = v_b$. Então temos um arco que "salta" as variáveis x_3, x_4 e x_5. O valor da subfunção a ser considerada no cômputo de $f_{v_a}|_{v_a}$ não é a função f_{v_b} como se poderia pensar, mas sim, $\overline{x_3} \cdot \overline{x_4} \cdot \overline{x_5} \cdot f_{v_b}$.

ZBDD são representações canônicas de funções booleanas. Minato [39] mostrou que importantes problemas combinatórios podem ser resolvidos eficientemente usando a representação por ZBDD.

ZBDDs se mostram adequados a aplicações que podem ser resolvidas pela representação e manipulação de conjuntos esparsos de vetores binários, ou seja, quando os dados de interesse ao serem codificados em vetores de n bits apresentam muito mais zeros do que uns. Representam-se então subconjuntos de vetores por funções de n variáveis. O tamanho do ZBDD em relação ao ROBDD chega a ser até n vezes menor.

4.3 Diagramas Booleanos Livres

Diagramas Booleanos Livres (*FBD - Free Boolean Diagrams*) são estruturas de dados derivadas dos FBDDs (variáveis de decisão podem ocorrer em qualquer ordem num caminho, mas no máximo uma única vez). A diferença está na inclusão de vértices funcionais (XOR e AND), além de permitir arcos complementados.

Diferentemente dos vértices de decisão, os vértices funcionais podem ocorrer diversas vezes num caminho. Um vértice v do tipo XOR calcula a função $f_v = f_{zero(v)} \oplus f_{um(v)}$. Um vértice v do tipo AND calcula a função $f_v = f_{zero(v)} \cdot f_{um(v)}$, onde os suportes dos cofatores são disjuntos. Isto é, $f_{zero(v)}$ e $f_{um(v)}$ não podem depender de variáveis comuns.

FBDs não são canônicos [40], mas o teste de equivalência e outras operações podem ser eficientemente calculados por meio de uma abordagem probalbilística [41]. Vide também [37], [42]

4.4 Diagramas de Decisão Funcionais

Diagramas de Decisão Funcionais (FDDs) [43] [44] são semelhantes a BDDs, mas utilizam a expansão de Reed-Muller (também conhecida como expansão de Davio negativa) ao invés da decomposição de Shannon. Ou seja, um vértice v rotulado pela variável x representa a função:

$f = f|_x \oplus x \cdot f_{\Delta_x}$, onde $f_{\Delta_X} = f|_x \oplus f|_{\overline{x}}$ denota a *diferença Booleana* de f com respeito a x.

Um *OFDD (FDD ordenado)* apresenta as mesmas restrições de ordem de variáveis que o OBDD, e as mesmas regras de redução das ZBDDs devem ser aplicadas (eliminação de vértices equivalentes repetidos e de vértices desnecessários).

OFDDs possuem muitas propriedades em comum com ROBDDs: A representação é canônica e muitas operações podem ser implementadas com algoritmos de complexidade polinomial [21]. Entretanto, além de usar uma regra de redução distinta da usada em ROBDDs, outra diferença que OFDDs apresentam é o fato de se ter que examinar mais de um caminho para se avaliar a função. Exite algoritmo de complexidade linear no tamanho do diagrama para avaliar f.

4.5 OKFDD

Para algumas classes de funções, FDDs são exponencialmente mais compactas que OBDDs, mas a recíproca também pode ser verdadeira [21]. O diagrama funcional de Kronecker (*OKFDD – Ordered Kronecker FDD*) [45] é um esquema híbrido que aproveita as vantagens de ambos. Para isto, a cada variável de entrada está associado um indicador do tipo de decomposição que ela permite: Reed-Muller, Davio positiva ou Shannon. Experimentos relatam um ganho médio de 35% de compactação em relação aos ROBDDs.

4.6 OBDD de Paridade

*POBDD*s (*Parity OBDD*) são representações como OBDD, mas em cada vértice não-terminal pode haver um número arbitrário de 1-sucessores $um_i(v)$ e 0-sucessores $zero_i(v)$. Assim cada vetor de entrada pode definir vários caminhos a serem computados. Se o número de caminhos da raiz ao terminal "1" for ímpar então a função booleana vale 1. Se for par vale 0 [46].

Segue desta definição que em um vértice v com 0-sucessores $w_1, \ldots, w_k$ e 1-sucessores $u_1, \ldots, w_l$, a função computada é:

$f_v = \overline{x}_i(f_{w_1} \oplus \cdots \oplus f_{w_k}) + x_i(f_{u_1} \oplus \cdots \oplus f_{u_l})$.

Waack [47] descreve um algoritmo para reduzir o numero de vértices de um POBDD e outro para o teste de satisfatibilidade, de complexidade polinomial com relação à quantidade de vértices dos diagramas de entrada.

4.7 Grafo do Valor de Saída

Jiang e Majihita [48] propuseram um diagrama que é um compromisso entre BDDs e tabelas-verdade. O referido diagrama é chamado de *OVAG – Output Value Array Graph*. OVAGs utilizam uma extensão para k bits da expansão de Shannon. Em [48] são apresentados algoritmos que manipulam OVAGs e alguns resultados preliminares mostram que o método pode ser útil em verificacção de circuitos lógicos, por apresentar bom desempenho para certas funções.

Chapter 5

Funções Incompletamente Especificadas

Em algumas circunstâncias, o valor de uma função para determinados vetores de entrada pode ser ignorado, ou então não é conhecido. Esta situação ocorre, por exemplo, em aprendizado automático de funções booleanas que descrevam filtros (*e.g.* de imagens) a partir de pares de treinamento (padrão a classificar, padrão classificado). Tais aplicações consideram relevantes apenas um subconjunto do domínio da função. Estas funções parciais, no contexto de álgebra booleana, são chamadas de funções incompletamente especificadas, ou funções incompletas.

O domínio de uma função booleana incompleta f é particionado em três conjuntos:

- Conjunto-1, ou *ON-set*, o conjunto dos vetores $\mathbf{x}$ de entrada para os quais $f(\mathbf{x}) = 1$. Denota-se por f_{on} a função característica deste conjunto.

- Conjunto-0, ou *OFF-set*, o conjunto dos vetores $\mathbf{x}$ de entrada para os quais $f(\mathbf{x}) = 0$. Denota-se por f_{off} a função característica deste conjunto.

- Conjunto-X, ou *DC-set*, o conjunto dos vetores $\mathbf{x}$ de entrada para os quais $f(\mathbf{x})$ não é especificado. Denota-se por f_{dc} a função característica deste conjunto.

Os elementos de Conjunto-X são chamados de *não-especificações* ou *don't cares*.

De forma geral, uma função incompleta cujo conjunto-X tem tamanho $\#dc$ define uma família de $2^{\#dc}$ funções, obtida pela atribuição de 0 e 1 às suas não-especificações.

Duas funções incompletas são *compatíveis* se elas podem ser tornadas equivalentes por uma adequada atribuição de valores às suas não-especificações.

5.1 BDDs de Funções Incompletas

Shiple *et al.* [49] abordam o problema de encontrar o menor ROBDD de uma função incompleta f que responda corretamente quando os vetores de entrada estão no conjunto-1 ou no conjunto-0. Os dados de entrada para o problema são a função f já na forma de um ROBDD (por exemplo, considerando os *don't cares* como 0), e a função f_{dc}, também na forma de ROBDD.

Este problema de minimização é chamado de EBM (*exact BDD minimization*) e o problema de decisão correspondente é da classe de complexidade NP. Em [49],[50] e [51] várias heurísticas para a solução do EBM são apresentadas e discutidas.

5.2 MBD

A fim de representar funções incompletamente especificadas, Jacobi [52] [53] [54] propôs o diagrama de decisão chamado *MBD – Modified Binary Diagram*. Um MBD é um BDD (com múltiplas raízes) acrescido de um vértice terminal denominado "X". Neste diagrama, as não especificações correspondem a caminhos que terminam neste vértice.

Chapter 6

Funções Discretas

Funções discretas são funções do tipo $f : S \to L$, onde S e L são conjuntos finitos. Em particular, se $S = \{0, 1\}^n$ e $L = \{0, 1, \ldots, k - 1\}$, a função é chamada de *pseudo-lógica*. Funções discretas do tipo $f : \{0, 1, \ldots, k-1\}^n \to \{0, 1, \ldots, k - 1\}$ são chamadas *funções da lógica k-valente*, ou funções k-valoradas.

Similarmente às funções booleanas, funções discretas podem ser representadas por tabelas ou por expressões compostas por operações de uma álgebra (multi-valorada) e as variáveis de entrada.

Diversos esforços em obter representações alternativas para funções discretas têm sido observados. Examinaremos aqui as representações baseadas em grafos de decisão.

6.1 BDD Multi-Terminal

Um BDD Multi-Terminal (MTBDD) [21], também chamado de Diagrama de Decisão Algébrico (ADD) [55], é usado para representar funções pesudo-lógicas. É um BDD modificado de modo que há um vértice terminal para cada valor do contra-domínio da função.

MTBDDs podem ser usados para representar também funções k-valoradas. Para tanto, escolhe-se uma codificação binária para as variáveis de entrada, e os vértices não-terminais são rotulados por um bit desta codificação. Para um dado problema, a codificação escolhida pode influir fortemente no diagrama.

MTBDDs compartilhados (SMTBDDs) são definidos em [20], onde funções de múltiplas saídas são selecionadas por meio de novas variáveis, como já

y	z	F
0	0	8
0	1	-12
1	0	10
1	1	-6

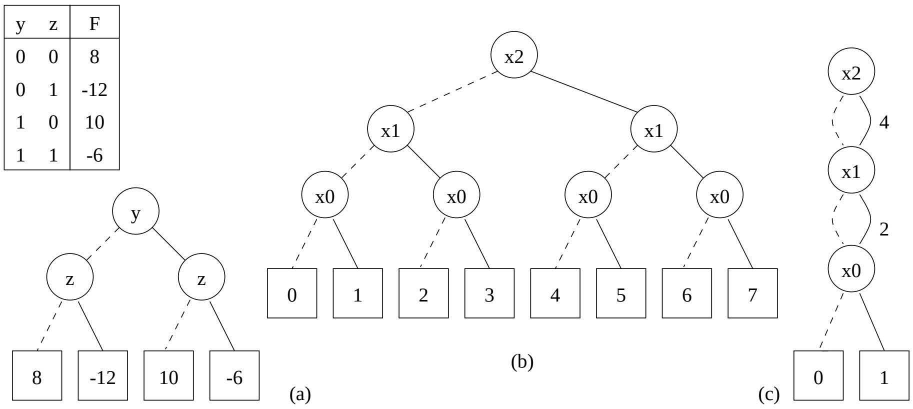


Figure 6.1: (a,b) MTBDDs e (c) EVBDD

mencionado (v. SBDD).

A Figura 6.1-a ilustra um MTBDD para a função descrita na tabela mostrada. A MTBDD da Figura 6.1-b representa a função $x_0 + 2x_1 + 4x_2$.

MTBDDs são ineficientes quando a faixa de valores de saída é grande. Para algumas aplicações, quando forem poucos os valores de saída, esta representação, pela simplicidade e similaridade com BDDs, pode ser aceitável.

6.2 BDDs com Arcos Valoradas

Outra abordagem para a representação de funções k-valoradas consiste em associar valores aos arcos de um MTBDD. Após a determinação do valor do nó terminal alcançado por uma atribuição de variáveis, soma-se a este os valores dos arcos do respectivo caminho. O resultado é o valor da função. A Figura 6.1-c mostra o diagrama para a função $x_0 + 2x_1 + 4x_2$. Quando omisso no desenho, o valor da aresta é zero.

Este diagrama é chamado de EVBDD (*Edge-valued BDD*). Esta representação cresce linearmente com o número de bits da saída [21]. A representação não é canônica mas pode tornar-se quando certas restrições na atribuição de valores nos arcos são impostas. Uma delas é que o valor da aresta $(v, zero(v))$ seja zero.

y	z	F
0	0	8
0	1	-12
1	0	10
1	1	-6

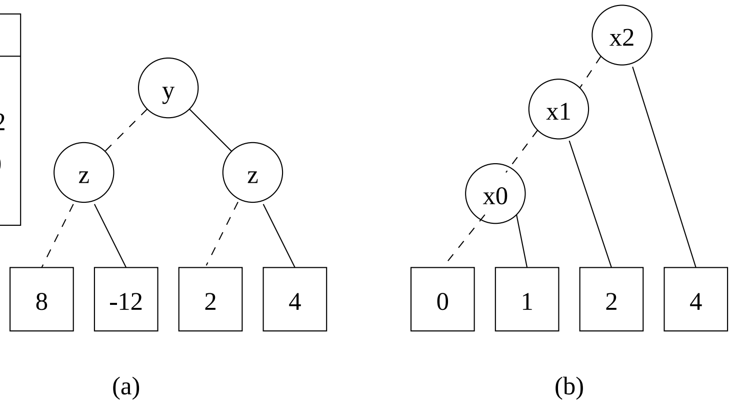


Figure 6.2: BMDs

6.3 Diagramas de Momento Binário

*BMD*s (*Binary Moment Diagrams*)[21] são diagramas que em cada vértice rotulado por x utilizam, ao invés da decomposição de Shannon, a seguinte:

$f = f|\overline{x} + x(f|x - f|\overline{x}),$ onde $x \in \{0, 1\}$.

$f|\overline{x}$ é o *momento constante* de f e é representado pelo vértcie $zero(v)$. $f|x - f|\overline{x}$ é o *momento linear* de f e é representado por $um(v)$. $+$ e $\cdot$ denotam soma e multiplicação.

BMDs são apropriadas para representar funções aritméticas. A Figura 6.3-a mostra o BMD da função dada pela tabela indicada e a Figura 6.3-b mostra o BMD da função $x_0 + 2x_1 + 4x_2$.

6.4 BMD Multiplicativas

Uma extensão à BMD consiste em incorporar pesos nos arcos, resultando numa representação chamada **BMD* [56]. Os pesos dos arcos combinam-se multiplicativamente, ao invés de somá-los como no EVBDD. Com *BMD funções como $X + Y$, $X - Y$, $X \cdot Y$ e 2^X têm representação de tamanho linear no tamanho da entrada [21].

6.5 Diagramas de Decisão Híbridos

Adaptando as idéias do OKFDD, criou-se um híbrido entre BMD e MTBDD, chamado *HDD* (*Hybrid Decision Diagramas*). Nesta representação, a cada variável de entrada está associado um dentre quatro tipos permitidos de decomposição, dentre elas a pontual ($f = (1-x) \cdot f|x_{\overline{x}}|x \cdot f_x$) e a de momento [21].

6.6 Diagramas de Decisão Multi-Valorados

MDD (*Multi-valued Decision Diagramas*) são diagramas para representar funções $f : \{0, 1, \ldots, k-1\} \rightarrow \{0, 1\}$, onde cada vértice não-terminal está associado a uma variável multi-valorada e possui um arco para cada valor possível. Assim, o vértice representa uma *k-decisão*.

Para representar funções k-valoradas há um vértice terminal para cada valor $\{0, 1, \ldots, k\}$ e o diagrama é chamado MTMBD.

De forma análoga aos MTBDDs, os MDDs e MTMDDs podem ser compartilhados (SMDDs e SMTMDDs) [57] [20].

Chapter 7

Conclusões

Este trabalho consistiu em fazer um levantamento bibliográfico na área de Representação de Funções Discretas por Diagramas de Decisão. A motivação para este trabalho provém da busca por soluções eficientes para a implementação de *máquinas morfológicas* binárias, como a que foi sugerida em [18], e da sua extensão para máquinas morfológicas para processamento de imagens em níveis de cinza.

Os primeiros resultados para alguns operadores morfológicos binários foram muito promissores. Um atual desafio é conseguir implementações de operadores morfológicos de imagens multi-níveis que sobrepujem as implementações convencionais. Outro desafio importante é caracterizar a generalização que os diagramas de funções incompletamente especificadas proporcionam. Este problema é fundamental para a síntese automática de operadores por aprendizado baseado em exemplos.

A literatura na área é vasta, e procuramos neste trabalho levantar os tipos de BDDs mais representativos. Desde o artigo de [9] a produção na área vem sustentando um ritmo acelerado. Apesar dos BDDs terem surgido na comunidade de Circuitos Digitais, os diagramas têm sido muito estudados pelos teóricos em Complexidade de Algoritmos e vm se mostrando aplicáveis em diversas áreas tais como compressão de dados e processamento de imagens.

Bibliography

[1] G. J. F. Banon and J. Barrera. Bases da Morfologia Matemática para Análise de Imagens Binárias. IX Escola de Computação, Pernambuco, July 1994.

[2] E. Fabricius. *Modern Digital Design and Switching Theory.* CRC Press, Boca Raton - CA, 1992.

[3] F. Preparata and R. Yeh. *Introduction to Discrete Structures for Computer Science and Engineering.* Computer Science and Information Processing. Addison-Wesley, 1973.

[4] A. A. G. Requicha. Representation of Rigid Solids: Theory, Methods and Systems. *ACM Computing Surveys*, 12(4):437–464, December 1980.

[5] N. S. Tomita. Programação Automática de Máquinas Morfológicas Binárias baseada em Aprendizado PAC. Master's thesis, Instituto de Matemática e Estatística - Universidade de São Paulo, SP, Brazil, March 1996.

[6] R. Jacobi. Síntese de Circuitos Lógicos Combinacionais. Décima Escola de Computação, Campinas, July 1996.

[7] C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell Syst. Tech. Journal*, 38:985–999, 1959.

[8] S. B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, June 1978.

[9] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[10] K. Karplus. Representing Boolean Functions with If-Then-Else DAGs. Technical Report UCSC-CRL-88-28, UCSC, November 1988.

[11] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of 27th ACM/IEEE Design Automation conference*, pages 40–45, 1990.

[12] J. V. Sanghavi, R. K. Ranjan, R. K. Brayton, and A. S. Vincentelli. High Performance BDD Package by Exploiting Memory Hierarchy. In *Proc. of 33rd DAC*, pages 635–640, Las Vegas, NE, June 1996.

[13] S. Manne, D. Grunwald, and F. Somenzi. Remembrance of Things Past: Locality and Memory in BDDs. In *Proc. of 34th DAC*, pages 196–201, Anaheim, CA, 1997.

[14] A. Stornetta. Implementation of an Efficient Parallel BDD Package. Master's thesis, Electrical and Computer Engineering - Univ. California, Santa Barbara, December 1995.

[15] Y. Amemiya N. Asahi, M. Akazawa. Binary-Decision-Diagram Device. *IEEE Transactions on Electron Devices*, 42(11), November 1995.

[16] M. Starkey and R. Bryant. Using Ordered Binary-Decision-Diagrams for Compressing Images and Image Sequences. Technical Report CMU-CS-95-105, School of Computer Science, Carnegie Mellon Univeristy, January 1995.

[17] L. Robert and G. Malandain. Fast Binary Image Processing Using Binary Decision Diagrams. *Computer Vision and Image Understanding*, 72(1):1–9, October 1998.

[18] H. M. F. Madeira, J. Barrera, R. Hirata Jr, and N. S. T. Hirata. A New Paradigm for the Architecture of Morphological Machines : Binary Decision Diagrams. In J. Stolfi and C. L. Tozzi, editors, *Proc. SIBGRAPI'99 - XII Brazilian Symposium in Computer Graphics and Image Processing*, pages 283–292, Campinas, SP, Brazil, November 1999.

[19] H. M. F. Madeira and J. Barrera. Incremental Evaluation of BDD-Represented Set Operators. In *Proceedings of the XIII Brazilian Symposium in Computer Graphics and Image Processing*, Gramado, October 2000. IEEE.

[20] H. Babu and T. Sasao. Time-Division Multiplexing Realizations of Multiple-Output Functions Based on Shared Multi-Terminal Multiple-Valued Decision Diagrams. *IEICE Trans. Inf. & Syst.*, E82-D(5), May 1999.

[21] R. E. Bryant. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. In *Proc. of ICCAD'95*, November 1995.

[22] P. Ashar, S. Devadas, and A. Ghosh. Boolean Satisfiability and Equivalence Checking Using General Binary Decision Diagrams. In *International Conference on Computer Design*, pages 259–264. IEEE, 1991.

[23] J. Jain, J. Bitner, M. Abadir, J. A. Abraham, and D. S. Fussell. Indexed BDDs: Algorithmic Advances in Techniques to Represent and Verify Boolean Functions. 1994.

[24] J. Gergov and C. Meinel. Analysis and Manipulation of Boolean Functions in Terms of Decision Graphs. In *Proc. of the Workshop on Graph-Theoretic Concepts in Computer Science*, volume 657 of *LNCS*, pages 310–320. Springer-Verlag, 1993.

[25] W. Masek. A Fast Algorithm for the String Editing Problem and Decision Graph Complexity. Master's thesis, MIT, 1976.

[26] J. Gergov and C. Meinel. Boolean Manipulation with Free BDDs. An Application in Combinational Logic Verification. In *Proc. of 13th IFIP World Computer Congress*, volume 1 of *IFIP Transactions A-51 "Technology and Foundations"*, pages 309–314, Hamburg, 1994. North Holland.

[27] C. Meinel. *Modified Branching Programs and Their Computational Power*, volume 370 of *LNCS*. Springer-Verlag, 1989.

[28] J. Gergov and C. Meinel. Efficient Boolean Manipulation with OBDD's Can be Extended to FBDD's. Technical Report 93-12, Informatik, Universtät Trier, Trier, 1993.

[29] S. Tani, K. Hamaguchi, and S. Yajima. The Complexity of the Optimal Variable Ordering Problem of Shared Binary Decision Diagrams. In *Proc. ISAAC'93*, volume 762 of *LNCS*, pages 389–398, 1993.

[30] B. Bollig and I. Wegener. Improving the Variable Ordering of OBDDs is NP-Complete. *IEEE Transactions on Computers*, 45:993–1002, 1996.

[31] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proc. IEEE/ACM ICCAD'93 - International Conference on Computer-Aided Design*, pages 42–47, Santa Clara, CA, November 1993.

[32] C. Meinel, F. Somenzi, and T. Theobald. Linear Sifting of Decision Diagrams . In *DAC'97*, Anaheim, CA, 1997.

[33] C. Meinel and Anna Slobodová. A Unifying Theoretical Background for Some BDD-based Data Structures. Technical Report 94-17, Informatik, Universtät Trier, Trier, 1994.

[34] C. Meinel, H. Sack, C. Stangier, and A. Wagner. Do We Really Need Common Variable Orders for Synthesizing OBDDs ? Technical Report 98-29, Informatik, Universtät Trier, Trier, 1998.

[35] J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. Complexity of Problems on Graphs Represented as OBDDs. In *Proc. of the 1998 Symposium on Theoretical Aspects of Computer Science*, 1998.

[36] K. Hayase. On Relationship Between Boolean Functions and Their Prime Implicants in OBDD Size. In *6th BDD Workshop*, Hongo, Tokyo 113, December 1995.

[37] B. Moret. Decision Trees and Diagrams. *Computing Surveys*, 1(4), December 1982.

[38] A. Narajan, J. Jain, M. Fujita, and A. Sangiovanni-Vincentelli. Partitioned ROBDDs - A Compact, Canonical and Efficient Manipulable Representation of Boolean Functions. In *Proc. IEEE/ACM ICCAD'97*, pages 547–554, San Jose, CA, 1996.

[39] S. Minato. Zero-suppressed BDDs for set Manipulation in Combinatorial Problems. In *30th Design Automation Conference*, pages 272–277, June 1993.

[40] A. Ghosh A. Shen, S. Devadas. Probabilistic Manipulation of Boolean Functions Using Free Boolean Diagrams. >=1993.

[41] A. Ghosh A. Shen, S. Devadas. Probabilistic Construction and Manipulation of Free Boolean Diagrams. In *Proc. International Conference on Computer-Aided Design*, pages 544–549, November 1993.

[42] M. Blum, A. K. Chandra, and M. N. Wegman. Free Boolean Graphs Can Be Decided Probabilistically in Polinomial Time. *Information Processing Letters*, 10(2):80–82, 1980.

[43] B. Becker, R. Drechsler, and R. Werchner. On the Relation between BDDs and FDDs. *Information and Computation*, 123:185–197, 1995.

[44] U. Kebschull, E. Schubert, and W. Rosentiel. Multilevel Logic Synthesis Based on Functional Decision Diagrams. In *Proc. European Design Automation Conf. (EDAC'92)*, pages 43–47, Brussels, 1992.

[45] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. Perkowski. Efficient Representation and Manipulation of Switching Functions Based on Ordered Kronecker Functional Decision Diagrams. In *31st Design Automation Conference*, pages 415–419, June 1994.

[46] R. Drechsler, M. Sauerhoff, and D. Sieling. The Complexity of the Inclusion Operation on OFDD's. *IEEE Transactions on Computer Aided Design of Integrated Circuits*, 1998.

[47] S. Waack. On the Descriptive and Algorithmic Power of Parity Ordered Decision Diagrams. In *Proc. 14th Symp. Theoretical Aspects of Comput. Science (STACS'97)*, volume 1200 of *LNCS*, pages 201–212. Springer Verlag, 1997.

[48] H. Jiang and J. C. Majithia. Suggestion for a New Representation for Binary Function. *IEEE Transaction on Computers*, 45(12), December 1996.

[49] T. R. Shiple, R. Hojati, A. L. S. Vincentelli, and R. K. Brayton. Heuristic Minimization of BDD's Using Don't Cares. In *Proceedings of 31st DAC*, pages 225–231, 1994.

[50] M. Sauerhoff and I. Wegener. On the Complexity of Minimizing the OBDD Size for Incompletely Specified Functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(11):1435–1437, November 1996.

[51] Y. Hong, P. A. Beerel, J. R. Burch, and K. L. McMillan. Safe BDD Minimization Using Don't Cares. In *Proc. of 34th DAC*, pages 208–213, Anaheim, CA, 1997.

[52] R. Jacobi and C. Trullemans. Incremental Reduction of Binary Decision Diagrams. In *Proc. ISCAS - International Symposium on Computer and Systems*, Singapore, 1991.

[53] R. Jacobi and A.-M. Trullemans. Boolean Mapping with Binary Decision Diagrams. In *Proc. IFIP International Workshop on Logic and Architecture Synthesis*, Grenoble, December 1993.

[54] R. Jacobi. *A Study of the Application of Binary Decision Diagrams in Multilevel Logic Synthesis.* PhD thesis, Faculté de Sciences Appliquées, Université Catholique de Louvain, Belgium, 1993.

[55] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and their Applications. In *Proc. of ICCAD'93*, November 1993.

[56] R. E. Bryant and Chen Yirng-An Chen. Verification of Arithmetic Circuits with Binary Moment Diagrams. In *32nd Design Automation Conference*, June 1995.

[57] H. Babu and T. Sasao. Shared Multi-Terminal Binary Decision Diagrams for Multiple-Output Functions. *IEICE Trans. Fundamentals*, E81-A(12), December 1998.