

unesp



UNIVERSIDADE ESTADUAL PAULISTA

Pós-Graduação em Engenharia Elétrica

**Formulação Matemática do Problema de
Otimização de Funções Booleanas Através do
Método de Expansão de Shannon**

FÁBIO RICARDO PONTES EMER

1210001365



Ilha Solteira - SP

unesp



**UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA
PROGRAM DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

**“Formulação Matemática do Problema de
Otimização de Funções Booleanas Através
do Método de Expansão de Shannon”**

DISSERTAÇÃO SUBMETIDA À FACULDADE DE ENGENHARIA – CAMPUS
DE ILHA SOLTEIRA – UNESP COMO PARTE DOS REQUISITOS
NECESSÁRIOS PARA A OBTENÇÃO DO TÍTULO DE MESTRE EM
ENGENHARIA (ME)

Autor: Fábio Ricardo Pontes Emer

Orientador: Alexandre César Rodrigues da Silva

Ilha Solteira, outubro de 2002



FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação/Serviço Técnico de Biblioteca e documentação da FEIS/UNESP

E53f Emer, Fábio Ricardo Pontes
Formulação matemática do problema de otimização de funções booleanas através do método de expansão de Shannon / Fábio Ricardo Pontes Emer – Ilha Solteira: [s.n.], 2002
xii, 104 p. : il.

Dissertação (mestrado) – Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2002.

Orientador: Alexandre César Rodrigues da Silva
Bibliografia: p. 72-74

1. Função booleana – Otimização. 2. Álgebra booleana 3. Algoritmos. 4. Expansão de Shannon

Formulação Matemática do Problema de Otimização de Funções Booleanas Através do Método de Expansão de Shannon

FÁBIO RICARDO PONTES EMER

DISSERTAÇÃO SUBMETIDA À FACULDADE DE ENGENHARIA -
CAMPUS DE ILHA SOLTEIRA – UNESP – COMO PARTE DOS
REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO TÍTULO DE
MESTRE EM ENGENHARIA ELÉTRICA (ME).

A. A. Carvalho

Prof. Dr. Aparecido Augusto de Carvalho
Coordenador

COMISSÃO EXAMINADORA:

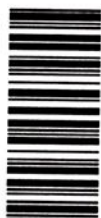
Alexandre César Rodrigues da Silva
Prof. Dr. Alexandre César Rodrigues da Silva – orientador

Evandro Mazina Martins
Prof. Dr. Evandro Mazina Martins

José Paulo Fernandes Garcia
Prof. Dr. José Paulo Fernandes Garcia

Ilha Solteira, outubro de 2002

1210001365



Proc. 053/2003-NPD 032/03

UNESP - "CAMPUS DE ILHA SOLTEIRA"	
SERVIÇO TÉCNICO DE BIBLIOTECA E DOCUMENTAÇÃO	
DATA DE CHEGADA	DATA DE TOMBAMENTO
12.03.03	31.03.03
AQUISIÇÃO	CLASSIFICAÇÃO
<i>Aieza</i> <i>Doações</i> <i>Antes</i> <i>R\$ 10.00</i>	<i>Te. 1365</i> <i>E53f</i>

10101004 ALGEBRA

sy - 186017
sy - 54767

*À minha mãe e a minha esposa,
pelo apoio e carinho.*



Agradecimentos

Agradeço a todos aqueles que contribuíram para realização desta dissertação. Em especial:

Prof. Dr. Alexandre César Rodrigues Silva pela orientação segura e criteriosa.

Funcionários da Secretaria da Pós-graduação, da Biblioteca, da Secretaria do Departamento de Engenharia Elétrica e do Laboratório de Computação pelo suporte técnico.

Professores da banca examinadora.

Direção e Gerência Financeira e Administrativa do Hospital Carlos Chagas pelo apoio.

Sr. João Baptista Pontes, meu avô e Sr. Namor Ribeiro, meu padrinho, pelos conselhos, dedicação e carinho.

Colegas do curso de Pós-graduação pelo companheirismo e troca de experiências.

Resumo

Este trabalho propõe a formulação do problema de otimização de funções booleanas como um problema matemático de programação linear inteiro 0 e 1.

Para tanto foi implementado em computador um algoritmo denominado **Expander**, baseado no Método de Expansão de Shannon, que ao gerar implicantes primos elimina em alguns casos aqueles não necessários a cobertura da função. Assim a formulação do problema é simplificada, podendo esta ser resolvido por métodos matemáticos clássicos, aproveitando-se ainda dos avanços nesta área.

Para avaliação do programa, este foi comparado ao Método de Quine-McCluskey, também implementado em computador em um programa chamado Pigene.

Ao se comparar memória, tempo de execução e quantidade de implicantes primos gerados, comprovou-se a eficiência do método que também respondeu de forma eficiente a funções com grande número de variáveis.



ABSTRACT

This work proposes the formulation of boolean function optimization problem as a mathematic problem of integer linear programming 0 and 1.

For that approach a program named Expander has been implemented in computer, based on the Shannon Expansion Method. This algorithm generates prime implicants eliminating those non needed for function cover. So, the formulation of covering problem is simplified, allowing the use of classic mathematic methods for the solution as well as all the advances in that area.

For comparison a program called Pigene based on Quine-McCluskey Method has been implemented.

The experiments where memory, run time and plenty of prime implicants generated have been compared, proved the method efficiency, that also delivered good results for function with large number of inputs.

Sumário

Capítulo I	1
Introdução	1
 Capítulo II – A Álgebra Booleana	 4
2. Álgebra de Boole	4
2.1 Função Booleana	5
2.2 Representação de uma Função Booleana	6
2.3 Introdução aos Princípios da Otimização Lógica Combinacional	9
2.3.1 Definições	10
2.4 Métodos para Otimização de Funções Booleanas	12
 Capítulo III – O Método de Quine-Mc Cluskey	 14
3.1.1 Algoritmo para a Geração dos Implicantes Primos	14
3.1.2 Tabela dos Implicantes Primos (2º fase)	16
3.2 O Algoritmo PIGENE	19
3.3 Problema de Cobertura sob a Óptica da Programação Linear 0 e 1	24
 Capítulo IV – O Teorema de Expansão de Shannon	 29
4.1 Introdução	29
4.2 Teorema de Expansão de Shannon aplicado na geração de Implicantes Primos	31
4.3 Algoritmo para a obtenção de implicantes primos pelo Método de	33

Expansão	
4.3.1 Descrição do Algoritmo	33
4.3.2 Estrutura de dados utilizada pelo algoritmo denominado Expander	40
4.3.3 Mecanismo utilizado para expandir os nós que representam as funções resíduos	43
4.4 Comparação de formulações obtidas com a aplicação dos Métodos de Expansão de Shannon e Quine-McCluskey.	47
Capítulo V – Resultados Obtidos	51
5.1 Tabulação dos resultados	51
5.2 Comparação da formulação do problema de cobertura utilizando-se o algoritmo Expander e Quine-McCluskey para geração de implicantes primos	54
Capítulo VI – Conclusões e Comentários	58
Referências Bibliográficas	60
Anexos	
Anexo A – Programa Pigene	63
Anexo B – Programa Expander	74
Anexo C – Tabela de Resultados	91



Sumário de Figuras

Figura 2.1	Espaço booleano tri-dimensional	5
Figura 2.2	Diagramas de decisão binário para a função $f(abc) = a'bc + ab'c + abc$.	8
Figura 2.3	Mapa de Karnaugh para a minimização da função $f_1(a,b,c,d) = \sum(0,1,8,9)$	13
Figura 3.1	Mapa de Karnaugh para a função $f_2(a,b,c,d) = \sum(0,1,2,5,7,8,9,10,13,15)$	15
Figura 3.2	Tabela de geração dos implicantes primos da função f_2 .	15
Figura 3.3	Comportamento do Método de Quine-McCluskey apresentado no Mapa de Karnaugh.	16
Figura 3.4	Tabela de implicantes primos (cobertura) da função f_2 .	17
Figura 3.5	Tabela de implicantes primos reduzida	18
Figura 3.6	Exemplo de arquivo de entrada	19
Figura 3.7a	Fila inicial	20
Figura 3.7b	Início da nova fila	20
Figura 3.7c	Fila inicial após o mintermo 1 passar para a nova fila	20
Figura 3.7d	Fila final ordenada	20

		xi
Figura 3.8	Primeira tabela	21
Figura 3.9	Estrutura da fila de Implicantes Primos	22
Figura 3.10	Arquivo “.qmc” de saída	23
Figura 3.11	Tabela de cobertura dos implicantes primos	23
Figura 3.12	Mapa de Karnaugh para a função $f_3(abc) = \sum(0,1,5) + d(2, 7)$	25
Figura 4.1	Passo 1 do algoritmo	34
Figura 4.2	Passo 2 do algoritmo.	35
Figura 4.3	Passo 3 do algoritmo	36
Figura 4.4	Método de Expansão de shannon aplicado à função $f(A,B,C,D)=\sum m(0,1,5,6,8,12,15) + \sum d(3,11)$	37
Figura 4.5	Implicantes Primos gerados no Mapa de Karnaugh para a função $f(A,B,C,D)=\sum m(0,1,5,6,8,12,15) + \sum d(3,11)$	38
Figura 4.6	Diagrama de decisão para a função $f(A,B,C) = \sum m(0,1,2,5,6,7)$	39
Figura 4.7	Implicantes primos para a função $F(A,B,C) = \sum m(0,1,2,5,6,7)$, posicionados no Mapa de Karnaugh	40
Figura 4.8	Campos do registro que compõe os nós da estrutura em árvore de decisão.	41
Figura 4.9	Registro da estrutura DEC que armazena os mintermos e estados don't care	41
Figura 4.10	Parte da estrutura de dados gerada para a função $F(A,B,C) = \sum m(2,3,7)$.	42

Figura 4.11	Mecanismo para expandir uma função em relação a uma determinada variável.	44
Figura 4.12	Tela do arquivo de entrada do Programa Expander	46
Figura 4.13	Tela do arquivo de saída do Programa Expander	46
Figura 5.1	Gráfico comparativo de utilização de memória	51
Figura 5.2	Gráfico de utilização de memória	52
Figura 5.3	Gráfico comparativo de tempo gasto para a geração de implicantes primos por cada algoritmo	53
Figura 5.4	Gráfico do número de implicantes primos gerados por cada algoritmo	53

Sumário de Tabelas

Tabela 2.1	Algumas propriedades do sistema algébrico booleano	4
Tabela 2.2	Tabela verdade para a função $f(abc) = a'b'c' + a'b'c + abc' + abc$	7

Capítulo I

Introdução

A simplificação de funções booleanas é parte essencial na redução dos custos da realização de circuitos lógicos e aritméticos binários.

Muitos dos algoritmos de minimização foram baseados nas idéias de Quine [2] e McCluskey [3] que deu origem ao Método de Quine-McCluskey [4] que requer a geração de todos os implicants primos e então a seleção de uma cobertura mínima, que constitui a solução mínima do problema.

Apesar do procedimento de Quine-McCluskey produzir uma solução mínima exata, a complexidade computacional do algoritmo, que aumenta exponencialmente com o número de mintermos [5], torna-o bastante limitado. O número de implicants primos de uma função com n variáveis pode ser tão grande quanto $3^n / n$ [6], o que torna o método bastante dispendioso em tempo e memória.

Em virtude disso, pesquisadores têm recorrido ao emprego de heurísticas, inicialmente gerando uma solução inicial e então a melhorando iterativamente. Os programas Mini [8], Presto [9] e Espresso – Exact [7] são exemplos de programas que se utilizam de técnicas heurísticas, sendo que esse último foi, por muitos anos, considerado o padrão em ferramenta para a otimização lógica a dois níveis.

Atualmente com o avanço das estações de trabalho renovou-se o interesse pela otimização lógica exata e surgiram vários algoritmos para geração de implicants primos de função bem como para obtenção da solução ótima (geração de implicants primos e cobertura).

Algoritmos que utilizam DDBO (diagrama de decisão binário ordenado) para a geração de implicants primos de forma eficaz têm sido amplamente empregados como no trabalho de Meinel [11] [12], Long [13], Thornton [14], Dreschler [15], Sauerhoff [16], Bryant [18].



Em trabalhos recentes como o de Fallah [10], busca-se a otimização exata aplicando-se técnicas de programação linear inteira, particularmente de relaxação de programação linear (caso particular de PLI), para solucionar o problema de cobertura.

Em publicações como de Heinrich [32], Wu Yu-Liang [33], Jain [38] a utilização de Diagramas de Decisão Binária é amplamente empregada na solução de problemas de minimização.

Observa-se, portanto, que as pesquisas envolvendo a otimização de funções booleanas tendem para uma formulação do problema, onde a estrutura em DDBO é bastante utilizada. Posteriormente aplicam-se técnicas matemáticas de otimização para a obtenção de uma solução ótima.

Dentro deste contexto é que este trabalho de pesquisa se insere, buscando através do método de Scheinman [23] a geração eficiente dos implicantes primos da função, simplificando assim a formulação do problema de cobertura, que poderá ser resolvida através de métodos clássicos matemáticos ou métodos desenvolvidos especificamente para este fim como em Fallah [10].

Atualmente o Programa Boom [39] é a ferramenta mais eficiente para a minimização de funções booleanas. O método compõe-se de três fases para funções de saída única:

1. Geração de implicantes;
2. Geração de implicantes primos;
3. Solução do problema de cobertura;

Para funções de múltiplas saídas o algoritmo executa outras fases que levam em conta a solução da cobertura do grupo e de cada saída independentemente.

Na fase de geração de implicantes chamada Coverage-directed search (CD-search) o algoritmo busca também a geração eficiente de implicantes, limitando a geração à somente aqueles realmente necessários à cobertura da função. A grande peculiaridade desta fase é que ao contrário dos outros métodos como o Espresso [7], os implicantes são gerados por adição de variáveis e não por eliminação, ou seja, a dimensão do mintermo é decrescida ao invés de aumentada.

Vários outros trabalhos foram desenvolvidos a partir do programa Boom como em Hlavicka [40] e Fiser [41].

Apresenta-se nos próximos capítulos aspectos sobre a Álgebra de Boole, o método de minimização de Quine-McCluskey, o algoritmo Pigene baseado neste método e sua estrutura de dados bem como a formulação do problema de cobertura da função. De forma análoga apresenta-se também o método de expansão de Shannon, o algoritmo Expander gerado para obtenção dos implicantes primos da função através deste método e a formulação do problema de cobertura.

Com a comparação de dezenas de casos, pretende-se avaliar a eficiência do algoritmo Expander. Os parâmetros avaliados são o consumo de memória, o tempo de processamento, e o número de implicantes primos gerados.

No Capítulo II apresenta-se um breve desenvolvimento sobre álgebra de Boole e suas definições fundamentais, para o entendimento deste trabalho.



Capítulo II

A Álgebra Booleana

2. Álgebra de Boole

Um sistema algébrico é uma combinação de um conjunto de uma ou mais operações. A álgebra Booleana é definida por um conjunto $B \supseteq \mathbf{B} \equiv \{0, 1\}$ e por duas operações, representadas por '+' e '.' as quais satisfazem a lei comutativa e distributiva e cujos elementos identidades são '0' e '1' respectivamente. Além disso qualquer elemento $a \in B$ tem um complemento, representado por a' , de modo que $a + a' = 1$ e que $a \cdot a' = 0$. Esses axiomas que definem a álgebra de Boole são muitas vezes referenciados como Postulados de Huntington.

As principais propriedades da álgebra de Boole são obtidas dos Postulados de Huntington e estão apresentados na tabela 2.1. Observe que a álgebra de Boole difere da álgebra ordinária no que se refere a aplicação da propriedade distributiva aplicada em ambos os operadores e devido a presença do complemento.

$a + (b + c) = (a + b) + c$	associativa
$a \cdot (b \cdot c) = (a \cdot b) \cdot c$	associativa
$a + a = a$	idempotência
$a \cdot a = a$	idempotência
$a + (a \cdot b) = a$	absorção
$a \cdot (a + b) = a$	absorção
$(a + b)' = a' \cdot b'$	De Morgan
$(a \cdot b)' = a' + b'$	De Morgan
$(a')' = a$	involução

Tabela 2.1. Algumas propriedades do sistema algébrico Booleano.

Existem muitos exemplos de sistemas algébricos booleanos, por exemplo, a teoria de conjuntos, cálculo de proposições e álgebra de boole aritmética. O enfoque que será dado a esse trabalho é a álgebra booleana binária, onde $B = \mathbf{B} \equiv \{0, 1\}$ e as operações ' + ', ' . ' representam a disjunção e a conjunção, respectivamente, e muitas vezes são chamadas de soma (OR) e produto (AND).

Quando variáveis binárias são associadas com as dimensões do espaço booleano, um ponto nesse espaço pode ser identificado pelos valores das variáveis correspondentes. Uma literal é uma instância de uma variável ou de seu complemento. Um produto de n literais representa um ponto no espaço booleano. Um produto de literais é chamado de cubo.

2.1 Função Booleana

Uma função booleana completamente especificada é um mapeamento entre espaços booleanos. Uma função com n entradas (variáveis) é um mapeamento representado como $f: B^n \rightarrow \{0, 1\}$. Pode-se utilizar a notação vetorial (espacial) para representar qualquer função Booleana. A figura 2.1 apresenta a notação espacial para representar um função booleana de 3 dimensões.

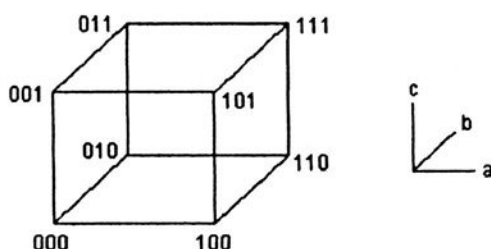


Figura 2.1. Espaço booleano tri-dimensional.

Uma função booleana incompletamente especificada é definida sobre um subconjunto de B^n . Os pontos onde a função não é definida são chamados de condições *don't care*. Estes pontos estão relacionados aos padrões de entradas que nunca podem ocorrer e para aquelas entradas para as quais as saídas não são observáveis.

Uma relação booleana é uma relação entre espaços booleanos. Ela pode ser vista como uma generalização de uma função booleana, onde um ponto no domínio pode ser associado a mais do que um ponto no co-domínio.

Qualquer função booleana pode ser representada como uma soma de produtos de n literais, chamada de mintermo da função. Alternativamente, uma função booleana pode ser representada como um produto de somas de n literais, chamada de maxtermo da função. Neste trabalho utiliza-se a notação soma de produtos, ou seja, a representação da função através de seus mintermos.

Dessa forma uma função booleana pode ser interpretada como o conjunto de seus mintermos e as operações e relações sobre funções booleanas, sobre um mesmo domínio, podem ser vista como operações e relações sobre o conjunto de seus mintermos.

Todo circuito lógico combinacional tem uma função booleana que o representa, podendo ser na forma soma de produto ou produto de soma. A meta de todos os projetos é a implementação de um circuito lógico contendo a menor quantidade possível de termos produtos (soma) e de literais.

2.2 Representação de uma Função Booleana

Existem diferentes formas de representar funções booleanas, as quais são classificadas como formas tabulares, expressões lógicas e diagramas de decisão binários.



As formas tabulares são tabelas bidimensionais. As tabelas são divididas em duas partes, correspondendo as entradas e saídas da função. As formas tabulares são muitas vezes chamadas de matriz personalidade da função.

A forma tabular mais simples é a chamada tabela verdade. A tabela verdade é uma listagem completa de todos os pontos no espaço de entrada booleano e o correspondente valor de saída. A tabela 2.2 apresenta a tabela verdade para a função $f(abc) = a'b'c + a'b c + abc' + abc$.

abc	f(abc)
000	0
001	1
010	0
011	1
100	0
101	0
110	1
111	1

Tabela 2.2 Tabela verdade para a função $f(abc) = a'b'c + a'b c + abc' + abc$.

Na forma de expressão, uma função booleana pode ser representada por uma expressão de literais unidas pelos operadores '+' e '.'. O uso de parênteses torna-se necessário dependendo da quantidade de níveis da expressão.

O número de níveis de uma função na forma de expressão, muitas vezes refere-se aos números de operadores primitivos a dois ou mais argumentos. A definição de primitiva depende do contexto em que é utilizado. Neste trabalho definem-se operadores primitivos as operações AND e OR.

As formas a dois níveis padrão são as chamadas soma de produtos de literais e produto de somas de literais. Todas as funções booleanas podem ser representadas por essas formas. Em particular, a soma de mintermos e produto de maxtermos são as formas canônicas.

A tabela verdade ou uma tabela de implicants pode ser representada por uma expressão soma de produtos, onde os produtos correspondem aos mintermos ou implicants, respectivamente.

Na representação de uma função booleana em diagrama de decisão binário, o diagrama representa um conjunto de decisões de valores binários que podem ser verdadeiras ou falsas. Um diagrama de decisão pode ser representado como uma árvore, onde as decisões estão associadas aos vértices. A figura 2.2 apresenta o diagrama de decisão binário para a função $f(abc) = a'bc + ab'c + abc$.

O diagrama de decisão binário foi inicialmente proposto por Lee [19] e posteriormente por Akers [20] para representar funções booleanas. Muitas variações de diagramas de decisões binárias têm sido propostas na literatura.

Bryant [18] apresentou o algoritmo chamado OBDD (ordered binary decision diagrams), um eficiente algoritmo para a manipulação de diagramas de decisão binário.

Todas essas representações visam facilitar a manipulação e otimização de uma dada função booleana.

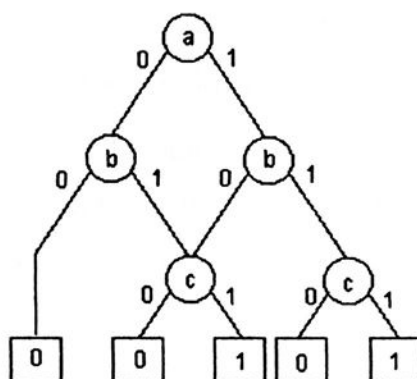


Figura 2.2 Diagramas de decisão binário para a função $f(abc) = a'bc + ab'c + abc$.

2.3 – Introdução aos Princípios da Otimização Lógica Combinacional.

O objetivo da minimização lógica a dois níveis é reduzir o tamanho da função booleana expressa na forma soma de produtos ou na forma produto de somas. Considerando que uma das formas pode ser obtida da outra aplicando-se o Teorema de De Morgan, que preserva o número de termos e de literais, este texto trata somente da representação soma de produtos, pois estas estruturas são extensamente utilizadas em PLA (Programmable Logic Array).

Uma estrutura PLA é uma macro-célula retangular que consiste de uma matriz de transistores configurados para formarem linhas em correspondência com os termos produtos e formar colunas em correspondência às entradas (variáveis) e saídas (funções). As colunas de entradas e saídas dividem a matriz em duas sub-matrizes chamadas, respectivamente, de plano de entrada e plano de saída.

A principal meta da minimização lógica é a redução dos termos produtos e das literais (variáveis). Outras metas de minimização também são utilizadas quando a representação a dois níveis deve ser implementada em estruturas que não a PLA.

A minimização lógica para funções de única saída e funções com múltiplas saídas seguem os mesmos princípios, porém é obvio que para funções com múltiplas saídas o problema torna-se mais complexo. Tal afirmação deve-se ao fato de que a minimização lógica de funções com múltiplas saídas deve explorar os termos produtos que poderão ser compartilhados por várias funções.

Este trabalho concentra-se no problema de minimização de funções booleanas com única saída, lembrando que técnicas de minimização para funções booleanas com múltiplas saídas são extensões da técnica aqui tratada.



2.3.1 Definições

Como já mencionado, existem diferentes formas para representar funções booleanas. Independente da forma de representação uma função booleana é definida como:

$$f: B^n \rightarrow \{0, 1, X\} \text{ onde } n = \text{número de variáveis.} \quad (1)$$

Todas as definições aqui apresentadas tratam de funções booleanas incompletamente especificadas, pois a funções completamente especificadas são casos especiais de funções que não têm condições *don't care* (X).

Para a saída definem-se os conjuntos on, off e dc, como um subconjunto de B^n cuja imagem é 1, 0 ou X respectivamente. As funções serão representadas como uma lista de implicantes sendo que as definições serão restritas as implicações de valores verdade (on) ou *don't care* (dc) da função. Então um implicante da função assume valores binários como segue: Um '1' implica em um valor verdade ou *don't care* da função e um '0' não implica nenhum valor da função.

Definição 1: Um implicante de uma função booleana $f: B^n \rightarrow \{0, 1, X\}$ é um par entrada e saída denotado por parte de entrada e parte de saída. A parte de entrada assume valores do conjunto $\{0, 1, X\}$ e representa um produto de literais. A parte de saída assume valores do conjunto $\{0, 1\}$. Para a parte de saída, um '1' implica um valor verdade ou *don't care* da função em correspondência com a parte de entrada.

Mintermos são implicantes sob restrições especiais.

Definição 2: Um mintermo de uma função booleana $f: B^n \rightarrow \{0, 1, X\}$ é um implicante cuja parte de entrada tem elementos no conjunto $\{0, 1\}$, isto é, um termo produto contendo exatamente n literais, e que implica valor verdade de uma e somente uma saída da função.

Um implicante corresponde a um subconjunto de mintermos da função (e de seus conjunto *don't care*). Dessa forma podem-se definir implicantes contidos, também chamados de implicantes cobertos e tamanhos (cardinalidade) de implicantes com o objetivo de relacionar implicantes com a sua cobertura e tamanho.

Definição 3: Uma cobertura para uma função booleana é um conjunto (lista) de implicantes que cobrem seus mintermos.

Adotou-se F para representar a cobertura da função f . O conjunto on, o conjunto off e o conjunto dc de uma função f podem ser representados por coberturas, onde implicantes e mintermos são relacionados com a função completamente especificada que as especifica. A cobertura do conjunto on, do conjunto off e do conjunto dc de uma função f são representados por F^{on} , F^{off} e F^{dc} , respectivamente.

Uma cobertura F de uma função f satisfaz ao limite $F^{on} \leq F \leq F^{off} \cup F^{dc}$. Pode-se notar que o limite superior é sempre satisfeito pela definição de implicante. O limite inferior está diretamente relacionado ao conceito de cobertura. Em funções completamente especificadas $F = F^{on}$.

O Tamanho (cardinalidade) de uma cobertura é o número de seus implicantes. Podem existir diferentes coberturas para uma mesma função booleana. As condições *don't care* devem ser utilizadas de forma eficiente para a redução do tamanho da cobertura de uma função incompletamente especificada.

Definição 4: Uma cobertura mínima é uma cobertura de menor cardinalidade.

Com o intuito de permitir obter a cardinalidade e conseqüentemente a comparação entre os métodos implementados, define-se como custo de um termo soma (OR) ou de um termo produto (AND) o número de variáveis (entradas) desses termos.

O objetivo da minimização lógica exata é determinar uma cobertura mínima da função f . Vários métodos para se obter a cobertura mínima são descritos na literatura, como em Hong [28], McGeer [35] e Hlavicka [39]. Algumas vezes é útil obter uma cobertura mínima local, pois a obtenção dessa cobertura pode ser conseguida através de pouco uso de memória e de tempo de processamento.

Definição 5: Uma cobertura é um mínimo local ou irredundante se a cobertura da função é uma cobertura que não está contida em nenhuma outra cobertura para a função.

Em outras palavras significa que a remoção de qualquer implicante da cobertura irredundante faz com que o novo conjunto de implicante não cubra a função, ou seja, em uma cobertura irredundante, nenhum implicante está contido em nenhum outro implicantes da cobertura.

Algumas outras definições tornam-se também necessárias.

Definição 6: *Um implicante é primo se não estiver contido em nenhum outro implicante da função. Uma cobertura é prima se seus implicantes são primos.*

Observe que essa definição está relacionada a todos os possíveis implicantes da função, e não apenas àqueles cuja cobertura está em consideração.

Alguns implicantes primos têm um propósito especial. Esses implicantes primos devem estar contidos em qualquer cobertura da função. Eles são chamados de implicantes primos essenciais.

Definição 7: *Um implicante primo é essencial se existe um mintermo da função coberto por somente aquele implicante primo, dentre todos os implicantes primos que cobrem a função.*

2.4 Métodos para Otimização de Funções Booleanas

O Mapa de Karnaugh é a forma mais conhecida para a simplificação de expressões booleanas. O mapa contém as mesmas informações contidas na Tabela Verdade, representada de um modo ligeiramente diferente.

Esse mapa contém uma célula para cada combinação de entrada, sendo que células adjacentes diferem em apenas um bit. Desse modo uma função com n variáveis contém 2^n células no mapa. A eficiência da minimização quando se utiliza o recurso do Mapa de Karnaugh depende da habilidade do projetista e torna-se impraticável mesmo para função de 5 ou 6 variáveis.

A minimização da função booleana $f(a,b,c,d) = (0,1,8,9)$ utilizando-se da manipulação algébrica é obtida através dos seguintes passos:

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

O Mapa de Karnaugh para a função $f_1(a,b,c,d) = \sum(0,1,8,9)$ com o padrão de minimização é apresentado na figura 2.3.

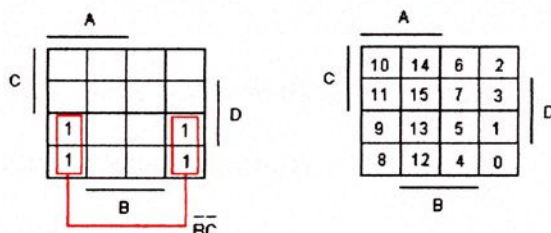


Figura 2.3. Mapa de Karnaugh para a minimização da função $f_1(a,b,c,d) = \sum(0,1,8,9)$

Note que a função está sendo representada pela soma de seus mintermos e estes estão representados na notação decimal, ou seja, o mintermo 0 corresponde ao produto $A'B'C'D'$, o mintermo 9 é representado por $AB'C'D$, e assim por diante.

Cada mintermo tem uma célula correspondente no Mapa de Karnaugh, conforme se pode observar na figura 2.3 pela aplicação sucessiva da propriedade $A.B + A.B' = A$ gerou-se o implicante $B'.C'$ que corresponde à solução de menor custo.

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 $A.B + A.B' = A(B + B') = A$.

O Método de Quine-McCluskey, e o método de obtenção de implicantes primos baseado no Teorema de Expansão de Shannon serão abordados em detalhes nos Capítulos III e IV.

Capítulo III

O Método de Quine-McCluskey

O método de Quine-McCluskey consiste de duas fases:

- Geração de todos os implicantes primos;
- Cobertura mínima dos mintermos.

3.1.1 Algoritmo para a Geração dos Implicantes Primos

(1ª Fase)

1. Separam-se os mintermos e os irrelevantes (*don't cares*) em caixas, sendo que em uma mesma caixa ficam os termos com o mesmo número de dígitos '1' na representação binária do termo produto. Este número de 1's é o índice da caixa e essa 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 de índice $i + 1$. Os termos são combinados de acordo com a propriedade $A.B + A.B' = A$, isto é, dois termos são combináveis se pertencerem a caixas adjacentes e diferirem por apenas um dígito na sua 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 que foram combinados pelo menos uma vez. Esta operação gera um novo "Grupo";
3. Combinam-se da mesma forma os termos gerados no passo anterior; o processo descrito no passo 3 continua até que não haja mais combinações

possíveis. Os termos não marcados constituem no conjunto de todos os implicants primos da função.

Aplicando-se a primeira fase do método de Quine-McCluskey a função $f_2(a,b,c,d) = \sum(0,1,2,5,7,8,9,10,13,15)$, cujo Mapa de Karnaugh está apresentado na figura 3.1, obtém-se a tabela apresentada na figura 3.2, 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;
- Os índices das caixas são dados pelo número de 1's na representação binária;
- Na geração dos grupos, pode-se produzir um mesmo impicante mais de uma vez, porém ele constará apenas uma única vez na tabela.

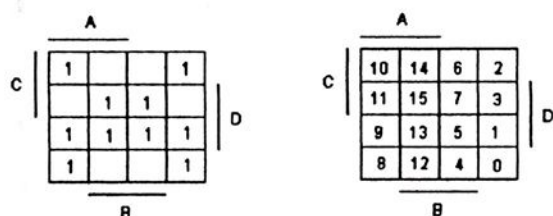


Figura 3.1 – Mapa de Karnaugh para a função $f_2(a,b,c,d) = \sum(0,1,2,5,7,8,9,10,13,15)$

Grupo 0			Grupo 1			Grupo 2		
Caixa	Mintermo	Marca	Caixa	Impicante	Marca	Caixa	Impicante	Marca
0	0000	⊗	0	000X	⊗	0	X00X	
1	0001	⊗	0	00X0	⊗	0	X0X0	
1	0010	⊗	0	X000	⊗	1	XX01	
1	1000	⊗	1	0X01	⊗	2	X1X1	
2	0101	⊗	1	X001	⊗			
2	1001	⊗	1	X010	⊗			
2	1010	⊗	1	100X	⊗			
3	0111	⊗	1	10X0	⊗			
3	1101	⊗	2	01X1	⊗			
4	1111	⊗	2	X101	⊗			
			2	1X01	⊗			
			3	X111	⊗			
			3	11X1	⊗			

Figura 3.2. Tabela de geração dos implicants primos da função f_2 .

Destaca-se neste exemplo que os termos do grupo 2 não foram combinados e, portanto não foram marcados, e formam o conjunto dos implicantes primos da função f_2 , ou seja função $f_2(a,b,c,d) = B'C' + B'D' + C'D + BD$. Pode-se notar que o custo inicial da função, dado pela soma de todos os mintermos é igual a 50 e que após a obtenção de todos os implicantes primos o custo diminuiu para 12.

Pode-se notar através dos Mapas de Karnaugh, apresentados na figura 3.3, as coberturas geradas pela primeira fase do Método de Quine-McCluskey.

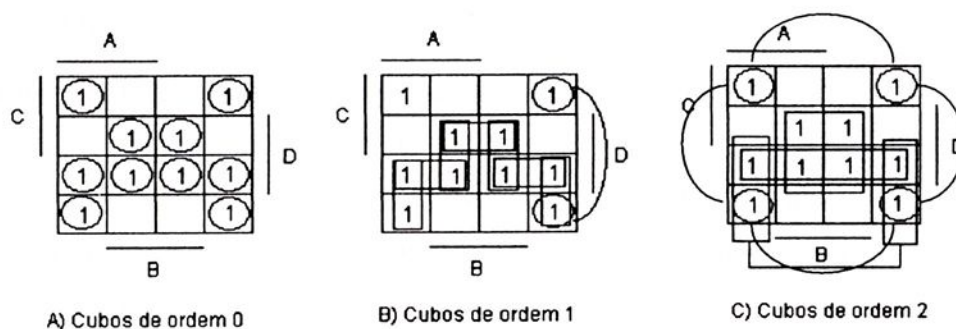


Figura 3.3. Comportamento do Método de Quine-McCluskey apresentado no Mapa de Karnaugh.

Na figura 3.3a nota-se que a cobertura é formada por todos os mintermos da função que na notação espacial correspondem aos cubos de ordem 0. Na figura 3.3b a cobertura é formada por implicantes cujos cubos são de ordem 1. Nesta figura nem todos os implicantes gerados foram representados para não sobrecarregar a figura, pois deveriam estar representados todos os implicantes contidos no grupo 1 da figura 3.2. Na figura 3.3c estão representados todos os implicantes primos gerados pela primeira fase do método estudado.

Os implicantes primos gerados foram os seguintes: $X00X$, $X0X0$, $XX01$ e $X1X1$.

3.1.2 Tabela dos Implicantes Primos (2ª fase)

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 da função. Os estados irrelevantes (termos don't care) 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.

A figura 3.4 apresenta a tabela de implicantes primos (tabela de cobertura) para a função f_2 formada com base no conjunto de implicantes primos que foram gerados na primeira fase do método e que estão apresentados no Grupo 2 da figura 3.2.

Implicantes	Mintermos de f2										
	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 3.4. Tabela de implicantes primos (cobertura) da função f_2 .

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 expressão mínima da função.

Esta tabela será reduzida com a retirada de linhas por serem consideradas essenciais (representam implicantes primos essenciais) e de colunas por terem sido cobertas, dando origem à expressão mínima da função. Com a eliminação de linhas e colunas obtém-se a tabela de implicantes primos apresentada na figura 3.5.

implicantes	mintermos	
	1	9
X00X	X	X
XX01	X	X

Figura 3.5. Tabela de implicantes primos reduzida.

Da figura 3.5 pode-se observar que faltam cobrir somente 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 primos possuem o mesmo número de literais e conseqüentemente o mesmo custo.

Dessa forma para esta função são obtidas duas expressões mínimas, ou seja:

$$f_2 = X0X0 + X1X1 + X00X = B'D' + BD + B'C'$$

ou

$$f_2 = X0X0 + X1X1 + XX01 = B'D' + BD + C'D$$

Foi apresentado em [4] que toda cobertura mínima de uma função booleana é solução de um problema de programação linear inteira 0 e 1. No referido trabalho o problema de cobertura foi abordado sob a óptica da programação linear inteira 0 e 1 e propôs-se um método de resolução baseado no algoritmo Simplex, modificado para considerar as particularidades do problema. No lugar do pivoteamento tradicional foi utilizada combinação linear das linhas do tableau para preservar os valores inteiros dos seus coeficientes.

Encontrada uma solução otimista, utilizou-se o Plano de Corte de Gomory para levar em conta somente as restrições de soluções inteiras.

Salienta-se, que se emprega duas fases distintas, a geração de implicantes primos, na primeira fase, e na segunda com o problema de cobertura.

Salienta-se, que se emprega duas fases distintas, a geração de implicantes primos, na primeira fase, e na segunda com o problema de cobertura.

Abordou-se inicialmente a fase de geração descrevendo-se um algoritmo desenvolvido com base no método de Quine-McCluskey chamado PIGENE, que gera todos os implicantes primos.

3.2 – O Algoritmo PIGENE

O programa abre um arquivo de entrada tipo texto com extensão “.min” e cria um arquivo de saída com extensão “.qmc”. Na primeira linha está digitada a característica da função: número de variáveis, o número de estados irrelevantes, o número de mintermos e após, os mintermos que a compõe (caso haja don't care states, estes serão digitados antes dos mintermos), como se vê na figura 3.6.



Figura 3.6 – Exemplo de arquivo de entrada

Criada a fila esta é analisada e ordenada de forma crescente. Esse processo cria uma fila paralela, que somente acrescenta um registro a mais, pois uma vez reescrito o termo na nova fila o antigo é eliminado e os apontadores atualizados. O processo de criação e ordenação dos mintermos e don't care é apresentado na figura 3.7

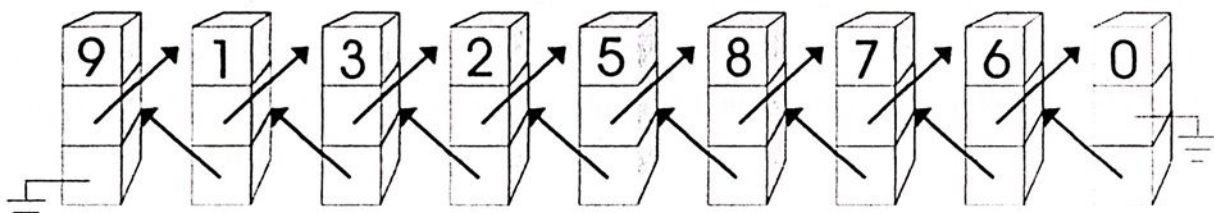


Figura 3.7a – Fila inicial

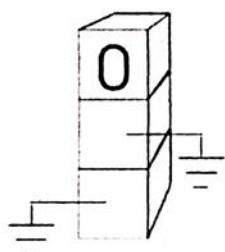


Figura 3.7b – início da nova fila

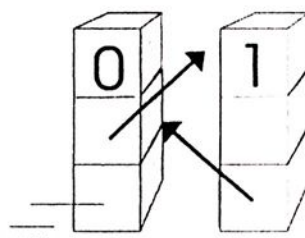


Figura 3.7c – Fila inicial após o mintermo
1 passar para a nova fila

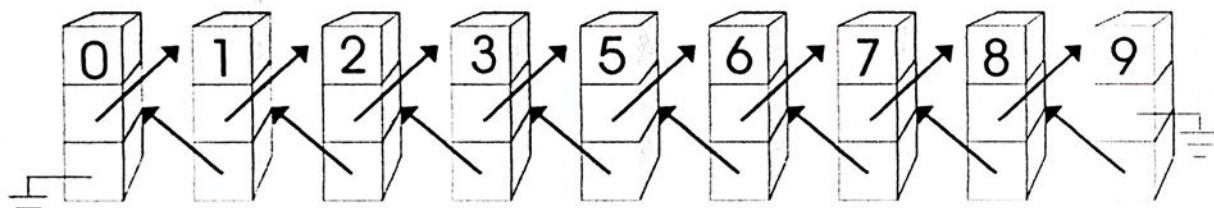


Figura 3.7d – Fila final ordenada

Quando a fila final da figura 3.7d está alocada na memória, a fila inicial já deixou de existir.

Estando a fila de entrada ordenada, o algoritmo inicia a contagem de números de "1s" contidos nos mintermos e don't care. Um novo campo é acrescentado no registro, permitindo que as caixas (passo 1 do algoritmo) possam ser rotuladas de acordo com o número de "1s" na sua representação. A figura 3.8 apresenta a fila ordenada de acordo com o número de uns.

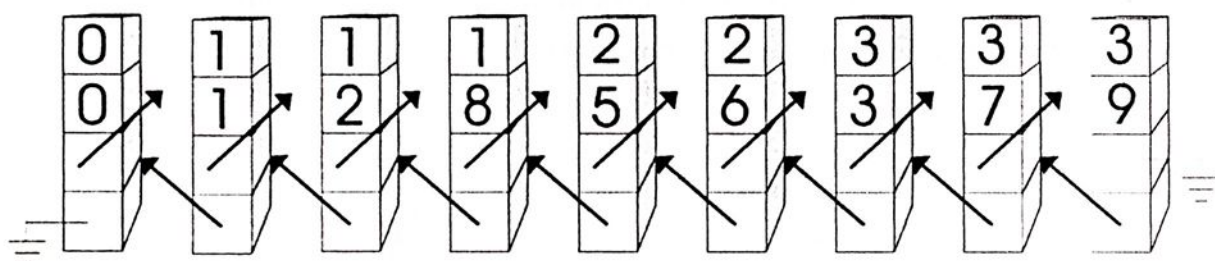


Figura 3.8 - primeira tabela

Após esta ordenação inicia-se o processo de comparar cada mintermo de uma caixa, com os mintermos da caixa subsequente, que possui um bit igual a 1 a mais. Nesta comparação procura-se identificar os termos onde se aplica o teorema $ab + ab' = a$ eliminando, desta forma, a variável do termo produto.

Pela simplicidade de se trabalhar com números decimais foi utilizada a notação: (a,b), que será utilizada no decorrer deste trabalho, onde "a" é a variável verdade e "b" indica a posição onde se eliminou a variável de entrada. Assim, o mintermo da SDP (soma do produto) original de uma função de 3 variáveis, por exemplo o número 7 será indicado como (7,0), pois 7 é 111. O implicante primo 1X0 será (4,2), 1X1 (5,2), etc.

Na função de comparação verifica-se o segundo componente da notação, correspondente a posição da variável eliminada e, caso o valor seja o mesmo, prossegue comparando bit a bit, até encontrar apenas uma diferença, que nada mais é do que verificar a adjacência. Quando a comparação de adjacência for positiva é

alocada memória novamente e tem início uma nova fila que representa a próxima tabela do algoritmo. Ao término da comparação do elemento com todos os outros da caixa subsequente, este é eliminado da fila liberando memória. Caso o elemento não seja adjacente com nenhum outro, será alocada memória novamente e uma fila de implicantes primos será iniciada. Também neste caso ele será eliminado da fila original.

Este processo segue até que não haja mais comparações ou até que a última tabela seja reduzida a uma só linha.

A partir deste ponto todas as tabelas alocadas na estrutura do tipo da figura 3.8 foram eliminadas, isto é, liberou-se memória. A estrutura resultante, na forma da figura 3.9, contém os implicantes primos ou seja, todos os implicantes primos da função. O programa irá imprimir a fila no arquivo de saída com extensão ".qmc", que terá a forma da figura 3.10. Todos os arquivos serão fechados, e o programa encerrado.

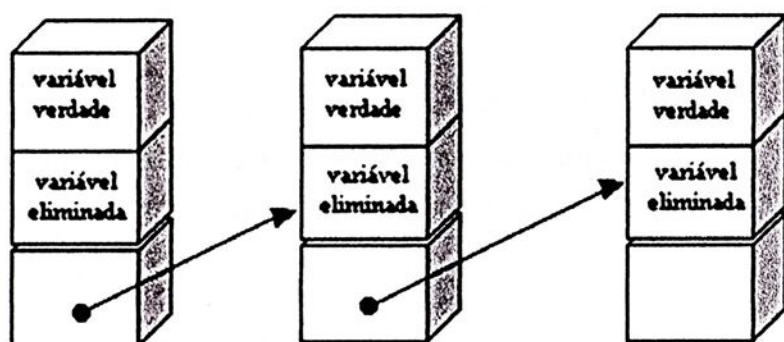


Figura 3.9 – Estrutura da fila de Implicantes Primos

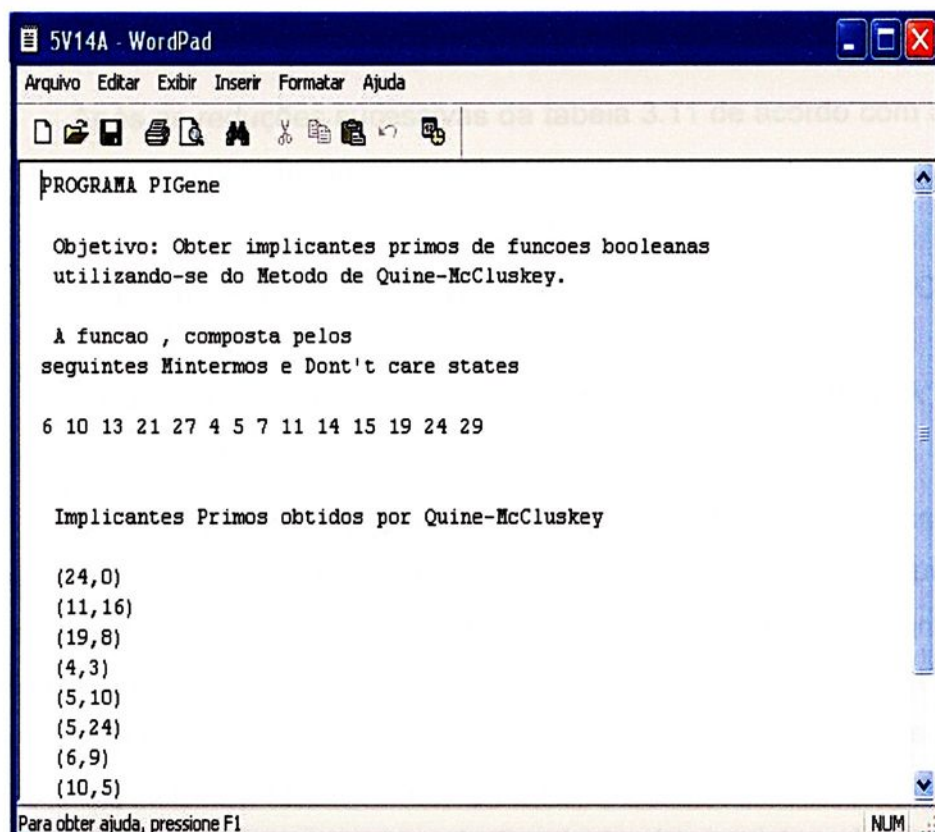


Figura 3.10 – Arquivo “.qmc” de saída

No método de Quine-McCluskey parte-se para a fase de cobertura, onde se monta uma tabela, conforme apresentada na figura 3.11, de onde se extrai a cobertura mínima da função.

Pela tabela da figura 3.11, construída com o resultado gerado pelo programa, figura 3.10, pode-se então partir para a fase de cobertura.

		1	2	4	6	7	10	11	12	13	14	15	16	18	19	29	30	31
(0,1)	0000X	X																
(18,1)	1001X													X	X			
(0,18)	X00X0		X										X	X				
(6,9)	0X11X				X	X					X	X						
(10,5)	01X1X						X	X			X	X						
(12,13)	011XX								X	X	X	X						
(13,18)	X11X1									X		X				X		X
(14,17)	X111X										X	X					X	X
(0,14)	0XXX0		X	X	X		X		X		X							

Figura 3.11. Tabela de cobertura dos mintermos

Após as reduções sucessivas da tabela 3.11 de acordo com a 2ª fase do método obtém-se a função mínima:

$$f(ABCD) = A'B'C'D' + AB'C'D + B'C'E' + A'CD + A'BD + BCE + BCD + A'E'$$

Observa-se que o termo (12,13) correspondendo a termo $A'BC$ foi eliminado na fase de cobertura.

Neste trabalho de pesquisa procurou-se métodos eficientes para a geração de implicantes primos, a partir dos quais o problemas de cobertura, tradicionalmente resolvido pela tabela de cobertura conforme apresentado na figura 3.11, pudesse ser formulado como um problema de programação linear inteira 0 e 1. Desta forma todos os avanços da área de programação matemática poderão ser empregados para se obter uma solução mínima para uma função booleana.

O problema de cobertura de função booleana visto pela óptica da programação matemática é apresentado a seguir.

3.3. Problema de Cobertura Sob a Óptica da Programação Linear Inteira 0 e 1

A obtenção de cobertura de uma função booleana com o menor número de termos produtos pode ser resolvido em dois passos. A geração de todos os implicantes primos (1ª fase do método de Quine-McCluskey) e a seleção de um conjunto mínimo de implicantes primos (2ª fase do método Quine-McCluskey), problema considerado não polinomial completo (N-P completo).

As regras tradicionais de coberturas 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 que as regras não possam mais ser aplicadas.

Em trabalhos como Silva [5], o problema de cobertura é considerado como um problema restritamente inteiro.

Dessa forma toda cobertura mínima de uma função booleana é solução de um problema de programação linear inteira 0 e 1 [5], ou seja toda cobertura mínima de uma função booleana é solução do seguinte problema de programação linear inteira:

Critério de Custo: O custo de uma cobertura é igual a soma aritmética ponderada de todos os cubos de B_n ;

Restrição 1: Para cada mintermos da função, a soma de todos os implicantes primos que cobrem o mintermo tem que ser maior ou igual a 1.

Dessa forma as restrições 1 são inequações do tipo “ ” cujo lado esquerdo é a soma dos implicantes primos (cubos) que cobrem um mintermo. Tem-se tantas restrições 1 quanto forem os mintermos da função.

Como um exemplo, considere a função f_3 cujo Mapa de Karnaugh está apresentado na figura 3.12.

	A					A			
C	1	X		1		5	7	3	1
			X	1		4	6	2	0
	B					B			

Figura 3.12. Mapa de Karnaugh para a função $f_3(abc) = \sum m(0, 1, 5) + d(2, 7)$.

Na primeira fase do método de Quine-McCluskey foram gerados os implicantes primos 00X, X01, 0X0 e 1X1 cujos custos são iguais a 3. Note que utilizou-se para descrever a função objetivo e as restrições a notação 00X para descrever o implicante primo ao invés de $A'B'$, por ser mais conveniente.

A cobertura da função f_3 formulada como um problema de programação linear inteira 0 e 1 é expressa como segue:

$$\text{MIN } 3 \cdot (00X + X01 + 0X0 + 1X1)$$

Sujeito as restrições:

$$00X + X01 = 1$$

$$00X + 0X0 = 1$$

$$X01 + 1X1 = 1$$

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

Na primeira restrição tem-se a soma dos implicantes primos que cobrem o mintermo 1. Na segunda restrição a soma dos implicantes primos que cobrem o mintermo 0 e a terceira os que cobrem o mintermo 5. Pode-se verificar que todos os mintermos estão contidos na cobertura da função e correspondem às restrições no problema de cobertura.

A solução desse problema matemático é dado por $f_3(abc) = 0X0 + X01 = A'C' + B'C$, cujo custo é igual a 6.

Conforme apresentado de forma resumida, o problema de simplificar funções booleanas com n variáveis, pode ser modelado como um problema de programação linear inteira 0 e 1. Trata-se de um problema de porte muito grande. É evidente que todas as características especiais deste problema devem ser exploradas com o intuito de solucioná-lo.

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

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

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

Considere a função $f_4(a,b,c,d) = \sum(4,5,7,12,14,15) + d(3,8,10)$. A primeira fase do método de Quine-McCluskey gera os seguintes implicants primos: $A = 1XX0$, $B = 111X$, $C = X111$, $D = 01X1$, $E = 0X11$, $F = X100$, $G = 010X$.

A cobertura da função formulada como um problema de programação matemática é a seguinte:

$$\text{MIN } 3.A + 4.(B + C + D + E + F + G)$$

Sujeito as restrições

$$G + F \geq 1$$

$$G + D \geq 1$$

$$E + D + C \geq 1$$

$$A + F \geq 1$$

$$A + B \geq 1$$

$$C + B \geq 1$$

$$A, B, C, D, E, F, G \in \{0, 1\}$$

É importante salientar que uma vez formulado o problema de cobertura como um problema de programação matemática, a questão da simplificação de funções booleanas se abre para a aplicação de todos os avanços nesta área de pesquisa.

O grande problema da primeira fase do método de Quine-McCluskey na geração de implicants primos é que são gerados todos os implicants da função para posterior obtenção dos implicants que são primos, condição necessária e suficiente para a formulação do problema de cobertura como um problema de programação linear inteira 0 e 1.

Isto tem motivado a busca de métodos eficientes para a geração de implicants primos sem que se tenha a necessidade da geração de todos os implicants da função, o que economizaria muito tempo de processamento e memória.

Esta busca de geração de implicantes primos eficiente, trata-se portanto, na busca de métodos que eliminam alguns implicantes primos, durante a fase de geração.

Empregando-se métodos algébricos baseados nas propriedades e teoremas da álgebra booleana, na função $f_4(a,b,c,d) = \sum m(4,5,7,12,14,15) + d(3,8,10)$, utilizada no exemplo anterior, o problema de cobertura é formulado como:

$$\text{MIN } 3(A_1 + B_1) + 4(C_1 + D_1)$$

Sujeito as restrições:

$$C_1 \geq 1$$

$$A_1 \geq 1$$

$$A_1 + D_1 \geq 1$$

$$B_1 + C_1 \geq 1$$

$$B_1 \geq 1$$

$$D_1 \geq 1$$

$$A_1, B_1, C_1 \text{ e } D_1 \in \{0,1\}$$

Nesta formulação $A_1 = 01X1$, $B_1 = 1XX0$, $C_1 = X100$ e $D_1 = X111$. Nota-se tratar de um modelo matemático, mais simplificado do que o apresentado anteriormente, cuja obtenção da solução, certamente, exigirá menos recurso computacional.

É evidente que o emprego de métodos algébricos torna-se impraticável ou de difícil implementação em computador.

Apresenta-se no próximo capítulo um algoritmo que implementa o Teorema de Expansão de Shannon, utilizado na geração de implicantes primos de uma função booleana. Avaliou-se a eficiência do algoritmo implementado comparando-o com o método de Quine-McCluskey.

Capítulo IV

O Teorema de Expansão de Shannon

4.1 - Introdução

A geração de implicants primos pelo Método de Expansão de Shannon é uma técnica iterativa, que permite obter implicants primos de uma dada função booleana através de simples operações sobre o conjunto de mintermos e/ou estados "don't care", representados em notação decimal, que descreve a função original. Tal método pode ser empregado em função com um grande número de variáveis. Todos, ou quase todos, os implicants primos que não estariam no conjunto solução são eliminados, simplificando, desse modo, a busca pela solução mínima.

O teorema diz que qualquer função booleana $f(x_1, x_2, \dots, x_n)$ pode ser expressa por:

$$f(x_1, x_2, \dots, x_n) = x_1 \cdot f(1, x_2, \dots, x_n) + x'_1 \cdot f(0, x_2, \dots, x_n) \quad (1)$$

Por indução perfeita, pode-se provar o referido teorema.

Prova: Faça x_1 assumir valor lógico 1. Conseqüentemente x'_1 assume valor lógico 0 e desse modo a equação (1) torna-se uma identidade, isto é:

$$f(x_1, x_2, \dots, x_n) = 1 \cdot f(1, x_2, \dots, x_n)$$

De modo similar, substituindo-se $x_1 = 0$ e $x'_1 = 1$ a equação (1) também será reduzida a uma identidade, provando-se desse modo o teorema.

Se o teorema for novamente aplicado em relação a variável x_2 , em cada um dos dois termos obtidos pela expansão em relação à variável x_1 (equação 1), obtém-se a seguinte expressão:

$$\begin{aligned} f(x_1, x_2, \dots, x_n) = & x_1 x_2 \cdot f(1, 1, x_3, \dots, x_n) + x_1 x'_2 \cdot f(1, 0, x_3, \dots, x_n) + \\ & + x'_1 x_2 \cdot f(0, 1, x_3, \dots, x_n) + x'_1 x'_2 \cdot f(0, 0, x_3, \dots, x_n) \end{aligned}$$

A expansão da função $f(x_1, x_2, \dots, x_n)$ sobre o restante das variáveis produz uma forma normal disjuntiva (soma de produto). De maneira similar, a aplicação repetida do dual do Teorema da Expansão sobre as mesmas variáveis produz a forma normal conjuntiva (produto de soma).

Em termos práticos, a maneira mais rápida e simples para se obter a forma soma de produto canônica de uma função booleana é resumida a seguir:

1 – Examine cada termo; Se ele é um mintermo, mantenha-o, e continue a examinar os outros termos;

2 – Em cada produto que não é um mintermo verifique qual a variável que está faltando; Para cada x_i que está faltando multiplique o produto por $(x_i + x'_i)$.

3 – Multiplique todos os produtos e elimine os termos redundantes.

Para exemplificar, a aplicação do procedimento acima descrito, considere que se deseja obter a forma soma de produto canônica para a função $f_1(x, y, z) = x'y + z' + xyz$.

$$\begin{aligned} f_1 &= x'y + z' + xyz = \\ &= x'y.(z + z') + (x + x') . (y + y').z' + xyz = \\ &= x'yz + x'y z' + xyz' + xy'z' + x'yz' + x'y'z' + xyz = \\ &= x'yz + x'y z' + xyz' + xy'z' + x'y'z' + xyz. \end{aligned}$$

Do exposto, pode-se concluir que a forma soma de produto canônica de qualquer função booleana é única. Essa afirmação pode ser provada assumindo que existem duas diferentes formas de soma de produto canônicas que representam uma função F qualquer. Desde que se assume que as formas são diferentes, elas devem diferir em pelo menos um dos mintermos, isto é, deve existir no mínimo um conjunto de valores para as variáveis $x_1, x_2, \dots, x_n$, de modo que uma das formas canônicas resulta em $f(x_1, x_2, \dots, x_n) = 0$, enquanto que a outra forma produz $f(x_1, x_2, \dots, x_n) = 1$, resultado este que contradiz a suposição de que ambas as formas canônicas representam a mesma função.

Duas funções booleanas, $f_1(x_1, x_2, \dots, x_n)$ e $f_2(x_1, x_2, \dots, x_n)$ são logicamente equivalentes (ou simplesmente equivalentes), se e somente se, ambas as funções têm os mesmos valores para toda e qualquer combinação das variáveis $x_1, x_2, \dots, x_n$. Desse modo, pode-se também concluir que duas funções booleanas são equivalentes, se e somente se, suas formas soma de produto canônicas são idênticas. Conseqüentemente, para provar a identidade de duas funções, é suficiente expandir ambas as funções em suas formas canônicas e compará-las.

4.2 – Teorema de Expansão de Shannon aplicado na geração de Implicantes Primos.

Considere que a constante binária a_i é o valor da função $f(x_1, x_2, \dots, x_n)$ para uma combinação de variáveis cujo código decimal é i . Desse modo, toda função booleana pode ser expressa na forma:

$$f(x_1, x_2, \dots, x_n) = a_0 x'_1 x'_2 \dots x'_n + a_1 x'_1 x'_2 \dots x_n + \dots + a_r x_1 x_2 \dots x_n$$

O fator a_i assume o valor 1 ou 0, se o correspondente mintermo está ou não está, respectivamente, contido na forma canônica da função. Existem $r = 2^n$ coeficientes, sendo que cada coeficiente pode assumir dois valores, 0 ou 1. Então, existem 2^{2^n} possíveis atribuições de valores para os coeficientes, e pode-se, desse modo, concluir que existem 2^{2^n} funções booleanas de n variáveis.

Note que uma função booleana de duas variáveis pode ser representada por sua forma soma de produto canônica como segue:

$$f(x, y) = a_0 x' y' + a_1 x' y + a_2 x y' + a_3 x y$$

Existem, então, 2^2 funções, correspondendo às 16 possíveis atribuições de valores (0 ou 1) para os termos a_0 , a_1 , a_2 e a_3 .

Conforme já mencionado, o Teorema de Expansão permite expandir qualquer função booleana de n variáveis sobre qualquer uma de suas variáveis. Quando uma função é expandida sobre uma variável x_1 , por exemplo, duas novas funções são obtidas. Uma multiplicando x_1 e a outra multiplicando x'_1 . Essas duas novas funções booleanas são funções somente das variáveis $x_2, x_3, \dots, x_n$ e são chamadas de funções resíduo de x_1 .

Considere que uma certa função é especificada na sua forma soma de produto canônica, ou seja $f(x_1, x_2, \dots, x_n) = \sum m_j$, onde m_j representa um termo produto qualquer, e o índice j é o decimal equivalente ao termo produto associado, quando o termo produto é expresso na sua forma binária, quando as literais suprimidas são representadas por 0, e as não suprimidas são representadas por 1. Desse modo, as funções são geralmente escritas como $f(x_1, x_2, \dots, x_n) = \sum j$, sendo que o m é implícito.

Considere também, que cada posição das variáveis assume um peso, que está ordenado, de modo que x_1 assume o maior peso binário e x_n assume o menor peso binário.

Quando as variáveis assumem pesos, conforme acima descrito, a expansão torna-se um processo muito simples. Em particular, se um conjunto de decimais define uma função, o resíduo, com relação à variável x_1 pode ser obtido como segue:

$$f(x_1, x_2, \dots, x_n) = \sum j = x_1 \cdot R_1 + x'_1 \cdot R_2$$

Onde: R_2 é a somatória dos números decimais que são menores que o peso assumido pela variável x_1 (variável que está sendo expandida) e R_1 é a somatória dos números maiores ou iguais ao peso de x_1 , sendo que de cada um desses números é subtraído o peso de x_1 .

Cada um dos resíduos pode agora, ser expandido em termos da variável x_2 , que corresponde à variável de maior peso do resíduo. Desse modo, pode-se expandir todas as variáveis da função.

Para exemplificar, considere a função $f(A,B,C,D) = \Sigma(0,1,8,9,10) = A R_1 + A' R_2$, onde a variável A é a de maior peso (igual a 8) e a variável D é a de menor peso (igual a 1), lembrando que as variáveis assumem os pesos ... 32, 16, 8, 4, 2 e 1.

$$R_2 = f(B,C,D) = \Sigma(0,1) \text{ e}$$

$$R_1 = f(B,C,D) = \Sigma(8-8),(9-8),(10-8) = \Sigma(0,1,2).$$

Observa-se, que após a função $f(A,B,C,D) = \Sigma(0,1,8,9,10)$ ser expandida em relação à variável A, cujo peso é 8, duas novas funções foram obtidas; a função $f(B,C,D) = \Sigma(0,1)$ e a função $f(B,C,D) = \Sigma(0,1,2)$, sendo que ambas não dependem mais da variável A .

Desse modo, a função:

$$\begin{aligned} f(A,B,C,D) &= A.\Sigma(0,1,2) + A'. \Sigma(0,1) \\ &= \Sigma(0,1) + A.\Sigma(0,1,2) + A'. \Sigma(0,1) \end{aligned}$$

Onde a somatória $\Sigma(0,1)$ é a parte do resíduo da variável A, que são os mesmos que A' . Esses decimais são redundantes na somatória multiplicando A e na somatória multiplicando A' , e representa os termos da soma padrão que diferem somente na variável A. Cumpre salientar, que quase a totalidade dos métodos de minimização de funções booleanas vale-se do teorema $AB + AB' = A$.

Esse processo pode ser repetido expandindo as funções resíduos em termos da variável B, C, e assim por diante.

4.3 – Algoritmo para a obtenção de implicants primos pelo Método de Expansão.

4.3.1 – Descrição do Algoritmo



O algoritmo para a obtenção dos implicants primos utilizando-se do método de Expansão de Shannon foi inicialmente estudado por Scheinman em 1962 [23]. Trata-se de um método manual e iterativo e de difícil entendimento. A partir dos estudos de Scheinman deu-se ao método uma abordagem através da mais avançada técnica de programação estruturada e nas estruturas de dados utilizados, empregou-se somente apontadores e alocação dinâmica de memória.

A estrutura de dados obtida é de extrema complexidade e de difícil programação, porém, o algoritmo utiliza somente duas operações aritméticas, a subtração e a potenciação, e uma operação lógica, a igualdade. O método será ilustrado, para maior clareza, utilizando-se a função $f(A,B,C,D)=\sum m(0,1,5,6,8,12,15)+d(3,11)$ de 4 variáveis, e consiste nos seguintes passos, lembrando no entanto, que a função poderia, teoricamente, conter qualquer número de variáveis:

1 – Ordene em uma coluna, de forma crescente, como na figura 4.1 as representações decimais dos mintermos e dos estados “don’t care” que representam a função. Marque os estados don’t care. Os “don’t care” devem ser considerados, pois, se convenientes, eles são utilizados para obter implicants primos de dimensões maiores. Considera-se dimensão de um impicante a quantidade de variáveis irrelevantes que eles possuem. Quanto menor a quantidade de variáveis (literais), na sua forma negada ou não, que um impicante possuir, maior será a sua dimensão. Essa coluna é considerada, no algoritmo proposto, como sendo o nó raiz.

0 ✓
1
2
3 ✓
5
6
8
11
12
15

Figura 4.1 – Passo 1 do algoritmo

2 – Considerando-se as variáveis ordenadas como ABCD em relação a posição no termo produto $ABCD = 2^3, 2^2, 2^1$ e 2^0 , vê-se que a variável A é a de maior peso (peso 8). Divida o conjunto formado no passo anterior em dois outros grupos de

colunas como exemplificado na figura 4.2; uma rotulada por A' e a outra rotulada por A. A coluna A' contém os termos (mintermos e " don't cares ") da função original, cuja representação decimal é menor do que 8 (peso da variável A).

A coluna A contém os termos da função original, cuja representação decimal é maior ou igual a 8, sendo que de cada um desses números é subtraído 8, valor correspondente à variável A que está sendo expandida.

		0
		1 ✓
		3 ✓
		5
		6
		8
		11
		12
		15
Peso 8		
A'		A
0		0
1		3
3		4
5		7
6		

Figura 4.2 – Passo 2 do algoritmo.

3 – Crie uma terceira coluna com o rótulo X, como apresentado na figura 4.3. Essa coluna indicará a redundância entre A e A'. Nesta coluna, têm-se as representações decimais que são comuns tanto da coluna A como da coluna A'. Esses decimais comuns à coluna A e A' devem ser marcados nas respectivas colunas para indicar que são termos redundantes em relação à variável A.

Se algum decimal contido na coluna X foi anteriormente marcado em ambas as colunas, A e A' eles também devem ser marcados na coluna X.

	0		
	1		
	3 ✓		
	5		
	6		
	8		
	11 ✓		
	12		
	15		
Peso 8			
	A'	A	X
0 ✓		0 ✓	0
1		3 ✓	3
3 ✓		4	
5		7	
6			

Figura 4.3 – Passo 3 do algoritmo

4 – Examine cada uma das colunas geradas. Se alguma coluna contém somente decimais marcados, toda a coluna deve ser eliminada, pois todos os seus termos estão contidos nas outras colunas.

5 - Cada uma das colunas criadas no passo 3 devem ser novamente expandidas em relação à variável B, e repete-se os passos 2, 3 e 4. Se um número da coluna B for resultado de um número já marcado o seu correspondente também deverá ser marcado, no exemplo seria o caso do número 3 da coluna B' do ramo X e do número 1 da coluna C do ramo B' do ramo X, conforme pode-se verificar na figura 4.4.

Quando a função é expandida em relação à variável final (a de menor peso), que no algoritmo proposto são consideradas como folhas, o resíduo sempre deve ser 0. Os passos 2,3,4 e 5 devem ser repetido para todas as variáveis até se obter resíduo igual a 0.

A figura 4.4 mostra o algoritmo totalmente desenvolvido para a função $f(A,B,C,D) = \sum m(0,1,5,6,8,12,15) + d(3,11)$ utilizada como exemplo.

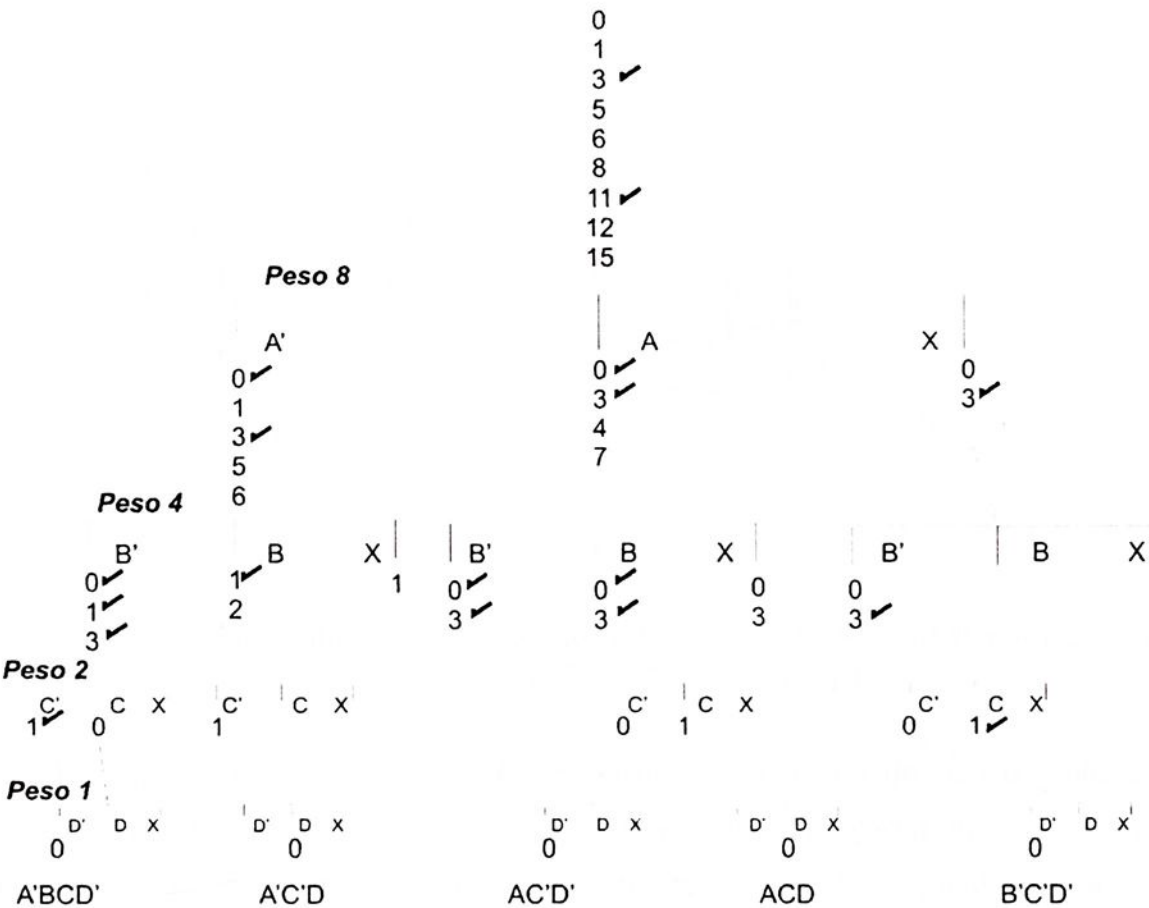


Figura 4.4 – Método de Expansão de shannon aplicado à função $f(A,B,C,D)=\sum m(0,1,5,6,8,12,15) + \sum d(3,11)$

6 – Obtenha os implicants primos da função traçando os caminhos que vão das folhas (variável de menor peso) até a raiz (variável de maior peso).

Os rótulos dos ramos contidos no caminho representam os implicants primos obtidos.

A figura 4.5 apresenta no Mapa de Karnaugh os implicants primos gerados pelo Método de Expansão para a função $f(A,B,C)= \sum m(0,1,5,6,8,12,15) + \sum d(3,11)$.

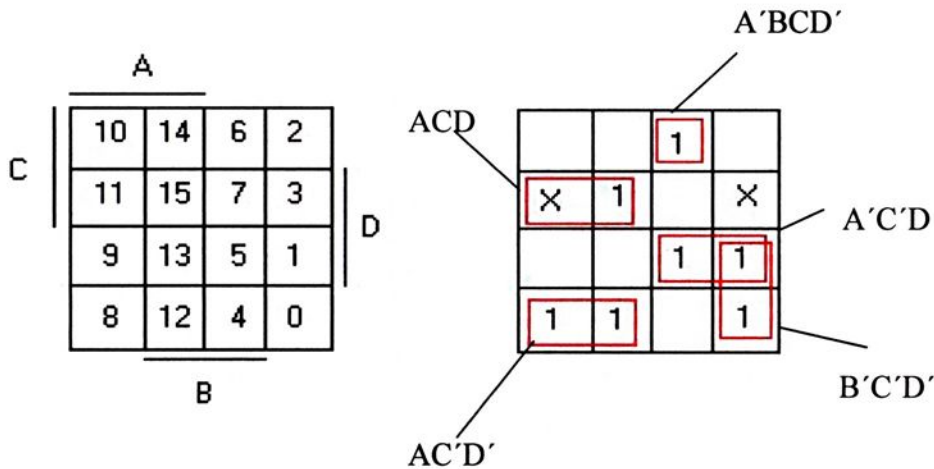


Figura 4.5. Implicantes Primos gerados no Mapa de Karnaugh para a função $f(A,B,C,D) = \Sigma m(0,1,5,6,8,12,15) + \Sigma d(3,11)$

Um aspecto muito importante a ser salientado é o fato de que o método de expansão obtém todos os implicantes primos essenciais da função e que alguns ou, às vezes, todos os implicantes primos que não estariam no conjunto solução são automaticamente eliminados no processo de construção do Diagrama de Decisão. Essa situação é ilustrada no exemplo 2.

Exemplo 2. Obter pelo método de expansão de Shannon, os implicantes primos da função $f(A,B,C) = \Sigma m(0,1,2,5,6,7)$.

Aplicando-se a primeira fase do método de Quine-McCluskey, que gera todos os implicantes primos da função, são obtidos os seguintes implicantes primos: $A'B'$, $A'C'$, AB , AC , $B'C$ e BC' .

A figura 4.6 apresenta o diagrama de decisão utilizado pelo algoritmo que obtém os implicantes primos pelo método de expansão.

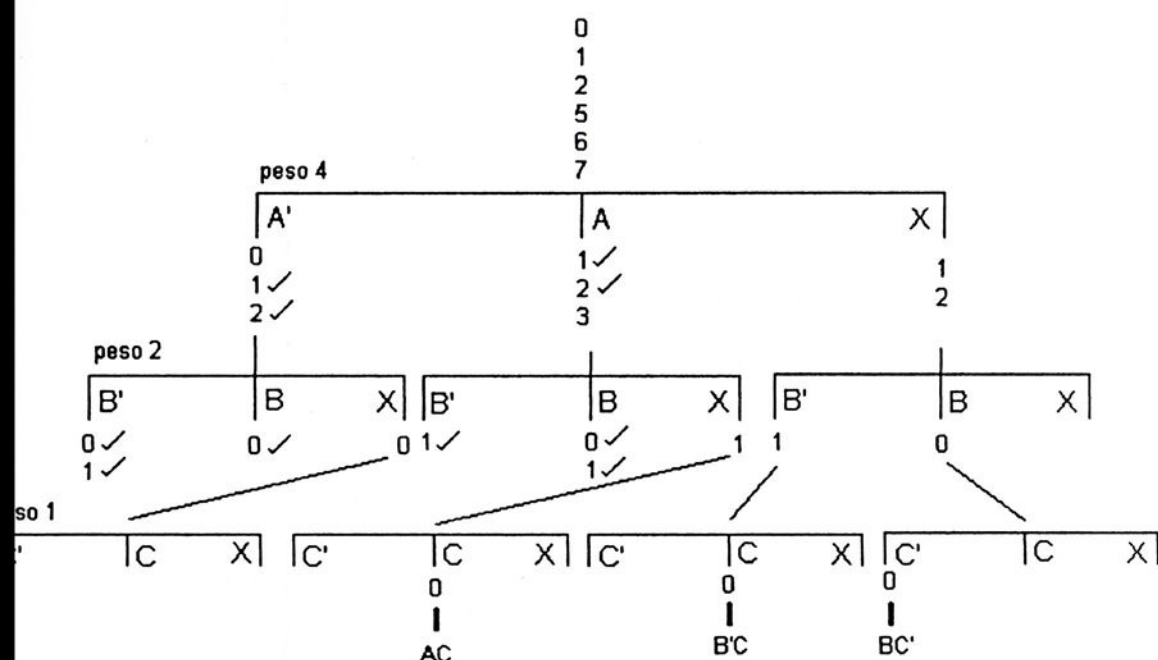


Figura 4.6. Diagrama de decisão para a função $f(A,B,C) = \sum m(0,1,2,5,6,7)$

No diagrama da figura 4.6 deve ser notado que o decimal 1 referente à variável B' do nó A' está marcado. Isto se deve ao fato de que na coluna referente à variável A' , ele já havia sido marcado, pois também havia na coluna A um decimal 1. A mesma observação pode ser feita para o decimal 0 referente à variável B do nó A . Como o decimal 2 da coluna referente à variável A tinha sido marcado, pois também está contido na coluna referente à variável A' , quando se subtrai desse decimal o peso 2, obtém o decimal 0, que foi originado do decimal 2 que já havia sido marcado pelo motivo já exposto.

A figura 4.7 apresenta no Mapa de Karnaugh os implicants primos gerados pelo Método de Expansão.

Pode-se notar pelo Mapa apresentado na figura 4.7 que os implicants primos AB e $A'B'$ não foram gerados pelo Método de Expansão. Nota-se, também, que todos os mintermos estão contidos em pelo menos um dos implicants primos.

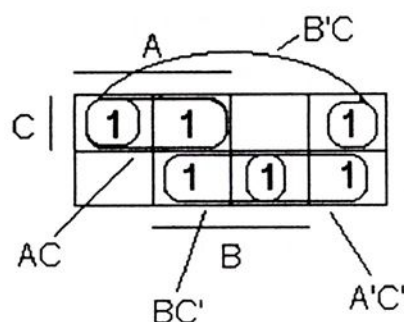


Figura 4.7. Implicantes primos para a função $f(A,B,C) = \sum m(0,1,2,5,6,7)$, gerados pelo método de expansão, posicionados no Mapa de Karnaugh

De posse dos implicantes primos, o problema de cobertura poderá ser formulado como um problema de programação linear inteira 0 e 1 [4].

É oportuno lembrar que o objetivo do programa é obter os implicantes primos da função booleana. Após obter o conjunto dos implicantes primos, a cobertura dos mintermos deve ser formulada como um problema de programação linear inteira 0 e 1 que será resolvido por métodos matemáticos clássicos.

4.3.2 – Estruturas de dados utilizadas pelo algoritmo denominado Expander.

Cada nó da árvore de decisão utilizada constitui-se em um registro cujos campos estão apresentados na figura 4.8.

O campo Peso armazena o peso que a variável expandida assume na representação binária do mintermo ou do estado "don't care". O campo Min_dec é um apontador que aponta para registro de uma outra estrutura chamada Dec que contém todos os mintermos e estados "don't care".

REGISTRO DDB



Figura 4.8. Campos do registro que compõe os nós da estrutura em árvore de decisão.

O registro da estrutura DEC está apresentado na figura 4.9. O campo Var_neg, do registro DDB, é um apontador que aponta para registros do tipo DDB. Este registro contém a função resíduo que deverá ser expandida no momento apropriado. O campo Flag_neg sinaliza se a função resíduo referente a variável negada da expansão da variável anterior deverá ser expandida. O campo Var_verd é um apontador que aponta para registros do tipo DDB e que também contém a função resíduo que deverá ser expandida no momento apropriado. O Flag_verd tem a mesma função que o Flag_neg, porém está relacionado com a variável verdade. O campo Var_irrel é um apontador que aponta para registros do tipo DDB que contém a função cujos termos são comuns tanto à função contida em Var_neg quanto à função contida em Var_verd. O Flag_irrel sinaliza se a função contida em Var_irrel deverá ser expandida. O apontador anterior tornou-se necessário para que a partir das folhas da árvore pudesse chegar até sua raiz, determinando, desse modo, os implicantes primos gerados pelo algoritmo.

REGISTRO DEC

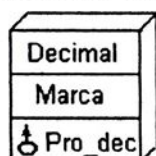


Figura 4.9. Registro da estrutura DEC que armazena os mintermos e estados don't care

O campo Decimal, do registro DEC, armazena um elemento que representa um mintermo ou um estado " don't care ". O passo 1 do algoritmo, anteriormente descrito, trabalha com esta estrutura de dados. O campo Marca é utilizado para indicar se o decimal correspondente pertence tanto à função contida em Var_neg quanto à função contida em Var_verd. O apontador Pro_dec aponta para o próximo registro Dec que armazena um outro mintermo ou estado " don't care ".

Duas outras estruturas são utilizadas, uma para armazenar informações sobre as variáveis (nós da árvore de decisão) que deverão ser expandidas e a outra para representar os nós terminais da estrutura de árvore (folhas). Através das folhas contidas nesta última estrutura, pode-se chegar até a raiz, extraíndo, desse modo, o implicante primo que está sendo representado pelo caminho percorrido.

A figura 4.10 apresenta parte da estrutura de dados gerada quando o algoritmo implementado trabalhar com a função $F(A,B,C) = \sum m(2,3,7)$.

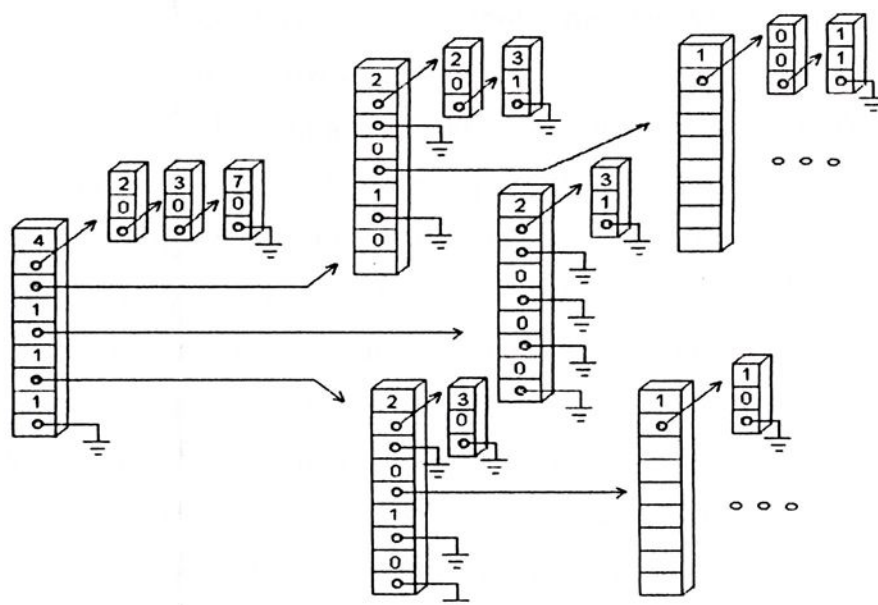


Figura 4.10. Parte da estrutura de dados gerada para a função $F(A,B,C) = \sum m(2,3,7)$.

Nota-se pela figura 4.10, que a estrutura de dados tem um certo grau de complexidade, pois como se pode notar, uma função de três variáveis e com apenas 3

mintermos gerou esse emaranhado de apontadores. Vale salientar, que o apontador denominado anterior no registro DDB, está apontando para o NULL para simplificar a visualização, sendo que na realidade ele aponta para estruturas anteriores.

Todos os nós da árvore são expandidos, um de cada vez e as novas funções resíduos geradas são inseridas numa fila, e tratadas no devido tempo. O mecanismo para se expandir um nó é apresentado a seguir.

4.3.3 - Mecanismo utilizado para expandir os nós que representam as funções resíduos.

Os passos 2, 3 e 4 são os procedimentos que expandem uma função resíduo em relação a uma determinada variável. O algoritmo inicia-se fazendo a expansão da variável que representa o nó raiz (variável de maior peso). Após o nó raiz ser tratado, deve-se selecionar um outro nó para que a função resíduo que ele representa seja expandido em relação à variável correspondente. A condição de expansão para um nó é estabelecida pelo flag correspondente, que sinaliza quando o respectivo nó contém função que deverá ser expandida.

A figura 4.11 ilustra o mecanismo utilizado pelo algoritmo para expandir um nó que representa uma função a ser expandida em relação a uma determinada variável.

Considere que o nó que está sendo expandido é o nó raiz. Com as informações contidas na estrutura rotulada por "1" na figura 4.11, verifica-se a possibilidade dos apontadores Var_neg e Var_verd apontarem para registros DDBs. O passo 2 do algoritmo é quem define se haverá ou não esses registros. Na figura 4.11 estão rotulados por "2" e "3", respectivamente. Uma vez definido esses registros, criam-se as estruturas rotuladas por "4" e "5", que contêm os mintermos e estados " don't cares " que são, respectivamente, menores e maiores ou iguais ao peso do nó que se está expandindo. Através do passo 3 do algoritmo, verifica-se a necessidade do apontador Var_irrel apontar para registro DDB, rotulado na figura 4.11 por "6". Esse registro será criado se houver decimais na estrutura "4", que são iguais a decimais contidos na estrutura "5". Com os elementos contidos nas estruturas "4" e "5" gera-se a estrutura rotulada por "7".

Os nós, que na figura 4.11 estão rotulados por “2”, “3” e “6”, também devem ser expandidos. Utilizou-se uma rotina que verifica se um nó gerado tem condições para expansão. Caso exista essa condição, o registro que corresponde ao nó a ser expandido é inserido numa fila, onde todos os registros nela contida serão tratados no devido tempo. O tamanho dessa fila é dinamicamente alterado. Desde que um registro é tratado ele é eliminado da fila, liberando, desse modo, memória para o sistema.

Quando o algoritmo detecta que um nó gerado corresponde a uma folha da árvore, esse nó é inserido numa quarta fila, pois é a partir desse nó que se caminha até a raiz para se extrair os implicantes primos representados pelos caminhos que vão das folhas até a raiz.

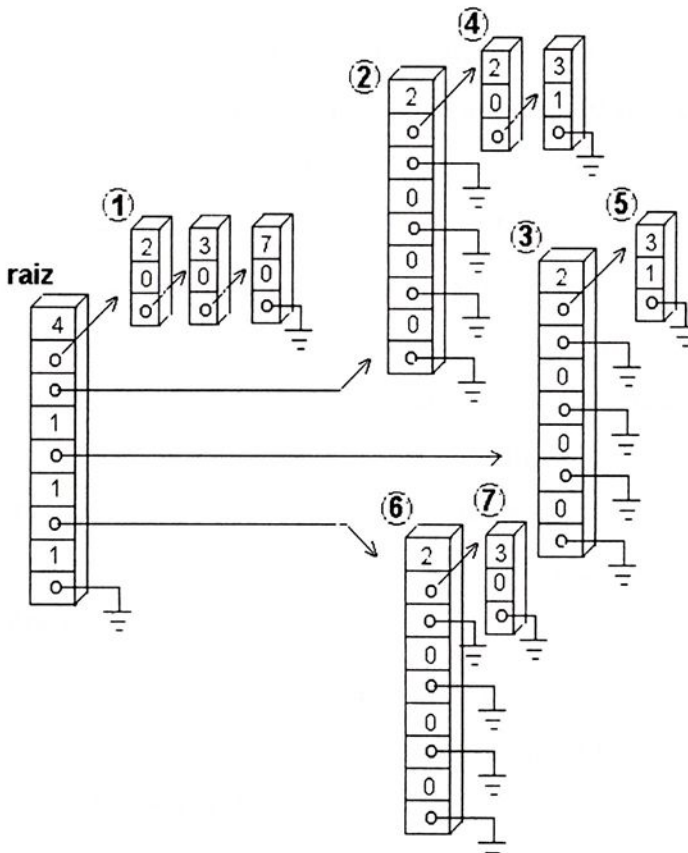


Figura 4.11. Mecanismo para expandir uma função em relação a uma determinada variável.

Como já foi mencionado, o algoritmo proposto utilizou, em abundância, da alocação dinâmica de memória. Na linguagem de programação C pode-se liberar e alocar memória para os registros dinamicamente, utilizando, respectivamente, as rotinas `free()` e `malloc()` da biblioteca padrão. Este é um mecanismo muito versátil para se trabalhar com estruturas de dados cujas dimensões não se conhecem a priori.

Caso a memória do sistema seja limitada e haja necessidade de armazenar informações durante um curto período de tempo, pode-se alocar (destinar) memória do sistema para armazenar tais informações e devolvê-la para a área de memórias livres ao terminar o trabalho.

Pelo fato de uma codificação dessa natureza poder ser causadora de uma infinidade de problemas e enganos, essa solução é recomendada somente quando houver pouca disponibilidade de memória no sistema ou quando trabalhar com estruturas de dados com um certo grau complexidade.

No caso da minimização de funções booleanas, essa solução tornou-se necessária e inevitável, pois as funções booleanas de uso prático contêm dezenas e até centenas de variáveis, além do que não se pode prever quantos são os mintermos e os estados "don't cares" que representam as funções.

Definir uma estrutura estática seria, com toda certeza, uma péssima implementação por dois motivos básicos. Primeiro, o tamanho fixo de uma estrutura, limita o tamanho das funções que poderiam ser trabalhadas, pois coloca um limite arbitrário no programa. Segundo, uma estrutura de tamanho fixo não aproveita a memória adicional que possa ser acrescentada ao computador, e caso haja eliminação de memória, como, por exemplo, devido a uma falha num circuito impresso, o programa não poderá operar.

É certo que para alguém que lê um programa em linguagem C, codificado somente utilizando apontadores e alocação dinâmica de memória, podem parecer linhas pouco decifráveis, porém é certo, também, que tal solução é de uma elegância indescritível.

A listagem completa do programa, intitulado Expander está apresentada, na íntegra, no apêndice B.



As figuras 4.12 e 4.13 exemplificam a forma de entrada e saída de dados do programa Expander.

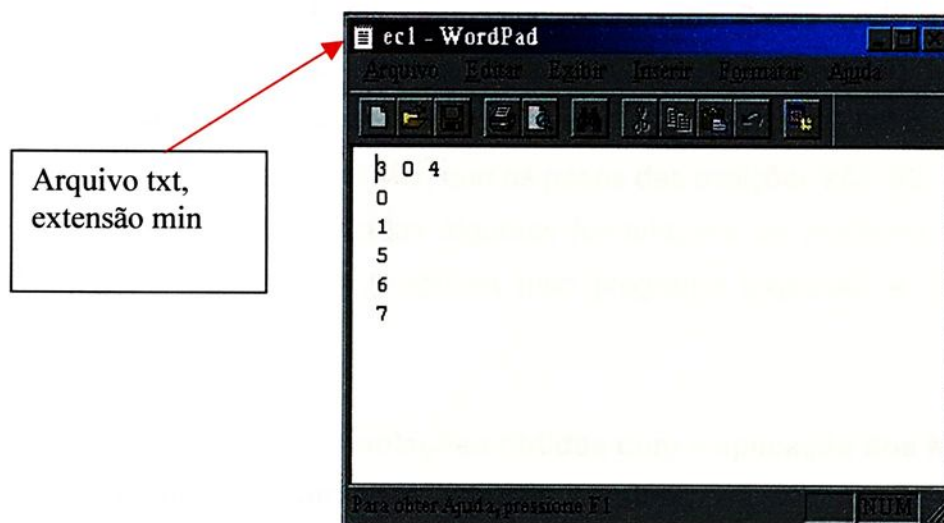


Figura 4.12 - Tela do arquivo de entrada do Programa Expander

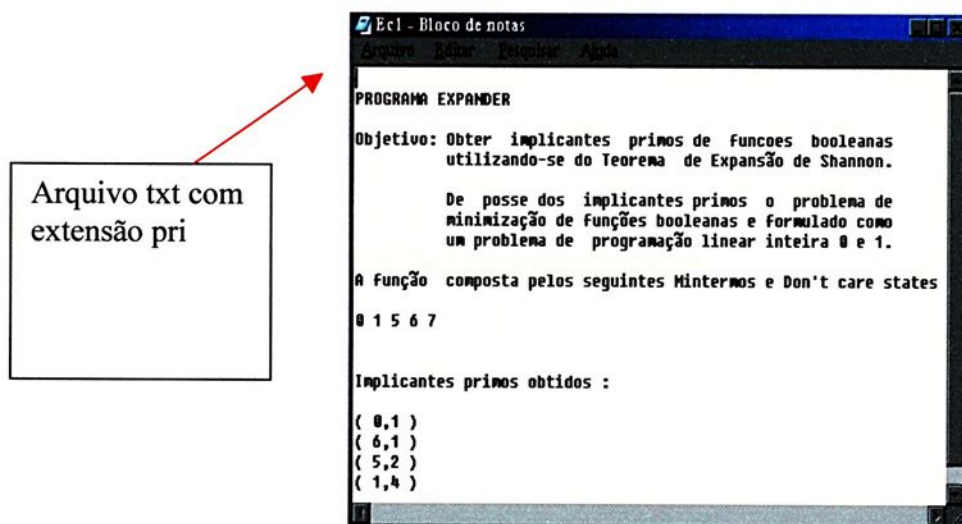


Figura 4.13 - Tela do arquivo de saída do Programa Expander

Observação: Na notação que foi utilizada para representar os implicantes primos obtidos, o termo (0,1) indica que o implicante primo, tem em sua representação binária um 0 como a soma dos pesos para as variáveis tidas como essenciais, e 1, como a soma dos pesos para as variáveis tidas como irrelevantes. Desse modo, a notação (0,1) corresponde à representação binária $000X = A'B'C'$. O termo (6,1) corresponde à representação binária $011X = A'BC$. Desse modo, (5,2) = $01X1 = A'BD$ e (1,4) = $0X01 = A'C'D$. Lembrando-se, mais uma vez, que os pesos das posições são: 32, 16, 8, 4, 2, 1.

A seguir serão apresentadas algumas formulações do problema de cobertura, formadas a partir de mintermos obtidos pelo programa Expander e pelo programa Pigene.

4.4 Comparação de formulações obtidas com a aplicação dos Métodos de Expansão de Shannon e Quine-McCluskey.

Caso 1

Para a função $f_1 = \sum m(0,3,11) + \sum d(1,2,13,15)$

Obteve-se os seguintes resultados:

PROGRAMA EXPANDER (Método de Expansão de Shannon)

(0,3) $\Rightarrow$ 00XX

(3,8) $\Rightarrow$ X011

Programa PIGENE (Método de Quine-McCluskey)

(3,8) $\Rightarrow$ X011

(11,4) $\Rightarrow$ 1 X11

(13,2) $\Rightarrow$ 11X1

(0,3) $\Rightarrow$ 00XX

Pode-se então, formular o problema matematicamente como um problema de programação linear inteira 0 e 1 da seguinte forma:

Formulação

Programa Expander

$$\text{Min } 3(00XX) + 4(X011)$$

Restrições

$$00XX \geq 1$$

$$X011 \geq 1$$

$$X011 + 00XX \geq 1$$

$$00XX, X011 \in \{0,1\}$$

Programa PIGENE

$$\text{Min } 3(00XX) + 4(X011 + 11X0 + 11X1)$$

restrições

$$00XX \geq 1$$

$$X011 + 00XX \geq 1$$

$$X011 + 11X1 \geq 1$$

$$00XX, X011, 11X0, 11X1 \in \{0,1\}$$

Para este caso nota-se a simplicidade da formulação para o Programa Expander onde se obtém a solução trivial $00XX=1$ e $X011=1$, por simples inspeção. Já o Programa PIGENE baseado em Quine-McCluskey possui solução mais complexa necessitando a aplicação de algum algoritmo de programação linear para resolvê-lo.

Caso 2

Para a função $f_2 = \sum m(4,8,9,15) + \sum d(0,11,12)$

Obteve-se os seguintes resultados:

PROGRAMA EXPANDER (Método de Expansão de Shannon)

$$(9,2) \Rightarrow 10X1$$

$$(11,4) \Rightarrow 1X11$$

$$(0,12) \Rightarrow XX00$$

Programa PIGENE (Método de Quine-McCluskey)

$$(8,1) \Rightarrow 100X$$

$$(9,2) \Rightarrow 10X1$$

$$(11,4) \Rightarrow 1X11$$

$$(0,12) \Rightarrow XX00$$

Formulação

Programa Expander

$$\text{Min } 3(XX00)+4(10X1+1X11)$$

restrições

$$XX00 \geq 1$$

$$10X1 \geq 1$$

$$1X11 \geq 1$$

$$XX00, 10X1, 1X11 \in \{0,1\}$$

Programa PIGENE

$$\text{Min } 3(XX00)+4(100X +10X1+1X11)$$

restrições

$$XX00 \geq 1$$

$$XX00+100X \geq 1$$

$$10X1+100X \geq 1$$

$$1X11 \geq 1$$

$$XX00, 100X, 10X1, 1X11 \in \{0,1\}$$

Este caso também tem solução trivial para o Método de Expansão de Shannon, pois tem-se $XX00=1$, $10X1=1$ e $1X11=1$. E para o método de Quine-McCluskey que gera todos os implicantes primos acrescenta-se $100X=0$ ao conjunto de implicantes primos.

Caso 3

Para a função $f_3 = \sum m(12,13) + \sum d(0,4,15)$

Obteve-se os seguintes resultados:

PROGRAMA EXPANDER (Método de Expansão de Shannon)

$$(13,2) \Rightarrow 11X1$$

$$(4,8) \Rightarrow X100$$

Programa PIGENE (Método de Quine-McCluskey)

$(0,4) \Rightarrow 0X00$

$(4,8) \Rightarrow X100$

$(12,1) \Rightarrow 110X$

$(13,2) \Rightarrow 11X1$

Formulação

Programa Expander

Min $4(11X1+X100)$

restrições

$11X1 \geq 1$

$X100 \geq 1$

$11X1, X100 \in \{0,1\}$

Programa PIGENE

Min $4(11X1+X100+0X00+X100)$

restrições

$11X1+110X \geq 1$

$X100+110X \geq 1$

$11X1, X100, 00X0, X100 \in \{0,1\}$

Mais uma vez obteve-se solução trivial para o Programa Expander, ou seja, $11X1=1$ e $X100=1$.

A condição necessária para se obter uma cobertura mínima é trabalhar com um universo contendo todos os implicantes primos da função. Desta forma pode-se obter todas as possíveis soluções de coberturas mínimas.

O algoritmo Expander elimina, no processo de expansão alguns dos implicantes primos diminuindo, dessa forma a obtenção de possíveis soluções de cobertura mínima.

Em se tratando de circuitos digitais as possíveis soluções correspondem a circuitos equivalentes, e de mesmo custo, podendo-se desta forma, utilizar-se de quaisquer das soluções.

Ainda que o método não garanta uma solução de custo mínimo, ele gera uma formulação mais simples e isso implica na facilidade na otimização por métodos matemáticos clássicos.

No próximo capítulo serão apresentados alguns resultados obtidos através dos dois algoritmos gerados.

Capítulo V

Resultados obtidos

5.1 Tabulação dos resultados

Para testar a eficiência dos algoritmos implementados, foram criadas várias funções booleanas. Os parâmetros utilizados para a comparação foram a quantidade de memória utilizada, tempo de execução e a quantidade de implicantes primos gerados.

Na figura 5.1 apresenta-se a quantidade de memória utilizada pelo Expander subtraída da quantidade de memória utilizada pelo PIGENE. O sinal de (-) foi utilizado somente para indicar que o expander utilizou menos memória do que o Pigene.

Pode-se notar que até seis variáveis o PIGENE utiliza menos memória aumentando exponencialmente a partir de sete, característica esta, já mencionada no Capítulo I.

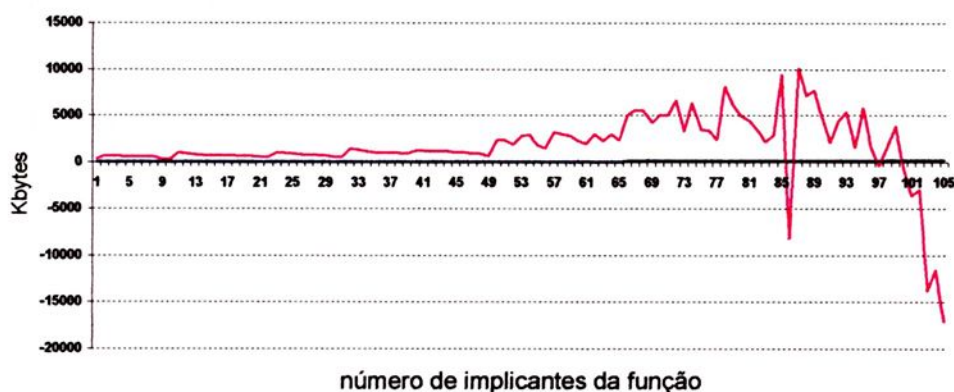


Figura 5.1. Gráfico comparativo de utilização de memória.

Pela figura 5.2 observa-se com clareza que a tendência exponencial do método de Quine-McCluskey não é seguida pelo método de expansão de Shannon.

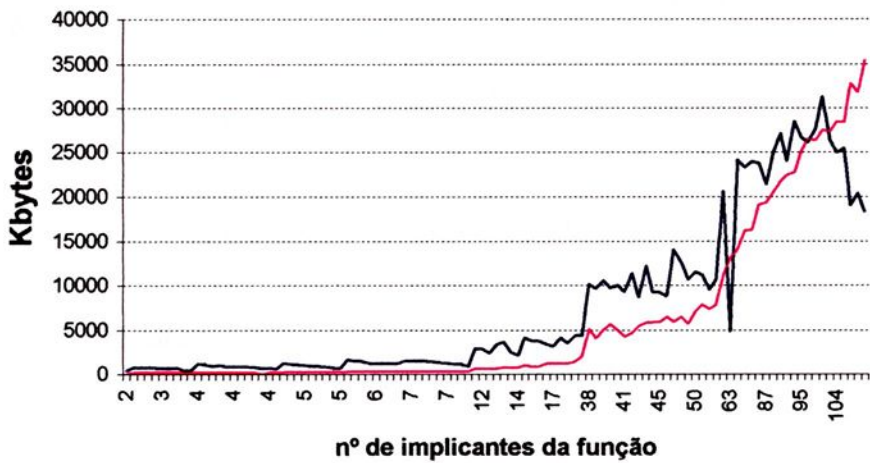


Figura 5.2. Gráfico de utilização de memória.

O que se observou pelos casos testados, é que a partir de 15 variáveis o Pigene começa a invadir a área de memória reservada ao sistema, não mais gerando resultados. Já, o Expander, gerou implicantes para uma função de 65536 mintermos, ou seja, uma função de 35 variáveis.

Na figura 5.3, pode se observar que o tempo consumido pelo Expander foi menor que o Pigene.

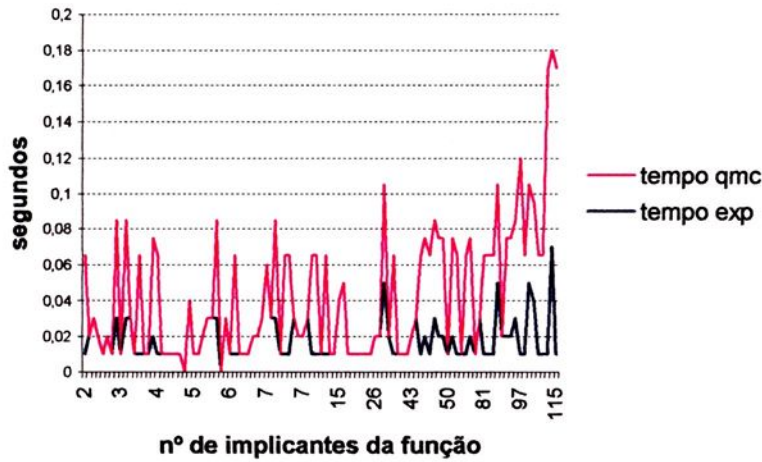


Figura 5.3 Gráfico comparativo de tempo gasto para a geração de implicantes primos por cada algoritmo

Na figura 5.4 apresenta-se um gráfico comparando o número de implicantes primos gerados pelos dois programas. Pode-se perceber que o número de implicantes primos gerados pelo programa Expander é igual ao gerado pelo PIGENE para funções com uma pequena quantidade de mintermos. À medida que se aumenta o número de mintermos, o programa Expander supera o Pigene, pois elimina alguns implicantes primos. O Pigene, no entanto, por basear-se no Método de Quine-McCluskey [4], sempre gera todos os implicantes.

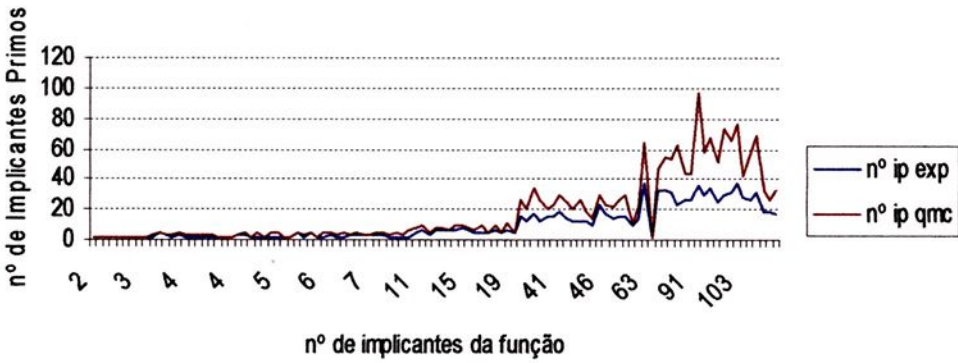


Figura 5.4. Gráfico do número de implicantes primos gerados por cada algoritmo.

5.2. Comparação da formulação do problema de cobertura utilizando-se o algoritmo Expander e Quine-McCluskey para geração de implicants primos.

Caso comparativo 1. para a função $f(A,B,C,D) = (0,1,2,5,7,8,9,10,13,15)$ obteve-se os seguintes resultados:

Quine-McCluskey	Expander
Imp. primos $X00X, X0X0, X1X1, XX01$	Imp. primos $X0X0, X1X1, XX01$
$\min 3(X00X+X0X0+X1X1+XX01)$	$\min 3(X0X0+X1X1+XX01)$
$X00X + X0X0 \quad 1$	$X0X0 \quad 1$
$X00X + XX01 \quad 1$	$XX01 \quad 1$
$X0X0 \quad 1$	$XX01 + X1X1 \quad 1$
$XX01 + X1X1 \quad 1$	$X1X1 \quad 1$
$X1X1 \quad 1$	
$X00X, X0X0, X1X1, XX01 \in \{0,1\}$	$X0X0, X1X1, XX01 \in \{0,1\}$

Deste exemplo duas considerações podem ser feitas a partir da formulação:

- O número de restrições foi menor para o Expander, pois ao eliminar o implicants primo $X00X$, eliminou também uma restrição. Lembrando que cada mintermo gera uma restrição.
- Para a função exemplo a solução tornou-se trivial para a formulação gerada pelo algoritmo Expander.

A solução mínima obtida é $X0X0=1, X1X1=1, XX01=1$.

Caso comparativo 2.

$$f(A,B,C,D,E) = \sum m(1,4,6,10,20,22,24,26) + d(0,11,16,27)$$

Quine-McCluskey

Imp. primos 0000X, 1X000, X01X0,
X0X00, X101X, 110X0

$$\min 5(0000X+1X000+110X0) + \\ 4(X01X0+X0X00+X101X+110X0)$$

$$0000X \geq 1$$

$$X01X0 \geq 1$$

$$X101X \geq 1$$

$$X01X0 + X0X00 \geq 1$$

$$1X000 + 110X0 \geq 1$$

$$X101X + 110X0 \geq 1$$

$$X00X, X0X0, X1X1, XX01 \in \{0,1\}$$

Expander

Imp. primos 0000X, 1X000, X01X0,
X0X00, X101X

$$\min 5(0000X+1X000+110X0) + \\ 4(X01X0+X0X00+X101X)$$

$$0000X \geq 1$$

$$X01X0 \geq 1$$

$$X101X \geq 1$$

$$X01X0 + X0X00 \geq 1$$

$$1X000 \geq 1$$

$$X0X0, X1X1, XX01 \in \{0,1\}$$

Novamente obteve-se uma solução trivial a partir da formulação pelo programa Expander 0000X = 1, X01X0 = 1, X0X00 = 0, X101X = 1 e 1X000 = 1, para o programa Pigene será preciso recorrer a algum método matemático para obter a solução mínima global.

Caso comparativo 3.

$f(A,B,C,D,E) = \sum m(7,8,9,12,13,14,19,23,24,27,29,30) + d(1,10,17,26,28,31)$

Quine-McCluskey

Imp. primos 0X001, 1XX11, X0111, X110X,
X1XX0, X0001, 100X1, 01X0X,
11X1X, 111XX

min 5(0X001+ X0111+ X1101+
X0001+100X1) +
4(1XX11+01X0X+11X1X+111XX)+
3(X1XX0)

- ST
- X0111 ≥ 1
 - X1XX0 + 01X0X ≥ 1
 - 0X001+ 01X0X ≥ 1
 - X110X + X1XX0 + 01X0X ≥ 1
 - X110X + 01X0X ≥ 1
 - X1XX0 ≥ 1
 - 1XX11 + 100X1 ≥ 1
 - 1XX11 X0111 ≥ 1
 - 1XX11 + 11X1X ≥ 1
 - X110X + 111XX ≥ 1
 - X1XX0 + 11X1X + 111XX ≥ 1

0X001, 1XX11, X0111, X110X, X1XX0,
X0001, 100X1, 01X0X, 11X1X, 111XX ∈
{0,1}

Expander

Imp. primos 0X001, 1XX11, X0111, X110X,
X1XX0

min 5(0X001+ X0111) + 4(1XX11+X110X)+
3(X1XX0)

- ST
- X0111 ≥ 1
 - X1XX0 ≥ 1
 - 0X001 ≥ 1
 - X110X + X1XX0 ≥ 1
 - X110X ≥ 1
 - 1XX11 ≥ 1
 - X0111+1XX11 ≥ 1

0X001, 1XX11, X0111, X110X, X1XX0 ∈
{0,1}

Fazendo-se uma simples inspeção, obtém-se no caso formulado utilizando o programa Expander a solução $0X001=1$, $1XX11=1$, $X0111=1$, $X110X=1$, $X1XX0=0$. Para o outro problema, formulado pelo método de Quine-McCluskey, é necessário utilizar alguma técnica de programação linear para resolvê-lo e ainda que o exemplo seja uma função modesta, deixa bem clara a eficiência do algoritmo que utiliza o método de expansão de Shannon.

Um outro comentário, é que o Programa Expander trabalha de forma mais eficiente com funções que contém don't care states, pois a marca de início, pode levar a eliminação de filas inteiras durante a aplicação do algoritmo.

Já para o programa PIGENE, que segue na íntegra o método manual tabular de Quine-McCluskey gerando todos os implicantes primos, seria necessário partir para a segunda fase do método que é conhecida como fase de cobertura, para reduzir a redundância de implicantes primos.

Os resultados que foram gerados com as funções utilizadas na comparação dos algoritmos estão apresentados no Anexo C.

Os Anexos A e B contém os arquivos fontes dos programas PIGENE e EXPANDER respectivamente.

No Capítulo VI apresentam-se as conclusões e os comentários finais.

Capítulo VI

Conclusões e Comentários

Este trabalho de pesquisa teve como principal objetivo a busca de métodos eficientes para a geração dos implicantes primos de uma função booleana. De posse dos implicantes primos gerados, a cobertura da função pode ser formulada como um problema de programação linear inteira 0 e 1.

Implementou-se, em programas de computador, uma técnica iterativa e manual, inicialmente proposta por Scheinman, que gera um conjunto reduzido de implicantes primos, porém o suficiente para se obter a solução de custo mínimo local.

Utilizou-se no desenvolvimento do algoritmo a teoria de expansão de Shannon em que todas as variáveis da função são expandidas, uma de cada vez, sendo que da expansão de uma variável obtém-se duas outras funções resíduo, que independem das variáveis expandidas.

A estrutura de dados utilizada em forma de diagrama de decisão binária é de extrema complexidade e de difícil programação, entretanto, o algoritmo utiliza somente duas operações aritméticas, a subtração e a potenciação, e uma operação lógica, a igualdade.

O algoritmo implementado, denominado Expander, foi comparado com o método de Quine-McCluskey, que também foi implementado em um programa denominado Pigene. Para avaliar a eficiência do Expander, utilizaram-se como parâmetros o consumo de memória, o tempo de execução e a quantidade de implicantes gerados. Verificou-se também a quantidade máxima de variáveis que uma função pode conter.

Pode-se constatar que o problema de cobertura gerada pelo Expander é bem mais simplificado do que a gerada pelo programa Pigene. Muitas das formulações têm soluções triviais.

Pode-se constatar que o problema de cobertura gerada pelo Expander é bem mais simplificado do que a gerada pelo programa Pigene. Muitas das formulações têm soluções triviais.

Com relação ao uso de memória pode-se notar que o Programa Expander consumiu menos memória, pois conseguiu-se gerar implicantes primos para funções de até 100 variáveis.

Em relação ao tempo de processamento verificou-se, que o Expander conclui o processo de geração de implicantes primos em um tempo bastante curto, sendo insignificante para funções com poucos mintermos.

Neste trabalho gerou-se uma alternativa eficiente para a obtenção de implicantes primos de uma função booleana, a partir dos quais o problema de cobertura pode ser formulado através de um problema de programação linear inteira 0 e 1.

Como continuação desse projeto sugere-se a comparação do Expander com programas que implementam algoritmos para a geração de implicantes primos através das aplicações sucessivas da operação de consenso aos ramos de uma árvore, cujos caminhos (nó até a folha) representam mintermos ou don't care state da função.

Pode-se também estudar a possibilidade de estender o algoritmo ora desenvolvido para tratar funções com múltiplas saídas.



Referências Bibliográficas

- [1] SHANNON, C. E. "The synthesis of two-terminal switching circuits", Bell Sys. Tech. J., 1948
- [2] QUINE, W., "The Problem of Simplifying Truth Functions", American Mathematical Monthly, Vol. 59, pp. 521-531, 1952.
- [3] MCCLUSKEY E., "Minimization of Boolean Function", The Bell System Technical Journal, Vol. 35, pp. 1417-1444, November, 1956.
- [4] MCCLUSKEY E. J., Logic Design Principles, Prentice Hall, N.J., 1986.
- [5] SILVA, A.C.R., Contribuição à Síntese de Circuitos Digitais Utilizando Programação Linear Inteira 0 e 1, Tese de Doutorado, UNICAMP, 1993.
- [6] SILVA, A.C.R., Contribuição à Minimização e Simulação de Circuitos Lógicos, Tese de Mestrado, UNICAMP, 1989.
- [7] BRAYTON, K. B., HACHTEL, G. D., MCMULLEN, C. T., SANGIOVANNI-VICENTELLI A. L., Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publisher, 1984.
- [8] HONG, S.J. CAIN, R. G. and OSTAPKO, D.L. "MINI: A heuristic Approach for logic minimization," IBM J of Res. And Dev., Vol. 18, pp. 443-458, September 1974
- [9] BROWN, D. W. "A State-Machine Synthesizer – SMS" Proc. 18th Design Automation Conference. pp 301-304 Nashville, june 1981
- [10] FALLAH, F. LIAO, S and DEVADAS, S. "Solving covering Problems Using LPR-Based Lower Bounds", IEEE Transactions on Very Large Scale Integration (VLSI) Systems. Vol. 8. february 2000
- [11] MEINEL, C.; SLOBODOV, A. "Unifying theoretical background for some BDD-based data structures", Formal Methods in System Design, October, 1997, v11 i3, p. 223-237.
- [12] MEINEL, CHRISTOPH; SLOBODOVA ANNA, "Speeding up variable reordering of OBDDs", Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processor, 1997, p. 338-343
- [13] LONG, WANGNING; MIN, YINGHUA; YANG, SHIYUAN; TONG, SHIBAI, "Short-time scaling of variable ordering of OBDDs", Journal of Computer Science and Technology, July, 1997, v12 i4, p. 366-371.

- [14] THORNTON, M.A., "Modified Haar transform calculation using digital circuit output probabilities", Proceedings of the International Conference on Information, Communications and Signal Processing, ICICS, 1997, v1, p. 52-58.
- [15] DRECHSLER, R.; BECKER, B.; GOECKEL, N., "Genetic algorithm for variable ordering of OBDDs", IEE Proceedings: Computers and Digital Techniques, November, 1996, v 143 i6, p. 364-368.
- [16] SAUERHOFF, MARTIN; WEGENER, INGO, "On the complexity of minimizing the OBDD size for incompletely specified functions", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, November, 1996, v 15, i11, p. 1435-1437.
- [17] LIAW, HEH-TYAN, LIN, CHENG-SHANG, "On the OBDD-representation of general Boolean functions", IEEE Transaction on Computers, June, v 4 i6, p. 661-664.
- [18] BRYANT, RANDAL E., "Symbolic Boolean manipulation with Ordered Binary-Decision Diagrams", ACM Computing Surveys, September, 1992, v 24 i3, p. 293-318.
- [19] LEE C., "Representation of Switching Circuits by Binary Decision Programs ", Bell System Technical Journal, Vol. 38, pp. 985-999, July, 1959.
- [20] AKERS S., " Binary Decision Diagram ", IEEE Transactions on Computers, Vol. C-27, pp. 509-516, June, 1978
- [21] MICHELI, G. DE, "Synthesis and Optimization of Digital Circuits", McGraw-Hill Inc, 1994
- [22] KOAVI, SVI, "Switching and Finite Automata Theory", McGraw-Hill Inc, 1978
- [23] SCHEINMAN, A. H. , "A Method for Simplifying Boolean functions", The Bell System Technical Journal, july, 1962, pp 1337-1346.
- [24] RUDELL R., SANGIOVANI-VICENTELII A, "Multiple-Valued Minimization for PLA Optimization", IEEE Transactions on CAD/ICAS, Vol. CAD-6, n°5, September, 1987, pp. 727-750.
- [25] DAGENAIS M., AGARWAL V., RUMIN N., "McBoole: A New Procedure for exact Logic Minimization", IEEE Transactions on CAD/ICAS, Vol. CAD-5 N° 1, Jan 1986, pp. 229-223.
- [26] COUDERT O, MADRE J., "Implicit and Incremental Computation of Primes and Essential Primes of Boolean Function", DAC, Proceeding of the Design Automation Conference, 1992, pp. 36 – 39.
- [27] COUDERT O, MADRE J, FRAISSE H., "A New Viewpoint on Two-Level Logic Minimization", DAC, Proceeding of the Design Automation Conference, 1993, pp. 625 – 630.

- [28] BRAYTON R., SOMENZI F., "An Exact Minimizer for Boolean Relations", ICCAD, Proceedings of the International Conference on Computer Aided Design, 1989, pp. 316 - 319.
- [29] WATANABE Y., BRAYTON R., "Heuristic Minimization of Multiple-valued Relation", IEEE Transactions on CAD/ICAS, Vol. CAD-12, No. 10, October, 1993, pp.1458 – 1472.
- [30] WANG Y., MCCROSKY C., "Solving Boolean Equations Using ROSOP Forms", IEEE Transactions on computers, Vol. 47, No. 2, February, 1998, pp. 171 – 177.
- [31] HEINRICH-LITAN L., MOLITOR P., "Least Upper bounds for the Size of OBDDs Using Symmetry Properties IEEE Transactions on computers, April 2000(Vol. 49, N° 4) , pp. 360-368.
- [32] WU YU-LIANG, FAN HONGBING, MAREK-SADOWSKA M., WONG C. K.." OBDD Minimization Based on Two-level Representation of Boolean Functions. IEEE Transactions on computers, Dec. 2000(Vol. 49, N° 12), pp 1371-1379.
- [33] DUBROVA E., MACCHIARULO L., "A Comment on 'Graph-based Algorithm for Boolean Function Manipulation IEEE Transactions on computers, Nov. 2000(Vol. 49, N° 11), pp. 1290-1292.
- [34] MCGEER P., SANGHAVI J., BRAYTON R., VINCENTELLI A. S., "Espresso Signature: A New Exact Minimizer for Logic Functions", IEEE Transactions on computers, Dec 2001.
- [35] SASAO T., BUTLER J. T., "Worst and Best Irredundant Som-of-Products Expressions, IEEE Transactions on computers Sep 2001(Vol. 50, N° 9), pp. 935-948.
- [36] BERNASCONI A., CODENOTTI B., VANDERKAM J. M., " A Characterization of Bent Functions in terms of Strongly Regular Graphs, IEEE Transactions on computers, Sep 2001(Vol 50, N° 9), pp. 984-985.
- [37] JAIN J., WEGENER I., MASASHIRO F., "A Note on Complexity of OBDD composition and Efficiency of Partioned-OBDDs over OBDDs", IEEE Transactions on computers, Nov. 2001(Vol 50, N° 11), pp. 1289-1290.
- [38] J.HLAVICKA, and P. FISER, BOOM – a Heuristic Boolean Minimizer. Proc. ICCAD-2001, San Jose, Cal. (USA), 4.-8.11.2001 (accepted for publication).
- [39] J.HLAVICKA, and P. FISER, A Heuristic method of two-level logic synthesis. Proc. The 5th World Multiconference on Systemics, Cybernetics and Informatics SCI'2001, Orlando, Florida (USA) 22-25.7.2001, vol. XII, pp. 283-288.
- [40] J P. FISER, and J.HLAVICKA, On the Use of Mutations in Boolean Minimization. Proc. Euromicro Symposium on Digital Systems Design, Warsaw (Poland) 4-6.9.2001, pp. 300-307.

*Universidade Estadual Paulista "Júlio de Mesquita Filho".
Departamento de Engenharia Elétrica
Área de Automação e Controle*

/ Programa: PIGENE.C*

*Objetivo: Obtém todos os implicantes primos de uma dada função booleana
aplicando iterativamente o Método de Quine-McCluskey.*

Programador: Fábio Ricardo Pontes Emer

Versão: 6.0

**/*

```
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <time.h>
#include <dos.h>
#include <conio.h>
```

```
FILE *le, *escreve;
char ler_arq[12], esc_arq[12];
int unmatched_bit, flag_master;
double start, end, ac_mem, max_mem=0, aux_max_mem=0;
```

/ Define estruturas */*

```
struct min
{
    int box;
    double minterm;
    int flag_sort;
    int flag_check;
    int pos_var_elim;
    struct min *next_min;
    struct min *last_min;
}size_min;
```

```
struct prime
{
    double prime_im;
    double irr_var;
    struct prime *next_pri;
}size_prime;
```

/ Apontadores para estruturas */*



```
struct min *new_min, *aux_min, *head, *tail, *haux, *taux, *aux, *aux1, *aux2;  
struct prime *new_pri, *head_p, *tail_p;
```

```
/* Subrotinas utilizadas */
```

```
/* FUNÇÕES PARA ALOCAÇÃO DE MEMÓRIA */
```

```
alloc_mem_min()  
{  
    struct min *min_var;  
    char *malloc();  
    min_var = (struct min *)malloc(sizeof(size_min));  
    if (min_var == 0)  
    {  
        printf("\n memória insuficiente");  
        return;  
    }  
    ac_mem+=sizeof(size_min);  
    new_min=min_var;  
    new_min->next_min = NULL;  
    new_min->last_min = NULL;  
    new_min->flag_check=0;  
    new_min->flag_sort=0;  
}  
  
alloc_mem_pri()  
{  
    struct prime *pri_var;  
    char *malloc();  
    pri_var = (struct prime*)malloc(sizeof(size_prime));  
    if (pri_var == 0)  
    {  
        printf("\n memória insuficiente");  
        return;  
    }  
    max_mem+=sizeof(size_prime);  
    new_pri=pri_var;  
    new_pri->next_pri = NULL;  
}
```

```
/* Função para contar o número de "uns" */
```



```
open_f()
{
char arq[8];

/* Pega o nome do arquivo a ser lido */
printf("MÉTODO DE QUINE-McCLUSKEY - PROGRAMA EXPANDER\n\n");
printf("Entre com o nome do arquivo para leitura '.min'\n ");
gets(arq);
start = clock()/CLK_TCK;
/* Copia o conteúdo da variável arq para a variável ler_arq */
strcpy(ler_arq,arq);

/* Acrescenta a extensão .MIN ao nome do arquivo */
strcat(ler_arq, ".min");

/* Abre o arquivo para leitura */
if((le = fopen(ler_arq, "r")) == NULL)
{
printf(" Falha na abertura do arquivo de leitura \n");
printf("passou aqui\n");
printf("\n %s", ler_arq);
sleep(2000);

exit(0);
}

/* Copia o conteúdo da variável arq para a variável esc_arq */
strcpy(esc_arq, arq);

/* Acrescenta a extensão .qmc ao nome do arquivo */
strcat(esc_arq, ".qmc");

/* Abre o arquivo para escrita */
if((escreve = fopen(esc_arq, "w")) == NULL)
{
printf(" Falha na abertura do arquivo de escrita \n");
exit(0);
}
}

count_one(min)
double min;
{
int countone = 1;
int aux, resto;
if( min == 0)
return(0);
aux = min;
```




```

while( aux > 1)
{
    resto = aux;
    aux /= 2;
    resto %= 2;
    if (resto == 1)
        ++countone;
}
return(countone);
}

```

```
double stringtodouble(s)
```

```
char s[];
```

```

{
    double val,pot;
    int i, sinal;

    for(i=0;s[i]!=' '|| s[i]!='\n' || s[i]!='\t';i++)
        /*pula espaço em branco*/
    sinal=1;
    if (s[i]=='+'|| s[i]=='-')/*sinal*/
        sinal=(s[i++]=='-')?1:-1;
    for (val=0;s[i]>='0' && s[i]<='9';i++)
        val=10*val+s[i]-'0';
    if(s[i]=='.')
        i++;
    for(pot=1;s[i]>='0'&& s[i]<='9';i++)
    {
        val=10*val + s[i]-'0';
        pot*=10;
    }
    return(sinal * val/pot);
}

```

```
/* FUNÇÃO PARA LER OS MINTERMOS E MONTAR A PRIMEIRA TABELA */
```

```
read_min()
```

```

{
    double aux,min_var;
    char line[10];
    fgets(line,10,le);
    while(fgets(line,10,le)!=NULL)
    {
        min_var = stringtodouble(line);
        aux = count_one(min_var);

        alloc_mem_min();
    }
}

```

```
if(head == NULL)
{
    tail = head = new_min;
    head->box=aux;
    head->minterm=min_var;
    head->pos_var_elim = 0;
}
else
{
    tail->next_min=new_min;
    new_min->last_min=tail;
    new_min->box=aux;
    new_min->minterm=min_var;
    new_min->pos_var_elim = 0;
    tail = new_min;
}
}
}
/* FUNÇÃO PARA ORDENAR A ENTRADA */
/* pois os don't care states podem não estar em ordem com os mintermos */

sort_data()
{
    haux=taux=NULL;
    while(head!=NULL)
    {
        aux_min=head;
        tail=head->next_min;
        while(tail!=NULL)
        {
            if (aux_min->minterm<tail->minterm)
            {
                tail=tail->next_min;
            }
            else
            {
                aux_min=tail;
                tail=tail->next_min;
            }
        }
    }
    alloc_mem_min();
    if(haux == NULL)
    {
        taux = haux = new_min;
        haux->box=aux_min->box;
        haux->minterm=aux_min->minterm;
        haux->pos_var_elim = 0;
    }
}
```

```

else
{
    taux->next_min=new_min;
    new_min->last_min=taux;
    new_min->box=aux_min->box;
    new_min->minterm=aux_min->minterm;
    new_min->pos_var_elim = 0;
    taux= new_min;
}
if (aux_min==head)
    head=head->next_min;
else
{
    aux_min->last_min->next_min=aux_min->next_min;
    if(aux_min->next_min!=NULL)
        aux_min->next_min->last_min=aux_min->last_min;
}

free(aux_min);

if(ac_mem>=max_mem)
{
    max_mem=ac_mem;
    ac_mem-=sizeof(size_min);
}
}
head=haux;
}

```

/ FUNÇÃO PARA ORDENAR A PRIMEIRA TABELA*/*

```

sort_tab()
{
    haux=taux=aux=NULL;
    tail=head;
    while( tail!=NULL)
    {
        aux=tail->next_min;
        while(aux!=NULL)
        {
            if(tail->box<=aux->box)
                aux=aux->next_min;
            else
            {
                aux1=aux;
                make_sort();
                if (aux==NULL)
                    break;
            }
        }
    }
}

```



```

        aux=aux->next_min;
    }
}
aux1=tail;
make_sort();
tail=tail->next_min;
free(aux1);
ac_mem-=sizeof(size_min);
}
taux=haux=NULL;
}

```

/ FUNÇÃO PARA ALOCAR MEMÓRIA PARA A TABELA ORDENADA */*

```

make_sort()
{
    int box_aux,box;
    box=taux->box;
    alloc_mem_min();
    if (haux==NULL)
    {
        head=taux=haux=new_min;
        box=-1;
    }
   iaux->next_min=new_min;
    box_aux=new_min->box=iaux->box;
    new_min->minterm=iaux->minterm;
    new_min->flag_check=iaux->flag_check;
    new_min->pos_var_elim=iaux->pos_var_elim;

    if (iaux->flag_sort==0)
    {
        if (box!=box_aux)
        {
            box=box_aux;
        }
       iaux->flag_sort=1;
       iaux=iaux->next_min;
    }
}

```

/ FUNÇÃO DE COMPARAÇÃO */*

```

comp_min()
{
    int min1,min2,box1,box2;
    struct min *era;

```

```

while (head!=NULL)
{
tail=head->next_min;
box1=head->box; ;
min1=head->minterm;
while ( tail!=NULL)
{
box2=tail->box;
min2=tail->minterm;
if (box2==box1+1)
{
if (head->pos_var_elim==tail->pos_var_elim)
comp_aux(min1,min2);
tail=tail->next_min;
}
else if (box2==box1)
tail=tail->next_min;
else
break;
}
make_pri();
era=head;
head=head->next_min;
free(era);
if (ac_mem>=max_mem)
{
max_mem=ac_mem;
ac_mem-=sizeof(size_min);
}
}
}

```

*/*FUNÇÃO PARA CRIAR FILA PRIMOS*/*

```

make_pri()
{
if (head->flag_check==0)
{
alloc_mem_pri();
if (head_p==NULL)
head_p=tail_p=new_pri;
else
tail_p->next_pri=new_pri;
new_pri->prime_im=head->minterm;
new_pri->irr_var=head->pos_var_elim;
tail_p=new_pri;
}
}

```

/ FUNÇÃO DE COMPARAÇÃO AUXILIAR*/*



```

comp_aux(min1,min2)
int min1,min2;
{
int a,b,aux,pos;
pos=match(min1,min2);
if (unmatch_bit==1)
{
aux2=haux;
while(aux2!=NULL)
{
if(aux2->minterm==min1 && aux2->pos_var_elim==(pos+tail->pos_var_elim))
{
head->flag_check=1;
tail->flag_check=1;
return;
}
aux2=aux2->next_min;
}
head->flag_check=1;
tail->flag_check=1;
alloc_mem_min();
flag_master=1;
if (haux==NULL)
    haux=taux=new_min;
else
    taux->next_min=new_min;
new_min->box=(tail->box)-1;
new_min->minterm=min1;
new_min->pos_var_elim=pos+tail->pos_var_elim;
taux=new_min;
}
}

```

*/*FUNÇÃO NÚCLEO DE COMPARAÇÃO*/*

```

match(aux1,aux2)
int aux1,aux2;
{
int aux,resto1,resto2,pos=1,pos_aux;
if (aux1>aux2)
{
aux=aux1;
aux1=aux2;
aux2=aux;
}
unmatch_bit=0;
while(aux2!=0)
{
resto1=aux1%2;

```




```

    resto2=aux2%2;
    aux1/=2;
    aux2/=2;
    if (resto1==resto2)
        pos*=2;
    else
    {
        unmatched_bit++;
        pos_aux=pos;
    }
}
return(pos_aux);
}

outprint_1()
{
    fprintf(escreve,"PROGRAMA PlsGene");
    fprintf(escreve,"\n\n Objetivo: Obter implicantes primos de funções booleanas");
    fprintf(escreve,"\n utilizando-se do Método de Quine-McCluskey.");
    fprintf(escreve,"\n\n A FUNÇÃO , composta pelos\n");
    fprintf(escreve,"seguintes Mintermos e Dont't care states\n\n");
    tail=head;
    while(tail!=NULL)
    {
        fprintf(escreve,"%0f ",tail->minterm);
        tail=tail->next_min;
    }
}

outprint_2()
{
    fprintf(escreve,"\n\n Implicantes Primo obtidos por QMC\n");
    tail_p=head_p;
    while(tail_p!=NULL)
    {
        fprintf(escreve,"\n (%0f,%0f)",tail_p->prime_im,tail_p->irr_var);
        tail_p=tail_p->next_pri;
    }
}

end=clock()/CLK_TCK;
fprintf(escreve,"\n\ntempo decorrido: %.9f seconds\n",end-start);
fprintf(escreve,"\nmemória total utilizada: %.0f bytes",max_mem);
printf("\n End of Program.....Press Any Key \n");
getch();
}

/* Função para fechar todos os arquivos */
close_file()
{
    fclose(le);
}

```



```
fclose(escreve);
}

main()
{
open_f();
read_min();
outprint_1();
sort_data();
taux=haux;
sort_tab();
fprintf(escreve, "\n");
flag_master=1;
while (flag_master==1)
{
flag_master=0;
comp_min();
head=tail=haux;
haux=taux=NULL;
}
outprint_2();
close_file();
/*clrscr();*/
}
```



/*

*Universidade Estadual Paulista "Júlio de Mesquita Filho".
Departamento de Engenharia Elétrica
Área de Automação e Controle*

Programa: Expander.c

*Objetivo: Obtém todos os implicantes primos de uma dada função booleana
aplicando iterativamente o Método de Expansão de Shannon.*

A estrutura de dados utilizada em forma de DDB (Diagrama de Decisão Binária).

Programador: Fábio Ricardo Pontes Emer

Versão: 4.2.1

*/

```
#include <stdio.h>
#include <io.h>
#include <string.h>
#include <time.h>
#include <dos.h>
#include <conio.h>
```

```
FILE *le, *escreve;
char ler_arq[12], esc_arq[12];
double start, end, ac_mem, max_mem=0, aux_max_mem=0;
```

/ Define-se a estrutura de dados para representar os mintermos e os
implicantes na estrutura de árvore*

```
*/
struct Dec
{
    double decimal;
    int marca;
    struct Dec *prox_dec;
} size_Dec;
```

/ Define-se a estrutura para os n da rvore de deciso binria */*

```
struct DDB
{
    double peso;
    struct Dec *min_dec;
    struct DDB *var_neg;
    int flag_neg;
    struct DDB *var_verd;
    int flag_verd;
```



```
struct DDB *var_irrel;
int flag_irrel;
struct DDB *anterior;
} size_DDB;

/* Define a estrutura da fila que armazena as variáveis que podem ser expandida */
struct Fila_var
{
    struct DDB *var_expan;
    struct Fila_var *prox_var;
} size_fila;

/* Define a estrutura da fila que armazena os implicantes primos */
struct Fila_primo
{
    struct DDB *variavel;
    struct Fila_primo *prox_primo;
} size_fila_primo;

/* Define a estrutura contendo os implicantes primos com notação decimal */
struct Fila_final
{
    double verdade;
    double dont;
    struct Fila_final *prox_termo;
} size_fila_final;

/* Define os apontadores para as estruturas */
struct DDB *raiz, *new_ddb;
struct Dec *new_dec, *aux_dec;
struct Fila_var *new_fila, *cabeca, *rabo;
struct Fila_primo *new_fila_primo, *cabeca_primo, *rabo_primo;
struct Fila_final *new_fila_final, *cabeca_final, *rabo_final;

int V_peso;
int Num_var;
int Num_dont;

int neg_flag_fim_neg = 0;
int neg_flag_fim_verd = 0;
int neg_flag_fim_irrel = 0;

int verd_flag_fim_neg = 0;
int verd_flag_fim_verd = 0;
int verd_flag_fim_irrel = 0;
```



```
int irrel_flag_fim_neg = 0;
int irrel_flag_fim_verd = 0;
int irrel_flag_fim_irrel = 0;

abre_arq()
{

    char arq[8];

    /* Pega o nome do arquivo a ser lido */

    printf("METODO DE EXPASAO DE SHANNON - PROGRAMA EXPANDER\n\n");
    printf("Entre com o nome do arquivo para leitura '.min'\n ");
    gets(arq);
    start = clock()/CLK_TCK;
    /* Copia o conteudo da variavel arq para a variavel ler_arq */
    strcpy(ler_arq, arq);

    /* Acrescenta a extensao .MIN ao nome do arquivo */
    strcat(ler_arq, ".MIN");

    /* Abre o arquivo para leitura */
    if((le = fopen(ler_arq, "r")) == NULL)
    {
        printf(" Falha na abertura do arquivo de leitura \n");
        exit(0);
    }

    /* Copia o conteudo da variavel arq para a variavel esc_arq */
    strcpy(esc_arq, arq);

    /* Acrescenta a extensao .pri ao nome do arquivo */
    strcat(esc_arq, ".pri");

    /* Abre o arquivo para escrita */
    if((escreve = fopen(esc_arq, "w")) == NULL)
    {
        printf(" Falha na abertura do arquivo de escrita \n");
        exit(0);
    }
}

fecha_arq()
{
    fclose(le);
    fclose(escreve);
    printf("\n programa finalizado ... Pressione alguma tecla \n");
    getch();
}
```



```
aloca_mem_DDB()
{
    struct DDB *info_DDB;
    char *malloc();

    info_DDB = (struct DDB *) malloc(sizeof(size_DDB));
    ac_mem+=sizeof(size_DDB);
    if (info_DDB == 0)
    {
        printf("\n O Sistema não dispõe de memória -* ESTRUTURA DDB *-");
        return;
    }
    new_ddb = info_DDB;
    new_ddb->min_dec = NULL;
    new_ddb->var_neg = NULL;
    new_ddb->var_verd = NULL;
    new_ddb->var_irrel = NULL;
    new_ddb->anterior = NULL;
    new_ddb->flag_neg = 0;
    new_ddb->flag_verd = 0;
    new_ddb->flag_irrel = 0;
}

aloca_mem_Dec()
{
    struct Dec *info_Dec;
    char *malloc();

    info_Dec = (struct Dec *) malloc(sizeof(size_Dec));
    ac_mem+=sizeof(size_Dec);
    if (info_Dec == 0)
    {
        printf("\n O Sistema não dispõe de memória -* ESTRUTURA DECIMAL *-");
        return;
    }
    new_dec = info_Dec;
    new_dec->prox_dec = NULL;
}

aloca_mem_fila()
{
    struct Fila_var *info_fila;
    char *malloc();
    info_fila = (struct Fila_var *) malloc(sizeof(size_fila));
    ac_mem+=sizeof(size_fila);
```




```
if (info_fila == 0)
{
    printf("\n O Sistema não dispõe de memória -* ESTRUTURA FILA EXPANDIR *-");
    return;
}
new_fila = info_fila;
new_fila->var_expan = NULL;
new_fila->prox_var = NULL;
}

aloca_mem_primo()
{
    struct Fila_primo *info_fila_primo;
    char *malloc();

    info_fila_primo = (struct Fila_primo *) malloc(sizeof(size_fila_primo));
    ac_mem+=sizeof(size_fila_primo);
    if (info_fila_primo == 0)
    {
        printf("\n O Sistema não dispõe de memória -* ESTRUTURA IMPLICANTES PRIMOS *-");
        return;
    }
    new_fila_primo = info_fila_primo;
    new_fila_primo->variavel = NULL;
    new_fila_primo->prox_primo = NULL;
}

aloca_mem_fila_final()
{
    struct Fila_final *info_fila_final;
    char *malloc();

    info_fila_final = (struct Fila_final *) malloc(sizeof(size_fila_final));
    ac_mem+=(sizeof(size_fila_final));
    if (info_fila_final == 0)
    {
        printf("\n O Sistema não dispõe de memória -* FILA FINAL *- ");
        return;
    }
    new_fila_final = info_fila_final;
    new_fila_final->verdade = 0;
    new_fila_final->dont = 0;
    new_fila_final->prox_termo = NULL;
}

double pot(num)
```



```
int num;
{
    double p;
    int i;
    p = 1;
    for( i = 1; i <= num; ++i)
    {
        p = p * 2.0;
    }
    return(p);
}
```

```
peso_var(num)
double num;
{
    int counter = 0;
    while (num != 1 )
    {
        num = num / 2;
        counter = counter + 1;
    }
    return(counter);
}
```

```
double stringtodouble(s)
char s[];
{
    double val,pot;
    int i, sinal;

    for(i=0;s[i]!='\0' || s[i]=='\n' || s[i]=='\t';i++)
        /* pula espaço em branco */
        sinal=1;
    if (s[i]=='+' || s[i]=='-')/*sinal*/
        sinal=(s[i++]=='-')?1:-1;
    for (val=0;s[i]>='0' && s[i]<='9';i++)
        val=10*val+s[i]-'0';
    if(s[i]=='.')
        i++;
    for(pot=1;s[i]>='0'&& s[i]<='9';i++)
    {
        val=10*val + s[i]-'0';
        pot*=10;
    }
    return(sinal * val/pot);
}
```

```
cria_fila()
{
    aloca_mem_fila();
    cabeca = new_fila;
    rabo = new_fila;
}

cria_fila_primo()
{
    aloca_mem_primo();
    cabeca_primo = new_fila_primo;
    rabo_primo = new_fila_primo;
}

cria_fila_final()
{
    aloca_mem_fila_final();
    cabeca_final = new_fila_final;
    rabo_final = new_fila_final;
}

cria_no_raiz()
{
    int contdont;
    char linha[100], *branco;
    char car_branco[] = " \t,";
    fgets(linha, 100, le);
    branco = strtok(linha, car_branco);
    Num_var = atoi(linha) - 1;
    branco = strtok(NULL, car_branco);
    Num_dont = atoi(branco);
    aloca_mem_DDB();
    raiz = new_ddb;
    raiz->peso = pot(Num_var);
    fgets(linha, 100, le);
    aloca_mem_Dec();
    new_dec->decimal = strtodouble(linha);
    contdont = 1;

    if(Num_dont != 0)
    {
        new_dec->marca = 1;
        contdont++;
    }
    raiz->min_dec = new_dec;
    aux_dec = raiz->min_dec;
    while(fgets(linha, 100, le) != NULL)
    {
        aloca_mem_Dec();
```




```

new_dec->decimal = strtodouble(linha);
if (contdont<=Num_dont)
new_dec->marca = 1;
else
new_dec->marca = 0;
aux_dec->prox_dec = new_dec;
aux_dec = new_dec;
contdont++;
}
}

obtem_dec_irrel(no_atual_2)
struct DDB *no_atual_2;
{
struct DDB *aux_ddb, *aux_ddb_irrel;
struct Dec *dec_neg, *dec_verd, *aux_dec_irrel;

aux_ddb = no_atual_2;
dec_neg = aux_ddb->var_neg->min_dec;
while (dec_neg != NULL)
{
dec_verd = aux_ddb->var_verd->min_dec;
while (dec_verd != NULL)
{
if (dec_neg->decimal == dec_verd->decimal)
{
aloca_mem_Dec();
if (aux_ddb->var_irrel->min_dec == NULL)
{
aux_ddb->var_irrel->min_dec = new_dec;
aux_dec_irrel = new_dec;
aux_dec_irrel->decimal = dec_neg->decimal;
if ((dec_neg->marca == 1) && (dec_verd->marca == 1))
{
aux_dec_irrel->marca = 1;
}
}
else
{
aux_dec_irrel->marca = 0;
}
}
}
else
{
aux_dec_irrel->prox_dec = new_dec;
aux_dec_irrel = aux_dec_irrel->prox_dec;
aux_dec_irrel->decimal = dec_neg->decimal;
if ((dec_neg->marca == 1) && (dec_verd->marca == 1))
{
aux_dec_irrel->marca = 1;
}
}
}
}

```

```

    }
    else
    {
        aux_dec_irrel->marca = 0;
    }
}
dec_neg->marca = 1;
dec_verd->marca = 1;
break;
}
dec_verd = dec_verd->prox_dec;
}
dec_neg = dec_neg->prox_dec;
}
}

obtem_ddb_irrel(no_atual_1)
struct DDB *no_atual_1;
{
    struct DDB *aux_ddb, *aux_ddb_irrel;
    struct Dec *aux_dec, *aux_dec_neg, *aux_dec_verd, *aux_dec_irrel;

    aux_ddb = no_atual_1;
    if ((aux_ddb->var_neg != NULL) && (aux_ddb->var_verd != NULL))
    {
        aloca_mem_DDB();
        aux_ddb->var_irrel = new_ddb;
        aux_ddb_irrel = aux_ddb->var_irrel;
        aux_ddb_irrel->anterior = aux_ddb;
        aux_ddb_irrel->flag_irrel = 1;
        Num_var = peso_var(aux_ddb->peso);
        Num_var = Num_var - 1;
        aux_ddb_irrel->peso = pot(Num_var);
    }
    if (aux_ddb->var_irrel != NULL)
    {
        obtem_dec_irrel(no_atual_1);
    }
}

incluir_var_fila(aux_fila_ddb)
struct DDB *aux_fila_ddb;
{
    if (cabeca->var_expan == NULL)
    {
        cabeca->var_expan = aux_fila_ddb;
    }
    else
    {

```

```
aloca_mem_fila();
rabo->prox_var = new_fila;
rabo = new_fila;
rabo->var_expan = aux_fila_ddb;
}
}

fila_expansao(var_exp)
struct DDB *var_exp;
{
    struct DDB *aux_exp_ddb;
    struct Dec *aux_exp_dec;
    int flag_expan = 1;

    aux_exp_ddb = var_exp;
    aux_exp_dec = aux_exp_ddb->min_dec;
    while ((flag_expan == 1) && (aux_exp_dec != NULL))
    {
        if (aux_exp_dec->marca == 0)
        {
            flag_expan = 0;
        }
        aux_exp_dec = aux_exp_dec->prox_dec;
    }
    if(flag_expan == 0)
    {
        incluir_var_fila(aux_exp_ddb);
    }
}

incluir_primo_fila(aux_primo_ddb)
struct DDB *aux_primo_ddb;
{
    if (cabeca_primo->variavel == NULL)
    {
        cabeca_primo->variavel = aux_primo_ddb;
    }
    else
    {
        aloca_mem_primo();
        rabo_primo->prox_primo = new_fila_primo;
        rabo_primo = new_fila_primo;
        rabo_primo->variavel = aux_primo_ddb;
    }
}

incluir_primo_fila_final(verd,dont)
double verd;
```



```
double dont;
{
  if (cabeca_final == rabo_final)
  {
    cabeca_final->verdade = verd;
    cabeca_final->dont = dont;
    rabo_final = NULL;
  }
  else
  {
    if (rabo_final == NULL)
    {
      rabo_final = cabeca_final;
    }
    aloca_mem_fila_final();
    rabo_final->prox_termo = new_fila_final;
    rabo_final = new_fila_final;
    rabo_final->verdade = verd;
    rabo_final->dont = dont;
  }
}
```

fila_implicante(primo)

```
struct DDB *primo;
{
  struct DDB *aux_primo_ddb;
  struct Dec *aux_primo_dec;

  aux_primo_ddb = primo;
  aux_primo_dec = aux_primo_ddb->min_dec;
  if((aux_primo_dec->decimal == 0) && (aux_primo_dec->marca == 0))
  {
    incluir_primo_fila(aux_primo_ddb);
  }
}
```

expansao(no_atual)

```
struct DDB *no_atual;
{
  struct DDB *aux_ddb, *aux_ddb_neg, *aux_ddb_verd;
  struct Dec *aux_dec, *aux_dec_neg, *aux_dec_verd;

  int flag_menor, flag_maior;
  double peso_base, num_lido;

  aux_ddb = no_atual;
```



```
peso_base = aux_ddb->peso;
aux_dec = aux_ddb->min_dec;
flag_menor = 0;
flag_maior = 0;
do
{
    if ((aux_dec->decimal < peso_base) && (flag_menor == 0))
    {
        aloca_mem_DDB();
        aux_ddb->var_neg = new_ddb;
        aux_ddb_neg = aux_ddb->var_neg;
        Num_var = peso_var(aux_ddb->peso);
        Num_var = Num_var - 1;
        aux_ddb_neg->peso = pot(Num_var);
        aux_ddb_neg->anterior = aux_ddb;
        aux_ddb_neg->flag_neg = 1;
        flag_menor = 1;
    }
    if ((aux_dec->decimal >= peso_base) && (flag_maior == 0))
    {
        aloca_mem_DDB();
        aux_ddb->var_verd = new_ddb;
        aux_ddb_verd = aux_ddb->var_verd;
        Num_var = peso_var(aux_ddb->peso);
        Num_var = Num_var - 1;
        aux_ddb_verd->peso = pot(Num_var);
        aux_ddb_verd->anterior = aux_ddb;
        aux_ddb_verd->flag_verd = 1;
        flag_maior = 1;
    }
    aux_dec = aux_dec->prox_dec;
} while ((aux_dec != NULL) && ((flag_menor == 0) || (flag_maior == 0)));
aux_dec = aux_ddb->min_dec;
do
{
    num_lido = aux_dec->decimal;
    if (num_lido < aux_ddb->peso)
    {
        aloca_mem_Dec();
        if (aux_ddb_neg->min_dec == NULL)
        {
            aux_ddb_neg->min_dec = new_dec;
            aux_dec_neg = new_dec;
            aux_dec_neg->decimal = num_lido;
            aux_dec_neg->marca = aux_dec->marca;
        }
        else
        {
            aux_dec_neg->prox_dec = new_dec;
```

```

    aux_dec_neg = aux_dec_neg->prox_dec;
    aux_dec_neg->decimal = num_lido;
    aux_dec_neg->marca = aux_dec->marca;
}
}
if (num_lido >= aux_ddb->peso)
{
    aloca_mem_Dec();
    if (aux_ddb_verd->min_dec == NULL)
    {
        aux_ddb_verd->min_dec = new_dec;
        aux_dec_verd = new_dec;
        aux_dec_verd->decimal = num_lido - aux_ddb->peso;
        aux_dec_verd->marca = aux_dec->marca;
    }
    else
    {
        aux_dec_verd->prox_dec = new_dec;
        aux_dec_verd = aux_dec_verd->prox_dec;
        aux_dec_verd->decimal = num_lido - aux_ddb->peso;
        aux_dec_verd->marca = aux_dec->marca;
    }
}
aux_dec = aux_dec->prox_dec;
} while (aux_dec != NULL);
obtem_ddb_irrel(no_atual);
if (aux_ddb->var_neg != NULL)
{
    if (aux_ddb->peso >= 2)
    {
        fila_expansao(aux_ddb->var_neg);
    }
    else
    {
        fila_implicante(aux_ddb->var_neg);
    }
}
if (aux_ddb->var_verd != NULL)
{
    if (aux_ddb->peso >= 2)
    {
        fila_expansao(aux_ddb->var_verd);
    }
    else
    {
        fila_implicante(aux_ddb->var_verd);
    }
}
if (aux_ddb->var_irrel != NULL)

```




```
{
  if (aux_ddb->peso >= 2)
  {
    fila_expansao(aux_ddb->var_irrel);
  }
  else
  {
    fila_implicante(aux_ddb->var_irrel);
  }
}
}
```



```
cria_arq_resul_1()
{
  struct DDB *aux_ddb;
  struct Dec *aux_dec;
  struct Fila_final *implicante;

  fprintf(escreve, "\nPROGRAMA EXPANDER \n\n");
  fprintf(escreve, "Objetivo: Obter implicantes primos de funções booleanas\n");
  fprintf(escreve, "      utilizando-se do Teorema de Expansão de Shannon.\n\n");
  fprintf(escreve, "      De posse dos implicantes primos o problema de \n");
  fprintf(escreve, "      minimização de funções booleanas e formulado como \n");
  fprintf(escreve, "      um problema de programação linear inteira 0 e 1.\n\n");
  fprintf(escreve, "A função , composta pelos seguintes Mintermos e Don't Care\n\n");
  aux_ddb = raiz;
  aux_dec = aux_ddb->min_dec;
  while(aux_dec != NULL)
  {
    fprintf(escreve, "%.0f ", aux_dec->decimal);
    aux_dec = aux_dec->prox_dec;
  }
}
```



```
double cria_arq_resul_2()
{
  struct DDB *aux_ddb;
  struct Dec *aux_dec;
  struct Fila_final *implicante;
  implicante = cabeca_final;
  fprintf(escreve, "\n\nImplicantes primos obtidos :\n\n");
  while(implicante != NULL)
  {
    fprintf(escreve, "(%.0f, %.0f)\n", implicante->verdade, implicante->dont);
    implicante = implicante->prox_termo;
  }
  end=clock()/CLK_TCK;
  fprintf(escreve, "\ntempo decorrido: %.9f seconds\n", (end-start));
}
```



```
fprintf(escreve, "\nmemória total utilizada: %.0f bytes", ac_mem);  
}
```

```
elimina_var()  
{  
    struct Fila_var *delete;  
    delete = cabeca;  
    cabeca = cabeca->prox_var;  
    ac_mem-=sizeof(*delete);  
    free(delete);  
}
```

```
elimina_fila_ini()  
{  
    struct Dec *delini;  
  
    delini=raiz->min_dec;  
    while(delini->prox_dec!=NULL)  
    {  
        free(delini);  
        delini = delini->prox_dec;  
    }  
}
```

```
obter_primos()  
{  
    struct Fila_primo *primo;  
    struct DDB *var_diagrama;  
  
    double v_verd;  
    double v_dont;  
  
    primo = cabeca_primo;  
    while (primo != NULL)  
    {  
        v_verd = 0;  
        v_dont = 0;  
        var_diagrama = primo->variavel;  
        while ( var_diagrama != raiz )  
        {  
            if (var_diagrama->flag_verd == 1)  
            {  
                v_verd = v_verd + var_diagrama->anterior->peso;  
            }  
            if (var_diagrama->flag_irrel == 1)  
            {  
                v_dont = v_dont + var_diagrama->anterior->peso;  
            }  
            var_diagrama = var_diagrama->prox;  
        }  
        primo = primo->prox;  
    }  
}
```

```
    v_dont = v_dont + var_diagrama->anterior->peso;
}
var_diagrama = var_diagrama->anterior;
}
incluir_primo_fila_final(v_verd,v_dont);
primo = primo->prox_primo;
}
}
```

```
inicializa_filas()
{
    cria_fila();
    cria_fila_primo();
    cria_fila_final();
    /*cria_no_raiz();*/
}
```

```
main()
{
    ac_mem=0;
    /*clrscr()*/;
    abre_arq();
    inicializa_filas();
    cria_no_raiz();
    cria_arq_resul_1();
    expansao(raiz);
    max_mem=ac_mem;
    while (cabeca != NULL)
    {
        expansao(cabeca->var_expan);
        aux_max_mem=ac_mem;
        if (max_mem<aux_max_mem)
            max_mem=aux_max_mem;
        elimina_var();
    }
    obter_primos();
    cria_arq_resul_2();
    fecha_arq();
    /*clrscr()*/;
    return 0;
}
```


Tabela com os resultados gerados pelos dois programas para os 105 casos estudados. Onde implicante primo, programa Expander e programa Pigene foram abreviados para ip, exp e pig, respectivamente.

caso	variáveis	dont care	mintermos	nº elementos	nº ip exp	nº ip pig	memória exp	memória pig	tempo exp	tempo pig
1	3	1	1	2	1	1	416	78	0,01	0,06
2	3	1	2	3	2	2	672	136	0,03	0,06
3	3	0	3	3	2	2	688	116	0,03	0,00
4	3	1	2	3	2	2	688	116	0,02	0,00
5	3	2	1	3	1	3	400	134	0,03	0,06
6	3	0	3	3	2	2	656	116	0,02	0,00
7	3	1	2	3	2	2	656	116	0,01	0,00
8	3	2	1	3	1	2	448	116	0,01	0,00
9	3	1	2	3	2	2	672	116	0,01	0,00
10	3	1	3	4	2	3	832	174	0,01	0,06
11	3	0	4	4	1	1	592	138	0,01	0,00
12	3	2	2	4	2	3	832	154	0,01	0,00
13	3	2	2	4	2	3	776	154	0,01	0,00
14	3	1	4	5	2	2	896	176	0,02	0,00
15	3	2	2	4	3	4	952	232	0,01	0,00
16	3	4	3	7	2	6	896	268	0,02	0,00
17	3	4	3	7	2	3	1120	274	0,02	0,00
18	4	1	2	3	2	2	776	116	0,02	0,00
19	4	2	2	4	1	1	632	13	0,01	0,00
20	4	0	4	4	4	4	1168	172	0,03	0,00
21	4	2	2	4	2	3	824	154	0,02	0,06
22	4	1	3	4	1	1	680	138	0,01	0,00
23	4	2	2	4	2	3	832	174	0,01	0,00
24	4	1	3	4	3	3	1024	154	0,01	0,00
25	4	2	2	4	2	3	936	154	0,01	0,06
26	4	2	3	5	2	4	848	192	0,03	0,00
27	4	2	3	5	2	2	1024	176	0,01	0,00
28	4	3	2	5	2	1	664	176	0	0,00
29	4	3	2	5	2	4	1008	212	0,01	0,00
30	4	3	2	5	2	4	896	192	0,03	0,00
31	4	1	4	5	1	2	680	176	0,03	0,06
32	4	1	4	5	3	4	1152	192	0,04	0,00
33	4	1	4	5	3	3	1232	214	0	0,00
34	4	2	4	6	3	5	1200	290	0,01	0,00
35	4	2	4	6	2	4	1168	272	0,02	0,00
36	4	1	5	6	4	4	1672	272	0,03	0,00
37	4	2	4	6	2	3	1488	254	0,01	0,00
38	4	3	3	6	2	4	1200	272	0,01	0,00

Tabela anexo C.1 - Casos 1 a 38

caso	variáveis	dont care	mintermos	nº elementos	nº ip exp	nº ip pig	memória pig	memória pig	tempo exp	tempo pig
39	4	3	3	6	2	2	1232	256	0,01	0,00
40	4	2	4	6	4	5	1440	290	0,01	0,06
41	4	2	4	6	2	3	1184	274	0,02	0,00
42	4	3	4	7	3	4	1472	292	0,06	0,00
43	4	2	5	7	3	4	1432	312	0,01	0,00
44	4	3	4	7	3	4	1320	292	0,01	0,06
45	4	3	4	7	2	3	1248	274	0,01	0,06
46	4	2	5	7	3	3	1440	314	0,03	0,06
47	4	2	5	7	3	3	1440	254	0,03	0,00
48	4	3	4	7	3	3	1440	274	0,03	0,00
49	4	4	3	7	2	4	1120	252	0,03	0,00
50	5	3	8	11	5	8	2944	624	0,03	0,00
51	5	5	7	12	6	10	2888	542	0,01	0,06
52	5	3	9	12	3	4	2400	534	0,01	0,06
53	5	3	10	13	7	8	3432	652	0,01	0,00
54	5	5	9	14	7	8	3664	786	0,01	0,06
55	5	3	11	14	6	9	2200	744	0,01	0,00
56	5	4	10	14	6	6	2488	688	0,01	0,00
57	5	5	10	15	8	10	4152	960	0,04	0,00
58	5	6	10	16	6	8	3768	812	0,05	0,00
59	5	7	10	17	5	10	3440	1160	0,01	0,00
60	5	8	9	17	4	4	3168	1252	0,01	0,00
61	5	5	12	17	5	6	3744	934	0,01	0,00
62	5	5	13	18	6	9	4104	1170	0,01	0,00
63	5	5	13	18	4	5	3472	1210	0,01	0,00
64	5	5	14	19	7	11	4312	1384	0,01	0,00
65	6	7	19	26	4	4	4332	2016	0,02	0,00
66	6	11	27	38	16	27	10128	5112	0,02	0,00
67	6	7	32	39	12	20	9624	4024	0,05	0,06
68	6	9	31	40	17	35	10576	5034	0,02	0,00
69	6	8	32	40	13	26	9736	5568	0,01	0,06
70	6	10	31	41	15	20	9992	4946	0,01	0,00
71	6	10	31	41	15	24	9272	4288	0,01	0,00
72	6	8	35	43	18	30	11304	4720	0,01	0,00
73	6	8	35	43	14	25	8680	5436	0,02	0,00
74	6	11	33	44	12	20	12136	5772	0,03	0,00
75	6	13	32	45	12	18	9224	5906	0,02	0,06
76	6	11	34	45	9	14	8816	6440	0,01	0,06
77	6	10	35	45	13	26	9320	5840	0,01	0,06
78	6	6	40	46	23	30	14000	5966	0,03	0,06
79	6	9	38	47	14	22	10600	5680	0,02	0,06
80	6	13	34	47	17	23	12656	6432	0,02	0,06
81	6	8	42	50	15	26	11464	7060	0,01	0,00
82	6	13	38	51	15	30	11144	7824	0,02	0,06

Tabela anexo C.2 - Casos 39 a 82



caso	variáveis	dont care	mintermos	n° elementos	n° ip exp	n° ip pig	memória pig	memória pig	tempo exp	tempo pig
83	6	8	43	51	10	11	9488	7330	0,01	0,06
84	6	14	40	54	14	22	10664	7784	0,01	0,00
85	6	18	45	63	1	1	4840	13058	0,02	0,06
86	7	8	55	63	38	64	20608	11204	0,01	0,06
87	7	9	69	78	33	47	24024	13968	0,01	0,00
88	7	20	59	79	33	55	23248	16126	0,03	0,00
89	7	22	59	81	31	53	23944	16276	0,01	0,06
90	7	28	59	87	26	44	21376	19342	0,01	0,06
91	7	19	68	87	24	62	23728	19080	0,01	0,06
92	7	26	65	91	27	43	24816	20388	0,05	0,06
93	7	27	65	92	36	97	27040	21732	0,02	0,00
94	7	26	66	92	29	57	23968	22352	0,02	0,06
95	7	12	81	93	34	67	28488	22716	0,02	0,06
96	7	21	74	95	25	51	26624	25054	0,03	0,06
97	7	26	71	97	30	73	26088	26486	0,01	0,11
98	7	22	79	101	31	66	27720	26284	0,01	0,06
99	7	13	90	103	38	76	31264	27468	0,05	0,06
100	7	16	87	103	28	42	26456	27308	0,04	0,06
101	7	25	79	104	26	58	24920	28492	0,01	0,06
102	7	19	87	106	31	69	25392	28416	0,01	0,06
103	7	23	86	109	18	33	19048	32818	0,01	0,16
104	7	29	81	110	19	26	20336	31836	0,07	0,11
105	7	16	99	115	17	33	18312	35366	0,01	0,16

Tabela anexo C.3 - Casos 83 a 105



UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Câmpus de Ilha Solteira
Programa de Pós-Graduação em Engenharia Elétrica
Av. Brasil Centro, 56
15385-000 Ilha Solteira - SP
www.dee.feis.unesp.br

