UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE TELEMÁTICA

Contribuição à Síntese de Circuitos Digitais Utilizando Programação Linear Inteira 0 e 1

Alexandre César Rodrigues da Silva

Banca examinadora: **Dr. Ivanil Sebastião Bonatti**
Dr. Magno Meirelles Ribeiro
Dr. Mário Lúcio Cortes
Dr. Carlos Alberto dos Reis Filho
Dr. Furio Damiani

Suplentes: **Dr. Edson Moschim**
Dr. Walter da Cunha Borelli

Dissertação apresentada à Faculdade de Engenharia Elétrica da UNICAMP, como requisito parcial para obtenção do título de Doutor em Engenharia Elétrica.

22 de setembro de 1993

Curriculum Vitae da Banca Examinadora

- **Prof. Dr. Ivanil Sebastião Bonatti** - Presidente

  Doutor em Automática - 1981 - Toulouse - França

  Título da Tese: Gestion de Réseaux de Service: Application au Réseau Telephonique Interurbain

  Local de trabalho: DT - FEE - UNICAMP

- **Prof. Dr. Magno Meirelles Ribeiro** - Membro

  Doutor em Automática - 1981 - Toulouse - França

  Título da Tese: Regulação de um Forno de Indústria de Cerâmica em Torno de um Regime Ótimo de Produção.

  Local de trabalho: DEE - UFMG

- **Prof. Dr. Mário Lúcio Cortes** - Membro

  Ph.D em Engenharia Elétrica - 1987 - U.S.A.

  Título da Tese: Temporary Failures in Digital Circuits: Experimental Results and Fault Modeling.

  Local de trabalho: DCC - IMECC - UNICAMP

- **Prof. Dr. Carlos Alberto dos Reis Filho** - Membro

  Doutor em Engenharia Elétrica - 1982 - UNICAMP

  Título da Tese: Correção de Curvatura em Fontes de Referência Tipo 'BANDGAP'

  Local de trabalho: DEMIC - FEE - UNICAMP

- **Prof. Dr. Furio Damiani** - Membro

  Doutor em Engenharia Elétrica - 1983 - UNICAMP

  Título da Tese: Contribuição ao Estudo de Estruturas MOS Implantadas com Cloro.

  Local de trabalho: DESIF - FEE - UNICAMP

- **Prof. Dr. Edson Moschim** - Suplente

  Doutor em Ciências Físicas - 1989 - Paris - França

  Título da Tese: Contribution a la Modélization de Composants Photoniques et systèmes de Communication Utilizant Fibres Optiques em Plastique comme Support de Transmission.

  Local de trabalho: DT - FEE - UNICAMP

- **Prof. Dr. Walter da Cunha Borelli** - Suplente

  Doutor em Eletrônica e Comunicações - 1983 - Kent - Inglaterra

  Título da Tese: Convolutional Codes for Multi-Level Data Transmission.

  Local de trabalho: DT - FEE - UNICAMP

Este trabalho é dedicado a meus pais
Carlos Rodrigues e Lourdes Sicca
pelo respeito e confiança com que nos educaram.

Agradecimentos

Sem a colaboração de várias pessoas, a realização desse trabalho teria sido extremamente penosa. Por isso manifesto minha gratidão:

- ao Prof. Dr. Ivanil Sebastião Bonatti, pela orientação, confiança e esforço constante dispensado no decorrer desses longos 7 anos de convívio intenso;

- ao Prof. Dr. Mário Lúcio Cortes, Prof. Dr. Magno Meirelles Ribeiro, Prof. Dr. Carlos Alberto dos Reis Filho, Prof. Dr. Furio Damiani e Prof. Dr. Edson Moschim pela gentileza da aceitação de comporem a banca examinadora dessa tese;

- um agradecimento especial ao Prof. Dr. Walter da Cunha Borelli pelas suas valiosas sugestões e presteza nas incontáveis ajudas solicitadas;

- aos colegas Paulo A. Giordano, Renato J. O. Figueiredo e Mauro M. Castanho pela participação em diversas fases da realização desse trabalho;

- ao suporte técnico sempre efetivo dos amigos Gorgônio de Araújo e Miguel Angelo Rozsas;

- aos Professores Dr. Paulo A. Valente Ferreira e Dr. Takaaki Ohishi pelas discussões sobre programação matemática.

- ao Prof. Dr. Akebo Yamakami pela orientação e incentivo quando aqui cheguei como aluno especial;

- à Faculdade de Engenharia Elétrica, em especial ao Departamento de Telemática que sempre procurou proporcionar a seus membros o que existe de mais sofisticado e atual em questão de hardware e software, indispensável ao desenvolvimento de qualquer pesquisa;

- à CAPES e ao CNPq pelo apoio financeiro, que possibilitou a realização deste trabalho.

Resumo

Este trabalho trata do problema de simplificação de funções booleanas e da redução de estados, em máquinas de estados finitos, modelando-os como um problema de programação matemática.

Na minimização lógica, os implicantes são gerados aplicando-se o algoritmo do consenso numa árvore binária que representa a função booleana. A cobertura mínima é obtida resolvendo-se um problema de programação linear inteira 0 e 1, cuja função objetivo é a soma ponderada de todos os implicantes primos e as restrições correspondem a soma dos implicantes primos que cobrem cada mintermo da função.

Na minimização de funções booleanas com múltiplas saídas o problema de cobertura mínima pode ser modelado como um problema matemático não linear dependendo do critério de otimização utilizado.

O método de geração de classes de compatibilidades máximas foi utilizado para a redução de estados. A função objetivo é formulada como a soma das classes primas sujeita às restrições de cobertura e fechamento.

Uma vez formulado como um problema de programação matemática, a minimização de funções booleanas e a redução de máquinas de estados se abrem para as novas técnicas desenvolvidas nessa área de pesquisa.

Abstract

This work proposes a method of dealing with the problem of boolean function minimization and finite state machine reduction by modeling each of them as a mathematical programming problem.

In the logic minimization, prime implicants are generated by applying the consensus algorithm in the binary decision tree that represents a boolean function. A minimal cover can then be obtained by solving an integer linear program with objective function as a weighing sum of prime implicants whose constraints are the sums of prime implicants covering each minterm.

In a multiple-output boolean function minimization, a minimal cover problem may be modelled as a non-linear mathematical problem depending on the specific optimization criterion that is used.

The method of generating the maximal compatibility classes has been used for the state reduction phase. The objective function is formulated as the sum of the prime classes, and the constraints are due to restrictions of covering and closure.

Once formulated as a mathematical programming problem, the boolean function minimization and the state machine reduction are opened to the new techniques that have been developed in this research area.

Conteúdo

Capítulo 1

Introdução

A minimização de funções booleanas tem recebido tratamento algorítmico a mais de 30 anos [13].

Para se obter a minimização exata de uma dada função booleana, a priori, deve-se, a partir de seus mintermos, gerar todos os implicantes primos e a partir desses, realizar uma cobertura mínima da função. Estas ações resultam num número muito grande de operações, o que tem ensejado uma busca contínua de métodos eficientes para tratar tal problema.

Como alternativa à complexidade da formulação exata, considerável esforço tem sido empregado no desenvolvimento de métodos quase ótimos utilizando-se de heurísticas.

Este capítulo apresenta os conceitos e definições necessários para o entendimento dos capítulos posteriores. As funções booleanas são definidas com a finalidade de descrever as propriedades dos circuitos combinacionais.

São apresentados dois algoritmos de minimização de funções booleanas, o clássico método de Quine-McCluskey [20] para permitir ao leitor a compreensão dos mecanismos básicos envolvidos na minimização de funções booleanas e o algoritmo Geração [25], por nós desenvolvido, utilizando-se de idéias heurísticas para ilustrar esta classe de métodos.

1.1 Introdução

O desenvolvimento de sistemas digitais, contendo poucas partes funcionais, como por exemplo circuitos integrados SSI (Small Scale Integration) e MSI (Medium Scale Integration), utilizava-se de técnicas de projeto onde os sucessivos passos eram pouco dependentes um dos outros. A decisão em relação a um dos passos não afetava significativamente todo o projeto. Além disso, os testes para verificar o comportamento funcional do sistema, podiam ser facilmente executados.

Em decorrência disso, erros de projetos, devido à má interpretação da descrição funcional, ou erros em alguns dos passos de projeto, podiam ser inspecionados localmente. O reprojeto ou reparos afetavam uma pequena parte do sistema, sem aumentar muito o custo e o tempo de desenvolvimento.

Alguns princípios e métodos foram criados e uma certa estabilidade caracterizou a atividade de projetos digitais nessa época. Devido ao pequeno porte dos circuitos projetados, não se sentia a necessidade de uso de métodos computacionais para a síntese de circuitos.

Com o aumento da complexidade dos circuitos integrados na tecnologia LSI (Large Scale Integration), em especial VLSI (Very Large Scale Integration), os passos envolvidos no projeto de sistemas digitais tornaram-se alvo de muitos estudos. Além disso, os VLSI não toleram erros de projeto e reparos, pois esses deslizes, além de onerosos, podem tornar o produto não competitivo no mercado [1]-[12].

No início dos anos 80, algumas atividades de projeto, como por exemplo, layout, teste e documentação, já estavam sendo apoiadas por poderosas ferramentas de projeto, porém, a síntese ainda era executada manualmente.

Somente em meados dos anos 80 e início dos anos 90, surgiram as primeiras ferramentas CAD para auxiliar a síntese, dentre estas, destacam-se as ferramentas AutoLogic da Mentor Graphics [9] e as ferramentas incorporadas nos pacotes de programação dos dispositivos lógicos programáveis, como por exemplo, da Altera [10] e da Xilinx [11].

Este trabalho de tese trata do desenvolvimento de algoritmos de síntese, em particular, das funções lógicas e das máquinas de estados.

O problema da simplificação de funções booleanas e da redução de estados, em máquinas seqüenciais síncronas, são vistos sob a óptica da programação matemática e propõe-se métodos eficientes para solucioná-los.

Nas seções deste capítulo são apresentados conceitos e definições necessários para o entendimento dos capítulos posteriores. As funções booleanas são definidas com a finalidade de descrever as propriedades dos circuitos combinacionais.

São apresentados dois algoritmos de minimização de funções booleanas, o clássico método de Quine-McCluskey para permitir ao leitor a compreensão dos mecanismos básicos envolvidos na minimização de funções booleanas e um algoritmo, proposto por nós, utilizando-se de idéias heurísticas para ilustrar esta classe de métodos.

No capítulo 2 é apresentada uma alternativa eficiente para a geração dos implicantes primos de uma função booleana. O método, intitulado GeraPlex, obtém todos os implicantes primos, através de sucessivas aplicações da operação de consenso aos ramos de uma árvore, onde cada caminho representa um mintermo ou irrelevante da função.

Da aplicação da operação de consenso aos ramos dos nós da árvore, define-se fusão, deslocamento e expansão que diminuem, mantêm e aumentam o número de caminhos da árvore,

respectivamente.

O capítulo 3 trata da minimização de funções booleanas, vista sob a óptica da programação matemática.

O problema de cobertura é formulado como um problema de programação linear inteira 0 e 1 e apresenta-se um método de resolução, baseado no algoritmo Simplex, modificado para considerar as particularidades do problema. No lugar do pivoteamento tradicional, foram utilizadas combinações lineares das linhas do tableau, para preservar os valores inteiros dos seus coeficientes. Encontrada uma solução otimista, utiliza-se o Plano de Corte de Gomory, para considerar as restrições de soluções inteiras.

A minimização de funções booleanas, com múltiplas saídas, é assunto do capítulo 4, onde apresenta-se o algoritmo MultPlex. As coberturas das funções são obtidas da solução de dois problemas de programação matemática, aqui chamados de problema de cobertura múltipla e problema de cobertura singular. O MultPlex representa uma contribuição expressiva na síntese de funções booleanas de múltiplas saídas.

No capítulo 5, apresenta-se o algoritmo ReduPlex, uma efetiva contribuição à solução da redução das máquinas de estados finitos, completa ou não completamente especificadas. Propõe-se um método numérico para a seleção do conjunto mínimo de classes primas, que define a máquina de estados finitos.

Para a redução de estados são apresentados, além do algoritmo proposto, os métodos de Partição de equivalência e da Tabela de Condições de Compatibilidade.

1.2 Definição da Álgebra Binária de Boole

No estudo dos circuitos lógicos binários, utiliza-se uma sub-álgebra da álgebra booleana, a Álgebra Booleana Binária, definida pelos seguintes axiomas:

$$A \neq 0 \Leftrightarrow A = 1$$
$$A \neq 1 \Leftrightarrow A = 0$$

A partir daqui, sempre que mencionarmos "Álgebra de Boole" estamos nos referindo a "Álgebra Binária de Boole".

A álgebra de Boole é um sistema algébrico que consiste do conjunto { 0, 1 }, duas operações binárias chamadas OR e AND e uma operação unária chamada NOT.

A operação OR, chamada soma lógica ou união, é representada por " + ". Então, A OR B é escrito como A + B que é chamado termo soma.

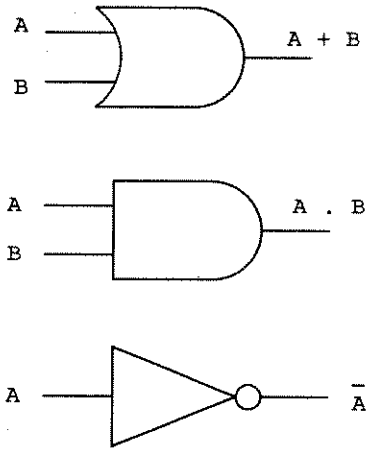
A operação AND, chamada produto lógico ou intersecção, é representada por " . ". Então, A AND B é escrito como A.B ou AB que é chamado termo produto.

A operação NOT, chamada complementação, é denotada por uma barra " ¯ " sobre a variável ou expressão a ser complementada. Então, o complemento de A é $\bar{A}$, e o complemento de (A+B).C = $\overline{(A+B).C}$.

As operações OR, AND e NOT são definidas conforme a tabela apresentada na figura 1.1a e podem ser representadas por circuitos elementares denominados gates ou portas, mostrados na figura 1.1b.

Entradas		Saída		
A	B	OR	AND	NOT A
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	0

a) Tabela verdade para os gates OR, AND e NOT



b) Símbolos que representam os gates OR, AND e NOT

Figura 1.1: Tabela verdade e símbolos que representam os gates OR, AND e NOT.

Um circuito composto de gates é denominado de circuito lógico. A figura 1.2 ilustra um exemplo de um circuito que realiza a expressão lógica A.B + C + D.E.

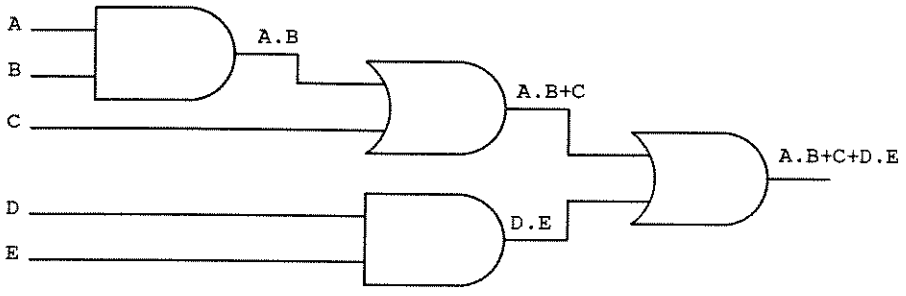


Figura 1.2: Exemplo de um circuito lógico.

1.2.1 Propriedades Básicas

Das definições de OR, AND e NOT as seguintes propriedades se verificam:

$$A + 1 = 1 \tag{1.1}$$

$$A.0 = 0 \tag{1.2}$$

$$A + 0 = A \tag{1.3}$$

$$A.1 = A \tag{1.4}$$

$$A + A = A \tag{1.5}$$

$$A.A = A \tag{1.6}$$

$$A + B = B + A \tag{1.7}$$

$$A.B = B.A \tag{1.8}$$

$$(A + B) + C = A + (B + C) \tag{1.9}$$

$$(A.B).C = A.(B.C) \tag{1.10}$$

$$A + \bar{A} = 1 \tag{1.11}$$

$$A.\bar{A} = 0 \tag{1.12}$$

$$A.(B + C) = A.B + A.C \tag{1.13}$$

$$A + (B.C) = (A + B).(A + C) \tag{1.14}$$

Todas essas propriedades podem ser provadas por indução perfeita (exaustivamente).

1.2.2 Expressões booleanas

Na lógica combinacional, a saída do circuito depende somente das entradas. Os problemas de lógica combinacional são normalmente descritos na forma de sentenças lógicas ou tabela verdade. Para projetar e implementar o circuito, obtém-se as expressões lógicas booleanas das funções da saída em termos das variáveis binárias que representam as entradas. As expressões lógicas são dadas ou na forma de soma de termos produto ou na forma de produto de termos soma. Essas expressões são então simplificadas e implementadas utilizando-se dispositivos lógicos.

Várias técnicas podem ser utilizadas para a simplificação das funções booleanas, como por exemplo, o Mapa de Karnaugh, Diagrama de Venn e o método tabular de Quine-McCluskey [20].

Define-se expressão booleana como a combinação de um número finito de variáveis booleanas e constantes 0 e 1 através de operações booleanas OR, AND e NOT.

As propriedades descritas a seguir, formam o conjunto de ferramentas básicas para a simplificação de expressões booleanas.

$$\overline{(A + B)} = \overline{A}.\overline{B} \tag{1.15}$$

$$\overline{(A.B)} = \overline{A} + \overline{B} \tag{1.16}$$

$$A + A.B = A \tag{1.17}$$

$$A.(A + B) = A \tag{1.18}$$

$$A + (\bar{A} + B) = A + B \tag{1.19}$$

$$A.(\bar{A} + B) = A.B \tag{1.20}$$

$$A.B + \bar{A}.C + B.C = A.B + \bar{A}.C \tag{1.21}$$

$$(A + B).(\bar{A} + C).(B + C) = (A + B).(\bar{A} + C) \tag{1.22}$$

As propriedades 1.1 a 1.22 permitem uma variedade imensa de manipulações nas expressões booleanas. Sempre que possível, elas permitem converter uma expressão em outra equivalente, com menos literais, onde literal é uma variável binária na sua forma original ou complementada.

1.3 Funções booleanas

Os valores assumidos por uma expressão para todas as combinações possíveis de suas variáveis definem uma função booleana.

Uma função booleana é qualquer correspondência do produto cartesiano B^n ao conjunto B^*.

$B^* = \{0, 1, X\}$

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

O conjunto $f^{-1}(1)$, formado por todas as n-uplas x $\in B^n$, tal que f(x) = 1, é chamado de conjunto UM e é formado por todos os mintermos da função.

De maneira similar, definem-se o conjunto ZERO, composto dos maxtermos e o conjunto DON'T, composto dos irrelevantes.

UM $= \{x \in B^n; f(x) = 1\}$

ZERO $= \{x \in B^n; f(x) = 0\}$

DON'T $= \{x \in B^n; f(x) = X\}$

Note que:

$$\text{UM} \cup \text{ZERO} \cup \text{DON'T} = B^n$$

$$\text{UM} \cap \text{ZERO} = \text{DON'T} \cap \text{ZERO} = \text{UM} \cap \text{DON'T} = \emptyset$$

Um implicante de uma função é um produto de literais, tal que, para toda n-upla, onde o implicante vale 1, a função vale 1 ou X.

Um implicante de uma função é primo, se nenhuma de suas literais pode ser eliminada sem que o produto restante deixe de ser implicante da função. Uma função booleana pode ser expressa como a soma lógica de seus implicantes primos.

Um implicante pode ser realizado por um circuito AND e por circuitos NOT (inversores).

Uma soma lógica pode ser realizada por um circuito OR.

Num espaço B^n, define-se sub-cubo a qualquer produto lógico, envolvendo até n literais distintas.

Assim, são sub-cubos no espaço B^4, os produtos:

$$C_{1111} = A.B.C.D$$

$$C_{111X} = A.B.C$$

$$C_{0000} = \overline{A}.\overline{B}.\overline{C}.\overline{D}$$

$$C_{000X} = \overline{A}.\overline{B}.\overline{C}$$

$$C_{01XX} = \overline{A}.B$$

$$C_{1XX1} = A.D$$

$$C_{0XX0} = \overline{A}.\overline{D}$$

$$C_{10XX} = A.\overline{D}$$

É evidente a associação entre as n-uplas no espaço $(B^*)^n$ e os sub-cubos, assim pode-se utilizar a notação compacta:

$$111\text{X} = \text{ABC}$$

$$01\text{XX} = \bar{\text{A}}\text{B}$$

Uma soma de produtos é uma cobertura de uma função, se para toda n-upla de B^n, onde a soma vale 1, a função vale 1 ou X e, para toda n-upla de B^n, onde a soma vale 0, a função vale 0 ou X.

Uma cobertura é irredundante, se nenhum de seus termos produto possam ser eliminados e, se em nenhum de seus termos, nenhuma das literais possam ser eliminadas, sem que esta deixe de ser uma cobertura da função.

Em um espaço $(B^*)^n$, existem 3^n sub-cubos.
Exemplo: $n = 2 \Longrightarrow 3^2 = 9$

00, 01, 0X, 10, 11, 1X, X0, X1 e XX

Em um espaço B^n, existem 2^n n-uplas.
Exemplo: $n = 2 \Longrightarrow 2^n = 4$

00, 01, 10, 11

Uma função booleana é definida através dos conjuntos disjuntos: UM, ZERO e DON'T, tais que:

$$\text{cardinal (UM)} + \text{cardinal (ZERO)} + \text{cardinal (DON'T)} = 2^n.$$

A toda n-upla de B^n estão associados 2^n sub-cubos de $(B^*)^n$, formados pela substituição sistemática de cada elemento da n-upla de B^n por X.
Exemplo: Sub-cubos associados a 010

010, 01X, 0X0, X10, 0XX, X1X, XX0 e XXX

1.3.1 Mapa de Karnaugh

O Mapa de Karnaugh é a forma mais conhecida para simplificar expressões booleanas. O mapa contém as mesmas informações da tabela verdade, representada de maneira ligeiramente diferente.

Esse mapa contém uma célula para cada combinação de entrada. Desse modo, uma função lógica com n variáveis possui 2^n células. As células são posicionadas tal que, somente uma variável de entrada muda de valor entre células adjacentes, permitindo combinar células de acordo com a regra $AB + A\bar{B} = A$.

O Mapa de Karnaugh para a função $f = \sum(0,2,3,6,7)$ é mostrado na figura 1.3.

A simplificação é feita graficamente eliminando a necessidade do uso da álgebra booleana para simplificar equações. A eficiência da minimização depende da habilidade do projetista e torna-se impraticável quando o número de variáveis é maior que seis. Nestes casos, utilizam-se métodos sistemáticos para a minimização de funções booleanas.

1.3.2 Formas Canônicas

Considere a tabela verdade apresentada na figura 1.4. Por possuir três variáveis, são possíveis $2^3 = 8$ combinações.

Da tabela verdade, pode-se obter a seguinte expressão lógica:

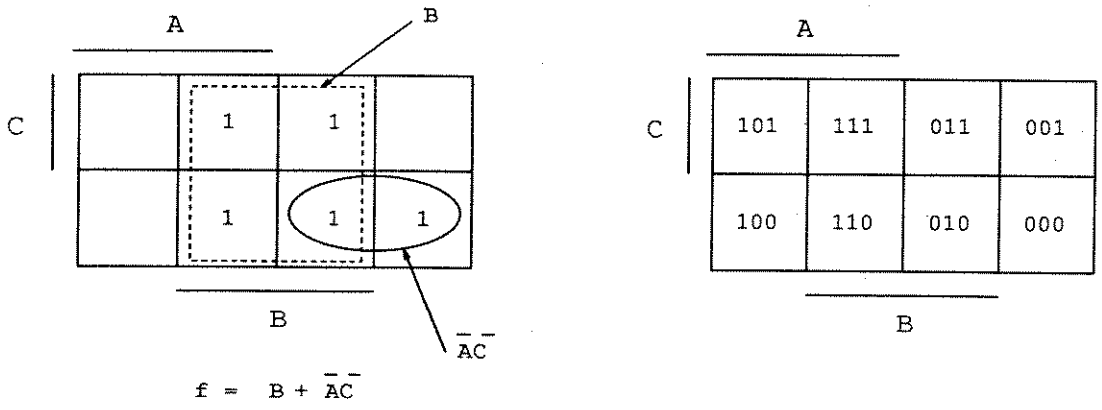


Figura 1.3: Simplificação da função f = $\sum$(0,2,3,6,7) utilizando-se o Mapa de Karnaugh.

Decimal	Entrada			Saída
	A	B	C	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Figura 1.4: Tabela verdade para uma função f.

$$f = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC$$

Essa expressão é chamada forma canônica soma de produtos.

Um termo produto que contém cada uma das n variáveis como fatores, complementada ou não, é chamado de mintermo.

Se um mintermo é representado por sua forma decimal, a expressão f é escrita da forma $f = \sum(0,2,3,6,7)$, pois f = 1 para as linha 0, 2, 3, 6 e 7 da tabela.

Uma expressão lógica também pode ser escrita como produto de soma. Isto é feito considerando-se as combinações em que f = 0.

Para a tabela verdade apresentada na figura 1.4, f = 0 para as linhas 1, 4 e 5, então:

$$\bar{f} = \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C$$

$$f = \overline{\bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C}$$

$$f = (A + B + \bar{C}).(\bar{A} + B + C).(\bar{A} + B + \bar{C})$$

O produto de somas pode ser escrito como $f = \prod(1,4,5)$.

A soma lógica que contém cada uma das n variáveis, complementadas ou não, é chamada de maxtermo.

A expressão formada de produto de todos os maxtermos para o qual a função vale 0 é chamada de forma canônica soma de produto.

1.4 Método de Quine-McCluskey

O método de Quine-McCluskey simplifica funções booleanas no sentido de baratear os circuitos digitais que as sintetizam.

A idéia fundamental do método é a aplicação repetida da propriedade:

$$AB + A\bar{B} = A\ (B + \bar{B}) = A$$

Exemplo 1. Para minimizar a função $f_1(A,B,C,D) = \sum(0,1,8,9)$, mostrada na figura 1.5, tem-se:

$$f_1 = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D$$

$$f_1 = \bar{A}\bar{B}\bar{C}(D + \bar{D}) + A\bar{B}\bar{C}(D + \bar{D}) = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}$$

$$f_1 = \bar{B}\bar{C}(A + \bar{A}) = \bar{B}\bar{C}$$

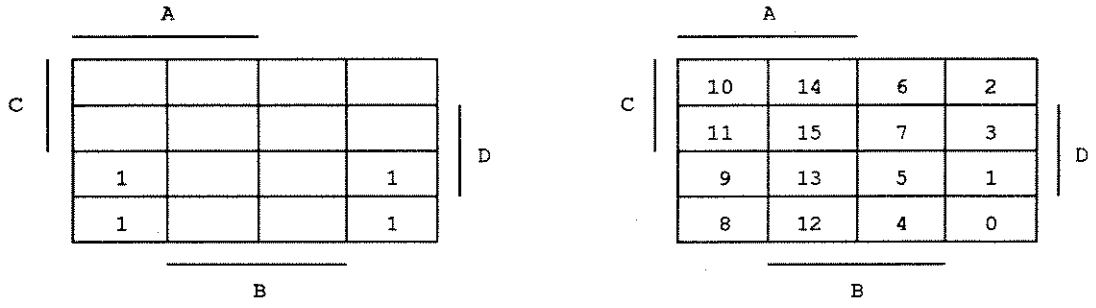


Figura 1.5: Mapa de Karnaugh para f_1(A,B,C,D) = $\sum$(0,1,8,9).

1.4.1 Sistematização do método

O método de Quine-McCluskey consiste de duas fases:

- Geração de implicantes primos;
- Cobertura dos mintermos.

Com os passos definidos a seguir, a primeira fase do método se torna sistemática e mais facilmente adaptada aos procedimentos computacionais.

1.4.1.1 Geração de Implicantes Primos

- 1. Separa-se os mintermos e irrelevante em "caixas", sendo que na mesma caixa ficam os termos com o mesmo número de dígitos '1' na sua forma binária. Este número de 1's é o índice da caixa. Esta operação define o GRUPO 0.
- 2. Para todo i no intervalo [0..N] (onde N é o número de variáveis da função), compara-se cada termo da caixa de índice i com todos os termos da caixa i+1. Os termos são combinados de acordo com o teorema $AB + A\bar{B} = A$, isto é, dois termos são combináveis se pertencerem a caixas adjacentes e diferirem por apenas um dígito na representação binária. Além disso, o termo da caixa de índice superior (i+1) deve ser sempre maior que o termo da caixa de índice inferior (i). Marca-se todos os termos combinados pelo menos uma vez. Esta operação gera um novo GRUPO.
- 3. Combina-se da mesma forma os termos gerados no passo anterior .
- 4. O processo acima (3) continua até que não haja mais combinação possível. Os termos não marcados constituem o conjunto de implicantes primos da função dada.

Exemplo 2. Aplicando-se a primeira fase do método à função $f_2 = \sum(0,1,2,5,7,8,9,10,13,15)$. apresentada na figura 1.6, obtém-se a tabela apresentada na figura 1.7, onde pode-se observar que:

- O grupo é dado pelo número de combinações ocorridas, ou seja, pelo número de posições irrelevantes ("X") na representação binária;
- As caixas são dadas pelo número de 1's na forma binária;

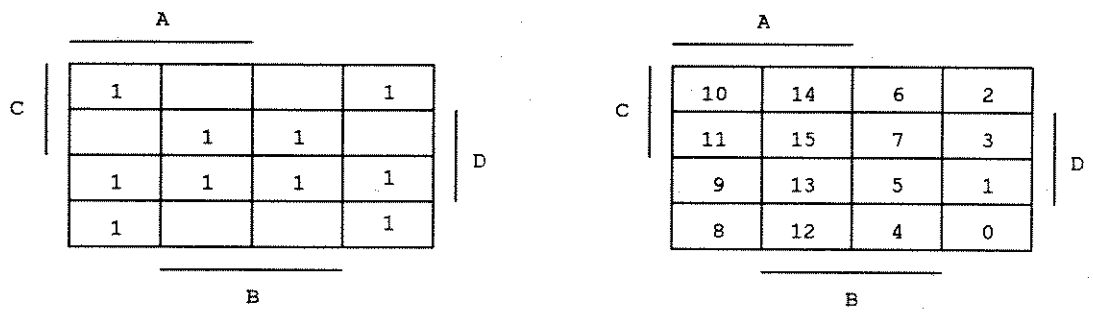


Figura 1.6: Mapa de Karnaugh de $f_2 = \sum(0,1,2,5,7,8,9,10,13,15)$.

- O grupo 0 é formado sempre pelos próprios mintermos e irrelevantes da função .
- Na geração dos demais grupos, pode-se produzir um mesmo implicante mais de uma vez, porém, ele constará apenas uma vez da tabela.

Caixa	Mintermo	Tick
0	0000	√
1	0001	√
1	0010	√
1	1000	√
2	0101	√
2	1001	√
2	1010	√
3	0111	√
3	1101	√
4	1111	√

Grupo 0

Caixa	Implicante	Tick
0	000X	√
0	00X0	√
0	X000	√
1	0X01	√
1	X001	√
1	X010	√
1	100X	√
1	10X0	√
2	01X1	√
2	X101	√
2	1X01	√
3	X111	√
3	11X1	√

Grupo 1

Caixa	Implicante	Tick
0	X00X	
0	X0X0	
1	XX01	
2	X1X1	

Grupo 2

Figura 1.7: Tabela para minimização de $f_2 = \sum(0,1,2,5,7,8,9,10,13,15)$.

Destaca-se neste exemplo, que apenas os termos do grupo 2 não foram combinados e, portanto, não foram marcados e formam o conjunto dos implicantes primos da função f_2 :

$$f_2(ABCD) = \bar{B}\bar{C} + \bar{B}\bar{D} + \bar{C}D + BD \tag{1.23}$$

O custo inicial da função, dado pela soma de todos os mintermos é 50; note a redução do custo para 12, apenas pela obtenção dos implicantes primos.

1.4.1.2 Cobertura dos mintermos

Após a obtenção do conjunto de todos os implicantes primos da função, monta-se uma tabela na qual as linhas são os implicantes primos e as colunas são os mintermos. Os estados irrelevantes não aparecem nesta tabela, uma vez que não precisam ser cobertos pela expressão mínima.

A tabela é montada colocando-se uma marca (X) nas intersecções entre os implicantes primos e os mintermos, indicando quais mintermos são cobertos por cada um dos implicantes primos.

O problema resume-se então, em selecionar um subconjunto mínimo de linhas da tabela, de modo que cada coluna tenha ao menos uma marca neste subconjunto, ou seja, todos os mintermos são cobertos por este subconjunto de implicantes primos. Tal subconjunto deve ser o que possui número mínimo de literais em relação a todos os demais que possam ser selecionados.

Para exemplificar, é montado a seguir a tabela dos implicantes primos de f_2 apresentada na figura 1.7, com base no seu conjunto de implicantes primos obtidos na equação 3.32. Esta tabela é mostrada na figura 1.8. Note na tabela que os mintermos 2, 7, 10 e 15 contêm uma única marca. Os implicantes primos que cobrem estes mintermos, são denominados implicantes primos essenciais e aparecem obrigatoriamente na fórmula mínima da função.

Implicantes	Mintermos de f_2									
	0	1	2	5	7	8	9	10	13	15
X00X	X	X				X	X			
X0X0	X		X			X		X		
XX01		X		X			X		X	
X1X1				X	X				X	X

Figura 1.8: Tabela dos implicantes primos de f_2.

Esta tabela será reduzida com a retirada de linhas por serem consideradas essenciais e de colunas por terem sido cobertas, dando origem à fórmula mínima da função. Assim é obtida a tabela da figura 1.9:

Implicante	1	9
X00X	X	X
XX01	X	X

Figura 1.9: Tabela reduzida dos implicantes primos de f_2.

Da figura 1.9, observa-se que falta cobrir apenas os mintermos 1 e 9, o que pode ser feito através dos implicantes primos X00X ou XX01. Neste caso, qualquer das escolhas resulta numa expressão mínima para a função f_2, já que ambos os implicantes possuem o mesmo número de literais.

Logo, para esta função são obtidas 2 expressões mínimas:

$$f_2 = \text{X0X0} + \text{X1X1} + \text{X00X} = \bar{\text{B}}\bar{\text{D}} + \text{BD} + \bar{\text{B}}\bar{\text{C}}$$

ou

$$f_2 = \text{X0X0} + \text{X1X1} + \text{XX01} = \bar{\text{B}}\bar{\text{D}} + \text{BD} + \bar{\text{C}}\text{D}$$

Apesar do procedimento de Quine-McCluskey produzir uma solução mínima exata, a complexidade computacional do algoritmo, que aumenta exponencialmente com o número de mintermos, torna-o inadequado para funções com mais de 10 variáveis devido a grande quantidade de memória utilizada e tempo de execução.

Em virtude disso, pesquisadores têm recorrido ao emprego de heurísticas, inicialmente gerando uma solução inicial e então melhorando-a iterativamente.

1.5 Algoritmo de Cobertura Irredundante Através da Geração de Padrão de Teste

Devido a ineficiência computacional dos métodos tradicionais de minimização de funções booleanas, vários algoritmos que geram soluções quase ótimas têm sido propostos na literatura [27] e [28].

Alguns seguem a estrutura das técnicas clássicas de minimização de funções lógicas, primeiro gerando todos os implicantes primos, mas no lugar de uma cobertura mínima, uma cobertura quase ótima é selecionada heuristicamente. Outros tentam simultaneamente identificar e selecionar implicantes, não necessariamente primos, para a cobertura.

Descreve-se a seguir, o algoritmo Geração [22], uma importante contribuição na minimização de funções booleanas, pois tem como principal característica o uso restrito de memória e simplicidade de implementação. O algoritmo obtém uma cobertura quase ótima de uma função booleana utilizando-se de algoritmos inicialmente desenvolvidos para a geração de padrões de teste de falhas.

Dada uma cobertura inicial da função, através de falhas f-e-1 nas entradas das portas AND's e falhas f-e-0 nas entradas da porta OR, as possíveis redundâncias da cobertura inicial da função são detectadas e eliminadas.

1.5.1 Conceitos e Definições

Circuitos lógicos estão sujeitos a falhas em seu funcionamento, podendo ser, por exemplo, resultado da deterioração física de algum componente ou algum defeito no processo de fabricação.

Define-se uma falha em um circuito lógico como sendo qualquer alteração que modifique a característica lógica deste circuito. Um teste para uma determinada falha é um padrão de sinais que aplicados nos pontos controláveis do circuito produzem nos pontos observáveis um valor lógico de acordo com a presença ou ausência de falha.

Diz-se que uma falha é detectável quando existir um vetor de entradas primárias que cause um erro em pelo menos uma de suas saídas primárias. Entende-se por erro um valor lógico distinto daquele do circuito sem defeito.

O teste de circuitos digitais é realizado aplicando-se a este circuito seqüências de estímulos nos terminais de entrada e observando-se as respostas nos terminais de saída. Comparando-se a

seqüência observada com a correspondente tabela verdade ou com a seqüência produzida por um circuito sem falhas, pode-se determinar a existência de erros.

Esses estímulos são denominados vetores de teste ou padrões de teste, onde cada estímulo é uma n-upla binária.

Cada padrão de teste tem associado a si um conjunto de falhas detectáveis. Diz-se neste caso, que o padrão de teste "cobre" estas falhas.

Um falha simples ao se propagar, do ponto com falha até um ponto observável, define um caminho que recebe o nome de caminho de propagação ou caminho sensibilizado.

Nos circuitos lógicos, quando não existir teste para uma determinada falha, esta é dita não detectável e o circuito é redundante com relação à falha. Este conceito é utilizado na simplificação das funções booleanas.

É sabido que uma expressão algébrica na forma canônica soma de produtos descrevendo a função, pode ser representada por um circuito lógico composto de inversores, portas AND's e uma porta OR, onde as variáveis de cada mintermo da função são as entradas para as portas AND's correspondentes.

Por exemplo, a função f(A,B) = AB + $A\bar{B}$ equivale ao circuito lógico apresentado na figura 1.10.

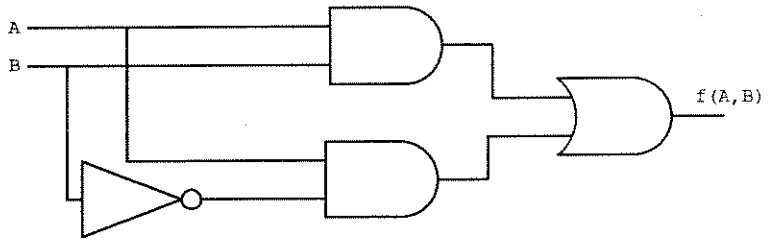


Figura 1.10: Circuito lógico representando a função f(A,B) = AB + $A\bar{B}$.

Considere que uma falha f-e-1 em uma das entradas de uma porta AND seja não detectável. Isto significa que o circuito possui uma redundância em relação ao ponto testado. Deste modo, a saída da porta AND não será alterada se fixar-se permanentemente esta entrada com o valor lógico "1".

Uma porta AND com n entradas, com uma delas fixa em "1", é logicamente equivalente a uma porta AND com n-1 entradas.

Do mesmo modo, uma falha f-e-0 não detectável na entrada de uma porta OR, equivale a eliminar a porta AND a ela conectada, o que na minimização equivale a eliminar um implicante da função.

É importante salientar que uma falha f-e-0 na entrada de uma porta AND equivale a uma falha f-e-0 na entrada da porta OR ao longo deste caminho.

Todas as falhas detectáveis ao longo de um caminho sensibilizado são ditas equivalentes. Se um conjunto de falhas são equivalentes, qualquer teste que detecta uma delas, irá detectar todas as outras. Deste modo, na geração de teste é necessário considerar somente uma falha de cada conjunto de falhas equivalentes.

1.5.2 O algoritmo Geração

O algoritmo de cobertura irredundante através da geração de padrões de teste utiliza-se da técnica de enumeração implícita na tentativa de encontrar um teste para a falha. Esta enumeração se faz através de uma pilha de teste que armazena as decisões nas variáveis e a porta que deu origem à decisão.

O algoritmo é bastante simples e utiliza-se de dois tipos de teste.

- Teste AND: Consiste em testar uma falha f-e-1 na entrada de uma porta AND;
- Teste OR: Consiste em testar uma falha f-e-0 na saída de uma porta AND, o que corresponde a testar uma falha f-e-0 na entrada da porta OR.

Deste modo, o algoritmo se resume em gerar testes para as falhas f-e-1 nas entradas das portas ANDs e gerar testes para as falhas f-e-0 nas entradas da porta OR.

As falhas f-e-0 nas entradas das porta AND's são equivalentes à falha f-e-0 na entrada correspondente da porta OR e são tratadas no teste OR, enquanto que as falhas f-e-1 nas entradas da porta OR são irrelevantes para o método, motivo pelo qual não são tratadas.

Aplicar um teste, significa verificar se a falha gerada pode ser propagada até a saída do circuito. ou seja, consiste na verificação da consistência do cubo teste.

Da aplicação de um teste, quatro situações podem ocorrer:

- Inconsistência no cubo teste (Conflito) :

  Um conflito ocorre quando não for possível garantir a propagação do sinal até a saída do circuito, indicando que o cubo teste produz o valor lógico 1 na saída de alguma outra porta AND do circuito.

- Inconsistência no cubo teste possuindo apenas uma posição redundante (Implicação) :

  A posição redundante deve ser alterada para garantir a propagação do sinal.

- Inconsistência no cubo teste possuindo mais de uma posição redundante (Decisão).

  Através da enumeração implícita, as posições redundantes são alteradas (decisão) para garantir a propagação do sinal. Esta enumeração se faz através de uma pilha de teste que armazena a decisão na variável e a porta que deu origem a esta decisão.

- Consistência no cubo teste: Neste caso, o cubo teste difere em pelo menos uma das posições das variáveis em cada porta AND, o que garante a propagação da falha.

1.5.3 Descrição detalhada do algoritmo Geração

- 1. Colocam-se os mintermos da função em uma fila de implicantes, ordenada de forma crescente de acordo com o número de seus sucessores à distância 1 (distância de Hamming [20]).

  Os mintermos com o mesmo número de sucessores à distância 1 são ordenados pelo número de sua representação decimal.

  Marca-se o início da fila à esquerda e o fim, à direita.

- 2. Anota-se para cada mintermo da fila as variáveis que, tendo seu valor complementado, levam a vértices à distância 1.

  Os mintermos que não possuem sucessores à distância 1 (vértices isolados) são retirados da fila, pois não serão cobertos por nenhum outro implicante.

- 3. Seleciona-se com um apontador o primeiro implicante da fila para efetuar o teste AND.

  O teste AND é gerado somente para as variáveis marcadas, visto que elas levam à mintermos a distância 1.

  Após gerado o cubo teste, verifica-se sua consistência através da aplicação a todos os outros implicantes da fila.

  A análise de consistência é feita para verificar se a falha se propaga para a saída do circuito. Esta operação é efetuada em duas etapas:

  Na primeira etapa, a aplicação é feita nos mintermos à direita do AND gerador do teste. Caso ocorra implicação, deve-se marcar o mintermo implicado para que, na ocorrência de um conflito, possa ser eliminada a variável sob teste e o mintermo onde ocorreu o conflito (inconsistência do teste), assim como os mintermos marcados.

  Isto corresponde à cobertura de todos os mintermos do implicante sob teste.

  Na segunda etapa, a aplicação é feita nos implicantes à esquerda do AND gerador do teste, somente se na primeira etapa não ocorreu conflito. Na ocorrência de implicação, o elemento implicado não deve ser marcado e no caso de conflito, deve-se eliminar somente a variável sob teste.

  Nesta aplicação sistemática, o cubo teste se modifica para garantir a consistência do teste. A determinação do teste se faz por enumeração implícita, isto é, a cada verificação de consistência, pode-se tomar decisões quanto ao valor das posições inicialmente redundantes, que só garantirão uma inconsistência uma vez esgotada todas as possibilidades de teste.

  Uma vez detectada uma inconsistência, elimina-se a variável sob teste, o que corresponde à expansão do implicante, e o mintermo onde ocorreu a inconsistência, o que corresponde a uma cobertura.

- 4. Se ainda existirem mintermos à direita do apontador, avança-se o apontador e retorna-se ao passo 3. Caso contrário, executa-se o passo 5.

- 5. Inserem-se os irrelevantes no final da fila e coloca-se o apontador no início da fila.

- 6. Repete-se o passo 3 apenas para os implicantes da fila.

- 7. Retorna-se o apontador ao início da fila e marca-se o fim da fila no último implicante. Note que todos os implicantes presentes na fila são implicantes primos.

- 8. Gere o teste OR para todos os implicantes da fila, aplicando-o aos demais implicantes primos.

  Em caso de inconsistência, elimina-se o implicante primo gerador do teste.

  Isto corresponde a reter na fila apenas os implicantes primos essenciais.

- 9. A cobertura irredundante da função corresponde a todos os implicantes primos que restaram na fila e os eventuais mintermos que foram separados no início do algoritmo por não possuírem sucessores à distância 1.

Um exemplo da utilização do algoritmo de Cobertura Irredundante através da Geração de Padrão de Teste, é apresentado detalhadamente.

Exemplo 5. Minimizar a função $f_5 = \sum(1,3,6,8) + D(10,11,12,13,14,15)$, cujo mapa de Karnaugh está apresentado na figura 1.11.

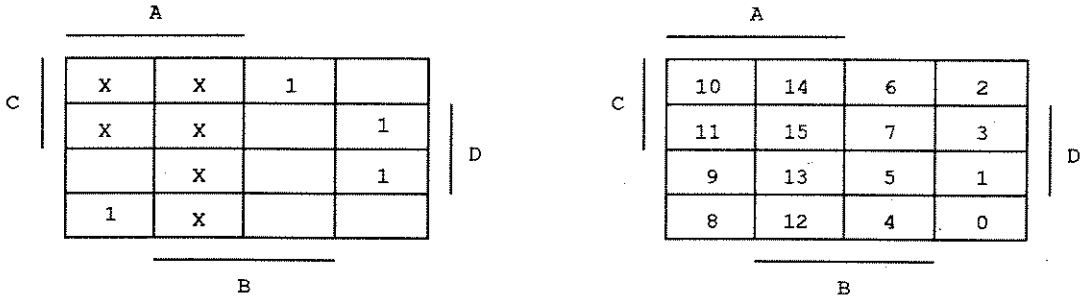


Figura 1.11: Mapa de Karnaugh para a função f_5.

Fila inicial dos mintermos: 1000 0110 0011 0001

Fila dos estados irrelevantes: 1111 1110 1101 1100 1011 1010

Fila dos mintermos após ordenação: 00$\dot{0}$1 $\dot{0}$110 $\dot{0}$0$\dot{1}$1 1$\dot{0}\dot{0}$0

Seleciona-se o primeiro mintermo na fila ordenada.

Gate gerador dos testes: 00$\dot{0}$1

Gere o teste AND para todas as variáveis marcadas neste mintermo iniciando pela mais significativa.

A única variável marcada é a de peso 1

Teste gerado para a variável de peso 1: 0011

Para gerar um teste, basta inverter o valor lógico da variável sob teste.

O teste gerado deve ser comparado com todos os outros mintermos da fila.

Gates comparados:

0110

0011 $\leftarrow$ Conflito.

Ocorreu conflito, pois o teste gerado coincide com os valores assumidos pelas variáveis do mintermo 3.

Neste caso, deve-se eliminar o mintermo onde ocorreu o conflito e a variável sob teste, o que corresponde em aumentar o implicante gerador do teste e eliminar o mintermo por ele coberto.

Fila após simplificação: 00X1 $\dot{0}$110 1$\dot{0}\dot{0}$0

Seleciona-se o próximo mintermo da fila para executar o teste AND.

Gate gerador dos testes: $\dot{0}$110

Teste gerado para a variável de peso 3: 1110

Gates comparados:

1000

00X1

Observe que inicialmente o teste é comparado com os elementos à direita do gate gerador do teste, para depois ser comparado com os implicantes à esquerda Seleciona-se o próximo mintermo da fila para gerar o teste AND.

Gate gerador dos testes: $1\dot{0}\dot{0}0$

Teste gerado para a variável de peso 2: 1100

Gates Comparados:

00X1

0110

Teste gerado para a variável de peso 1: 1010

Gates comparados:

00X1

0110

Após gerar o teste AND para todas as variáveis de todos os mintermos, deve-se incluir na fila de implicantes os eventuais irrelevantes e novamente verificar o teste AND para as variáveis ainda marcadas de todos os implicantes da fila de implicantes.

Fila após incluir os irrelevantes: 00X1 $\dot{0}$110 $1\dot{0}\dot{0}0$ 1111 1110 1101 1100 1011 1010

Gate gerador dos testes: 00X1

Para este implicante não foi gerado nenhum teste, pois a única variável, que tendo seu valor complementado leva a vértices à distância 1, já foi eliminada na fase anterior.

Gate gerador dos testes: $\dot{0}$110

Teste gerado para a variável de peso 3: 1110

Gates comparados:

1000

1111

1110 ← conflito

Fila após a simplificação: 00X1 X110 $1\dot{0}\dot{0}0$ 1111 1110 1101 1100 1011 1010

Seleciona-se o próximo implicante para o teste AND.

Gate gerador dos testes: $1\dot{0}\dot{0}0$

Teste gerado para a variável de peso 2: 1100

Gates comparados:

1111

1101

1100 ← conflito

Fila após simplificação: 00X1 X110 $1X\dot{0}0$ 1111 1110 1101 1011 1010

Teste gerado para a variável de peso 1: 1X10

Gates comparados:
1111
1101
1011
1010 ← Implicacão

Ocorreu uma implicação no cubo teste contendo uma posição redundante (posição que pode ser alterada para garantir a propagação da falha).

A posição redundante do teste deve ser fixada com o valor oposto à posição correspondente do elemento onde ocorreu a implicação.

Teste após sua alteração: 1110

Gates comparados:
00X1
X110 ← Conflito

O conflito ocorreu num elemento à esquerda do implicante gerador do teste devendo ser eliminada somente a variável geradora do teste.

Fila após simplificação: 00X1 X110 1XX0 1111 1110 1101 1011 1010

Após testar todos os implicantes da fila de implicantes, verifica-se a cobertura desses implicantes, gerando o teste OR para cada um deles e comparando com os outros implicantes da fila.

Cobertura dos implicantes:

Gate gerador dos teste: 00X1

Gates comparados:
X110
1XX0

Gate gerador dos testes: X110

Gates comparadados:
1XX0 ← Implicação

Teste após sua alteracação: 0110

Gates comparados:
00X1

Gate gerador dos testes: 1XX0

Gates comparados:
00X1
X110 ← Decisão

Ocorreu uma decisão no cubo teste, porém, com duas possibilidades de alteração para garantir a propagação. Optou-se por fixar a posição menos significativa com o valor oposto a da mesma posição do implicante que gerou o conflito.

Teste após sua alteração: 1X00

Decisão tomada:

Teste : 1XX0

Conflito: X110

Bit alterado: 1

O cubo teste anterior, a decisão, o elemento onde ocorreu o conflito e o bit do cubo teste

alterado são inseridos na pilha de decisões, para que no caso de um conflito, possa-se retornar a este ponto e tomar uma outra decisão que garanta a propagação da falha. Após verificar a cobertura para todos os implicantes, restará na fila de implicantes somente os implicantes primos essenciais da função.

Cobertura irredundante: 00X1 X110 1XX0

A figura 1.12 mostra o mapa de Karnaugh da função f_5, onde pode-se constatar que os implicantes 00X1, X110 e 1XX0 constituem-se numa cobertura irredundante da função.

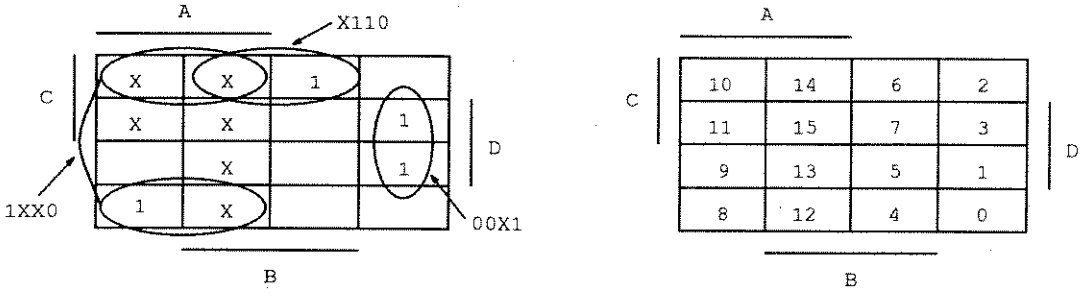


Figura 1.12: Cobertura irredundante para a função f_5.

1.6 Conclusão

Neste capítulo foram apresentados os conceitos básicos da álgebra booleana utilizadas em todo o trabalho.

As funções booleanas foram definidas a fim de descrever as propriedades dos circuitos combinacionais.

Nas seções 1.4 e 1.5 foram apresentados os algoritmos de Quine-McCluskey e o algoritmo Geração para a simplificação de funções booleanas.

O enfoque clássico na minimização consiste na obtenção de todos os implicantes primos que realizam a cobertura mínima da função.

Tais métodos são inadequados para considerar funções com mais de 10 variáveis, pois a fase de geração dos implicantes primos exige grande esforço de CPU e grande espaço em memória e a fase de cobertura, em geral, consome mais tempo ainda.

Portanto, é interessante a busca de métodos alternativos, com o objetivo de reduzir o uso de memória e tempo computacional.

O método Geração, que obtém uma cobertura irredundante (não se garante que a solução obtida é mínima) para uma dada função booleana, foi incluído neste trabalho com o objetivo de apresentar um algoritmo heurístico de simplificação de funções booleanas. Sua principal característica é a simplicidade de implementação com o conseqüente alto desempenho computacional. A fase de cobertura é feita simultaneamente à fase de geração dos implicantes primos.

Capítulo 2

Geração de Implicantes Primos Através do Consenso

Nos projetos de sistemas digitais envolvendo funções combinacionais, necessita-se de eficientes algoritmos computacionais para minimizar funções booleanas com um grande número de variáveis.

Neste capítulo apresenta-se o algoritmo GeraPlex, que obtém todos os implicantes primos de uma dada função booleana através de sucessivas aplicações da operação de consenso aos ramos de uma árvore, onde cada caminho (nó até folha) representam um mintermo ou irrelevante da função.

Resultados experimentais mostraram que o algoritmo proposto é uma alternativa eficiente quando comparado com o tradicional método tabular de geração dos implicantes primos.

2.1 Introdução

Na simplificação de funções booleanas de grande porte (tipicamente de 10 a 30 variáveis), o número de implicantes primos pode crescer de maneira explosiva ($\frac{3^n}{n}$ [39], onde n = número de variáveis), portanto, a obtenção de todos os implicantes primos requer grande quantidade de memória e tempo de execução.

Alguns autores tratam o problema de uma forma algébrica [31], [37], outros utilizam-se de tabelas e árvores [33], [34], [36] e [30].

Quine [32] e Tison [35] utilizam-se do consenso iterativo para achar todos os implicantes primos de uma função booleana.

Neste capítulo, apresenta-se o Geraplex, um algoritmo que obtém todos os implicantes primos de uma função booleana através de sucessivas aplicações do consenso aos ramos de uma árvore binária que representa a tabela verdade da função a ser minimizada. Nesta árvore, cada caminho, que vai da raiz a folha, representa um mintermo ou irrelevante da função.

Da aplicação da operação de consenso aos ramos da árvore três situações podem ocorrer: fusão, expansão e deslocamento, que diminui, aumenta e mantém o número de caminhos da árvore, respectivamente.

Resultados experimentais indicam que o método proposto é uma eficiente alternativa para a resolução do problema considerado, pois é de fácil manipulação para os operadores lógicos, e muito adequado para representar circuitos combinacionais.

2.2 O consenso entre dois termos produto

Dado dois termos produtos ψ_1 e ψ_2 possuindo uma única literal, que ocorre negada em um dos termos e não negada no outro, define-se consenso entre esses dois termos como sendo o produto ψ_3 obtido da justaposição de ψ_1 e ψ_2, eliminando-se esta literal e as repetições de quaisquer outras das literais.

Exemplo 1.

- O consenso de $\bar{B}$ e AB = A
- O consenso de $\bar{A}.\bar{B}.D$ e $A.\bar{B}.C.D = \bar{B}.C.D$
- O consenso de $\bar{A}.\bar{B}.D.E$ e $A.\bar{B}.C.D = \bar{B}.C.D.E$
- Não existe o consenso entre $\bar{A}.\bar{B}$ e A.B.C

Teorema 1. O consenso ψ_3 de ψ_1 e ψ_2 implica logicamente ψ_1 OR ψ_2 [38].

2.3 Método do consenso iterativo e eliminação para determinar todos os implicantes primos

O método do consenso iterativo [38] para a obtenção dos implicantes primos de uma função booleana, consiste na aplicação das operações de consenso e eliminação, até que estas operações não sejam mais aplicáveis.

- 1. Consenso: Acrescenta-se como termo produto, o consenso de dois termos, se este não for coberto por nenhum termo produto da soma;
- 2. Eliminação: Elimina-se qualquer termo produto que cobre outro.

Exemplo 2. Seja a seguinte soma de produtos:

$$\bar{A}.B.\bar{C}.D + \bar{A}.B.C.D + A.\bar{B}.\bar{C}.\bar{D} + A.\bar{B}.\bar{C}.D + A.B.\bar{C}.D$$

- O consenso entre os termos $A.\bar{B}.\bar{C}.\bar{D}$ e $A.\bar{B}.\bar{C}.D$ resulta no termo $A.\bar{B}.\bar{C}$.

$$\bar{A}.B.\bar{C}.D + \bar{A}.B.C.D + A.\bar{B}.\bar{C}.\bar{D} + A.\bar{B}.\bar{C}.D + A.B.\bar{C}.D + A.\bar{B}.\bar{C}$$

- Os termos $A.\bar{B}.\bar{C}.\bar{D}$ e $A.\bar{B}.\bar{C}.D$, cobertos por $A.\bar{B}.\bar{C}$, são eliminados.

$$\bar{A}.B.\bar{C}.D + \bar{A}.B.C.D + A.B.\bar{C}.D + A.\bar{B}.\bar{C}$$

- O Consenso entre os termos $A.B.\bar{C}.D$ e $A.\bar{B}.\bar{C}$ resulta no termo $A.\bar{C}.D$.

$$\bar{A}.B.\bar{C}.D + \bar{A}.B.C.D + A.B.\bar{C}.D + A.\bar{B}.\bar{C} + A.\bar{C}.D$$

- O termo $A.B.\bar{C}.D$, coberto por $A.\bar{C}.D$, é eliminado .

$$\bar{A}.B.\bar{C}.D + \bar{A}.B.C.D + A.\bar{B}.\bar{C} + A.\bar{C}.D$$

- Consenso entre os termos $\bar{A}.B.\bar{C}.D$ e $\bar{A}.B.C.D$ resulta no termo $\bar{A}.B.D$.

$$\bar{A}.B.\bar{C}.D + \bar{A}.B.C.D + A.\bar{B}.\bar{C} + A.\bar{C}.D + \bar{A}.B.D$$

- Os termos $\bar{A}.B.\bar{C}.D$ e $\bar{A}.B.C.D$, cobertos pelo termo $\bar{A}.B.D$, são eliminados.

$$A.\bar{B}.\bar{C} + A.\bar{C}.D + \bar{A}.B.D$$

- Consenso entre os termos $A.\bar{C}.D$ e $\bar{A}.B.D$ resulta no termo $B.\bar{C}.D$.

$$A.\bar{B}.\bar{C} + A.\bar{C}.D + \bar{A}.B.D + B.\bar{C}.D$$

- Consenso entre os termos $\bar{A}.B.D$ e $A.\bar{C}.D$ resulta no termo $B.\bar{C}.D$.

$$A.\bar{B}.\bar{C} + A.\bar{C}.D + \bar{A}.B.D + B.\bar{C}.D$$

As operações de consenso e eliminação já não podem ser aplicadas, portanto, $A.\bar{B}.\bar{C} + A.\bar{C}.D + \bar{A}.B.D + B.\bar{C}.D$ são os implicantes primos da função.

O Mapa de Karnaugh e os implicantes primos gerados pelo método do consenso, para a função do exemplo 2 são apresentados na figura 2.1.

Nota-se pelo Mapa de Karnaugh da figura 2.1, que todos os implicantes obtidos são implicantes primos, porém, esta cobertura não constitui-se numa cobertura mínima, pois o implicante primo $A.\bar{C}.D$ cobre mintermos já cobertos por outros implicantes primos.

O problema de cobertura mínima é tratado no capítulo 3.

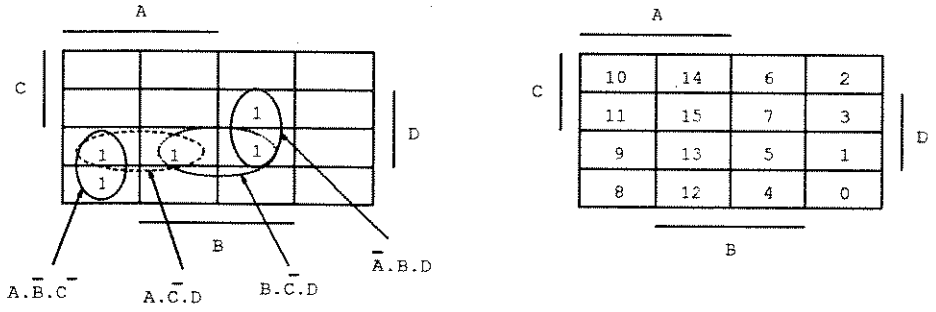


Figura 2.1: Mapa de Karnaugh para uma função de 4 variáveis.

2.4 Método proposto para a geração dos implicantes primos

O algoritmo proposto (GeraPlex) para a geração dos implicantes primos, de uma dada função booleana, utiliza-se da operação de consenso aplicada aos ramos de uma árvore binária que representa a função a ser minimizada.

Uma árvore binária é uma forma equivalente à tabela verdade, utilizada para representar os mintermos de uma função booleana. A figura 2.3 é uma representação equivalente à tabela verdade mostrada na figura 2.2.

Nota-se pela figura 2.3 que a raiz da árvore (variável A) representa o bit mais significativo e as folhas (variável D) representam os bits menos significativos. Um mintermo (linha da tabela verdade cuja função assume o valor 1) é representado por um caminho da árvore, que vai da raiz à folha. Dessa forma, o caminho 1001 representa o mintermo 9.

Esta forma de representação de uma função booleana é bastante conveniente para implementação computacional, visto que, um mesmo ramo pode ser utilizado por mais de um mintermo, o que significa menos utilização de memória.

No algoritmo GeraPlex, a geração dos implicantes primos tem início analisando-se a possibilidade de se aplicar o consenso aos ramos da árvore pertencentes aos nós do nível mais distante da raiz. Após aplicar o consenso aos ramos dos nós de um nível, avança-se para o próximo nível, no sentido da raiz. Desse modo, quando analisa-se um nó, todas as sub-árvores abaixo dele já devem ter sido analisadas.

A aplicação do consenso e eliminação aos ramos da árvore terá como efeito uma alteração na árvore original, que será caracterizada por uma das seguintes situações:

- Fusão (consenso seguido de duas eliminações): A fusão ocorre quando dois ramos de um nó geram um terceiro ramo. O ramo gerado cobre os outros dois ramos do nó. Neste caso os ramos cobertos são eliminados.

  No Mapa de Karnaugh, a fusão corresponde à geração de um implicante a partir de dois mintermos, ou à geração de um implicante de dimensão maior a partir de dois implicantes. O implicante gerado cobre os dois implicantes já existentes.

  Exemplo 3. A figura 2.4a apresenta uma parte da árvore binária que representa a função, cujo Mapa de Karnaugh está apresentado na figura 2.4b.

Dec	A	B	C	D	f
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Figura 2.2: Tabela verdade para uma função booleana.

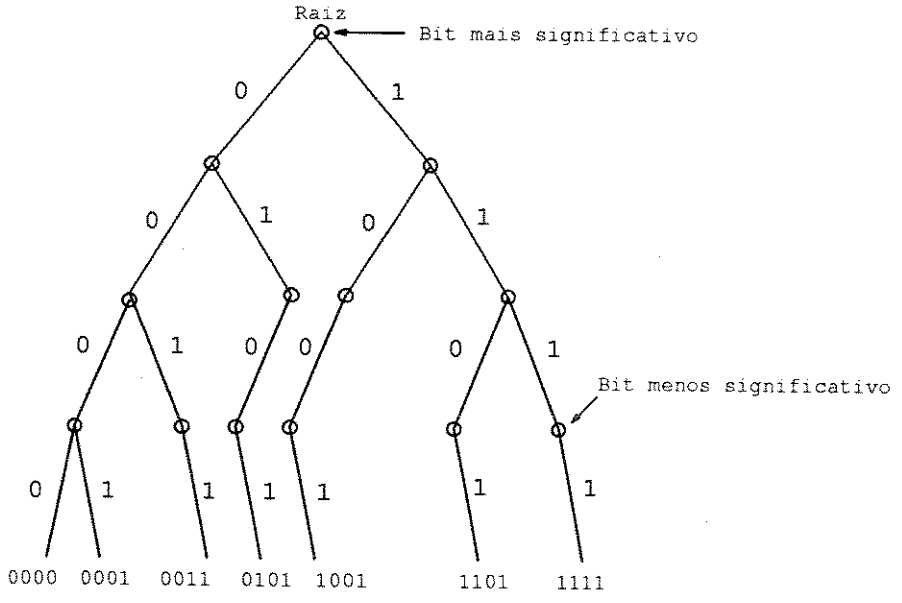
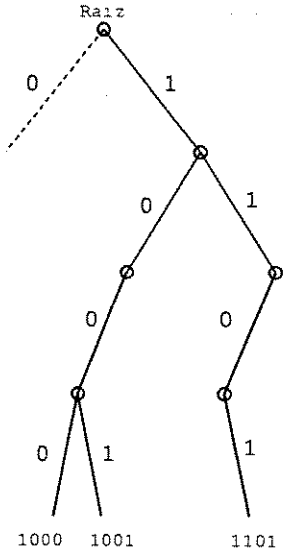
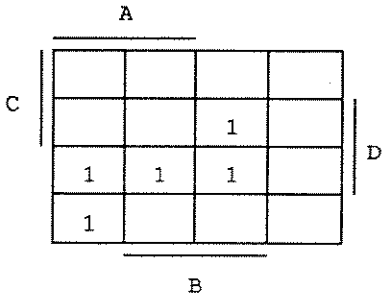


Figura 2.3: Diagrama binário para a função apresentada na figura 2.2.



A) Parte da árvore binária de uma função booleana



B) Mapa de Karnaugh de uma função booleana.

Figura 2.4: Mapa de Karnaugh e parte do diagrama binário e de uma função booleana utilizada como exemplo para a aplicação do GeraPlex.

Aplicando-se o consenso aos ramos dos bits menos significativos (nível 0) dos caminhos 1000 e 1001, obtém-se o novo caminho 100X. O X na posição menos significativa indica que essa posição pode assumir o valor lógico "1" ou "0".

Note na figura 2.5a que na árvore foi gerado o caminho 100X e que os caminhos 1000 e 1001 foram eliminados. No Mapa de Karnaugh, corresponde à geração do implicante primo 100X a partir dos mintermos 1000 e 1001, como pode ser visto na figura 2.5b.

Esta operação diminui o número de caminhos da árvore.

- Deslocamento (consenso seguido de uma eliminação): O deslocamento ocorre quando dois ramos de um nó geram um terceiro ramo, porém, o ramo gerado cobre apenas um dos outros dois ramos. Neste caso, o ramo coberto é eliminado.

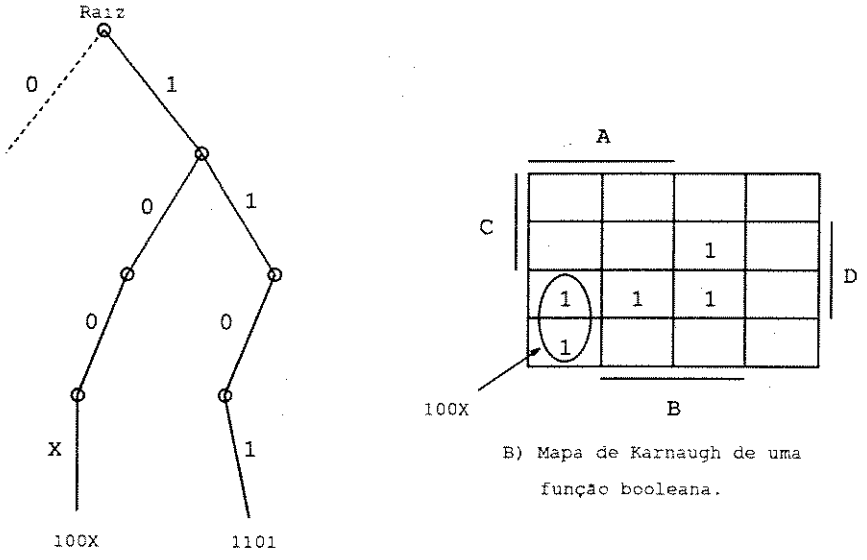
  No Mapa de Karnaugh, o deslocamento corresponde à geração de um implicante, utilizando-se parte de um outro implicante. O implicante gerado cobre apenas um dos implicantes.

  Exemplo 4. Aplicando-se, na árvore apresentada na figura 2.5a, o consenso aos ramos do nó de nível 2, cujos caminhos são 100X e 1101, obtém-se o caminho 1X01.

  Note na figura 2.6a que foi gerado na árvore o caminho 1X01 e eliminado o caminho 1101. No Mapa de Karnaugh, corresponde à geração do implicante 1X01, como pode ser visto na figura 2.6b.

  Esta operação mantém o número de caminhos da árvore.

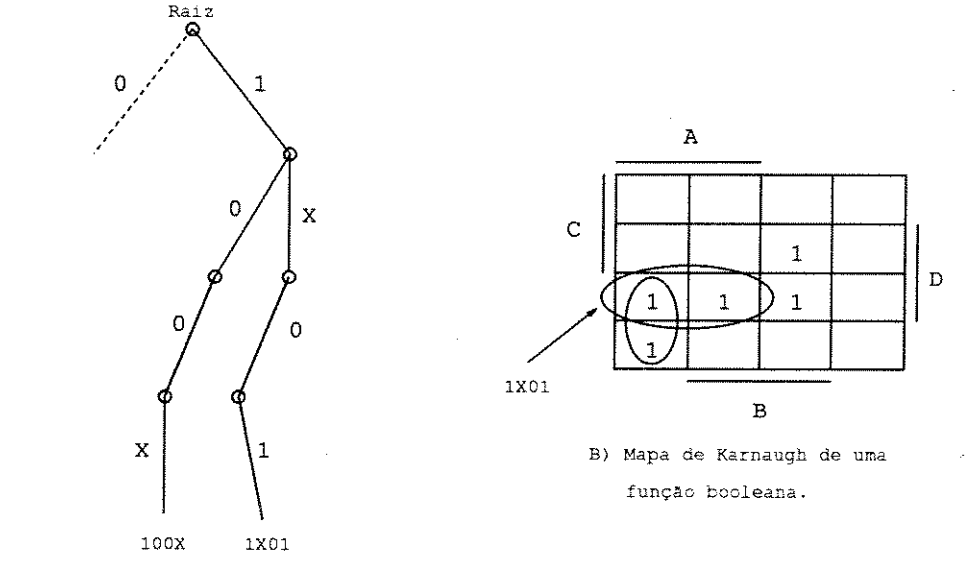
- Expansão (consenso sem eliminação): A expansão ocorre quando dois ramos de um nó geram um terceiro ramo, porém o ramo gerado não cobre nenhum dos outros dois.



A) Parte da árvore binária de uma função booleana

B) Mapa de Karnaugh de uma função booleana.

Figura 2.5: Aplicação do consenso resultando numa fusão.



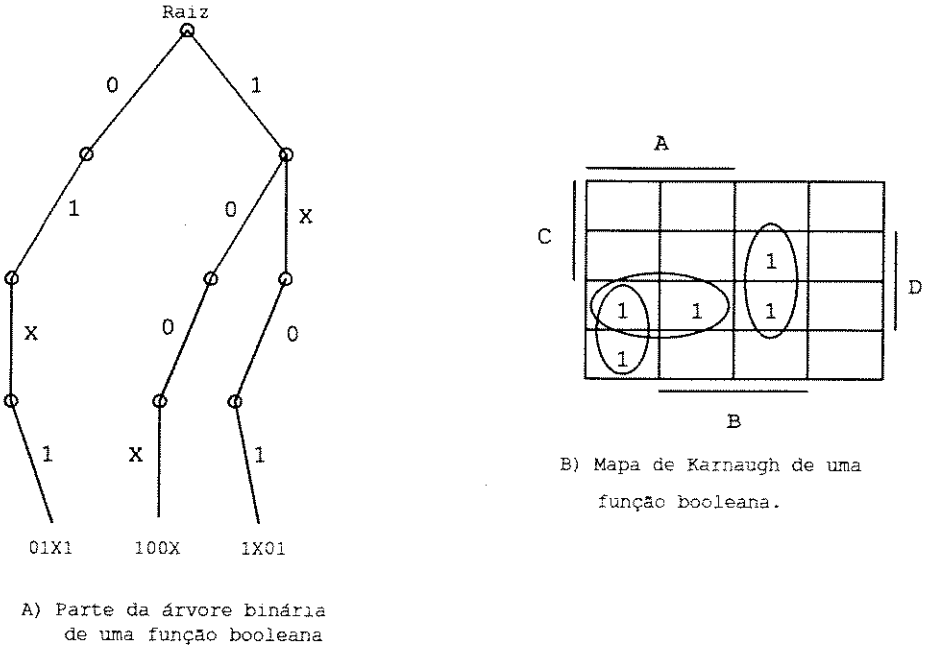
A) Parte da árvore binária de uma função booleana

B) Mapa de Karnaugh de uma função booleana.

Figura 2.6: Aplicação do consenso resultando num deslocamento.

No Mapa de Karnaugh, a expansão corresponde à geração de um implicante a partir de dois outros, sendo que o implicante gerado não cobre nenhum dos outros dois.

Exemplo 5. A figura 2.7a apresenta parte da árvore binária, que representa a função cujo Mapa de Karnaugh está apresentado na figura 2.7b.



A) Parte da árvore binária de uma função booleana

B) Mapa de Karnaugh de uma função booleana.

Figura 2.7: Aplicação do consenso resultando num expansão.

Aplicando-se o consenso aos ramos do nó de nível 3, cujos caminhos são 01X1 e 1X01, obtém-se o caminho X101.

Note que na figura 2.8a foi introduzido o caminho X101 e que nenhum outro caminho da árvore foi eliminado.

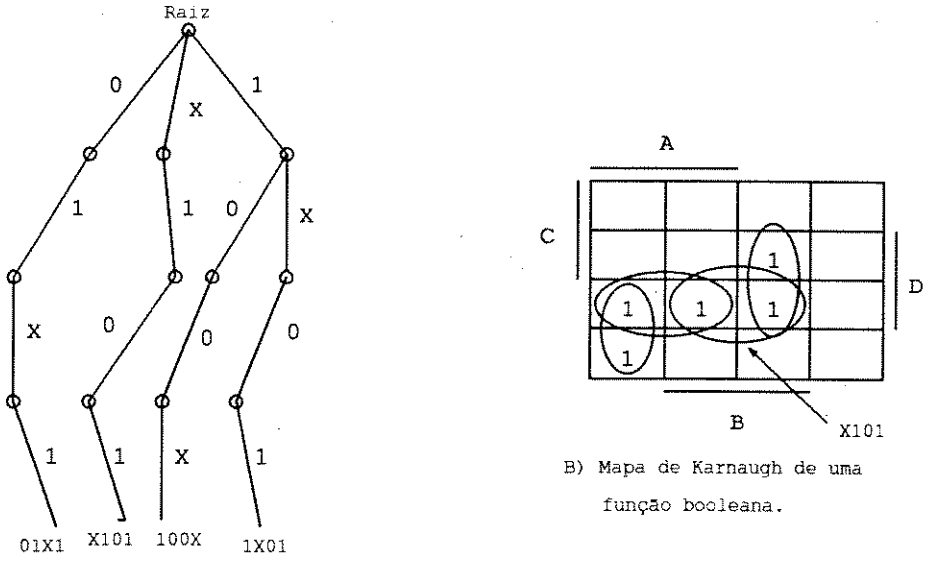
2.4.1 Descrição resumida do algoritmo GeraPlex

O algoritmo GeraPlex obtém os implicantes primos de uma função booleana, e, além disso, determina os implicantes primos essenciais, com vistas a formulação de um problema de programação matemática que minimiza a representação da função.

- 1. Montagem das Árvores

  Monta-se duas árvores com a mesma estrutura; uma contendo somente os mintermos da função (árvore dos mintermos) e a outra contendo os mintermos e os irrelevantes (árvore dos implicantes). A árvore dos mintermos é utilizada no final do algoritmo para se determinar quais dos implicantes primos gerados são essenciais. Na árvore de implicantes são gerados os implicantes primos da função através das operações de fusão, expansão e deslocamento.

  Na lista ligada em estrutura de árvore utilizada pelo algoritmo, cada registro tem três apontadores rotulados por "zero", "um" e "dc".



A) Parte da árvore binária de uma função booleana

B) Mapa de Karnaugh de uma função booleana.

Figura 2.8: Diagrama binário e Mapa de Karnaugh de uma função booleana.

Os mintermos e irrelevantes da função são representados por um caminho que vai da raiz (bit mais significativo) a uma folha (bit menos significativo) da árvore, onde os apontadores utilizados pelo caminho são os dígitos que representam os mintermos ou os irrelevantes.

- 2. Geração dos implicantes primos

  Os implicantes primos da função são gerados na árvore de implicantes através de sucessivas aplicações do Método do Consenso aos ramos da árvore.

  Para evitar a geração de implicantes não primos, deve-se primeiro aplicar, para nós no mesmo nível da árvore o consenso a ramos que produzem a fusão, para depois aplicar o consenso a ramos que produzem o deslocamento ou a expansão.

  Para cada nível analisado, deve-se eliminar os implicantes não primos.

  Após analisar o nó da raiz, a árvore de implicantes só contém caminhos que representam os implicantes primos da função.

  O passo seguinte, consiste em determinar os caminhos que representam os implicantes primos essenciais da função.

- 3. Determinação dos implicantes primos essenciais:

  - 3.1. Seleciona-se um mintermo (caminho) na árvore de mintermos;
  - 3.2. Verifica-se na árvore de implicantes quantos cobrem o mintermo selecionado;
  - 3.3. Se existe apenas um implicante que cobre o mintermo selecionado, este é considerado essencial, sendo removido da árvore. Elimina-se também todos os mintermos cobertos pelo implicante eliminado;
  - 3.4. Retorna-se o passo 3.1 até que todos os mintermos tenham sido analisados.

No final deste procedimento, a árvore de implicantes contém somente implicantes primos que não são essenciais. Isto corresponde, no Mapa de Karnaugh, a um caso cíclico, podendo ser formulado como um problema de programação linear inteira.

Exemplo 6: Seja a função de 3 variáveis f(a,b,c), mostrada no Mapa de Karnaugh da figura 2.9.

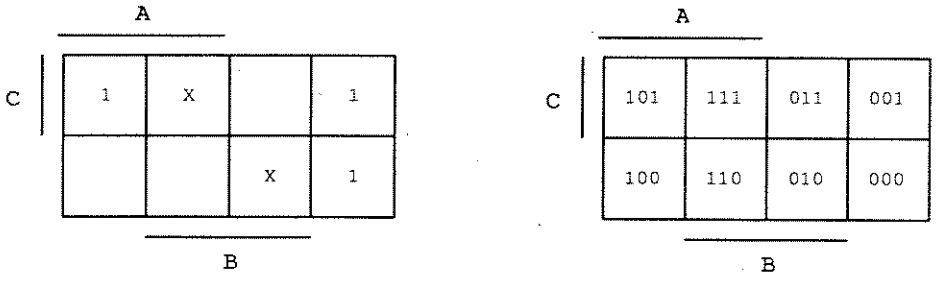


Figura 2.9: Mapa de Karnaugh para a função f(a,b,c)

A árvore inicial dos implicantes primos é mostrada na figura 2.10. A geração dos implicantes primos inicia-se pela aplicação do consenso aos ramos dos nós do nível 0. A árvore da figura 2.11, apresenta o resultado da aplicação do consenso aos caminhos 000 e 001, gerando o caminho 00X. Note que esta é uma operação de fusão, pois o caminho gerado cobre os outro dois.

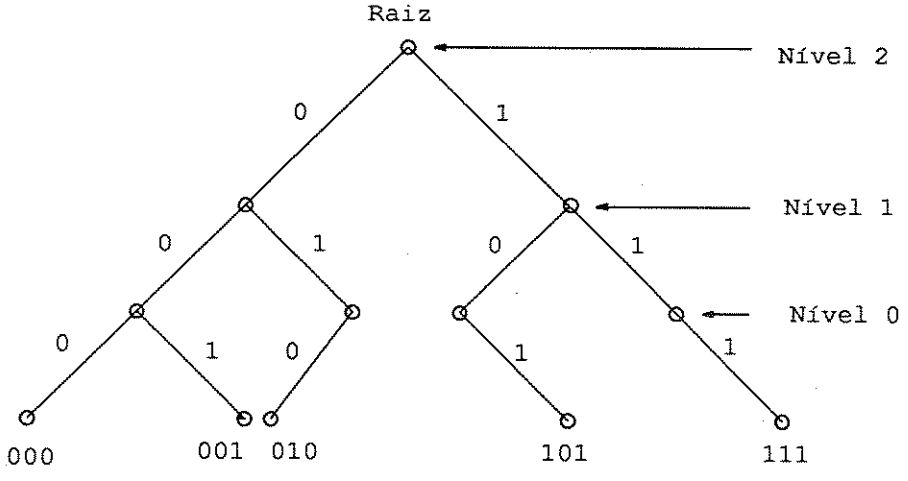


Figura 2.10: Árvore de implicantes primos para o exemplo 6.

Após aplicar o consenso aos ramos de um nível da árvore, avança-se para os nós do próximo nível no sentido da raiz.

A árvore da figura 2.12 apresenta o resultado da aplicação do consenso aos ramos dos nós de nível 1. O consenso entre os caminhos 101 e 111, gera o caminho 1X1, que cobre os outros dois caminhos. Observe que o consenso entre os caminhos 00X e 010 gera o caminho 0X0, que cobre somente o caminho 010. Esta operação é definida como deslocamento.

Para o nó de nível 2, na árvore apresentada na figura 2.12, o consenso aplicado aos caminhos 00X e 1X1 gera o caminho X01. O caminho gerado não cobre nenhum dos outros dois caminhos. Essa operação é chamada de expansão, aumenta o número de caminhos na árvore, como pode ser observado na figura 2.13.

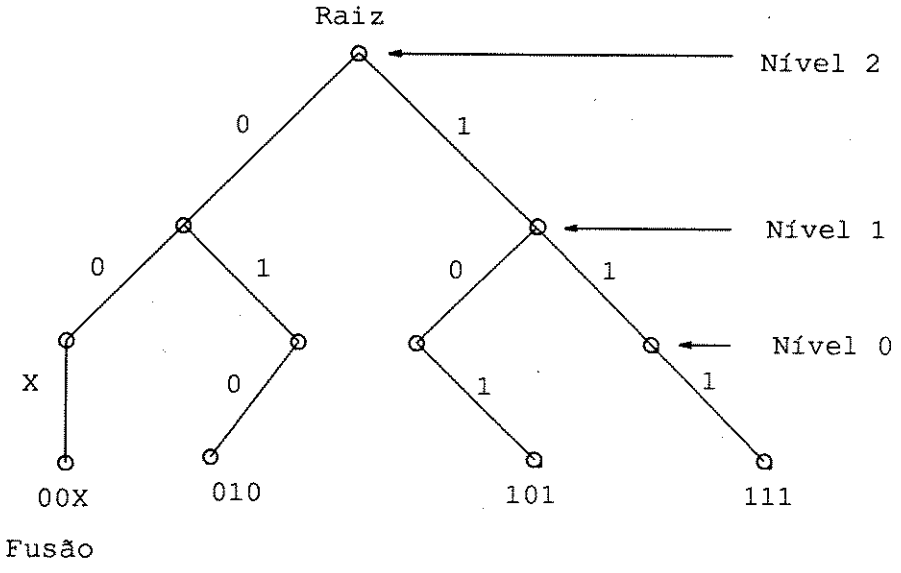


Figura 2.11: Árvore de implicantes após a aplicação do consenso aos nós do nível 0.

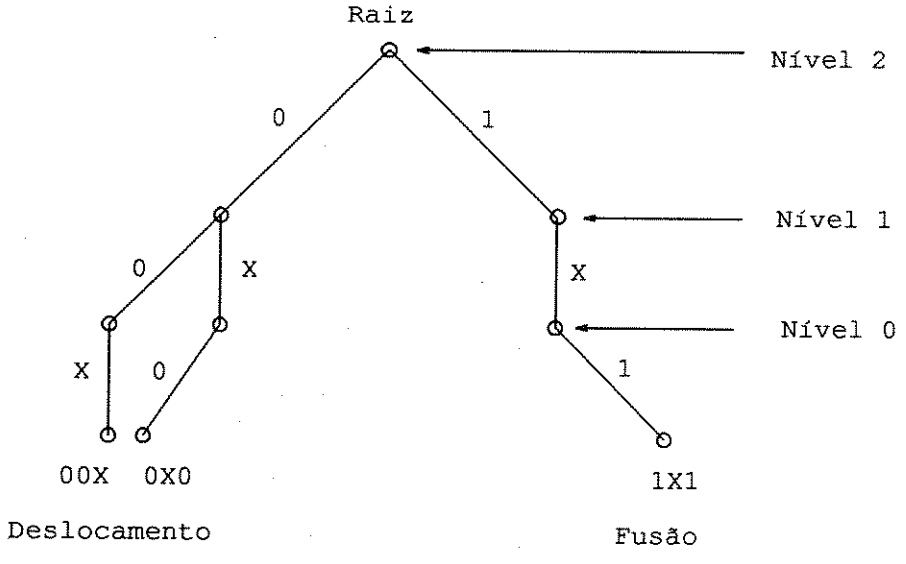


Figura 2.12: Árvore de implicantes após a aplicação do consenso aos nós do nível 1.

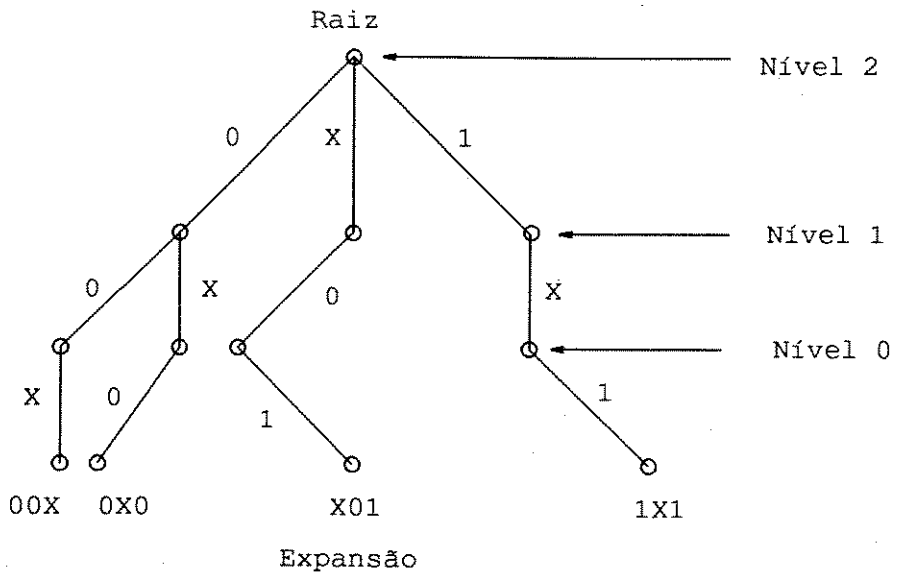


Figura 2.13: Árvore de implicantes após a aplicação do consenso aos nós do nível 2.

Observe que na árvore apresentada na figura 2.13, não se pode mais aplicar o consenso. Todos os caminhos da árvore representam implicantes primos da função, como mostra o Mapa de Karnaugh da figura 2.14.

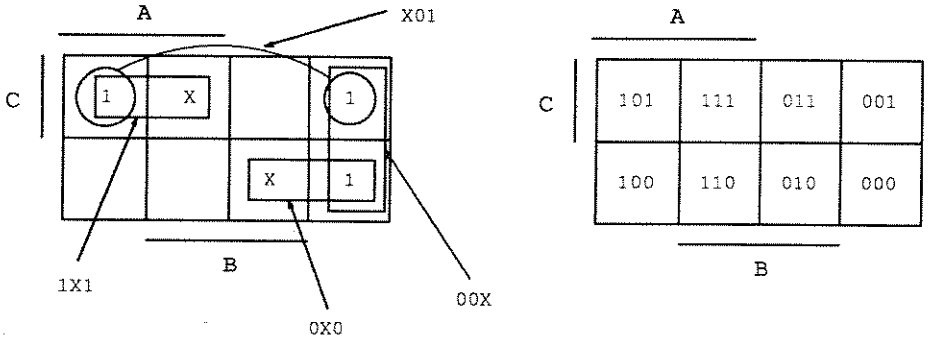


Figura 2.14: Mapa de Karnaugh apresentando a cobertura para a função apresentada na figura 2.9.

Exemplo 7. Seja a função f(a,b,c), figura 2.15, cuja árvore de implicantes primos é apresentada na figura 2.16.

As figuras 2.17, 2.18 e 2.19 mostram as transformações ocorridas na árvore de implicantes primos, decorrente da aplicação do GeraPlex.

Com o propósito de validar o algoritmo GeraPlex, dezenas de casos de funções booleanas foram simplificadas e comparadas com o resultado obtido pelo método de Quine-McCluskey.

Para todos os casos estudados, o GeraPlex obteve os mesmos implicantes primos obtidos pelo método tabular.

Também foram estudadas funções denominadas Tudo Um, isto é, funções contendo todos os mintermos possíveis. Estas funções foram consideradas, pois exigem grande espaço em memória.

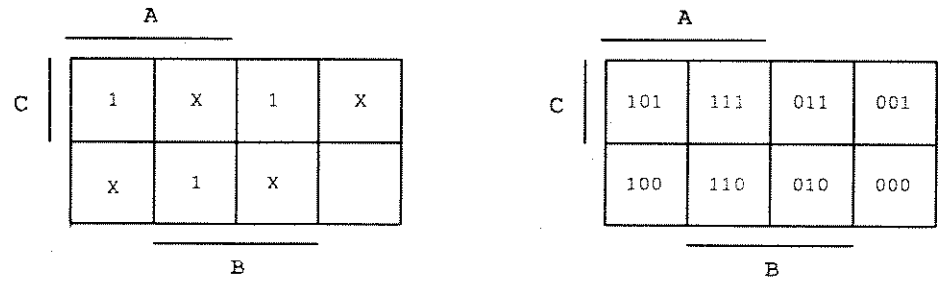


Figura 2.15: Mapa de Karnaugh para uma função de três variáveis.

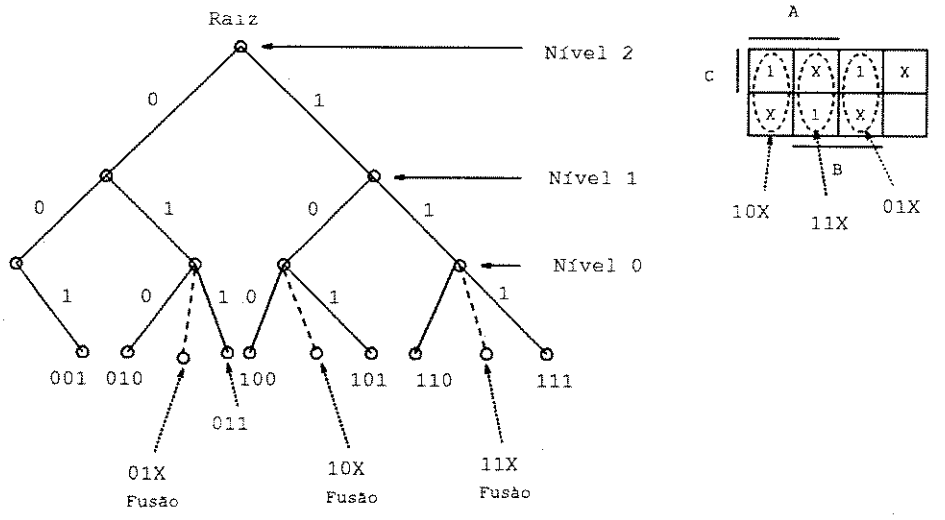


Figura 2.16: Árvore de implicantes primos para a função apresentada na figura 2.15.

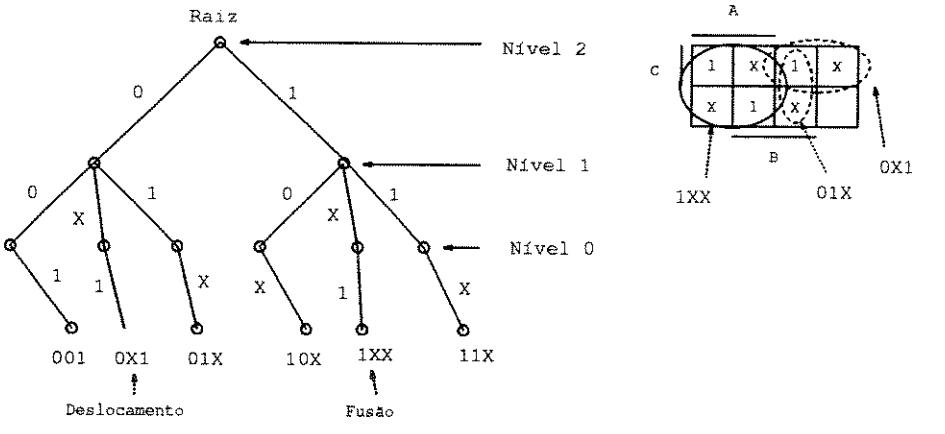


Figura 2.17: Árvore de implicantes primos após a aplicação do consenso aos ramos do nível 0.

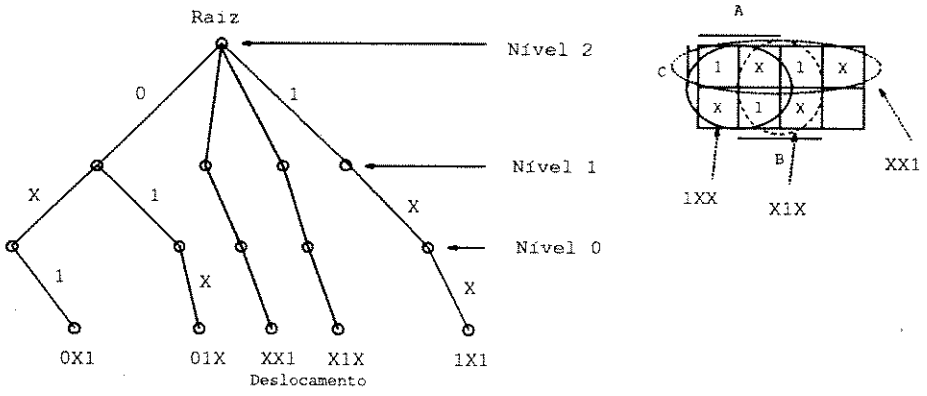


Figura 2.18: Árvore de implicantes primos após a aplicação do consenso aos ramos do nível 1.

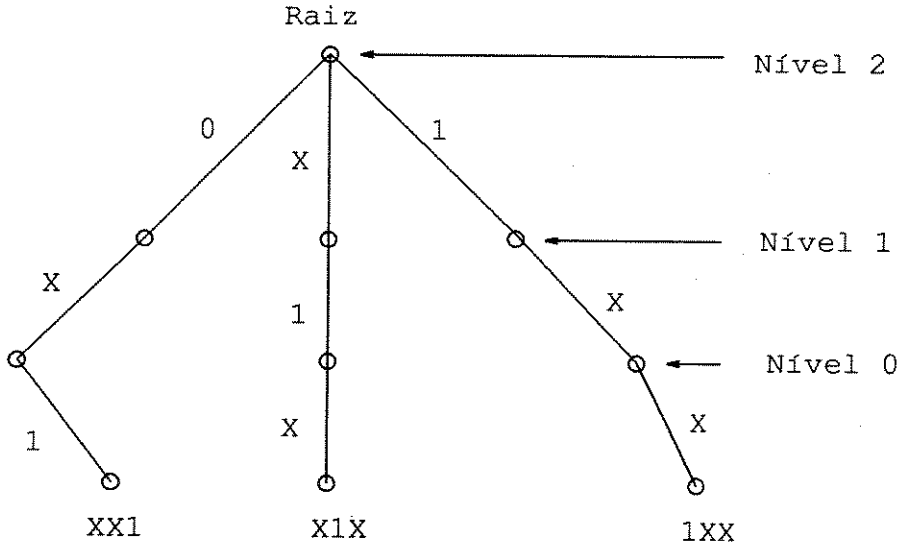


Figura 2.19: Árvore de implicantes primos após a aplicação do consenso aos ramos do nível 2.

2.5 Conclusão

Este capítulo apresentou uma alternativa eficiente para gerar todos os implicantes primos de uma função booleana.

Os implicantes da função são gerados aplicando-se a operação de consenso e eliminação aos ramos de uma árvore binária, cujos caminhos (raiz até folha) representam os mintermos e irrelevantes da função.

Da aplicação da operação de consenso aos ramos da árvores, três situações podem ocorrer: fusão, expansão e deslocamento, que diminui, aumenta e mantém, respectivamente, os caminhos na árvore.

No final do processo, os caminhos que permanecem na árvore representam os implicantes primos da função, que no Método de Quine-McCluskey corresponde a um Mapa Cíclico.

Dezenas de funções booleanas foram minimizadas, e os resultados obtidos, comparados com o tradicional método tabular para a geração de implicantes primos. Em todos os casos, o GeraPlex obteve os mesmos implicantes primos e apresentou desempenho superior em tempo e memória.

Para testar o uso de memória, utilizou-se a classe de funções booleanas, aqui chamada Tudo Um, que contém todos os possíveis mintermos. Essas funções não têm utilidade prática, mas constituem-se num bom teste para medir o desempenho computacional de um algoritmo para a minimização de funções booleanas.

Capítulo 3

Cobertura de Funções Booleanas Através de Programação Inteira 0 e 1

Toda cobertura mínima de uma função booleana é solução de um problema de programação linear inteira 0 e 1.

Este capítulo apresenta o problema de cobertura de funções booleanas sob a óptica da programação linear inteira 0 e 1 e propõe um método de resolução baseado no algoritmo Simplex, modificado para considerar as particularidades do problema. No lugar do pivoteamento tradicional, foram utilizadas combinações lineares das linhas do tableau para preservar os valores inteiros dos seus coeficientes.

Encontrada uma solução otimista (solução ótima, porém infactível), utiliza-se o Plano de Corte de Gomory para levar em conta somente as restrições de soluções inteiras.

O algoritmo, intitulado Plex, teve tempo de execução menor que o programa Espresso [39] e em todos os casos obteve a cobertura mínima, o mesmo não ocorrendo para o Espresso.

3.1 Introdução

A obtenção de cobertura de uma função booleana com o menor número de termos produtos pode ser resolvido em dois passos.

O primeiro requer a geração de todos os implicantes primos da função. Vários algoritmos tratam do assunto. No capítulo 2, foi apresentado o algoritmo GeraPlex, que obtém todos os implicantes primos de uma função booleana através da aplicação da operação de consenso aos ramos de uma árvore binária, onde cada caminho da árvore representa um mintermo ou irrelevante da função.

O segundo passo, envolve a seleção de um conjunto mínimo de implicantes e é chamado problema de cobertura.

As técnicas clássicas [45], [46], para a obtenção de uma cobertura mínima, utilizam algumas regras para simplificar a tabela de cobertura. Essas regras consistem na busca de linhas dominantes. colunas dominantes e a remoção de linhas essenciais.

Essas regras são aplicadas até que todos os mintermos estejam cobertos ou até as regras não poderem ser mais aplicadas.

Foi considerado por vários autores [42], [43], [44], [50] e [54] que o problema de cobertura pode ser expresso como um problema linear restritamente inteiro.

Neste capítulo apresenta-se o algoritmo Plex, que obtém uma cobertura mínima de uma função booleana através da solução de um problema de programação linear inteira 0 e 1.

3.2 O problema de cobertura sob a óptica da programação matemática

Toda cobertura mínima de uma função booleana é solução do seguinte problema de programação linear inteira.

critério C: O custo de uma cobertura é igual a soma aritmética ponderada de todos os sub-cubos de $(B^*)^n$.

restrição 1: Para toda n-upla, pertencente a UM, a soma aritmética dos sub-cubos, associados à n-upla, é maior ou igual a 1.

restrição 0: Para toda n-upla, pertencente a ZERO, a soma aritmética dos sub-cubos, associados à n-upla, é igual a zero.

Problema: Minimizar o critério C, escolhendo-se os sub-cubos, tal que as restrições 1 e 0 sejam satisfeitas.

Com o intuito de permitir a comparação entre o método aqui apresentado e as soluções clássicas de simplificações de funções booleanas, define-se como custo de uma porta lógica o seu número de entradas.

Por exemplo, a função F(AB) = A.B + A.$\bar{B}$, apresentada na figura 3.1 tem um custo igual a 6, ou seja, o custo é dado pelo fan-in das portas ANDs e da porta OR. Note que o inversor tem custo zero.

Exemplo 1: Seja a função de 3 variáveis f(A,B,C), mostrada no mapa de Karnaugh da figura 3.2.

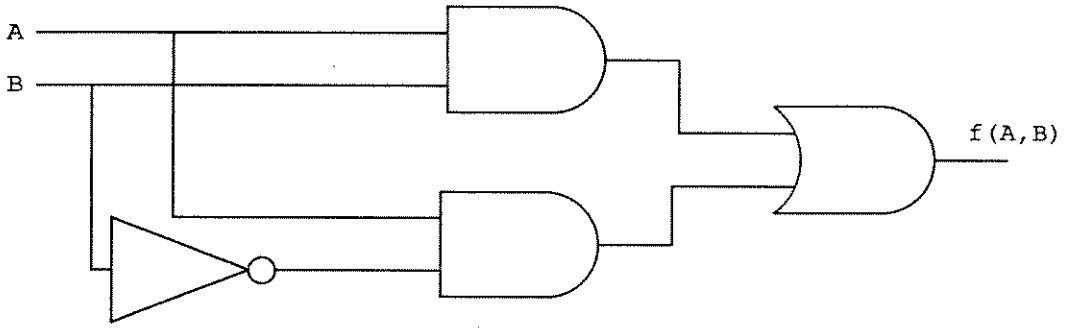


Figura 3.1: Circuito lógico representando a função F(AB) = A.B + A.$\bar{B}$, cujo custo é igual a 6.

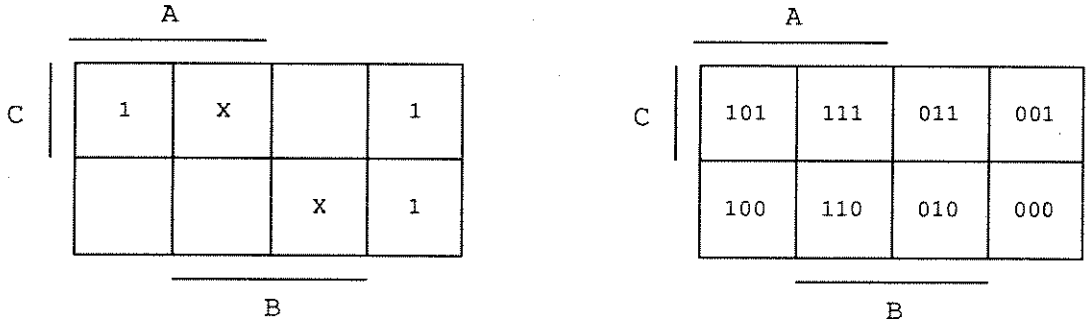


Figura 3.2: Mapa de Karnaugh para a função f(A,B,C)

O critério de custo a ser minimizado é a soma ponderada dos sub-cubos que cobrem as n-uplas pertencente ao conjunto UM e ao conjunto ZERO.

As restrições 1 são inequações do tipo " ≥ 1 " cujo lado esquerdo é a soma dos implicantes primos (sub-cubos) que cobrem um mintermo (n-upla). Têm-se tantas restrições 1 quanto forem os mintermos da função.

As restrições 0 são equações do tipo " $= 0$ ", cujo lado esquerdo é a soma dos implicantes primos que cobrem os maxtermos da função.

O problema pode ser formulado como:

Critério de custo:

$$1.(0XX + X0X + XX0 + 1XX + X1X + XX1) + 3.(00X + 0X0 + X00 + X00 + 01X + 0X1 + X01 + 10X + 1X0 + X10 + 11X + 1X1 + X11) + 4.(000 + 001 + 010 + 011 + 100 + 101 + 110 + 111)$$

Restrições 1:

$$101 + 10X + 1X1 + X01 + 1XX + X0X + XX1 + XXX \geq 1$$
$$000 + 00X + 0X0 + X00 + 0XX + X0X + XX0 + XXX \geq 1$$
$$001 + 00X + 0X1 + X01 + 0XX + X0X + XX1 + XXX \geq 1$$

Restrições 0:

$$011 + 01X + 0X1 + X11 + 0XX + X1X + XX1 + XXX = 0$$
$$100 + 10X + 1X0 + X00 + 1XX + X0X + XX0 + XXX = 0$$
$$110 + 1XX + 1X0 + X10 + 1XX + X1X + XX0 + XXX = 0$$

Onde cada tripla é uma variável aritmética pertencente ao conjunto { 0, 1 }.

Observe que, cada estado irrelevante corresponde a uma equação ou inequação a menos, resultando num maior grau de liberdade na busca da solução.

As restrições 0 possuem soluções triviais e considerando-se algumas simplificações o problema pode ser escrito como:

MIN 3.(00X + X01 + 0X0 + 1X1)

$$X01 + 1X1 \geq 1$$
$$00X + X01 \geq 1$$
$$00X + 0X0 \geq 1$$
00X, X01, 0X0, 1X1 $\in$ { 0, 1 }

Cuja solução é f(A,B,C) = 0X0 + X01 = $\overline{A}.\overline{C} + \overline{B}.C$, com Custo = 6.

Conforme apresentado, o problema de simplificar funções booleanas com n variáveis, pode ser modelado como um problema de programação linear inteira (0 ou 1) com 3^n variáveis e 2^n restrições, cada uma contendo 2^n variáveis.

Para n = 10, tem-se 59.049 variáveis e 1024 restrições e para n = 30 tem-se aproximadamente 200 trilhões de variáveis com 1 bilhão de restrições.

É evidente que todas as características especiais deste problema devem ser exploradas com o intuito de reduzir as dimensões do problema.

É compreensível também que tenha aparecido na literatura uma série de algoritmos que propõem soluções quase ótimas para esta classe de problemas [25]-[27]-[28]-[60].

O método clássico de Quine-McCluskey [20] pode ser visto como um método intuitivo de resolução deste problema linear.

A fase de geração dos implicantes primos pode ser entendida, na óptica de programação matemática, como a formulação do problema, isto é, na obtenção explícita das restrições e na resolução das equações triviais.

Assim, as restrições de igualdade (restrições Zero) são resolvidas implicitamente na fase de geração dos implicantes primos.

A fase de cobertura dos mintermos corresponde, na programação matemática, à solução do problema de minimização propriamente dito.

Exemplo 2: Considere a função booleana cujo Mapa de Karnaugh está apresentado na figura 3.3.

Aplicando-se o algoritmo GeraPlex (fase de geração dos implicantes primos), obtém-se os seguintes implicantes primos: 1XX0, 111X, X111, 01X1, 0X11, X100 e 010X.

A tabela de implicantes primos para o exemplo é mostrado na figura 3.4.

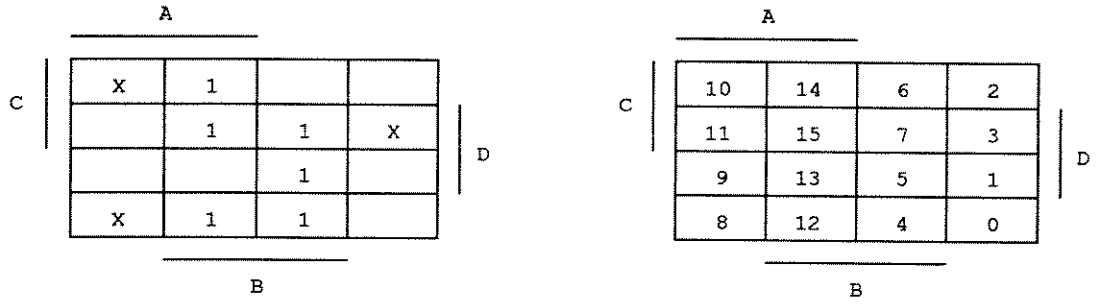


Figura 3.3: Mapa de Karnaugh para a função f(A,B,C,D)

	4	5	7	12	14	15
1XX0				X	X	
111X					X	X
X111			X			X
01X1		X	X			
0X11			X			
X100	X			X		
010X	X	X				

Figura 3.4: Tabela de implicantes primos para a função apresentada na figura 5.2.

Observe que na tabela de implicantes primos, tem-se uma linha para cada implicante primo e uma coluna para cada mintermo da função. Os irrelevantes não necessitam de cobertura, portanto, não aparecem na tabela. Note também que cada coluna possui pelo menos duas marcas "X", que no método de Quine-McCluskey corresponde a um caso cíclico.

Da tabela cíclica pode-se formular um problema de programação linear inteira 0 e 1, onde os implicantes primos da função são as variáveis do problema e as restrições são inequações de " ≥ 1 ", cujo lado esquerdo é a soma dos implicantes primos que cobrem cada mintermo.

Desse modo, têm-se tantas variáveis quantos forem os implicantes primos e tantas restrições quantos forem os mintermos.

Considerando-se a = 1XX0, b = 111X, c = X111, d = 01X1, e = 0X11, f = X100 e g = 010X tem-se o seguinte problema de programação linear:

MIN 3.a + 4(b + c + d + e + f + g)

$g + f \geq 1$
$g + d \geq 1$
$e + d + c \geq 1$
$f + a \geq 1$
$b + a \geq 1$
$c + b \geq 1$
a, b, c, d, e, f, g $\in$ { 0,1 }

Observe que nesse caso, o problema de programação matemática não tem solução trivial e corresponde, no método de Quine-McCluskey, a um mapa cíclico.

Observe também que a função objetivo é dada pela soma ponderada dos implicantes primos e que as variáveis são os implicantes primos que cobrem cada mintermo.

A restrição g + f $\geq$ 1 refere-se ao mintermo 4. Pela tabela de implicantes apresentada na figura 3.4 pode-se verificar que na coluna referente ao mintermo 4, tem-se uma marca "X" nas linhas X100 e 010X, ou seja, esses implicantes cobrem o mintermo em questão.

Apresenta-se, a seguir, os passos do algoritmo Plex, que resolve o problema de programação linear inteira, formulado a partir de uma tabela de implicantes primos cíclica. O Plex é uma adaptação do algoritmo Simplex e utiliza o Plano de Corte de Gomory para considerar somente as soluções inteiras do problema linear.

O algoritmo Plex consiste nos seguintes passos:

- 1. Montagem do tableau;
- 2. Obtenção de uma base inicial relaxando-se as restrições de integralidade;
- 3. Verificação da factibilidade da solução básica inicial obtida;
- 4. Obtenção de uma base factível, se a base obtida não o for;
- 5. Minimização da função objetivo;

- 6. Reconsiderando-se a integralidade das variáveis, faz-se a verificação da factibilidade da solução ótima obtida;
- 7. Obtenção de uma solução inteira através do Corte de Gomory se a solução obtida não o for.

Aplicando-se o algoritmo Plex ao problema de programação matemática formulado para o exemplo 2, tem-se como solução os implicantes primos 1XX0, 010X e X111 que corresponde a uma cobertura mínima, como pode ser verificado pelo Mapa de Karnaugh apresentado na figura 3.5.

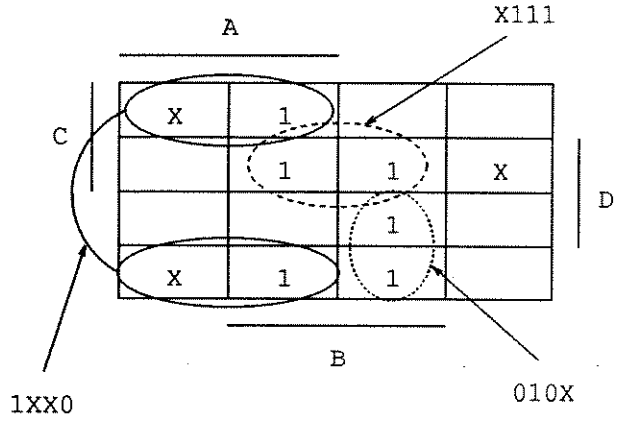


Figura 3.5: Mapa de Karnaugh apresentando a cobertura para a função apresentada na figura 5.2.

Dessa forma, a segunda fase do método de Quine-McCluskey pode ser interpretada como uma aplicação "intuitiva" de um algoritmo de programação linear inteira no problema de cobertura dos vértices da função.

É importante enfatizar três aspectos:

- Qualquer variante da interpretação tecnológica do custo que resulte em funções lineares das variáveis definidas, não altera a generalidade do método aqui proposto.
- O método conhecido por "Branching" [119], para tratar dos casos cíclicos, é um método de busca exaustiva e é apenas uma solução satisfatória dessa questão;
- Uma vez formulada como um problema de programação matemática, a questão da simplificação de funções booleanas se abre para a aplicação de todos os avanços nesta área.

Retomando-se o exemplo 2, pode-se ilustrar os passos envolvidos no processo de minimização utilizando o algoritmo Plex.

Reescrevendo o problema, em uma notação mais apropriada para a programação matemática, isto é, na forma matricial padrão, tem-se o tableau apresentado na figura 3.6.

A matriz foi arranjada de maneira que suas colunas estão ordenadas pelos custos relativos de cada implicante primo, onde o de menor custo está na primeira coluna (da esquerda).

Observe que todas as linhas possuem pelo menos dois coeficientes iguais a 1, indicando que se trataria de uma tabela cíclica para o método de Quine-McCluskey.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
0	3	4	4	4	4	4	4	0	0	0	0	0	0
1	0	0	0	0	0	1	1	-1	0	0	0	0	0
1	0	0	0	1	0	0	1	0	-1	0	0	0	0
1	0	0	1	1	1	0	0	0	0	-1	0	0	0
1	1	0	0	0	0	1	0	0	0	0	-1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	-1	0
1	0	1	1	0	0	0	0	0	0	0	0	0	-1

Figura 3.6: Tableau Plex inicial para o exemplo 2.

As figuras 3.7, 3.8, 3.9, 3.10, 3.11, 3.12 e 3.13 mostram as transformações ocorridas no tableau para se encontrar uma base inicial factível. O círculo no elemento 1, identifica o elemento pivô para a respectiva iteração.

Na figura 3.7 foi selecionado o elemento pivô para a primeira iteração.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
0	3	4	4	4	4	4	4	0	0	0	0	0	0
1	0	0	0	0	0	1	1	-1	0	0	0	0	0
1	0	0	0	1	0	0	1	0	-1	0	0	0	0
1	0	0	1	1	1	0	0	0	0	-1	0	0	0
1	(1)	0	0	0	0	1	0	0	0	0	-1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	-1	0
1	0	1	1	0	0	0	0	0	0	0	0	0	-1

Figura 3.7: Tableau apresentando o elemento pivô para a primeira iteração.

Pivoteando-se nesse elemento (eliminação de Gauss-Jordan), obtém-se o tableau apresentado na figura 3.8.

Na figura 3.8 é selecionado um outro elemento pivô para a segunda iteração. Esse elemento pivô deve estar numa linha que ainda não tem elemento pivô. O tableau após o pivoteamento é apresentado na figura 3.9.

Na figura 3.9 é selecionado o elemento pivô para a terceira iteração. O tableau após o pivoteamento é apresentado na figura 3.10.

Na figura 3.10 é selecionado o elemento pivô para a quarta iteração. O tableau após o pivoteamento é apresentado na figura 3.11.

Na figura 3.11 é selecionado o elemento pivô para a quinta iteração. O tableau após o pivoteamento é apresentado na figura 3.12

Na figura 3.12 é selecionado o elemento pivô para a sexta iteração. O tableau após o pivoteamento é apresentado na figura 3.13.

Observe que o tableau da figura 3.13 apresenta uma base inicial para o problema, porém, esta

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
-3	0	4	4	4	4	1	4	0	0	0	3	0	0
1	0	0	0	0	0	1	1	-1	0	0	0	0	0
1	0	0	0	1	0	0	1	0	-1	0	0	0	0
1	0	0	1	1	1	0	0	0	0	-1	0	0	0
1	1	0	0	0	0	1	0	0	0	0	-1	0	0
0	0	(1)	0	0	0	-1	0	0	0	0	1	-1	0
1	0	1	1	0	0	0	0	0	0	0	0	0	-1

Figura 3.8: Tableau apresentando o elemento pivô para a segunda iteração.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
-3	0	0	4	4	4	5	4	0	0	0	-1	4	0
1	0	0	0	0	0	1	1	-1	0	0	0	0	0
1	0	0	0	1	0	0	1	0	-1	0	0	0	0
1	0	0	(1)	1	1	0	0	0	0	-1	0	0	0
1	1	0	0	0	0	1	0	0	0	0	-1	0	0
0	0	1	0	0	0	-1	0	0	0	0	1	-1	0
1	0	0	1	0	0	1	0	0	0	0	-1	1	-1

Figura 3.9: Tableau apresentando o elemento pivô para a terceira iteração.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
-7	0	0	0	0	0	5	4	0	0	4	-1	4	0
1	0	0	0	0	0	1	1	-1	0	0	0	0	0
1	0	0	0	(1)	0	0	1	0	-1	0	0	0	0
1	0	0	1	1	1	0	0	0	0	-1	0	0	0
1	1	0	0	0	0	1	0	0	0	0	-1	0	0
0	0	1	0	0	0	-1	0	0	0	0	1	-1	0
0	0	0	0	-1	-1	1	0	0	0	1	-1	1	-1

Figura 3.10: Tableau apresentando o elemento pivô para a quarta iteração.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
-7	0	0	0	0	0	5	4	0	0	4	-1	4	0
1	0	0	0	0	0	1	1	-1	0	0	0	0	0
1	0	0	0	1	0	0	1	0	-1	0	0	0	0
0	0	0	1	0	1	0	-1	0	1	-1	0	0	0
1	1	0	0	0	0	1	0	0	0	0	-1	0	0
0	0	1	0	0	0	-1	0	0	0	0	1	-1	0
1	0	0	0	0	(-1)	1	1	0	-1	1	-1	1	-1

Figura 3.11: Tableau apresentando o elemento pivô para a quinta iteração.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
-7	0	0	0	0	0	5	4	0	0	4	-1	4	0
1	0	0	0	0	0	(1)	1	-1	0	0	0	0	0
1	0	0	0	1	0	0	1	0	-1	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	-1	1	-1
1	1	0	0	0	0	1	0	0	0	0	-1	0	0
0	0	1	0	0	0	-1	0	0	0	0	1	-1	0
-1	0	0	0	0	1	-1	-1	0	1	-1	1	-1	1

Figura 3.12: Tableau apresentando o elemento pivô para a sexta iteração.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
-12	0	0	0	0	0	0	-1	5	0	4	-1	4	0
1	0	0	0	0	0	(1)	1	-1	0	0	0	0	0
1	0	0	0	(1)	0	0	1	0	-1	0	0	0	0
0	0	0	(1)	0	0	0	-1	1	0	0	-1	1	-1
0	(1)	0	0	0	0	0	-1	1	0	0	-1	0	0
1	0	(1)	0	0	0	0	1	-1	0	0	1	-1	0
0	0	0	0	0	(1)	0	0	-1	1	-1	1	-1	1

Figura 3.13: Tableau Plex apresentando uma base inicial factível para o problema.

base não é ótima, pois há variáveis de custo cujo coeficiente é menor que zero. Essas variáveis são candidatas a entrar na base.

O Mapa de Karnaugh apresentado na figura 3.14 mostra que a solução inicial não é ótima.

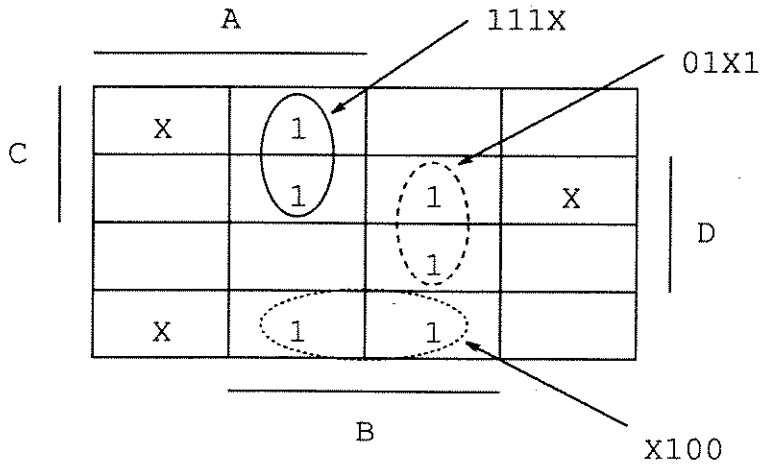


Figura 3.14: Mapa de Karnaugh apresentando uma cobertura não ótima para a função apresentada na figura 5.2.

É oportuno deixar explícito que a busca de uma solução factível mínima é um procedimento iterativo e que muitas soluções básicas factíveis são consideradas. Cada uma dessas soluções satisfazem o sistema original de equações.

Com mais dois passos do Plex, mostrados nas figuras 3.15 e 3.16, obtém-se a solução ótima do problema, cuja cobertura mínima é dada pelos implicantes primos 1XX0, X111 e 010X. A figura 3.5 apresenta esta cobertura no mapa de Karnaugh.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
-11	0	0	0	0	0	1	0	4	0	4	-1	4	0
1	0	0	0	0	0	1	1	-1	0	0	0	0	0
0	0	0	0	1	0	-1	0	1	-1	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	-1	1	-1
1	1	0	0	0	0	1	0	0	0	0	-1	0	0
0	0	1	0	0	0	-1	0	0	0	0	1	-1	0
0	0	0	0	0	1	0	0	-1	1	-1	1	-1	1

Figura 3.15: Primeira iteração na busca da solução ótima.

Note que o resultado obtido pelo Plex é o mesmo resultado obtido pela aplicação da segunda fase do algoritmo de Quine-McCluskey.

O programa Espresso gerou os seguintes implicantes primos na cobertura irredundante: X100, 01X1 e 111X, que corresponde a um custo igual a 12, enquanto que o Plex gerou uma cobertura com um custo igual a 11.

	1XX0	111X	X101	01X1	010X	X100	0X11	s1	s2	s3	s4	s5	s6
-11	0	1	0	0	0	0	0	4	0	4	0	3	0
1	0	0	0	0	0	1	1	-1	0	0	0	0	0
0	0	0	0	1	0	-1	0	1	-1	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	-1
1	1	1	0	0	0	0	0	0	0	0	0	-1	0
0	0	1	0	0	0	-1	0	0	0	0	1	-1	0
0	0	-1	0	0	1	1	0	-1	1	-1	0	0	1

Figura 3.16: Segunda iteração na busca da solução ótima.

3.3 Estudos de casos

Uma série exaustiva de testes mostraram que a cobertura exata de funções booleanas, cuja tabela de implicantes é cíclica, pode ser obtida formulando-se o problema como um problema de programação linear inteira 0 e 1.

Todos os resultados obtidos pelo Plex foram comparados com os resultados obtidos pelo algoritmo de Quine-McCluskey e pelo programa Espresso. Excluídos os casos onde a cobertura mínima se faz apenas com os implicantes primos essenciais, aproximadamente 60% dos casos tiveram a base inicial factível não ótima, 40% tiveram base inicial factível e 10% tiveram a base inicial infactível (utiliza-se a primeira fase do algoritmo).

Em todos os casos, o Plex obteve a solução mínima, o mesmo não ocorrendo com o programa Espresso.

Estuda-se a seguir, alguns casos que evidenciam as características do algoritmo proposto.

3.3.1 Caso cuja base inicial não é ótima

Exemplo 3. Seja a função de 4 variáveis f(A,B,C,D) mostrada no mapa de Karnaugh apresentado na figura 3.17.

A aplicação do algoritmo GeraPlex resulta no seguinte problema de programação matemática:

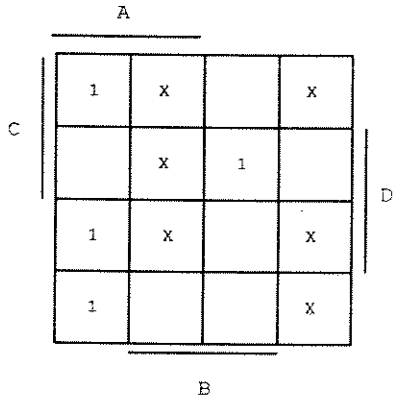


Figura 3.17: Mapa de Karnaugh para uma função de 4 variáveis.

Min 3.(a + b) + 4.(c + d + e)

$a + d \geq 1$
$a + b \geq 1$
$b + c \geq 1$
$e \geq 1$
a, b, c, d, e $\in$ { 0,1 }.

onde: a = X0X0, b = X00X, c = 1X01, d = 1X10, e = X111

E = 1 e assim tem-se o tableau Plex inicial apresentado na figura 3.18.

	X0X0	X00X	1X01	1X10	s1	s2	s3
0	3	3	4	4	0	0	0
1	1	1	0	0	-1	0	0
1	0	1	1	0	0	-1	0
1	1	0	0	1	0	0	-1

Figura 3.18: Tableau Plex inicial para o exemplo

Pivoteando convenientemente, tem-se o tableau mostrado na figura 3.19.
Portanto X0X0, X00X e 1X10 é uma base factível para o Plex.
A solução inicial: X0X0 = 1X01 = 0 e X00X = 1X10 = 1 com custo 7 não é ótima, pois o custo relativo da primeira variável de folga é negativo.
Com mais um passo do Plex, tem-se a solução ótima cujo tableau é apresentado na figura 3.20.
f(A,B,C,D) = X0X0 + X00X + X111 = $\overline{B}.\overline{D} + \overline{B}.\overline{C} + B.C.D.$

É importante realçar que o fato das variáveis serem inteiras não foi usado explicitamente. Contudo, devido às características próprias desta classe de problemas, as soluções quase sempre

	X0X0	X00X	1X01	1X10	s1	s2	s3
-7	0	0	0	0	-1	4	4
0	1	0	-1	0	-1	1	0
1	0	1	1	0	0	-1	0
1	0	0	1	1	1	-1	-1

Figura 3.19: Tableau apresentando uma base inicial factível.

	X0X0	X00X	1X01	1X10	s1	s2	s3
-6	0	0	1	1	0	3	3
1	1	0	0	1	0	0	-1
1	0	1	1	0	0	-1	0
1	0	0	1	1	1	-1	-1

Figura 3.20: Tableau Plex ótimo.

foram 0 ou 1.

Na resolução de um problema de programação linear, nem sempre a solução ótima ocorre em pontos cujas coordenadas são números inteiros, condição indispensável para o problema de minimização de funções booleanas.

3.3.2 Caso em que é necessário utilizar o Plano de Corte de Gomory

Exemplo 4. Seja a função de 4 variáveis f(A,B,C,D) mostrada no mapa de Karnaugh da figura 3.21.

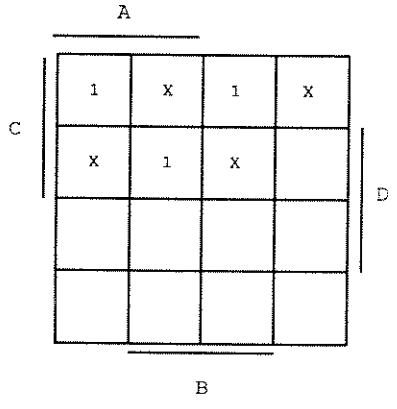


Figura 3.21: Mapa de Karnaugh para f(ABCD)

Aplicando-se o método GeraPlex, obtém-se os seguintes implicantes primos: X11X, XX10 e 1X1X, todos com o custo = 3.

O problema de cobertura pode ser formulado como:

MIN 3.X11X + 3.XX10 + 3.1X1X

$$X11X + XX10 \geq 1$$
$$XX10 + 1X1X \geq 1$$
$$X11X + 1X1X \geq 1$$
$$X11X, XX10, 1X1X \in \{ 0,1 \}.$$

A figura 3.22 apresenta o tableau Plex inicial.

b	X11X	XX10	1X1X	s1	s2	s3
0	3	3	3	0	0	0
1	1	1	0	-1	0	0
1	0	1	1	0	-1	0
1	1	0	1	0	0	-1

Figura 3.22: Tableau IntPlex inicial.

Após pivotear convenientemente, chega-se numa base inicial para o problema, como pode ser visto pela figura 3.23. Observe que essa não é uma base inicial ótima, pois há coeficiente de custo negativo.

b	X11X	XX10	1X1X	s1	s2	s3
-6	0	0	-3	0	3	3
1	1	0	1	0	0	-1
1	0	1	1	0	-1	0
1	0	0	2	1	-1	-1

Figura 3.23: Tableau Plex apresentando uma base inicial não ótima.

A figura 3.24 apresenta o tableau Plex após algumas iterações, na busca de uma solução otima.

Observe na figura 3.24 que a base está sendo representada pelo dígito 2. Isto ocorreu pois o algoritmo Plex utiliza-se de combinações lineares das linhas do tableau para preservar os valores inteiros dos seus coeficientes. Esse tableau é semelhante ao tableau apresentado na figura 3.25.

A solução apresentada na figura 3.25 é uma solução otimista, pois os coeficientes do vetor b são números não inteiros. Para obter a integralidade da solução, insere-se uma restrição de corte e aplica-se o dual Plex. Este procedimento é repetido até que se encontre a integralidade da solução.

A figura 3.26 apresenta o tableau Plex após inserir-se a restrição de corte.

Após um passo do dual Plex, obtém-se o tableau apresentado na figura 3.27.

b	X11X	XX10	1X1X	s1	s2	s3
-3	0	0	0	1	1	1
1	2	0	0	-1	1	-1
1	0	2	0	-1	-1	1
1	0	0	2	1	-1	-1

Figura 3.24: Tableau IntPlex apresentando uma solução otimista.

b	X11X	XX10	1X1X	s1	s2	s3
-9/2	0	0	0	3/2	3/2	3/2
1/2	1	0	0	-1/2	1/2	-1/2
1/2	0	1	0	-1/2	-1/2	1/2
1/2	0	0	1	1/2	-1/2	-1/2

Figura 3.25: Tableau IntPlex apresentando uma solução otimista.

b	X11X	XX10	1X1X	s1	s2	s3	
-3	0	0	0	1	1	1	0
1	2	0	0	-1	1	-1	0
1	0	2	0	-1	-1	1	0
1	0	0	2	1	-1	-1	0
-1	0	0	0	-1	-1	-1	2

Restrição de corte

Figura 3.26: Tableau Plex após inserir a restrição de corte.

b	X11X	XX10	1X1X	s1	s2	s3	
-3	0	0	0	0	0	0	1
1	1	0	0	0	1	0	-1
1	0	1	0	0	0	1	-1
0	0	0	1	0	-1	-1	1
1	0	0	0	1	1	1	-2

Figura 3.27: Tableau Plex na condição de otimalidade.

Nota-se no tableau da figura 3.27 que todos os custos relativos, ou são nulos, ou positivos, indicando uma condição de otimalidade e que agora os coeficientes b são todos positivos ou nulos. A solução ótima é dada por XX10 + 1X1X.

3.3.3 Caso em que a base inicial é infactível.

Exemplo 5: Seja a função booleana f(ABCDE) representada no mapa de Karnaugh apresentado na figura 3.28.

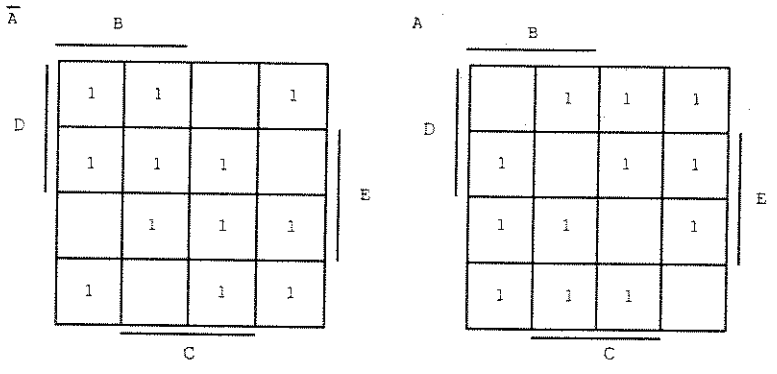


Figura 3.28: Mapa de Karnaugh para uma função de 5 variáveis.

O algoritmo GeraPlex produz os seguintes implicantes primos:
a=11X0X, b=1X1X0, c=10X1X, d=1X0X1, e=01X1X, f=0X1X1, g=0X0X0,
h=00X0X, i=X1110, j=X1101, k=X1011, l=X0111, m=X1000, n=X0100,
o=X0010, p=X0001
que resulta no seguinte problema linear:

$$\text{Min } 4.(a+b+c+d+e+f+g+h)+5.(i+j+k+l+m+n+o+p)$$

$$\begin{array}{lll} a+b\geq 1 & c+l\geq 1 & f+j\geq 1 \\ a+d\geq 1 & c+o\geq 1 & f+l\geq 1 \\ a+j\geq 1 & d+k\geq 1 & f+h\geq 1 \\ a+m\geq 1 & d+p\geq 1 & g+h\geq 1 \end{array}$$

$$
\begin{array}{lll}
b+n \geq 1 & e+f \geq 1 & g+o \geq 1 \\
b+i \geq 1 & e+i \geq 1 & g+m \geq 1 \\
b+c \geq 1 & e+k \geq 1 & h+n \geq 1 \\
c+d \geq 1 & e+g \geq 1 & h+p \geq 1
\end{array}
$$

Utilizando-se o algoritmo Plex, obtém-se a cobertura mínima composta dos seguintes implicantes primos : 1X1X0, 11X0X, 1X0X1, 10X1X, 0X1X1, 01X1X, 0X0X0 e 00X0X, que corresponde a um custo igual a 32.

É importante frisar que, neste caso, a base inicial encontrada não era factível, e que através da introdução de uma variável artificial e da resolução do problema linear modificado, encontrou-se uma base factível e, com posterior uso da minimização, encontrou-se a solução ótima.

Utilizando-se o programa Espresso, obtém-se a cobertura irredundante composta dos seguintes implicantes:X0100, X1110, X0001, X1011, 0X0X0, 0X1X1, 11X0X e 10X1X, que corresponde a um custo igual a 36.

3.4 Conclusão

Neste capítulo foi discutido a utilização da programação matemática no problema de cobertura de funções booleanas. Uma vez formulada, como um problema de programação matemática, a questão da simplificação de funções booleanas se abre para a aplicação de todos os avanços nesta área. Qualquer variante da interpretação tecnológica do custo, que resulte em funções lineares das variáveis definidas, não altera a generalidade do método aqui proposto.

O algoritmo de Quine-McCluskey foi analisado sob a óptica da programação matemática, para evidenciar que pode ser visto como uma solução "intuitiva" do problema.

Foi apresentado um algoritmo, intitulado Plex, que obtém a cobertura mínima de uma função booleana, resolvendo-se um problema de programação linear 0 e 1, formulado a partir de todos os implicantes primos gerados pela aplicação da operação do consenso aos caminhos de uma árvore binária, que representam os mintermos e irrelevantes de uma função booleana.

O algoritmo Plex é uma modificação do algoritmo Simplex. No lugar do tradicional pivoteamento, foram utilizadas combinações lineares das linhas do tableau para preservar os valores inteiros dos seus coeficientes. Encontrada uma solução otimista (solução ótima, porém infactível). utilizou-se o Plano de Corte de Gomory, para preservar somente os valores inteiros de seus coeficientes.

Uma série exaustiva de testes mostrou a viabilidade desse tipo de enfoque no problema de cobertura. É interessante registrar que em todos esses testes o Plex teve tempo de execução menor que o do programa Espresso e em todos os casos obteve a cobertura mínima, o mesmo não ocorrendo para o Espresso.

Capítulo 4

Minimização de Funções Booleanas com Múltiplas Saídas

Muitas vezes, em projetos digitais, o circuito tem várias saídas ao invés de apenas uma. A implementação de várias funções booleanas com as mesmas variáveis poderia ser feita independentemente, pelas técnicas descritas nos capítulos anteriores, mas uma considerável economia ocorrerá se essas funções forem implementadas como um todo. Essa economia decorre do uso comum por duas ou mais funções de um termo produto, que é implementado uma única vez e compartilhado pelas funções.

Neste capítulo, apresenta-se o algoritmo MultPlex, que minimiza funções booleanas com múltiplas saídas. Para a fase de geração dos implicantes primos, utilizou-se um método tabular, estendido para considerar funções com múltiplas saídas [61], [62] e [63].

As coberturas das funções são resultados de dois problemas de programação matemática, aqui chamados de problema de cobertura múltipla e problema de cobertura singular.

Os testes realizados mostraram que este tipo de enfoque resolve o problema de minimização de funções de múltiplas saídas de maneira eficiente, e representa um avanço considerável para esta classe de problemas.

4.1 Introdução

O problema de simplificar funções booleanas com um grande número de variáveis e múltiplas saídas, onde as saídas dependem das mesmas variáveis, estava sem uma solução satisfatória [61], [62], [68].

Seguramente as funções para as múltiplas saídas podem ser minimizadas separadamente, mas esse procedimento não leva a uma solução de menor custo.

Ambos, o Mapa de Karnaugh e o método de Quine-McCluskey, podem ser estendidos para simplificar várias funções simultaneamente, através do compartilhamento dos termos comuns. Contudo, isto torna-se impraticável se o número de variáveis ou o número de funções forem grandes. Pior ainda se ambas forem grandes.

Uma variedade de algoritmos, por exemplo, [57], [60], [62], [67] e [70] foram desenvolvidos na tentativa de solucionar esse problema.

Um desses algoritmos, o McBoole [64], acha uma cobertura mínima de uma função booleana com múltiplas saídas expressa como uma lista de cubos. Baseando-se na manipulação eficiente de grafos e em técnicas de particionamento, o McBoole encontra quase sempre uma cobertura mínima global para uma função com múltiplas saídas, tornando-o muito atrativo para funções com até 20 variáveis de entrada e até 20 funções de saídas.

O critério de otimização é um fator de projeto que deve estar bem definido. Em outras palavras, o que o projetista deseja otimizar? É o número de gates, a quantidade de chips, o número de interconexões, os tempos de atrasos, o custo, ou talvez uma combinação de tudo isso?

Desse modo, a necessidade de otimização é altamente dependente da tecnologia a ser empregada e do volume de produção.

Este capítulo formula o problema de cobertura de funções booleanas com múltiplas saídas e apresenta o algoritmo MultPlex, que obtém a cobertura mínima, resolvendo-se problemas de programação linear inteira 0 e 1. O critério adotado é o de minimizar o número de portas ANDs, na realização da função. Neste caso, a cobertura formulada como um problema de programação matemática, pode ser escrita como um problema de programação linear inteira 0 e 1.

Para a geração dos implicantes primos, utilizou-se um método tabular, adaptado para considerar funções com múltiplas saídas.

As coberturas das funções de saídas são soluções de dois problemas de programação matemática.

No primeiro problema, chamado problema de cobertura múltipla, formula-se o programa matemático cujas variáveis são os implicantes primos singulares e os implicantes primos múltiplos das funções de saídas. Gera-se uma restrição de cobertura para cada mintermo de todas as funções de saídas.

No segundo problema, chamado problema de cobertura singular, as variáveis são as variáveis solução do primeiro problema. Desse modo, têm-se tantos problemas singulares quantos forem o número de saída. Na função objetivo de cada problema, consideram-se somente os implicantes primos solução da cobertura múltipla, que cobrem a função em questão. Nas restrições de cobertura, consideram-se somente os mintermos da função.

O projeto do codificador de linha HDB3 [75], mostra a vantagem de se considerar funções booleanas com múltiplas saídas em projetos de circuitos digitais.

4.2 Conceitos e definições

Define-se funções booleanas com múltiplas saídas como sendo um conjunto de m funções que dependem das mesmas n variáveis.

Por exemplo, as funções $f_1(A,B,C) = \sum(0,2,6)$ e $f_2(A,B,C) = \sum(2,6,7)$, mostradas na figura 4.1.

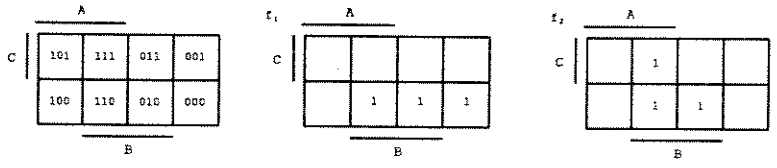


Figura 4.1: Tabela verdade para as funções f_1 e f_2.

Uma forma direta de simplificar funções booleanas com múltiplas saídas, é achar a cobertura mínima para cada função independentemente, entretanto, este procedimento não leva ao circuito mais econômico.

Exemplo 1. Sejam as funções f_1 e f_2 apresentadas na figura 4.1.

Minimizando-se as funções f_1 e f_2 separadamente, ou seja, através do método apresentado no capítulo 3, tem-se as coberturas mostradas na figura 4.2, cujos circuitos são apresentados na figura 4.3, com custo de implementação igual a $12 = (2 + 2 + 2) + (2 + 2 + 2)$.

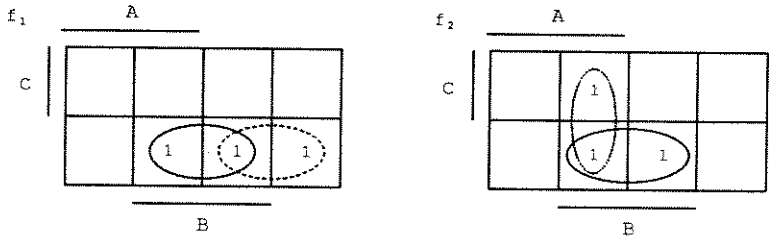


Figura 4.2: Cobertura para as funções f_1 e f_2 simplificadas individualmente.

Observe na figura 4.3, que a porta lógica que implementa o implicante $B.\bar{C}$ aparece em ambas as funções.

Minimizando-se as funções f_1 e f_2 como uma função de múltiplas saídas, tem-se o circuito mostrado na figura 4.4.

Observe que o termo comum às duas funções foi implementado uma única vez e compartilhado por ambas as funções, com custo total igual a 10.

Esse exemplo deixa claro a vantagem de se identificar os termos comuns e garantir a realização de custo mínimo da função.

Há casos, entretanto, não tão triviais, em que o termo comum às funções não está tão evidente: geralmente esses termos não são implicantes primos de nenhuma das funções. O exemplo 2 ilustra esta situação.

Exemplo 2. Sejam as funções booleanas $f_3(A,B,C) = \sum(0,1,2)$ e $f_4(A,B,C) = \sum(2,6,7)$, cujos Mapas de Karnaugh estão apresentados na figura 4.5.

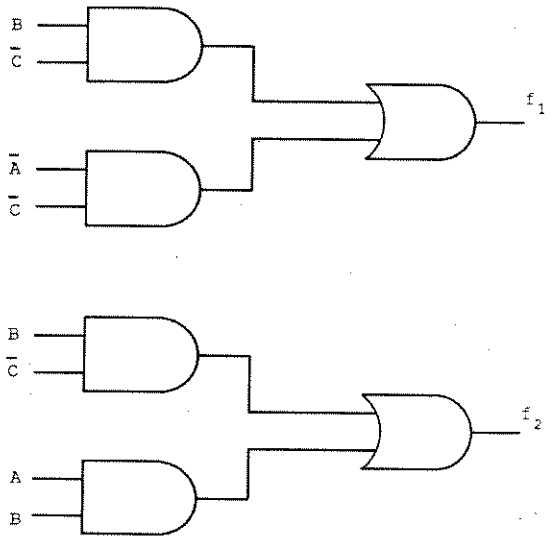


Figura 4.3: Circuitos lógicos das funções apresentadas na figura 6.2.

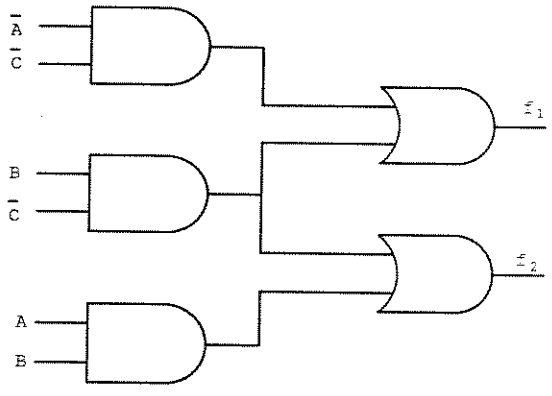


Figura 4.4: Circuitos das funções f_1 e f_2 minimizadas como uma função de múltiplas saídas.

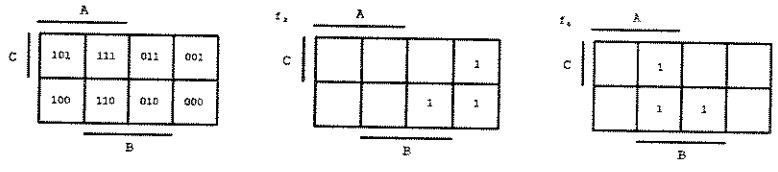


Figura 4.5: Mapas de Karnaugh para as funções f_3 e f_4.

Minimizando-se as funções f_3 e f_4 individualmente, têm-se $f_3 = \bar{A}.\bar{B} + \bar{A}.\bar{C}$ e $f_4 = A.B + B.\bar{C}$, cujas coberturas estão apresentadas na figura 4.6 e os circuitos correpondentes, na figura 4.7. O custo total é igual a 12.

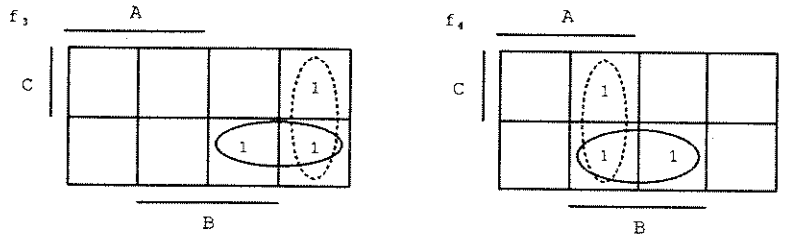


Figura 4.6: Coberturas para as funções f_3 e f_4 minimizadas individualmente.

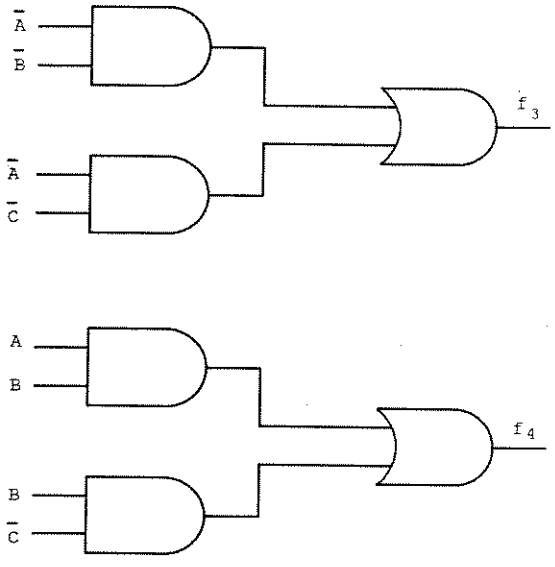


Figura 4.7: Circuitos que implementam as coberturas apresentadas na figura 6.6.

Minimizando-se as mesmas funções como múltiplas saídas, tem-se o circuito apresentado na figura 4.8 e as coberturas na figura 4.9. O custo para implementar este circuito é igual a 11.

O que se evidencia, no exemplo 2, é que o termo produto $\bar{A}.B.\bar{C}$ é um implicante primo do produto lógico $f_3.f_4$, mas não é um implicante primo nem de f_3 e nem de f_4 isoladamente, como pode ser verificado pelo Mapa de Karnaugh apresentado na figura 4.9.

Dessa forma, mostrou-se a vantagem de se gerar, além dos implicantes primos específicos de cada função, os implicantes comuns a duas ou mais funções.

Define-se implicante primo singular, como sendo o que cobre mintermos, pertencentes apenas a cada uma das funções de saída.

Define-se implicante primo múltiplo, como sendo o que cobre mintermos de duas ou mais funções de saída.

Para a minimização de funções booleanas com múltiplas saídas, os implicantes candidatos a pertencerem a cobertura mínima, correspondem a todos os implicantes de f_1, f_2, f_3, $f_1.f_2$, $f_1.f_3$, $f_2.f_3$, $f_1.f_2.f_3$, etc.

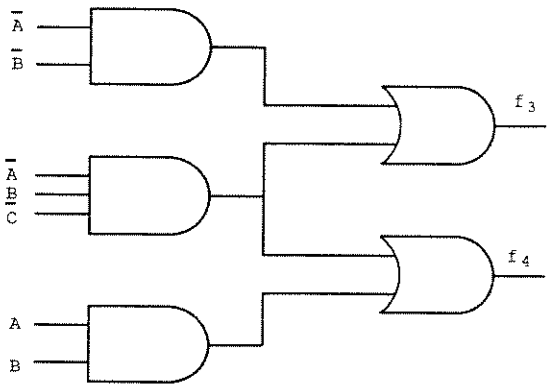


Figura 4.8: Circuito que implementa as coberturas apresentadas na figura 6.9.

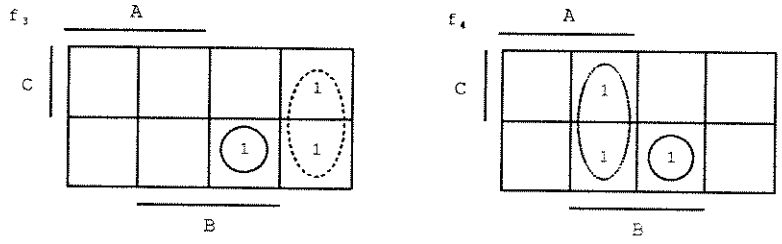


Figura 4.9: Coberturas para as funções f_3 e f_4 minimizadas como função de múltiplas saídas.

Descreve-se na próxima seção, um método tabular para a geração de implicantes primos de funções com múltiplas saídas.

4.3 Geração de implicantes primos para funções com múltiplas saídas

Para a cobertura de funções de múltiplas saídas, considera-se, além dos implicantes primos singulares, os implicantes primos múltiplos.

O método tabular de Quine-McCluskey, para a minimização de funções booleanas com saída simples, pode ser estendido para considerar funções com múltiplas saídas.

No caso de funções com múltiplas saídas, os mintermos e os irrelevantes devem conter, além do identificador (representação binária de um mintermo ou irrelevante), uma outra seqüência de bits, chamada de rótulo, que assinala quais funções de saída incluem o produto, especificado pelo identificador. Desse modo, a notação 0100 11 significa que o mintermo 4 (0100) pertence às funções f_1 e f_2 (11), e, 11X0 01 significa que o implicante 11X0 cobre somente mintermos pertencentes a função f_2 (01).

Para a geração dos implicantes primos singulares e implicantes primos múltiplos, os mintermos e os irrelevantes são combinados do mesmo modo que no método de Quine-McCluskey, porém:

- Só podem ser combinados mintermos ou irrelevantes comuns (assinalados para as mesmas funções de saída);

- Quando dois mintermos ou irrelevantes combinarem-se, os bits do rótulo do implicante re-

sultante terá o dígito 1, somente nas posições onde ambos os rótulos dos mintermos ou irrelevantes combinados têm 1, ou seja, aplica-se o operador AND entre os bits dos rótulos.

Por exemplo, a combinação dos mintermos 0110 10 e 1110 11, gera o implicante X110 10. que pertence somente a f_1.

- Marcar um implicante como tendo sido combinado, somente se seu rótulo está contido no rótulo do implicante resultante da combinação.

Exemplo 3. Sejam as funções de 3 variáveis f(A,B,C), mostradas nos mapas de Karnaugh, da figura 4.10.

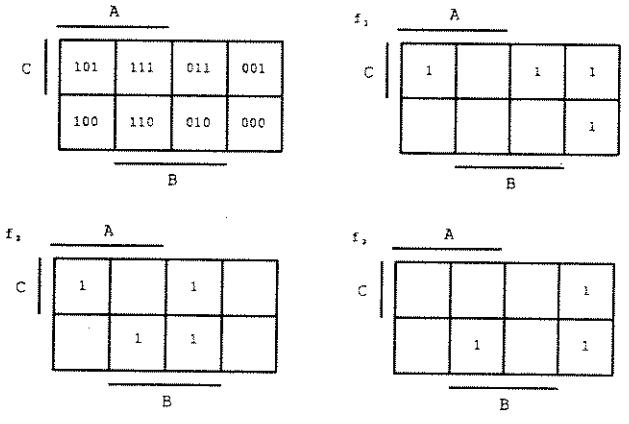


Figura 4.10: Mapas de Karnaugh para funções de 3 variáveis.

A figura 4.11 apresenta a tabela de geração dos implicantes primos para as funções apresentadas na figura 4.10, considerando-as como uma função de múltiplas saídas. Pode-se notar que os implicantes primos gerados são:

011 110 101 110 110 011 00X 101 0X1 100 X01 100 01X 010 X10 010

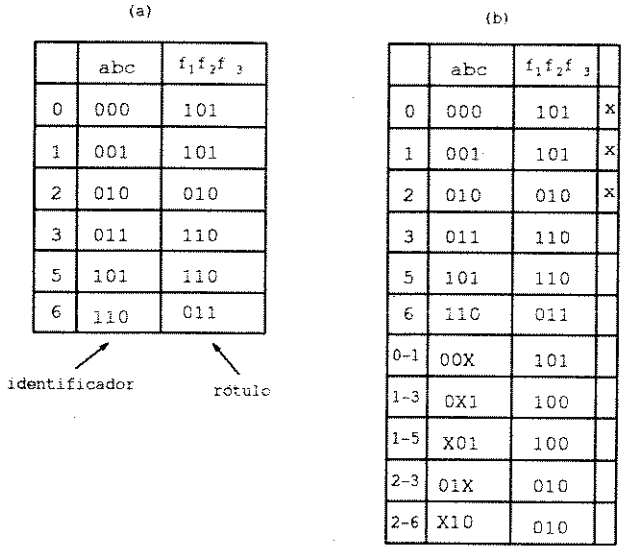
O procedimento para selecionar os implicantes primos que formam a soma mínima poderia ser feito por intermédio de uma tabela de implicantes primos, que é similar à técnica utilizada para funções de saída simples.

Entretanto, este método que já é bastante complexo para saída simples, torna-se extremamente penoso e ineficiente para o caso de múltiplas saídas.

Na seção seguinte, apresenta-se o problema de cobertura de funções booleanas com múltiplas saídas, visto sob a óptica da programação matemática.

4.4 Cobertura de funções com múltiplas saídas sob a óptica da programação matemática

O problema de cobertura de funções booleanas com múltiplas saídas, pode ser formulado como um problema de programação matemática.



	abc	$f_1f_2f_3$
0	000	101
1	001	101
2	010	010
3	011	110
5	101	110
6	110	011

	abc	$f_1f_2f_3$	
0	000	101	x
1	001	101	x
2	010	010	x
3	011	110	
5	101	110	
6	110	011	
0-1	00X	101	
1-3	0X1	100	
1-5	X01	100	
2-3	01X	010	
2-6	X10	010	

Figura 4.11: Tabela de geração dos implicantes primos para as funções apresentadas na figura 6.10.

4.4.1 Minimização do número de portas ANDs

Para este critério de otimização, a cobertura das funções de saída, formulada como um problema de programação matemática, é similar com o da formulação apresentada no capítulo 3. A diferença é que, no caso de funções com múltiplas saídas, a cobertura é efetuada em duas etapas.

Na primeira etapa, chamada de cobertura múltipla, as variáveis do problema de programação matemática, são os implicantes primos singulares e os implicantes primos múltipos, obtidos pelo método tabular.

As restrições de cobertura são tantas quanto forem os mintermos de todas as funções de saída. Se um mintermo está incluído em duas funções, têm-se duas restrições para esse mintermo.

A solução da cobertura múltipla é obtida utilizando-se o algoritmo Plex.

Obtida a solução da cobertura múltipla, inicia-se a segunda etapa da cobertura.

Na segunda etapa, chamada cobertura singular, tem-se um problema de programação matemática para cada função de saída, onde as variáveis para cada cobertura, são os implicantes primos, solução da primeira etapa, que cobrem a função em questão.

Desse modo, o problema matemático para a cobertura singular é exatamente igual ao descrito no capítulo 3.

Retomando-se o exemplo 3, tem-se como cobertura múltipla, a solução do seguinte problema de programação matemática.

$$\text{MIN } 3.(00\text{X} + 0\text{X1} + \text{X01} + 01\text{X} + \text{X10}) + 4.(011 + 101 + 110)$$

$$\left.\begin{array}{l} 00X \geq 1 \\ 00X + 0X1 + X01 \geq 1 \\ 011 + 0X1 \geq 1 \\ 101 + X01 \geq 1 \end{array}\right\} \Rightarrow f_1$$

$$\left.\begin{array}{l} 01X + X10 \geq 1 \\ 011 + 01X \geq 1 \\ 101 \geq 1 \\ 110 + X10 \geq 1 \end{array}\right\} \Rightarrow f_2$$

$$\left.\begin{array}{l} 00X \geq 1 \\ 00X \geq 1 \\ 110 \geq 1 \end{array}\right\} \Rightarrow f_3$$

$$00\text{X}, 0\text{X1}, \text{X01}, 01\text{X}, \text{X10}, 011, 101, 110 \in \{ 0, 1 \}$$

Note que o critério de custo é dado pela soma ponderada dos implicantes primos múltiplos e dos implicantes primos singulares, e que há uma restrição para cada mintermo de todas as funções de saída. Deste modo, as quatro primeiras restrições referem-se, respectivamente, à cobertura dos mintermos 0, 1, 3 e 5 de f_1.

As restrições redundantes podem ser eliminadas.

A solução deste problema é dada pelos implicantes primos 101 110, 110 011, 00X 101, 0X1 100 e 01X 010.

As coberturas singulares para essas funções de saídas são de soluções triviais.

A função f_1 é coberta pelos implicantes primos 101, 00X e 0X1, cujo custo total desses ANDs é igual a 7.

A função f_2 é coberta pelos implicantes primos 101, 110 e 01X, cujo custo é igual a 8.

A função f_3 é coberta pelos implicantes primos 110 e 00X, cujo custo é igual a 5.

O custo total dos ANDs, para realizar as funções f_1, f_2 e f_3, como uma função de múltiplas saídas, é 12, pois os mintermos 101 e 110 e o implicante 00X aparecem duas vezes nas funções.

O custo do circuito, ANDs e ORs, é igual a $12 + 3 + 3 + 2 = 20$.

As mesmas três funções, minimizadas como funções de saída simples, são cobertas pelos seguintes implicantes primos:

$f_1 = 00\text{X} + 0\text{X1} + \text{X01}$, cujo custo AND é igual a 6;
$f_2 = 101 + 01\text{X} + \text{X10}$, cujo custo AND é igual a 7;
$f_3 = 110 + 00\text{X}$, cujo custo AND é igual a 5.

O custo total do circuito, para a realização das três funções é $18 + 3 + 3 + 2 = 26$.

O algoritmo MultPlex, que obtém uma cobertura de custo mínimo para uma função booleana de múltiplas saídas, consiste nos seguintes passos:

- 1. Geração dos implicantes primos pelo método tabular;
- 2. Montagem do Tableau;

- 3. Obtenção de uma base inicial, relaxando-se as restrições de integralidade;
- 4. Verificação da factibilidade da solução básica inicial obtida;
- 5. Obtenção de uma base inicial factível, se a base obtida não o for;
- 6. Minimização da função objetivo;
- 7. Reconsiderando-se a integralidade das variáveis, faz-se a verificação da factibilidade da solução ótima obtida;
- 8. Obtenção de uma solução inteira, através de Planos de Corte de Gomory, se a solução obtida não o for;
- 9. Geração da cobertura singular, para cada uma das funções de saída;
- 10. Execução dos passos 2, 3, 4, 5, 6, 7 e 8 para cada uma das formulações de cobertura singular, geradas no passo anterior.

Exemplo 4. Sejam as funções de 4 variáveis f(A,B,C,D), mostradas nos mapas de Karnaugh da figura 4.12.

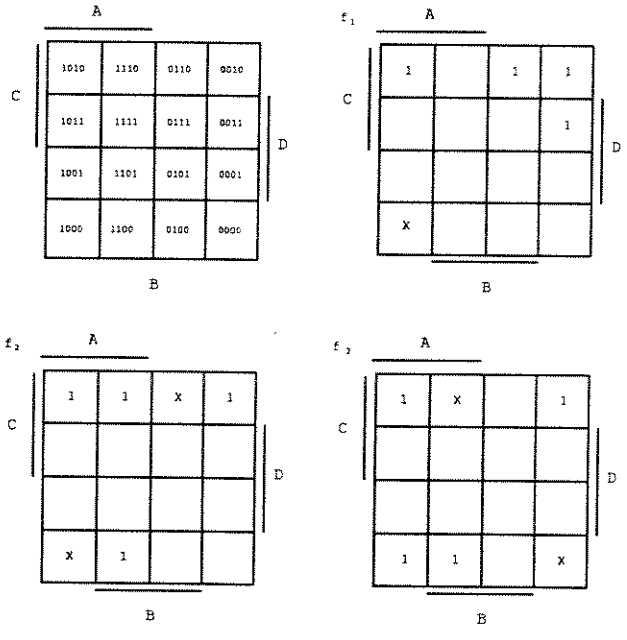


Figura 4.12: Mapas de Karnaugh para funções de 4 variáveis.

A primeira fase do método, gera os seguintes implicantes primos:

0X10 110, 001X 100, X010 111, 10X0 111, X0X0 001, XX10 010 e 1XX0 011

A formulação para a cobertura múltipla é dada por:

MIN 3.(X0X0 + XX10 + 1XX0) + 4.(0X10 + 001X + X010 + 10X0)

$$\left.\begin{array}{l} 0X10 + 001X + X010 \geq 1 \\ 001X \geq 1 \\ 0X10 \geq 1 \\ X010 + 10X0 \geq 1 \end{array}\right\} \Rightarrow f_1$$

$$\left.\begin{array}{l} 0X10 + X010 + XX10 \geq 1 \\ X010 + 10X0 + XX10 + 1XX0 \geq 1 \\ 1XX0 \geq 1 \\ XX10 + 1XX0 \geq 1 \end{array}\right\} \Rightarrow f_2$$

$$\left.\begin{array}{l} X010 + X0X0 \geq 1 \\ 10X0 + X0X0 + 1XX0 \geq 1 \\ X010 + 10X0 + X0X0 \geq 1 \\ 1XX0 \geq 1 \end{array}\right\} \Rightarrow f_3$$

X0X0, XX10, 1XX0, 0X10, 001X, X010 e 10X0 $\in$ { 0, 1 }

A solução ótima, para este problema linear inteiro, é dada por:

0X10 110 001X 100 1XX0 011 X010 111

cujas coberturas das funções são mostradas na figura 4.13.

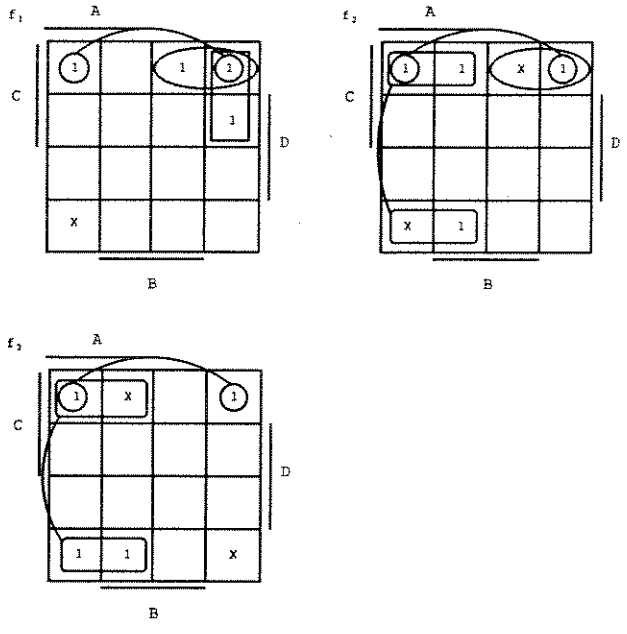


Figura 4.13: Coberturas para as funções f_1, f_2 e f_3, apresentadas na figura 6.12.

Pode-se notar, pelo mapa de Karnaugh da função f_2, que o implicante primo 0X10 é redundante, pois só cobre mintermo já coberto por outro implicante primo. Portanto, após obter a solução para a cobertura múltipla, deve-se fazer uma cobertura singular, uma para cada função de saída, cujas variáveis são os implicantes primos, obtidos como solução da cobertura múltipla.

A cobertura singular para f_1 é dada por:

MIN 4.(0X10 + 001X + X010)

$0X10 + 001X + X010 \geq 1$
$001X \geq 1$
$0X10 \geq 1$
$X010 + 10X0 \geq 1$
0X10, 001X e X010 $\in$ { 0, 1 }

A solução para este problema é dada por:
001X, 0X10 e X010.
A cobertura singular para f_2 é dada por:

MIN 3.1XX0 + 4.(0X10 + X010)

$0X10 + X010 \geq 1$
$1XX0 + X010 \geq 1$
$1XX0 \geq 1$
$1XX0 \geq 1$
1XX0, 0X10 e X010 $\in$ { 0, 1 }

A solução para este problema é dada por:
1XX0 e X010.
Note que o implicante primo 0X10, pertencente a solução da cobertura múltipla, foi eliminado da cobertura da figura f_2, pois esse implicante é essencial somente para a cobertura da função f_1.
A cobertura singular para f_3 é dada por:

MIN 3.1XX0 + 4.X010

$X010 \geq 1$
$1XX0 \geq 1$
$1XX0 + X010 \geq 1$
$1XX0 \geq 1$
11XX0 e X010 $\in$ { 0, 1 }

A solução para este problema é dada por:
1XX0 e X010.
O custo para a realização dos ANDs das 3 funções, como uma função de múltiplas saídas, corresponde a 11, enquanto que o custo dos ANDs para a realização das 3 funções, como função de saída simples, corresponde a 17.

4.5 Estudo de caso - Codificador de linha HDB3

O código de linha HDB3 tem sido utilizado por muitos anos e é recomendado pela CCITT para sistemas de transmissão digital, principalmente de 2048 Kbits por segundos [75].

A figura 4.14 apresenta o diagrama de estados do HDB3. No diagrama de estados, cada estado é origem de duas setas de transição, uma para a entrada "1" e outra para a entrada "0".

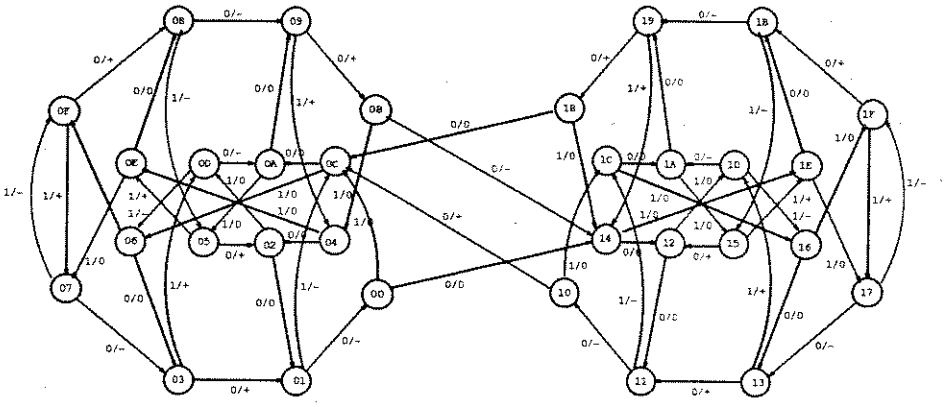


Figura 4.14: Diagrama de estados para o código de linha HDB3.

O HDB3 procura corrigir o efeito das longas cadeias de "0" na saída do codificador, substituindo quatro zeros consecutivos pelas seqüências B00V ou 000V, onde B indica um pulso de acordo com a regra bipolar e V indica um pulso que viola a regra bipolar.

Quando a máquina de estados que representa o codificador HDB3 é implementada, utilizando-se como elemento de memória, flip-flops tipo D, têm-se as seguintes funções combinacionais que controlam os elementos de memória e as saídas:

f_{D0}(A,B,C,D,E,F) = $\sum$(2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 13, 16, 17, 30, 31, 34, 35, 38, 39, 42, 43, 46, 47, 50, 51, 54, 55, 58, 59, 62, 63)

f_{D1}(A,B,C,D,E,F) = $\sum$(4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 13, 28, 29, 30, 31, 36, 37, 38, 39, 44, 45, 46, 47, 52, 53, 54, 55, 60, 61, 62, 63)

f_{D2}(A,B,C,D,E,F) = $\sum$(0, 8, 16, 24, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63)

f_{D3}(A,B,C,D,E,F) = $\sum$(9, 10, 11, 12, 13, 14, 15, 16, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 48, 49, 50, 51, 52, 53, 54, 55)

f_{D4}(A,B,C,D,E,F) = $\sum$(0, 8, 17, 18, 19, 20, 21, 22, 23, 25, 26, 27, 28, 29, 30, 31 ,48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63)

f_{Z0}(A,B,C,D,E,F) = $\sum$(1, 3, 5, 7, 8, 9, 11 ,13 ,15 ,16 ,17 ,19 ,21 ,23 ,25 ,27 ,29 ,31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63)

f_{Z1}(A,B,C,D,E,F) = $\sum$(1, 7, 8, 11, 13, 17, 23, 27, 29, 33, 39, 43, 45, 49, 55, 59, 61)

A cobertura dessas sete funções combinacionais, minimizadas como função de saída simples são mostradas a seguir:

f_{D0}(ABCDEF) =XXXX1X

f_{D1}(ABCDEF) = XXX1XX

f_{D2}(A,B,C,D,E,F) = 1XXXXX + XXX000

f_{D3}(A,B,C,D,E,F) = 01X000 + 0X11XX + 0X1X1X + 0X1XX1 + 1X0XXX

f_{D4}(A,B,C,D,E,F) = X1X1XX + X1XX1X + X1XXX1 + 11XXXX + 00X000

f_{Z0}(A,B,C,D,E,F) = 01000X + 00100X + XXXXX1

f_{Z1}(A,B,C,D,E,F) = XX1101 + XX1011 + 001000 + XX0111 + XX0001

O custo total dos ANDs para implementar essas sete funções é igual a 88.

A cobertura dessas mesmas funções, minimizadas como funções de múltiplas saídas são mostradas a seguir:

f_{D0}(A,B,C,D,E,F) = XXXX1X

f_{D1}(A,B,C,D,E,F) = XXX1XX

f_{D2}(A,B,C,D,E,F) = 00X000 + 01X000 + 1XXXXX

f_{D3}(A,B,C,D,E,F) = 01X000 + 0X11XX + 0X1X1X + 0X1XX1 + 1X0XXX

f_{D4}(A,B,C,D,E,F) = 00X000 + X1X1XX + X1XX1X + X1XXX1 + 11XXXX

f_{Z0}(A,B,C,D,E,F) = 001000 + 01000X + XXXXX1

f_{Z1}(A,B,C,D,E,F) = 001000 + XX0001 + XX0111 + XX1101 + XX1011

O custo dos ANDs para implementar as mesmas funções como função de múltiplas saídas, é igual a 76, pois, os implicantes primos 00X000, 01X000 e 001000 são comuns a duas funções.

A cobertura múltipla gerou um problema de 87 variáveis e 215 restrições.

Vale observar que o algoritmo MultPlex obteve o mesmo resultado obtido pelo programa Espresso.

4.6 Conclusão

Este capítulo tratou da minimização de funções booleanas com múltiplas saídas, evidenciando a necessidade de se considerar esse tipo de implementação em projetos de sistemas digitais.

O algoritmo MultPlex foi apresentado através de alguns estudos de casos.

Para a fase de geração dos implicantes primos, utilizou-se um método tabular, modificado para considerar funções booleanas com múltiplas saídas.

A fase de cobertura é efetuada em duas etapas, chamadas de cobertura múltipla e cobertura singular.

Na cobertura múltipla, consideram-se as funções de saída como um todo e formula-se um problema de programação linear inteira com os implicantes primos múltiplos e singulares. Com o resultado da cobertura múltipla, formulam-se as coberturas singulares, onde as funções de saída são consideradas isoladamente.

O algoritmo MultPlex representa uma contribuição definitiva na simplificação de funções booleanas de múltiplas saídas, pois define esta simplificação como um problema de programação matemática que foi resolvido por um método computacionalmente eficaz.

Capítulo 5

Redução de Estados, em Máquinas de Estados Finitos

A obtenção de máquinas seqüenciais, sem estados redundantes, é um problema de grande importância na síntese de circuitos seqüenciais, devido à redução de custo de sua implementação. Esta importância fica revigorada na busca de circuitos seqüenciais, totalmente testáveis, pois, a eliminação de redundâncias é condição necessária para a determinação dos testes.

O objetivo deste capítulo é apresentar um algoritmo para reduzir o número de estados internos, em máquinas seqüenciais de estados finitos.

Este algoritmo representa uma contribuição na solução do problema de redução de estados em máquinas de estados finitos completa ou não completamente especificada, propondo um método numérico para a seleção do conjunto mínimo de classes primas que definem as máquinas.

Para a solução do problema matemático, formulado a partir das classes primas, utilizou-se método de programação linear inteira e o Plano de Corte de Gomory, para considerar somente as soluções inteiras.

5.1 Introdução

A síntese de circuitos seqüenciais, auxiliada por computador, é uma necessidade cada vez maior para acompanhar a crescente complexidade dos circuitos VLSI (Very Large Scale Integrated). O uso sistemático de métodos computacionais diminui a possibilidade de não funcionamento correto do circuito, aumentando a confiabilidade do processo de síntese, além de reduzir o tempo despendido durante esta fase do projeto.

Nas metodologias de projeto top-down, a partir de uma descrição comportamental do sistema, passa-se a representações estruturais que podem ser fisicamente implementadas. No caso particular dos circuitos seqüenciais síncronos, estes são descritos em dois blocos principais: um, contendo a parte combinacional do circuito, e outro, sua memória.

O comportamento de um circuito pode ser descrito através de um diagrama de estados, no qual pode-se visualizar seu comportamento sob as várias condições de entrada. Em geral, é um conveniente ponto de partida para o projeto de circuitos seqüenciais. Este diagrama pode ser representado na forma de uma tabela, conhecida pelo nome de tabela de estados.

Na descrição comportamental de um circuito seqüencial, é freqüente a introdução de redundâncias próprias ao raciocínio humano. Assim, em geral, o número de estados, da descrição original, pode ser reduzido sem perda da descrição comportamental do circuito, permitindo a obtenção de máquinas seqüenciais com um número mínimo de estados.

O problema de minimizar o número de estados internos em máquinas de estados finitos, vem sendo tratado desde 1954, quando Huffman [76] desenvolveu um método para projetar circuitos seqüenciais, que é a base dos métodos atuais. Os trabalhos de Mealy [77], McCluskey [82], Paull e Unger [79] foram importantes no desenvolvimento das bases formais da definição do problema da minimização do número de estados internos das máquinas seqüenciais.

Em 1965, Grasselli e Luccio [84] propuseram um método de minimização do número de estados em máquinas seqüenciais, baseados nos conceitos de cobertura e fechamento. Conforme salientado por esses autores, esta metodologia permitia a possibilidade de uma formulação deste problema como um problema de programação linear inteira. Neste capítulo, essas idéias foram desenvolvidas permitindo sua formulação sistemática, e um método de programação foi especializado para a resolução desta classe de problemas.

5.2 Conceitos e Definições

Uma máquina de estados finitos é um modelo abstrato para descrever um circuito seqüencial síncrono. Por circuito seqüencial síncrono, entendem-se as máquinas seqüenciais onde os elementos de memória são claramente identificáveis (flip-flops) e nos quais está presente uma entrada particular, denominada relógio, que sincroniza os eventos a que está sujeita esta máquina. Seu comportamento é descrito como uma seqüência de eventos que ocorrem em instantes de tempos discretos.

O modelo de uma máquina de estados finitos síncrona, pode ser representado pela figura 5.1, na qual se destacam dois componentes:

- Parte combinacional;
- Memória.

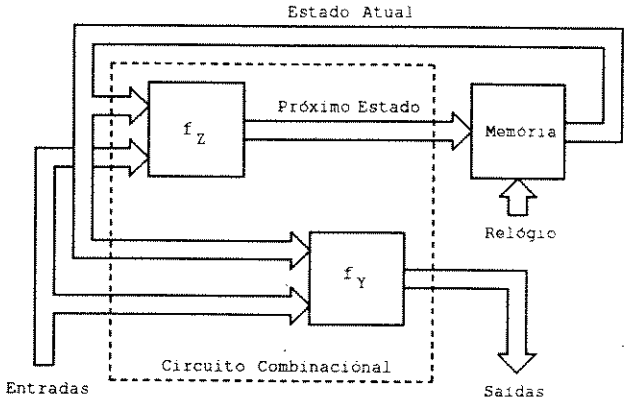


Figura 5.1: Modelo de uma máquina de estados finitos síncrona.

A memória armazena o estado atual da máquina e a parte combinacional realiza a função de saída f_Y e a função de próximo estado f_Z. Dependendo do estado atual e das entradas, f_Z gera o próximo estado e f_Y gera as saídas. A operação do circuito é sincronizada pela freqüência do relógio.

O comportamento da máquina pode ser descrito através de um diagrama de estados (grafo orientado), no qual cada estado é representado por um nó e os arcos representam as transições. O rótulo sobre o arco representa os valores da entrada e da saída. A figura 5.2 apresenta um exemplo de uma máquina de estados com 4 estados e suas correspondentes transições. Note que nesse exemplo, há uma variável binária de entrada e 3 variáveis binárias de saída. Neste exemplo foi utilizada a notação de Mealy [102], na qual a saída é função do estado atual e das entradas da máquina.

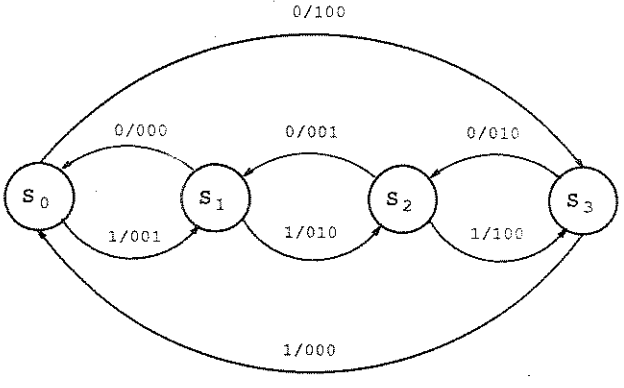


Figura 5.2: Diagrama de estados de uma máquina de Mealy.

O diagrama de estados pode ser apresentado na forma de uma tabela conhecida como tabela de estados, conforme ilustrado na figura 5.3. Se as saídas e os próximos estados são especificados para todas as combinações de entradas e estados, a máquina é completamente especificada, caso contrário, diz-se que a máquina é não completamente especificada.

Com o intuito de redução de estados em máquinas sequenciais, é conveniente introduzir-se os conceitos de equivalência e de compatibilidade entre estados.

Dois estados considerados como estados iniciais de uma máquina completamente especificada são equivalentes entre si, se as correspondentes seqüências de saídas forem as mesmas e os próximos

Est. Atual	P.Estado , Saída	
	Entradas	
	0	1
S_0	S_3 ,100	S_1,001
S_1	S_0,000	S_2,010
S_2	S_1,001	S_3,100
S_3	S_2,010	S_0,000

Figura 5.3: Tabela de estados correspondente ao diagrama de estados apresentado na figura 2.2

estados estiverem na mesma classe de equivalência para qualquer seqüência de entrada.

Esses estados equivalentes podem ser combinados em uma única classe, reduzindo-se o número de estados internos da máquina.

As classes de equivalências compostas por todos os estados equivalentes entre si, de uma dada máquina, são denominadas de classes máximas e, por definição, são disjuntas.

Uma máquina que só contenha classes máximas de equivalências é mínima, isto é, contém um número mínimo de classes de estados [102].

Nem sempre é possível determinar, para máquinas não completamente especificadas, se dois estados são equivalentes entre si, pois a equivalência implica que tanto a saída como o próximo estado sejam definidos para todas as possíveis seqüências de entrada.

Uma seqüência de entrada é dita aplicável em uma máquina, em um dado estado inicial, se o resultado dessa aplicação só resultar em transições para estados definidos.

Dois estados, considerando-os como estados iniciais, são compatíveis se as seqüências de saídas, ambas especificadas, forem as mesmas e as seqüências de próximos estados estejam nas mesmas classes de compatibilidade para toda seqüência de entrada aplicável .

Uma classe de estados constitui uma classe de compatibilidade se todos os seus pares de membros forem compatíveis entre sí, assim, a classe contendo os estados A, B e C formam uma classe de compatibilidade, se A for compatível com B, B com C e A com C.

Note que, para máquinas não completamente especificadas não se aplica a propriedade da transitividade. A conseqüência disso, é que as classes de compatibilidade não são necessariamente disjuntas.

Com o intuito do estabelecimento de um algoritmo para a redução de máquinas não completamente especificadas, introduz-se os seguintes conceitos.

Uma classe de compatibilidade que não está contida em nenhuma outra classe, é chamada de classe máxima de compatibilidade.

Dois estados constituem-se num par de condição de compatibilidade, se a compatibilidade entre eles é condição necessária para que um conjunto de estados forme uma classe de compatibilidade.

Uma classe de compatibilidade C_2 é excluída por uma classe de compatibilidade C_1, se C_1

contém C_2 e o conjunto de pares de condições de compatibilidade da classe C_2, contém o conjunto de pares de condições de compatibilidade da classe C_1.

Uma classe de compatibilidade que não é excluída por nenhuma outra classe, é chamada de classe prima de compatibilidade.

Note que, se uma classe de compatibilidade tem conjunto de pares de condições vazio, então não pode ser excluída por nenhuma outra, e portanto, é uma classe prima de compatibilidade.

Propriedade: Toda classe máxima de compatibilidade é prima [84].

5.3 Redução de estados em máquinas completamente especificada

A redução de estados em máquinas completamente especificadas é baseada no conceito de classe de equivalência de estados. Apresenta-se a seguir, duas técnicas para esta redução, denominadas de partição de equivalências e de tabela de condições [76]. O intuito da inclusão desse tópico na tese é para evidenciar as generalidades envolvidas na redução de máquinas não completamente especificadas tratadas na próxima seção.

Para a redução desse tipo de máquina, que é relativamente simples, pode-se utilizar a técnica de partição de equivalência e a tabela de implicações entre outras.

5.3.1 Partição de equivalência

Na redução por partição de equivalência, supõe-se inicialmente, que todos os estados da máquina original são equivalentes entre si.

Após essa aglutinação, faz-se uma primeira partição analisando-se exclusivamente as inconsistências de saídas.

- Partição de saída.

  Separa-se os estados em classes, tais que estados numa mesma classe tenham as mesmas saídas.

  Após essa partição, realiza-se uma análise das inconsistências entre os próximos estados.

- Partição de estados.

  Separa-se os estados em classes de equivalências, tais que, seus próximos estados estejam nas mesmas classes de equivalências.

Exemplo 1. Considere a tabela de estados apresentada na figura 5.4, onde a aplicação da partição de saída resulta nas classes A, C, E e B, D, F .

Utiliza-se neste trabalho, a notação ACE para designar a classe composta dos estados A, C e E.

A classe ACE tem BDF como próximos estados para a entrada 0 e os estados CE para a entrada 1.

Est. Atual	P.Estado , Saída	
	Entradas	
	0	1
A	D,1	E,0
B	D,0	F,0
C	B,1	E,0
D	B,0	F,0
E	F,1	C,0
F	C,0	B,0

Figura 5.4: Tabela de estados de uma máquina completamente especificada.

A classe BDF tem os estados BF como próximos estados para a entrada 1 e BCD para entrada 0. Os estados BD e o estado C estão em classes distintas. Este conflito é resolvido separando-se a classe de estados BDF nas classes BD e F.

A classe ACE tem CE como próximos estados para a entrada 1 e BDF para a entrada 0. Desde que BD e F estejam em classes distintas, a classe ACE é dividida nas classes AC e E.

A tabela reduzida da figura 5.5, tem quatro classes e é equivalente à tabela original com seis estados.

5.3.2 Tabela de condições

As classes de estados equivalentes podem ser determinadas através da tabela de condições, de modo que todos os pares de estados sejam examinados para o estabelecimento de possíveis equivalências.

Os seguintes passos são necessários:

- Monta-se a tabela de condições de modo que, todos os pares de estados possam ser analisados. Se dois estados são equivalentes, insere-se na célula correspondente um "*", se não são equivalentes, insere-se um "X", e no caso de haver condições possíveis de equivalências, insere-se estas condições na célula correspondente;
- Verifica-se a consistência da tabela de condições, isto é, remove-se sistematicamente as condições que não podem ser satisfeitas;
- Obtém-se as classes máximas de equivalência, através da transitividade.

Est. Atual	P.Estado , Saída	
	Entradas	
	0	1
AC	BD, 1	E, 0
BD	BD, 0	F, 0
E	F, 1	AC, 0
F	AC, 0	BD, 0

Figura 5.5: Tabela reduzida do exemplo 1.

Estas classes são disjuntas e resultam numa máquina mínima.

Exemplo 2. Na figura 5.6 é apresentada a tabela de condições para a máquina de estado do exemplo 1, apresentado na figura 5.4.

B	X				
C	BD	X			
D	X	*	X		
E	DF CE	X	CE BF	X	
F	X	CD BF	X	BF BC	X
	A	B	C	D	E

Figura 5.6: Tabela de condições da máquina de estados descrita na figura 2.4.

Note que o par de estados DF pode ser equivalente somente se os pares BF e BC o forem. Assim, esses dois pares são inseridos na célula correspondente ao par DF.

Na análise de consistência da tabela, a não equivalência de pares condições são removidas, assim, por exemplo, na célula corresponde ao par DF, as condições BF e BC são removidas, substituindo-se por um "X", indicando a não equivalência entre os estados DF decorrente da não equivalência entre os estados B e C.

Na figura 5.7 é apresentada a tabela de condições, após a remoção das inconsistências. Através da propriedade de transitividade da relação de equivalência, os pares de equivalência são expandidos até se obter as classes máximas de equivalência, que neste exemplo resulta em AC, BD, E e F.

B	X				
C	*	X			
D	X	*	X		
E	X	X	X	X	
F	X	X	X	X	X
	A	B	C	D	E

Figura 5.7: Tabela de implicação após verificada sua consistência.

5.4 Redução de estados em máquinas não completamente especificadas

A tabela de estados para máquinas não completamente especificadas, é consideravelmente mais difícil de ser reduzida do que as máquinas completamente especificadas, pois esta redução passa. a priori, pela determinação das seqüências aplicáveis de entrada.

Descreve-se a seguir, um método baseado no de Grasselli e Luccio [84], denominado método de cobertura e fechamento via programação matemática, utilizado para minimizar o número de estados internos em máquinas não completamente especificadas. Este método garante a obtenção de uma máquina mínima e permite a aplicação de programação matemática em sua implementação.

Este método consiste de três fases:

- Obtenção das classes máximas de compatibilidades;
- Geração das classes primas;
- Obtenção de um conjunto mínimo de classes primas, que constitui-se numa cobertura e fechamento da máquina original.

O método de cobertura e fechamento é apresentado através de um exemplo, que foi selecionado para evidenciar as diferenças entre a proposição original de Grasselli e Luccio, que dá ênfase em uma formulação tabular, e a proposta apresentada neste capítulo, cuja ênfase é na formulação e solução via programação matemática.

Exemplo 3.

Inicialmente, deve-se gerar para a máquina de estado da figura 5.8, as classes máximas de compatibilidade, o que é feito através da tabela de pares de condições de compatibilidade.

A figura 5.9a apresenta a tabela de pares de condições de compatibilidades e a figura 5.9b sua tabela consistente.

As classes máximas de compatibilidades são geradas a partir da tabela consistente, e resultam nas classes DE, ACD, BC e BE.

Na máquina reduzida, as classes retidas (cobertura) para sua descrição devem ser consistentes. isto é, o conjunto de condições de compatibilidade deve ser satisfeito por estas classes (fechamento).

Est. Atual	P.Estado , Saída			
	Entradas			
	1	2	3	4
A	-,-	-,-	E,1	-,-
B	C,0	A,1	B,0	-,-
C	C,0	D,1	-,-	A,0
D	-,-	E,1	B,-	-,-
E	B,0	-,-	C,-	B,0

Figura 5.8: Tabela de estados de uma máquina não completamente especificada.

B	X			
C	*	AD		
D	BE	AE	DE	
E	CE	BC	BC AB	BC
	A	B	C	D

(a)

B	X			
C	*	AD		
D	BE	X	DE	
E	X	BC	X	BC
	A	B	C	D

(b)

Figura 5.9: Tabela de condição (a) e sua tabela consistente (b).

Desta forma, é necessário se explicitar para cada classe, suas condições de compatibilidade. que podem sempre ser descritas por pares de condições.

As classes máximas de compatibilidade e seus pares de condições são:

$$DE \rightarrow BC$$
$$ACD \rightarrow DE, BE$$
$$BC \rightarrow AD$$
$$BE \rightarrow BC$$

As classes máximas de compatibilidade são primas, e as demais classes primas podem ser obtidas decompondo-se as classes máximas em subclasses e retendo-se aquelas que são primas.

Note que somente as classes máximas de compatibilidade com conjuntos de pares de condições não vazios precisam ser decompostas.

As classes primas de compatibilidades e seus pares de condições são:

$$DE \rightarrow BC$$
$$ACD \rightarrow DE, BE$$
$$BC \rightarrow AD$$
$$BE \rightarrow BC$$
$$AC \rightarrow \emptyset$$
$$AD \rightarrow BE$$
$$CD \rightarrow DE$$
$$B \rightarrow \emptyset$$
$$D \rightarrow \emptyset$$
$$E \rightarrow \emptyset$$

Note que as classes AC, B, D e E são classes primas sem condições de compatibilidade.

Para a determinação da máquina mínima, Grasselli e Luccio utilizam-se da tabela de cobertura e fechamento, que é uma extensão da técnica empregada para a redução da tabela de implicantes primos, utilizada pelo método de Quine-McCluskey na minimização de funções booleanas. considerando as restrições de condições de compatibilidade.

A figura 5.10 apresenta a tabela de cobertura e fechamento para o exemplo, que é composta de duas partes, uma relativa a cobertura e outra relativa ao fechamento.

A parte de cobertura é obtida por colunas que representam os estados originais da máquina e por linhas que representam suas classes primas. A marca " + " no cruzamento da tabela, indica que um determinado estado está contido numa dada classe.

A parte de fechamento é obtida definindo-se como coluna os pares de condições de compatibilidades e como linha as classes primas. Uma mesma condição dá origem a tantas colunas quantas forem o número de vezes que ela se constituir em condição para as classes primas. A marca " # " no cruzamento da tabela, indica que a condição está contida na classe correspondente, e a marca " * " indica que esta condição é necessária para a compatibilidade da classe em questão. Note que esta situação ocorre apenas uma vez em cada coluna.

Na proposta original de Grasselli e Luccio, a redução da tabela de cobertura é feita através

	A	B	C	D	E	BC	DE	BE	AD	BC	BE	DE
DE				+	+	*	#					#
ACD	+		+	+			*	*	#			
BC		+	+			#			*	#		
BE		+			+			#		*	#	
AC	+		+									
AD	+			+					#		*	
CD			+	+								*
B		+										
D				+								
E					+							

Cobertura Fechamento

Figura 5.10: Tabela de cobertura e fechamento para a máquina do exemplo 3.

de regras que visam a eliminação de linhas ou colunas da tabela, preservando as propriedades de cobertura e de condições de compatibilidade entre as classes retidas. Entre essas regras, destacam-se as seguintes:

- Elimine as classes primas que não possuem condições de compatibilidade e não são essenciais para os estados e não cobrem nenhuma das condições das demais classes primas.
- Selecione as classes primas, cujas colunas contêm apenas uma marca X, isto é, são essenciais, indicando que o estado ou a condição expressos nessas colunas são cobertos apenas por uma classe prima. Elimine todas as colunas cobertas por estas classes, que farão parte da solução final.
- Elimine as classes primas cobertas por uma outra classe que não tem condições de compatibilidade, isto é, se uma linha contém marcas X onde as demais também têm, então, todas as demais são eliminadas.
- Elimine os estados e as condições que contêm, em cada linha, as mesmas marcas que aquelas de uma outra coluna de referência, ou que na coluna de referência não haja marca alguma nesta linha.
- Elimine as classes primas que não cobrem estados ou condições, isto é, que contêm apenas condições de compatibilidade.

Aplicando-se estas regras, os estados da máquina original são cobertos e as condições de compatibilidade satisfeitas.

Quando não se puder reduzir a tabela de cobertura e fechamento, ela é dita cíclica. Neste caso, pode-se empregar o método de Petrick (método algébrico) para eliminar o impasse na aplicação

das regras e conseqüente determinação da máquina mínima. O método de Petrick é trabalhoso e consiste na obtenção de uma expressão booleana contendo todas as possíveis soluções do problema. Esta expressão consiste do produto lógico de uma série de funções que são somas lógicas obtidas a partir de cada coluna da tabela cíclica de cobertura e fechamento.

Ao invés de tratar o problema de cobertura e fechamento através de análises de exclusão, eliminação e seleção de classes primas de uma forma tabular, propõe-se neste trabalho a formulação de um problema de programação matemática, onde as classes de estados serão interpretadas como variáveis aritméticas 0 ou 1.

Na formulação do problema de programação matemática e na aplicação de algoritmos de solução, ACD = 1 indica que a classe prima composta pelos estados A, C e D faz parte da máquina reduzida e ACD = 0, que esta classe prima não faz parte da solução.

Assim, retornando-se à descrição da máquina através de sua classes primas e seus correspondentes pares de condições de compatibilidade, é possível formular-se um problema de programação matemática que descreva o problema de maneira simples e objetiva, contendo restrições de cobertura e fechamento.

Cada classe prima é considerada como uma variável do problema, independente do número de estados incluído na classe.

A função objetivo a ser minimizada consiste na soma das variáveis, pois o custo de cada classe é unitário, indicando o interesse de obtenção de uma máquina com o menor número possível de classes, sem se importar com quais delas se obtém a solução.

As restrições expressas nas desigualdades numéricas são de dois tipos, um representando a cobertura e outro representando o fechamento.

A formulação do problema de cobertura e fechamento para o exemplo é:

$$\text{MIN } B + D + E + DE + ACD + BC + BE + AC + AD + CD$$

$$ACD + AC + AD \geq 1$$
$$B + BC + BE \geq 1$$
$$ACD + BC + AC + CD \geq 1$$
$$D + DE + ACD + AD + CD \geq 1$$
$$E + DE + BE \geq 1$$
$$BC \geq DE$$
$$DE \geq ACD$$
$$BE \geq ACD$$
$$ACD + AD \geq BC$$
$$BC \geq BE$$
$$BE \geq AD$$
$$DE \geq CD$$
$$\text{B, D, E, DE, ACD, BC, BE, AC, AD, CD} \in \{\ 0,1\ \}$$

A desigualdade $ACD + AC + AD \geq 1$ significa que pelo menos uma dessas variáveis deve ser igual a 1 na solução do problema, para que o estado A, da máquina original, seja coberto por uma dessas classes primas.

A desigualdade $BC \geq DE$ significa que se a variável DE for igual a 1 na solução do problema, então a variável BC também deverá ser igual a 1, para que a condição de compatibilidade da classe DE seja satisfeita, isto é, se a classe prima DE estiver presente na máquina reduzida, então BC também deverá estar.

A solução desse problema é: BC=BE=AD=1 e B=D=E=DE=ACD=AC=CD=0, indicando que as classes primas BC, BE e AD formam uma cobertura fechada para os estados originais da máquina.

Esta solução foi obtida através do algoritmo proposto neste trabalho e denominado ReduPlex, que é apresentado em detalhes na próxima seção.

Est. Atual	P.Estado , Saída			
	Entradas			
	1	2	3	4
BC	BC,0	AD,1	BC,0	AD,0
BE	BC,0	AD,1	BC,0	BC,0
AD	-,-	BE,1	BE,1	-,-

Figura 5.11: Tabela mínima de estados correspondente a máquina original apresentada na figura 2.8.

5.4.1 ReduPlex

O método proposto, intitulado ReduPlex, para a redução do número de estados em máquinas seqüenciais, aplica-se às máquinas completa ou não completamente especificadas e consiste nos seguintes passos:

- 1 Monte a tabela de condições de Compatibilidade;
- 2 Obtenha a tabela consistente entre pares de estados;
- 3 Obtenha as classes máximas de compatibilidade;

  Se as classes máximas de compatibilidade são disjuntas, então a máquina reduzida tem tantos estados quantos forem as classes máximas de compatibilidade.
- 4 Se as classes máximas de compatibilidade não forem disjuntas, obtenha as classes primas de compatibilidade a partir das classes máximas;
- 5 Formule o problema de programação matemática a partir das classes primas e resolva-o;
- 6 Obtenha a máquina reduzida a partir da solução do problema de programação matemática.

5.5 Estudo de Casos

Através de alguns exemplos, evidenciam-se diferentes aspectos numéricos envolvidos na redução de estados em máquinas seqüenciais, quando da aplicação do algoritmo proposto.

5.5.1 Máquina M_1

Aplica-se o algoritmo ReduPlex a uma máquina completamente especificada para ilustrar o término do algoritmo na determinação e análise das classes máximas de compatibilidade, que neste caso são disjuntas.

A figura 5.12.a apresenta a tabela de estados da máquina M_1, e o resultado final da aplicação do algoritmo. A figura 5.13.a apresenta as condições de compatibilidade e a 5.13.b, apresenta as condições consistentes.

Estado Atual	Próximo, Saída	
	Entradas	
	0	1
A	A, 0	B, 1
B	E, 1	F, 1
C	E, 0	G, 0
D	G, 1	C, 1
E	B, 0	C, 1
F	G, 0	E, 0
G	B, 0	F, 1

(a) Tabela de Transição da Máquina M1.

Estado Atual	Próximo, Saída	
	Entradas	
	0	1
EG	BD, 0	CF, 1
CF	EG, 0	EG, 0
BD	EG, 1	CF, 1
A	A, 0	BD, 1

(b) Tabela Reduzida da Máquina M1

Figura 5.12: Tabela de estados da máquina M_1 e sua respectiva tabela reduzida.

As classes máximas de compatibilidade obtidas a partir da tabela consistente são: EG = {E,G}, CF = {C,F}, BD = {B,D}, A = {A}.

Note que EG representa uma classe máxima de compatibilidade, composta pelos estados E e G.

Neste caso, as classes máximas de compatibilidade são disjuntas, ou seja, não existe nenhum estado da máquina original M_1 que pertença a mais de uma classe máxima de compatibilidade e, portanto, formam uma máquina mínima.

5.5.2 Máquina M_2

A figura 5.14.a apresenta uma máquina NÃO completamente especificada cuja máquina reduzida tem classes de compatibilidade disjuntas. Este exemplo ilustra uma máquina que poderia ser

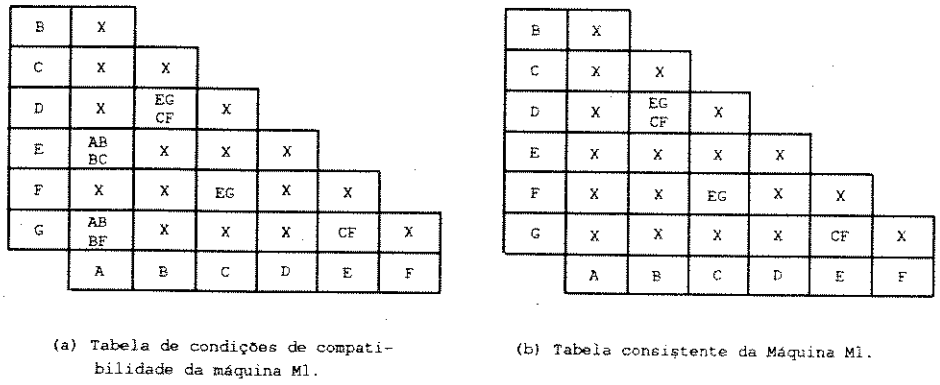


Figura 5.13: Tabela de condições de compatibilidade da máquina M_1 e sua respectiva tabela consistente.

completada na sua especificação e gera uma máquina compatível mínima.

A figura 5.15.a apresenta a tabela de condições de compatibilidade e a figura 5.15.b o resultado da eliminação das inconsistências da máquina M_2.

Estado Atual	Próximo, Saída	
	Entradas	
	0	1
A	A,0	B,1
B	-,1	F,1
C	E,0	G,0
D	E,1	-,-
E	B,0	C,1
F	G,0	E,0
G	B,0	F,1

(a) Tabela da transição da Máquina M2

Estado Atual	Próximo, Saída	
	Entradas	
	0	1
EG	BD,0	CF,1
CF	EG,0	EG,0
BD	EG,1	CF,1
A	A,0	BD,1

(b) Tabela reduzida da Máquina M2

Figura 5.14: Tabela de estados da máquina M_2 e sua respectiva tabela reduzida.

As classes máximas de compatibilidade da máquina M_2 são representadas pelos conjuntos EG, CF, BD e A, que, por serem disjuntos, formam uma máquina mínima, apresentada na figura 5.14.b.

Note que a máquina da figura 5.14.b é a mesma da figura 5.12.b. A máquina M_2 poderia ser completada na sua especificação original para tornar-se equivalente a máquina M_1, isto, entretanto, não é sempre possível como mostrado pelo exemplo da máquina M_3 a seguir.

5.5.3 Máquina M_3

Esta é uma máquina não completamente especificada cuja, máquina reduzida não resulta de nenhuma completação da sua especificação original.

A figura 5.16.a apresenta a máquina M_3 e a figura 5.17, a sua correpondente tabela consistente.

B	X					
C	X	X				
D	X	*	X			
E	AB BC	X	X	X		
F	X	X	EG	X	X	
G	AB BF	X	X	X	CF	X
	A	B	C	D	E	F

(a) Tabela de condições de compatibilidade da máquina M2

B	X					
C	X	X				
D	X	*	X			
E	X	X	X	X		
F	X	X	EG	X	X	
G	X	X	X	X	CF	X
	A	B	C	D	E	F

(b) Tabela consistente para a máquina M2.

Figura 5.15: Tabela de condições de compatibilidade da máquina M_2 e sua respectiva tabela consistente.

Estado Atual	Próximo, Saída	
	Entradas	
	0	1
A	A, 0	C, 0
B	B, 0	B, –
C	B, 0	A, 1

(a) Tabela de transição da Máquina M3

Estado Atual	Próximo, Saída	
	Entradas	
	0	1
AB	AB, 0	BC, 0
BC	–, 0	AB, 1

(b) Tabela reduzida da Máquina M3

Figura 5.16: Tabela de estados da máquina M_3 e sua respectiva tabela reduzida.

B	BC	
C	X	AB
	A	B

Figura 5.17: Tabela consistente da máquina M_3.

As classes máximas de compatibilidade da máquima M_3 são representadas pelos conjuntos BC e AB, que não são disjuntos.

Neste caso , para a redução da máquina é necessário obter as condições de compatibilidade para as classes máximas e resolver o problema de programação linear inteira associado.

As classes primas de compatibilidade e seus respectivos conjuntos de pares de condições de compatibilidade são os seguintes:

$$BC \rightarrow AB$$
$$AB \rightarrow BC$$
$$A \rightarrow \emptyset$$
$$B \rightarrow \emptyset$$
$$C \rightarrow \emptyset$$

A partir desses conjuntos, formula-se o problema de programação linear inteira (PLI):

MIN $A + B + C + AB + BC$

$$A + AB \geq 1$$
$$B + AB + BC \geq 1$$
$$C + BC \geq 1$$
$$AB \geq BC$$
$$BC \geq AB$$
$$\text{A, B, C, AB, BC} \in \{\ 0, 1\ \}$$

Resolvendo-se o PLI, obtém-se como solução mínima as classes primas AB e BC.

5.5.4 Máquina M_4

Na resolução do PLI, relaxa-se a condição de integralidade e aplica-se o algoritmo Simplex. Se a solução obtida no problema relaxado for toda inteira, esta solução é também solução do problema original, caso contrário, ativa-se a restrição de integralidade e aplica-se o Plano de Corte de Gomory, para considerar somente as soluções inteiras.

Este exemplo ilustra um caso de redução de estados de máquinas não completamente especificada, cujo problema de programação linear inteira exige a aplicação do Plano de Corte de Gomory.

A figura 5.18.a apresenta a máquina M_4, a figura 5.19.a, apresenta a tabela de condições de compatibilidade e a figura 5.19.b a correspondente tabela consistente.

As classes máximas de compatibilidade da máquina M_4 são: DE, ACD, BC, BE.

Note que estes conjuntos não são disjuntos, sendo, então, necessário obter-se as condições primas de compatibilidade.

As classes primas de compatibilidade e seus respectivos conjuntos de pares de condições de compatibilidade são:

Estado Atual	Próximo, Saída			
	Entradas			
	0	1	2	3
A	-,-	-,-	E,1	-,-
B	C,0	A,1	B,0	-,-
C	C,0	D,1	-,-	A,0
D	-,-	E,1	B,-	-,-
E	B,0	-,-	C,-	B,0

(a) Tabela de transição da Máquina M4

Estado Atual	Próximo, Saída			
	Entradas			
	0	1	2	3
DE	BC,0	DE,1	BC,-	BC,0
ACD	ACD,0	DE,1	BE,1	ACD,0
BC	ACD,0	ACD,1	BC,0	ACD,0
BE	BC,0	ACD,1	BC,0	BC,0

(b) Tabela reduzida da Máquina M4

Figura 5.18: Tabela de estados da máquina M_4 e a respectiva tabela reduzida.

B	X			
C	*	AD		
D	BE	AE	DE	
E	CE	BC	BC AB	BC
	A	B	C	D

(a) Tabela de condições de compatibilidade da máquina M4.

B	X			
C	*	AD		
D	BE	X	DE	
E	X	BC	X	BC
	A	B	C	D

(b) Tabela compatível para a máquina M4.

Figura 5.19: Tabela de condições de compatibilidade da máquina M_4 e sua respectiva tabela consistente.

$DE \rightarrow BC$
$ACD \rightarrow BE, DE$
$BC \rightarrow AD$
$BE \rightarrow BC$
$A \rightarrow \emptyset$
$B \rightarrow \emptyset$
$C \rightarrow \emptyset$
$D \rightarrow \emptyset$
$E \rightarrow \emptyset$

O problema de programação linear inteira formulado é:

$$\text{MIN } A + B + C + D + E + DE + ACD + BC + BE$$

$$A + ACD \geq 1$$
$$B + BC + BE \geq 1$$
$$C + ACD + BC \geq 1$$
$$D + DE + ACD \geq 1$$
$$E + DE + BE \geq 1$$
$$BC \geq DE$$
$$BE \geq ACD$$
$$DE \geq ACD$$
$$ACD \geq BC$$
$$BC \geq BE$$
$$\text{A, B, C, D, E, DE, ACD, BC, BE} \in \{\ 0,1\ \}$$

Note que na restrição de fechamento para a classe de compatibilidade BC, o par de condição de compatibilidade AD foi substituído pela classe prima que o contém.

Para este problema de PLI, a solução relaxada assume os valores, A = 0.5, DE = 0.5, ACD = 0.5, BC = 0.5 e BE = 0.5, sendo necessário aplicar o Plano de Corte de Gomory, cujo resultado é DE = 1, ACD = 1, BC = 1 e BE = 1, que corresponde à solução reduzida.

Um aspecto importante neste exemplo é que, devido a falta da propriedade de transitividade entre classes de estados compatíveis, várias máquinas reduzidas compatíveis à máquina M_4 podem ser geradas. A figura 5.20 apresenta uma outra máquina reduzida compatível à máquina M_4, alternativa a solução apresentada na figura 5.18b.

Estado Atual	Próximo, Saída			
	Entradas			
	0	1	2	3
DE	BE, 0	BE, 1	BC, -	BE, 0
ACD	BC, 0	DE, 1	BE, 1	ACD, 0
BC	BC, 0	ACD, 1	BE, 0	ACD, 0
BE	BC, 0	ACD, 1	BC, 0	BE, 0

Figura 5.20: Máquina reduzida compatível a máquina M_4.

5.5.5 Máquina M_5

Para que a máquina reduzida obtida seja mínima, é necessário gerar todas as subclasses primas das classes máximas de compatibilidade (primas por definição). A ausência dessa decomposição

não garante que a máquina obtida seja mínima. Este exemplo, extraído do artigo de Grasselli e Luccio [84] ilustra este aspecto.

A figura 5.21 apresenta a tabela de estados da máquina M_5.

Estado Atual	Próximo, Saída						
	Entradas						
	1	2	3	4	5	6	7
A	A,0	-,-	D,0	E,1	B,0	A,-	-,-
B	B,0	D,1	A,-	-,-	A,-	A,1	-,-
C	B,0	D,1	A,1	-,-	-,-	-,-	G,0
D	-,-	E,-	-,-	B,-	B,0	-,-	A,-
E	B,-	E,-	A,-	-,-	B,-	E,-	A,1
F	B,0	C,-	-,1	H,1	F,1	G,0	-,-
G	-,-	C,1	-,-	E,1	-,-	G,0	F,0
H	A,1	E,0	D,1	B,0	B,-	E,-	A,1

Figura 5.21: Tabela de estados da máquina M_5.

Utilizando-se a tabela de condições de compatibilidade, obtém-se as seguintes classes máximas de compatibilidades: CFG, DEH, BCD, ABDE, AG.

Os conjuntos de pares de condições de compatibilidade para as classes primas de compatibilidade são:

$$CFG \rightarrow CD, EH$$
$$DEH \rightarrow AB, AD$$
$$BCD \rightarrow DE, AB, AG$$
$$ABDE \rightarrow \emptyset$$
$$AG \rightarrow \emptyset$$
$$C \rightarrow \emptyset$$
$$F \rightarrow \emptyset$$
$$H \rightarrow \emptyset$$

Note que as classes máximas de compatibilidade foram decompostas somente em subconjuntos unitários.

O formulação do problema de programação linear inteira (PLI) associado é:

$$\text{MIN } C + F + H + CFG + DEH + BCD + ABDE + AG$$

$$ABDE + AG \geq 1$$
$$BCD + ABDE \geq 1$$
$$C + CFG + BCD \geq 1$$
$$DEH + BCD + ABDE \geq 1$$
$$DEH + ABDE \geq 1$$
$$F + CFG \geq 1$$
$$CFG + AG \geq 1$$
$$H + DEH \geq 1$$
$$BCD \geq CFG$$
$$DEH \geq CFG$$
$$ABDE \geq DEH$$
$$DEH + ABDE \geq BCD$$
$$ABDE \geq BCD$$
$$AG \geq BCD$$
$$\text{C, F, H, CFG, DEH, BCD, ABDE, AG} \in \{\ 0,1\ \}$$

Note que para a restrição de fechamento para a classe máxima de compatibilidade BCD, o par de condição de compatibilidade DE foi substituído pela soma algébrica das classes primas que o contém. A solução inteira para este problema de programação linear inteira é: CFG, DEH, BCD. ABDE, AG.

Uma das máquinas reduzidas para esta solução é apresentada na figura 5.22.

Estado Atual	Próximo, Saída						
	Entradas						
	1	2	3	4	5	6	7
CFG	BCD,0	BCD,1	AG,1	DEH,1	CFG,1	AG,0	CFG,0
DEH	ABDE,1	DEH,0	ABDE,1	BCD,0	ABDE,0	DEH,-	AG,1
BCD	BCD,0	ABDE,1	ABDE,1	BCD,-	ABDE,0	AG,1	AG,-
ABDE	ABDE,0	ABDE,-	ABDE,0	ABDE,1	ABDE,0	ABDE,1	ABDE,1
AG	AG,0	CFG,1	DEH,0	BEH,1	BCD,0	AG,0	CFG,0

Figura 5.22: Uma das máquinas reduzidas para a máquina M_5.

Decompondo-se as classes máximas de compatibilidade em todas as subclasses primas, obtém-se as seguintes classes primas e seus pares de condições de compatibilidade:

$ABDE \rightarrow \emptyset$
$BCD \rightarrow AB, AG, DE$
$CFG \rightarrow CD, EH$
$DEH \rightarrow AB, AD$
$BC \rightarrow \emptyset$
$CD \rightarrow AG, DE$
$CF \rightarrow CD$
$FG \rightarrow CD, FG$
$FG \rightarrow EH$
$DH \rightarrow \emptyset$
$AG \rightarrow \emptyset$
$F \rightarrow \emptyset$

A formulação do problema de programação liner inteira (PLI) associado é:

MIN $F + ABDE + AG + BCD + DEH + CFG + BC + CD + DH + CF + CG + FG$

$ABDE + AG \geq 1$
$ABDE + BCD + BC \geq 1$
$BCD + CFG + BC + CD + CF + CG \geq 1$
$ABDE + BCD + DEH + CD + DH \geq 1$
$ABDE + DEH \geq 1$
$CFG + CF + FG + F \geq 1$
$AG + CFG + CG + FG \geq 1$
$DEH + DH \geq 1$
$ABDE + DEH \geq BCD$
$AG \geq BCD$
$ABDE \geq DEH$
$CD + BCD \geq CFG$
$DEH \geq CFG$
$ABDE + DEH \geq CD$
$AG \geq CD$
$CD + BCD \geq CF$
$CD + BCD \geq CG$
$FG + CFG \geq CG$
$DEH \geq FG$
F, ABDE, AG, BCD, DEH, CFG, BC, CD, DH, CF, CG, FG $\in$ { 0,1 }

Resolvendo-se este problema de PLI, obtém-se as seguintes classes primas, solução mínima do problema: ABDE, DEH, BC, FG.

Uma das máquinas mínimas para esta solução é apresentada na figura 5.23.

Estado Atual	Próximo, Saída						
	Entradas						
	1	2	3	4	5	6	7
ABDE	ABDE, 0	ABDE, 1	ABDE, 0	ABDE, 1	ABDE, 0	ABDE, 1	ABDE, 1
DEH	ABDE, 1	DEH, 0	ABDE, 1	BC, 0	BC, 0	DEH, -	ABDE, 1
BC	BC, 0	DEH, 1	ABDE, 1	-, -	ABDE, -	ABDE, 1	FG, 0
FG	BC, 0	BC, 1	-, 1	DEH, 1	FG, 1	FG, 0	FG, 0

Figura 5.23: Uma das máquinas mínimas para a máquina M_5.

5.6 Conclusão

Neste capítulo apresentou-se uma contribuição na solução do problema de redução de estados em máquinas de estados finitos completa ou não completamente especificada, propondo um método numérico para a seleção do conjunto mínimo de classes primas que definem as máquinas.

Para a solução do problema matemático formulado, a partir das classes primas e suas condições de compatibilidade, utilizou-se o algoritmo Simplex, modificado para considerar as particularidades do problema.

No caso de se obter uma solução otimista não inteira, utilizou-se o Plano de Corte de Gomory para considerar somente as soluções inteiras.

O estudos de vários casos mostraram que o método é uma alternativa eficiente (tempo e memória) para a solução do problema de redução de estados em máquinas de estados finitos.

Capítulo 6

Conclusões

Este trabalho tratou da síntese de circuitos digitais, empregando a programação matemática nas fases de projeto aqui abordadas.

No capítulo 1, foram apresentados os conceitos básicos da álgebra de Boole e as notações utilizadas nos demais capítulos do trabalho. A apresentação do método de Quine-McCluskey para a minimização de funções booleanas, forneceu ao leitor a base necessária para o entendimento do método de cobertura, através da programação linear inteira 0 e 1, apresentada no capítulo 3.

Apresentou-se também, um método quase ótimo para a minimização de funções boolenas, uma importante contribuição para contornar a ineficiência computacional dos tradicionais métodos de minimização de funções booleanas.

No capítulo 2, foi apresentado o algoritmo GeraPlex, que obtém todos os implicantes primos de uma função booleana, aplicando-se a operação de consenso aos ramos de uma árvore binária, onde os caminhos (raiz até folha) representam os mintermos e os irrelevantes da função.

Da aplicação da operação de consenso aos caminhos da árvore, define-se fusão, expansão e deslocamento, que diminuem, aumentam e mantêm, repectivamente, os caminhos na árvore.

Este algoritmo mostrou-se eficiente, em termos de tempo e memória, quando comparado com o tradicional método tabular de geração de implicantes primos. O GeraPlex pode ser estendido para considerar funções com múltiplas saídas.

No capítulo 3, tratou-se da cobertura de funções booleanas, sob a óptica da programação matemática e propôs-se um método baseado no algoritmo Simplex, modificado para considerar as particularidades do problema. No lugar do pivoteamento tradicional, foi utilizado combinações lineares das linhas do tableau para preservar os valores inteiros dos seus coeficientes.

Encontrada uma solução otimista (solução ótima, porém infactível), utilizou-se o Plano de Corte de Gomory para levar em conta somente as restrições de soluções inteiras.

Uma série exaustiva de testes, mostrou a viabilidade desse tipo de enfoque no problema de cobertura.

Um aspecto importante é a possibilidade de se utilizar todos os avanços na área da programação matemática para a solução do problema de cobertura de funções booleanas.

No capítulo 4, o problema de funções booleanas com múltiplas saídas foi abordado.

A cobertura das múltiplas funções, quando o critério de otimização é o número de gates ANDs, pôde ser formulado como um problema de programação linear inteira 0 e 1 e resolvido em duas etapas, chamadas de cobertura múltipla e cobertura singular.

O Algoritmo MultPlex, desenvolvido para a resolução deste problema, representa uma contribuição significativa, pelo seu desempenho computacional e pela abertura desta classe de problemas à programação matemática.

No capítulo 5, foi apresentado o algoritmo ReduPlex, uma contribuição na solução do problema de redução de estados em máquinas de estados finitos completa ou não completamente especificadas.

Foi proposto um método numérico para a seleção do conjunto mínimo de classes primas de estados, que definem as máquinas e que mostrou-se eficiente e adequado à solução deste problema.

Apêndice A

Programação Linear

Este apêndice foi inserido no texto para fornecer ao leitor as idéias básicas da programação linear.

O algoritmo Plex, apresentado no capítulo 3, é uma versão modificada do método Simplex, motivo pelo qual este foi descrito neste apêndice.

Descreveu-se também, o método de duas fases, dando ênfase ao método de uma única variável artificial.

O algoritmo Plex considera as restrições inteiras, em particular as restrições 0 e 1, e para tanto utilizou-se o Plano de Corte de Gomory, cuja aplicação necessita do Dual Simplex, utilizado para transformar a solução otimista numa solução ótima.

A.1 Introdução

O problema de otimização consiste em maximizar ou minimizar uma função-objetivo, cujas variáveis estão sujeitas a restrições.

Quando a função-objetivo é linear e as variáveis estão sujeitas a restrições de igualdade e ou restrições de desigualdades, utiliza-se a programação linear para resolver tal problema.

O principal método para resolver problemas de programação linear é o método Simplex, que foi inicialmente apresentado por Dantzig em 1949 [117].

O método Simplex é um procedimento iterativo que permite, dada uma solução inicial, determinar uma solução ótima de maneira sistemática.

A.2 Formulação do problema linear

O problema de programação linear pode ser formulado como:

$$
\begin{aligned}
&\text{Minimizar } c_1x_1 + c_2x_2 + ... + c_nx_n \\
&\text{Sujeito a} \\
&a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \geq b_1 \\
&a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \geq b_2 \\
&\qquad\qquad . \quad . \quad . \quad . \quad . \\
&\qquad\qquad . \quad . \quad . \quad . \quad . \\
&\qquad\qquad . \quad . \quad . \quad . \quad . \\
&a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \geq b_m \\
&x_i \geq 0, \text{ para i = 1, 2, ..., n.}
\end{aligned}
$$

que em notação matricial tem-se

$$
\begin{aligned}
&\text{Min } Z = CX \\
&AX \geq B \\
&X \geq 0
\end{aligned}
$$

onde

$$
\begin{aligned}
&\text{C} = [\ c_j\ ] : \text{vetor linha} \\
&\text{X} = [\ x_j\ ] : \text{vetor coluna} \\
&\text{B} = [\ b_i\ ] : \text{vetor coluna} \\
&\text{A} = [\ a_{ij}\ ] : \text{matriz } m \times n
\end{aligned}
$$

A função $Z = c_1x_1 + c_2x_2 + ... + c_nx_n$ é chamada de função-objetivo a ser minimizada. Os coeficientes $c_1, c_2, ..., c_n$, representam os custos unitários de cada variável e $x_1, x_2, ..., x_n$ são as variáveis a serem determinadas.

A inequação

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i$$

representa a i-ésima restrição. Os coeficientes a_{ij} e b_i para $i = 1, 2, ..., m$ e $j = 1, 2, ..., n$ são chamados de dados tecnológicos do problema.

O problema pode ser escrito como:

$$Min \quad Z = \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_{ij}x_j \geq b_i$$

para i= 1, 2, ..., m e $x_1, x_2, ..., x_n \geq 0$

O conjunto de variáveis $x_1, x_2, ..., x_n$, satisfazendo todas as restrições, é chamado de ponto factível ou vetor factível. O conjunto de todos esses pontos constitui a região ou espaço factível.

O teorema fundamental da programação linear [117] estabelece que, se o problema linear tem solução, então tem uma solução num dos pontos extremos da região factível, denominados vértices. Os vértices da região factível recebem o nome de soluções básicas. Neste texto, solução significa solução factível que minimiza o critério.

A.3 O método Simplex

O método Simplex é o algoritmo que sistematiza o processo de obtenção de pontos extremos. na região factível, à procura da solução ótima. O algoritmo inicia-se com uma solução básica factível e move-se para uma solução básica factível que decresce o valor da função objetivo, até uma solução ótima ser encontrada ou certificar-se de que a função-objetivo é ilimitada ou que o problema é infactível.

O tableau Simplex é a tabela utilizada para sistematizar as iterações do método Simplex. A figura A.1 apresenta um tableau Simplex genérico.

O método Simplex consiste nos seguintes passos:

- 1. Transformação das desigualdades em igualdades, através do uso das variáveis de excesso, levando à chamada forma preparada.

  Uma solução ótima para qualquer problema de programação linear ocorre num dos vértices da região factível. Para achar as coordenadas dos vértices, precisa-se resolver o sistema de equações lineares formado pelas restrições. Como não se pode trabalhar com desigualdades, deve-se transformá-las em igualdades, subtraindo-se de cada desigualdade uma nova variável, não negativa, denominada variável de excesso. Essa variável compensa o possível excesso da restrição em relação ao valor determinado para o lado direito da restrição.

- 2. Uso do tableau Simplex para a obtenção sistemática das soluções (resolução do sistema pelo método de Gauss-Jordan, que é definido através de operações denominadas de pivoteamento).

	xB1		xBr		xBm		xj		xk		
	0	...	0	...	0	...	zj-cj	...	zk-cK	...	cBb
xB1	1	...	0	...	0	...	y1j	...	y1k	...	b1
	⋮		⋮		⋮		⋮		⋮		⋮
xBr	0	...	1	...	0	...	yrj	...	yrk	...	br
	⋮		⋮		⋮		⋮		⋮		⋮
xBm	0	...	0	...	1	...	ymj	...	ymk	...	bm

Figura A.1: Tableau Simplex genérico.

O sistema possui m+n variáveis e m equações. Portanto pode-se fixar n dessas variáveis no limite inferior 0 e encontrar uma única solução para o sistema de equações resultante.

- 3. Busca da próxima solução.

  A mudança da solução, em busca da solução ótima, é feita anulando-se uma das m variáveis. que constituem uma base para o sistema de equações, e portanto introduzindo na base uma das n variáveis que estava fora da base.

  O método Simplex vai escolher de modo adequado a nova variável que se anula e a variável nula que se torna, em princípio, diferente de zero, mantendo sempre n variáveis nulas. Assim. tem-se sempre uma nova solução, através da resolução de um sistema de m equações com m+n incógnitas.

  O índice da variável não básica (de valor nulo), que entra na base, é determinado a partir de uma análise dos custos relativos da função; seja k o índice encontrado.

  A variável básica, que vai se anular no lugar da variável não básica, candidata a entrar na base, será definida pelo menor valor da relação entre b_i e o coeficiente y_{ik}. O índice da linha do tableau que contém este mínimo é denotado por r.

$$\mathrm{r} = \arg\ \{\ Min_i\{\tfrac{b_i}{y_{ik}} \geq 0\}\ \}$$

- 4. Transformação do tableau para obter uma nova solução.

  Para se obter uma nova solução, é necessário que o índice da variável que vai entrar na base esteja associado à matriz identidade, obtida fazendo-se um pivoteamento no elemento y_{rk}.

- 5. Repete-se os passos 3 e 4 até se achar a solução ótima.

  Na solução ótima, os custos relativos serão não negativos.

A.3.1 Exemplo

$$\text{Min } Z = -x_1 - 3x_2$$

$$x_1 + x_2 \leq 6$$
$$-x_1 + 2x_2 \leq 8$$
$$x_1, x_2 \geq 0$$

Sua forma padrão é:

$$\text{Min } Z = -x_1 - 3x_2$$

$$x_1 + x_2 + s_1 = 6$$
$$-x_1 + 2x_2 + s_2 = 8$$
$$x_1, x_2, s_1, s_2 \geq 0$$

Observe que foram acrescentadas as variáveis de excesso s_1 e s_2, que devem ser não negativas. A figura A.2 apresenta o tableau inicial, cujas variáveis básicas são $I = \{s_1, s_2\}$. .

x1	x2	s1	s2	
-1	-3	0	0	0
1	1	1	0	6
-1	2	0	1	8

Figura A.2: Tableau inicial apresentando uma base inicial factível.

A variável x_2 (cujo coeficiente de custo relativo é -3) é candidata a entrar na base.
A variável que sai da base é s_2, pois $\frac{8}{2}$ é menor que $\frac{6}{1}$.
Para que a variável x_2 entre na base, pivoteia-se no elemento $y_{1,1} = 2$.
Na figura A.3 apresenta-se o tableau, após o pivoteamento, cuja nova base é definida por $I = \{s_1, x_2\}$.

x1	x2	s1	s2	
-5/2	0	0	3/2	12
3/2	0	1	-1/2	2
-1/2	1	0	1/2	4

Figura A.3: Tableau após o pivoteamento no elemento $y_{1,1}$.

Note que ainda existe custo relativo negativo $\{-\frac{5}{2}\}$; isto indica que, se a variável x_1 entrar na base, pode-se melhorar a função-objetivo.

De maneira similar ao pivoteamento anterior, x_1 deve entrar na base e s_1 sair, portanto, pivoteia-se no elemento $y_{1,1} = \frac{3}{2}$.

O novo tableau é apresentado na figura A.4, onde os custos relativos são não negativos, indicando que se obteve a solução ótima, $x_1 = \frac{4}{3}$ e $x_2 = \frac{14}{3}$.

x1	x2	s1	s2	
0	0	5/3	2/3	46/3
1	0	2/3	-1/3	4/3
0	1	0	1/3	14/3

Figura A.4: Tableau após o pivoteamento no elemento $y_{1,1}$.

A.4 Método das duas fases

Para a aplicação do método Simplex, é necessário que exista uma solução básica inicial. Se não se dispõe de uma, é possível obtê-la através da resolução de um Simplex artificial, denominado primeira fase.

Neste apêndice, é apresentado um método eficaz de determinação de uma solução básica factível inicial através do método de uma única variável artificial.

No algoritmo Plex, utilizou-se a técnica de uma variável artificial, para se chegar numa base inicial factível. Nesta técnica, adiciona-se uma única variável artificial x_a, com coeficientes -1 em cada restrição, cuja componente do vetor b' é menor do que zero.

A variável artificial x_a entra na base, selecionando a linha pivô r, de acordo com o critério.

$$\mathrm{r} = \arg\min_i\{b_i\}$$

Inserindo-se a variável artificial x_a na base e removendo-se a variável x_r (pivoteando na linha r e coluna referente a variável artificial), tem-se um novo valor para o vetor lado direito b.

Desse modo, tem-se uma solução básica factível para o problema artificial. Resolvendo-se o problema artificial através do algoritmo Simplex, encontra-se uma solução ótima para esse problema artificial, que tem necessariamente a variável artificial fora da base. Desta forma, a solução ótima do problema artificial constitui-se numa solução básica inicial factível para o problema original. Recompondo-se os coeficientes de custo do problema original, tem-se, portanto, o problema na sua forma preparada.

A.5 O problema de programação linear inteira

Na resolução de um problema de programação linear, nem sempre a solução ótima ocorre em pontos cujas coordenadas são números inteiros, condição indispensável na formulação dos problemas tratados nesta tese.

Existem técnicas para se determinar soluções inteiras ótimas, dentre as quais destaca-se o Plano de Corte de Gomory [117].

Um problema de programação linear inteira pode ser escrito como:

$$Min \quad Z = \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} a_{ij}.x_j \geq b_i$$

para i= 1, 2, ..., m e $x_1, x_2, ..., x_n \geq 0$
$x_1, x_2, ..., x_n$ inteiros

Um método de se resolver este problema é relaxar a restrição de integralidade e resolver o problema como um programa linear. Se na otimalidade todas as variáveis têm valor inteiro. então tem-se uma solução ótima para o programa linear inteiro. Caso contrário, deve-se adicionar uma nova restrição ao problema linear. Essa restrição adicional deverá eliminar a solução da programação linear não inteira ótima, sem eliminar qualquer solução inteira factível candidata a ótima.

Então, adiciona-se a nova restrição ao tableau ótimo e aplica-se o método Dual Simplex (apresentado na próxima seção) no novo programa linear. A nova solução pode ser ou não ou inteira. O procedimento de adicionar restrições de corte é repetido, até que todas as soluções inteiras sejam encontradas, ou um resultado infactível seja encontrado indicando que o problema não tem solução inteira.

Para gerar a restrição de corte, considere um tableau Simplex, resultado de numa solução não inteira (b'_r é não inteiro).

Considere as colunas j, tais que:

- P_{rj} é a parte inteira contida de y'_{rj};
- P_r é a parte inteira contida de b'_r;
- F_{rj} é a parte fracionária de y'_{rj}, ou seja $F_{rj} = y_{rj} - P_{rj}$;
- F_r é a parte fracionária de b'_r, ou seja $F_r = b'_r - P_r$.
- P_{ri}, P_r, F_{ri} e F_r são nulo para as demais colunas.

A restrição de corte é dada por:

$$\sum_{j} F_{rj} x_j \geq F_r$$

Esta inequação é acrescentada no conjunto de restrições e é transformada numa equação de modo que sua variável de folga pertença a nova base.

A.6 O método Dual Simplex

Para cada problema linear, denominado Primal, existe um outro problema linear associado, denominado Dual.

O problema dual é obtido do problema primal, utilizando-se as seguintes condições:

- Se a função-objetivo do problema primal é minimizada, a função-objetivo do problema dual é maximizada;
- O número de variáveis do dual é igual ao número de restrições do primal;
- O número de restrições do dual é igual ao número de variáveis do primal;
- Os coeficientes de custo do primal formam o vetor lado direito (b) do dual;
- O vetor lado direito do primal (b) forma a função-objetivo do dual;
- Todas as variáveis são não negativas em ambos os problemas.

Vértices não factíveis do problema Primal são vértices factíveis do problema Dual [117].

A implicação desse resultado é que, enquanto o Simplex primal inicia-se factível e não ótimo e mantém a factibilidade até encontrar o ótimo, um procedimento similar pode ser desenvolvido para problemas que se iniciam infactíveis.

O método dual Simplex resolve o problema dual diretamente do tableau Simplex (primal). Em cada iteração move-se, de uma solução básica factível do problema dual para uma solução básica factível melhorada, até a otimalidade do dual (e também do primal) ser encontrada ou então concluir-se que o dual é ilimitado e que, consequentemente, o primal é infactível.

O método dual Simplex consiste nos seguintes passos:

- 1. Determina-se uma base I para o Primal;
- 2. Se $b' \geq 0$, pare; a solução é ótima. Caso contrário, selecione a linha correspondente a $b_r = \text{mínimo } \{b_i\}$.
- 3. Se $y_{rj} \geq 0$ para todo j (variável fora da base), pare; o dual é ilimitado e o primal é infactível. Caso contrário, selecione a coluna pivô k, com:

$$\frac{z_k - c_k}{y_{rk}} = maximo_j \{ \frac{z_j - c_j}{y_{rj}} \quad : \quad y_{rj} < 0 \}$$

- 4. Pivoteie em y_{rk} e retorne ao passo 2.

x1	x2	x3	s1	s2	s3	
0	0	0	1	1	1	-3
1	0	0	-1/2	-1/2	1/2	1/2
0	1	0	-1/2	1/2	-1/2	1/2
0	0	1	1/2	-1/2	-1/2	1/2

Figura A.5: Tableau infactível obtido após várias iteração do Simplex.

A.6.1 Exemplo

Como exemplo de aplicação do Corte Fracional Dual de Gomory, considere o tableau infactível (no sentido que a solução é não inteira) apresentado na figura A.5.

Para gerar a nova restrição, seleciona-se a varíavel básica x_1 (não inteira):

$$x_1 - \frac{1}{2}s_1 - \frac{1}{2}s_2 + \frac{1}{2}s_3 = \frac{1}{2}$$

Obtém-se P_r, P_{rj}, F_{rj} e F_r (r=1) como:

$$P_1 = 0,\ P_{14} = -1,\ P_{15} = -1,\ P_{16} = 0$$

$$F_1 = \frac{1}{2},\ F_{14} = \frac{1}{2},\ F_{15} = \frac{1}{2},\ F_{16} = \frac{1}{2}$$

A restrição de corte é:

$$\frac{1}{2}s_4 + \frac{1}{2}s_5 + \frac{1}{2}s_6 \geq \frac{1}{2}$$

A figura A.6 apresenta o tableau, após inserir-se a restrição de corte, no qual é aplicado o Dual Simplex.

x1	x2	x3	s1	s2	s3	s4	
0	0	0	1	1	1	0	-3
1	0	0	-1/2	-1/2	1/2	0	1/2
0	1	0	-1/2	1/2	-1/2	0	1/2
0	0	1	1/2	-1/2	-1/2	0	1/2
0	0	0	-1/2	-1/2	-1/2	1	-1/2

Figura A.6: Tableau após inserir-se a restrição de corte.

A figura A.7 apresenta o tableau após a aplicação de um passo do dual Simplex.

Observe que foi obtido uma solução ótima e factível, ou seja, a solução é agora inteira, com $x_1 = 1$ e $x_2 = 0$ e $x_3 = 1$.

x1	x2	x3	s1	s2	s3	s4	
0	0	0	0	0	0	2	-4
1	0	0	0	0	1	-1	1
0	1	0	-1	0	-1	1	0
0	0	1	1	0	0	-1	1
0	0	0	1	1	1	-2	1

Figura A.7: Tableau após a aplicação do dual Simplex.

Apêndice B

Descrição detalhada do algoritmo GeraPlex

O algoritmo GeraPlex obtém os implicantes primos de uma função booleana, e além disso, determina os implicantes primos essenciais, com vistas a formulação de um problema de programação matemática que minimiza a representação da função.

Este apêndice descreve o algoritmo GeraPlex detalhadamente.

- 1 - Montagem das Árvores

  A árvore de mintermos contém somente os mintermos da função enquanto que a árvore de implicantes contém os mintermos e os irrelevantes. O procedimento para se montar as duas árvores é o mesmo.

  - 1.1 - Cria-se um nó como sendo o nó raiz. Cada nó da árvore é composto por três apontadores rotulados por ZERO, DC e UM. Os mintermos e irrelevantes são representados por caminhos que vão da raiz (bit mais significativo) a uma folha (bit menos significativo) da árvore, onde os apontadores utilizados pelos ramos dos caminhos são os dígitos dos bits que representam o mintermo ou irrelevante.
  - 1.2 - Seleciona-se um mintermo (irrelevante) para ser inserido como um ramo (elo ente dois nós) da árvore.
  - 1.3 - Seleciona-se o bit mais significativo do mintermo (irrelevante) e utiliza-se o apontador correspondente ao nó raiz para apontar para o próximo nó da árvore.
  - 1.4 - Seleciona-se o próximo bit do mintermo (irrelevante) e utiliza-se o apontador correspondente para apontar para o próximo nó da árvore.
  - 1.5 - Repete-se o passo 1.4 até que o bit menos significativo seja inserido como um ramo da árvore. Todos os apontadores do nó apontado pelo nó do bit menos significativo (folha) apontam para o NULL.
  - 1.6 - Se ainda houver mintermo (irrelevantes) que ainda não foi inserido na árvore retorna-se ao passo 1.2, caso contrário segue-se para o passo 1.7.
  - 1.7 - Marca-se com um asterisco todos os nós que representam as folhas da árvore (bit menos significativo) cujo os apontadores ZERO e UM são utilizados.
  - 1.8 - Seleciona-se um nó marcado com asterisco e caminha-se para o próximo nó, no sentido da raiz. Marque esse nó com um asterisco caso os caminhos dos ramos dos apontadores ZERO e UM desse nó apontam para nós que estão marcados com asterisco.
  - 1.9 - Repete-se o passo 1.8 até que todos os nós marcados sejam analisados.

    Os ramos correspondentes aos nós marcados são chamados de sub-árvores completas. Este procedimento de marcar as sub-árvores completas torna o passo de geração de implicantes primo mais eficiente.

- 2 - Geração dos Implicantes Primos

  - 2.1 - Seleciona-se um dos nós das folha (nó mais distante da raiz)
  - 2.2 - Verifica-se se o nó selecionado está marcado com um asterisco (sub-árvore completa). Em caso afirmativo, elimine o ramo cujo apontador é ZERO e desloque o ramo cujo apontador é UM para o apontador DC.
  - 2.3 - Havendo nó das folhas (nível 0) ainda não analisados, retorne ao passo 2.1.
  - 2.4 - Avance para o próximo nível de nós no sentido da raiz.

- 2.5 - Seleciona-se um nó desse nível e verifica-se se o nó está marcado com um asterisco. Em caso afirmativo elimina-se o ramo cujo apontador é ZERO e desloca-se o ramo cujo apontador é UM para o apontador DC, caso contrário verifica-se se um dos apontadores desse nó aponta para um nó marcado com um asterisco. Em caso afirmativo mantém-se o ramo cujo apontador aponta para o nó marcado e desloca-se o outro ramo para o apontador DC. Se o nó analisado não aponta para nós marcados com asterisco segue-se para o passo 2.6 caso contrário segue-se para o passo 2.10.
- 2.6 - Verifica-se se os apontadores ZERO e UM desse nó são utilizados. Em caso afirmativo segue-se para o passo 2.7, caso contrário segue-se para o passo 2.10.
- 2.7 - Compara-se o caminho (ramos entre vários nós) cujo apontador é zero com o caminho cujo apontador é UM. Se os caminhos forem iguais elimina-se o ramo cujo apontador é ZERO e desloca-se o ramo cujo apontador é UM para o apontador DC (Fusão), feito isso segue-se para o passo 2.10, caso contrário segue-se para o passo 2.8.
- 2.8 - Verifica-se se o caminho de um dos apontadores está contido no caminho do outro apontador. Em caso afirmativo, o ramo do caminho contido é deslocado para o apontador DC (deslocamento), feito isso segue-se para o passo 2.11, caso contrário segue-se para o passo 2.9.
- 2.9 - Gera-se caminhos que estejam contidos em ambos os ramos ZEROS e UM do nó (expansão).
- 2.10 - Se existir nó nesse nível que ainda não foi analisado retorne ao passo 2.5, caso contrário segue-se para o passo 2.11.
- 2.11 - Se o nó da raiz ainda não foi analisado retorna-se ao passo 2.4, caso contrário segue-se para o passo 3.

• 3 - Determinação dos implicantes primos essenciais

- 3.1 - Seleciona-se um caminho da árvore de mintermos e verifica-se quantos caminhos na árvore de implicante cobrem o mintermo selecionado. Se apenas um implicante cobre o mintermo selecionado elimina-se esse implicantes da árvore marcando-o como essencial e elimina-se da árvore de mintermos todos os mintermos cobertos pelo implicante marcado.
- 3.2 - Repete-se o passo 3.1 até que todos os mintermos da árvore de mintermos tenham sido tratados.
- 3.3 - Se a árvore de mintermos contém ramos que não foram eliminados, verifica-se quais os implicantes restantes na árvore de implicantes cobrem esses mintermos e formula-se o problema de programação matemática, caso contrário fim.

Apêndice C

Descrição detalhada do algoritmo Plex

Este apêndice descreve detalhadamente o algoritmo intitulado Plex, que obtém a cobertura mínima de uma função booleana, resolvendo-se um problema de programação linear 0 e 1, formulado a partir de todos os implicantes primos gerados pela aplicação da operação do consenso (algoritmo GeraPlex) aos caminhos de uma árvore binária, que representam os mintermos e irrelevantes de uma função booleana.

- 1 - Montagem do Tableau

  - 1.1 - A primeira linha do Tableau representa a função objetivo e é ordenada de forma crescente, da esquerda para a direta, de acordo com os coeficientes de custo. As variáveis (implicantes primos) da função objetivo são obtidas pela aplicação da operação de consenso numa estrutura em árvore ternária onde cada caminho da árvore representa um vértice da função.

    As outras linhas da Tableau representam restrições (mintermos) do problema.

    A intersecção de uma linha (mintermo) com uma coluna (implicante primo) tem valor igual a 1 se o implicante primo cobre o mintermo. Nas demais intersecções é atribuído o coeficiente 0.

  - 1.2 - Uma coluna adicional é inserida à esquerda do Tableau, representando o vetor b (recurso). Essa coluna tem, na primeira linha, o coeficiente igual a 0 e nas demais linhas tem coficiente igual a 1 (cobertura de cada mintermo por pelo menos 1 implicante primo).

  - 1.3 - Procure, a partir da segunda coluna, por linhas contendo um único coeficiente igual a 1. O implicante primo, correspondente a essa coluna, é um implicante primo essencial, podendo ser eliminado do Tableau. Elimine também todas as linhas (mintermos), contendo 1 nessa coluna. Este passo não é necessário se na montagem inicial do tableau não foram considerados os implicantes primos essenciais.

    O custo referente ao implicante primo eliminado deve ser armazenado para efeito da determinação do custo total de realização da função.

  - 1.4 - Elimine, se for o caso, as colunas que só possuem coeficientes de restrições iguais a zero;

  - 1.5 - Insira, à direita do Tableau, tantas colunas (variáveis de folga) quantas forem as restrições. Em cada uma dessas colunas é atribuído um coeficiente igual a -1, de modo que cada linha tenha apenas um coeficiente igual a -1. Os coeficientes de custo da função objetivo para estas colunas adicionais são iguais a 0.

    O Tableau resultante está na forma padrão para a aplicação do Plex. É importante frisar que os problemas de simplificação de funções booleanas que dão origem a mapas cíiclicos no método de Quine-McCluskey são tratados nesta fase.

- 2. Obtenção de uma base inicial.

  - 2.1 - Selecione na segunda coluna, de cima para baixo, um coeficiente igual a 1. Esse elemento é denominado pivô. Pivoteie neste elemento.

    O pivoteamento (eliminação de Gauss-Jordan) consiste em efetuar operações elementares, com as linhas, para obter uma coluna, contendo um único coeficiente igual a 1 (elemento pivô).

    É importante frisar que para aumentar a eficiência desta operação os coeficientes são armazenados como variáveis inteiras, e quando necessário é realizada uma operação de mínimo múltiplo comum para evitar a divisão de dois números inteiros que poderia resultar em um coeficiente não inteiro.

- 2.2 - Procure na próxima coluna, à direita, um outro elemento pivô. Esse elemento deve estar numa linha que ainda não tem pivô. Pivoteie nesse elemento. Se a coluna em questão não tiver elemento pivô, avance para a próxima coluna;
- 2.3 - O passo 2.2 é repetido até que todas as linhas tenham um elemento pivô.

- 3. Verificação da factibilidade da solução base inicial obtida.

  Se a base inicial obtida for factível, siga para o passo 5, caso contrário, siga para o passo 4.

  Uma base inicial é factível, se todos os coeficientes da primeira coluna (b) forem maior ou igual a 0. Observe que não se considera nesta etapa as restrições de integralidade.

- 4. Obtenção de uma base inicial factível, se a base obtida não o for.

  - 4.1 - Uma coluna adicional (variável artificial) é inserida à direita do Tableau, com coeficiente -1 para cada restrição com $b \leq 0$. Os coeficientes de custo do Tableau são armazenados, e são substituídos por um coeficiente de custo artificial, igual a 1 na coluna referente à variável artificial e por coeficientes nulos nas demais colunas.
  - 4.2 - Pivoteie na coluna referente à variável artificial e na linha (restrição) com o coeficiente b mais negativo.
  - 4.3 - Procure pelo coeficiente de custo mais negativo (variável candidata a entrar na base);
  - 4.4 - Pivoteie na coluna referente à variável a entrar na base e na linha referente à menor relação (positiva ou nula) entre os coeficientes do vetor b e os coeficientes da coluna da variável.
  - 4.5 - Se houver pelo menos um coeficiente de custo menor que zero volte ao passo 4.3.
  - 4.6 - Se o valor da função objetivo é zero e a variável artificial é zero siga para o passo 4.7, caso contrário pare. O problema original não tem solução básica factível.
  - 4.7 - Elimine a coluna referente à variável artificial. Os coeficientes de custo do problema artificial são substituídos pelos coeficientes de custo armazenados no passo 4.1.
  - 4.8 - Se os coeficientes de custo relativo as variáveis da base não forem nulos zere-os através de pivoteamento.

- 5. Minimização da função objetivo.

  - 5.1 - Se todos os coeficientes de custo têm valor maior ou igual a zero, siga para o passo 5.5, caso contrário siga para o passo 5.2;
  - 5.2 - Procure pelo coeficiente de custo mais negativo (variável candidata a entrar na base).
  - 5.3 - Pivoteie na coluna referente à variável a entrar na base e na linha referente à menor relação (positiva ou nula) entre os coeficientes do vetor b e os coeficientes da coluna da variável.
  - 5.4 - Se houver pelo menos um coeficiente de custo menor que zero, volte para o passo 5.2.

- 5.5 - A solução do problema relaxado é obtida, fixando-se as variáveis fora da base com o valor zero; assim, as variáveis, pertencentes à base, serão iguais aos coeficientes do vetor b.

- 6. Verificação da factibilidade integral da solução ótima obtida.
  - 6.1 - Se a solução básica ótima for composta de zeros e uns, pare, caso contrário, siga para o passo 7.

- 7. Obtenção de uma solução básica ótima inteira.
  - 7.1 - Selecione, de cima para baixo, uma linha cujo coeficiente b é não inteiro. Esta linha gera uma nova restrição (inequação de $\geq$) denominada de corte, composta pelas partes fracionárias dos coeficientes das variáveis não básicas e do elemento b desta linha.
  - 7.2 - Transforme esta inequação numa equação de modo que a variável de folga seja um componente para a nova base. Insira esta equação no tableau e aplique o dual simplex.
    - 7.2.1 - Procure por uma linha cujo componente b é o mais negativo (variável a sair da base);
    - 7.2.2 - Pivoteia na linha referente à variável a sair da base e na coluna referente à menor relação entre os coeficientes do vetor custo relativo e os coeficientes (negativos) da linha da variável a sair da base;
    - 7.2.3 - Se houver algum componente do vetor b negativo volte ao passo 7.2.1;
  - 7.3 - Se o vetor b ainda tem componente não inteiro retorne ao passo 7.1, caso contrário avance para o passo 7.4.
  - 7.4 - Adicione, ao custo obtido, o armazenado no passo 1.3.

Apêndice D

Descrição detalhada do algoritmo MultPlex

Neste apêndice descreve-se o algoritmo MultPlex, que minimiza funções booleanas com múltiplas saídas utilizando-se programação linear inteira 0 e 1.

Para a fase de geração dos implicantes primos, utilizou-se um método tabular, estendido para considerar funções com múltiplas saídas.

- 1. Geração dos implicantes primos pelo método tabular

  Para a geração dos implicantes primos sigulares e implicantes primos múltiplos, os mintermos e os irrelevantes são combinados do mesmo modo que no método de Quine-McCluskey, porém:

  - 1.1 Só podem ser combinados mintermos ou irrelevantes comuns (assinalados para as mesmas funções de saída);
  - 1.2 Quando dois mintermos ou irrelevantes combinarem-se, os bits do rótulo do implicante resultante terá o dígito 1, somente nas posições onde ambos os rótulos dos mintermos ou irrelevantes combinados têm 1.
  - 1.3 Marcar um implicante como tendo sido combinado, somente se seu rótulo está contido no rótulo do implicante resultante da combinação.

- 2 - Montagem do Tableau

  - 2.1 - A primeira linha do Tableau representa a função-objetivo e é ordenada de forma crescente, da esquerda para a direta, de acordo com os coeficientes de custo. As variáveis (implicantes primos singulares e implicantes primos múltiplos) da função-objetivo são obtidas pelo tradicional método tabular para a geração de implicantes primos, estendidos para considerar funções booleanas com múltiplas saídas.

    As outras linhas da Tableau representam restrições (mintermos) do problema.

    A intersecção de uma linha (mintermo) com uma coluna (implicante primo) tem valor igual a 1, se o implicante primo cobre o mintermo. Nas demais intersecções é atribuído o coeficiente 0. Tem-se tantas restrições de cobertura quanto forem os mintermos de todas as funções de saídas.
  - 2.2 - Uma coluna adicional é inserida à esquerda do Tableau, representando o vetor b (recurso). Essa coluna tem, na primeira linha, o coeficiente igual a 0 e, nas demais linhas, coficiente igual a 1 (cobertura de cada mintermo por, pelo menos, 1 implicante primo).
  - 2.3 - Procure, a partir da segunda coluna, linhas, contendo um único coeficiente, igual a 1. O implicante primo, correspondente a essa coluna, é um implicante primo essencial, podendo ser eliminado do Tableau. Elimine também todas as linhas (mintermos), contendo 1 nessa coluna. Este passo não é necessário, se, na montagem inicial do tableau, não foram considerados os implicantes primos essenciais.

    O custo, referente ao implicante primo eliminado, deve ser armazenado, para efeito da determinação do custo total de realização da função.
  - 2.4 - Elimine, se for o caso, as colunas que só possuem coeficientes de restrições iguais a zero;
  - 2.5 - Insira, à direita do Tableau, tantas colunas (variáveis de folga) quantas forem as restrições. Em cada uma dessas colunas é atribuído um coeficiente igual a -1, de modo que cada linha tenha apenas um coeficiente igual a -1. Os coeficientes de custo da função-objetivo, para estas colunas adicionais, são iguais a 0.

O Tableau resultante está na forma padrão, para a aplicação do Plex. É importante frisar que os problemas de simplificação de funções booleanas, que dão origem a mapas ciclicos, no método de Quine-McCluskey são tratados nesta fase.

- 3. Obtenção de uma base inicial.

  - 3.1 - Selecione na segunda coluna, de cima para baixo, um coeficiente igual a 1. Esse elemento é denominado pivô. Pivoteie neste elemento.

    O pivoteamento (eliminação de Gauss-Jordan) consiste em efetuar operações elementares, com as linhas, para obter uma coluna, contendo um único coeficiente igual a 1 (elemento pivô). É importante frisar que, para aumentar a eficiência desta operação. os coeficientes são armazenados como variáveis inteiras, e, quando necessário, é realizada uma operação de mínimo múltiplo comum, para evitar a divisão de dois números inteiros, que poderia resultar em um coeficiente não inteiro.
  - 3.2 - Procure na próxima coluna, à direita, um outro elemento pivô. Esse elemento deve estar numa linha que ainda não tenha pivô. Pivoteie nesse elemento.

    Se a coluna em questão não tiver elemento pivô, avance para a próxima coluna;
  - 3.3 - O passo 3.2 é repetido, até que todas as linhas tenham um elemento pivô.

- 4. Verificação da factibilidade da solução base inicial obtida.

  Se a base inicial obtida, for factível, siga para o passo 6, caso contrário, siga para o passo 5.

  Uma base inicial é factível, se todos os coeficientes da primeira coluna (b) forem maior ou igual a 0. Observe que não se consideram nesta etapa as restrições de integralidade.

- 5. Obtenção de uma base inicial factível, se a base obtida não o for.

  - 5.1 - Uma coluna adicional (variável artificial) é inserida, à direita do Tableau, com coeficiente -1, para cada restrição com $b \leq 0$. Os coeficientes de custo do Tableau são armazenados e substituídos por um coeficiente de custo artificial, igual a 1, na coluna referente à variável artificial e, por coeficientes nulos, nas demais colunas.
  - 5.2 - Pivoteie na coluna referente à variável artificial e na linha (restrição) com o coeficiente b mais negativo.
  - 5.3 - Procure pelo coeficiente de custo mais negativo (variável candidata a entrar na base);
  - 5.4 - Pivoteie na coluna referente à variável a entrar na base e na linha referente à menor relação (positiva ou nula), entre os coeficientes do vetor b e os da coluna da variável.
  - 5.5 - Se houver pelo menos um coeficiente de custo menor que zero, volte ao passo 5.3.
  - 5.6 - Se o valor da função-objetivo é zero e a variável artificial é zero, siga para o passo 5.7, caso contrário, pare. O problema original não tem solução básica factível.
  - 5.7 - Elimine a coluna, referente à variável artificial. Os coeficientes de custo do problema artificial são substituídos pelos de custo armazenados no passo 5.1.

- 5.8 - Se os coeficientes de custo, relativos às variáveis da base, não forem nulos, zere-os. através de pivoteamento.

- 6. Minimização da função objetivo.

  - 6.1 - Se todos os coeficientes de custo têm valor maior ou igual a zero, siga para o passo 6.5, caso contrário, siga para o passo 6.2;
  - 6.2 - Procure pelo coeficiente de custo mais negativo (variável candidata a entrar na base).
  - 6.3 - Pivoteie na coluna, referente à variável a entrar na base e na linha referente à menor relação (positiva ou nula) entre os coeficientes do vetor b e os da coluna da variável.
  - 6.4 - Se houver pelo menos um coeficiente de custo menor que zero, volte para o passo 6.2.
  - 6.5 - A solução do problema relaxado é obtida, fixando-se as variáveis fora da base com o valor zero; assim, as variáveis, pertencentes à base, serão iguais aos coeficientes do vetor b.

- 7. Verificação da factibilidade integral da solução ótima obtida.

  - 7.1 - Se a solução básica ótima for composta de zeros e uns, pare, caso contrário, siga para o passo 8.

- 8. Obtenção de uma solução básica ótima inteira.

  - 8.1 - Selecione, de cima para baixo, uma linha cujo coeficiente b é não inteiro. Esta linha gera uma nova restrição (inequação de $\geq$), denominada de corte, composta pelas partes fracionárias dos coeficientes das variáveis não básicas e do elemento b desta linha.
  - 8.2 - Transforme esta inequação numa equação, de modo que a variável de folga seja um componente para a nova base. Insira esta equação no tableau e aplique o dual simplex.
    - 8.2.1 - Procure uma linha, cujo componente b é o mais negativo (variável a sair da base);
    - 8.2.2 - Pivoteia na linha, referente à variável a sair da base e na coluna, referente à menor relação entre os coeficientes do vetor custo relativo e os coeficientes (negativos) da linha da variável a sair da base;
    - 8.2.3 - Se houver algum componente do vetor b negativo, volte ao passo 8.2.1;
  - 8.3 - Se o vetor b ainda tenha componente não inteiro, retorne ao passo 8.1, caso contrário, avance para o passo 9.

- 9. Geração das coberturas singulares para todas as funções de saída.

  - 9.1 - Selecione uma função de saída para gerar a cobertura singular;

- 9.2 - A função-objetivo é dada pela soma ponderada dos implicantes primos, solução da cobertura múltipla, obtidos no passo 5.3.7, cujo rótulo contém 1 na posição corresponde à função em questão;
- 9.3 - Gere uma desigualdade do tipo qeg 1 para cada mintermo da função;
- 9.4 - Se todas as funções de saída tiverem cobertura singular siga para o passo 10, caso contrário, retorne ao passo 9.1;

- 10. Execute os passo 2 até 8 para cada uma das formulações obtidas, no passo anterior.

Apêndice E

Descrição detalhada do algoritmo ReduPlex

Neste apêndice descreve-se detalhadamente o algorimto ReduPlex, utilizado para reduzir máquinas de estados finitos completa ou incompletamente especificada.

A seleção do conjunto mínimo de classes primas que definem as máquinas reduzidas, é obtido resolvendo um problema de programação linear inteira 0 e 1.

- R1- Monte a Tabela de Condições de Compatibilidade.

  Nas linhas da coluna mais a esquerda da tabela são listados, de cima para baixo, os estados da tabela de estados com exceção do primeiro estado;

  Nas colunas da última linha da tabela, a partir da segunda coluna, são listados, da esquerda para a direita, todos os estados da tabela de transição com exceção do último estado.

  Mantenha na tabela somente as células, formadas por pares de estados, pertencentes a matriz triangular inferior pois a tabela deve conter somente uma célula para cada par de estados.

  Nas células cujos estados são compatíveis insira um " * " e nas células incompatíveis insira um " X ", caso contrário insira na célula as condições para que os estados sejam compatíveis.

- R2 - Obtenha as condições de compatibilidade entre pares de estados (tabela consistente).

  Caminhe no sentido da direita para a esquerda verificando se as colunas contêm células cujas condições de compatibilidade são inconsistentes. Nas células contendo condições de compatibilidade inconsistentes insira um " X ". A cada alteração na tabela retorne ao seu início à direita.

- R3 - Obtenha as classes máximas de compatibilidade.

  Inicie na coluna mais a direita da tabela de condições de compatibilidade, deslocando-se para a esquerda até encontrar uma coluna contendo algum par compatível. Liste todos os pares compatíveis desta coluna.

  Continue analisando a próxima coluna contendo ao menos um par compatível. Se o estado desta coluna for compatível com todos os membros de alguma classe de compatibilidade adicione este estado a tal classe de compatibilidade. Se o estado for compatível com somente parte de uma classe de compatibilidade, forme uma nova classe de compatibilidade que inclua esta parte e o estado em questão.

  Crie novas classes de compatibilidade com os pares não incluídos em nenhuma outra classe de compatibilidade.

  Após analisar uma coluna, avance para a próxima até que todas as colunas sejam consideradas.

  Os conjunto final de classes de compatibilidade são as classes máximas de compatibilidade.

  Verifique se as classes máximas de compatibilidade são disjuntas. Em caso afirmativo o conjunto de classes máximas de compatibilidade forma uma cobertura mínima para a tabela de estados, caso contrário avance para o passo seguinte.

- R4 - Obtenha as classes primas de compatibilidade a partir das classes máximas de compatibilidade.

  São primas todas as classes máximas de compatibilidade cujo conjunto de condições é vazio. Para a obtenção das demais classes primas, decomponha as classes máximas de compatibilidade, cujo conjunto de condições não é vazio em subconjuntos unitários. Esses conjuntos unitários também são primos e devem ser considerados na formulação do problema de programação matemática.

Na implementação computacional do algoritmo optou-se pela decomposição das classes máximas somente em subconjuntos unitários, pois esta decomposição contorna o problema da obtenção de todos os subconjuntos das classes máximas, sem prejudicar o propósito desse trabalho que é a formulação do problema de cobertura e fechamento como um problema de programação matemática.

- R5 - Formule o problema de programação matemática a partir do conjunto de classes primas.

  Minimize a função objetivo com custo relativo de cada classe prima igual a 1 e com as restrições de cobertura e de fechamento.

  Restrições de cobertura: Para cada estado da tabela de transição gere uma desigualdade maior ou igual a 1, cujo lado esquerdo é a soma algébrica das variáveis que representam as classes primas contendo o estado. Deste modo, tem-se tantas restrições de cobertura quantos forem os estados da tabela de estados.

  Restrições de fechamento (Consistência das condições de compatibilidade): Para cada par de condição de compatibilidade gere uma desigualdade maior ou igual, cujo lado direito representa uma classe prima de compatibilidade e o lado esquerdo corresponde a soma algébrica das classes primas que contém o par de condições de compatibilidade para a classe prima em questão. Deste modo, tem-se tantas restrições de fechamento quanto forem as condições de compatibilidade.

  Formulado o problema de programação matemática, elimine as restrições de fechamento redundantes e resolva o problema de programação matemática como segue (Veja apêndice para as bases matemáticas que suportam os próximos passos do algoritmo).

  - I1 - Montagem do Tableau
    - I1.1 - A primeira linha do Tableau representa a função objetivo e é ordenada de forma crescente, da esquerda para a direta, de acordo com os coeficientes de custo, que para os casos de redução de estados assumem valor 1. As variáveis (classes primos) da função objetivo são obtidas no passo R4. As outras linhas do Tableau representam as restrições de cobertura e fechamento.
      A interseção de uma linha (condições de cobertura e fechamento) com uma coluna (classes prima) tem valor igual a 1 se a classe prima esta presente na restrição. Nas demais intersecções é atribuído o coeficiente 0.
    - I1.2 - Uma coluna adicional é inserida à esquerda do Tableau, representando o vetor b (recurso). Essa coluna tem, na primeira linha, o coeficiente igual a 0, nas linhas que representam restrições de cobertura têm coeficientes iguais a 1 e nas linhas que representam restrições de fechamento têm coeficientes iguais a 0.
    - I1.3 - Procure, a partir da segunda coluna, por linhas contendo um único coeficiente igual a 1. A classe prima, correspondente a essa coluna, é uma classe prima essencial, podendo ser eliminado do Tableau. Elimine também todas as linhas, contendo 1 nessa coluna. Este passo não é necessário se na montagem inicial do tableau não foram consideradas as classes primas esseciais.
      O custo referente a classe prima eliminada deve ser armazenado para efeito da determinação do custo total de realização da função.

  * I1.4 - Elimine, se for o caso, as colunas que só possuem coeficientes de restrições iguais a zero;
  * I1.5 - Insira, à direita do Tableau, tantas colunas (variáveis de folga) quantas forem as restrições. Em cada uma dessas colunas é atribuído um coeficiente igual a -1. de modo que cada linha tenha apenas um coeficiente igual a -1. Os coeficientes de custo da função objetivo para estas colunas adicionais são iguais a 0. O Tableau resultante está na forma padrão para a aplicação do Intplex.

- I2. Obtenção de uma base inicial.
  * I2.1 - Selecione na segunda coluna, de cima para baixo, um coeficiente igual a 1. Esse elemento é denominado pivô. Pivoteie neste elemento.
    O pivoteamento (eliminação de Gauss-Jordan) consiste em efetuar operações elementares, com as linhas, para obter uma coluna, contendo um único coeficiente igual a 1 (elemento pivô).
    É importante frisar que para aumentar a eficiência desta operação os coeficientes são armazenados como variáveis inteiras, e quando necessário é realizada uma operação de mínimo múltiplo comum para evitar a divisão de dois números inteiros que poderia resultar em um coeficiente não inteiro.
  * I2.2 - Procure na próxima coluna, à direita, um outro elemento pivô. Esse elemento deve estar numa linha que ainda não tem pivô. Pivoteie nesse elemento.
    Se a coluna em questão não tiver elemento pivô, avance para a próxima coluna;
  * I2.3 - O passo I2.2 é repetido até que todas as linhas tenham um elemento pivô.

- I3. Verificação da factibilidade da solução base inicial obtida.
  Se a base inicial obtida for factível, siga para o passo I5, caso contrário, siga para o passo I4.
  Uma base inicial é factível, se todos os coeficientes da primeira coluna (b) forem maior ou igual a 0. Observe que não se considera nesta etapa as restrições de integralidade.

- I4. Obtenção de uma base inicial factível, se a base obtida não o for.
  * I4.1 - Uma coluna adicional (variável artificial) é inserida à direita do Tableau, com coeficiente -1 para cada restrição com $b < 0$.
    Os coeficientes de custo do Tableau são armazenados, e são substituídos por um coeficiente de custo artificial, igual a 1 na coluna referente à variável artificial e por coeficientes nulos nas demais colunas.
  * I4.2 - Pivoteie na coluna referente à variável artificial e na linha (restrição) com o coeficiente b mais negativo.
  * I4.3 - Procure pelo coeficiente de custo mais negativo (variável candidata a entrar na base);
  * I4.4 - Pivoteie na coluna referente à variável a entrar na base e na linha referente à menor relação (positiva ou nula) entre os coeficientes do vetor b e os coeficientes da coluna da variável.
  * I4.5 - Se houver pelo menos um coeficiente de custo menor que zero volte ao passo I4.3.

  * I4.6 - Se o valor da função objetivo é zero e a variável artificial é zero siga para o passo I4.7, caso contrário pare. O problema original não tem solução básica factível.
  * I4.7 - Elimine a coluna referente à variável artificial. Os coeficientes de custo do problema artificial são substituídos pelos coeficientes de custo armazenados no passo I4.1.
  * I4.8 - Se os coeficientes de custo relativo as variáveis da base não forem nulos zere-os através de pivoteamento.

- I5. Minimização da função objetivo.
  * I5.1 - Se todos os coeficientes de custo têm valor maior ou igual a zero, siga para o passo I5.5, caso contrário siga para o passo I5.2;
  * I5.2 - Procure pelo coeficiente de custo mais negativo (variável candidata a entrar na base).
  * I5.3 - Pivoteie na coluna referente à variável a entrar na base e na linha referente à menor relação (positiva ou nula) entre os coeficientes do vetor b e os coeficientes da coluna da variável.
  * I5.4 - Se houver pelo menos um coeficiente de custo menor que zero, volte para o passo I5.2.
  * I5.5 - A solução do problema relaxado é obtida, fixando-se as variáveis fora da base com o valor zero; assim, as variáveis, pertencentes à base, serão iguais aos coeficientes do vetor b.

- I6. Verificação da factibilidade integral da solução ótima obtida.

  Se a solução básica ótima for composta de zeros e uns, pare, caso contrário, siga para o passo I7.

- I7. Obtenção de uma solução básica ótima inteira.
  * I7.1 - Selecione, de cima para baixo, uma linha cujo coeficiente b é não inteiro. Esta linha gera uma nova restrição (inequação de $\geq$) denominada de corte, composta pelas partes fracionárias dos coeficientes das variáveis não básicas e do elemento b desta linha.
  * I7.2 - Transforme esta inequação numa equação de modo que a variável de folga seja um componente para a nova base.

    Insira esta equação no tableau e aplique o dual simplex.
    - I7.2.1 - Procure por uma linha cujo componente b é o mais negativo (variável a sair da base);
    - I7.2.2 - Pivoteia na linha referente à variável a sair da base e na coluna referente à menor relação entre os coeficientes do vetor custo relativo e os coeficientes (negativos) da linha da variável a sair da base;
    - I7.2.3 - Se houver algum componente do vetor b negativo volte ao passo I7.2.1;
  * I7.3 - Se o vetor b ainda tem componente não inteiro retorne ao passo I7.1, caso contrário avance para o passo I7.4.
  * I7.4 - Adicione, ao custo obtido, o armazenado no passo I1.3.

- R6 - Crie uma máquina reduzida a partir das classes máximas de compatibilidade obtidas pela resolução do problema linear formulado no passo anterior.

  - R6.1. Selecione uma classe de compatibilidade pertencente a solução mínima do problema;
  - R6.2. Verifique para cada entrada na tabela original quais os próximos estados e a saída dos estados contidos na classe de compatibilidade selecionada no passo R6.1;
  - R6.3. A transição para cada entrada, na tabela reduzida, é feita da classe de compatibilidade selecionada no passo R6.1 para a classe de compatibilidade que contém os estados obtidos no passo R6.2.

    Se mais de uma classe de compatibilidade contém estes estados, escolha uma arbitrariamente.

    Se para todos os estados da classe de compatibilidade em questão os próximos estados forem não especificados então a máquina reduzida tem próximo estado não especificado para esta transição;
  - R6.4. A saída para a transição é especificada de acordo com a saída da tabela original, isto é, se todas as saídas são não especificadas esta transição tem saída não especificada na tabela reduzida.
  - R6.5. Se todas as classes de compatibilidades foram analisadas pare, caso contrário retorne ao passo R6.1 (selecione outra classe de compatibilidade).

Bibliografia

[1] Darringer J. A., Joyner W. H., Berman C. L., Trevillyan L., "Logic Synthesis Through Local Transformations", IBM J. Res. Develop., vol. 25, no. 4, July, 1981, p. 272.

[2] Lipp H. M., "Methodical Aspects of Logic Synthesis", Proceedings of the IEEE, vol. 71, no. 1, January, 1983, p. 88.

[3] Gilkinson J. L., Lewis S. D., Winter B. B., Hekmatpour A., "Automated Technology Mapping", IBM J. Res. Develop., vol. 28, n0. 5, September, 1984, p. 546.

[4] Darringer J. A., Brand D., Gerbi J. V., Trevillyan L., "LSS: A System for Production Logic Synthesis", IBM J. Res. Develop., vol. 28, no. 5, September, 1984, p. 537.

[5] Okuda N., Sugai M., Goto N., "Semicustom and Custom LSI Technology", Proceedings of the IEEE, vol. 74, n0. 12, December, 1986, p. 1636.

[6] Weber T. S., Weber R. F., Jansch-Pôrto I., Wagner F. R., "Métodos de Validação de Sistemas Digitais", VI Escola de Computação, 1988.

[7] Cavin R. K., Hilbert J. L., "Design of Integrated Circuits: Directions and Challenges", Proceedings of the IEEE, vol. 78, no. 2, February, 1990, p. 418.

[8] Theeuwen F., "Automatic Logic Synthesis", Proceedings of I Brazilian Microelectronic School, March, 1990, p. 61.

[9] Personal learning program, version 7.0, "Introduction to Logic Synthesis with AutoLogic and AutoLogic Blocks", Mentor Graphics, 1991.

[10] ALTERA, "Data Book", 1991.

[11] XILINX, "The Programmable Gate Array Data Book", 1991.

[12] Composano R., "High-Level Synthesis", Proceedings of II Brazilian Microelectronic School, March, 1992, p. 93.

[13] Akers S. B., "A Truth Table Method for the Synthesis of Combinational Logic", IRE Trans. on Electronic Computers, December, 1961, p. 601.

[14] Das S. R., "Comments on Á New Algorithm for Generating Prime Implicants", IEEE Trans. Computers, vol. C-20, December, 1971, p.1614.

[15] Hwa H. R., "A Method for Generating Prime Implicants of Boolean Expression", IEEE Trans. on Computers, June, 1974, p. 637.

[16] Sureshchander "Minimization of Switching Functions - A Fast Technique", IEEE Trans. on Computers, July, 1975, p.753.

[17] Hulme B. L., Worrell R. B., "A Prime Implicant Algorithm with Factoring", IEEE Trans. on Computers, November, 1975, p. 1129.

[18] Mendelson E. "Álgebra Booleana e Circuitos de Chaveamento", McGraw-Hill, 1977.

[19] Brayton K. B., Hachtel G. D., McMullen C. T., Sangiovanni-Vincentelli A. L., "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

[20] McCluskey, E.J., "Logic Design Principles; with Emphasis on Testable Semicustom Circuits", Prentice-Hall, N.J., 1986.

[21] Fleisher H., Giraldi J., Phoenix R., Tavel M., "Minimizability of Random Boolean Functions", IEEE Trans. on Computers, vol. 38, April, 1989, p. 593.

[22] Silva A. C. R., "Contribuição à Minimização e Simulação de Circuitos Lógico", Tese de Mestrado - FEE - UNICAMP, Novembro, 1989.

[23] Kabakçioğlu A. M., Varshney P. K., Hartmann C. R. P., "Application of Information Theory to Switching Function Minimisation", IEE Proceedings-E, vol. 137, no. 1, September, 1990, p. 389.

[24] Poswig J., "Disjoint Decomposition of Boolean Functions", IEE Proceedings-E, vol. 138, no. 1, January, 1991, p. 48.

[25] Silva A. C. R., Madureira M. C., Bonatti I. S., "Cobertura não Redundante de funções Lógicas através da Geração de Padrões de Teste de Falhas", Revista Brasileira de Computação, vol. 6, no. 3, janeiro/março, 1991, p. 13.

[26] Varma D., Trachtenberg E. A., "Computation of Reed-Muller Expansions of Incompletely Specified Boolean Functions from Reduced Representations", IEE Proceedings-E, vol. 138, no. 2 March, 1991, p. 85.

[27] Caruso G., "Near Optimal Factorization of Boolean Functions", IEEE Trans. on Computer-Aided Design, vol. 10, no. 8, August, 1991, p. 1072.

[28] Ghosh, A., Devadas S., Newton A. R., "Heuristic Minimization of Boolean Relations Using Testing Techniques", IEEE Trans. on Computer-Aided Design, vol. 11, no. 9, September, 1992, p. 1166.

[29] Falkowski B. J., Schäfer I., Perkowski M. A., "Effective Computer Methods for the Calculation of Rademacher-Walsh Spectrum for Completely and Incompletely Specified Boolean Functions", IEEE Trans. on Computer-Aided Design, vol. 11, no. 10, October, 1992, p. 1207.

[30] Quine W. V., "The Problem of Simplifying Truth Functions", Amer. Math. Mon., vol. 59, October, 1952, p. 521.

[31] R. J. Nelson, "Simplest Normal Truth Function", J. Symbolic. Logic, vol. 20, June, 1954, pp. 105.

[32] Quine W. V., "A Way to Simplify Truth Functions", Amer. Math. Mon., vol. 62, November, 1955, p. 627.

[33] Scheinman A. H., "A Method for Simplifying Boolean Functions", The Bell System Technical Journal, July, 1962, pp. 1336.

[34] Das S. R., Choudhury A. K., "Maxterm Type Expressions of Switching Functions and their Prime Implicants", IEEE Trans. Electronic Computers, vol. EC-14, December, 1965, pp. 920.

[35] Tison P., "Generalization of Consensus Theory and Application to the Minimization of Boolean Function", IEEE Trans. Electronic Computer, vol. EC-16, August, 1967, p. 446.

[36] Slagle J. R., Chang C. L., Lee R. C. T., "A New Algorithm for Generating Prime Implicants", IEEE Trans. Computers, vol. C-19, April, 1970, pp. 304.

[37] Hulme B. L., Worrel R. B., "A Prime Implicant Algorithm with Factoring", IEEE Trans. Computers, November, 1975, pp. 1129.

[38] Mendelson E. "Álgebra Booleana e Circuitos de Chaveamento", McGraw-Hill, 1977.

[39] Brayton K. B., Hachtel G. D., McMullen C. T., Sangiovanni-Vincentelli A. L., "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

[40] Karplus K., "Representing Boolean Functions with IF-Then-Else DAGs", UCSC-CRL-88-28, University of California, November, 1988.

[41] Chakravarty S., "A Characterization of Binary Decision Diagrams", IEEE Trans. on Computers, vol. 42, February, 1993, pp. 129.

[42] Lawler E. L., "Note 'Minimization of Switching Circuits Subject to Reliability Conditions'", IRE Trans. on Electronic Computers, EC-10, December, 1961, p. 781.

[43] Einhorn S. N., "The Use of the Simplex Algorithm in the Mechanization of Boolean Switching Function by Means of Magnetic Cores", IRE Trans. on Electronic Computers, EC-10, December, 1961, p. 615.

[44] Minnick R. C., "Comments on 'The Use of the Simplex Algorithm in the Mechanization of Boolean Switching Functions by Means of Magnetic Cores'", IRE Trans. on Electronic Computers, EC-11, August, 1962, p. 573.

[45] Pyne I. B., McCluskey E. J. Jr, "The Reduction of Redundancy in Solving Prime Implicant Tables", IRE Trans. on Electronic Computers, EC-11, August, 1962, p. 473.

[46] Gimpel J. F., "A Reduction Technique for Prime Implicant Tables", IEEE Trans. on Electronic Computers, EC-14, August, 1965, p. 535.

[47] Muroga S., Ibaraki T., "Design of Optimal Switching Networks by Integer Programming". IEEE Trans. on Computers, vol. C-21, no. 6, June, 1972, p. 573.

[48] Bubenik V., "Weighting Method for the Determination of the Irredundant Set of Prime Implicants", IEEE Trans. on Computers, vol. C-21, December, 1972, p. 1449.

[49] Noe P. S., Rhyne V. T., Suraratrungsi S., "Comments on 'Weighting Method for the Determination of the Irredundant Set of Prime Implicants'", IEEE Trans. on Computers, June. 1974, p. 646.

[50] Trevillyan L. H., Joyner W., Berman C. L, "Global Flow Analysis in Automatic Logic Design", IEEE Trans. on Computers, vol. C-35, no. 1, January, 1986, p. 77.

[51] Butler J. T., Schueller K. A., "On the Equivalence of Cost Functions in the Design of Circuits by Costtable", IEEE Trans. on Computers, vol. 39, no. 6, June, 1990, p. 842.

[52] Sung Je Hong e Saburo Muroga, "Absolute Minimization of Completely Specified Switching Functions", IEEE Trans. on Computers, vol. 40, no. 1, January, 1991, p. 53.

[53] Malik A. A., Brayton R. K., Newton A. R., Sangiovanni-Vicentelli A., "Reduced Offsets for Minimization of Binary-Valued Functions", IEEE Trans. on Computer-Aided Design, vol. 10, no. 4, April, 1991, p. 413.

[54] Berman C. L., Trevillyan L. H., "Global Flow Optimization in Automatic Logic Design", IEEE Trans. on Computer-Aided Design, vol. 10, no. 5, May, 1991, p. 557.

[55] Mileto F., Putzolu G., "Average Values of Quantifies Appearing in Multiple Output Boolean Minimization", IEEE Trans. on Electronic Computers, col. EC-14, August, 1965, p. 542.

[56] Morreale E., "Recursive Operator for Prime Implicant and Irredundant Normal Form Determination", IEEE Trans. on Computrs, vol. C-19, June, 1970, p. 504.

[57] Hong S. J., Cain R. G., Ostapko D. L., "MINI: A Heuristic Approach for Logic Minimization", IBM J. Res. Develop, September, 1974, p. 443.

[58] Reush B., "Generation of Prime Implicants from Subfunctions and a Unifying Approach to the covering Problem", IEEE Trans. on Computers, vol. C-24, September, 1975, p. 924.

[59] Rhyme T., Noe P. S., McKinney M. H., Pooch U. W., "A New Technique for the Fast Minimization of Switching Functions", vol. C-26, August, 1977, p.757.

[60] Arevalo Z., Bredeson J. G., "A Method to Simplify a Boolean Function in a Near Minimal Sum-of-Products for Programmable Logic Array", IEEE Trans. on Computers, vol. C-27, November, 1978, p. 1028.

[61] Frederick J. Hill e Gerald R. Peterson, "Introduction to Switching Theory and Logic Design", John Wiley & Sons, 1981.

[62] Brayton K. B., Hachtel G. D., McMullen C. T., Sangiovanni-Vincentelli A. L., "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, 1984.

[63] McCluskey, E.J. "Logic Design Principles; with Emphasis on Testable Semicustom Circuits". Prentice-Hall, N.J., 1986.

[64] Dagenais M. R., Agarwal V. K., Rumin N. C., "McBOOLE: A New Procedure for Exact Logic Minimization", IEEE Trans. on Computer-Aided Design, Vol. CAD-5, no. 1, January, 1986, p. 229.

[65] Bryant R. E., "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans on Computers, vol. C-35, no. 8, August, 1986, p. 677.

[66] McMullen C., Shearer J., "Prime Implicants, Minimun Cover, and the Complexity of Logic Simplification", IEEE Trans on Computers, vol. C-35, no. 8, August, 1986, p. 761.

[67] Karplus K., "Using if-then-else DAGs for Multi-Level Logic Minimization", UCSC-CRL-88-29, University of California, November, 1988.

[68] Perkins S. R., Rhyne T., "An Algorithm for Identifying and Selecting the Prime Implicants of a Multiple-Output Boolean Function", IEEE Trans. on Computer-Aided Design, vol. 7, no. 11, November, 1988, p. 1215.

[69] A. E. A. Almaini, "Eletronic Logic Systems", Prentice Hall International, 1989.

[70] Gurunath B., Biswas N. N., "An Algorithm for Multiple Output Minimization", IEEE Trans. on Computer-Aided Design, vol. 8, no. 9, September, 1989, p. 1007.

[71] Wey C-L., Chang T-Y., "An Efficient Output Phase Assignment for PLA Minimization", IEEE Trans. on Computer-Aided Design, vol. 9, no. 1, January, 1990, p. 1.

[72] Biswas N. N., "On Covering Distant Minterms by the Camp Algorithm", IEEE Trans. on Computer-Aided Design, vol. 9, no. 7, July, 1990, p. 786.

[73] Sasao T. "Bounds on the Average Number of Products in the Minimum Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functioins", IEEE Trans. on Computers, vol. 40, no. 5, May, 1991, p. 645.

[74] Berman C. L., Trevillyan L. H., "Global Flow Optimization in Automatic Logic Design", IEEE Trans. on Computer-Aided Design, vol. 10, no. 5, May, 1991, p. 557.

[75] Keogh D. B., "The State Diagram of HDB3", IEEE Trans. on Communications, vol. com-32, n0. 11, November, 1984, p. 1222.

[76] Huffman, D.A., "The Synthesis of Sequential Switching Circuits", J. Franklin Inst., 257, March, 1954, pp. 161.

[77] Mealy, G.H., "A Method for Synthesizing Sequential Circuits", The Bell System Technical Journal, September, 1955, pp. 1045.

[78] Ginsburg, S., "On the Reduction of Superfluous States in a Sequential Machine", Journal of the Association for Computing Machinery, vol. 6 no.2, April, 1959, pp. 259.

[79] Paull M.C., Unger, S.H., "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions", IRE Trans. on Electronic Computers, vol. EC-8, September, 1959, pp. 356.

[80] Narasimhan, R., "Minimizing Incompletely Specified Sequential Switching Functions", IRE Trans. on Electronic Computers, vol. EC-10, March, 1961, pp. 531.

[81] Weissman, J., "Boolean Algebra, Map Coloring, and Interconnections", American Mathematical Monthly, September, 1962, pp. 609.

[82] McCluskey, E. J. Jr., "Minimum-State Sequential Circuits for a Restricted Class of Incompletely Specified Flow Tables", The Bell System Technical Journal, November, 1962, pp. 1759.

[83] Marcus, M.P., "Derivation of Maximal Compatibles Using Boolean Algebra", IBM Journal of Research and Development, November, 1964, pp. 537.

[84] Grasselli, A., Luccio, F., "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks", IEEE Trans. on Electronic Computers, vol. EC-14, June, 1965, pp. 350.

[85] Unger, S.H., "Flow Table Simplification - Some Useful Aids", IEEE Trans. on Electronic Computers, vol. EC-14, June, 1965, pp. 472.

[86] Choudhury, A.K., Basu, A.K., Desarkar, S.C., "On the Determination of the Maximum Compatibility Classes", IEEE Trans. on Electronic Computers, July 1969, pp. 665.

[87] Brown, F.M., "Comment on The Determination of Maximum Compatibility Classes", IEEE Trans. on Electronic Comoputers, December, 1969, pp. 459.

[88] Curtis, H.A., "Systematic Procedures for Realizing Synchronous Sequential Machines Using Flip-Flop Memory: Part I", IEEE Trans. on Electronic Computers, vol. C-18, no. 12, December, 1969, pp. 1121.

[89] Curtis, H.A., "Systematic Procedures for Realizing Synchronous Sequential Machines Using Flip-Flop Memory: Part II", IEEE Trans. on Electronic Computers, vol. C-19, no. 1, January, 1970, pp. 66.

[90] Kella J., "State Minimization of Incompletely Specified Sequential Machines", IEEE Trans. on Electronic Computers, vol. C-19, no. 4, April, 1970, pp.342.

[91] House, R.W., Stevens, D.W., "A New Rule for Reducing CC Tables", IEEE Trans. on Electronic Computers, November, 1970, pp.1108.

[92] Bemmetts, R.G., Washington, J.L., Lewin D.W., "A Computer Algorithm for State Table Reduction", The Radio and Electronic Engineer, vol. 42, no. 11, November 1972, pp. 513.

[93] Kim J., Newborn M.M., "The Simplification of Sequential Machines with Inputs Restrictions", IEEE Trans. on Computers, December, 1972, pp. 1440.

[94] Yang, C., Babinski M.K., "Comments on Closure Partition Method for Minimizing Incomplete Sequential Machines", IEEE Trans. on Computers, January, 1975, pp. 106.

[95] Rubin, F., "Worst Case Bounds for Maximal Compatible Subsets", IEEE Trans. on Computers, August, 1975, pp. 830.

[96] Handoko F., "A discussion on two Algorithms for Determinimg Maximum Compatibles", IEEE Trans. on Computers, August, 1975, pp. 838.

[97] Yang, C., "On the Equivalence of Two Algorithms for Finding All Maximal Compatibles", IEEE Trans. on Computers, October, 1975, pp.977.

[98] Rao, C.V.S., Biswas, N.N., "Minimization of Incompletely Specified Sequential Machines", IEEE Trans. on Computers, vol. C-24, no.11, November, 1975, pp. 1089.

[99] Yang, C., Tarpy, M.A., "An Algorithm for Deriving All Pairs of Compatible States by Closure Classes", IEEE Trans. on Computers, February, 1976, pp. 202.

[100] Rao, C.V.S., Biswas, N.N., "Further Comments on Closure Partition Method for Minimizing Incompletely Sequential Machines", IEEE Trans. on Computers, July, 1976, pp. 767.

[101] Vink, H.A., Dolder, V.D, Al, J., "Reduction of CC-Table Using Multiple Implication", IEEE Trans. on Computers, vol. c-27, no.10, October, 1978, pp. 961.

[102] Kohavi, Zvi, "Switching and Finite Automata Theory", McGraw-Hill, 1978.

[103] Yamamoto, M., "A method for Minimizing Incompletely Specified Sequential Machines", IEEE Trans. on Computers, vol. c-29, no.8, August, 1980, pp. 732.

[104] Hill, Frederick J., Peterson, Gerald R., "Introduction to Switching Theory and Logic Design", John Wiley & Sons, 1981.

[105] McCluskey Edward J., "Logic Design Principles with Emphasis on Testable Semicustom Circuits", Printece Hall, 1986.

[106] Almaini, A. E. A., "Electronic Logic Systems", Prentice Hall, 1989.

[107] Ranerup, K., Philipson, L., "Optimization of Finite State Machines Using Subroutines", Research Report, Department of Computer Engineering, Lund University, Sweden.

[108] Devadas S, Ma H-K. T., Newton R., Sangiovanni-Vicentelli, A., " Irredundant Sequential Machines Via Optimal Logic Synthesis", IEEE Trans. on Computers-Aided Design, Vol. 9, No 1, January 1990, pp. 8.

[109] Devadas S., Ma H-K. T., "Easily Testable PLA-Based Finite State Machines", IEEE Trans. on Computer-Aided Design, Vol. 9, No. 6, June 1990, pp. 604.

[110] Drusinsky-Yoresh, D., "Symbolic Cover Minimization of Fully I/O Specified Finite State Machines", IEEE trans. on Computer-Aided Design, Vol. 9, No. 7, July 1990, pp. 779.

[111] Devadas, S., Keutzer, K., "A unified Approach to the Synthesis of Fully Testable Sequential Machines", IEEE Trans. on Computer-Aided Design, Vol. 10, No. 1, January 1991, pp. 39.

[112] Ashar, P., Devadas, S., Newton, A. R., "Irredundant Interactiong Sequential Machines Via Optimal Logic Synthesis", IEEE Trans. on Computer-Aided Design, vol. 10, No. 3 March 1991, pp. 311.

[113] Devadas, S., "Optimizing Interacting Finite State Machines Using Sequential Don't Cares", IEEE Trans. on Computers-Aided Design, vol. 10, No. 12, December 1991, pp. 1473.

[114] Mao W., Milne J. G., "An Automated Proof Technique for Finite-State Machine Equivalence", Research Report HDV-16-91, Department of Computer Science, University of Strathclyde, Scotland, May, 1991.

[115] McCaskill G. A. , Milne J. G., "Hardware Description and Verification Using the Circal-System", Research Report HDV-24-92, Department of Computer Science, University of Strathclyde, Scotland, June, 1992.

[116] McCaskill G. A. , Milne J. G., "Sequential Circuit Analysis with a BDD based Process Algebra System", Research Report HDV-25-93, Department of Computer Science, University of Strathclyde, Scotland, January, 1993.

[117] Mokhtar S. Bazaraa, John J. Jarvis, Hanif D. Sherali, "Linear Programming and Network Flows", John Wiley & Sons, 1990.

[118] Nemhauser, G. L., Wolsey, L. A., "Integer and Combinatorial Optimization", Wiley, 1988.

[119] Garfinkel, R. S., Nemhauser, G. L., "Integer Programming", Wiley, 1972.

[120] Salkin, H. M., "Integer Programming", Wesley, 1975.