AVALIAÇÃO DO PROBLEMA DE ORDENAÇÃO EM DIAGRAMAS DE DECISÃO BINÁRIA

Alexandre Soares Alves

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Claudson Ferreira Bornstein, Ph.D.

Prof. Edil Severiano Tavares Fernandes, Ph.D.

Prof. Jayme Luiz Szwarcfiter, Ph.D.

Prof. Sérgio Vale Aguiar Campos , Ph.D.

RIO DE JANEIRO, RJ - BRASIL

FEVEREIRO DE 2001

ALVES, ALEXANDRE SOARES

Avaliação do Problema de Ordenação em Diagramas de Decisão Binária. [Rio de Janeiro] 2001

XI, 65 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computacão, 2001)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - Diagramas de Decisão Binária

2 - Ordenação de Variáveis

I. COPPE/UFRJ II. Título (série)

Aos meus pais e meus filhos

Agradecimentos

Agradeço:

ao professor Claudson Ferreira Bornstein, meu orientador, pela confiança, estímulo e paciência a mim dedicados, mas sobretudo pelo exemplo de profissionalismo e dedicação durante o desenvolvimento deste trabalho;

à Maria Clícia Steling de Castro, minha grande amiga, que com seu espírito impetuoso e dedicado empenhou várias horas de trabalho me auxiliando durante os momentos em que mais precisei;

ao professor Nelson Maculan, pelo incentivo e confiança em mim depositados desde do início deste trabalho;

ao Instituto de Pesquisas da Marinha (IPqM) pela liberação concedida para que este trabalho pudesse ser desenvolvido;

à Cláudia, Solange, Sueli, Mercedes, Ari, Lúcia e Marli que fazem parte do corpo administrativo do Programa de Engenharia de Sistemas e Computação, que com dedicação e competência ajudam a construí-lo a cada dia e a mantê-lo. Agradeço também aos técnicos, Júlio(s), Adilson, Carlos e Frederico. Agradeço ainda a Tia Gercina que com sua simpatia e bom humor contribuiu para o bem estar do ambiente em que este trabalho foi realizado;

à minha família pelo apoio e compreensão pelas longas horas de ausência de um convívio tão prazeroso. Em especial, aos meus pais Edio e Iacy, e aos meus filhos Breno, Danilo e Thaís pelo carinho e incentivo;

aos meus companheiros Selma, Ana Mirtes e Moises, pela amizade, pelos desabafos e pelas longas conversas que ajudaram a superar aqueles momentos difíceis que passamos durante este período.

à tantos mais que compartilharam comigo desse caminho.

Às vezes é preciso não só sonhar, mas também é preciso trabalhar, se dedicar e acreditar para alcançar um objetivo.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M. Sc.)

Avaliação do Problema de Ordenação em Diagramas de Decisão Binária

Alexandre Soares Alves

FEVEREIRO/2001

Orientador: Claudson Ferreira Bornstein
Programa: Engenharia de Sistemas e Computação

Diagramas de Decisão Binária (BDDs) têm se mostrado uma ferramenta essencial no processo de verificação automática de sistemas. A solução de uma grande classe de problemas de interesse prático recai na utilização de BDDs. Quando é imposta uma ordenação às variáveis que descrevem o sistema, o BDD passa a ser denominado BDD ordenado (OBDD). Os OBDDs têm como característica essencial o fato de representar unicamente cada função tornando viável a verificação de sistemas.

O tamanho de um OBDD pode variar muito de acordo com a ordenação de suas variáveis. Encontrar uma boa ordenação de variáveis é fundamental para manipulação de funções Booleanas. Nosso estudo é baseado em informações sobre a topologia de circuitos combinacionais para se obter um limite superior para o tamanho de um OBDD. A partir da descrição do circuito é construído um grafo de representação do circuito. A estimativa do tamanho do OBDD é feita sem que seja necessária a sua construção. Dada uma ordenação qualquer, estima-se através de cortes no grafo de representação do circuito, um limite superior para cada nível do OBDD. Utilizamos esta estimativa juntamente com um método de reordenação de variáveis (*sifting*), de modo a obter uma boa ordenação que viabilize a aplicação de métodos pré-existentes de reordenação de variáveis.

Além disso, utilizamos esta nova ordenação para verificar a eficácia deste método para estimar o tamanho do OBDD resultante. Os resultados mostram que para a maioria dos exemplos utilizados a ordenação sugerida com a utilização da estimativa, reduz o tamanho dos OBDDs. Em alguns casos esta redução é bastante significativa.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M. Sc.)

Evaluating the Ordering Problem of Binary Decision Diagrams

Alexandre Soares Alves

FEBRUARY/2001

Advisor: Claudson Ferreira Bornstein
Department: Systems Engeneering and Computer Science Program

Binary Decision Diagrams (BDDs) play an important roll in the process of automatic system verification. The solution to a large class of practical problems involves the use of BDDs. When an order is imposed on the variables that describe the system, the BDD is called an ordered BDD (OBDD). An essential property of OBDDs is the fact that they uniquily represent each function, thus enabling the systems verification process. The size of an OBDD can vary a lot according to the order imposed on its variables. It becomes crucial to find a good ordering for an OBDD in order to manipulate Boolean functions.

Our study is based on topological information of combinational circuits, which is used to obtain an upper bound for the size of an OBDD. We map the circuit onto a graph whose nodes have a one-to-one correspondance with the gates of the circuit. Cuts on this circuit are used to obtain an estimate for the size of the OBDD without requiring the OBDD to be actually built.

This estimate can be used along with a reordering technique in order to obtain good initial orders (without ever constructing the OBDD), thus enabling the application of pre-exhisting variable ordering methods.

This new ordering is also compared to the initial order given, in order to assess the goodness of our estimation method. The experimental results show that in most cases the proposed ordering results in smaller OBDDs. In some cases it results in a more pronounced reduction in size.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nos últimos anos temos observado o crescimento da complexidade dos projetos de *hardware* e de *software*. Garantir a correção destes projetos, tornou-se um problema mais difícil. Por outro lado, *hardware* e *software* são, agora, partes integrantes de nossas vidas. Portanto, a correção de seus projetos é fator indispensável. Em particular, em certas aplicações críticas tais como instrumentação médica, sistemas de controle de tráfego aéreo, entre outras, as falhas podem ter conseqüências catastróficas. Mesmo para aquelas aplicações consideradas não tão críticas, como por exemplo projetos de circuitos digitais, erros de projetos podem resultar em enormes prejuízos econômicos, devido a larga escala que certos dispositivos são fabricados.

1.1 Verificação Formal de Sistemas

Há muitas razões práticas para aplicar métodos de verificação formal em sistemas computacionais. Uma delas é o alto custo da correção de erros em projetos digitais. O custo tem aumentado com o crescimento do nível de integração das tecnologias de circuitos digitais. Portanto, há um crescimento na demanda de metodologias de projetos que resultem em projetos corretos já na primeira fase de fabricação.

Os simuladores são exemplos de ferramentas utilizadas para se encontrar erros antes da fabricação, que modelam o comportamento de um sistema para um determinado padrão. O uso de simulação pode apresentar alguns problemas. Um deles é criar um conjunto de padrões de entrada que seja suficiente para expor qualquer comportamento incorreto do sistema. Um outro problema é determinar a saída correta do sistema dadas essas condições, para ser comparada com a saída simulada. O aumento da densidade de integração permitiu que funções de alto nível fossem implementadas em *hardware*, e como resultado o problema da simulação tornou-se

mais complexo. O que parece ser necessário é uma maneira precisa de especificar o comportamento correto do *hardware*. Um modo de se tratar esse tipo de problema é utilizando a verificação formal.

A verificação formal tem três elementos básicos: um modelo matemático do sistema a ser verificado; uma linguagem formal para especificar a correção do problema; e metodologias para provar essa correção. Uma característica que muitas metodologias de verificação automática têm em comum é que elas requerem uma pesquisa exaustiva no espaço de estados do modelo. Devido a natureza combinatória do problema, o tamanho desse espaço de estados pode ser, e normalmente é, exponencial em relação ao tamanho do sistema modelado. Este crescimento exponencial do espaço de estados é o fator limitante para aplicação de metodologias de verificação automática em grandes sistemas.

1.2 Por que OBDDs?

Muitos problemas em projetos de sistemas digitais, otimização combinatória, lógica matemática e inteligência artificial podem ser formulados em termos de operações sobre um domínio finito. Por meio de uma codificação binária dos elementos desse domínio, os problemas podem ser reduzidos a operações de valores Booleanos. Com uma representação simbólica de funções Booleanas, podemos expressar um problema numa forma geral. Resolvendo o problema generalizado, através de manipulação de funções Booleanas, fornecemos, então, as soluções para um grande número de instâncias específicas do problema. Portanto, um método eficiente para representar e manipular funções Booleanas simbolicamente pode conduzir a solução de uma grande classe de problemas complexos.

Os diagramas de decisão binários (BDDs-*Binary Decision Diagrams*) têm se mostrado uma ferramenta essencial no processo de verificação automática de sistemas. Quando é imposta uma ordenação às variáveis que descrevem o sistema, o BDD passa a ser denominado BDD ordenado (OBDD-*Ordered BDD*. Os OBDDs têm como característica essencial o fato de ser uma representação canônica de funções Booleanas. Assim, o processo de verificação de sistemas torna-se viável.

As operações em funções Booleanas podem ser implementadas através de algoritmos em grafos operando nos OBDDs. Em muitos domínios, os problemas podem ser expressos inteiramente em termos de operações com OBDDs, tal que uma completa

enumeração do espaço do problema (tabelas verdade, grafos de transição de estados, ou árvores de busca) nunca precise ser construída. Os pesquisadores têm resolvido problemas que não seriam possíveis de solucionar através das técnicas tradicionais usando OBDDs.

1.3 Uma Visão Geral e o Escopo da Tese

Como os requisitos de memória e tempo de processamento aumentam com o crescimento do tamanho de um OBDD, é importante manter seu tamanho tão pequeno quanto possível. Entretanto, o tamanho de um OBDD pode variar muito de acordo com a ordenação de suas variáveis. É comum encontrarmos OBDDs que têm tamanho exponencial para uma determinada ordenação, enquanto que para outra ordenação de suas variáveis, o OBDD apresenta um tamanho linear em relação ao número de variáveis da função. Assim, encontrar uma boa ordenação de variáveis para gerar um OBDD é essencial para manipulação de funções Booleanas através deste tipo de representação.

Obter a ordenação ótima das variáveis para gerar um OBDD é um problema NP-completo[5, 6, 9, 21]. Na literatura encontramos algumas heurísticas para manipular funções que apresentam representações em OBDD de grandes proporções.

Um método para obtenção de ordenações de variáveis consiste na utilização de uma busca local baseada em trocas de posição das variáveis. Isto é, a partir de uma representação inicial em OBDD da função, são realizadas trocas de posição na ordem entre variáveis, de modo a se obter uma representação mais compacta do OBDD. Na prática, este método produz boas ordenações, mas os resultados dependem da ordenação inicial. As ordenações iniciais são normalmente determinadas a partir de informações sobre a topologia do circuito. Portanto, os métodos de ordenação de variáveis que encontram uma boa ordenação inicial são muito necessários.

O problema de reordenação de variáveis para OBDDs é um problema típico de combinatória com um espaço muito grande de soluções. Além disso, segundo [21] não é conhecido nenhum método eficiente, para uma avaliação exata da ordenação de variáveis em relação ao tamanho do OBDD correspondente para uma ordenação. Mais precisamente, dado um OBDD para uma função f e uma ordenação de variáveis π, não existe nenhum procedimento eficiente que compute uma estimativa do tamanho do OBDD para a função f segundo a ordenação π. A única forma

exata de avaliação da ordenação é se construir o OBDD, o que pode ser realizado eficientemente apenas quando o tamanho do OBDD resultante for polinomial com relação ao número de suas variáveis. Este fato é um complicador para o problema, pois heurísticas que escolhem muitos candidatos a uma boa ordenação não podem evitar a construção de OBDDs para sua avaliação. Por esta razão, métodos de reordenação de variáveis em alguns casos, podem levar muito tempo para encontrar uma solução. Pode ainda não ser possível a aplicação destes, devido a restrições de tempo de computação e espaço de memória para a construção de um OBDD inicial.

O fato de não se conhecer um método eficiente que seja capaz de avaliar o tamanho de um OBDD, para uma dada ordenação de suas variáveis motivou o desenvolvimento deste trabalho. Nosso estudo é baseado nas idéias apresentadas em [4, 16]. Utilizando informações sobre a topologia de circuitos obtém-se um limite superior para o tamanho de um OBDD. Assim, a partir de funções Booleanas que expressam o comportamento de um circuito combinacional é construído um grafo de representação do circuito, cujo número de nós é da mesma ordem de grandeza do número de portas lógicas. Mais especificamente, a estimativa do tamanho do OBDD é feita sem que seja necessário a sua construção. Então, a partir de uma ordenação qualquer estima-se através de cortes, no grafo de representação do circuito, um limite superior para cada nível do OBDD. Utilizamos esta métrica juntamente com um método de reordenação de variáveis (*sifting*), de modo a obter uma melhor ordenação inicial. Desta forma, podemos verificar o desempenho da métrica com relação a seqüência de ordenação sugerida e o tamanho dos OBDDs resultantes. Isto vem viabilizar a aplicação de métodos de reordenação de variáveis cujas eficiências dependem de uma representação inicial em OBDD para sua execução.

Os resultados mostram que, para a maioria dos exemplos utilizados, a ordenação sugerida com a utilização da estimativa reduz o tamanho dos OBDDs. Em alguns casos esta redução é bastante significativa.

Este trabalho está organizado da seguinte forma. No Capítulo 2 abordamos os conceitos de funções Booleanas, circuitos lógicos e formas de representação de funções Booleanas. No Capítulo 3 apresentamos detalhadamente as formas de construção, as propriedades e características dos diagramas de decisão binária. Os efeitos da ordenação das variáveis em representações OBDDs, bem como alguns métodos de ordenação de variáveis são abordados no Capítulo 4. No Capítulo 5 estão descritas as idéias de Berman e McMillan[4, 16] que encontra um limite superior para o

tamanho de BDDs. Estas idéias forneceram a base para o desenvolvimento de nosso método. No Capítulo 6 descrevemos nosso método de avaliação de tamanhos de OBDDs e apresentamos os resultados obtidos. No Capítulo 7 apresentamos nossas considerações finais.

Capítulo 2

Conceitos Básicos

Neste capítulo são apresentados alguns conceitos básicos sobre circuitos lógicos e álgebra Booleana, que possibilitam a formalização do estudo e a manipulação de funções Booleanas.

2.1 Circuitos Lógicos

Os circuitos lógicos podem ser modelados como elementos de um sistema, que a partir de um conjunto de sinais de entrada, produzem um conjunto de sinais de saída. Tais conjuntos podem ser representados por um vetor de variáveis discretas. No caso particular de um sistema elétrico digital, seus sinais podem assumir somente dois valores.

A modelagem dos circuitos lógicos, que apresentam um conjunto finito de estados, pode ser realizado utilizando matemática discreta. A estrutura matemática empregada para modelar circuitos lógicos, de natureza binária, é a álgebra de Boole, desenvolvida pelo matemático George Boole, em 1850.

Nas próximas seções abordamos os tipos de descrição de circuitos lógicos e a síntese desses circuitos. A descrição de circuitos lógicos permite que seja obtida uma função Booleana que representa o circuito a ser sintetizado. O processo de síntese envolve a manipulação de funções Booleanas.

2.1.1 Tipos de Descrição de Circuitos Lógicos

Os circuitos lógicos podem ser descritos de três formas diferentes: funcional; comportamental e estrutural. Cada uma destas formas são abordadas a seguir.

Na descrição funcional, o circuito é representado pelo mapeamento das entradas nas saídas. Ou seja, o circuito é expresso em termos de uma correspondência entre

vetores binários de entrada e vetores binários de saída. Normalmente são utilizadas tabelas verdade para esta representação, como ilustrado na Figura 2.1. Nela as variáveis x, y e z representam os sinais de entrada, e a variável F representa o sinal de saída. Nesta tabela são representadas todas as possíveis combinações das variáveis de entrada e seu valor correspondente de saída.

x	y	z	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Figura 2.1: Exemplo de uma Tabela Verdade

A representação comportamental de um circuito lógico expressa o seu funcionamento em termos de equações derivadas da álgebra de Boole. As relações causais entre as variáveis de entradas e de saídas são estabelecidas por meio de um sistema de equações Booleanas, como pode ser visto na equação 2.1. Equações Booleanas estão definidas na seção 2.2.

$$F = \overline{xyz} + \overline{x}y + xyz \tag{2.1}$$

Uma descrição estrutural pode ser vista na Figura 2.2. Ela consiste de uma combinação de portas lógicas (G1 a G6), que representam o comportamento do circuito lógico, cujos sinais de entrada são representados pelas variáveis x, y, z, w e t, e os sinais de saída por F_1 e F_2.

2.1.2 Síntese de Circuitos

A síntese de um circuito combinacional é o processo de geração de uma combinação de dispositivos a partir de sua descrição inicial[12]. O ponto de partida para a síntese é, de forma mais geral, uma descrição funcional do circuito. Entretanto, é possível iniciar o processo de síntese a partir de qualquer um dos três tipos de descrição abordados anteriormente (funcional, comportamental e estrutural). A síntese de circuitos lógicos pode ser realizada de duas maneiras distintas, denominadas síntese de dois-níveis e síntese multi-nível.

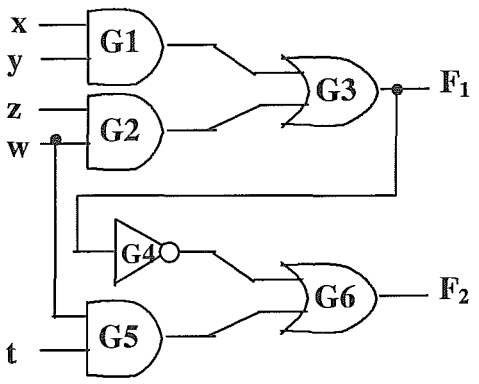


Figura 2.2: Circuito Lógico

A síntese de dois-níveis é realizada através da otimização de funções Booleanas na forma de soma de produtos. Uma soma de produtos é obtida com a utilização de dois níveis de portas lógicas. No primeiro nível são obtidos os produtos dos sinais de entrada (termos produto). No segundo nível, é realizada a soma dos termos produto, produzindo, então, o sinal de saída. O problema, do ponto de vista lógico, consiste na geração de uma soma de produtos com o menor número de termos produto.

Na síntese multi-nível ocorre a otimização de funções Booleanas, onde não há restrição quanto ao número de níveis lógicos que um sinal de entrada pode percorrer até gerar um sinal de saída.

2.2 Álgebra Booleana

A álgebra Booleana é uma estrutura abstrata, cujas operações são definidas sobre elementos genéricos e não em relação a um problema particular. Portanto, diversos problemas podem ser modelados por esta álgebra, desde de que sejam satisfeitos os axiomas básicos que a definem. A seguir abordamos, mais detalhadamente, a representação matemática da álgebra Booleana e seus axiomas básicos.

A álgebra Booleana é uma estrutura matemática composta pelos seguintes símbolos $B, \cdot, +, -, 0, 1$. O B é um conjunto que contém dois elementos distintos, 0 e 1[1]. Estes elementos representam, respectivamente, o mínimo e o máximo em B sobre os quais são definidas três operações: produto ("$\cdot$", conjunção, E lógico); soma ("+", disjunção, OU lógico) e negação ("−", complemento lógico).

Além disso, estas operações apresentam os seguintes axiomas:

[1]Os elementos 0 e 1 podem representar sinais elétricos individuais e podem corresponder à 1 *bit*

$$
\begin{aligned}
&a \cdot a = a \\
&a + a = a \\
&a \cdot (b \cdot c) = (a \cdot b) \cdot c \\
&a + (b + c) = (a + b) + c \\
&a \cdot b = b \cdot a \\
&a + b = b + a \\
&a + a \cdot b = a \\
&a \cdot (a + b) = a \\
&a \cdot (b + c) = (a \cdot b) + (a \cdot c) \\
&a + (b \cdot c) = (a + b) \cdot (a + c) \\
&a \cdot \overline{a} = 0 \\
&a + \overline{a} = 1 \\
&a \cdot 0 = 0 \\
&a + 1 = 1
\end{aligned}
$$

onde a, b e c representam variáveis qualquer arbitrárias que pode assumir o valor 0 ou 1.

O espaço Booleano pode ser n-dimensional. Neste caso $B^n = B \times B \times B \times \ldots \times B$ e as operações são estendidas *bit* a *bit*. Assim temos:

$$
\begin{aligned}
&0 = (0, 0, \ldots, 0) \\
&1 = (1, 1, \ldots, 1) \\
&a + b = (a_1 + b_1, a_2 + b_2, \ldots, a_n + bn) \\
&a \cdot b = (a_1 \cdot b_1, a_2 \cdot b_{12}, \ldots, a_n \cdot b_n)
\end{aligned}
$$

Através da álgebra Booleana podemos definir as chamadas funções Booleanas que abordamos a seguir.

2.2.1 Funções Booleanas

Uma função Booleana f é um mapeamento do tipo:

$$f : B^n \rightarrow B$$

onde B^n é o domínio e B a imagem da função.

O domínio de f é o conjunto de 2^n vetores binários correspondendo à n variáveis Booleanas, $X = (x_1, x_2, \ldots, x_{n-1}, x_n)$, e sua imagem B assume os valores 0 e 1.

Assim, uma variável Booleana define uma coordenada no espaço Booleano. Denominamos literal uma variável na forma direta (x) ou complementada ($\overline{x}$). Dessa forma, a imagem de um vetor de entrada X em f é definida por:

$$f(X) = f(x_n, x_{n-1}, \ldots, x_2, x_1)$$

Uma função Booleana cuja imagem é um único valor binário é usada para para modelar circuitos lógicos de uma única saída. Entretanto, no caso mais geral, circuitos lógicos possuem diversas saídas. Portanto, é mais conveniente a utilização de funções com múltiplas saídas para representar tais circuitos.

Uma função Booleana f com múltiplas saídas é um mapeamento $f : B^n \rightarrow B^m$ de um conjunto de vetores binários de entrada X em um conjunto de vetores binários de saída Z.

2.2.2 Formas de Representação de Funções Booleanas

Em muitas áreas do conhecimento, problemas tais como Inteligência Artificial, Combinatória, projetos de circuitos lógicos digitais entre outros podem ser modelados ou representados através de funções Booleanas . Dada a diversidade e a abrangência das áreas é importante se ter ferramentas eficientes capazes de representar e manipular funções Booleanas.

Por esse motivo foram desenvolvidos vários métodos para representar e manipular funções Booleanas, tais como as representações clássicas: tabelas verdades, mapa de Karnaugh[18] e soma de produtos canônicos. Entretanto, estes métodos têm se mostrado impraticáveis, pois para funções de n variáveis as suas representações têm tamanho que chega a 2^n. Operações como testar se para uma dada expressão existe atribuição de valores para as variáveis de entrada que leve a função ao valor 1, ou mesmo testar a equivalência entre duas expressões Booleanas são problemas NP-completos[10].

Na prática, utilizando-se melhores algoritmos de representação e manipulação de funções Booleanas, muitas vezes consegue-se evitar que a representação tenha tamanho exponencial. Algumas dessas representações são as expressões Booleanas e os diagramas de decisão que estão descritos a seguir.

Expressões Booleanas

A manipulação algébrica de funções Booleanas baseia-se no uso de variáveis para representar os elementos de um conjunto B. Operadores, variáveis e constantes são utilizados na formação de expressões Booleanas.

Pode-se definir uma expressão Booleana recursivamente como: uma constante $c \in B$ é uma expressão Booleana; qualquer variável $x \in X$ é uma expressão Booleana; se F, G, H são expressões Booleanas, então $(F + G)$, $(F \cdot G)$ e $\overline{H}$ também o são; não existe outras expressões além daquelas obtidas pela aplicação das regras citadas. A expressão 2.2 é um exemplo de uma expressão Booleana.

$$f = (x_1 \cdot x_2) + x_3 \cdot \overline{(x_1 + \overline{x_2})} \tag{2.2}$$

As expressões Booleanas denotam funções Booleanas. Tais funções podem ser obtidas pela avaliação da expressão para cada vetor do domínio da função.

As expressões Booleanas proporcionam uma forma mais eficiente de representação de funções do que as tabelas verdade. Muitos métodos de síntese e manipulação de funções baseiam-se em operações algébricas que visam evidenciar sub-expressões comuns, reduzindo, assim a complexidade da expressão global; isto é conhecido como forma fatorada da expressão.

Uma forma fatorada é uma expressão Booleana onde o operador negação só é aplicado às variáveis de entrada e não sobre uma sub-expressão. Para derivar uma forma fatorada de uma expressão Booleana é necessário aplicar o teorema de De Morgan, que está evidenciado a seguir.

Teorema de De Morgan

Em qualquer álgebra Booleana:

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

Expressões Booleanas e formas fatoradas são representações comportamentais que têm também uma interpretação estrutural se associarmos aos operadores lógicos as portas lógicas correspondentes, como na Figura 2.3:

A interpretação estrutural da função $f = (x_1 \cdot x_2) + x_3 \cdot \overline{(x_1 + \overline{x_2})}$ é apresentada na Figura 2.4.

Portas lógicas têm implementação como circuitos elétricos. Assim, um circuito elétrico representado como uma função Booleana pode ser visto como um grafo acíclico. Cada nó n_i está associado a uma função Booleana f_i e a uma variável y_i que a representa no circuito. Um arco do grafo é definido por um par ordenado de nós (n_i, n_f) onde n_i é o início e n_f é o fim do arco. Diz-se que n_i está conectado a uma entrada de n_f e que n_f está conectado a uma saída de n_i. Cada porta lógica

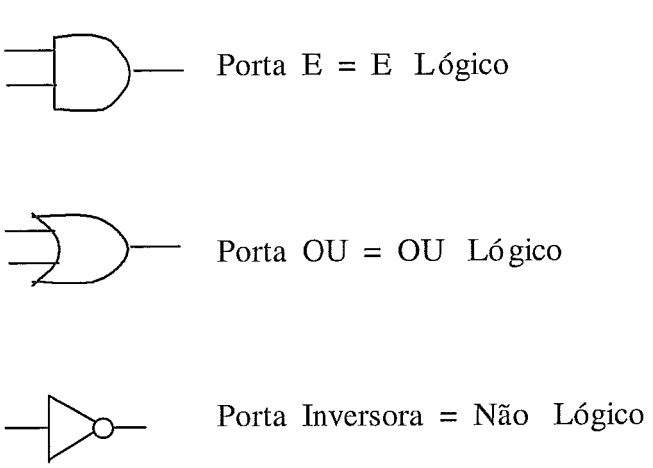


Figura 2.3: Portas Lógicas

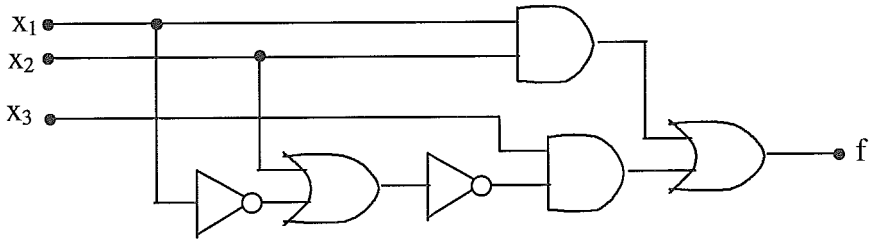


Figura 2.4: Representação Estrutural da Função

pode ser vista como um nó do grafo. As ligações entre as portas são equivalentes aos arcos do grafo. A Figura 2.5 ilustra o grafo que representa o circuito lógico mostrado na Figura 2.4.

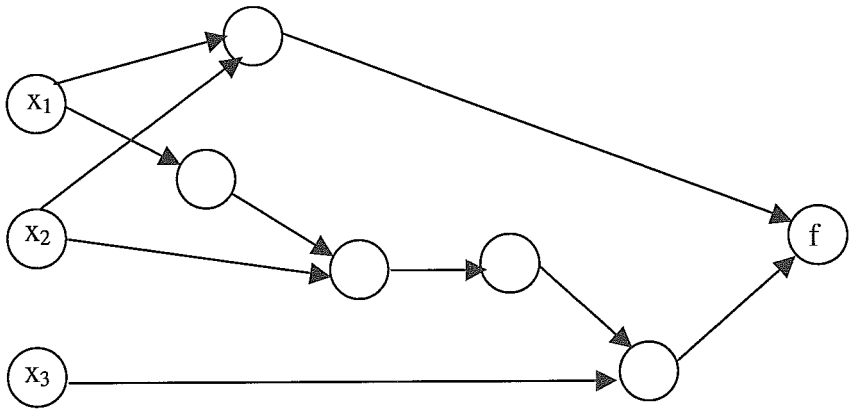


Figura 2.5: Grafo Correspondente a Representação Estrutural de uma Função

Qualquer grafo baseado em funções Booleanas pode ser considerado um modelo matemático de um circuito lógico genérico. Assim pode ser definido um sistema de equações Booleanas onde os nós intermediários realizam a associação de variáveis com expressões Booleanas.

Formas Normais

Formas normais são expressões Booleanas do tipo soma de produtos (forma normal disjuntiva) e produtos de somas (forma normal conjuntiva). Neste trabalho seguimos o enfoque tradicional que analisa o problema de lógica de dois-níveis apenas para o caso de soma de produtos. Isto porque o raciocínio realizado para este caso pode ser aplicado ao produto de somas devido ao teorema de De Morgan.

Uma forma normal particularmente importante é a forma normal canônica. A forma normal disjuntiva canônica é a soma de mintermos. Um mintermo é um produto onde estão presentes todas as variáveis da função (de forma direta ou complementada). Um mintermo é representado por m_i, onde i é o número decimal correspondente ao número binário definido pelo valor das variáveis. Considere $X = \{x_1, x_2, x_3\}$ como o conjunto de varáveis de uma função. Assim, $m_3 = \overline{x_3} \cdot x_2 \cdot x_1$ é o mintermo correspondente ao número binário 011.

Qualquer função Booleana pode ser expressa como uma soma de mintermos. Neste caso, indica-se em quais termos ela equivale a 1. Suponha, por exemplo, uma função f definida como $f = m_1 + m_3$. Esta função é equivalente a definição $f = \overline{x_3 x_2} x_1 + \overline{x_3} x_2 x_1$.

As formas normais baseadas em mintermos apresentam uma importante propriedade. Considere o problema de verificar se duas funções são idênticas. Naturalmente, basta verificar se elas assumem os mesmos valores nos mesmos termos. Se elas são expressas como somas de mintermos, a verificação é imediata: basta comparar diretamente os mintermos de ambas. Entretanto, isso não é válido para uma soma de produtos qualquer. De fato, a utilização de produtos com números de literais variados permite expressar a mesma função de diversas formas. Considere as seguintes expressões:

$$(x_1 \cdot x_2) + (\overline{x_2} \cdot x_3)$$
$$(x_1 \cdot x_2) + (\overline{x_2} \cdot x_3) + (x_1 \cdot x_3)$$
$$(x_1 \cdot x_2 \cdot x_3) + (x_1 \cdot \overline{x_2} \cdot x_3) + (\overline{x_1} \cdot \overline{x_2} \cdot x_3)$$

Qualquer uma dessas três expressões representa a mesma função. Portanto, existe apenas uma única expressão em soma de mintermos que descreve f. Quando o termo canônico é aplicado a um método de representação de funções Booleanas, ele indica que cada função tem uma única forma de representação naquele método. Assim uma função expressa na forma de mintermos pode ser dita uma forma canônica

de representação.

A canonicidade é uma propriedade importante para as formas de representação de funções Booleanas, pois permite a verificação de equivalência entre elas. Este problema é conhecido como verificação Booleana.

Diagramas de Decisão Binária

Os diagramas de decisão binária (BDDs - *Binary Decision Diagrams*) são grafos acíclicos que são compostos por dois tipos de nós, denominados nós intermediários e nós terminais. Cada nó intermediário está associado a até dois nós sucessores através de arcos rotulados 0 e 1. Os nós terminais não apresentam sucessores. Cada nó intermediário é associado a uma variável de decisão, enquanto que aos nós terminais estão associadas às constantes 0 e 1. Isto é, os valores que uma função pode assumir.

Para se achar o valor da função, deve-se percorrer o grafo, a partir da raiz, até um nó terminal, utilizando em cada nó intermediário, a aresta que corresponde ao valor da variável. O valor da função corresponde ao nó folha encontrado. A Figura 2.6 mostra uma representação em BDD de uma função Booleana. Os nós a, b e c são nós intermediários, os demais são nós terminais.

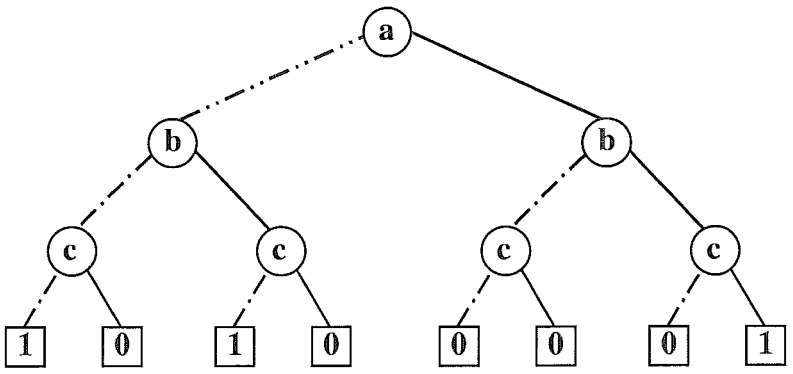


Figura 2.6: Diagrama de Decisão Binária

Este tipo de estrutura foi utilizado primeiramente por Lee[14] para representar circuitos eletrônicos. Mais tarde, Akers [1], [2] utilizou BDDs para análise, simulação e testes de circuitos digitais. Então Bryant[6] impôs uma restrição de ordenação às variáveis e o compartilhamento de subgrafos que representam funções equivalentes, o que resultou na criação dos OBDDs (*Ordered Binary Decision Diagram*) diagramas ordenados de decisão binária. Estes diagramas são abordados mais detalhadamente no Capítulo 3.

A utilização dos OBDDs proporcionou o desenvolvimento de diversas estruturas similares, generalizações ou restrições adequadas aos mais diversos tipos de problemas. Dentre estas, podemos citar o ROBDD (*Reduced Ordered Binary Decision Diagram*) ou também chamado forma canônica forte, proposto por Karplus[13]; o ZBDD (*Zero Supressed BDD*) proposto por Minato[17]e o MBDD (*Modified Binary Decision Diagram*) utilizado por Jacobi[12], onde foi incluído um terceiro nó terminal para denotar as não-especificações (*don't care*) da função representada.

Capítulo 3

Diagrama de Decisão Binária e sua Forma Canônica

Neste capítulo é detalhada a forma de representação de funções Booleanas através de grafos direcionados e acíclicos, conhecida como diagramas de decisão binária (BDD - *Binary Decision Diagram*). Além disso apresentamos suas características e propriedades que motivaram o desenvolvimento de uma variação desta representação chamada de diagrama ordenado de decisão binária(OBDD - *Ordered Binary Decision Diagram*). Esta é uma forma canônica para representação de funções Booleanas.

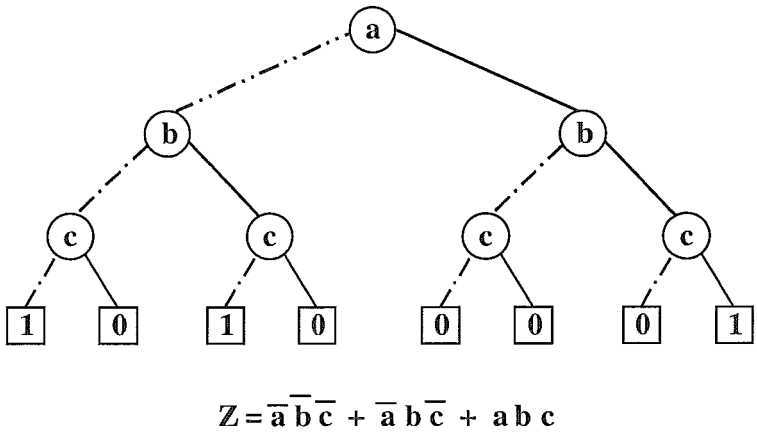
3.1 Representação por Árvore Binária de Decisão

Um diagrama binário de decisão pode ser visualizado como uma árvore binária. Os nós internos desta árvore são rotulados pelo nome das variáveis de entrada da função Booleana que se quer representar. E os nós terminais (nós folhas) são rotulados com 1 ou 0.

Dada uma representação de uma função Booleana através de uma árvore binária completa, é imediata a verificação da relação de equivalência com a tabela verdade da função que o BDD representa. Um exemplo pode ser visto na Figura 3.1. Para encontrar os mintermos que compõem a função representada, deve-se seguir todos os caminhos a partir da raiz até os nós terminais rotulados com 1. De cada nó saem duas arestas, uma à esquerda e outra à direita. Um nó representa uma variável, e as arestas representam uma atribuição de valor à variável. Convencionase que as arestas à esquerda (ilustradas por uma linha pontilhada) representam uma atribuição de 0 à variável, enquanto que as arestas à direita (ilustradas por uma linha cheia) representam uma atribuição de 1 à variável. Desta forma um caminho

da raiz a um nó terminal, rotulado com 1, corresponde a um mintermo da tabela verdade.

a	b	c	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



$$Z = \bar{a}\,\bar{b}\,\bar{c} + \bar{a}\,b\,\bar{c} + a\,b\,c$$

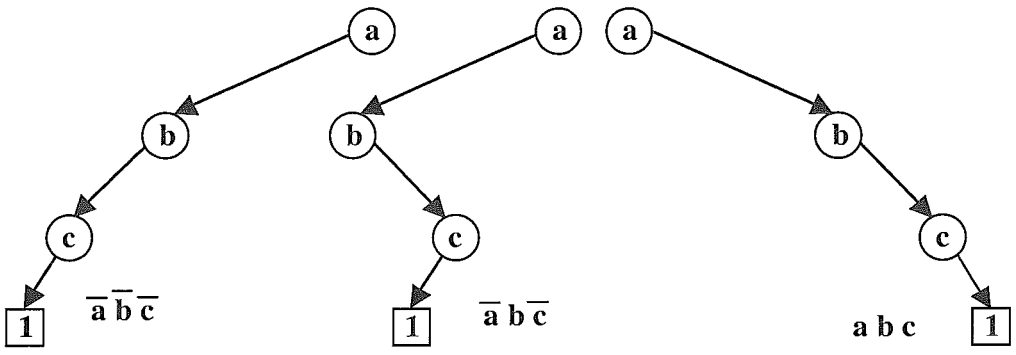


Figura 3.1: Construção da Árvore de Decisão Binária a partir de uma Tabela Verdade

3.2 Representação Compacta de BDDs

Um BDD pode ser representado através de uma árvore binária ou ainda através de um grafo direcionado acíclico como na Figura 3.2. No caso de um grafo direcionado acíclico, sub-expressões comuns podem ser representadas uma única vez. No exemplo 2 da Figura 3.2 o nó terminal é uma sub-expressão comum. Em geral, um nó interno também pode ter mais de um ancestral, representando uma sub-expressão comum. Veremos mais como obter este tipo de representação mais compacta na seção 3.7, aplicado a BDDs nos quais as variáveis seguem uma mesma ordenação, em todos os caminhos da raiz aos nós terminais. O fato de termos uma ordenação facilita a representação da função utilizando sub-expressões comuns.

Ao se percorrer todos os caminhos até atingir os nós terminais rotulados com 1, pode-se obter uma soma de produtos, onde os termos não são necessariamente

compostos por todas as variáveis de entrada. Da mesma forma que na árvore binária cheia, as arestas da direita de um BDD são representadas por uma linha cheia, e contribuem para a expressão com a variável associada ao nó na forma direta. Enquanto que as arestas da esquerda são representadas por uma linha pontilhada, e a variável associada apresenta-se na sua forma complementada na expressão. No exemplo 1 da Figura 3.2, a expressão Booleana é composta por dois termos produtos. Os caminhos da esquerda contribuem com os nós a e b. Portanto, eles contribuem com as variáveis complementadas ($\overline{a}\overline{b}$). No caminho da direita, os nós a e c contribuem com as variáveis associadas diretamente (ac).

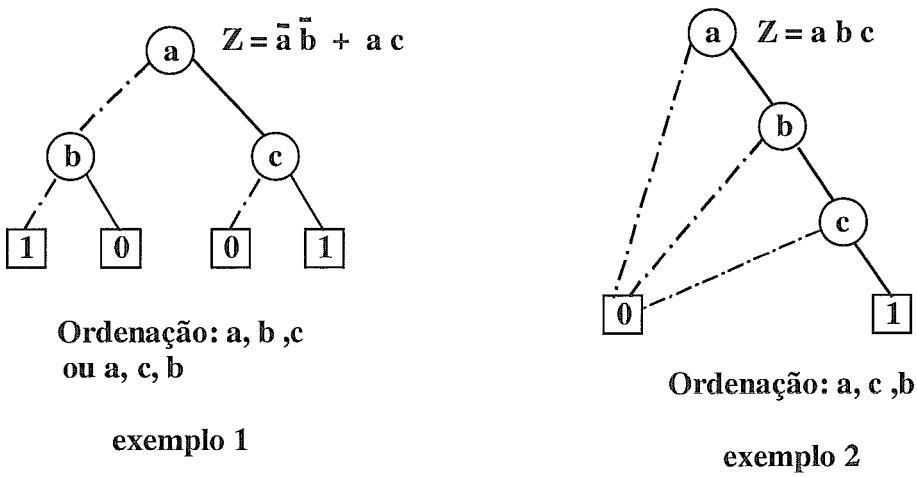


Figura 3.2: Diagrama de Decisão Binária

A ordenação nesta representação é obtida através do nível que cada variável ocupa no BDD, em relação às demais variáveis. No exemplo 1 da Figura 3.2, b e c estão no mesmo nível, e a está em um nível acima. Portanto, duas ordenações são possíveis: a, b, c ou a, c, b. No exemplo 2, desta mesma figura, a ordenação é obtida de forma mais simples, pois as variáveis ocupam níveis diferentes no BDD. A ordenação de um BDD é muito importante no seu processo de construção, influenciando de forma decisiva seu tamanho e também todo o processo de síntese.

Algumas das formas de construção de BDDs podem ser baseadas em tabelas verdade ou na expansão de Shannon. As próximas seções descrevem estas formas de construção.

3.3 Construção do BDD através de Tabela Verdade

Uma das formas de se construir um BDD é através da tabela verdade da função que se quer representar. Como mencionado, esta relação é direta.

Para construir o BDD devemos primeiro escolher uma ordenação para ele e rearrumar a tabela verdade com as variáveis na mesma seqüência. Em seguida, seleciona-se a raiz de acordo com a ordenação e para cada mintermo se o valor da variável do nó na tabela for 0, o nó terá um filho para a esquerda, caso contrário, um filho para a direita. O mesmo procedimento deve ser aplicado para todas as variáveis, obedecendo a ordenação definida. Para terminar um ramo da árvore, é necessário acrescentar um nó terminal com rótulo 1. Este procedimento deve ser repetido para todos os mintermos da tabela verdade. No final de todo o procedimento, alguns nós podem não ter duas arestas (direita e esquerda). Então pode-se atribuir a eles um nó terminal com rótulo 0.

A Figura 3.3 ilustra o procedimento de construção de um BDD através da tabela verdade. A ordenação escolhida foi a, b e c. A parte superior da figura mostra a construção dos três caminhos da árvore relativos aos mintermos com valor 1. A parte inferior mostra a árvore cheia.

3.4 Construção do BDD através da expansão de *Shannon*

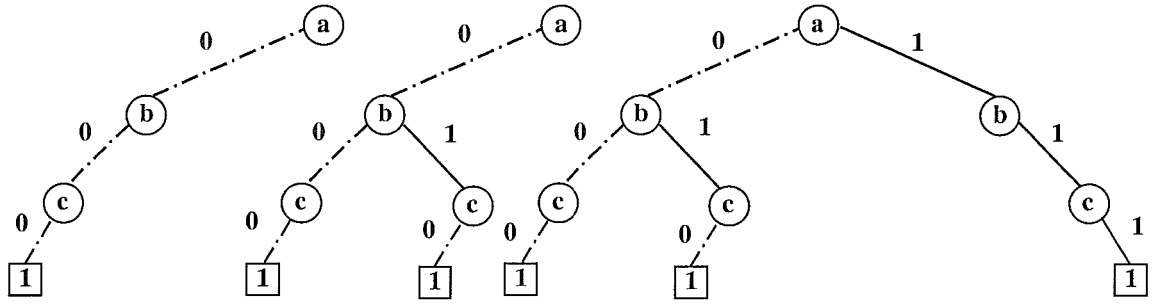
Uma forma alternativa de construção de um BDD utiliza a idéia da expansão de *Shannon*[6]. A expansão de *Shannon* de uma função f sobre uma variável x_i é:

$$f = x_i \cdot f|_{x_i=1} + \overline{x_i} \cdot f|_{x_i=0} \tag{3.1}$$

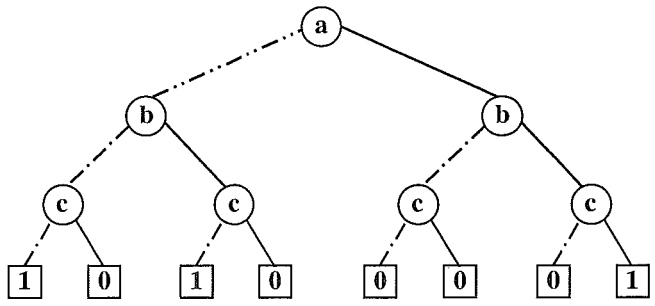
Examinando-se um nó qualquer de um BDD, observa-se que a expansão expressa em 3.1 se aplica perfeitamente. $f|_{x_i=1}$, corresponde a sub-árvore direita. De modo análogo, $f|_{x_i=0}$ corresponde a sub-árvore esquerda.

Considere a mesma função da Figura 3.1. A partir dela, obtemos a função $f = \overline{a}(\overline{b}\overline{c} + b\overline{c}) + abc$ aplicando-se a expansão de *Shannon* para a raiz do BDD.

Para se construir o BDD a partir da expansão de *Shannon*, como mostrado na Figura 3.4, escolhe-se a variável que, segundo a ordenação, será a raiz do BDD (nó a).



a	b	c	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



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

Figura 3.3: Construção de BDD utilizando Tabela Verdade

Calcula-se, então, o valor de duas novas funções f_{dir} e f_{esq} a partir da função original, atribuindo-se à variável da raiz os valores 1 e 0, respectivamente. As funções f_{dir} (bc) e f_{esq} $(\overline{b}\overline{c} + b\overline{c})$ estão associadas às sub-árvores direita e esquerda da raiz. Estas duas novas funções dão origem a duas novas sub-árvores, que são ligadas à raiz do BDD. Para cada uma delas escolhe-se uma variável para ocupar a raiz da sub-árvore. Esta variável deverá ser a próxima na seqüência da ordenação, que esteja presente na função. Deve-se ter cuidado nesta escolha, pois algumas variáveis são eliminadas no processo de avaliação das novas funções. Escolhida a variável para ocupar o nó raiz da sub-árvore, repete-se o processo de forma recursiva, calculando-se as funções f_{dir} e f_{esq} relativas àquele nó. O processo termina quando o resultado da avaliação de uma das funções (f_{dir} ou f_{esq}) for 0 ou 1. Neste caso, não é necessário calcular novas funções para aquela sub-árvore. Cria-se então um nó terminal. Quando todos as sub-árvores atingirem um nó terminal o BDD terá sido construído.

Além de se extrair de um BDD os mintermos da função Booleana associada na forma de soma de produtos, também é possível obtermos uma forma fatorada para a função, este ponto é abordado a seguir.

3.5 Extração de uma expressão Booleana fatorada de um BDD

Para se extrair uma expressão Booleana fatorada a partir de um BDD, deve-se percorrer o BDD em pré-ordem até o nível 0 (raiz do BDD). A função relativa à sub-árvore enraizada por um nó v qualquer pode ser escrita como $f(v) = v \cdot f(r_d) + \overline{v} \cdot f(r_e)$, onde r_d e r_e são raízes das sub-árvores à direita e à esquerda de v.

Na Figura 3.5 a função $f = a(b + \overline{b}c)$ é reconstruída de acordo com a expansão de *Shannon*.

É importante notar que pode não se obter a mesma função $f = a(b + c)$, que foi usada para construção do BDD, quando extraímos uma função fatorada percorrendo o BDD de baixo para cima. Isto mostra que a representação de funções Booleanas através de BDDs, ainda apresenta problemas para testes e manipulação das mesmas. Questões como teste de satisfatibilidade, isto é, se existem valores associados as variáveis de entrada que levem a função ao valor 1 ou teste de equivalência entre funções Booleanas, apresentam soluções do tipo NP-Completa ou co-NP-Completa[10]. O desejo de obter uma representação canônica para funções Booleanas motivou Randal E. Bryant[6] a desenvolver a estrutura chamada OBDD(*Ordered Binary Decision Diagram*) diagrama ordenado de decisão binária, que é descrita a seguir.

3.6 Diagramas Ordenados de Decisão Binária - OBDDs

Um BDD representa uma função Booleana como um grafo acíclico e direcionado. Para um OBDD é imposta uma ordenação total (menor que - $<$) sobre o conjunto de variáveis. Em um OBDD as variáveis aparecem em todos os caminhos da raiz aos nós terminais respeitando a ordem $<$. Na árvore de decisão ilustrada na Figura 3.1, por exemplo, as variáveis são ordenadas como $a < b < c$. Em princípio, a ordenação das variáveis pode ser feita arbitrariamente. Porém, na prática a seleção de uma boa ordenação é fator crucial para a construção e manipulação eficiente de funções. Este aspecto é discutido mais detalhadamente no capítulo 4.

Além da restrição de ordenação sobre as variáveis da função a ser representada pelo OBDD, foram definidas em[6] algumas regras de transformação para esta estrutura, que reduz seu tamanho sem, contudo, alterar a função. A representação

reduzida de um OBDD possui a propriedade de ser uma representação canônica de uma função Booleana. Bryant demonstrou em[6] que dado uma ordenação de variáveis o OBDD reduzido que representa uma função qualquer é único. Portanto, o uso de tais estruturas simplificam os testes de funções Booleanas, tais como equivalência e satisfabilidade.

3.7 OBDD Reduzido

Para descrever um OBDD na sua forma reduzida, é necessário definir três regras de transformação para estes grafos, que não alteram a função representada. Cada uma destas regras estão definidas a seguir.

A regra de remoção de nós terminais duplicados corresponde a eliminação de todos os nós terminais duplicados, deixando apenas um nó rotulado com 1 e outro rotulado com 0. Em seguida, redireciona-se todos os arcos que antes chegavam aos nós terminais removidos para os nós terminais que permaneceram no grafo. A Figura 3.6 ilustra esta transformação.

Na regra de remoção de nós não-terminais duplicados deve-se considerar: se dois nós não-terminais u e v estão associados à mesma variável, isto é, $var(u) = var(v)$; e se possuem ainda, os mesmos filhos à esquerda e à direita, $esq(v) = esq(u)$ e $dir(v) = dir(u)$. Se estas condições são atendidas, então, um dos nós pode ser eliminado. Da mesma forma que na regra de remoção de nós não-terminais, redireciona-se todos os arcos que chegavam ao nó eliminado para o nó que permaneceu no grafo. Esta transformação pode ser vista na Figura 3.7.

Considerando o grafo obtido na Figura 3.6 podemos identificar que os dois nós destacados em negrito no grafo superior da Figura 3.7 possuem as condições para a remoção. Assim, podemos gerar o grafo mostrado na parte inferior da figura.

Na regra de remoção de nós redundantes, se um nó v tem filho à esquerda igual ao filho à direita, isto é, se $esq(v) = dir(v)$, este é um nó redundante. Portanto, pode ser removido. Novamente após a remoção do nó, redireciona-se os arcos que chegavam ao nó removido para seu filho. A Figura 3.8 ilustra esta transformação.

Considerando agora, o grafo resultante da Figura 3.7, destacamos, em negrito, no grafo superior da Figura 3.8 os nós redundantes que podem ser removidos. Assim, após a remoção obtemos o grafo reduzido mostrado na parte inferior da figura.

Partindo de qualquer BDD que satisfaça a propriedade de ordenação, pode-se re-

duzir seu tamanho aplicando-se repetidamente as regras de transformação descritas. Essas regras devem ser aplicadas em cada nível do grafo, a partir dos nós terminais até a raiz, o que garante a forma canônica da representação. O termo OBDD é usado para se referir ao grafo com a máxima redução que obedece a alguma ordenação. Em geral, as regras de transformação devem ser aplicadas repetidamente, já que a cada transformação poderão surgir novas oportunidades de aplicação de qualquer uma das três regras. Este processo termina com a obtenção da função na sua forma canônica.

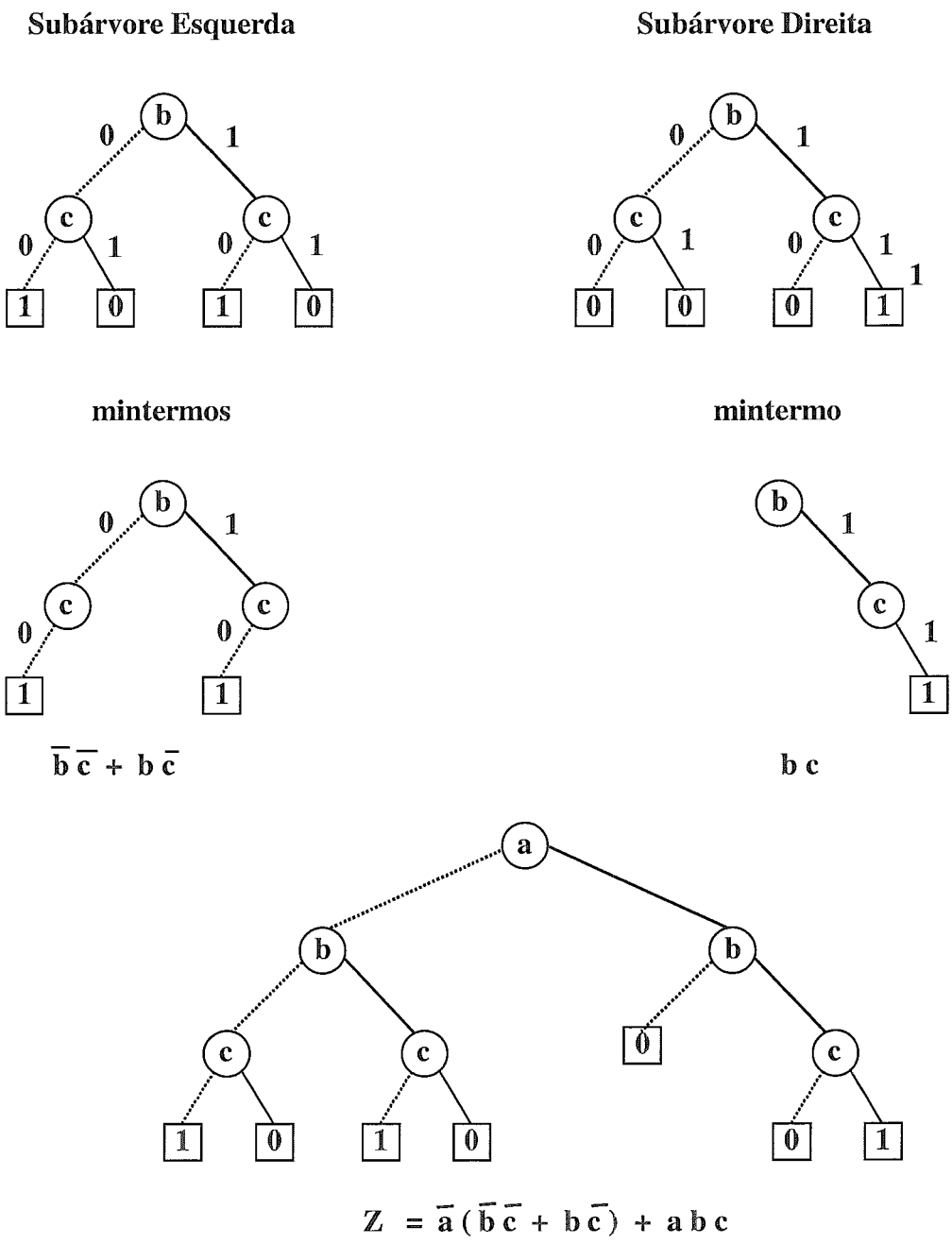


Figura 3.4: Construção de BDD utilizando Expansão de *Shannon*

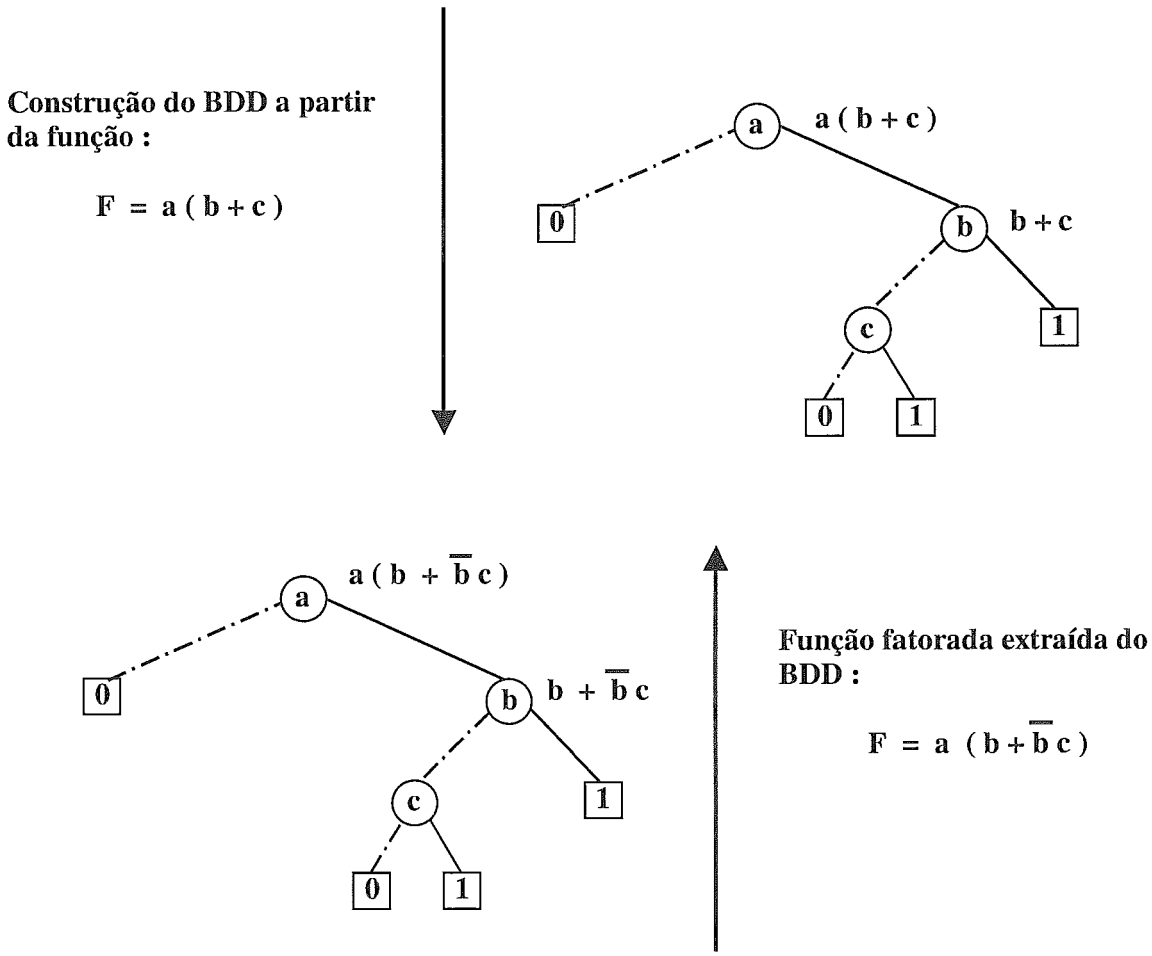


Figura 3.5: Expressão Fatorada

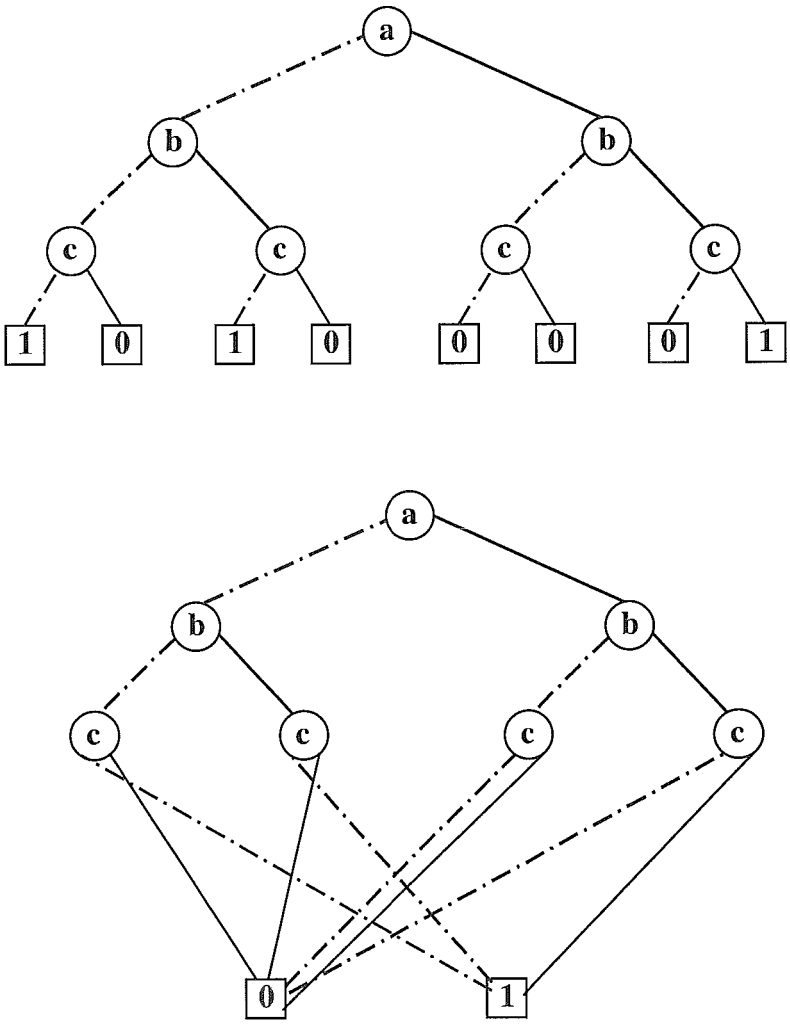


Figura 3.6: Remoção de Nós Terminais Duplicados

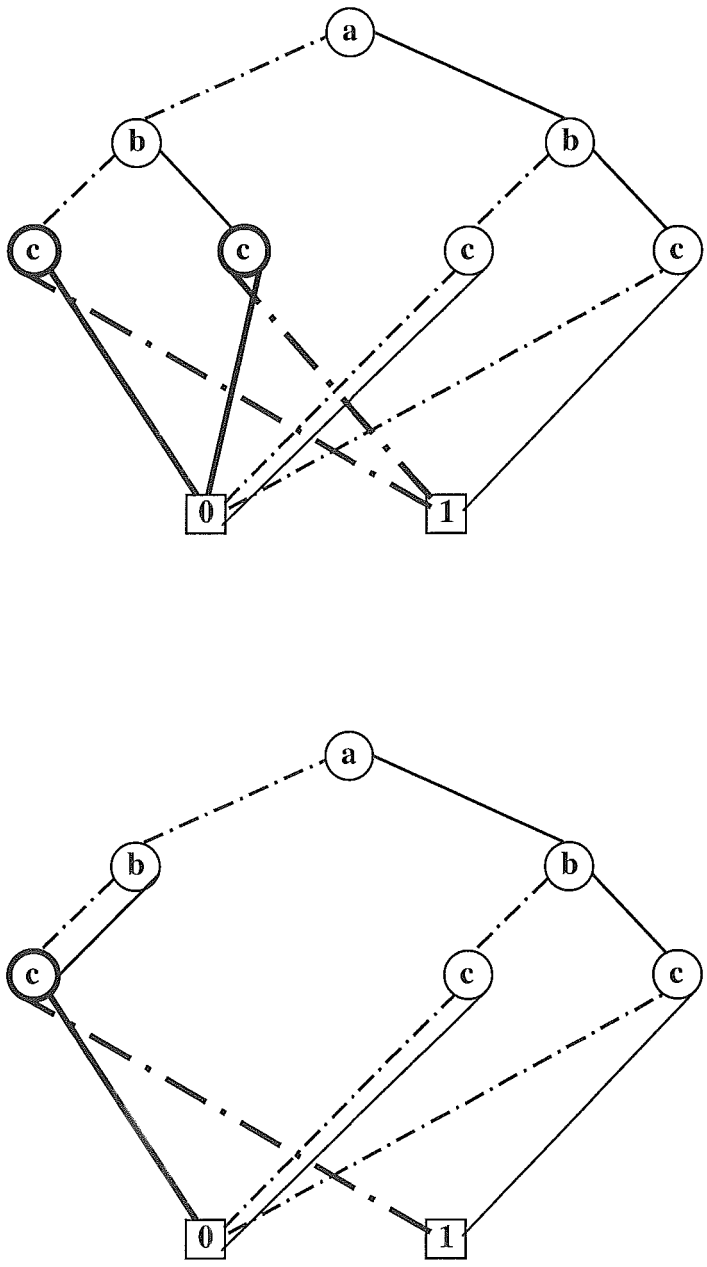


Figura 3.7: Remoção de Nós Não-Terminais Duplicados

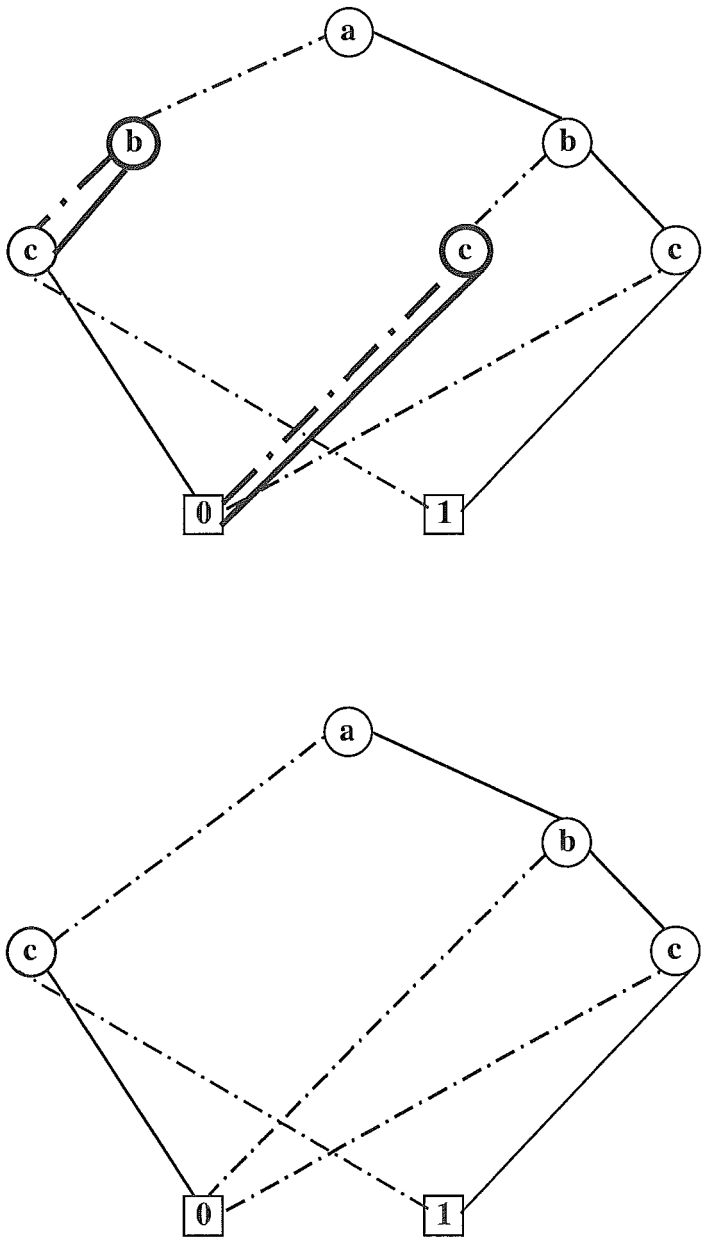


Figura 3.8: Remoção de Nós Redundantes

Capítulo 4

Efeitos da Ordenação de Variáveis em OBDDs

Neste capítulo é mostrada a influência da ordenação das variáveis de entrada sobre o tamanho do OBDD associado a uma função Booleana. Além disso, são expostos alguns métodos de ordenação de variáveis que visam a obtenção de OBDDs compactos.

4.1 Ordenação de Variáveis

A forma e o tamanho de um OBDD depende da ordenação de suas variáveis. A Figura 4.1 mostra dois OBDD's para a função Booleana representada pela expressão $a_1b_1 + a_2b_2 + a_3b_3$. No grafo à esquerda da figura, as variáveis obedecem a ordenação $a_1 < b_1 < a_2 < b_2 < a_3 < b_3$. Já no grafo à direita, a ordenação utilizada é $a_1 < a_2 < a_3 < b_1 < b_2 < b_3$.

Se generalizarmos essa função para $a_1, \ldots, a_n$ e $b_1, \ldots, b_n$, obtemos a expressão $a_1b_1 + \ldots + a_nb_n$. A primeira ordenação corresponde a $a_1 < b_1 \ldots < a_n < b_n$, resultando num OBDD com $2n$ nós não-terminais, um para cada variável. Por outro lado, a segunda ordenação $a_1 < \ldots < a_n < b_1 < \ldots < b_n$, resulta num OBDD com $2(2^n - 1)$ nós não-terminais.

Este exemplo mostra claramente que a ordenação das variáveis de um OBDD pode influenciar fortemente no seu tamanho. Note que neste exemplo, o tamanho de um OBDD varia de linear a exponencial no número de nós não-terminais com a mudança da ordenação de suas variáveis de entrada. Assim, em muitas aplicações práticas, a escolha de uma ordenação pode representar a diferença entre a viabilidade ou não da manipulação, ou mesmo da construção do OBDD.

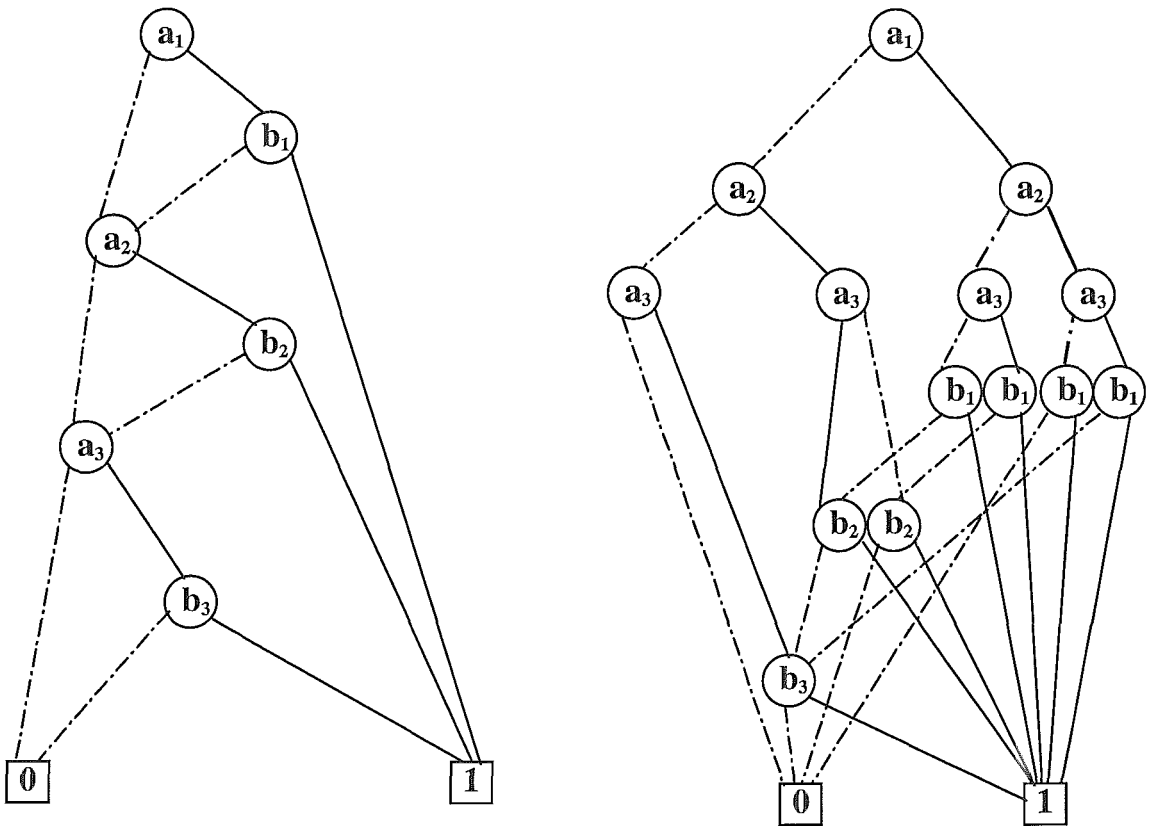


Figura 4.1: Efeito da Ordenação de Variáveis

As variações no tamanho de um OBDD, devido as diversas possibilidades de ordenação, dependem também da natureza da função a ser manipulada. Para certas funções, uma simples inversão na ordem de suas variáveis de entrada resulta numa grande variação no tamanho do OBDD que a representa. Por outro lado, OBDD's que representam funções simétricas, onde o valor da função depende apenas do número de argumentos iguais a 1, são em geral, insensíveis às variações na ordenação de suas variáveis de entrada [7]. Existem ainda, funções Booleanas, tais como as que representam multiplicação de inteiros, para as quais não é possível se obter uma representação em OBDD da função, cujo tamanho seja polinomial, qualquer que seja a ordenação considerada para suas variáveis de entrada [6].

Fica claro então, que existe uma grande dificuldade em se estabelecer, automaticamente, para qualquer função dada, a melhor ordenação para sua representação em OBDD. No entanto, quando é possível encontrar uma boa ordenação, que seja capaz de resultar em uma representação compacta do OBDD da função, este tipo de representação se mostra muito mais vantajoso que os demais tipos de representação existentes, para funções Booleanas. Assim, é importante o estudo de métodos de

ordenação que resultem em representações compactas de OBDD's.

Nas próximas seções são descritas algumas características de representações compactas de OBDD's para funções Booleanas.

4.1.1 Propriedades Práticas de Ordenação de Variáveis

Na maioria das funções utilizadas na prática, foram observadas empiricamente [17] duas propriedades que, normalmente, estão presentes em representações compactas de seus OBDD's. Essas propriedades dizem que:

- as variáveis que estão mais relacionadas entre si e que apresentam alguma independência em relação as demais variáveis da função, apresentam-se juntas nas ordenações. Por exemplo, no caso da função Booleana que representa um somador de *n-bits*, a ordenação que resulta na melhor representação OBDD, é aquela em que as variáveis de mesmo índice estão ordenadas em pares. Supondo um somador com n=2 a melhor ordenação é a_1, b_1, a_2, b_2;

- as variáveis que exercem grande influência na geração da saída de uma função apresentam-se mais próximas à raiz do OBDD. Por exemplo, quando é possível distingüir entre variáveis de controle e variáveis de dado, numa função de um circuito digital, em geral, a melhor ordenação é aquela onde as variáveis de controle estão em níveis acima aos das variáveis de dado.

Assim, quando é possível encontrar uma ordenação para as variáveis de uma função, que satisfaça ambas as propriedades, normalmente, obtêm-se uma representação OBDD compacta para a função. Entretanto, na maioria dos casos, é muito difícil combinar, ou mesmo identificar tais propriedades. Assim, são necessários estudos e o desenvolvimento de métodos que forneçam boas ordenações para as estruturas OBDD's.

4.1.2 Características dos Métodos de Ordenação de Variáveis

Devido a grande aplicação das funções Booleanas em várias áreas, sobretudo em síntese de circuitos digitais, foram desenvolvidos diversos trabalhos em ordenação de variáveis para estruturas OBDD's. Tais trabalhos baseiam-se nas mais diferentes idéias, como informações sobre as topologias de circuitos, informações funcionais das

variáveis envolvidas, informações comportamentais da função a ser representada, entre outras. Entretanto, segundo Butler[8], não se conhece nenhum método que seja capaz de obter, de forma eficiente, boas ordenações para todo e qualquer circuito ou função Booleana.

Em geral, para uma dada função de n variáveis $f(x_1, \ldots, x_n)$ existem $n!$ ordenações que podem ser escolhidas. De fato, determinar a ordenação ótima das variáveis é um problema NP-completo[6, 5, 9, 21]. O melhor algoritmo conhecido para determinar a ordenação ótima das variáveis, é baseado em programação dinâmica e apresenta complexidade de $O(n^2 3^n)$. Porém, na prática, só é adequado para funções com um número reduzido de variáveis[9].

Existem basicamente duas abordagens distintas para o desenvolvimento de métodos de ordenação de variáveis. Os métodos que buscam sempre a melhor ordenação possível, sem contudo apresentar nenhum compromisso com o tempo necessário para a busca da solução, e aqueles que buscam uma solução, que nem sempre representa a melhor, porém, o fazem dentro de um período de tempo pré-estabelecido.

Os métodos desenvolvidos para ordenação de variáveis, que resultam numa representação OBDD compacta, também podem ser diferenciados pelo tipo de abordagem adotada para o problema. Normalmente, pode-se identificar dois tipos distintos de abordagem para o problema de redução de estruturas OBDD's:

- encontrar uma ordenação apropriada antes da geração do OBDD;
- redução do tamanho do grafo, através da reordenação das variáveis de entrada, para um OBDD gerado segundo uma ordenação qualquer de suas variáveis.

A grande desvantagem apresentada pela reordenação de variáveis está no fato de que muitas vezes, não se consegue gerar o OBDD para uma dada ordenação inicial. Esta ordenação pode impor um número tão elevado de nós a estrutura do grafo, que em termos práticos, surgem restrições como capacidade de memória e tempo de processamento, que impossibilitam a construção do grafo. Todavia, existem aplicações, tais como síntese lógica, que requerem muitas operações lógicas após a construção do OBDD inicial. Nestes casos, a reordenação de variáveis é uma técnica importante, pois as operações lógicas podem aumentar consideravelmente a estrutura do grafo resultante.

As técnicas de ordenação ou reordenação de variáveis podem envolver uma busca local sendo baseadas na troca de posição entre variáveis adjacentes. Os algoritmos baseados em busca local apresentam a desvantagem de algumas vezes gastar muito tempo de computação até que se consiga uma solução. Caso a ordenação utilizada no grafo inicial esteja bastante afastada de uma solução considerada razoável, são necessárias várias trocas de posição entre variáveis, até que se consiga uma solução satisfatória. Além disso, existe a possibilidade do processo encontrar um mínimo local, que pode estar muito longe de uma melhor solução, mascarando desta forma a possibilidade de se obter uma melhor ordenação, que reduza ainda mais o tamanho do grafo. Entretanto, estes algoritmos nunca produzem resultados piores do que os iniciais [17]. Assim, são algoritmos apropriados para serem utilizados como última etapa do processo de ordenação, em conjunto com algum outro método ou algoritmo.

Nas próximas seções são apresentados alguns métodos de ordenação de variáveis, tais como *simulated annealing*, largura mínima, *sifting* e *interleaving*. Nestes métodos estão implementadas algumas das principais idéias utilizadas nos métodos de ordenação de variáveis existente.

4.2 *Simulated Annealing*

O método *simulated annealing*[3] é baseado na idéia de trocas aleatórias das posições das variáveis de entrada de um OBDD. Ele implementa uma analogia ao processo físico de fundição de materiais. Quando submetemos os materiais a altas temperaturas, suas moléculas ficam em estado de completa desordem e a medida que a temperatura decresce as moléculas buscam um estado de equilíbrio.

Na implementação do *simulated annealing* é necessário fixar um valor inicial para a temperatura do sistema. A cada nova iteração do algoritmo a temperatura deve decrescer de uma taxa, pré-estabelecida, até atingir um valor estabelecido como ponto de congelamento. O ponto de congelamento é a condição de parada do algoritmo.

No contexto de geração de um OBDD, o estado do sistema é representado por uma ordenação de variáveis. Associado a este estado deve haver uma função de custo, que represente o tamanho do OBDD, para aquela ordenação. Variando-se, então, a ordenação obtêm-se novos custos. O algoritmo tenta melhorar o custo, assim como acontece no processo físico, onde a medida que a temperatura decresce

é reduzido o nível de energia do sistema de modo a promover uma organização perfeita entre as moléculas do material.

A partir de uma ordenação inicial, o algoritmo promove mudanças aleatórias nas posições de duas variáveis. Isto gera um novo custo. Então, é computada a variação de custo (Δc). Caso esta variação seja negativa, isto representa que houve uma redução no tamanho do OBDD. Então, esta nova ordenação passa a ser a ordenação atual do sistema. Por outro lado, se a variação Δc for positiva, ela pode ou não ser aceita e indica que houve um crescimento da estrutura do OBDD. A probabilidade de aceitação desta ordenação é definida por $e^{-\Delta c/t}$, onde t é a temperatura atual do sistema. Caso a nova ordenação não seja aceita, recupera-se a ordenação anterior. A probabilidade de se aceitar uma ordenação que gere $\Delta c \geq 0$, dificulta que a função de custo fique confinada a um mínimo local.

Aceitando trocas ruins, isto é aquelas que aumentam o tamanho do OBDD, o algoritmo espera que em passos futuros, as trocas subseqüentes resultem numa ordenação mais favorável. É importante lembrar que este é um método heurístico. Inicialmente, no algoritmo de *simulated annealing* quando as temperaturas estão mais altas, a probabilidade de se aceitar trocas ruins é maior. Entretanto, conforme a temperatura decresce, o sistema tende a buscar soluções cada vez mais próximas de um mínimo.

O algoritmo possui uma condição de equilíbrio para cada temperatura, cuja finalidade é definir uma ordenação. Para o sistema passar para uma temperatura mais baixa, deve ocorrer um número fixo pré-estabelecido de iterações (mudança na ordenação) sem que uma nova ordenação tenha sido obtida. Isto faz com que o algoritmo encontre uma melhor solução entre uma vizinhança de soluções (um mínimo local).

4.3 Método da Largura Mínima

O método da largura mínima, proposto em [17], é um algoritmo guloso, que busca a melhor ordenação sobre um ponto de vista global. Porém, depende de uma função de custo que avalia o efeito do posicionamento de uma variável na ordenação geral, no tamanho do grafo a ser formado. Este método é baseado em trocas de variáveis e depende de uma representação inicial de um OBDD para sua aplicação. Antes da descrição do algoritmo são necessários alguns esclarecimentos sobre a função de

custo. No método descrito nesta seção, n denota o número de variáveis de entrada. E cada variável é identificada por um índice como $x_1, x_2, \ldots, x_n$, onde a variável de maior índice está localizada na posição mais alta do grafo (próximo a raiz). A seguir descrevemos como definir a função de custo e o algoritmo propriamente dito.

4.3.1 A Função de Custo - Largura de OBDD

Define-se largura de OBDD em um determinado nível k, indicada por W_k, como o número de arestas que cruzam a seção do OBDD entre x_k e x_{k+1},onde as arestas que apontam para o mesmo nó contribuem apenas como uma aresta, no número total de arestas que cruzam a seção. A Figura 4.2 ilustra este conceito. Na seção entre os nós x_2 e x_1 na figura, embora existam quatro arestas cruzando esta seção, a largura da seção é computada como $W_1 = 3$, pois existem duas arestas que apontam para o mesmo nó x_1.

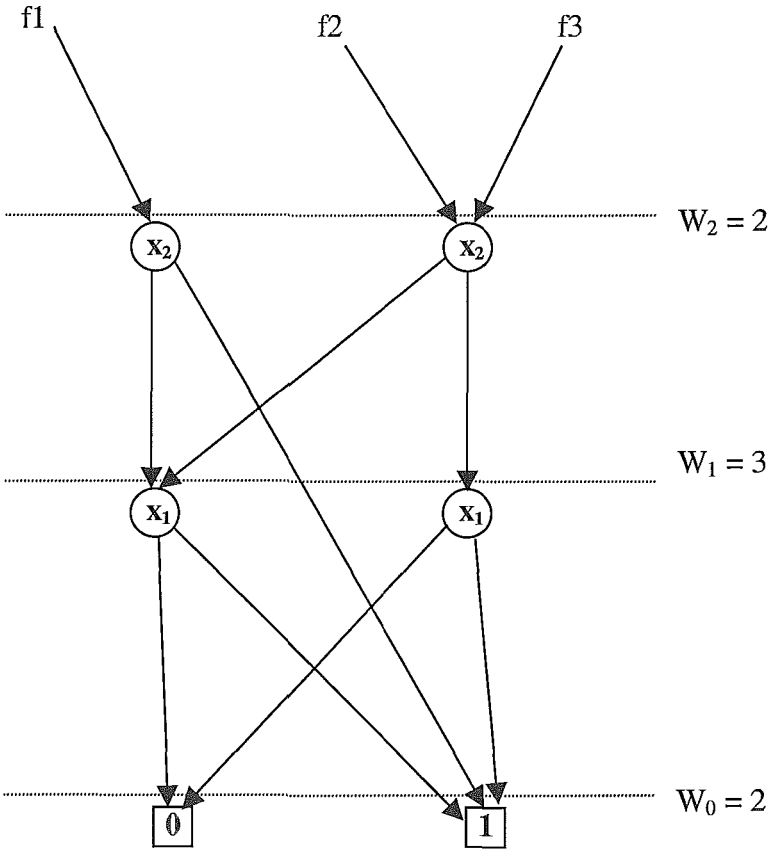


Figura 4.2: Largura de OBDD

Em [17] é apresentado o seguinte teorema que define a função de custo baseado na largura do OBDD.

Teorema 1 *A largura de OBDD, W_k, é constante para qualquer permutação entre $x_1, \ldots, x_k$ e qualquer permutação $x_{k+1}, x_{k+2}, \ldots, x_n$.*

Prova: Cada aresta que cruza a seção do grafo entre x_{k+1} e x_k representa uma sub-função, obtida associando um vetor de variáveis Booleanas $\{0,1\}^{n-k}$, às variáveis de entrada pertencentes ao caminho da raiz até a aresta em questão. Os OBDDs apresentam a propriedade que arestas representando a mesma sub-função apontam para o mesmo nó[6]. Portanto, W_k exprime o número de todas as sub-funções obtidas associando-se todos os padrões do vetor $\{0,1\}^{n-k}$ às variáveis de entradas $x_{k+1}, x_{k+2}, \ldots, x_n$.

Todas as sub-funções, que são obtidas pela associação de todos padrões do vetor, são unicamente representadas no OBDD, com a mesma ordenação de variáveis $x_1, x_2, \ldots, x_k$. Para qualquer permutação entre estas variáveis, o número de sub-funções é constante, pois as sub-funções são ainda representadas de forma única. Logo W_k nunca varia para qualquer permutação entre $x_1, x_2, \ldots, x_k$. ■

4.3.2 O Algoritmo

Inicialmente é escolhida uma variável do conjunto de variáveis de entrada. Esta variável é colocada na posição mais alta do OBDD. Ela será, então, a variável x_n. Em seguida, outra variável é escolhida, no conjunto das variáveis restantes, e então é fixada na segunda posição mais alta do grafo. Assim esta será a variável x_{n-1}. Da mesma forma, todas as variáveis são previamente escolhidas e fixadas a partir das posições mais altas até as mais baixas do grafo.

Quando é escolhido um $x_k (1 \leq k \leq n)$, as variáveis com índices superiores a k já estão fixadas em suas posições no grafo. Logo, a parte superior do grafo não mais se altera. A escolha de x_k afeta apenas a parte do grafo inferior a ela. Portanto, a tarefa a cada passo é escolher um x_k, de modo que a parte inferior a x_k no grafo seja reduzida. Isto é, a variável x_k é escolhida de modo que W_{k-1}, a largura de OBDD na seção entre x_k e x_{k-1}, seja mínima, como na Figura 4.3 . No caso de haver duas variáveis candidatas que forneçam a mesma largura W_{k-1}, a variável escolhida é a que ocupa a posição mais alta na ordenação inicial. Entretanto, é difícil prever a priori o quanto a parte inferior do grafo será reduzida, pois as variáveis que serão posicionadas abaixo de x_k não foram ainda fixadas. Assim, para se evitar um procedimento de *back tracking* para reposicionamento, x_k é escolhido segundo uma avaliação da função de custo.

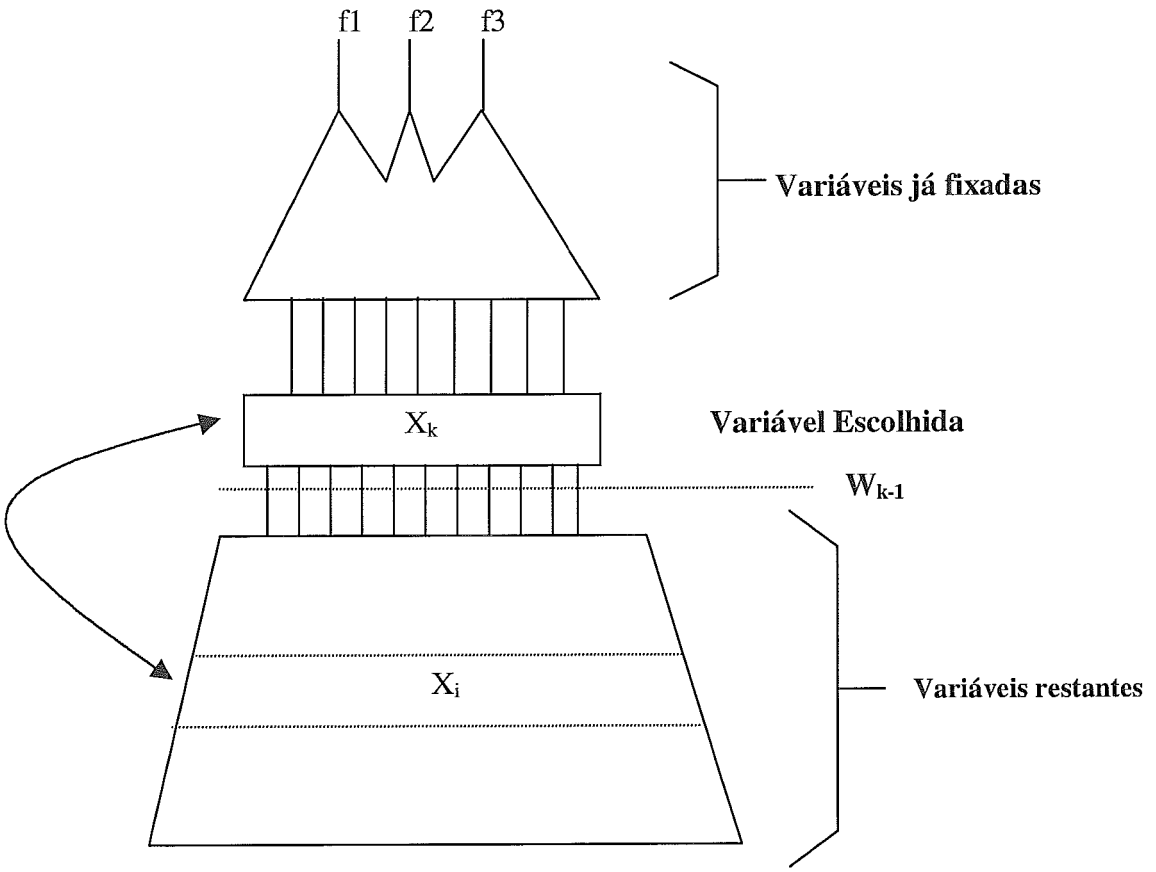


Figura 4.3: Permutação de Variáveis

É necessário então, que a função de custo utilizada forneça uma boa estimativa do tamanho mínimo do grafo para a escolha de x_k. Além disso, a função deve ser calculada dentro de um tempo razoável. Portanto, para satisfazer estes requisitos, Minato[17] definiu a largura de OBDD como a função de custo a ser utilizada neste algoritmo.

O método da largura mínima apresenta complexidade igual $O(n^2G)$, onde G é o tamanho médio do OBDD, já que seu tamanho varia durante o processo de reordenação. Comparado com outros métodos de ordenação de variáveis, este método apresenta bons resultados. Isto é, apresenta soluções bem próximas as encontradas pelas melhores ordenações, num período de tempo razoável. Entretanto, este método tem como desvantagem o fato de ser um método de reordenação de variáveis. Isto significa que depende da geração de um OBDD inicial, para que ele possa ser aplicado.

4.4 Método *Sifting*

O algoritmo de *sifting*, apresentado em [19], também é um método de reordenação de variáveis. Ele baseia-se na busca da melhor posição para uma variável, assumindo que as demais variáveis do grafo estão fixas em suas posições. Então, se existem n variáveis no grafo, existem n posições potenciais para uma variável, incluindo sua posição corrente. Entre essas n posições, o método tem como objetivo encontrar a posição, para a variável em questão, que reduza o tamanho do OBDD inicial.

É fato conhecido, que na prática, não se pode-se encontrar o melhor posicionamento para uma variável numa representação OBDD, assumindo que todas as demais variáveis estão fixas em suas posições, através de uma análise de baixa complexidade do OBDD [6]. Portanto, a melhor posição para uma variável é determinada por uma enumeração de força-bruta, como descrito a seguir.

Inicialmente, as variáveis são ordenadas em ordem decrescente, baseado no número de nós em cada nível do grafo. Então, cada variável é submetida ao processo de busca da melhor posição assumindo que todas as demais variáveis estão fixas. Este processo é realizado apenas uma vez para cada variável do grafo.

4.4.1 O Algoritmo

Após uma ordenação inicial das variáveis, uma variável é trocada de posição com sua sucessora, até que esta atinja o último nível do grafo. Então, o algoritmo realiza trocas de posição entre a variável e sua predecessora até que esta atinja o topo do grafo. Verifica-se em qual das posições, ocupada pela variável no grafo, obteve-se o melhor resultado com relação a redução do OBDD. Em seguida, as trocas necessárias são realizadas até que a variável em questão, ocupe a sua melhor posição no grafo.

A Figura 4.4 mostra as permutações de variáveis, quando o algoritmo é aplicado para a variável x_4. No exemplo as sete posições possíveis para x_4 são exploradas realizando-se nove trocas adjacentes. A melhor posição é restabelecida com mais seis trocas adicionais (pior caso).

Note que o tamanho do grafo pode crescer significativamente após poucas trocas de variáveis e então eventualmente ser reduzido, abaixo de seu tamanho inicial. Assim, é aceita a melhor posição observada, independente dos aumentos intermediários ocorridos na sequência.

O algoritmo *sifting* necessita de $O(n^2)$ trocas de níveis adjacentes no grafo. Cada

$x_1, x_2, x_3, \mathbf{x_4}, x_5, x_6, x_7$	inicial
$x_1, x_2, x_3, x_5, \mathbf{x_4}, x_6, x_7$	troca (x_4, x_5)
$x_1, x_2, x_3, x_5, x_6, \mathbf{x_4}, x_7$	troca (x_4, x_6)
$x_1, x_2, x_3, x_5, x_6, x_7, \mathbf{x_4}$	troca (x_4, x_7)
$x_1, x_2, x_3, x_5, x_6, \mathbf{x_4}, x_7$	troca (x_7, x_4)
$x_1, x_2, x_3, x_5, \mathbf{x_4}, x_6, x_7$	troca (x_6, x_4)
$x_1, x_2, x_3, \mathbf{x_4}, x_5, x_6, x_7$	troca (x_5, x_4)
$x_1, x_2, \mathbf{x_4}, x_3, x_5, x_6, x_7$	troca (x_3, x_4)
$x_1, \mathbf{x_4}, x_2, x_3, x_5, x_6, x_7$	troca (x_2, x_4)
$\mathbf{x_4}, x_1, x_2, x_3, x_5, x_6, x_7$	troca (x_1, x_4)
$x_1, \mathbf{x_4}, x_2, x_3, x_5, x_6, x_7$	troca (x_4, x_1)
$x_1, x_2, \mathbf{x_4}, x_3, x_5, x_6, x_7$	troca (x_4, x_2)
$x_1, x_2, x_3, \mathbf{x_4}, x_5, x_6, x_7$	troca (x_4, x_3)
$x_1, x_2, x_3, x_5, \mathbf{x_4}, x_6, x_7$	troca (x_4, x_5)
$x_1, x_2, x_3, x_5, x_6, \mathbf{x_4}, x_7$	troca (x_4, x_6)
$x_1, x_2, x_3, x_5, x_6, x_7, \mathbf{x_4}$	troca (x_4, x_7)

Figura 4.4: Permutação de Variáveis no Método *Sifting*

troca de variável tem complexidade proporcional a largura do grafo. Para controlar a complexidade de pior caso, podemos supor por exemplo que a busca numa direção particular pode ser terminada se o grafo crescer para o dobro de seu tamanho original.

4.5 Método *Interleaving*

Os métodos de *simulated annealing*, largura mínima e *sifting* são métodos de reordenação de variáveis, o que implica na necessidade de se ter uma representação inicial em OBDD de uma função Booleana. Entretanto, o método *interleaving* [11] difere dos demais métodos apresentados, não só por se basear em informações topológicas de circuitos, mas sobretudo por ser um método desenvolvido especificamente para circuitos de múltiplas saídas. Este fato possibilita a geração de OBDDs reduzidos de funções Booleanas, além de ser um método de ordenação de variáveis.

A maioria dos algoritmos de ordenação de variáveis, baseado em topologia de circuitos, seguem o princípio da busca em largura e em profundidade através de um circuito, desde suas saídas até suas entradas primárias[20, 15]. Basicamente, uma ordenação das variáveis de entrada é determinada na mesma ordem em que elas são visitadas a partir de suas saídas. Esta estratégia é baseada na observação que as entradas, cujas conexões são topologicamente próximas no circuito, devem

permanecer próximas na ordenação. Esses algoritmos de ordenação foram desenvolvidos, inicialmente, para circuitos de uma única saída, apresentando na maioria dos casos resultados satisfatórios [11]. No caso de circuitos com múltiplas saídas, o mesmo tipo de algoritmo é aplicado a cada saída em ordem de prioridade. Esta ordem de prioridade das saídas é determinada de modo que a saída considerada logicamente mais complexa tenha a maior prioridade. Os parâmetros como profundidade das portas e o número de portas relacionadas com a saída influenciam a medida de complexidade lógica [20, 15].

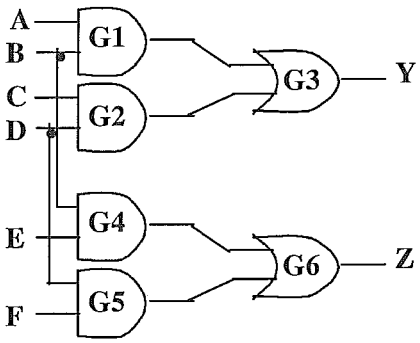
Como mencionado, algoritmos convencionais de ordenação de variáveis baseados em topologia de circuitos foram desenvolvidos para circuitos com apenas uma saída. E depois, foram adaptados para circuitos de múltiplas saídas. Entretanto, as ordenações produzidas por estes métodos não são necessariamente boas para todas as funções (saídas) do circuito. Para algumas funções que representam determinadas saídas de circuitos, não é nem mesmo possível gerar uma representação em OBDD, usando-se as ordenações encontradas pelos métodos convencionais. Muito embora possam existir ordenações que possibilitem a geração de um OBDD. Tal fato motivou o desenvolvimento do algoritmo de Interleaving que está descrito a seguir.

4.5.1 O Algoritmo

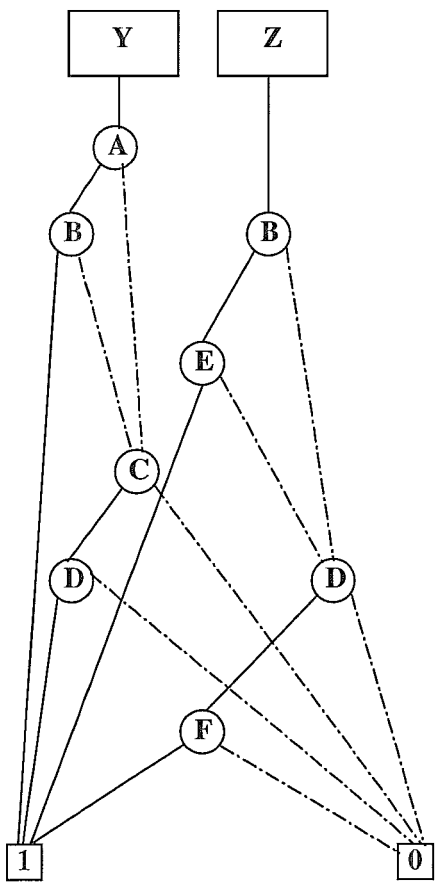
O princípio do método de *interleaving* é combinar a ordenação obtida para a saída de mais alta prioridade, com as demais ordenações, obtidas para as saídas de menor prioridade. Além disso, é necessário manter o máximo possível, a ordenação para todas as saídas. Este fato não ocorre nos algoritmos convencionais, onde a única ordenação que é garantida é aquela obtida para a função de maior prioridade. Caso hajam variáveis que não se relacionem com a saída de maior prioridade, estas são anexadas em ordem de prioridade das saídas, ao conjunto ordenado de variáveis. Ao serem anexadas estas variáveis devem obedecer a mesma ordem em que aparecem quando as funções as quais se relacionam são submetidas ao processo de ordenação. No algoritmo *interleaving* as variáveis que ainda não figuram no conjunto ordenado de variáveis, por não se relacionarem com as funções já submetidas ao processo de ordenação, são intercaladas no conjunto na tentativa de se manter a ordenação para todas as funções do circuito.

O algoritmo *interleaving* é ilustrado usando-se o exemplo do circuito mostrado na Figura 4.5(a). Assumindo que a saída Y tem prioridade mais alta que a saída Z.

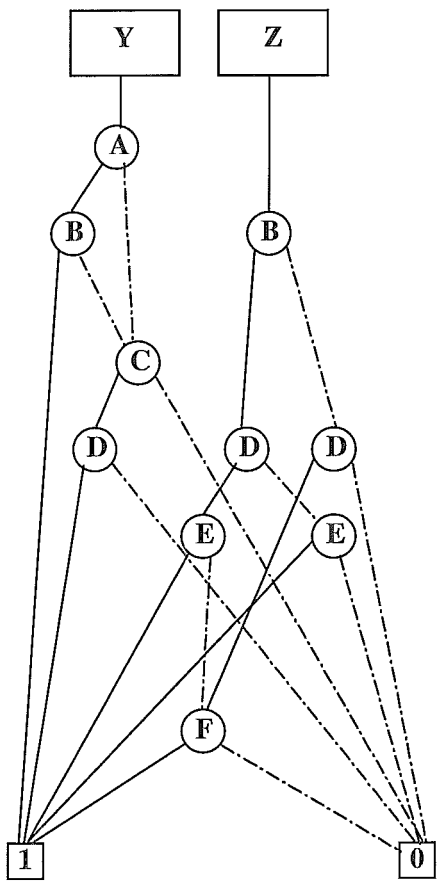
Primeiramente, foi aplicado um algoritmo de ordenação convencional cuja ordenação obtida para a saída Y foi $\{A, B, C, D\}$. Da mesma forma obtêm-se a ordenação $\{B, E, D, F\}$ para a saída Z. Cada ordenação representa uma boa ordenação para suas respectivas funções de saída, de acordo com o algoritmo de ordenação convencional utilizado. Assim, se usarmos um algoritmo convencional, a ordenação gerada para o circuito completo é $\{A, B, C, D, E, F\}$ como mostrado na Figura 4.6(b). Isto é feito anexando-se $\{E, F\}$, que não se relacionam com Y, mas tem relação com Z, ao final da ordenação $\{A, B, C, D\}$ obtida para Y. A ordenação $\{A, B, C, D, E, F\}$ obtida por este processo é boa para Y. Porém, não é uma boa ordenação para Z. Se agora aplicarmos o algoritmo de *interleaving*, encontrar-se a seguinte ordenação $\{A, B, E, C, D, F\}$. Esta ordenação é boa tanto para Y quanto para Z. E é obtida intercalando-se $\{A, B, C, D\}$ com $\{E, F\}$, como mostra a Figura 4.6(a). Na Figura 4.5(c), observa-se que o OBDD gerado para a ordenação $\{A, B, C, D, E, F\}$, através do método convencional, é maior que o OBDD (Figura 4.5(b)), gerado pela ordenação $\{A, B, E, C, D, F\}$ obtida pelo método de *Interleaving*.



(a) Ciruito com múltiplas saídas

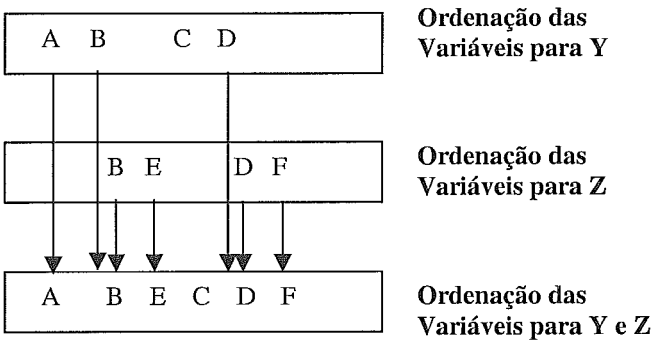


(b) OBDD - usando ordenação obtida pelo Algoritmo Interleaving

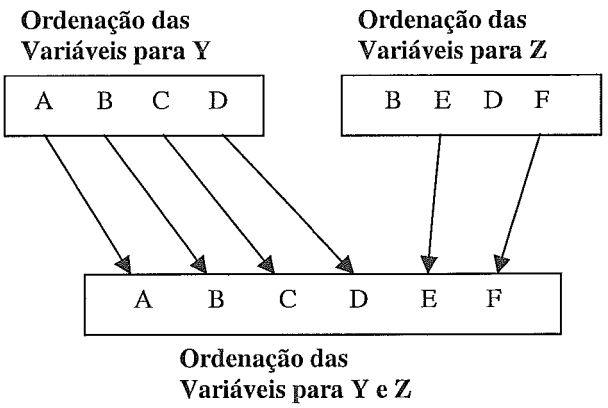


(c) OBDD - usando ordenação obtida pelo Algoritmo Convencional

Figura 4.5: Representação do Circuito em OBDD



(a) Ordenação obtida pelo método Interleaving



(b) Ordenação obtida pelo método Convencional

Figura 4.6: Diferentes Ordenações

Capítulo 5

Avaliação do Tamanho de OBDDs

Neste capítulo são apresentadas as idéias de Berman[4] e mais recentemente de McMillian[16], para avaliação do tamanho de um OBDD.

5.1 Largura de OBDD

Em [4], Berman provou um limite superior para o tamanho de OBDDs, necessário para representar circuitos de largura limitada. Um circuito tem largura limitada se seus elementos puderem ser arranjados numa ordem linear, tal que qualquer corte numa seção do circuito cruze um número máximo de fios w, chamada de largura do circuito. Existe uma ordenação de variáveis do circuito, tal que o tamanho do OBDD é limitado por $n2^w$, onde n é o número de entradas primárias do circuito. Esse resultado aplica-se apenas se a ordem for topológica, isto é, todos os fios seguem uma única direção. Em [16] este resultado foi generalizado, para mostrar que se w_f limita o número de fios em qualquer corte na direção direta, e w_r limita o número de cortes na direção reversa, então o tamanho do OBDD será limitado por $n2^{w_f 2^{w_r}}$. No caso onde $w_r = 0$, esta fórmula se reduz ao resultado de Berman.

Nas próximas seções definimos os conceitos de seção direta, seção reversa e seção de um OBDD.

5.1.1 Seção Direta e Seção Reversa de um Circuito

Seja $L = (G, <)$ uma ordenação linear das portas lógicas de um circuito. As entradas e saídas primárias do circuito são classificadas como uma instância especial das portas lógicas, de modo a simplificar as definições. Dada uma ordenação L, a seção direta de um circuito na porta g corresponde ao número de fios conectados a uma saída de uma porta g_1 e a uma entrada de uma porta g_2, tal que $g_1 \leq g$ e $g < g_2$,

como na Figura 5.1. A seção reversa é o número de fios conectados a uma saída de uma porta g_1 e a uma entrada de uma porta g_2, tal que $g_2 \leq g$ e $g < g_1$, como na Figura 5.2. Nos circuitos em questão, nenhum fio está conectado às saídas de duas portas distintas, assim esses dois conjuntos são disjuntos. A ordenação L é dita topológica quando todas as seções reversas são vazias[16].

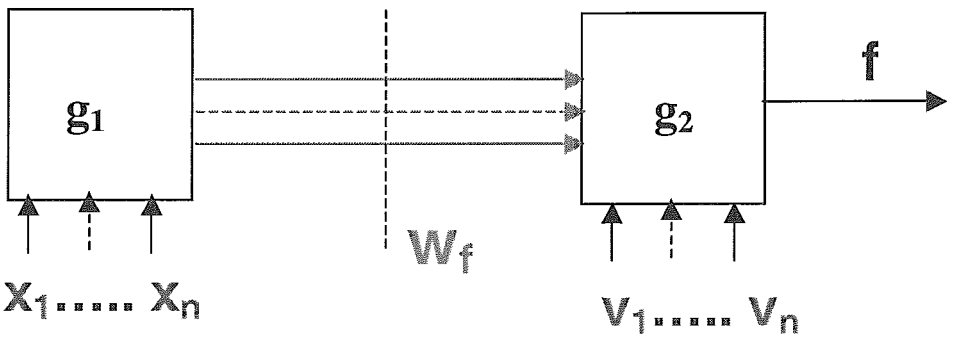


Figura 5.1: Seção Direta de um Circuito

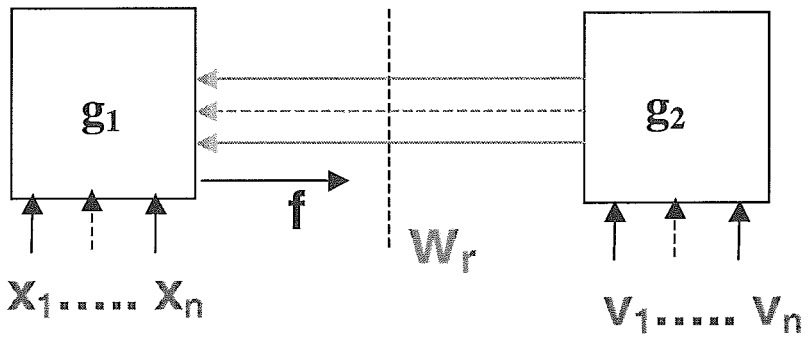


Figura 5.2: Seção Reversa de um Circuito

A seção direta do circuito sob a ordenação L, denotada como w_f, é o tamanho máximo da seção direta em qualquer porta g. Similarmente a seção reversa de um circuito sob a ordenação L, denotada de w_r, é o tamanho máximo da seção reversa em qualquer porta do circuito.

5.1.2 Seção de um OBDD

Uma seção de um OBDD no nível i é o conjunto de nós rotulados com a variável v_i. Note que as variáveis de um OBDD são aqui numeradas partindo do nó raiz para os nós terminais, já que isto torna a explicação mais simples.

A largura de seção w_p de um OBDD p é o tamanho máximo de qualquer seção de p. Note que a largura de seção, aqui definida é menor ou igual a largura de OBDD definifa na Seção 4.3 por Minato[17]. O tamanho de um OBDD é a soma de suas

seções. Portanto, o tamanho de um OBDD pode ser limitado por nw_p, onde n é o número de variáveis.

É simples demonstrar que o tamanho da seção de um OBDD no nível i é o número de funções distintas,

$$f_{px}(v_i, \ldots, v_n) = f_p(x_1, \ldots, x_{i-1}, v_i, \ldots, v_n)$$

que dependem de v_i, onde $x = (x_1, \ldots, x_{i-1})$ é um vetor Booleano e f_p é a função representada por p. Em [16] foi enunciado o seguinte teorema, limitando o tamanho de um OBDD em termos das larguras das seções diretas e reversas do circuito que ele representa.

Teorema 2 *Se um circuito que implementa uma função f tem seção direta w_f e seção reversa w_r para alguma ordenação linear L, então o OBDD p que representa a função f tem o tamanho limitado por $n2^{w_f 2^{w_r}}$, onde n é o número de entradas do circuito.*

Prova: Associe as variáveis $v_1, \ldots, v_n$ de um OBDD com as entradas do circuito, tal que para todo $i \leq j$, $v_i \leq v_j$. O tamanho da i-éssima seção do OBDD resultante pode ser limitado como descrito a seguir.

Seja $x = (x_1, \ldots, x_{i-1})$ um vetor Booleano. Divida o circuito escolhendo qualquer porta lógica g, tal que $v_{i-1} \leq g < v_i$. Seja Y a seção direta em g e Z a seção reversa, como ilustrado na Figura 5.3.

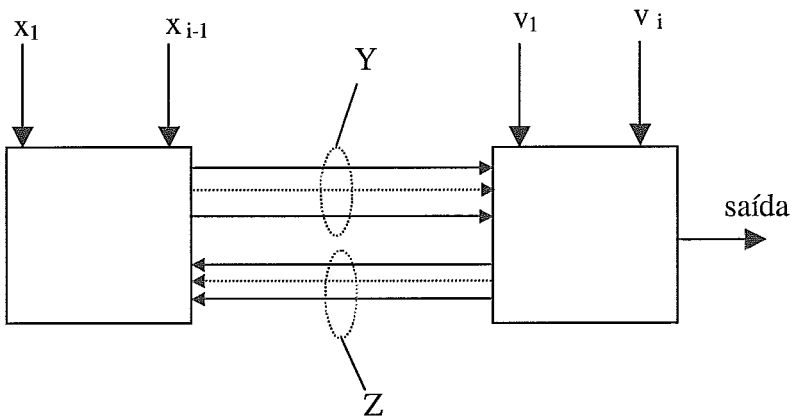


Figura 5.3: Circuito com $|Y|$ Seções Diretas e $|Z|$ Seções Reversas

Para qualquer valor de x, Y é uma função de Z, essa função determina $f_x(v_i, \ldots, v_n)$. O número de funções Booleanas com $|Z|$ entradas e $|Y|$ saídas é $2^{|Y|2^{|Z|}}$. ■

Para ilustrar uma maneira de se obter o número de funções Booleanas com $|Z|$ entradas e $|Y|$ saídas, considere um bloco com n entradas x e uma saída Y, como na Figura 5.4.

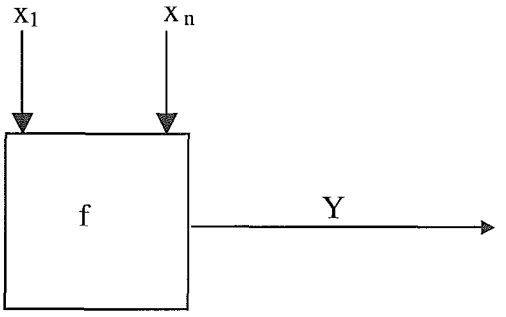


Figura 5.4: Circuito de n Entradas e uma Saída

Podemos dizer que $f(x_1, \ldots, x_n) = Y$, onde Y pode assumir dois valores distintos $\{0, 1\}$, para qualquer vetor $(x_1, \ldots, x_n)$. Se tivermos o mesmo bloco com $|Y|$ saídas, podemos dizer que temos para qualquer vetor dado $(x_1, \ldots, x_n)$, $2^{|Y|}$ possibilidades de valores diferentes para $(Y_1, \ldots, Y_n)$.

Assim podemos dizer que o número de funções possíveis para um dado vetor $(x_1, \ldots, x_n)$ é:

$$|f_x| = 2^{|Y|}$$

Considere agora o esquema apresentado na Figura 5.5.

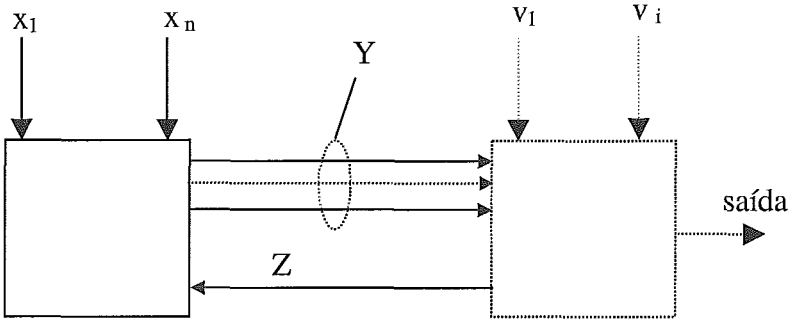


Figura 5.5: Circuito com apenas uma Seção Reversa

Para um dado vetor $(x_1, \ldots, x_n)$ podemos ter dois conjuntos distintos de funções $f_x(Z)$, ou seja $f_x(0)$ e $f_x(1)$, ocorre que independente do valor de Z cada f_x pode assumir até $2^{|Y|}$ valores distintos. Logo, temos $2^{2|Y|}$ valores distintos de funções possíveis.

Se considerarmos agora o esquema mostrado na Figura 5.3. Podemos dizer que para um dado vetor $(x_1, \ldots, x_n)$ temos $2^{|Y|}$ funções f_x distintas possíveis de ocorrer.

Assim temos então um total de $2^{|Y|2^{|Z|}}$ funções possíveis. Isto limita o número total de funções distintas f_x, que por sua vez limita a largura da seção do OBDD no nível i. Sabemos que $|Y| \leq w_f$ e $|Z| \leq w_r$. Portanto, o tamanho total do OBDD é limitado por $n2^{w_f 2^{w_r}}$.

As idéias de Berman e McMillan estabelecem um limite superior para o tamanho de OBDDs. Partindo desta observação desenvolvemos um método capaz de estimar o tamanho de um OBDD, para uma ordenação de suas variáveis, sem que seja necessário a sua construção. Este método está descrito no Capítulo 6.

Capítulo 6

Estimando o Tamanho de OBDDs

Os métodos mais conhecidos de ordenação de variáveis não oferecem nenhum tipo de garantia quanto a qualidade das ordenações produzidas. Para vários métodos, em especial para os métodos de reordenação de variáveis de OBDDs, é necessária a construção de um OBDD inicial para que tais métodos possam ser aplicados. No entanto, nem sempre é possível estabelecer uma ordenação das variáveis que possibilite a construção inicial de um OBDD.

Neste capítulo descrevemos um método capaz de estimar o tamanho de OBDDs para uma dada ordenação de suas variáveis, sem que seja necessária a sua construção.

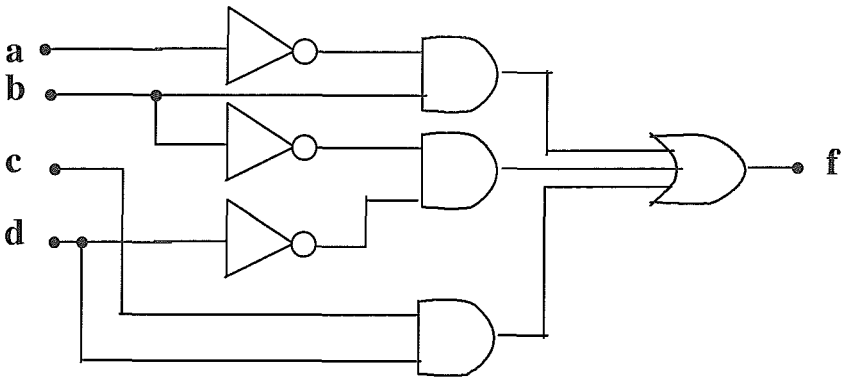
Esta estimativa pode ser utilizada na busca de uma ordenação inicial que permita a construção de um OBDD, para que então um dos métodos de reordenação existentes possa ser aplicado, ou ainda possa conduzir a um novo método de ordenação de variáveis.

Apresentamos neste capítulo, também, os resultados de nossos experimentos.

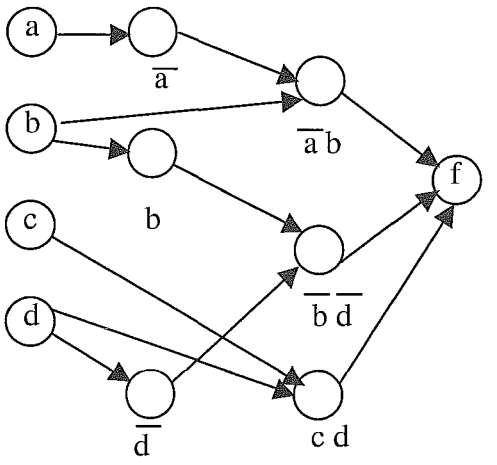
6.1 A Idéia

Obtemos uma estimativa do tamanho de um OBDD através da avaliação das seções direta e reversa de uma possível implementação do circuito que corresponde a função que o OBDD representa.

A partir de um circuito qualquer podemos construir um grafo direcionado que o representa. Cada entrada e saída do circuito é representada por um nó, bem como cada uma de suas portas lógicas. Na verdade as saídas podem ser representadas pela última porta que a define. As arestas do grafo são direcionadas na mesma direção que a corrente elétrica atravessa o circuito. Um exemplo de transformação de circuito em grafo está ilustrado na Figura 6.1.



Circuito Lógico



Grafo de Representação do Circuito

Figura 6.1: Um Circuito Lógico e o seu Grafo Correspondente

Dada uma ordenação para as variáveis de entrada $X = x_1, \ldots, x_n$ do circuito, utilizamos cortes mínimos neste grafo para calcular, de forma automática, as seções diretas e reversas (descritas na Seção 5.1) de uma possível implementação do circuito. Para tanto utilizamos o algoritmo de corte *Ford-Fulkerson* com pesos unitários. Após avaliarmos as seções para todos os níveis do OBDD, podemos calcular um limite superior para cada nível. Somamos os limites para cada seção obtendo assim, uma estimativa para o tamanho do OBDD com a ordenação de variáveis utilizada.

A seguir descrevemos como estimamos um limite superior para um OBDD a partir de cortes no grafo de representação do circuito.

6.1.1 Cálculo das Seções Diretas e Reversas Através de Cortes

Imagine um circuito qualquer com uma única saída, no qual as entradas estão divididas em dois grupos. Imagine ainda que o primeiro grupo de entradas deve ser parcialmente processado em um bloco de circuito, e que alguns fios passam deste primeiro bloco para um segundo bloco, no qual se juntam as variáveis do segundo grupo de variáveis para permitir que a função como um todo seja avaliada.

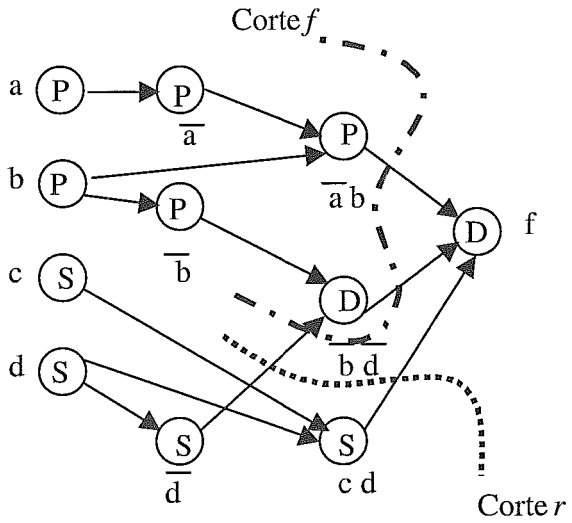


Figura 6.2: Três Tipos de Nós em um Grafo

A Figura 6.2 ilustra um grafo no qual "separamos" as variáveis a e b das demais. Temos então três tipos de nós no grafo: aqueles cujo valor pode ser calculado no primeiro bloco (P); aqueles cujo valor depende das entradas do primeiro grupo, mas cujo valor também depende de uma ou mais variáveis do segundo grupo (D); e aqueles cujo valor depende exclusivamente das variáveis no segundo grupo (S).

Em geral, um corte que separe as variáveis de entrada do primeiro grupo dos nós do tipo D corresponde ao número de fios que devem entrar no segundo bloco do circuito para uma possível implementação do circuito. No caso da Figura 6.2 este corte f tem valor igual a 2, que corresponde ao número de fios necessários para as sub-funções $(\overline{a}b, \overline{b})$, na direção direta daquela seção do circuito, como mostrado na Figura 6.3.

Nós que se encontrem no corte podem ser do tipo P ou D. Nós do tipo P podem ter seu valor avaliado no primeiro bloco do circuito enquanto que nós do tipo D podem ser parcialmente avaliados. Ou seja, se um nó corresponde por exemplo a um E lógico, com apenas algumas de suas entradas já calculadas, podemos calcular

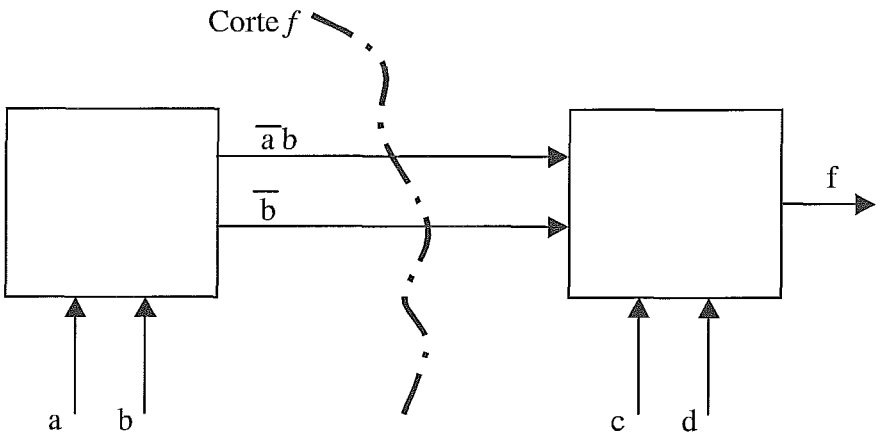


Figura 6.3: Seção Direta do Circuito

o E destas entradas no primeiro bloco. Depois enviamos um fio com este valor já calculado para uma porta E adicional que terá também como entradas aquelas entradas da porta original cujo valor não pode ser calculado no primeiro bloco do circuito. Esta nova porta E só será avaliada no segundo bloco do circuito.

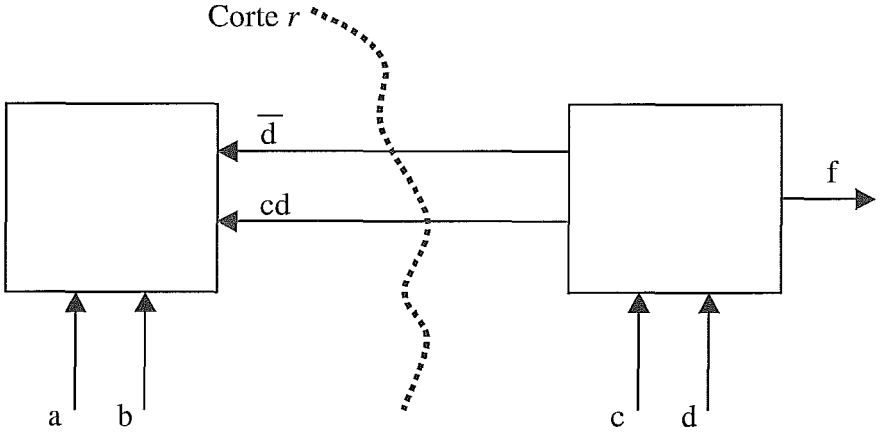


Figura 6.4: Seção Reversa do Circuito

De modo análogo, encontramos o corte que separa as entradas do segundo grupo dos nós do tipo D. Neste caso, o valor do corte r representa o número de fios na direção reversa, na mesma seção do circuito, como é mostrado na Figura 6.4. No exemplo da Figura 6.2 este corte também tem valor igual a 2.

Na verdade, o primeiro nível sempre contribui com um único nó na estimativa. O segundo nível é estimado pela separação da primeira variável e das demais; o terceiro nível é estimado pela separação entre as duas primeiras variáveis e as demais, e assim por diante. Para cada divisão em dois grupos das variáveis de entrada, calculamos um tamanho de corte direto f e um tamanho de corte reverso r, e utilizamos a

melhor estimativa, i.e., o menor valor dentre 2^f e 2^{2^r}. Na obtenção de nossa métrica consideramos todas as $n-1$ possíveis divisões de variáveis em dois grupos que respeitam a ordenação que está sendo avaliada. Em cada passo, procuramos os cortes mínimos direto e reverso, que nos fornecem um limite superior para o número de variáveis no nível correspondente do OBDD a ser construído. A soma destas estimativas fornece um limite superior para o tamanho do OBDD.

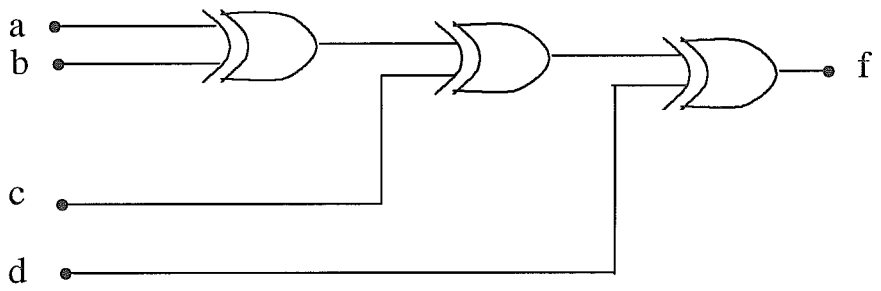


Figura 6.5: Uma Implementação para Função Paridade.

	Seção Direta	Seção Reversa	Estimativa
$a\|bcd$	1	3	$2^1 = 2$
$ab\|cd$	1	2	$2^1 = 2$
$abc\|d$	1	1	$2^1 = 2$
Total			7

Tabela 6.1: Estimativa para o Tamanho do OBDD da Função Paridade para a Ordenação $a < b < c < d$

	Seção Direta	Seção Reversa	Estimativa
$b\|cda$	1	3	$2^1 = 2$
$bc\|da$	2	2	$2^2 = 4$
$bcd\|a$	3	1	$2^{2^1} = 4$
Total			11

Tabela 6.2: Estimativa para o Tamanho do OBDD da Função Paridade para a Ordenação $b < c < d < a$

Na Figura 6.5 é mostrado um circuito que implementa a função paridade. Na Tabela 6.1 estão representadas os valores das seções diretas e reversas obtidas para a ordenação $a < b < c < d$ para o circuito desta figura. São também apresentadas as

estimativas obtidas para cada nível do OBDD e a estimativa total para o tamanho do OBDD (está acrescida de uma unidade, pois inclui também o primeiro nível). Na Tabela 6.2 são mostrados os mesmos parâmetros que são exibidos na Tabela 6.1, para a ordenação $b < c < d < a$. Neste caso, a seção reversa forneceu uma melhor estimativa para o número de nós no último nível.

Note que as estimativas, apresentadas pelo nosso método, diferem, embora esta função seja uma função simétrica, e o tamanho de sua representação em OBDD não varia com as variações de sua ordenação.

No nosso método as informações topológicas influenciam as estimativas para o tamanho de um OBDD. A Figura 6.6 mostra uma outra implementação para a função paridade onde, para a topologia apresentada, as variáveis a e b parecem precisar estar próximas para que se obtenha uma boa ordenação. O mesmo acontece com o par de variáveis c e d. Entretanto o circuito da figura implementa a mesma função paridade que o circuito da Figura 6.5, e seu OBDD não apresenta variação de tamanho para diferentes ordenações de suas variáveis.

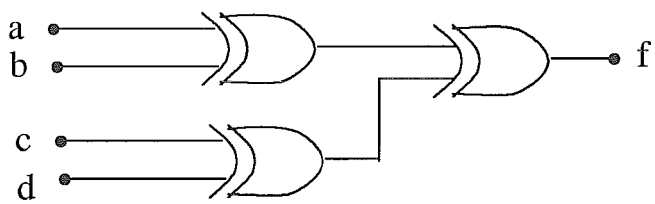


Figura 6.6: Uma Outra Implementação para Função Paridade.

Estimativa para circuitos com Múltiplas Saídas

O método descrito para estimar o tamanho de um OBDD para uma dada ordenação baseia-se em limites superiores no número de nós em cada nível de um OBDD. Para tratar de circuitos com múltiplas saídas, aplicamos o mesmo método descrito na seção anterior a cada uma das saídas do circuito. Como a soma dos tamanhos de OBDDs individuais é maior ou igual que o tamanho do OBDD que representa simultaneamente todas as funções que compõem o circuito, obtemos um limite superior para o tamanho do OBDD desejado.

Para os circuitos de múltiplas saídas, também construímos um grafo de representação, onde cada nó do grafo está associado a uma porta lógica do circuito. Cada saída representa uma função diferente do circuito. Estas funções são avaliadas separadamente uma a uma. Definida a função(saída) a ser avaliada, descartam-se

todos os nós do grafo que não se relacionam com a função que está sendo avaliada. Na Figura 6.7 estes nós que não se relacionam com a saída $*$ são representadas em pontilhado. No grafo resultante aplica-se o mesmo procedimento descrito na Subseção 6.1.1. Obtendo-se então uma estimativa para o tamanho da função analisada. Repete-se o mesmo procedimento para as demais funções(saídas) do circuito. Após a avaliação de todas as saídas do circuito, temos uma estimativa para o tamanho do OBDD de cada saída. Soma-se as estimativas e obtem-se a estimativa para o tamanho do OBDD que representa o circuito para a ordenação utilizada.

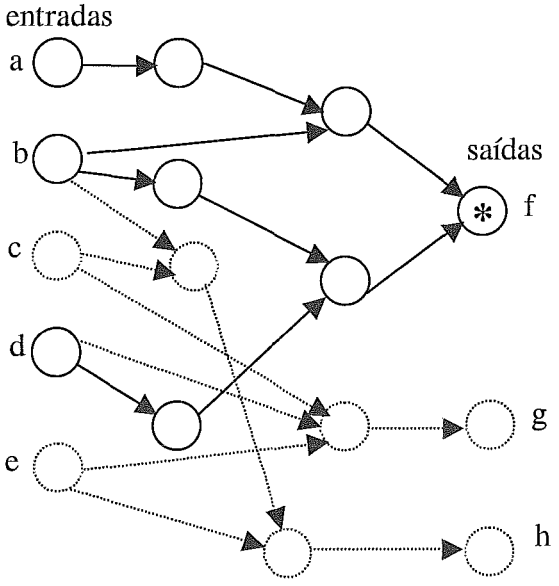


Figura 6.7: Grafo de um Circuito com Múltiplas Saídas

6.1.2 Descrição dos Experimentos

Implementamos um algoritmo que calcula a estimativa para o tamanho de um OBDD como descrito na seção 6.1. O algoritmo tem como parâmetros de entrada um circuito em formato *BLIF* (*Berkeley Logic Interchange Format*), e uma ordenação inicial para as variáveis de entrada do circuito. Para estudar qualitativamente a estimativa implementamos também uma heurística de *sifting*, semelhante a descrita no Capítulo 4. Esta heurística utiliza nossas estimativas para várias ordenações e considera aquela que apresenta a melhor estimativa.

Os circuitos que utilizamos nos testes são circuitos combinacionais multiníveis que fazem parte do *Benchmark do International Workshop on Logic Synthesis and Optimization - 1991*, que inclui circuitos do ISCAS 85

e ISCAS 89. Os arquivos *BLIF* destes circuitos estão disponíveis em *http://www.ece.pdx.edu/polo/function/LGSynth91/cmlexamples/*.

A Tabela 6.3 apresenta para cada circuito utilizado como parâmetro de entrada seu nome, sua função lógica e algumas de suas características, como número de variáveis de entrada e saída, e a quantidade aproximada de portas lógicas presentes no circuito.

Os resultados foram obtidos utilizando uma máquina *Sun Ultra 1* com 64M*bytes* de memória e um *Pentium III* com 128M*bytes* de memória. Empregamos o utilitário *nanotrav* do pacote CUDD-2.3.0 da Universidade do Colorado, por Fabio Somenzi, para medir o tamanho das representações em OBDD dos exemplos. O utilitário *nanotrav* representa arestas de complemento. Portanto, o tamanho reportado pode ser um pouco menor que o tamanho do OBDD que resultaria sem estas arestas de complemento.

Análise dos Resultados

A Tabela 6.4 apresenta para cada circuito utilizado como parâmetro de entrada seu nome, uma estimativa de tamanho do OBDD para a ordenação inicial de suas variáveis, um fator A que indica a variação na estimativa de tamanho dos OBDDs, o tamanho do OBDD para a ordenação original das variáveis, o tamanho do OBDD para ordenação das variáveis sugerida pela aplicação do nosso método e um fator B que indica a variação dos tamanhos dos OBDDs. O fator A é a razão entre a estimativa de tamanho do OBDD, para a ordenação de suas variáveis, sugerida por nosso método e a estimativa de tamanho do OBDD para a ordenação original de suas variáveis. O fator B é a razão entre o tamanho do OBDD para a ordenação original de suas variáveis e o tamanho do OBDD para a ordenação, de suas variáveis, sugerida por nosso método.

O gráfico da Figura 6.8 ilustra através de uma comparação entre os fatores A e B, da Tabela 6.4, o comportamento do nosso método para cada circuito. Esses circuitos são representados por um par de barras no gráfico. O fator A é representado por uma barra preta e o fator B representado por uma barra branca. Os pares de barras estão dispostos da esquerda para direita na mesma ordem em que os circuitos aparecem na Tabela 6.4.

Os resultados apresentados na Tabelas 6.4 mostram que o método descrito é sensível ao tamanho de um OBDD para ordenações destes. Na maioria dos exemplos,

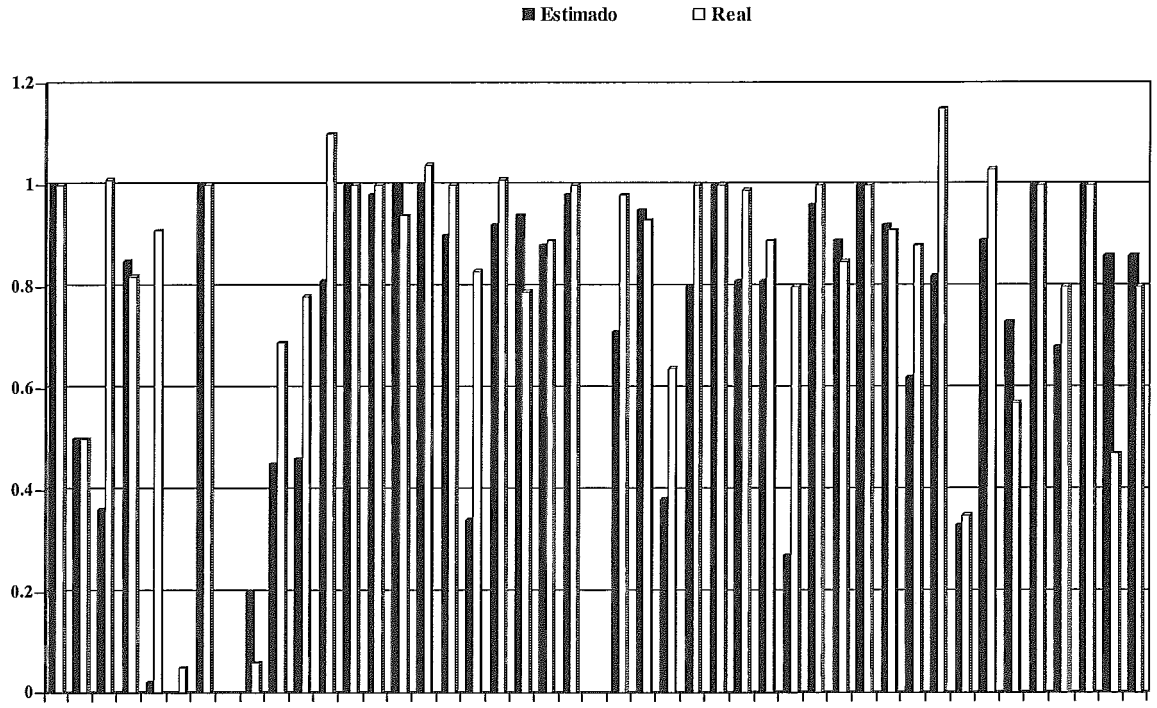


Figura 6.8: Gráfico Comparativo entre a Variação da Estimativa A e a Variação de Tamanho B

as ordenações obtidas com base na estimativa de tamanho do OBDD, resulta numa redução de tamanho dos OBDDs, como pode ser observado através dos fatores A e B da Tabela 6.4 ou no gráfico da figura 6.8.

Em alguns casos, como por exemplo para os circuitos C880, CM150, CM151 e cmp, a redução do tamanho do OBDD, obtida com a ordenação das variáveis sugerida pelo método, é bastante significativa. Isto se deve ao fato de tais circuitos apresentarem uma ordenação inicial bastante ruim, o que possibilita uma reordenação de suas variáveis com base na topologia do circuito.

Em outros casos, como nos exemplos 9symml, CM138 ou b1, não verificamos alterações no tamanho dos OBDDs produzidos, seja utilizando a ordenação que nosso método sugere ou utilizando a heurística de *sifting*. Isto se deve a natureza das funções implementadas pelos circuitos ou ao tamanho reduzido do circuito.

Para outros exemplos, tais como pm1 e CM42, pode-se observar pelo fator B, na Tabela 6.4 ou pelas duas maiores barras brancas do gráfico mostrado na Figura 6.8,

que a ordem das variáveis sugerida pelo nosso método aumenta o tamanho do OBDD, embora o fator A indique que deveria ocorrer o contrário.

É importante notar que o espaço de busca utilizado para determinar a melhor estimativa foi limitado, porque a heurística *sifting* não explora todas as possíveis combinações de ordenação das variáveis. Isto indica que os resultados encontrados foram bastante favoráveis.

Na maioria dos exemplos, a ordem sugerida gera um OBDD menor que a ordem original. Assim, realizamos testes para identificar se as ordenações que geramos eram apropriadas para reordenação. Para tal, utilizamos o método de reordenação *sifting* do pacote CUDD 2.3.0, e obtivemos os resultados apresentados na Tabela 6.5. Para cada circuito utilizado como parâmetro de entrada, esta tabela inclui: seu nome; o tamanho do OBDD para uma ordenação original de suas variáveis; o tamanho do OBDD após a reordenação, por *sifting* da ordem original das variáveis; o tamanho do OBDD após a reordenação, por *sifting*, da ordem de variáveis sugerida por nosso método; e um fator de variação C. O fator C é a razão entre o tamanho do OBDD após a reordenação da ordem de variáveis sugerida por nosso método e o tamanho do OBDD após a reordenação da ordem original das variáveis.

Os resultados apresentados na Tabela 6.5 demonstram que na maioria dos exemplos, a partir de uma ordenação inicial sugerida por nosso método, obtivemos através da heurística *sifting* ordenações que geram OBDDs de tamanhos iguais ou menores aos obtidos a partir da reordenação de suas ordens originais. Isto sugere que as ordenações produzidas por nosso método são um bom ponto de partida para as heurísticas de reordenação de variáveis.

Nome do Circuito	Função do Circuito	Entrada	Saída	Número Aproximado de Portas
9symml	Contador de Uns	9	1	43
ADDERFDS	Somador	33	17	223
C1355	Corretor de Erros	41	32	546
C17	Lógica	5	2	6
C1908	Corretor de Erros	33	25	880
C880	ULA e Controle	60	26	383
CM138	Lógica	6	8	17
CM150	Lógica	21	1	69
CM151	Lógica	12	2	33
CM162	Lógica	14	5	43
CM163	Lógica	16	5	42
CM42	Lógica	4	10	17
CM82	Lógica	5	3	27
CM85	Lógica	11	3	38
alu2	ULA	10	6	335
alu4	ULA	14	8	681
b1	Lógica	3	4	13
b9	Lógica	41	21	125
c8	Lógica	28	18	164
cc	Lógica	21	20	47
cht	Lógica	47	36	229
cmb	Lógica	16	4	41
comp	Lógica	32	3	151
cordic	Lógica	23	2	102
count	Contador	35	16	143
cu	Lógica	14	11	48
decod	Decodificador	5	16	22
f51ml	Aritmética	8	8	43
frg1	Lógica	28	3	105
i1	Lógica	25	16	46
lal	Lógica	26	19	114
misex1	-	8	7	32
misex2	-	25	18	29
mux	Mux	21	1	91
pcle	Lógica	19	9	68
pcler8	Lógica	27	17	84
pm1	Lógica	16	13	39
sct	Lógica	19	15	91
tcon	Lógica	17	16	41
term1	Lógica	34	10	358
unreg	Lógica	36	16	97
vda	Lógica	17	39	585
vg2	-	25	8	110
x2	Lógica	10	7	42

Tabela 6.3: Exemplos dos Circuitos Utilizados como Parâmetros de Entrada e suas Características

Nome do Circuito	Estimativa Inicial	Variação da Estimativa (A)	Tamanho Inicial	Tamanho Obtido	Variação de Tamanho (B)
9symml	148	1.00	25	25	1.00
ADDERFDS	1253270504	0.50	327677	163856	0.50
C1355	225045632	0.36	45922	46170	1.01
C17	26	0.85	11	9	0.82
C1908	1881971444	0.02	36007	32717	0.91
C880	226009313	0.00	346660	17182	0.05
CM138	128	1.00	18	18	1.00
CM150	131350	0.00	131071	91	0.00
CM151	1068	0.20	511	31	0.06
CM162	572	0.45	67	46	0.69
CM163	244	0.46	55	43	0.78
CM42	104	0.81	20	22	1.10
CM82	40	1.00	16	16	1.00
CM85	122	0.98	38	38	1.00
alu2	658	1.00	231	217	0.94
alu4	5890	1.00	1182	1230	1.04
b1	20	0.90	7	7	1.00
b9	3238	0.34	178	147	0.83
c8	1566	0.92	136	138	1.01
cc	400	0.94	101	80	0.79
cht	588	0.88	150	134	0.89
cmb	1750	0.98	36	36	1.00
comp	3868860	0.00	458698	488	0.00
cordic	2324	0.71	45	44	0.98
count	1310	0.95	234	218	0.93
cu	1484	0.38	59	38	0.64
decod	320	0.80	32	32	1.00
f51m	218	1.00	39	39	1.00
frg1	1376562	0.81	204	201	0.99
i1	242	0.81	56	50	0.89
lal	3700	0.27	165	132	0.80
misex1c	280	0.96	41	41	1.00
misex2c	814	0.89	136	116	0.85
mux	532	1.00	383	383	1.00
pcle	408	0.92	87	79	0.91
pcler8	1094	0.62	139	122	0.88
pm1	382	0.82	46	53	1.15
sct	2984	0.33	161	56	0.35
tcon	108	0.89	33	34	1.03
term1	144330	0.73	580	331	0.57
traffic	20	1.00	9	9	1.00
unreg	590	0.68	147	118	0.80
vda	163208	1.00	4345	4345	1.00
vg2	77968	0.86	1044	493	0.47
x2	264	0.86	69	55	0.80

Tabela 6.4: Resultados Obtidos a partir da Ordenação Sugerida pelo Método Proposto

Nome do Circuito	Tamanho Inicial	Tamanho c/ Sifting	Sugerida c/ Sifting	Variação de Tamanho (C)
9symml	25	25	25	1.00
ADDERFDS	327677	82	82	1.00
C1355	45922	30775	30897	1.00
C17	11	9	8	0.89
C1908	36007	7153	7909	1.11
C880	346660	7064	5420	0.77
CM138	18	18	18	1.00
CM150	131071	33	33	1.00
CM151	511	17	17	1.00
CM162	67	31	30	0.97
CM163	55	27	36	1.33
CM42	20	20	21	1.05
CM82	16	16	16	1.00
CM85	38	36	36	1.00
alu2	231	162	160	0.99
alu4	1182	603	514	0.85
b1	7	7	7	1.00
b9	178	110	113	1.03
c8	136	83	83	1.00
cc	101	60	52	0.87
cht	150	90	105	1.17
cmb	36	29	30	1.03
comp	458698	141	146	1.04
cordic	45	43	43	1.00
count	234	81	82	1.01
cu	59	32	33	1.03
decod	32	32	32	1.00
f51m	39	39	39	1.00
frg1	204	93	93	1.00
i1	56	37	49	1.32
lal	165	86	67	0.78
misex1c	41	35	35	1.00
misex2c	136	86	82	0.95
mux	383	16	16	1.00
pcle	87	42	42	1.00
pcler8	139	86	94	1.09
pm1	46	41	41	1.00
sct	161	65	48	0.74
tcon	33	25	25	1.00
term1	580	163	142	0.87
traffic	9	8	8	1.00
unreg	147	82	92	1.12
vda	4345	507	507	1.00
vg2	1044	229	154	0.67
x2	69	37	37	1.00

Tabela 6.5: Resultados Obtidos pelo Método de Reordenação *Sifting* a partir de uma Ordenação Sugerida pelo Método Proposto

Capítulo 7

Conclusões

Propomos neste trabalho um método que fornece uma estimativa do tamanho de um OBDD para uma ordenação de suas variáveis. Nosso método gera uma estimativa baseada num limite superior para o número de nós dos níveis do OBDD. Os parâmetros para o cálculo desse limite são obtidos através de cortes num grafo que possui informações sobre a topologia do circuito que se quer representar. Para analisar qualitativamente as estimativas produzidas por nosso método, implementamos uma heurística de reordenação de variáveis - *sifting*. Esta heurística utiliza nossas estimativas para várias ordenações de variáveis e considera aquela que apresenta a melhor estimativa.

Nos experimentos realizados utilizamos 44 circuitos combinacionais multi-níveis, pertencentes ao *Benchmark do International Workshop on Logic Synthesis and Optimization - 1991*. Os resultados obtidos pela aplicação do método podem ser considerados satisfatórios. Na maioria desses experimentos, as informações sobre a topologia, contida nos exemplos, foram suficientes para que o método traduzisse as variações de tamanho que seus OBDDs estão sujeitos devido a mudanças na ordem das variáveis.

Como esperado, as estimativas obtidas por nosso método se mostraram insensíveis às mudanças na ordenação das variáveis para os circuitos que implementam funções de natureza simétrica.

Nossos resultados demonstraram que para a maioria dos exemplos, a partir de uma ordenação inicial sugerida por nosso método, obtivemos através da heurística *sifting* ordenações que geravam OBDDs de tamanhos iguais ou menores aos obtidos a partir da reordenação das ordens originais. O que sugere que as ordenações que produzimos são um bom ponto de partida para uma heurística de reordenação de

variáveis.

Trabalhos Futuros

A heurística que utilizamos para avaliar o nosso método realiza uma busca local. Seria interessante a obtenção de um método de ordenação que abranja todo o espaço de permutações. Existem atualmente algoritmos aproximativos que minimizam medidas muito semelhantes aquela que utilizamos como limite superior para o tamanho do OBDD.

Acreditamos que o algoritmo que resolve o problema de corte mínimo para um arranjo linear *(Min. Cut Linear Arrangement)*, ou alguma variante deste problema, se aplicado ao grafo que representa um circuito, produzirá ordenações apropriadas.

Referências Bibliográficas

[1] B. Akers. Binary decision diagrams. *IEEE Trans. On Comuters*, C-27(6), June 1978.

[2] B. Akers. Functional testing uising binary decision diagrams. *8th International Symposium on Fault Tolerant Computing*, pp.82-92, June 1979.

[3] M. Lobbig B. Bolling and I. Wegener. Simulated annealing to improve variable orderings for Ordered Binary Decision Diagrams. *IWLS, 5b:5.1-5.10*, 1995.

[4] C.L. Berman. Ordered binary decision diagrams and circuit structure. *International Conference on Computer Design - Cambrige - NY*, (392395), October 1989.

[5] B. Bolling. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. Comp.*, 1996.

[6] R. E. Bryant. Graph-based algorithm for boolean function manipulation. *IEEE Trans. On Cuters*, C-35(677-691), August 1986.

[7] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. June 1992.

[8] Kapur R. Butler K. M., Ross D. E. and Mercer M. R. Heuristics to computer variable ordering for efficient manipulation of ordered binary decision diagrams. *ACM/IEEE Proc. 28th DAC*, (417-420), June 1991.

[9] S. Friedman. Finding the optimal variable ordering for Binary Decision Diagrams. *DAC*, 1987.

[10] M.R. Garey and Johnson. *Computer and Intractibility: A guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[11] Goichi Ootomo Hiroshige Fujii and Chikahiro Hori. Interleaving based variable ordering methods for Ordered Binary Decision Diagrams. *ICCAD*, 1993.

[12] R. Jacobi. *A study of the Application of Binary Decision Diagrams on Multi-Level Logic Synthesis.* PhD thesis, Universit'e Catholique de Louvain, Belgium., Dec. 1993.

[13] K. Karplus. Representing boolean function with if-then-else dags. *Internal Report UCSC-CRL-88-28*, November 1988.

[14] C. Y. Lee. Representing of switching circuits by binary decision diagrams. *BSTJ*, pp.985-999(38), July 1959.

[15] H. Fujisawa M. Fujita and Y. Matsunaga. Variable ordering algotithms for Ordered Binary Decision Diagrams and their evaluation. *IEEE Trans. Computer-Aided Desing*, Jan. 1993.

[16] Kenneth L. McMillan. *Symbolic Model Checking - An approac to the state explosion problem.* PhD thesis, Carnegie Mellon University, United States, May 1992.

[17] S. Minato. Zero-supressed bdds for set manipulation in combinatorial problems. In *Proceedings of 30th Design Automation Conference*, November 1993.

[18] M.Karaugh. The map method for synthesis of combinacional logic circuits. *Transactions of AIEE, Part I, Communications and Electronics*, pp.593-99(72), 1953.

[19] R. Rudell. Dynamic variable ordering for Ordered Binary Decision Diagrams. *ACM/IEEE ICCAD, 42-43*, 1993.

[20] R. K. Brayton S. Malik, A. R. Wang and A. Sangiovanni-Vincentelli. Logic Verification using Binary Decision Diagrams in a logic synthesis environments. *ICCAD*, 1988.

[21] A. Slobodov'a and C. Meinel. Sample method for minimization of OBDDs. *IWLS, pp 311-316*, 1998.