UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”

FACULDADE DE ENGENHARIA

CAMPUS DE ILHA SOLTEIRA

JULIANA DE FÁTIMA FRANCISCANI

CONSENSO ITERATIVO: GERAÇÃO DE IMPLICANTES PRIMOS PARA MINIMIZAÇÃO DE FUNÇÕES BOOLEANAS COM MÚLTIPLAS SAÍDAS

Ilha Solteira

2016

JULIANA DE FÁTIMA FRANCISCANI

CONSENSO ITERATIVO: GERAÇÃO DE IMPLICANTES PRIMOS PARA MINIMIZAÇÃO DE FUNÇÕES BOOLEANAS COM MÚLTIPLAS SAÍDAS

Dissertação apresentada à Faculdade de Engenharia – UNESP – Campus de Ilha Solteira como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica. Área de Conhecimento: Automação.

Prof. Dr. Alexandre César Rodrigues da Silva
Orientador

Ilha Solteira
2016

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

F819C

Franciscani, Juliana de Fátima.
Consenso iterativo: geração de implicantes primos para minimização de funções booleanas com múltiplas saídas / Juliana de Fátima Franciscani. -- Ilha Solteira: [s.n.], 2016
118 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Área de conhecimento: Automação, 2016

Orientador: Alexandre Cesar Rodrigues da Silva
Inclui bibliografia

1. Geração de implicantes primos. 2. Minimização de funções booleanas com múltiplas saídas. 3. Método de minimização de funções booleanas com múltiplas saídas.

unesp

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Ilha Solteira

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Consenso Iterativo: Geração de Implicantes Primos para Minimização de Funções Booleanas com Múltiplas Saídas

AUTORA: JULIANA DE FÁTIMA FRANCISCANI
ORIENTADOR: ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovada como parte das exigências para obtenção do Título de Mestra em ENGENHARIA ELÉTRICA, área: AUTOMAÇÃO pela Comissão Examinadora:

Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Profa. Dra. BERENICE CAMARGO DAMASCENO
Departamento de Matemática / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. MARCOS AMORIELLE FURINI
Coordenação da Área de Indústria / Instituto Federal de São Paulo - Câmpus de Votuporanga

Ilha Solteira, 31 de agosto de 2016

AGRADECIMENTOS

Primeiramente agradeço a Deus pela vida.

A meus familiares e pessoas queridas que sempre me apoiaram e incentivaram, meu muito obrigada.

A meu querido esposo, Ivan Oliveira Lopes, pelo companheirismo, dedicação e amor. Saiba que sem você não teria conseguido. Obrigada pelo seu apoio incondicional ao longo deste processo e de tantos outros! Obrigada por acreditar em mim, mesmo quando eu não acreditava mais.

Ao pequeno Arthur, filho amado, que me fortalece e faz acreditar que posso ser uma pessoa melhor a cada dia.

Agradeço ao meu orientador Prof. Dr. Alexandre César Rodrigues da Silva, pela confiança, além da indiscutível compreensão em momentos difíceis. O senhor é um profissional incrível. Muito obrigada por todo comprometimento, dedicação e paciência.

Ao Instituto Federal de São Paulo, por propiciar nos últimos meses a possibilidade de dedicação aos estudos em tempo integral.

RESUMO

Com a evolução e difusão do desenvolvimento de equipamentos utilizando microtecnologia e nanotecnologia, circuitos cada vez menores, mais eficientes e que consomem menos energia, são necessários. Os métodos de minimização de funções booleanas tornam-se relevantes por possibilitarem a otimização de circuitos lógicos, através da geração de circuitos que possuam a mesma funcionalidade, porém, minimizados. Estudos na área de minimização de funções booleanas são realizados há muito tempo, e estão sendo adaptados às novas tecnologias. A geração de implicantes primos de uma função booleana é um dos passos para a cobertura dos mintermos da função e, consequentemente, para a obtenção da função de custo mínimo. Neste trabalho, a Primeira Fase do Método de Quine-McCluskey para Funções Booleanas com Múltiplas Saídas (QMM) foi implementada para posterior comparação com os Métodos Propostos GPMultiplo e MultiGeraPlex (baseados na filosofia do algoritmo GeraPlex). Os métodos propostos geram os implicantes primos de uma função booleana com múltiplas saídas e utilizam a operação de consenso iterativo para comparar dois termos. Os resultados obtidos, através da comparação do GPMultiplo, MultiGeraPlex e da Primeira Fase do Método de QMM, puderam comprovar que a aplicação dos métodos propostos torna-se mais viável e vantajosa por permitir menor tempo de execução e uso de memória, menor quantidade de implicantes gerados e de comparações entre os termos.

Palavras-chave: Consenso iterativo. Funções booleanas. Implicantes primos. Minimização. Múltiplas saídas.

ABSTRACT

With the evolution and spread of the development of equipment using microtechnology and nanotechnology, circuits in need are smaller, more efficient and consume less power. Methods of Minimizing Boolean Functions become important as they allow optimization of logic circuits by generating circuits having the same functionality, but minimized. Studies in Minimizing Boolean Functions area are carried out long ago, and are being adapted to new technologies. The generation of prime implicants of a Boolean function is one of the steps for covering the function of the minterms, and consequently to obtain the minimum cost function. In this work, the first phase of the Quine-McCluskey Method for Booleans Functions with Multiple Output (QMM) was implemented for comparison with Proposed Methods GPMultiplo and MultiGeraPlex (based on the philosophy of GeraPlex algorithm). The proposed methods generates the prime implicants of a Boolean Function with Multiple Output and using the iterative consensus operation to compare two terms. The results obtained by comparing the GPMultiplo, MultiGeraPlex and the first phase of the QMM Method, were able to prove that the application of the proposed methods becomes more feasible and advantageous, by allowing smaller execution time, number of implicants and number of comparisons.

Keywords: Consensus iterative. Booleanas functions. Primes implicants. Minimization. Multiple outputs.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o aumento da produção dos dispositivos eletrônicos digitais e sua ampla difusão, bem com a tendência de equipamentos cada vez menores e com alto desempenho, o estudo de técnicas que possibilitem a redução de circuitos lógicos é muito importante. Métodos de minimização de funções booleanas se tornam relevantes por possibilitarem a otimização dos circuitos lógicos através da geração de circuitos minimizados.

Minimização de função booleana possibilita redução do número de literais[1] e de portas lógicas na construção de circuito elétrico, tendo como resultado menor dissipação de calor, menor área ocupada e maior velocidade de execução, proporcionando melhor desempenho.

A geração dos implicantes[2] de uma função booleana é uma das etapas do processo de minimização, no qual a função mínima obtida deve ser composta pelo conjunto de implicantes primos[3] que possua o menor custo. Ou seja, o circuito lógico minimizado deverá possuir menor número de entradas nas portas lógicas (AND e OR).

Eficiência na geração de implicantes primos é um fator importante na fase de cobertura dos mintermos[4] em métodos de minimização de funções booleanas. Muitos algoritmos com essa finalidade têm sido propostos na literatura, utilizando métodos exatos ou não.

Como o problema de minimização tem seu custo computacional elevado com o aumento de variáveis de entrada, o estudo relacionado às estruturas capazes de otimizar e minimizar o custo computacional dos métodos é de suma importância. Pela possibilidade de redução do circuito lógico, a análise e a otimização no processo de geração dos implicantes primos é um ponto crucial para melhor desempenho do processo.

Seja a minimização realizada para funções com múltiplas saídas ou saídas simples,

[1] Literal pode ser considerado como uma variável complementada ou não.

[2] Implicante é um termo produto, em que todas as variáveis, pertencentes a função, aparecem uma única vez, complementada ou não (MENDELSON, 1970).

[3] Quando não há possibilidades de reduzir um implicante, este é dito como implicante primo (MENDELSON, 1970).

[4] Um produto (ou termo produto) que contenha as n variáveis de uma função como fatores, sendo essas complementadas ou não, é definido como mintermo (MENDELSON, 1970).

para circuitos reversíveis[5] ou não, o tema é de grande relevância. Isso porque possibilita a redução no número de portas lógicas na composição do circuito lógico e, consequentemente, menor dissipação de energia, independente do método utilizado.

Um dos objetivos do trabalho é o estudo e a análise de alguns métodos de otimização de funções booleanas. Assim sendo, desenvolveu-se dois métodos de geração de implicantes primos para funções booleanas com múltiplas saídas, baseado na proposta de redução de custo de circuitos lógicos. Para atingir o objetivo, inicialmente, foi implementada uma adaptação do algoritmo GeraPlex, desenvolvido por Silva (SILVA, 1993), que gera todos os implicantes primos através da operação de consenso entre dois termos para uma única função. Os métodos desenvolvidos foram denominados GPMultiplo e MultiGeraPlex, algoritmos esses utilizados para geração de implicantes primos de funções booleanas com múltiplas saídas, e que utilizam como estrutura de dados a árvore ternária.

Os algoritmos propostos geram os implicantes primos, de posse dos implicantes primos, o problema de cobertura dos mintermos é formulado como um problema de programação linear inteira 0 e 1. Os algoritmos são detalhados no Capítulo 4, seções 4.2 e 4.3.

Para possibilitar a comparação dos métodos, implementou-se a primeira fase do método de Quine-McCluskey adaptado para Múltiplas Saídas, descrita no Capítulo 3, seção 3.3.

O trabalho está estruturado em 6 capítulos. Neste, apresenta-se uma breve introdução ao problema de Minimização de Funções Booleanas, e sua contextualização, bem como objetivos, importância e metodologia utilizada para o desenvolvimento do mesmo. No Capítulo 2, são apresentados os fundamentos básicos relacionados à Álgebra Booleana, assim como alguns conceitos importantes para melhor compreensão do trabalho.

No Capítulo 3, conceitos e características de funções booleanas com múltiplas saídas e, o método de Quine-McCluskey para múltiplas saídas foram descritos. Os algoritmos propostos, GPMultiplo e MultiGeraPlex, para geração de implicantes primos das funções booleanas com múltiplas saídas são descritos e exemplificados no Capítulo 4. Alguns dos resultados gerados com a execução da Primeira Fase do Método de Quine-McCluskey para Funções com Múltiplas Saídas, do GPMultiplo e do MultiGeraPlex são apresentados no Capítulo 5. Por fim, no Capítulo 6, são expostas as considerações finais.

Na próxima seção, alguns trabalhos relacionados são apresentados, esses tratam de

[5]Um circuito é dito como reversível, se para cada valor de saída y da função, produz o valor de x de modo que $f(x) = y$. Por definição deve-se ter para cada entrada, uma saída correspondente (função bijetora).

assuntos como consenso iterativo, estrutura de armazenamento em árvore e minimização de funções booleanas com múltiplas saídas. Esses temas foram de suma importância para o desenvolvimento da pesquisa e dos algoritmos propostos.

A proposta do trabalho se difere dos métodos que são apresentados principalmente na representação da função booleana. As propostas utilizam árvore ternária para o armazenamento dos termos da função. O consenso é aplicado a cada nível da árvore (do *bit* menos significativo para o mais significativo), não havendo necessidade de rotação dos nós.

Outro ponto relevante do trabalho é a forma da aplicação do consenso entre os caminhos da árvore (até o nó folha), o que pode resultar em redução do número de termos (quando é aplicada a Fusão entre os caminhos), pode manter o mesmo número de termos (quando consegue eliminar apenas um dos caminhos de origem) ou aumentar, quando não se elimina nenhum dos termos, e há a geração de um novo caminho na árvore (novo nó folha é gerado).

1.1 TRABALHOS RELACIONADOS

Estudos na área de minimização de funções booleanas são realizados há muito tempo; em meados da década de 50 até final da década de 60, foram desenvolvidas várias pesquisas na área. Alguns anos depois, com a queda no valor dos componentes do circuito, o interesse e as pesquisas na área não evoluíram muito. Tais estudos voltaram a ter importância na década de 80, com a utilização de Matriz Lógica Programável (*Program Logic Array*, *PLA*), empregada na implementação de funções booleanas, na qual cada termo produto da função era implementado como uma linha na matriz.

Por possibilitar a implementação de funções com múltiplas saídas, o *PLA* impulsionou o desenvolvimento de métodos para minimização de várias funções como sendo uma única, com múltiplas saídas. Nestes casos, os implicantes primos compartilhados são gerados uma única vez, reduzindo o número de portas lógicas pertencentes ao circuito lógico e, consequentemente, o custo da função resultante.

Dezenas de métodos para a minimização de funções booleanas já foram desenvolvidos e aprimorados. A maioria deles parte do princípio básico dos métodos clássicos, como o Mapa de Karnaugh (KARNAUGH, 1953) e o método de Quine-McCluskey (MCCLUSKEY, 1986), que é um método exato.

O Mapa de Karnaugh (KARNAUGH, 1953), método mais clássico para minimização

de funções booleanas, utiliza-se da ideia de tabelas. Parte do princípio no qual os mintermos ou maxtermos[6] podem ser representados em células de uma tabela. Neste método, não há necessidade do emprego de expressões algébricas e é um método que depende da habilidade do usuário.

No início da década de 50, Quine (QUINE, 1952) propôs um método para encontrar formas normais simples disjuntivas de uma função verdade. Em outro trabalho, apresentou a utilização da operação de consenso entre dois termos adjacentes para gerar todos os implicantes primos de uma função. O procedimento não considera estados irrelevantes (*don't care states*) e era utilizado para funções simples (QUINE, 1955).

Nelson (NELSON, 1955a, 1955b), desenvolveu um método algébrico, baseado na ideia proposta por Quine (QUINE, 1952), para encontrar todos os implicantes primos de uma função verdade. Diferente de Quine, o método de Nelson pode ser aplicado para funções na forma conjuntiva e disjuntiva. Outra diferença apresentada por Nelson, é que o método não realiza a combinação entre todos os termos adjacentes.

Em 1956, também baseado no processo de minimização proposto por Quine (QUINE, 1955), McCluskey (MCCLUSKEY, 1956) apresentou um método de geração de implicantes primos e de cobertura dos mintermos para obtenção da função mínima. O autor apresentou as etapas para a geração da tabela de cobertura para obtenção dos implicantes primos e formas alternativas para implementar o procedimento. Os estados irrelevantes de uma função são considerados no método.

Em 1961, McCluskey desenvolveu um método para minimizar funções booleanas com múltiplas saídas. Descreveu a geração da soma mínima por funções que possuem um grande número de estados irrelevantes, e explicou como é formada a tabela de cobertura para encontrar o função mínima para funções com múltiplas saídas (MCCLUSKEY, 1961),.

Mott (MOTT, 1960) descreveu uma nova forma algébrica que obtém implicantes primos de forma mais simples. Utiliza uma extensão da técnica utilizada por Quine (QUINE, 1955), em que o consenso iterativo é aplicado entre os termos. Três fases compõem a simplificação. Na primeira, compara-se os termos aplicando o consenso entre eles. Na segunda fase, cada termo gerado é definido por seu par, e uma lista de implicações redundantes é obtida. Para a terceira fase, os termos da função são reescritos utilizando a representação obtida pela lista de implicações e uma composição das variáveis é realizada

[6]Uma soma (ou termo soma) que contenha as n variáveis de uma função como fatores, sendo essas complementadas ou não, é definida como maxtermo(MENDELSON, 1970).

a partir da utilização da lei distributiva. Nessa fase ocorre a eliminação de redundâncias.

O método de Quine-McCluskey (MCCLUSKEY, 1986), busca a minimização para geração dos implicantes primos de uma função booleana. Método tabular e iterativo clássico, muito utilizado na literatura como base ou comparativo a outros procedimentos. É composto por duas fases, sendo a primeira a geração de implicantes primos e, a segunda, a cobertura dos mintermos da função. O método não utiliza nenhuma heurística, e como fundamento principal, utiliza o teorema $AB + \overline{A}B = B$ repetidamente, entre os termos da função. Os termos da função são classificados em caixas, de acordo com a quantidade de dígitos $1's$ que possui e inseridos no grupo inicial. O comparativo é realizado entre todos os termos pertencentes a uma caixa, com todos os termos pertencentes a próxima caixa. Os implicantes gerados são armazenados em um novo grupo e todo o processo é repetido até que não seja gerado mais implicantes, ou que apenas em um novo grupo contenha apenas uma caixa. Após a geração dos implicantes primos, a fase de cobertura dos mintermos é realizada, e a função mínima obtida.

Tison, em 1967, apresentou a extensão da operação de consenso aplicada a um maior número de termos, e não somente a dois, para encontrar os implicantes primos e a soma mínima de uma função booleana. Em seu algoritmo, funções de saída simples e saída múltiplas podem ser minimizadas, com a presença ou não de *don't care states* (TISON, 1967).

Das e Choudhury (DAS; CHOUDHURY, 1965), desenvolveram um método tabular para geração mais eficiente de todos os implicantes primos de uma função booleana a partir da expressão em maxtermos, representada por números decimais. Se baseou na abordagem algébrica proposta por Nelson (NELSON, 1955a), na qual utiliza-se a ideia de Expansão sucessiva para geração de todos os implicantes primos de uma função booleana.

Em 1970, um novo algoritmo para geração de implicantes primos foi proposto (SLAGLE; CHANG; LEE, 1970). Segundo os autores, é um algoritmo eficiente, e difere de métodos apresentados em (QUINE, 1955; MCCLUSKEY, 1961; TISON, 1967), por não gerar o mesmo implicante primo mais de uma vez, não necessitar de grande capacidade de memória para a execução e não requerer grande número de operações básicas. Uma lista de prioridade, baseada na frequência das variáveis que compõe a função é utilizada para que seja selecionada a variável a ser expandida. O processo de Expansão da função em torno dessas variáveis é realizado para gerar os implicantes. Em alguns casos implicantes não primos podem ser gerados.

Das (DAS, 1971) comenta a respeito do algoritmo proposto em (SLAGLE; CHANG;

LEE, 1970). Salienta que a ideia básica para o desenvolvimento do algoritmo não é nova, e que os conceitos já haviam sido apresentados em outros artigos como em (NELSON, 1955a; MCCLUSKEY, 1961; SCHEINMAN, 1962; TISON, 1967; DAS; CHOUDHURY, 1965).

Das e Khabra (DAS; KHABRA, 1972) desenvolveram um algoritmo que utiliza uma técnica tabular e iterativa para a geração dos implicantes primos, que são representados pelas colunas de uma tabela. De acordo com os autores, uma função booleana, pode ser representada através dos mintermos ou maxtermos. O método gera os implicantes primos através da Expansão em torno de um literal.

Ainda no mesmo segmento de geração de implicantes primos através da aplicação da operação de consenso, Hulme e Worrell (HULME; WORRELL, 1975) apresentaram um algoritmo semelhante ao de Nelson (NELSON, 1955a). Difere deste pois, além de apresentar as operações de complemento e Expansão, inclui a operação de fatoração para a geração dos implicantes. Esta operação é aplicada antes da realização de cada complemento.

Kuo (KUO; CHOU, 1986) propôs um novo conceito para a operação de consenso, chamado consenso assimétrico (Acons). No trabalho, apresenta um algoritmo, que de acordo com o autor, é rápido, e que pode gerar todos os implicantes primos essenciais[7] sem gerar uma cobertura dos mintermos pertencentes à função booleana. Utiliza-se para a detecção dos implicantes primos essenciais um algoritmo de verificação de tautologia. Os implicantes primos essenciais são obtidos a partir da primeira geração de um conjunto de implicantes primos, que são denominados primos parcialmente essenciais. Um teste é realizado com o intuito de verificar se o primo parcialmente essencial é na verdade um implicante primo essencial.

Um algoritmo para geração de implicantes primos de uma função booleana similar ao Método de Tison (TISON, 1967) é proposto por Kean (KEAN; TSIKNIS, 1990). O algoritmo apresenta duas diferenças: a primeira é que o consenso é aplicado em relação ao conjunto de literais que ocorrem na entrada; e a segunda é que não será aplicado o consenso entre os termos de um mesmo conjunto.

Silva (SILVA, 1993) desenvolveu uma técnica que utiliza a operação de consenso iterativo entre dois termos de uma função booleana. Os termos são representados em uma estrutura de árvore, na qual cada caminho da árvore está relacionado a um termo da função. Denominada GeraPlex, a técnica gera todos os implicantes primos de uma

[7]Se um mintermo é coberto por um único implicante primo, este é dito como implicante primo essencial.

função, porém diferente do método de Quine-McCluskey, não gera todos os implicantes. Três situações podem ocorrer ao aplicar o consenso e gerar um novo termo: Fusão (quando o termo gerado cobre os dois de origem e esses são eliminados da árvore), Deslocamento (quando o termo gerado cobre apenas um dos termos de origem e este é eliminado da árvore) e a Expansão (quando o termo gerado não cobre os termos de origem).

Em dois artigos, (FISER; RUCKY; VANOVA, 2008; FISER; TOMAN, 2009), os autores utilizam o consenso entre dois termos para geração dos implicantes primos de uma função booleana. A representação dos termos é em estrutura de árvore ternária. A proposta é muito similar a de Silva (SILVA, 1993) na representação dos termos como caminho em uma árvore, porém, utiliza a operação de consenso apenas nos nós folhas[8] da árvore. Para que todas as variáveis sejam analisadas, n (número de variáveis) rotações na árvore são realizadas para se obter o resultado final.

Outros trabalhos relacionados à minimização de funções booleanas por meio da geração de implicantes primos através da aplicação da operação de consenso podem ser encontrados em (RHYNE et al., 1977; OGUNBIYI; HENLEY, 1981; BENEDIKTSSON; SWAIN, 1992; STRZEMECKI, 1992; MARQUIS, 1995; HELAL; BHARGAVA, 1995; ALEXE et al., 2004; CHEN; HUO; CHEN, 2009).

Lee (LEE, 1959) introduz uma representação alternativa para circuitos denominada Programa de Decisão Binária. No artigo o autor descreve através de exemplos como seria a formulação do circuito pelo programa de decisão binária. A ideia é baseada na seguinte instrução, $Tx; A, B$. Toda instrução do programa de decisão binária segue o mesmo princípio, em que, se a variável x for 0, a próxima instrução do programa será para o endereço A e, se x for 1, leva a próxima instrução do programa para o endereço B. Uma tabela de instrução é montada e assim a função pode ser minimizada.

A retomada da utilização do programa de decisão binária na área de minimização, aconteceu alguns anos depois da proposta de Lee (LEE, 1959), após mudanças relacionadas à tecnologia e ao custo de memórias. A partir de então, vários trabalhos relacionados a decisão binária foram desenvolvidos e aprimorados, como em (BOUTE, 1976; AKERS, 1978; THAYSE; DAVIO; DESCHAMPS, 1978; CERNY; MANGE; SANCHEZ, 1979; MINATO; ISHIURA; YAJIMA, 1990; TODA, 2016).

Akers (AKERS, 1978) propôs uma representação gráfica de funções booleanas através de uma estrutura de dados denominada Diagrama de Decisão Binária (DDB). No artigo

[8]Nós folhas de uma árvore, são os nós da extremidade e que não possuem nós filhos (CORMEN et al., 2009).

o autor descreve e analisa a estrutura de dados proposta e a implementa.

Em 1979, o conceito de uma árvore de decisão binária foi estendido, para que houvesse a representação de várias funções de saída. Os autores, em seu artigo (CERNY; MANGE; SANCHEZ, 1979), propõem uma técnica de minimização para funções completamente especificadas e com múltiplas saídas, representada através de árvore de decisão binária. Dois procedimentos para a síntese de memória mínima pela árvore de decisão são apresentados: um teórico e determinístico e a outro mais eficiente de natureza heurística.

Embora a ideia de representar funções booleanas como DDB tenha sido proposta em 1978, a aplicação ganhou notoriedade em 1986, com Bryant (BRYANT, 1986). O autor formulou um conjunto de algoritmos para a construção e operação nestas estruturas. Bryant propôs também uma maior restrição na ordenação de decisão nos vértices da árvore.

Em 1987, Friedman (FRIEDMAN; SUPOWIT, 1987) propôs um algoritmo para encontrar uma variável de ordenação ótima para a inclusão dos termos no DDB. Porém, a manipulação das funções e os resultados intermediários são realizados através de tabelas verdades extendidas.

Em 1991, baseado no algoritmo de Friedman (FRIEDMAN; SUPOWIT, 1987), os autores Ishiura, Sawada e Yajima (ISHIURA; SAWADA; YAJIMA, 1991), implementaram um novo algoritmo exato e propuseram uma melhoria gradual no método de minimização de DDB. No método proposto a função intermediária é representada também através de DDB, além de inserir podas no diagrama o que reduz o custo computacional da proposta.

Fiser e Hlavicka (FISER; HLAVICKA, 2001), propuseram um método de minimização de funções booleanas, denominado BOOM. O método representa os implicantes primos da função em árvore ternária. Toda a manipulação e o armazenamento dos termos da função também são realizados em árvores ternárias. Segundo os autores, o método utiliza uma abordagem *top-down* na inclusão dos literais para a geração dos implicantes. BOOM usa o conjunto de entrada como referência para determinar se a tentativa de solução está correta ou não.

O método BOOM foi detalhado em 2003, os autores apresentaram a estrutura, a heurística, o problema de cobertura, a minimização iterativa e a Expansão de implicantes. Uma abordagem para aplicação em funções com múltiplas saídas também é apresentada no artigo (HLAVICKA; FISER, 2001).

Em 2008, Fiser, Rucky e Vanova (FISER; RUCKY; VANOVA, 2008) descreveram

a utilização e manipulação de árvore ternária para a minimização de funções booleanas completamente especificadas. Apresentaram o método *Pupik*, baseado na redução da árvore ternária. Os termos produtos são inseridos na árvore, e cada caminho do nó raiz até a folha representa um mintermo da função. As manipulações são realizadas apenas nos nós folhas e são realizadas tantas rotações quanto forem o número de variáveis da função. Utilizam a Fusão dos termos vizinhos para a geração dos implicantes primos. Uma atualização em 2009 foi proposta pelos autores, incluindo a possibilidade de minimização de funções incompletamente especificadas (FISER; TOMAN, 2009).

Funções booleanas com múltiplas saídas são estudadas desde a década de 60, tornando-se muito importante na década de 80 e 90 com a implementação de PLA. Atualmente, com o advento da lógica reversível, minimização de funções booleanas com múltiplas saídas voltou a ser muito estudada.

Em 1961, Thomas Bartee em (BARTEE, 1961), apresentou um método para minimização de funções booleanas com múltiplas saídas. O método de Thomas Bartee constitui-se de três etapas: definição do conjunto de implicantes, baseado na tabela de combinação que contem as relações de entrada e saída do conjunto de termos; definição do conjunto de implicantes primos de múltiplas saídas, gerados a partir da conversão do conjunto de implicantes, baseado no teorema de redução; seleção dos subconjuntos dos implicantes primos.

Um método de minimização de funções booleanas muito importante foi proposto por Brayton (BRAYTON, 1984) em 1984. Denominado Espresso-II, foi desenvolvido com o objetivo de contornar as limitações do algoritmo de Quine-McCluskey. Utiliza-se de dois conceitos: Expansão e redução. A Expansão consiste em ampliar ao máximo todos os implicantes gerados e remover os que são cobertos pelo implicante expandido. A partir do conjunto gerado, é executada a fase de cobertura redundante, muito similar à segunda fase do método de Quine-McCluskey. Diferente deste, o Espresso, a partir da redução, tenta encontrar uma solução melhor que a encontrada anteriormente. Esse processo consiste em reduzir um implicante para o menor número de literais possível, e que ainda cobre a função booleana original. A Expansão e a redução são aplicadas iterativamente até que se localize uma cobertura melhor que a última encontrada. O Espresso trabalha também com funções booleanas com múltiplas saídas, e incompletamente especificadas.

A partir do Espresso, outros métodos surgiram, utilizando a mesma ideia de Expansão, de geração e de simplificação sucessiva de soluções. Muitos trabalhos utilizam como base de comparação o algoritmo Espresso e possuem características que aperfeiçoam e

melhoram os resultados obtidos por ele.

Agrawal e Biswas (AGRAWAL; AGRAWAL; BISWAS, 1985) propuseram um algoritmo para trabalhar com minimização de funções com múltiplas saídas, o qual não gera todos os implicantes primos, e utiliza o conceito de adjacência e frequência da variável no conjunto dos termos para definição do Expansão. Mesmo não contendo heurística, de acordo com o autor, mostrou ser eficiente para grandes quantidades de variáveis de entrada.

Um novo procedimento para minimização lógica denominado McBoole, baseado no armazenamento e manipulação dos termos através de grafos e de técnicas de particionamento foi proposto (DAGENAIS; AGARWAL; RUMIN, 1985). A cobertura mínima do método é baseada no método de Quine-McCluskey.

Em 1986, os autores apresentaram uma revisão no método (DAGENAIS; AGARWAL; RUMIN, 1986). Um novo procedimento e uma nova estrutura foram implementados, três listas são utilizadas, uma para armazenar os cubos formados pelos *don't care states*, DC, outra para todos os implicantes primos, U, e por último a lista R, que contem os cubos que fazem parte da cobertura mínima. A estrutura de grafo direcionado é utilizada para realização da cobertura.

Em 1987, Rudell e Sangionvanni-Vicentelli desenvolveram um algoritmo heurístico, o Espresso-MV, e um algoritmo exato, o Espresso-EXACT, ambos para minimização de funções booleanas com múltiplas saídas. O primeiro é uma extensão do Espresso-II e se difere: na leitura dos termos, na qual realiza a decomposição da função em três conjuntos, cobertura do conjunto ON, OFF e DC. O processamento inicia pela Expansão dos termos pertencentes ao conjunto ON em implicantes primos (denominada fase de Expansão). A seleção de um subconjunto destes implicantes compõe a fase redundante, na qual inclui a formação da tabela de implicantes primos e a cobertura mínima para o implicante da tabela. Os implicantes primos essenciais então são identificados na fase Essencial e são removidos da solução, do final da iteração de cobertura. A fase de Redução é a transformação da cobertura redundante de implicantes primos (solução mínima local) em uma nova cobertura. O Espresso-EXACT, uma nova técnica que utiliza da fase REDUNDANTE a tabela dos implicantes primos da função, e a cobertura do problema é realizada utilizando um algoritmo para determinar a solução mínima para o problema (RUDELL; SANGIOVANNI-VINCENTELLI, 1987).

Um algoritmo para minimização de funções com múltiplas saídas foi proposto por Gurunath e Biswas (GURUNATH; BISWAS, 1989). Utiliza uma técnica eficiente para

determinar o cubo primo essencial, mesmo sem gerar todos os cubos primos. Quatro procedimentos são priorizados: selecionar cubos primos essenciais; selecionar cubos primos principais válidos; selecionar cubos de interseção; selecionar cubos exclusivos.

Em 2003 foi proposto um algoritmo heurístico para minimização booleana em dois níveis, que diferente dos demais métodos da literatura, encontra primeiro a cobertura do *on-sets*, e então a estrutura de implicantes é derivada desta cobertura e do vetor da matriz de entrada. Implicantes não primos das funções simples são computados; são produzidos somente implicantes necessários para cobrir o *on-sets*. Esse procedimento faz com que o algoritmo seja rápido e mais eficiente para funções com um grande número de variáveis. Para a composição da cobertura da matriz de saída, o método agrupa os termos encontrando cobertura de retângulos, ou seja, forma retângulos ao agrupar os valores 1 de cada termo (FISER; HLAVICKA; KUBATOVA, 2003).

Outros trabalhos que tratam de funções booleanas com múltiplas saídas podem ser vistos em (RUDELL; SANGIOVANNI-VINCENTELLI, 1987; DAGENAIS; AGARWAL; RUMIN, 1986; PERKINS; RHYNE, 1988; GURUNATH; BISWAS, 1989; BENEDIKTSSON; SWAIN, 1992; COUDERT, 1994; LUBA, 1995; BRAYTON; KHATRI, 1999; FISER; HLAVICKA; KUBATOVA, 2003; CHOWDHURY; NAZMUL; BABU, 2006; FEN et al., 2009; JIANLIN et al., 2010a, 2010b).

Algoritmos exatos consomem muito tempo e possuem limitação no tamanho de instâncias. Por isso, o foco de muitos pesquisadores é no desenvolvimento de heurísticas ou metaheurísticas, ou na associação e aplicação dessas em métodos exatos, para encontrar soluções boas ou quase ideais dentro de um período razoável de tempo. Metaheurísticas são utilizadas como estratégias para minimizar de forma eficiente funções booleanas, principalmente aplicadas à fase de cobertura dos mintermos, dentre os procedimentos desenvolvidos pode-se citar Algoritmos Genéticos (BANSAL; AGARWAL, 2013; ASTHANA; VERMA; RATAN, 2014; SHIRINZADEH; SOEKEN; DRECHSLER, 2015; BAJPAYEE et al., 2016; KAZIMIROV; REIMEROV, 2016), Otimização de Colônia de Formigas (*Ant Colony Otimization - ACO*) (AL-SHIHABI; ARAFEH; BARGHASH, 2015; AL-SHIHABI, 2016), e variações dessas idéias como: *Binary Black Hole Algorithm* (SOTO et al., 2016), *Binary Firefly Algorithm* (CRAWFORD et al., 2014),*Binary Bat Algorithm* (DAHI; MEZIOUD; DRAA, 2015; CRAWFORD et al., 2015).

Atualmente com a popularização e miniaturização de equipamentos eletrônicos, o foco dos estudos a respeito de minimização de funções booleanas aplica-se também na área de lógica reversível. Os circuitos reversíveis são mais adequados para os equipamentos

eletrônicos por dissiparem menos calor e, o que implica em menor perda de informação, isto é, quase nula. Alguns trabalhos na literatura a respeito de minimização lógica para circuitos reversíveis podem ser encontrados em (MILLER; MASLOV; DUECK, 2003; HUNG et al., 2006; WILLE et al., 2012; DRECHSLER; WILLE, 2012; MORRISON; RANGANATHAN, 2013; BAHAR; WAHEED; HABIB, 2014; RAWSKI; SZOTKOWSKI, 2015; DEB et al., 2015; WILLE; LYE; NIEMANN, 2016).

Muitos dos trabalhos apresentados foram utilizados para aprimorar a técnica desenvolvida e também para reforçar a importância do estudo e implementação de métodos para minimização de funções booleanas, seja para a geração de implicantes primos ou para a cobertura do mintermos. Salienta-se que a utilização de funções booleanas com múltiplas saídas atualmente se tornou foco de pesquisas, principalmente devido a área de lógica reversível.

No próximo capítulo, são apresentados alguns conceitos relacionados à álgebra booleana, e assim como fundamentos para melhor entendimento deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

O número de variáveis que compõem uma função booleana pode afetar diretamente o custo do processamento computacional, por esse fator estar diretamente ligado ao número de portas lógicas que compõem um circuito. Otimizar as funções booleanas antes de implementá-las em *hardware*, pode trazer algumas vantagens em relação ao número de entradas nas portas lógicas na composição do circuito lógico. Com a redução no número de entradas das portas lógicas, consequentemente, há uma diminuição do custo do *hardware*, uma menor dissipação de calor gerado pelo *chip* e, também, um aumento na velocidade de processamento.

Diversos teoremas e propriedades podem ser aplicados a uma função booleana com intuito de reduzir o número de variáveis, minimizando o custo do circuito elétrico. Além disso, métodos e algoritmos com o intuito de otimizar ainda mais as funções booleanas vêm sendo desenvolvidos e aprimorados a cada dia.

Neste capítulo, são apresentadas algumas propriedades relacionadas à álgebra booleana. Também são descritos alguns fundamentos a respeito funções booleanas, assim como simplificação destas.

2.1 ÁLGEBRA BOOLEANA

A álgebra Booleana pode ser definida por um conjunto de termos finitos (formados por variáveis[1] ou constantes que assumem os valores (0, 1), por operadores binários ($+$, $\cdot$) e unário (complemento, representado por $\overline{\ }$), definidos sobre os valores das variáveis.

Os valores e operadores da álgebra booleana são análogos aos do cálculo de preposições. Os valores 0 e 1, correspondem a $falso$ e $verdadeiro$, respectivamente, enquanto os operadores $+$, $\cdot$ e $\overline{\ }$ correspondem a *and*, *or* e *not* da lógica de proposições. Um circuito digital pode ser construído utilizando-se as portas lógicas OR, AND e NOT. As operações

[1] Variável é um símbolo usado para representar uma grandeza lógica, e possui valor 1 ou 0. (FLOYD, 2007)

com cada um dos operadores são apresentadas na Tabela 1.

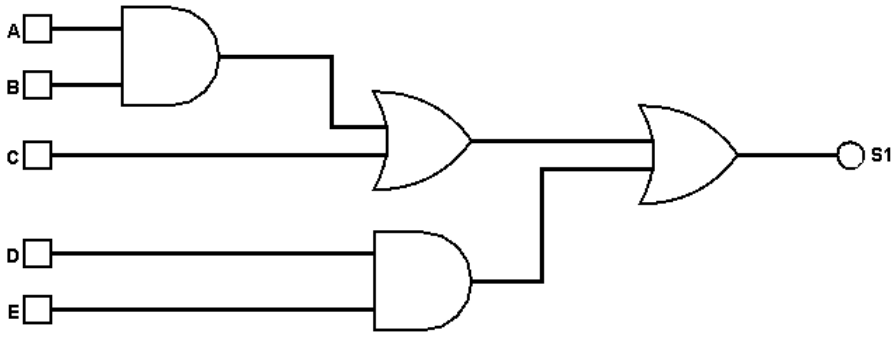
Tabela 1 – Operações OR, AND e NOT

Variáveis		Operações		
A	B	$A+B$	$A \cdot B$	$\overline{A}$
0	0	0	0	1
0	1	1	0	
1	0	1	0	0
1	1	1	1	

A operação OR é equivalente à adição booleana, e a soma dos literais de uma expressão é dita como termo-soma. Já a multiplicação booleana está relacionada à operação AND, e o termo-produto é a multiplicação de literais de uma expressão.

Um circuito composto por portas lógicas é denominado circuito lógico. A Figura 1 ilustra um exemplo de um circuito lógico que realiza a expressão lógica $A \cdot B + C + D \cdot E$.

Figura 1 – Representação de uma Expressão Lógica em um Circuito Lógico



Fonte: Elaborado pelo Autor.

As simplificações são realizadas devido a diversas razões, dentre elas, para que a expressão fique mais simples e, consequentemente, mais fácil de entender, e também por possibilitar menor probabilidade de erros na interpretação. A simplificação quando implementada corretamente, proporciona maior eficiência aos sistemas digitais, devido a um número menor de portas utilizadas, diminuição da dissipação de calor e menor tempo de processamento.

2.1.1 Álgebra booleana binária

Uma expressão booleana é uma expressão matemática, composta por variáveis e termos (conjunto de variáveis). A simplificação de uma expressão booleana pode levar a algoritmos e a circuitos lógicos mais eficazes. Pode ser minimizada da mesma forma

como as expressões algébricas. De acordo com o matemático George Boole, um conjunto de regras define as proposições cujos resultados são: verdadeiro ou falso. Em relação à lógica digital, as regras são utilizadas para descrever os circuitos em que o estado pode ser 1 ou 0, verdadeiro ou falso, respectivamente (FLOYD, 2007).

Assim como na álgebra tradicional, na booleana algumas regras, leis e postulados são fundamentais para que a minimização de expressões possa ser realizada (FLOYD, 2007). Na Tabela 2, os postulados são apresentados.

Tabela 2 – Postulado Booleano

P1:	$A = 1$ se $A \neq 0$
P2:	$A = 0$ se $A \neq 1$
P3:	$0 \cdot 0 = 0$
P4:	$1 + 1 = 1$
P5:	$0 + 0 = 0$
P6:	$1 \cdot 1 = 1$
P7:	$1 \cdot 0 = 0 \cdot 1 = 0$
P8:	$1 + 0 = 0 + 1 = 1$
P9:	$\overline{1} = 0$
P10:	$\overline{0} = 1$

Fonte: Elaborado pelo Autor.

Na Tabela 3, as leis da Álgebra de Boole podem ser visualizadas. As expressões podem ser escritas utilizando-se os operadores OR e AND, ou vice-versa. Essa inversão das operações é conhecida como dualidade, como pode ser observado na primeira e segunda linhas de cada propriedade.

Tabela 3 – Leis da Álgebra de Boole

Comutativa	$A + B = B + A$ $A \cdot B = B \cdot A$
Associativa	$(A + B) + C = A + (B + C)$ $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
Distributiva	$A \cdot (B + C) = A \cdot B + A \cdot C$ $A + (B \cdot C) = (A + B) \cdot (A + C)$

Fonte: Elaborado pelo Autor.

Na Lei Comutativa a ordem das variáveis, nas quais a função OR/AND é aplicada, não altera o resultado. Na Lei Associativa, quando é aplicada uma operação OR/AND

em mais de duas variáveis, a ordem em que elas são agrupadas também não modifica o resultado final. A Distributiva expressa o processo de fatoração no qual a variável comum A é fatorada em termos-produto ou em termos-soma.

Os Teoremas da Álgebra Booleana, representados na Tabela 4, são de suma importância para a minimização de expressões, visto que podem reduzir consideravelmente o número de variáveis existentes.

Tabela 4 – Teoremas da Álgebra de Boole

T1:	$0 + A = A$ $0 \cdot A = 0$
T2:	$1 + A = 1$ $1 \cdot A = A$
T3: Identidade	$A + A = A$ $A \cdot A = A$
T4:	$(A \cdot B) + (A \cdot \overline{B}) = A$ $(A + B) \cdot (A + \overline{B}) = A$
T5: Redundante	$A + A \cdot B = A$ $A \cdot (A + B) = A$
T6: Complemento	$A + \overline{A} = 1$ $A \cdot \overline{A} = 0$
T7:	$A + (A \cdot \overline{B}) = A + B$ $A \cdot (A + \overline{B}) = A \cdot B$
T8:	$\overline{\overline{A}} = A$
T9:	$(A + B) \cdot (A + C) = A + B \cdot C$
T10:De Morgan	$(\overline{A + B}) = \overline{A} \cdot \overline{B}$ $(\overline{A \cdot B}) = \overline{A} + \overline{B}$

Fonte: Elaborado pelo Autor.

2.2 FUNÇÕES BOOLEANAS

Uma função booleana é definida pelos valores que esta pode assumir a partir das combinações possíveis de suas variáveis. Ou seja, uma função booleana $f(x_1, ..., x_n)$ é a correspondência que associa um elemento da Álgebra de Boole com cada uma das 2^n combinações das variáveis x_1, ...,x_n. As combinações possíveis de x_1,x_2,...,x_n formam a tabela verdade, que define uma função (SILVA, 1989).

Seja uma função booleana, representada pela expressão 1.

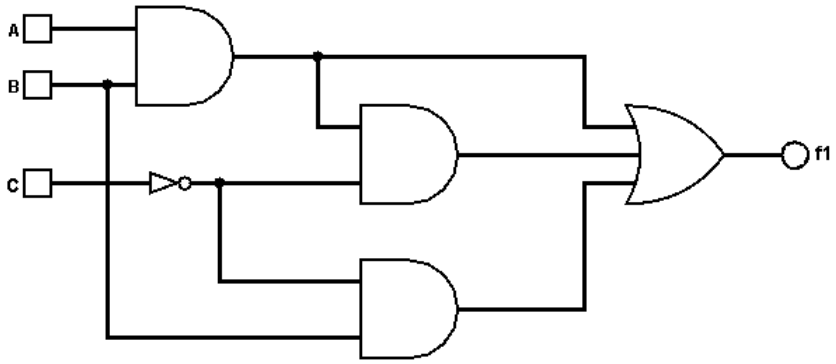
$$f_1(A, B, C) = A \cdot B + A \cdot B \cdot \overline{C} + B \cdot \overline{C} \tag{1}$$

a operação representada por $\cdot$ pode ser omitida, logo a mesma função pode ser reescrita:

$$f_1(A, B, C) = AB + AB\overline{C} + B\overline{C}$$

O circuito lógico referente a função f_1, pode ser visualizado na Figura 2.

Figura 2 – Representação de f_1 em um Circuito Lógico



Fonte: Elaborado pelo Autor.

A tabela verdade, que define a função, apresentada na expressão 1, pode ser visualizada na Tabela 5.

Tabela 5 – Tabela Verdade para a Função f_1

Variáveis			Termo Produto			Resultado da Função
A	B	C	AB	$AB\overline{C}$	$B\overline{C}$	$f_1(A, B, C)$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	1	1
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	1	1	1
1	1	1	1	0	0	1

Fonte: Elaborado pelo Autor.

Outra forma de representar uma função booleana é através da derivação da tabela verdade, na qual uma expressão pode ser encontrada através da soma de todos os termos cuja função assume o valor 1. Cada termo dessa expressão é constituído pelo produto dos literais (variável da função complementada ou não).

A partir da representação de f_1 pela Tabela 5, a função pode ser reescrita utilizando-se as linhas referentes às variáveis nas quais a função resultou em 1. Nesse caso, devem ser selecionadas três linhas da tabela, em que:

- $A = 0$, $B = 1$ e $C = 0 \rightarrow$ Termo: $\overline{A}B\overline{C}$
- $A = 1$, $B = 1$ e $C = 0 \rightarrow$ Termo: $AB\overline{C}$
- $A = 1$, $B = 1$ e $C = 1 \rightarrow$ Termo: ABC

Logo, a função booleana, f_1, pode ser reescrita, através da forma canônica, como a soma dos produtos (termos pertencentes a função), representação esta chamada de **soma de produtos**, como descrito na expressão 2.

$$f_{1'}(A, B, C) = \overline{A}B\overline{C} + AB\overline{C} + ABC \tag{2}$$

Um produto (ou termo-produto) é uma expressão booleana que consiste de conjunto de literais que não envolve as mesmas variáveis. Conforme já definido, o produto canônico ou mintermo, é um produto em que cada variável ocorre uma única vez (sendo essa complementada ou não). Isto é, mintermo é o termo produto resultante da combinação das variáveis, cujo resultado da função é 1.

Logo, para o exemplo, os mintermos pertencentes a função $f_{1'}$ são: $\overline{A}B\overline{C}$,$AB\overline{C}$ e ABC. Em decimais, os valores correspondentes seriam: 2, 6 e 7. Assim, a função f_1 também pode ser descrita conforme a expressão 3, em que m representa os mintermos da função dada, e $\sum$ a soma dos mintermos.

$$f_{1'}(A, B, C) = \sum m(2, 6, 7) \tag{3}$$

De forma análoga, pode-se obter a representação de uma função booleana através do produto da soma. Porém, os termos considerados para a composição da função são aqueles em que o resultado da função é 0. De acordo com a Tabela 5, a função f_1 pode ser reescrita utilizando-se as linhas nas quais o resultado da função é 0. As linhas da tabela correspondente ao resultado 0 são:

- $A = 0$, $B = 0$ e $C = 0 \rightarrow$ Termo: $\overline{A}\ \overline{B}\ \overline{C}$
- $A = 0$, $B = 0$ e $C = 1 \rightarrow$ Termo: $\overline{A}\ \overline{B}\ C$
- $A = 0$, $B = 1$ e $C = 1 \rightarrow$ Termo: $\overline{A}\ B\ C$

- $A = 1$, $B = 0$ e $C = 0 \rightarrow$ Termo: $A\ \overline{B}\ \overline{C}$
- $A = 1$, $B = 0$ e $C = 1 \rightarrow$ Termo: $A\ \overline{B}\ C$

A função booleana f_1 pode ser reescrita, através da forma canônica, produto da soma dos termos pertencentes a função, conforme expressão 4, representação esta chamada de produto da soma.

$$f_{1''}(A, B, C) = (\overline{A} + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + C) \cdot (\overline{A} + B + C) \cdot (A + \overline{B} + \overline{C}) \cdot (A + \overline{B} + C) \quad (4)$$

Uma soma (ou termo soma) que contenha as n variáveis de uma função como fatores, sendo essas complementadas ou não, é definida como **maxtermo** (MENDELSON, 1970). Ou seja, o maxtermo pertencente a uma função é o termo soma resultante da combinação das variáveis, em que o resultado da função é 0. O conjunto de maxtermos de uma função é representado por M.

Para o exemplo descrito na expressão 4, os maxtermos pertencentes a função $f_{1''}$ são: $\overline{A} + \overline{B} + \overline{C}$, $\overline{A} + \overline{B} + C$, $\overline{A} + B + C$, $A + \overline{B} + \overline{C}$, $A + \overline{B} + C$. Em decimais, os valores correspondentes seriam: 0, 1, 3, 4 e 5. Assim a função definida pela Tabela 5 pode ser reescrita conforme expressão 5, em que $\prod$ representa o produto dos maxtermos pertencentes a função dada.

$$f_{1'}(A, B, C) = \prod M(0, 1, 3, 4, 5) \quad (5)$$

Existem algumas funções que certas combinações das variáveis de entrada correspondem a situações que são irrelevantes para o funcionamento do projeto. Logo, para estas combinações, o resultado da função é descrito por um estado irrelevante (*Don't Care State - DC*), representado por X ou por um traço $(-)$. Quando existem estados irrelevantes em uma função, o conjunto destes é representado pela letra D.

Considere a função f_2, conforme expressão 6, a tabela verdade com a representação do resultado da função pode ser visualizada na Tabela 6.

$$f_2(A, B, C) = \sum (m(1, 3, 4) + D(0, 6)) \quad (6)$$

Os *don't care states* são considerados na Tabela 6, sendo estes representados no resultado da função pelo caracter X. Podem ser utilizados tanto na representação de soma de produto quanto de produto de soma.

Para o exemplo representado na Tabela 6, os *don't care states* estão presentes em:

Tabela 6 – Tabela Verdade para a Função f_2

Variáveis			Resultado da Função
A	B	C	$f_2(A, B, C)$
0	0	0	X
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	X
1	1	1	0

Fonte: Elaborado pelo Autor.

- $A = 0$, $B = 0$ e $C = 0 \rightarrow$ Termo: $\overline{A}\ \overline{B}\ \overline{C}$
- $A = 1$, $B = 1$ e $C = 0 \rightarrow$ Termo: $A\ B\ \overline{C}$

Um exemplo prático da utilização de estados irrelevantes, seria no monitoramento do nível (baixo e alto) de água em uma caixa d'água. Utiliza-se dois sensores, um no topo da caixa, e o outro próximo a base. Sempre que a caixa estiver vazia ou cheia um sinal sonoro é enviado, representado por 1. Quando em contato com a água o sensor terá o sinal 1 e quando não estiver em contato o sinal será 0.

Para este exemplo, duas entradas (sinais dos dois sensores) e uma saída (sinal do alarme) fazem parte do problema. Quatro combinações de entrada são possíveis (00, 01, 10 e 11). Para a combinação 01 a saída é irrelevante, pois não há possibilidade do sensor que está na base da caixa ter sinal 0 (ausência de água) e o sensor no topo da caixa ter sinal 1 (presença de água).

2.3 SIMPLIFICAÇÃO DE FUNÇÕES BOOLEANAS

A simplificação de funções booleanas é muito importante por possibilitar a redução de uma expressão de um circuito lógico em uma forma mais simples, com um custo menor. Neste trabalho, entende-se por menor custo, a expressão que contiver menor número de termos ou variáveis, o que implica em um circuito com menor quantidade de variáveis de entrada nas portas lógicas.

Os métodos mais frequentes para minimizar uma expressão ou função booleana são: Método Algébrico, Mapa de Karnaugh (KARNAUGH, 1953) e Método Tabular como o de Quine-McCluskey (MCCLUSKEY, 1986).

2.3.1 Método algébrico

Utiliza-se das leis, regras e teoremas da álgebra booleana para minimizar as expressões. Porém, depende muito da familiaridade e habilidade de cada projetista em aplicá-las, o que pode resultar em expressões que não estejam na forma mínima.

As regras e os teoremas da álgebra booleana, apresentadas na subseção 2.1.1 deste capítulo, são utilizados para minimizar funções booleanas.

A partir das regras e teoremas apresentados nas Tabelas 2, 3 e 4, dada uma função f_3, representada pela expressão 7, é apresentado a seguir em detalhes o conjunto de passos para minimizar a função dada. Salienta-se que este não é o único caminho para a minimização da função.

$$f_3(A, B, C) = A((B + \overline{C})(\overline{B} + C)) + AB + (\overline{A} + \overline{B})(B + \overline{C}) \tag{7}$$

Resolução passo a passo, de minimização de f_3 utilizando o método algébrico:

- $f_3 = A((B + \overline{C})(\overline{B} + C)) + AB + (\overline{A} + \overline{B})(B + \overline{C})$

  Lei Distributiva: $(B + \overline{C})(\overline{B} + C) = B\overline{B} + BC + \overline{B}\,\overline{C} + C\overline{C}$

  Lei Distributiva: $(\overline{A} + \overline{B})(B + \overline{C}) = \overline{A}B + \overline{A}\,\overline{C} + B\overline{B} + \overline{B}\,\overline{C}$

- $f_3 = A(B\overline{B} + BC + \overline{B}\,\overline{C} + C\overline{C}) + AB + (\overline{A}B + \overline{A}\,\overline{C} + B\overline{B} + \overline{B}\,\overline{C})$

  T6 Complemento: $B\overline{B} = 0$

  T6 Complemento: $C\overline{C} = 0$

- $f_3 = A(0 + BC + \overline{B}\,\overline{C} + 0) + AB + (\overline{A}B + \overline{A}\,\overline{C} + 0 + \overline{B}\,\overline{C})$

- $f_3 = A(BC + \overline{B}\,\overline{C}) + AB + \overline{A}B + \overline{A}\,\overline{C} + \overline{B}\,\overline{C}$

  Lei Distributiva: $A(BC + \overline{B}\,\overline{C}) = ABC + A\overline{B}\,\overline{C}$

- $f_3 = ABC + A\overline{B}\,\overline{C} + AB + \overline{A}B + \overline{A}\,\overline{C} + \overline{B}\,\overline{C}$

  T5 Redundante: $ABC + AB = AB$

  T5 Redundante: $A\overline{B}\,\overline{C} + \overline{B}\,\overline{C} = \overline{B}\,\overline{C}$

- $f_3 = AB + \overline{B}\,\overline{C} + \overline{A}B + \overline{A}\,\overline{C}$

  T4: $AB + \overline{A}B = B$

- $f_3 = B + \overline{B}\,\overline{C} + \overline{A}\,\overline{C}$

  Lei Distributiva : $= B + \overline{B}\,\overline{C} = (B + \overline{B})(B + \overline{C})$

- $f_3 = (B + \overline{B})(B + \overline{C}) + \overline{A}\,\overline{C}$

  T6 Complemento: $B + \overline{B} = 1$

- $f_3 = 1(B + \overline{C}) + \overline{A}\,\overline{C}$

- $f_3 = B + \overline{C} + \overline{A}\,\overline{C}$

  O literal $\overline{C}$ é colocado em evidência

- $f_3 = B + \overline{C}(1 + \overline{A})$

  T2: $1 + \overline{A} = 1$

- $f_3 = B + \overline{C}(1)$

- $f_3 = B + \overline{C}$

Os passos são bem intuitivos, assim como a ordem de simplificação, o que demanda conhecimento e habilidade por parte do usuário. Não há uma forma exata, ou um procedimento único para que a minimização de funções através do método algébrico.

Quanto maior o número de variáveis mais complexo se torna a utilização do método. Por isso, resultados diferentes podem ser alcançados e não há garantia de que a função obtida seja a de menor custo.

Considerando a minimização de forma manual, é apresentado o Mapa de Karnaugh, método visual, o que o torna mais usual que o algébrico, mas também limitado quanto ao número de variáveis.

2.3.2 Mapa de Karnaugh - MK

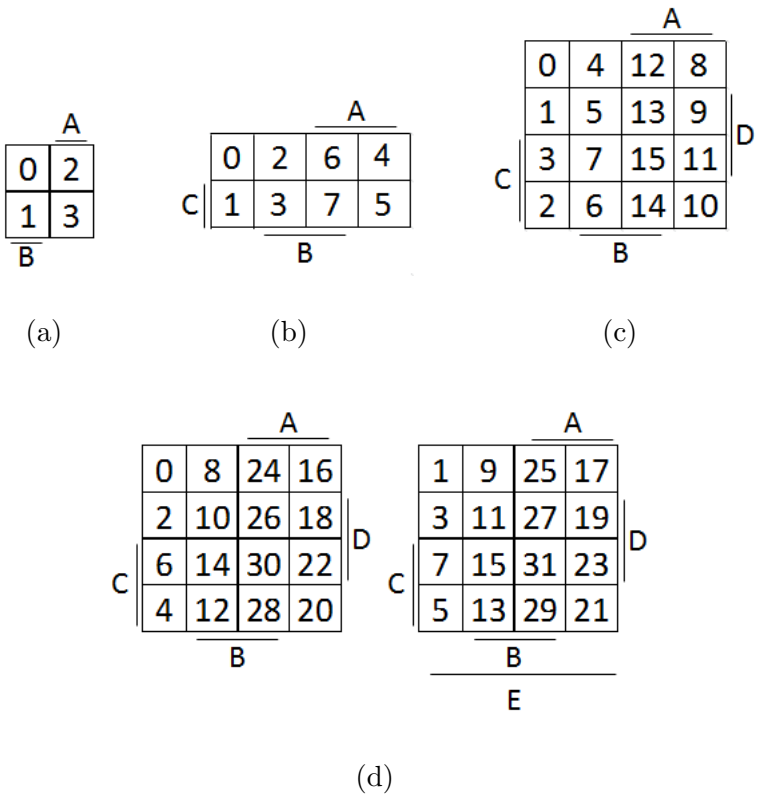
Método sistemático para simplificação de expressões Booleanas, baseado na identificação visual de grupos de mintermos que podem ser simplificados (KARNAUGH, 1953). MK é considerado como um arranjo especial da tabela verdade, por conter a representação de todos os termos da função.

Para uma função que contenha n variáveis de entrada, o Mapa de Karnaugh é formado por 2^n células. Por exemplo, uma função com 5 variáveis é representada no mapa com 32 células.

A representação do MK de duas, três, quatro e cinco variáveis pode ser visualizada nas Figuras 3(a), 3(b), 3(c) e 3(d), respectivamente.

Na representação, considera-se a variável A como a mais significativa do termo. Quanto maior o número de variáveis, mais complexa a análise e a utilização do método. Como é uma representação gráfica, é muito utilizado para resolução de problemas de até 6 variáveis.

Figura 3 – Mapa de Karnaugh com 2, 3, 4 e 5 variáveis



Fonte: Elaborado pelo Autor.

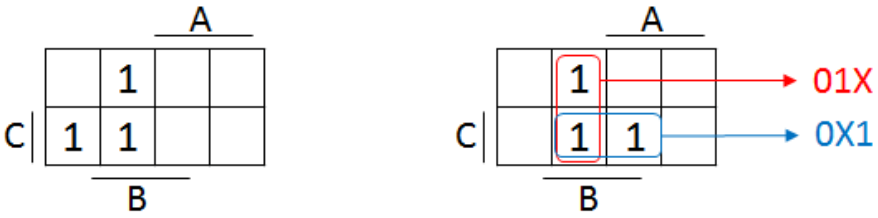
A partir de uma tabela verdade ou de uma função booleana, monta-se o Mapa de Karnaugh inserindo o valor 1 ou 0 na célula que representa o termo (mintermo ou maxtermo, respectivamente).

Dada a função f_4, expressão 8, a representação dos termos da função no Mapa de Karnaugh e implicantes gerados a partir da simplificação podem ser observadas nas Figuras 4(a) e 4(b).

$$f_4(A, B, C) = \sum m(2, 3, 7) \tag{8}$$

No mapa, os mintermos 2 ($\overline{A}B\overline{C}$) e 3 ($\overline{A}BC$) são adjacentes, ou seja, diferem em apenas um *bit*, por isso podem ser combinados, resultando na composição $01X$ ou $\overline{A}B$. O

Figura 4 – Exemplo de Montagem e Simplificação dos termos de f_4 no Mapa de Karnaugh



(a) (b)

Fonte: Elaborado pelo Autor.

mesmo ocorre com os mintermos 3 ($\overline{A}BC$) e 7 (ABC), que resulta no implicante primo $X11$ ou BC.

Após a simplificação, a função f_4, expressão 8, pode ser reescrita como:

$$f_{4'}(A, B, C) = \overline{A}B + BC$$

.

A simplificação se dá quando é possível agrupar células adjacentes, ou seja, células que são vizinhas (inferior, superior, direito, esquerdo), isto é, que diferem em apenas um *bit*. O mapa deve ser analisado de forma cilíndrica, células na última linha são adjacentes às correspondentes na primeira linha. As células na coluna mais à direita (extremidade) são adjacentes à correspondentes na coluna mais à esquerda.

Outro exemplo pode ser visualizado, considerando a função f_5, expressão 9.

$$f_5(A, B, C, D) = \sum m(0, 2, 5, 7, 13, 15) \tag{9}$$

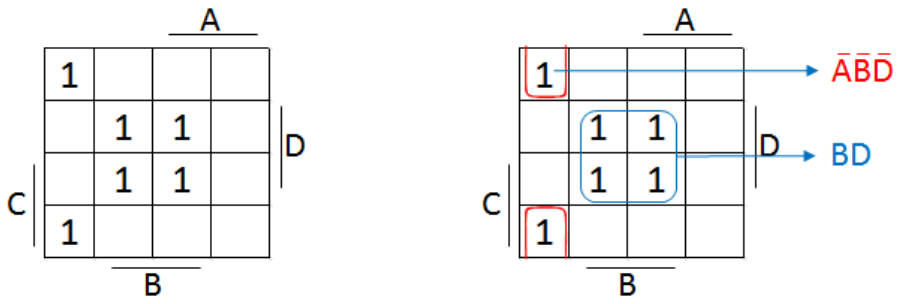
A montagem do Mapa de Karnaugh, referente a expressão 9, pode ser visualizada na Figura 5(a). Na Figura 5(b) tem-se o resultado da simplificação.

Os subcubos gerados (agrupamento adjacente das células) formam o conjunto de implicantes primos de uma função e, para f_5 este é composto por: $\overline{A}\,\overline{B}\,\overline{D}$ e BD.

Após a minimização, a função f_5 pode ser reescrita como:

$$f_{5'}(A, B, C, D) = \overline{A}\,\overline{B}\,\overline{D} + BD$$

Figura 5 – Simplificação da função f_5 utilizando Mapa de Karnaugh



(a) (b)

Fonte: Elaborado pelo Autor.

Os estados irrelevantes, *don't care states* também devem ser usados no mapa para a obtenção de uma expressão mínima, e são representados por X. Agrupamentos que contenham apenas *don't care states* não devem ser considerados na função mínima.

Seja a função f_6, expressão 10, que possui *don't care states*. A representação dos termos de f_6 no Mapa de Karnaugh pode ser visualizada na Figura 6(a), e o resultado obtido na Figura 6(b).

$$f_6(A, B, C, D) = \sum(m(0, 2, 5, 7, 8, 14) + D(10, 11)) \tag{10}$$

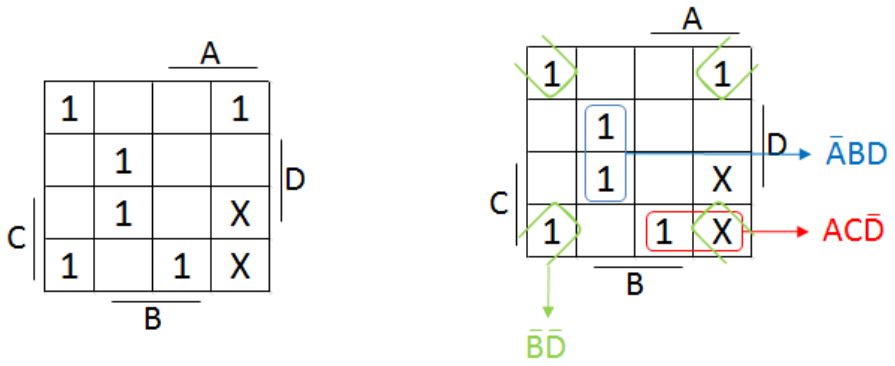
Os subcubos formados a partir da combinação das células adjacentes são ditos como implicantes, gerados a partir da associação entre mintermos e também entre mintermos e *don't care states*. Os implicantes primos gerados pela minimização de f_6 são: $\overline{A}BD$, $\overline{B}\ \overline{D}$, $AC\overline{D}$.

A função mínima obtida a partir da utilização da simplificação, através do Mapa de Karnaugh, está representada na expressão 11.

$$f_{6'}(A, B, C) = \overline{A}BD + \overline{B}\ \overline{D} \tag{11}$$

A função mínima é obtida após a análise de todos os implicantes primos gerados. O conjunto dos subcubos deve cobrir todos os mintermos, ou seja, um minterm deve ser coberto pelo menos por um dos subcubos selecionados. A designação dos implicantes primos pode ser feita da seguinte maneira:

Figura 6 – Representação e Minimização dos termos de f_6 utilizando o Mapa de Karnaugh



(a) (b)

Fonte: Elaborado pelo Autor.

- Caso exista subcubo formado por apenas um elemento, ou seja, mintermo isolado, este deverá pertencer a função mínima, pois não está contido em nenhum outro;
- Verificar se existe algum subcubo que não esteja contido em nenhum outro, se existir, este também deverá fazer parte da função mínima;
- Caso exista mais de um subcubo que possa ser selecionado para a cobertura de mintermos, a escolha deve ser feita de acordo com o número de variáveis, ou seja, é escolhido o menor. Se estes forem de mesmo tamanho, a escolha pode ser feita aleatoreamente.

Como já definido, um implicante é um termo produto, em que todas as variáveis, pertencentes a função, aparecem uma única vez, complementada ou não. Os mintermos devem ser cobertos por, pelo menos, um implicante. Quando não há possibilidades de reduzir um implicante, este é dito como implicante primo.

Os conceitos apresentados neste capítulo são importantes para melhor compreensão dos próximos. O Mapa de Karnaugh representa uma forma visual para minimização de funções booleanas, não sendo muito utilizado para funções com mais de 6 variáveis.

No Capítulo 3 é apresentado a importância de funções booleanas com múltiplas saídas, assim como a aplicação do Mapa de Karnaugh para o tipo de função. A formulação do problema de cobertura através da programação linear inteira 0 e 1 também é descrita. Para fins de comparação com as propostas desenvolvidas, a Primeira Fase do Método de

Quine-McCluskey para Múltiplas Saídas é apresentada também no capítulo.

3 MINIMIZAÇÃO DE FUNÇÕES BOOLEANAS COM MÚLTIPLAS SAÍDAS

Muitas vezes, circuitos lógicos possuem várias saídas ao invés de apenas uma. Atualmente, grande avanço na área de minimização de funções com múltiplas saídas se deve à aplicação desta no desenvolvimento de circuitos para microequipamentos.

Muitos problemas reais envolvem a concepção de um sistema com mais de uma saída. Por exemplo, um problema com três entradas, A, B e C e duas saídas, X e Y, pode ser tratado de duas formas. Como dois problemas distintos (no qual cada função seria mapeada individualmente e a solução mínima encontrada para cada uma delas), ou como um único sistema com três entradas e duas saídas (funções são minimizadas em conjunto). Através de um único sistema pode ocorrer a partilha de portas lógicas, o que gera economia na elaboração do circuito lógico.

Uma função booleana com múltiplas saídas pode ser definida como sendo um conjunto de m funções que dependam das mesmas n variáveis. Em que m e n devem ser número inteiros maiores que 1.

Neste capítulo é apresentado alguns conceitos básicos a respeito de funções booleanas com múltiplas saídas, assim como sua representação no Mapa de Karnaugh. O problema de cobertura também é descrito neste capítulo, no qual a formulação para o problema é realizada através da Programação Linear Inteira (PLI) 0 e 1. Para fins de posterior comparação com as propostas desenvolvidas, é descrita e exemplificada a Primeira Fase do Método de Quine-McCluskey adaptada para Funções com Múltiplas Saídas.

3.1 REPRESENTAÇÃO E SIMPLIFICAÇÃO DE FUNÇÃO BOOLEANA COM MÚLTIPLAS SAÍDAS UTILIZANDO O MAPA DE KARNAUGH

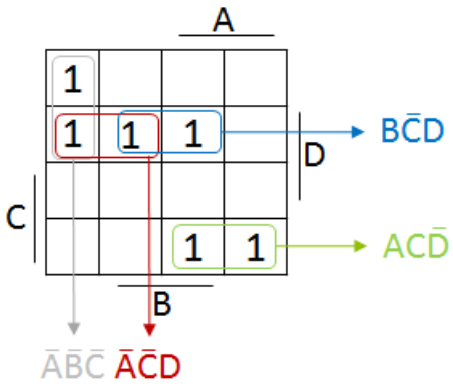
O Mapa de Karnaugh pode ser estendido para a simplificação de funções booleanas com múltiplas saídas. Considere $F_{M_1} = \{f_7, f_8, f_9\}$. As funções de F_{M_1} estão descritas na

expressão 12.

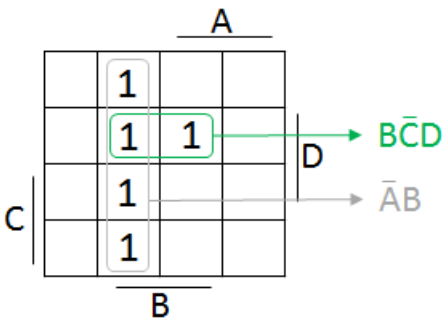
$$\begin{aligned} f_7 &= \sum(m(0,1,5,10,13,14)) \\ f_8 &= \sum(m(4,5,6,7,13)) \\ f_9 &= \sum(m(5,10,13,14)) \end{aligned} \tag{12}$$

Para múltiplas saídas, inicialmente representa-se, no Mapa de Karnaugh, cada função separadamente, conforme Figuras 7(a), 7(b) e 7(c).

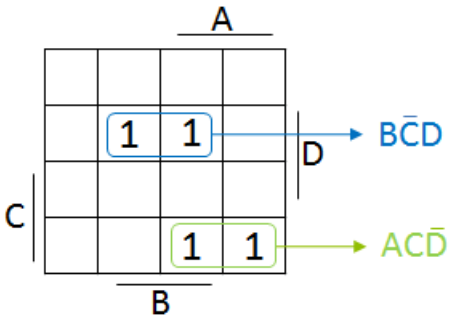
Figura 7 – Representação dos Implicantes Primos das Funções de F_{M_1} no Mapa de Karnaugh



(a) f_7



(b) f_8



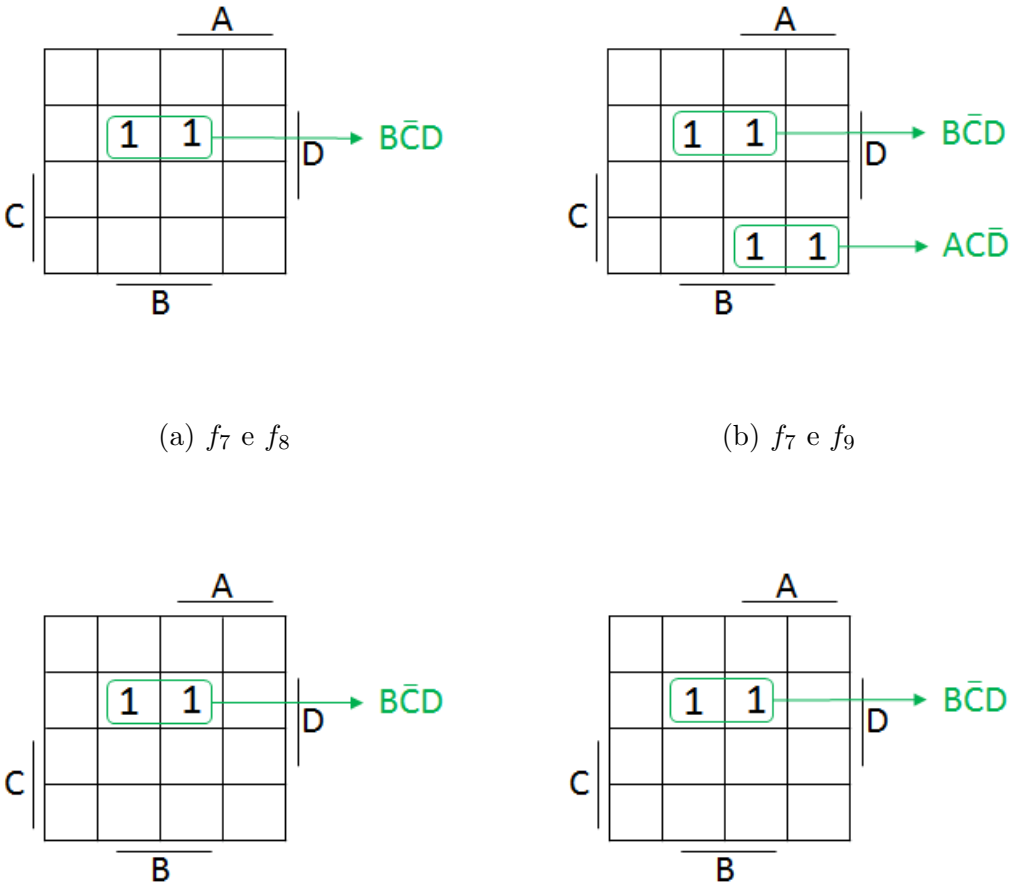
(c) f_9

Fonte: Elaborado pelo Autor.

Em seguida, o mapa deve ser elaborado para as composições: f_7 e f_8; f_7 e f_9; f_8 e f_9, assim como para f_7, f_8 e f_9. A combinação dos termos em comum deve ser feita, assim

como nos mapas anteriores. As Figuras 8(a), 8(b) representam os implicantes obtidos pelo Mapa de Karnaugh da combinação das funções, estes são compostos pela seleção dos termos que cobrem os mintermos em comuns a essas funções. O mapa do implicante primo comum às três funções pode ser observado na Figura 8(d).

Figura 8 – Representação dos Implicantes Primos das Combinações das Funções de F_{M_1} no Mapa de Karnaugh



(a) f_7 e f_8

(b) f_7 e f_9

(c) f_8 e f_9

(d) f_7, f_8 e f_9

Fonte: Elaborado pelo Autor.

O implicante primo $AC\overline{D}$ cobre os mintermos 10 e 14 pertencentes às funções f_7 e f_9. Os mintermos 5 e 13, estão presentes nas três funções e são cobertos pelo implicante primo $B\overline{C}D$.

A função mínima encontrada para F_{M_1} é:

$f_7 = AC\overline{D} + B\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}$;

$f_8 = B\overline{C}D + \overline{A}B$;

$f_9 = AC\overline{D} + B\overline{C}D$.

Como as funções possuem implicantes primos comuns, esses são gerados uma única vez e compartilhados entre as funções. Logo, contabiliza-se as entradas das portas AND do implicante primo comum uma única vez. O $custo_{F_{M_1}} = custo_{or} + custo_{and}$, o $custo_{and} = 11$ e o $custo_{or} = 7$. O custo da função mínima para F_{M_1} é 18.

Pode-se observar, ao minimizar as funções separadamente, que implicantes primos são gerados e contabilizados em duplicidade, o que não ocorre quando se minimiza funções com múltiplas saídas.

Se considerarmos cada função individualmente, os custos relacionados às funções mínimas seriam: $custo_{f_7} = 12$, $custo_{f_8} = 7$ e $custo_{f_9} = 8$, contabilizando um $custo_{total} = 27$. A diferença de custo reforça a vantagem da minimização através de funções com múltiplas saídas como um único sistema.

Minimizar individualmente cada função não representa a melhor solução para a minimização de múltiplas saídas. Quando se trata de minimizar funções booleanas com múltiplas saídas, alguns casos não triviais acontecem. Muitas vezes um termo comum à duas funções, pode não ser implicante primo dessas funções, mas é um implicante primo múltiplo. Isto é, é um implicante primo gerado a partir da combinação das duas funções.

Considere o conjunto de funções, $F_{M_2} = \{f_{10}, f_{11}\}$, representadas na expressão 13.

$$\begin{aligned} f_{10}(A,B,C) &= \sum m(0,2,6) \\ f_{11}(A,B,C) &= \sum m(0,1,3) \end{aligned} \tag{13}$$

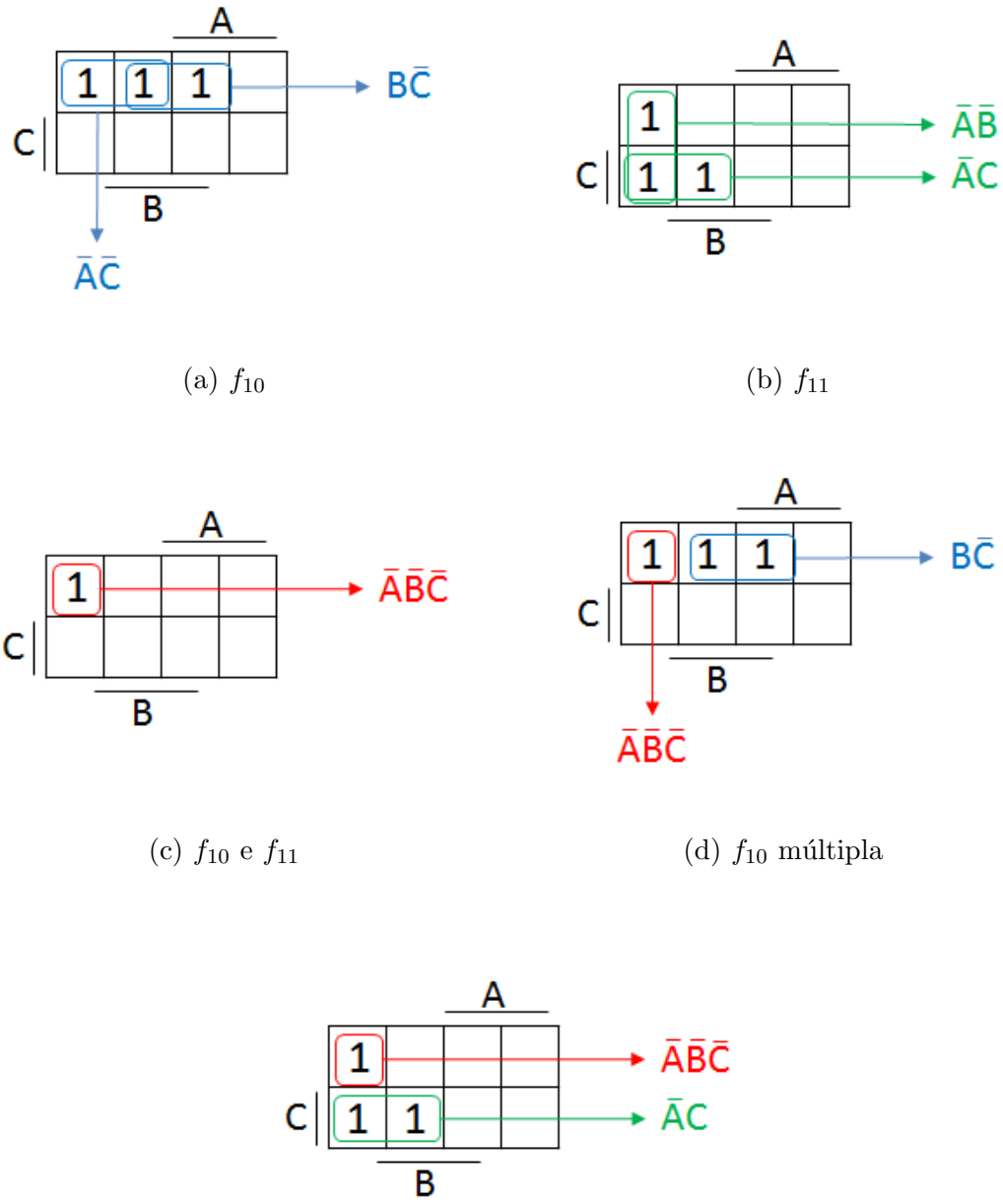
A simplificação das funções f_{10} e f_{11}, está representada no Mapa de Karnaugh, conforme Figuras 9(a) e 9(b), assim como a combinação entre as funções na Figura 9(c).

Após a cobertura, os implicantes primos gerados são: $\{X10, 10\}$, $\{0X1, 01\}$ e $\{000, 11\}$. Os Mapas de Karnaugh que representam o resultado ótimo para cada uma das funções que compõe F_{M_2} estão representados nas Figuras 9(d) e 9(e).

Os implicantes primos que compõem a função mínima de F_{M_2} são $\{X10, 10\}$ + $\{0X1, 01\}$ + $\{000, 11\}$. O circuito é apresentado na Figura 10, como pode ser observado, o número de entradas nas portas AND e OR contabilizam o custo total da função, $custo_{total} = 11$.

Os circuitos mínimos implementados para f_{10} e f_{11} podem ser visualizados nas Figuras 11(a) e 11(b), respectivamente. O custo mínimo total, ao minimizar as funções

Figura 9 – Representação dos Implicantes Primos de F_{M_2} no Mapa de Karnaugh



(a) f_{10}

(b) f_{11}

(c) f_{10} e f_{11}

(d) f_{10} múltipla

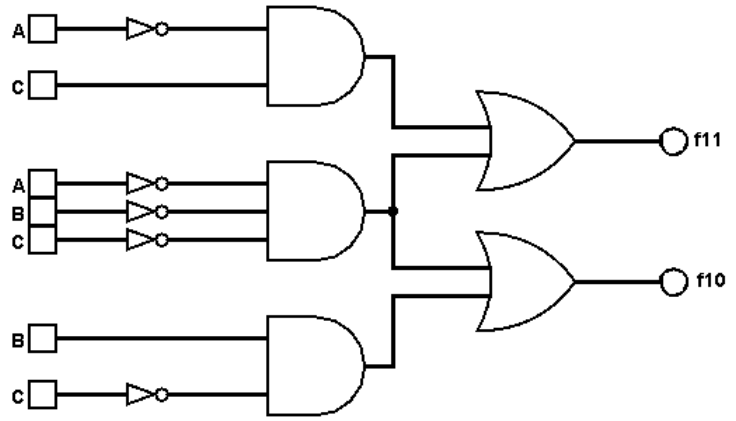
(e) f_{11} múltipla

Fonte: Elaborado pelo Autor.

separadamente, é $custo_{f_{10}} + custo_{f_{11}} = 12$.

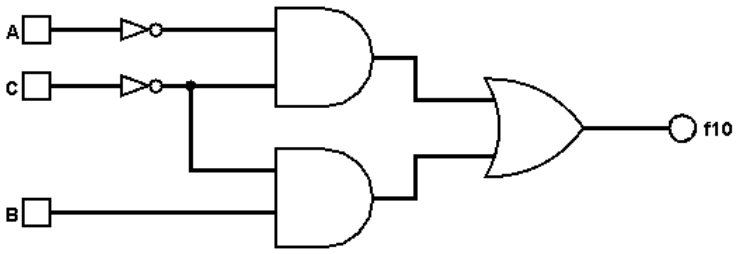
O custo referente a minimização de F_{M_2} é menor se comparado ao custo de f_{10} e f_{11}. Isso pois um implicante primo comum é compartilhado pelas duas funções. Este implicante primo é considerado um implicante primo múltiplo, pois só é gerado a partir da combinação das funções.

Figura 10 – Circuito Lógico que representa a função mínima de F_{M_2}

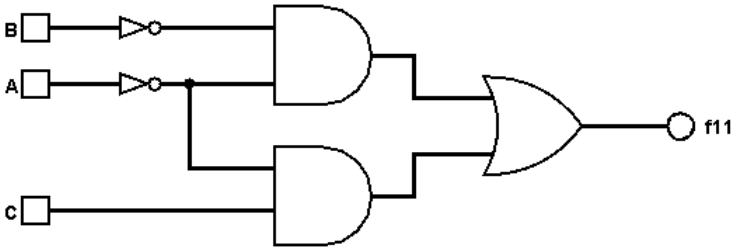


Fonte: Elaborado pelo Autor.

Figura 11 – Circuitos Lógico que representam as funções mínimas de f_{10} e de f_{11}



(a) f_{10}



(b) f_{11}

Fonte: Elaborado pelo Autor.

3.2 COBERTURA DE MINTERMOS DE FUNÇÕES BOOLEANAS COM MÚLTIPLAS SAÍDAS ATRAVÉS DE PROGRAMAÇÃO LINEAR INTEIRA (PLI) 0 E 1

A Cobertura é um problema clássico, que consiste em encontrar um conjunto de soluções que cobrem um conjunto de necessidades ao menor custo possível. Existem muitas aplicações relacionadas ao Problema de Cobertura, como alocação de serviços, arquivos de seleção em um banco de dados, balanceamento de linhas de produção, simplificação de

expressões booleanas, entre outros.

No trabalho o problema de cobertura é utilizado para simplificar expressões booleanas. Utiliza-se para a formulação do problema a programação linear inteira 0 e 1.

Após gerar todos os implicantes primos da função, a próxima fase para obter uma função booleana de custo mínimo é a cobertura dos mintermos. Como os algoritmos propostos executam apenas a primeira etapa do processo de minimização, o problema de cobertura é expresso como um problema de programação linear inteiro 0 e 1.

A partir dos implicantes primos obtidos, defini-se, para cada um deles, uma variável para o problema de cobertura dos mintermos. Inicialmente, deve-se ter uma função objetivo com o critério de custo a ser minimizado, formada pelos implicantes primos e o custo relacionado a cada um. Para o cálculo do custo, considera-se o custo associado a quantidade de entradas das portas lógicas envolvidas.

Em seguida, defini-se as restrições para cada mintermo da função, na qual a soma dos implicantes primos que cobrem o mintermo analisado deve ser maior ou igual a 1. A quantidade de restrições que comporão o problema a ser formulado está relacionada com o número de mintermos da função, isto é, para cada mintermo pertencente a função deve-se ter uma restrição associada (SILVA, 1993).

Para formular o problema de cobertura de mintermos para uma função booleana com múltiplas saídas duas etapas devem ser realizadas: A Cobertura Múltipla e a Cobertura Singular (SILVA, 1993).

Na cobertura múltipla são gerados Implicantes Primos Singulares (IPS) e Implicantes Primos Múltiplos (IPM). IPM são implicantes primos comum a mais de uma função. Após a cobertura múltipla, formula-se para cada função o problema de cobertura singular, para a geração dos implicantes primos singulares. Na etapa de cobertura singular, o conjunto de IPS gerado é a solução mínima encontrada para a função analisada.

De posse dos implicantes primos, a cobertura múltipla deve ser representada da seguinte forma:

- Formula-se a função objetivo para o problema:

  Cada implicante primo pode ser descrito por uma variável;

  Cada implicante primo deve estar associado ao custo relacionado ao número de entrada das portas lógicas (AND e OR);

  A função objetivo é composta pela soma dos implicantes primos associado a seu

custo.

- Para cada função pertencente ao problema, um conjunto de restrições deve ser elaborado;

  Para cada mintermo da função, a soma dos implicantes primos que o cobre deve ser maior ou igual 1.

As restrições que estão em duplicidade podem ser descartadas da formulação do problema.

A partir da solução obtida, analisa-se o resultado e caso não se tenha a solução mínima, deverá ser realizada a cobertura singular. A formulação da cobertura singular é realizada a partir do conjunto de implicantes primos obtidos da cobertura múltipla. Logo:

- Formula-se a função objetivo relacionada a função:

  Cada implicante primo (gerado a partir da cobertura múltipla) associado a função pode ser representado por uma variável;

  A função objetivo deve ser composta pela soma dos implicantes primos (associado a seu custo), que cobrem os mintermos da função analisada.

- Para cada mintermo da função analisada gera-se uma restrição, na qual a soma dos implicantes primos que cobrem o mintermo deve ser maior ou igual a 1.

Para cada função deve-se fazer a cobertura singular. O resultado obtido, conjunto de implicantes primos singulares, representa a solução mínima para a função analisada.

Existem *softwares* que utilizam a programação linear para a formulação do problema de cobertura, tais como: LINDO[1], AIMMS[2], AMPL[3], GAMS[4], VISUAL XPress[5], entre outros. Cada um possui suas particularidades e podem ser aplicados para tipos distintos de problemas.

Neste trabalho utilizou-se o software LINDO, como ferramenta para solucionar os Problemas de Programação linear inteira 0 e 1.

[1]Informações e download em www.Lindo.com
[2]Informações e download em www.aimms.com
[3]Informações e download em www.ampl.com
[4]Informações e download em www.gams.com
[5]Informações em http: www.fico.com/en/wp-content/secure_upload/Xpress-Mosel-User-Guide.pdf

3.2.1 *Software* LINDO

O *software* LINDO (*Linear, INteractive, and Discrete Optimizer*) tem como base o Método Simplex (DANTZIG et al., 1955), que é eficaz para resolver programas lineares.

Para formular um problema para ser resolvido pelo software LINDO, os itens descritos a seguir são necessários:

- Os comandos MAX (para maximizar) e MIN (para minimizar) devem iniciar a linha que se refere à Função Objetivo;
- Logo após a função objetivo, próxima linha do arquivo, deve-se declarar SUBJECT TO (sujeito a). Pode ser utilizado também ST ou S.T.;
- Em seguida, o conjunto de restrições deve ser inserido;
- Finaliza-se com a declaração do comando END.

Deve-se atentar que as variáveis utilizadas na função objetivo, devem possuir no máximo 8 caracteres.

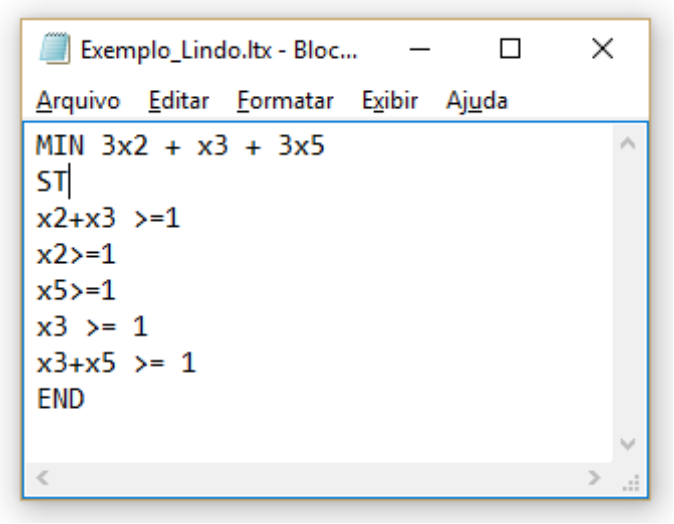
Como o problema a ser resolvido deve possuir apenas resultados inteiros 0 e 1, segundo Silva (SILVA, 1993) pode-se utilizar o Corte Fracional Dual de Gomory quando casos com resultados não inteiros ocorrem.

Uma alternativa para forçar que as variáveis assumam valores inteiros, é inserir no arquivo de entrada do LINDO, logo abaixo do comando END, o comando INT a, em que a refere-se ao número de implicantes primos pertencentes a função objetivo, ou uma linha de INT *varivel*, para cada variável da função objetivo.

O arquivo de entrada para o *software* LINDO pode ser visualizado na Figura 12, nele estão definidas a função objetivo e as restrições relacionadas a uma função booleana.

Na próxima seção, a Primeira Fase do Método de Quine-McCluskey para Funções com Múltiplas Saídas é descrita, assim como o algoritmo implementado e a estrutura de dados utilizada.

Figura 12 – Exemplo de Arquivo de Entrada para o *Software* Lindo



Fonte: Elaborado pelo Autor.

3.3 PRIMEIRA FASE DO MÉTODO DE QUINE-MCCLUSKEY PARA MÚLTIPLAS SAÍDAS - QMM

O Método de Quine-McCluskey (MCCLUSKEY, 1986) é um método de minimização de funções booleanas tabular e iterativo clássico, muito utilizado na literatura como base ou comparativo a outros procedimentos. É composto por duas fases, sendo a primeira a geração de implicantes primos e, a segunda, a cobertura dos mintermos. Suas vantagens em relação ao Mapa de Karnaugh são a não subjetividade, não dependendo de habilidades e de treinamento por parte do usuário para encontrar a expressão mínima. Pode ser programável para ser executado em computador.

No trabalho implementou-se a Primeira Fase do Método de Quine-McCluskey adaptado para Funções com Múltiplas Saídas. Os testes gerados foram comparados aos testes realizados nos métodos propostos.

Uma breve descrição da primeira Primeira Fase do Método de QMM pode ser visualizado a seguir:

1. Verifica-se cada mintermo e *don't care state*, classificando-os em caixas de acordo com a quantidade de dígitos 1 presentes no termo. O Grupo m é formado após a classificação dos termos em suas devidas caixas, m inicia-se em 0 e é incrementado a cada novo grupo formado;

2. No Grupo m, realiza-se a comparação de cada termo de uma caixa i (considere i a quantidade de dígito 1, em que o valor máximo de i é o número de variáveis

(n)), com todos os termos da caixa $i+1$, e assim sucessivamente, até que todos os termos da caixa i tenham sido comparados com todos da caixa $i+1$. Este processo é repetido enquanto $(i+1) <= n$;

Em cada comparação, verifica-se primeiramente as funções que cada termo pertence. Se existir função em comum, os termos podem ser analisados.

Se existir função em comum, e se os termos diferenciarem-se em apenas um *bit*, são combinados, gerando um novo termo, para um novo grupo $(m+1)$.

3. Deve-se marcar os termos combinados, essa marca é feita se:

   Os dois termos são marcados, se as *tags* de função dos termos forem iguais;

   Um dos termos será marcado se a sua *tag* de função estiver contida na do outro termo.

4. Os termos do novo grupo são classificados conforme a quantidade de dígitos 1, assim como feito no passo 1; A comparação é realizada conforme passo 2 e a marcação conforme passo 3.

5. O fim das comparações ocorre quando as comparações realizadas entre os termos de um grupo não gera nenhum implicante, ou quando se tem apenas uma caixa no novo grupo;

6. Os termos não marcados de todos os grupos constituem o conjunto de implicantes primos da função.

3.3.1 Algoritmo primeira fase do método de QMM e fluxograma

Neste trabalho foi implementada apenas a Primeira Fase do Método de Quine-McCluskey para Múltiplas Saídas. O resultado (conjunto dos implicantes primos) é armazenado em um arquivo. O algoritmo referente a primeira fase do método está descrito conforme o Algoritmo 1.

A parada do algoritmo ocorre quando não é gerado um novo Grupo, ou quando um Grupo criado contém somente uma caixa.

Na Figura 13, pode-se observar o Fluxograma que representa os passos do algoritmo desenvolvido para a geração dos implicantes primos de uma função, de acordo com a Primeira Fase do Método de Quine-McCluskey.

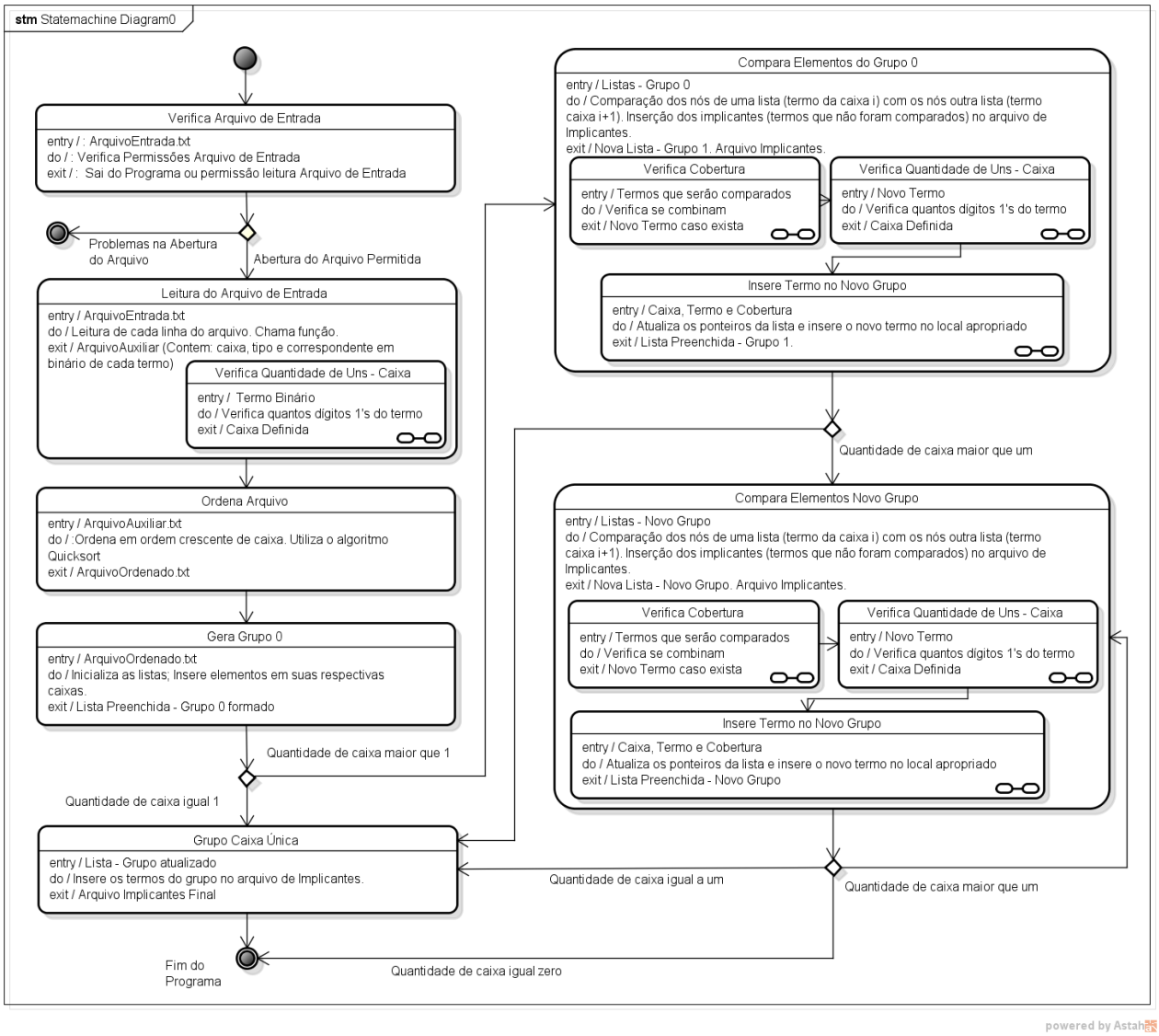
Algoritmo 1: Algoritmo Primeira Fase do Método de Quine-McCluskey adaptado para Funções com Múltiplas Saídas

```
   Entrada: Arquivo contendo os mintermos e don't care de cada função
   Saída: Arquivo contendo os Implicantes Primos Gerados
 1 início
 2     Percorrer Arquivo de Entrada;
 3     Verificar a quantidade de caixas de cada termo;
 4     Inserir cada termo na lista referente ao número de 1's que possui (caixa correspondente) -
         Formação do Grupo0 ;
 5     repita
 6         Verificar número de caixas, nCaixas, do Grupo;
 7         se nCaixas>1 então
 8             Selecionar caixa inicial i;
 9             Selecionar caixa i + 1;
10             Inicializar a variável de controle x;
11             repita
12                 Selecionar o termo na posição x da caixa i do Grupo;
13                 Inicializar a variável de controle y;
14                 repita
15                     Selecionar o termo na posição y da caixa i + 1 do Grupo;
16                     Verificar comparação entre as funções de cada termo e depois dos termos;
17                     se comparação pode ser realizada então
18                         Gerar novo termo;
19                         Incluir novo termo no novo Grupo, em sua respectiva caixa;
20                         Marcar os termos comparados, quando pertinente;
21                     fim
22                     Atualizar y, ir para o próximo termo;
23                 até Existir termo da caixa i + 1 do Grupo;
24                 Atualizar x, ir para o próximo termo;
25                 Inicializar y na primeira posição da caixa i + 1;
26             até Existir termo da caixa i do Grupo;
27             Atualizar i, fazer comparação entre as próximas caixas;
28         fim
29         senão
30             Inserir termos da lista no arquivo de implicantes primos;
31             Grupo recebe valor nulo;
32         fim
33         Verificar no Grupo os termo não marcados e inserir no arquivo de Implicantes Primos;
34         Selecionar o novo Grupo Gerado;
35     até Existir Grupo;
36 fim
```

Para melhor compreensão do fluxograma apresentado, segue uma breve descrição das principais funções desenvolvidas:

- **LeituraArquivoEntrada()**: Lê o arquivo de entrada, onde a primeira linha representa a quantidade de variáveis da função, a segunda linha a quantidade de funções (isto é, a quantidade de saída). As próximas linhas representam os mintermos e *don't care states* seguidos da *tag* referente as funções que pertence. Cada linha (termo e *tag* de função) é lida, e o termo é analisado, para verificar o número de dígitos 1s que cada um possui (para elaboração e posterior classificação

Figura 13 – Fluxograma Referente as Etapas do Algoritmo da Primeira Fase do Método de QMM



Fonte: Elaborado pelo Autor.

de acordo com as caixas);

- **OrdenaArquivo()**: Ordena o arquivo de entrada para facilitar a inserção das informações na lista de lista. Os termos são ordenados de forma crescente de acordo com o número de 1s (caixa correspondente);

- **GeracaoDoGrupo()**: Utilizada para a inserção de um termo na lista de lista; os termos pertencentes a mesma caixa são incluídos na mesma lista;

- **ComparaElementosGrupo()**: Toda a comparação e inserção de um termo em um novo grupo é realizada a partir dessa função. Inicialmente, compara-se as *tags* de função dos termos. Caso exista função em comum, analisa-se os termos e, caso

estes combinem, um novo elemento é gerado. Os termos são marcados de acordo com o tipo de combinação realizada. O novo elemento é inserido no novo Grupo. Este processo é realizado até que o Grupo 0 seja percorrido até a última caixa. Também verifica-se, nesta etapa, quais termos não foram marcados e os mesmos são selecionados e armazenados em um arquivo de implicantes primos;

- **ComparaElementosNovoGrupo()**: Compara os termos do novo grupo. Distingue-se da função anterior, pois os termos contidos no novo grupo diferem dos termos inseridos no *Grupo*0. Isso porque estes possuirão uma lista contendo a sua origem, que é a cobertura. Utiliza-se a função **InsereTermoNovoGrupo()** para agregar o novo elemento ao novo grupo que será criado. O processo é similar a **GeraçãoDoGrupo()** e a **ComparaElementosGrupo()**;

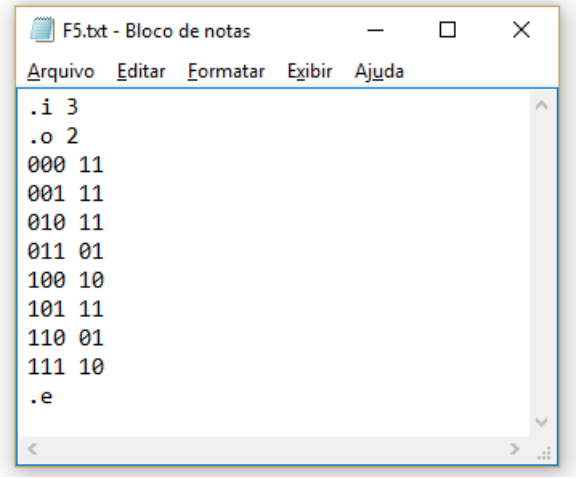
- **GrupoCaixaUnica()**: utilizada para selecionar os elementos de um grupo que possui apenas uma caixa. Estes são incluídos no arquivo de implicantes primos. Isso porque, como não haverá comparações, esses termos não são analisados.

A parada do algoritmo ocorre quando não é gerado um novo Grupo, ou quando o novo Grupo contém somente uma caixa.

3.3.2 Arquivo de entrada e estrutura de dados utilizados

O arquivo de entrada utilizado para a execução do método segue o padrão de arquivo de entrada do software Espresso (BRAYTON, 1984), conforme representado na Figura 14.

Figura 14 – Exemplo de Arquivo de Entrada padrão Espresso



Fonte: Elaborado pelo Autor.

A primeira linha do arquivo, ".i 3", o valor decimal representa a quantidade de variáveis de entrada da função. A segunda linha, ".o 2", a quantidade de funções, isto é, a quantidade de saídas. Até o ".e", que significa fim do arquivo, cada linha representa o termo e as funções que este pertence (o dígito 1 significa que o termo está presente na função, e o 0 que não está presente). Por exemplo, considerando a linha "010 11"do arquivo, o significado destas duas seqüências binárias é: a primeira representação indica o termo em binário (decimal 2), e a segunda, a quais funções este termo pertence, no exemplo, o termo pertence às duas funções.

Na implementação do método, para o armazenamento dos termos da função, comparativos e armazenamento dos implicantes gerados utilizou-se a estrutura de dados Lista. A representação da lista pode ser observada na Figura 15(a), geração do Grupo 0, e na Figura 15(b), que corresponde a geração do Grupo 1 (que representa um Novo Grupo).

A seguir, um exemplo de geração dos implicantes primos utilizando a Primeira Fase do Método de Quine-McCluskey para Funções com Múltiplas Saídas, é apresentado. A solução mínima é obtida através da formulação do problema como um problema de programação linear inteira 0 e 1.

3.3.3 Exemplos de minimização utilizando a primeira fase do método de Quine-McCluskey para funções com múltiplas saídas e PLI 0 e 1

Considere o exemplo a seguir, no qual a expressão 14, $F_{M_3} = \{f_{12}, f_{13}, f_{14}\}$, é composta pelas funções descritas:

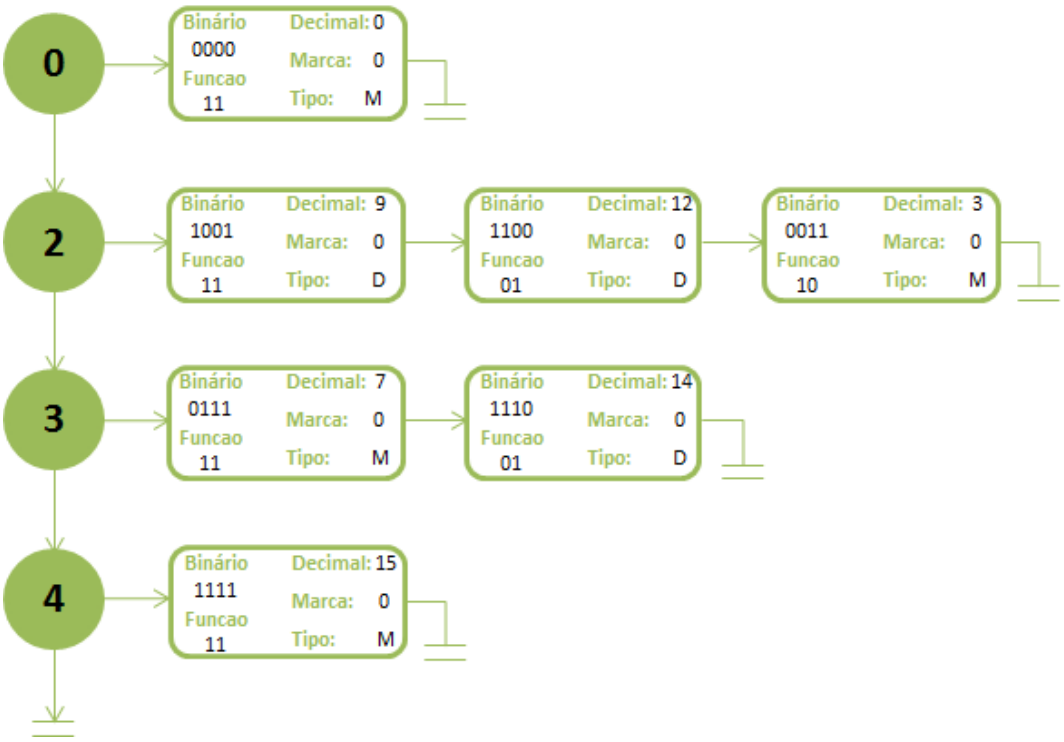
$$\begin{aligned} f_{12}(A,B,C,D) &= \sum m(0,1,2,3,6,7) \\ f_{13}(A,B,C,D) &= \sum m(6,7,8,9,10,14) \\ f_{14}(A,B,C,D) &= \sum m(0,1,2,3,8,9) \end{aligned} \tag{14}$$

O funcionamento da Primeira Fase do Método de QMM pode ser visualizado na Tabela 7. Os termos das funções, dispostas na expressão 14, estão representados no Grupo 0, conforme a Tabela 7(a). O Grupo 1, formado pelos implicantes gerados a partir das combinações do Grupo 0, pode ser visualizado na Tabela 7(b). E o Grupo 3, combinação dos elementos do grupo anterior, disposto na Tabela 7(c).

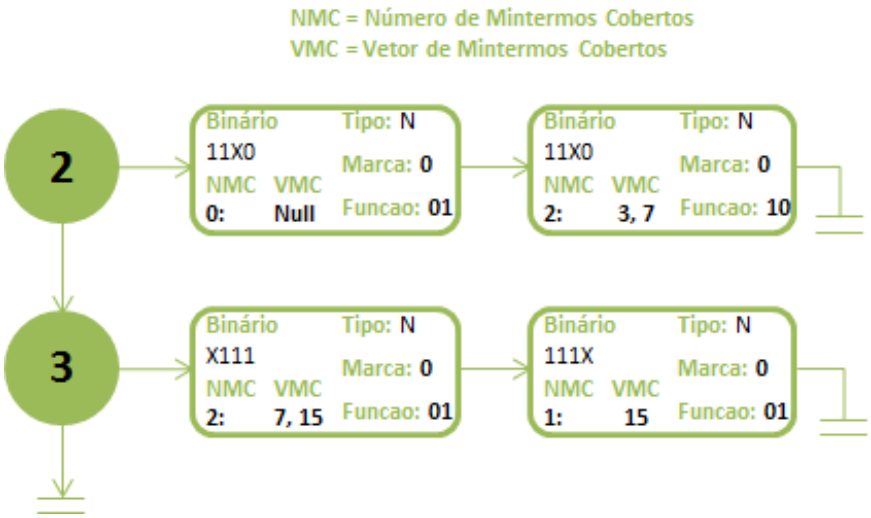
Os implicantes primos gerados, após a aplicação da Primeira Fase do Método, podem ser visualizados na Tabela 8.

A partir dos implicantes primos gerados, deve-se formular o problema de cobertura. No Método de Quine-McCluskey, a segunda fase é destinada para a cobertura dos mintermos, porém, esta não foi implementada no trabalho e a cobertura é formulada

Figura 15 – Exemplo da Estrutura de Dados Lista de Lista utilizada na Implementação do Método de QMM



(a) Grupo 0



(b) Grupo 1

Fonte: Elaborado pelo Autor.

Tabela 7 – Geração de Implicantes Primos de F_{M_3}

(a) Grupo 0

Grupo 0				
Caixa	Binário	Decimal	Função	Marca
0	0000	0	101	*
1	0001	1	101	*
	0010	2	101	*
	1000	8	011	*
2	0011	3	101	*
	0110	6	110	*
	1001	9	011	*
	1010	10	010	*
3	0111	7	110	*
	1110	14	010	*

(b) Grupo 1

Grupo 1				
Caixa	Termo	Função	Cobertura	Marca
0	$000X$	101	$0, 1$	*
	$00X0$	101	$0, 2$	*
	$X000$	001	$0, 8$	*
1	$00X1$	101	$1, 3$	*
	$X001$	001	$1, 9$	*
	$001X$	101	$2, 3$	*
	$0X10$	100	$2, 6$	*
	$100X$	011	$8, 9$	
	$10X0$	010	$8, 10$	
2	$0X11$	100	$3, 7$	*
	$011X$	110	$6, 7$	
	$X110$	010	$6, 14$	
	$1X10$	010	$10, 14$	

(c) Grupo 2

Grupo 2				
Caixa	Termo	Função	Cobertura	Marca
0	$00XX$	101	$0, 1, 2, 3$	
	$X00X$	001	$0, 1, 8, 9$	
	$0X1X$	100	$2, 3, 6, 7$	

Fonte: Elaborado pelo Autor.

Tabela 8 – Implicantes Primos gerados pela Primeira Fase do Método de Quine-McCluskey para Funções com Múltiplas Saídas para F_{M_3}

Implicantes Primos	Funções	Cobertura
$00XX$	101	$0, 1, 2, 3$
$0X1X$	100	$2, 3, 6, 7$
$011X$	110	$6, 7$
$X00X$	001	$0, 1, 8, 9$
$X110$	010	$6, 14$
$100X$	011	$8, 9$
$10X0$	010	$8, 10$
$1X10$	010	$10, 14$

Fonte: Elaborado pelo Autor.

como um problema de programação linear inteira 0 e 1.

As representações, no Mapa de Karnaugh, dos implicantes primos das funções de F_{M_3}, podem ser visualizados nas Figuras 16(a), 16(b) e 16(c).

A combinação das funções de F_{M_3} estão representadas nos Mapas de Karnaugh, e podem ser visualizadas nas Figuras 17(a), 17(b) e 17(c).

A partir dos implicantes primos gerados, o problema de cobertura pode ser formulado, de acordo com a seção 3.2, e a solução mínima é obtida.

Considere $x_1 = 00XX$, $x_2 = 0X1X$, $x_3 = 011X$, $x_4 = X00X$, $x_5 = X110$, $x_6 = 100X$, $x_7 = 10X0$ e $x_8 = 1X10$, para a representação de cada implicante primo da função.

A formulação da função objetivo é composta pela soma dos implicantes primos, associados ao custo.

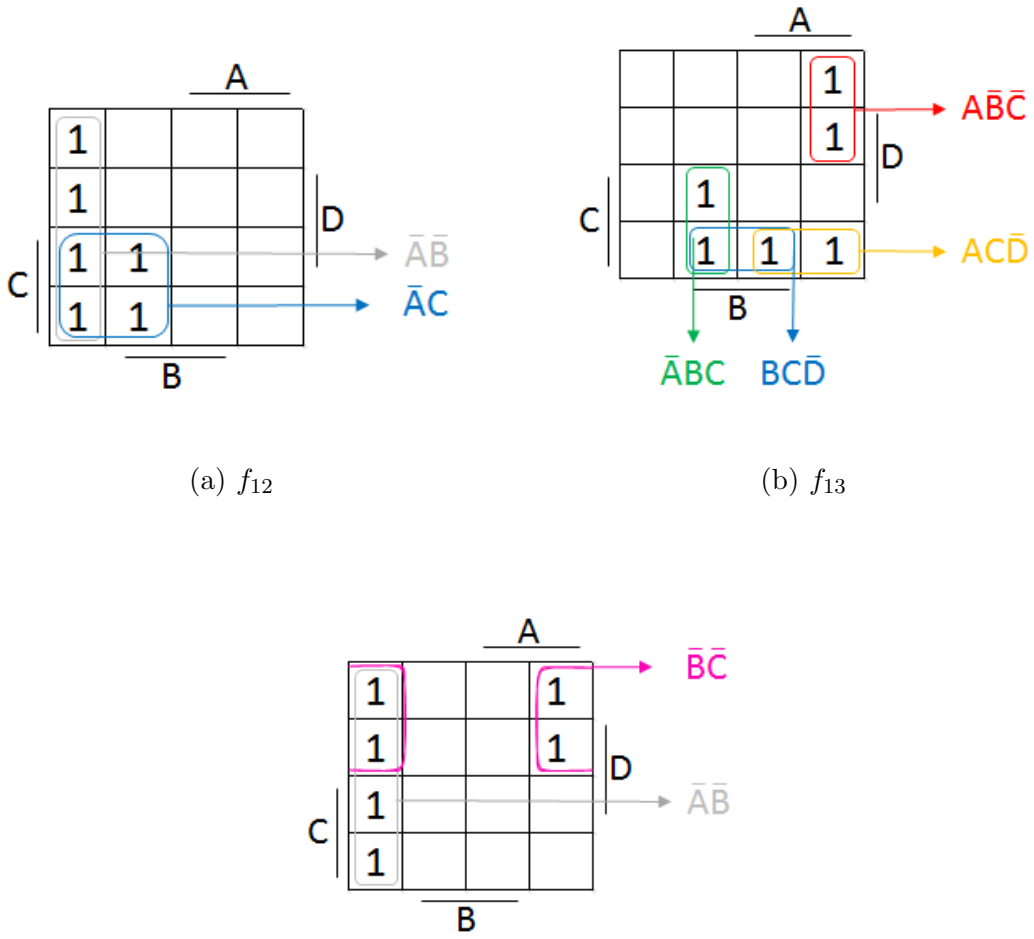
Conforme já descrito, formula-se a cobertura múltipla e a cobertura singular. Inicialmente obtem-se os implicantes primos múltiplos com a cobertura múltipla, como descrita na Tabela 9.

Elimina-se as restrições redundantes, e a formulação pode ser descrita como apresentada na Tabela 10.

Definidas as restrições para cada função, pode-se gerar os implicantes primos a partir da cobertura múltipla. O conjunto de implicantes primos obtidos é: x_1, x_3, x_6 e x_8. Os implicantes primos x_2, x_4, x_5 e x_7 não são considerados para a próxima fase (cobertura singular).

Formula-se a cobertura singular, isto é, a cobertura para cada função individualmente, como pode ser visualizada na Tabela 11.

Figura 16 – Representação dos Implicantes Primos das Funções de F_{M_3} no Mapa de Karnaugh



(a) f_{12}

(b) f_{13}

(c) f_{14}

Fonte: Elaborado pelo Autor.

A cobertura singular para cada função de F_{M_3}:

$f_{12} = 00XX + 011X$;

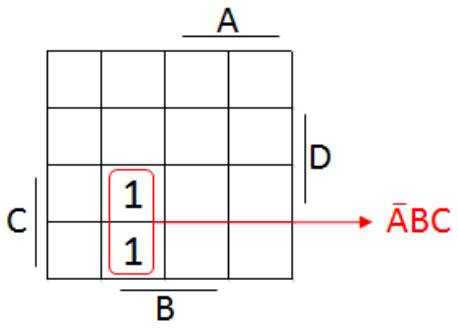
$f_{13} = 011X + 100X + 1X10$;

$f_{14} = 00XX + 100X$.

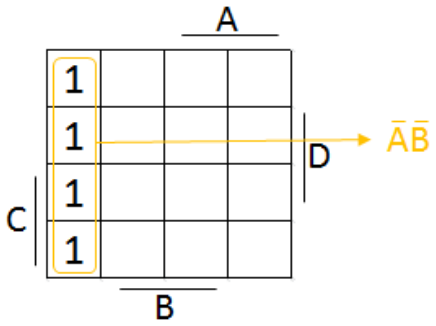
O $custo_{F_{M_3}}$ é composto pela soma do número de entradas nas portas AND referentes aos implicantes primos gerados (as entradas nas portas AND dos implicantes primos compartilhados, são contabilizadas apenas uma vez), e também pelo número de entradas nas portas OR. São 11 entradas em portas AND e são 7 entradas nas portas OR. O custo total para implementar F_{M_3} é $11 + 7 = 18$.

A solução mínima para cada uma das funções de F_{M_3} pode ser visualizado nos Mapas

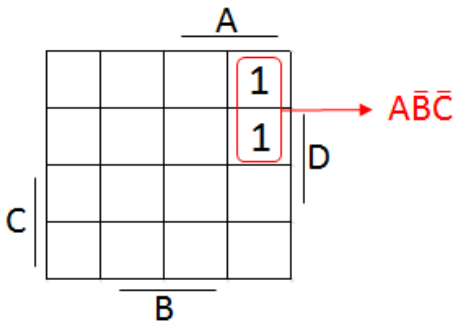
Figura 17 – Representação dos Implicantes Primos das combinações das funções de F_{M_3} no Mapa de Karnaugh



(a) f_{12}, f_{13}



(b) f_{12}, f_{14}



(c) f_{13}, f_{14}

Fonte: Elaborado pelo Autor.

de Karnaugh, que estão representado na Figura 18.

Se considerarmos a minimização das funções de forma simples, minimizando f_{12}, f_{13} e f_{14}, separadamente, a cobertura é obtida como seguinte:
$f_{12} = 00XX + 011X$,custo 7;
$f_{13} = 011X + 100X + 1X10$, custo 12;
$f_{14} = 00XX + 100X$, custo 7.

O custo total é obtido através da soma dos custos de cada uma das funções, logo, o $custo_{total} = custo_{f_{12}} + custo_{f_{13}} + custo_{f_{14}} = 26$.

Foi apresentado neste Capítulo, uma breve introdução a funções booleanas com

Tabela 9 – Cobertura Múltipla de F_{M3}

$MIN\ 3x_1 + 3x_2 + 4x_3 + 3x_4 + 4x_5 + 4x_6 + 4x_7 + 4x_8$	
f_{12}	$x_1 \geq 1$
	$x_1 \geq 1$
	$x_1 + x_2 \geq 1$
	$x_1 + x_2 \geq 1$
	$x_2 + x_3 \geq 1$
	$x_2 + x_3 \geq 1$
f_{13}	$x_3 + x_5 \geq 1$
	$x_3 \geq 1$
	$x_6 + x_7 \geq 1$
	$x_6 \geq 1$
	$x_7 + x_8 \geq 1$
	$x_5 + x_8 \geq 1$
f_{14}	$x_1 + x_4 \geq 1$
	$x_1 + x_4 \geq 1$
	$x_1 \geq 1$
	$x_1 \geq 1$
	$x_4 + x_6 \geq 1$
	$x_4 + x_6 \geq 1$
$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in \{0, 1\}$	

Fonte: Elaborado pelo Autor.

Tabela 10 – Cobertura Múltipla de F_{M_3} sem linhas em duplicidade

$MIN\ 3x_1 + 3x_2 + 4x_3 + 3x_4 + 4x_5 + 4x_6 + 4x_7 + 4x_8$
$x_1 \geq 1$
$x_1 + x_2 \geq 1$
$x_2 + x_3 \geq 1$
$x_3 + x_5 \geq 1$
$x_3 \geq 1$
$x_6 + x_7 \geq 1$
$x_6 \geq 1$
$x_7 + x_8 \geq 1$
$x_5 + x_8 \geq 1$
$x_1 + x_4 \geq 1$
$x_4 + x_6 \geq 1$
$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in \{0, 1\}$

Fonte: Elaborado pelo Autor.

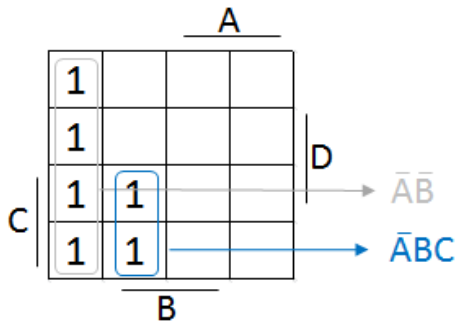
múltiplas saídas, bem como a importância dessas na construção de um circuito lógico. Também foi mostrado como representar as funções com múltiplas saídas no Mapa de

Tabela 11 – Cobertura Singular para as funções de F_{M_3}

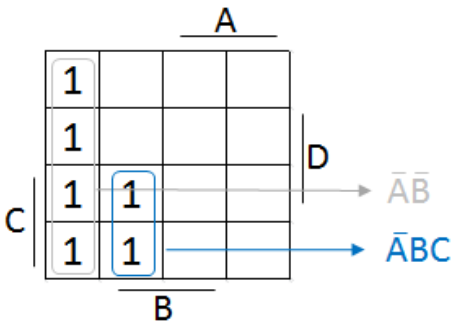
Cobertura Singular f_{12}	Cobertura Singular f_{13}	Cobertura Singular f_{14}
$MIN\ 3x_1 + 4x_3$	$MIN\ 4x_3 + 4x_6 + 4x_8$	$MIN\ 3x_1 + 4x_6$
$x_1 + x_3 \geq 1$	$x_3 \geq 1$	$x_1 + x_6 \geq 1$
$x_3 \geq 1$	$x_6 \geq 1$	$x_6 \geq 1$
$x_1, x_3\ \epsilon\ \{0, 1\}$	$x_8 \geq 1$	$x_1, x_6\ \epsilon\ \{0, 1\}$
	$x_3, x_6, x_8\ \epsilon\ \{0, 1\}$	

Fonte: Elaborado pelo Autor.

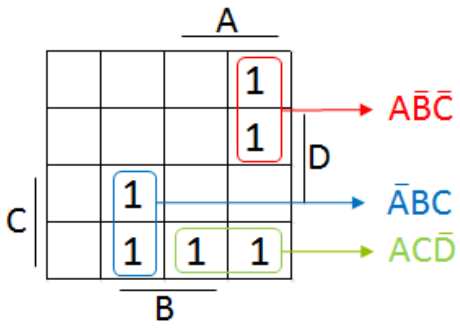
Figura 18 – Representação dos Implicantes Primos, Função Mínima de cada uma das funções de F_{M_3} no Mapa de Karnaugh



(a) Função Mínima de f_{12}



(b) Função Mínima de f_{13}



(c) Função Mínima de f_{14}

Fonte: Elaborado pelo Autor.

Karnaugh. Para a cobertura dos mintermos de uma função o problema de cobertura foi apresentado como PLI 0 e1. A Primeira Fase do Método de Quine-McCluskey adaptado

para Múltiplas Saídas, foi descrita e exemplificada.

No Capítulo 4 são apresentados os dois algoritmos propostos para geração de implicantes primos para funções com múltiplas saídas.

4 MÉTODOS PROPOSTOS PARA GERAÇÃO DE IMPLICANTES PRIMOS PARA FUNÇÕES BOOLEANAS COM MÚLTIPLAS SAÍDAS

Neste capítulo são apresentados os algoritmos propostos GPMultiplo e o MultiGeraPlex. Os métodos geram os implicantes primos da função; no primeiro, todos os implicantes primos são gerados, já no MultiGeraPlex, nem todos. As propostas implementadas representam os mintermos e *don't care states* em árvore ternária. Cada caminho na árvore é a representação de um termo da função, e a lista de função a qual o termo pertence é armazenada no nó folha da árvore. Após a geração dos implicantes primos, para a cobertura utiliza-se PLI 0 e 1 para a obtenção da função de custo mínimo.

4.1 CARACTERÍSTICAS DOS ALGORITMOS PROPOSTOS

Nesta seção alguns fundamentos e características comuns aos algoritmos desenvolvidos são apresentadas. Como as propostas utilizam o consenso iterativo para a geração dos implicantes primos e a estrutura de dados árvore ternária para o armazenamento e detalhes da árvore, estes dois itens são descritos.

4.1.1 Estrutura de árvore ternária utilizada na implementação dos Métodos Propostos

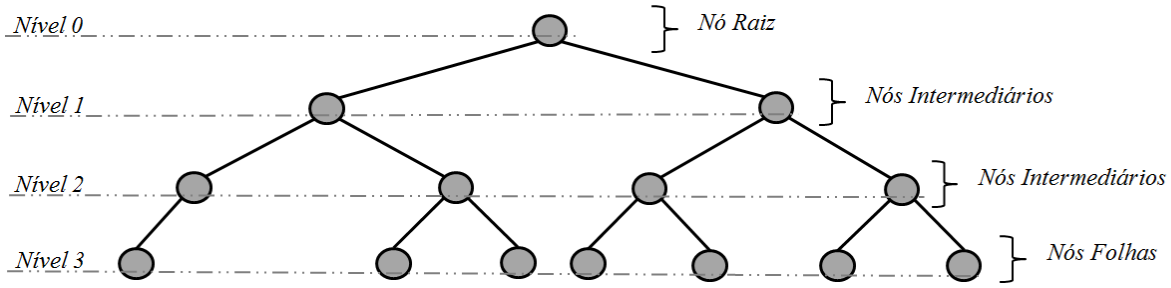
Neste trabalho, utiliza-se uma estrutura baseada em uma árvore de decisão, que consiste de uma hierarquia de nós internos e externos que são conectados por ramos.

Considera-se nó ou vértice de uma árvore o conjunto de elementos que armazena informações e é ligado através de ramos ou arestas a outros nós. O nó é classificado em: nó raiz, nó intermediário ou interno e nó folha ou externo, (conforme mostrado na Figura 19):

- nó raiz: nó do topo da árvore, não possui nó pai, e pertence ao nível 0;

- nó intermediário ou interno: vértices que estão no caminho entre o nó raiz e nós folhas;
- nó folha ou externo: não possui filhos, é a extremidade inferior da árvore. Pertence ao último nível.

Figura 19 – Estrutura de Dados Árvore



Fonte: Elaborado pelo Autor.

O nó interno, também conhecido como nó decisório ou nó intermediário, é a unidade de tomada de decisão que avalia através de teste lógico qual será o próximo nó descendente ou filho.

O procedimento de uma árvore de decisão pode ser descrito como: apresenta-se um conjunto de dados ao nó raiz da árvore, que dependendo do resultado do teste lógico usado pelo nó, a árvore ramifica-se para um dos nós filhos. Este procedimento é repetido até que se alcance um nó folha.

Neste trabalho, a árvore de decisão é uma árvore ternária, em que cada nó intermediário divide-se exatamente em três nós descendentes, o nó esquerdo, o nó do meio e o nó direito.

Seja uma árvore enraizada T, formada por um conjunto finito de nós ou vértices (elementos que armazenam informações), definidos como v. A árvore é uma estrutura que é formada por quatro conjuntos disjuntos de nós: um nó raiz, r, uma sub-árvore da esquerda, uma sub-árvore do meio e uma sub-árvore da direita.

De acordo com Cormen (CORMEN et al., 2009), o comprimento do caminho desde a raiz r até um nó v é a profundidade de v em T. A profundidade, conhecida também como nível, inicia-se em 0 no nó raiz, e é incrementada a cada nó, até chegar no vértice folha.

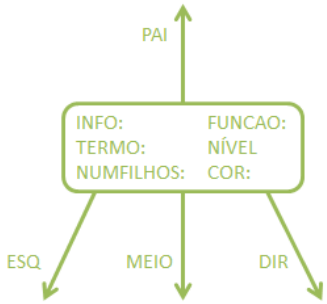
Nós filhos do mesmo nó pai, são considerados irmãos, e podem ser comparados. Por isso, a análise se dá a partir de nós contidos em cada nível da árvore.

Adota-se no trabalho como ramo o conjunto de termos e arestas que compõe o caminho

do nó analisado (nível atual) até uma das folhas.

Na implementação das propostas, cada nó da árvore é definido como descrito na Figura 20.

Figura 20 – Estrutura do Nó Implementado



Fonte: Elaborado pelo Autor.

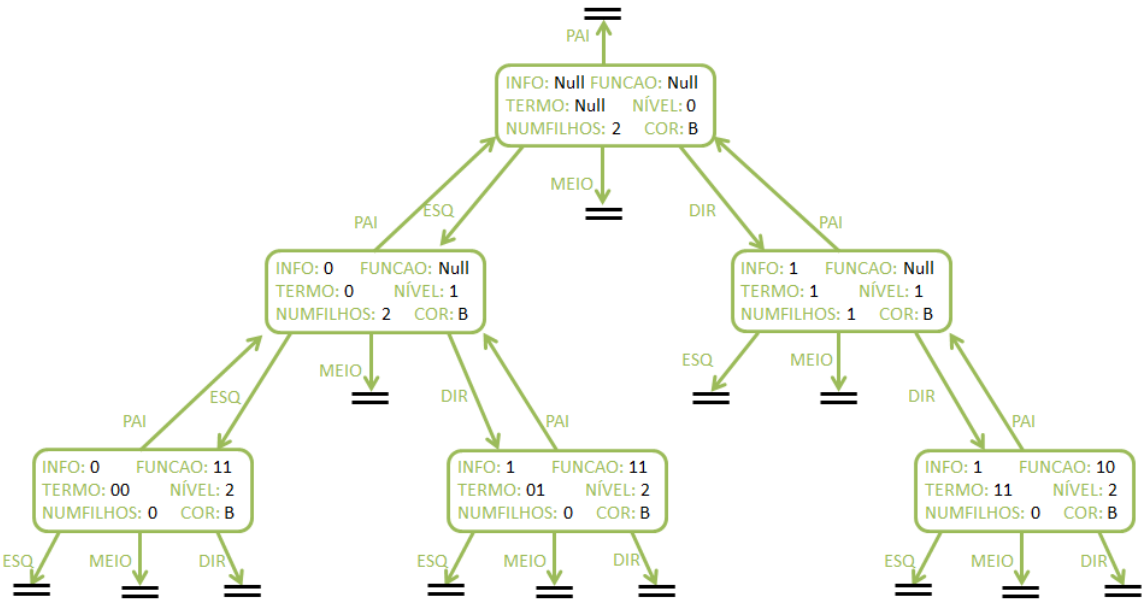
O nó implementado é composto por dez variáveis:

- $info$: representa o *bit* de uma determinada posição do termo, definido por 0, 1 ou X. Utilizado como condição para inserção de novos nós;
- *termo*: conjunto de caracteres que representa os *bits* correspondentes ao termo até a posição em questão (nível equivale a posição do caracter no termo). Importante para geração e inserção de um novo termo na árvore;
- $numFilhos$: contém o número de filhos que o nó possui. Quando valor 0, representa o nó folha;
- nível: número inteiro que sinaliza qual o nível o nó pertence. Utilizado para comparação entre os nós e também para indicar o nó raiz (nível 0);
- *cor*: caracter utilizado para sinalizar o percurso na árvore, pode ser B, P, C ou X. B: significa que o nó não foi visitado e está livre para ser percorrido; P: nó já foi visitado e os filhos comparados; C: nó já foi visitado, porém ainda possui filhos que não foram comparados; X: nó não pode ser lido, visitado;
- $funcao$: conjunto de caracteres que representa um vetor de caracteres 0,1. A quantidade de caracteres está relacionada a quantidade de funções. Defini-se 1 para presente, e 0 para não presente. Ex. uma *tag* 110, (três saídas ou funções) significa que o termo está presente na primeira e segunda função e não está presente na terceira;

- *esq*: ponteiro para estrutura da árvore, representa a sub-árvore da esquerda, percorrida ou criada quando o caracter do termo for 0;
- *dir*: ponteiro para estrutura da árvore, representa a sub-árvore da direita, percorrida ou criada quando o caracter do termo for 1;
- *meio*: ponteiro para estrutura da árvore, representa a sub-árvore do meio, percorrida ou criada quando o caracter do termo for X;
- *pai*: ponteiro para estrutura da árvore, representa o nó pai, antecessor do nó analisado, utilizado para fazer o percurso de subida na árvore e definir os nós que poderão ser comparados (filhos do mesmo nó pai).

A estrutura de dados implementada para representar os termos da função pode ser visualizada na Figura 21.

Figura 21 – Estrutura de Dados Árvore Utilizada na Representação dos Termos de uma Função

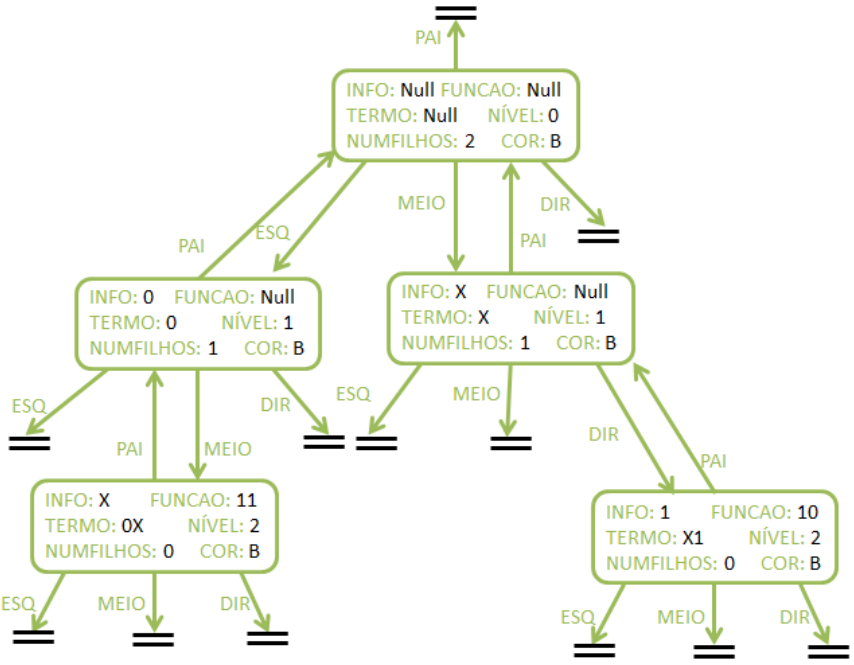


Fonte: Elaborado pelo Autor.

Inicialmente a sub-árvore do meio não é utilizada pois os termos são binários. Após as comparações e geração de um novo termo, a estrutura pode ser retratada como na Figura 22.

A seguir, a definição de consenso iterativo e exemplos de sua aplicação são apresentados. Isto pois, ao comparar dois ramos de uma árvore, aplica-se o consenso iterativo entre eles, para a possível geração de um novo termo (implicante).

Figura 22 – Estrutura de Dados Árvore com a Inserção de um Novo Termo



Fonte: Elaborado pelo Autor.

4.1.2 Consenso iterativo

O consenso ocorre entre dois termos produtos, quando existe uma única variável presente nos termos, sendo obrigatoriamente complementada em um deles. O resultado obtido, é um termo contendo as demais variáveis dos termos produtos combinados.

A operação de consenso entre dois vértices (que devem ser irmãos) é aplicada iniciando-se nos nós folhas da árvore. Alguns exemplos da operação de consenso são apresentados na Tabela 12.

Tabela 12 – Exemplos da operação de consenso

$(ABCD, ABC\overline{D}) \rightarrow ABC$
$(A\overline{B}CD, A\overline{B}\,\overline{C}) \rightarrow A\overline{B}D$
$(\overline{B}\,\overline{C}, AB\overline{C}) \rightarrow A\overline{C}$
$(\overline{A}CD, ABC) \rightarrow BCD$

Fonte: Elaborado pelo Autor baseado em (MENDELSON, 1970).

Utiliza-se o consenso iterativo entre dois ramos da árvore, filhos de um mesmo nó. A partir de uma relação de consenso, como $(aB, \overline{a}C) \rightarrow BC$, a variável a é eliminada a partir da igualdade 1, em que $a + \overline{a} = 1$.

Considerando a operação de consenso na estrutura da árvore, quando este é aplicado,

ao eliminar a variável que está complementada em um termo e não complementada no outro, um novo caminho é inserido na árvore. Utiliza-se o caracter X, para representar o nó filho, pertencente ao novo ramo gerado.

Cada variável pertencente a função está representada em um nível da árvore, sendo o *bit* mais significativo inserido mais no topo da árvore (nó filho da raiz), e o menos significativo na folha. Para uma função $f(A, B, C)$, tem-se a representação da variável A como o *bit* mais significativo e do C como o menos, sendo este representado no nó folha.

Nas próximas seções os algoritmos propostos são apresentados e funções são dadas como exemplos, para a geração dos implicantes primos utilizando os dois métodos passo a passo.

4.2 GPMULTIPLO - GPM

O GPMultiplo foi desenvolvido para geração de todos os implicantes primos de funções booleanas com múltiplas saídas. O algoritmo foi baseado na ideia do método GeraPlex Modificado. Além da representação dos mintermos e *don't care state* na árvore, utiliza-se uma informação relacionada às funções que o mintermo ou o *don't care state* pertencem. Essa informação é armazenada no nó folha de cada ramo da árvore. No método, antes da verificação do consenso entre os ramos, as funções são analisadas.

No método GPMultiplo a operação de consenso é aplicada entre os ramos da árvore e baseia-se nas situações apresentadas em (SILVA, 1993), que são: Fusão, Deslocamento e Expansão. Nesta proposição, antes da verificação do consenso entre os ramos da árvore, são analisadas as funções às quais eles pertencem.

Denomina-se F o conjunto de funções a serem minimizadas, e $\{f_1, f_2, \ldots, f_n\}$, as funções dadas, em que n representa o número máximo de funções. Seja T o conjunto de termos a serem analisados pertencentes a cada função, este pode ser composto por mintermos, *don't care states* ou a combinação destes, e é definido como t. Seja R a representação utilizada para definir o termo t e as funções que este pertence F_t, $R = \{t, F_t\}$.

Neste trabalho, o conjunto F é representado por uma sequência binária, de tamanho n (número total de funções a serem analisadas), em que 1 sinaliza a presença do termo na função, e 0 a ausência deste.

Como já mencionado, neste trabalho adota-se ramo como o caminho do nó analisado (nível atual) até a folha. Ao gerar um novo ramo, a inserção deste na árvore é realizada

na sub-árvore do meio do nó analisado, e esta é ramificada até atingir o último nível.

Dado o conjunto de funções $F = \{f_1, f_2, f_3, f_4\}$, seja t_1 o mintermo 9, que pertence a função f_1 e f_3, o conjunto de funções do termo é dado como $F_{t_1} = \{1010\}$. Considere R a representação da t-upla $\{t, F_t\}$, logo, para o termo t_1, tem-se: $R_1 = \{t_1, F_{t_1}\}$.

Aplica-se o consenso entre dois termos, nós folhas da árvore, denotados por R_1 e R_2, e o ramo gerado é representado por $R_g = \{t_g, F_{t_g}\}$.

Considere $R_1 = \{t_1, F_{t_1}\}$ e $R_2 = \{t_2, F_{t_2}\}$ a representação de dois mintermos e o conjunto de funções que eles pertencem, e $R_g = \{t_g, F_{t_g}\}$ o ramo gerado a partir da operação do consenso iterativo entre t_1 e t_2. As situações que podem ocorrer com a aplicação do método GeraPlexMultiplo são: Fusão, Fusão Parcial, Pseudo Fusão, Deslocamento, Pseudo Deslocamento e Expansão.

- **Fusão**: $F_{t_1} = F_{t_2}$ e, ao aplicar a operação de consenso entre (t_1, t_2), o termo gerado, t_g, cobre os de origem. Os termos analisados são eliminados, reduzindo, consequentemente, o número total de caminhos na árvore. $R_g = \{t_g, F_{t_g}\}$, $F_{t_g} = F_{t_1} = F_{t_2}$;

- **Fusão Parcial**: $F_{t_1} \subset F_{t_2}$, ou seja, F_{t_1} é um subconjunto próprio de F_{t_2}. O termo gerado, t_g, cobre t_1 e t_2, porém, apenas t_1 é eliminado;

- **Pseudo Fusão**:$F_{t_1} \not\subset F_{t_2}$ $F_{t_1} \cap F_{t_2} \neq \emptyset$, ou seja, t_1 e t_2 estão presentes em pelo menos uma mesma função, e existe pelo menos uma função distinta em cada um dos F_t. O consenso entre t_1 e t_2, resulta em t_g, que cobre t_1 e t_2, porém, eles não são eliminados. O conjunto de função de t_g é formado por F_{t_1} e F_{t_2};

- **Deslocamento**: $F_{t_1} = F_{t_2}$ ou $F_{t_1} \subset F_{t_2}$, consenso é aplicado entre t_1 e t_2; obtem-se t_g, que cobre apenas t_1. Este é eliminado e o conjunto de funções de t_g, $F_{t_g} = F_{t_1}$;

- **Pseudo Deslocamento**: $F_{t_1} \subset F_{t_2}$, consenso aplicado entre t_1 e t_2, gera t_g, que cobre apenas o t_2. Para esse caso, não há eliminação;

- **Expansão**: será aplicada quando existir função em comum entre F_{t_1} e F_{t_2}; ao aplicar o consenso, o termo gerado t_g, não cobre t_1 e nem t_2. Deve ser feita a análise de F_{t_1} e F_{t_2} para a composição de F_{t_g}.

4.2.1 Algoritmo GPMultiplo e fluxograma

O GPMultiplo foi desenvolvido conforme os passos do Algoritmo 2. O arquivo de entrada foi padronizado no trabalho e segue o modelo de entrada do *software* Espresso, já apresentado na Figura 14.

Algoritmo 2: Parte do Algoritmo GPMultiplo

```
   Entrada: Arquivo contendo os mintermos e don't care da função, sub-árvore Esquerda otimizada
 1 início
 2     Percorrer Arquivo de Entrada;
 3     Inserir na árvore cada mintermo e don't care;
 4     se Sub-árvore esquerda não é nula então
 5         para i = nivel − 1 até i = 1 faça
 6             repita
 7                 Encontrar o nó pai;
 8                 Verificar as funções que cada termo pertence;
 9                 se Existir função em comum então
10                     Aplicar o consenso entre os ramos;
11                     se (Consenso pode ser aplicado) então
12                         Gerar t_g;
13                         Inserir t_g na árvore;
14                         se Funções Iguais então
15                             Atualizar F_{t_g}, se necessário;
16                             se Fusão então
17                                 Eliminar t_1 e t_2;
18                                 Atualizar o número de filhos do nó pai;
19                             senão se Deslocamento então
20                                 Eliminar t_i coberto por t_g;
21                             fim
22                             senão se Expansão então
23                                 Atualizar o número de filhos do nó pai;
24                             fim
25                         senão se F_{t_1} ⊂ F_{t_2} então
26                             Atualizar F_{t_g}, se necessário;
27                             se Fusão Parcial ou Deslocamento então
28                                 Eliminar t_1;
29                             senão se Pseudo Deslocamento ou Expansão então
30                                 Atualizar o número de filhos do nó pai;
31                             fim
32                         fim
33                         senão se F_{t_1} ⊄ F_{t_2} e F_{t_1} ∩ F_{t_2} ≠ ∅ então
34                             Atualizar o número filhos do nó pai;
35                         fim
36                     fim
37                 fim
38                 Atualizar ponteiros de comparação;
39             até (Existir ramo a ser comparado no nível);
40             Atualizar o nível nível da árvore;
41         fim
42     fim
43 fim
```

Considere t_1 e t_2 os termos da árvore que estão sendo comparados e t_g o que foi gerado

pela operação de consenso entre t_1 e t_2. F_{t_1} e F_{t_2} o conjunto das funções que os termos t_1 e t_2 pertencem, respectivamente, e F_{t_g} o conjunto de funções de t_g.

O algoritmo representa parte do código implementado; a sub-árvore direita também deve ser analisada. As sub-árvores esquerda e direita devem ser otimizadas (eliminação dos termos em duplicidade) e, a partir desse ponto, a comparação no nível 0 se inicia, combinando os ramos da sub-árvore esquerda com os da sub-árvore direita.

Após a finalização das comparações, a varredura para eliminar as duplicidades deve ser realizada no nível 0, sub-árvore do meio. O resultado obtido é o conjunto de implicantes primos da função.

Na Figura 23, pode-se visualizar o fluxograma que representa os passos do algoritmo do GPMultiplo. Na Figura 24 a função Consenso por Nível é detalhada para melhor entendimento do algoritmo.

4.2.2 Exemplo de minimização utilizando GPMultiplo e PLI 0 e 1

Considere a expressão 15, que descreve o conjunto F_{M_4}, no qual as funções pertencentes são:

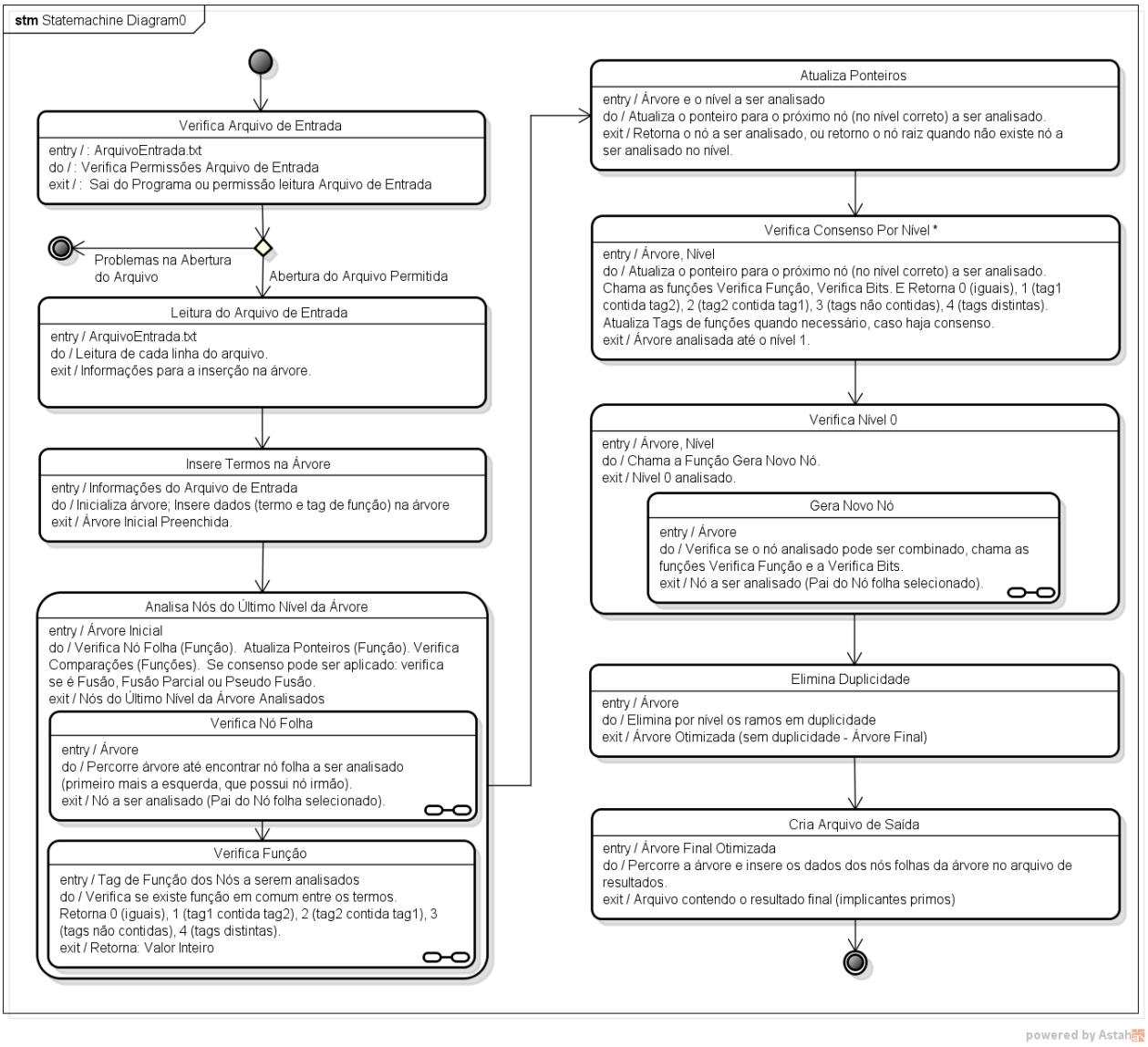
$$\begin{aligned} f_{15} &= \sum m(4,5,7) \\ f_{16} &= \sum m(1,4,5) \\ f_{17} &= \sum m(0,2,4) \end{aligned} \tag{15}$$

Na Figura 25 podem ser visualizados os mintermos representados na estrutura de dados árvore. Em cada nó folha estão inseridos o termo t e o conjunto de funções que pertence F_t.

Os passos da aplicação do GPMultiplo podem ser visualizados nas árvores das Figuras 26, 27, 28, nas quais cada situação de operação de consenso é descrita e os resultados obtidos apresentados.

Considere, $t_1 = \{100\}$, $F_{t_1} = \{111\}$, $R_1 = \{100, 111\}$ e $t_2 = \{101\}$, $F_{t_2} = \{110\}$, $R_2 = \{101, 110\}$, a partir da análise dos nós folhas que representam estes termos, e observando que existem funções em comum entre eles, a operação de consenso pode ser aplicada no nó pertencente ao nível 2. Uma **Fusão parcial** ocorre, já que $F_{t_2} \subset F_{t_1}$, e apenas t_2 é eliminado. O termo gerado $t_g = 10X$ é inserido na árvore, isto é, um novo nó folha é criado, como mostrado na Figura 26(a).

Figura 23 – Fluxograma que Representa os Passos do Algoritmo GPMultiplo



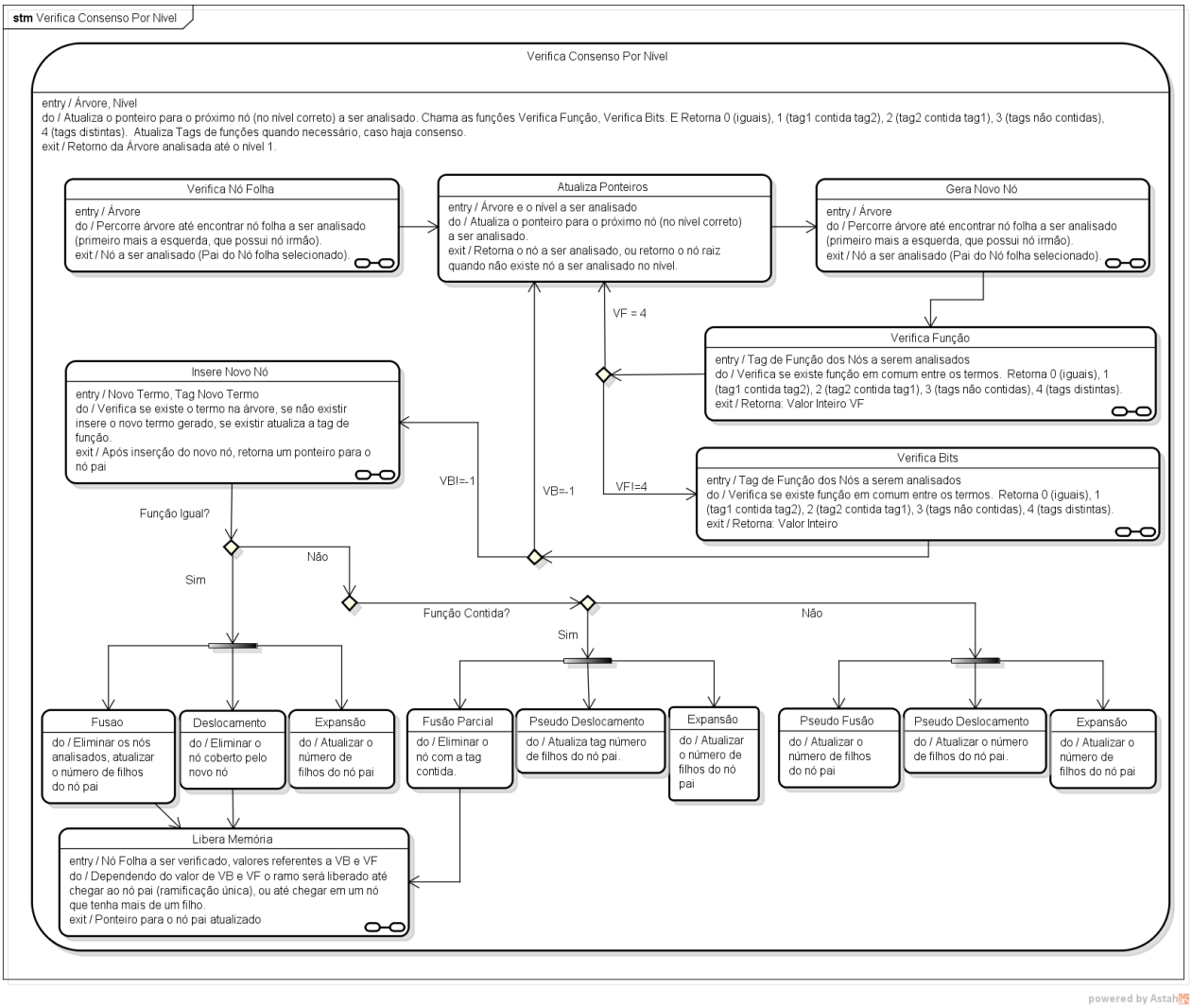
Fonte: Elaborado pelo Autor.

O resultado após o consenso entre os termos está representado na Figura 26(b). A representação do novo nó folha criado é $R_g = \{10X, 110\}$. Ainda analisando o mesmo nível da árvore, percebe-se que não há mais nós para serem analisados, por isso, o processo se inicia no nível 1.

Dois nós pais podem ser verificados nesse nível, considere inicialmente o nó mais a esquerda da árvore (sub-árvore esquerda). No qual os nós filhos são $R_1 = \{0\mathbf{00}, 001\}$ e $R_2 = \{0\mathbf{10}, 001\}$. Uma **Fusão** é obtida, logo, t_1 e t_2 são eliminados e o termo resultante t_g é composto por $R_g = \{0\mathbf{X0}, 001\}$.

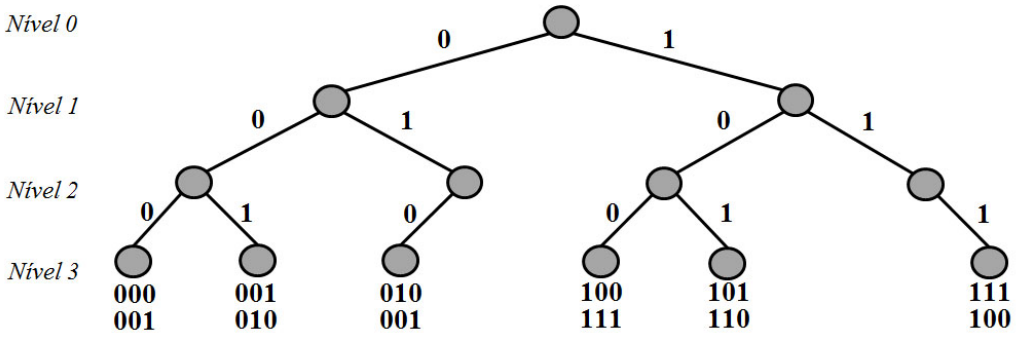
Ainda no nível 1, o nó da sub-árvore direita é analisado. Considere $R_1 = \{1\mathbf{0X}, 110\}$ e $R_2 = \{1\mathbf{11}, 100\}$, o consenso aplicado gera um **Deslocamento**, o que possibilita a

Figura 24 – Fluxograma que Representa os Passos do Algoritmo GPMultiplo



Fonte: Elaborado pelo Autor.

Figura 25 – Representação dos termos das funções de F_{M_4} em árvore

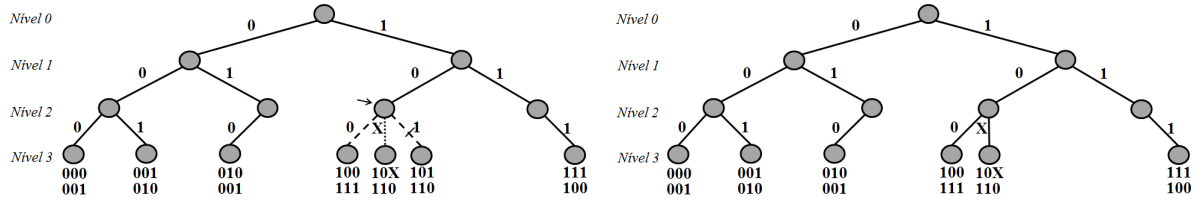


Fonte: Elaborado pelo Autor.

inclusão de $R_g = \{1\mathbf{X1}, 100\}$ e a eliminação de t_2. As situações ocorridas no nível 1, supracitadas, podem ser visualizadas na Figura 27(a), e o resultado na Figura 27(b).

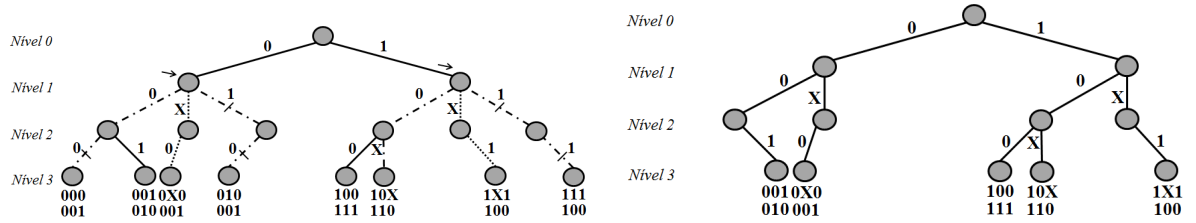
Analisando o nível 0, seja $R_1 = \{\mathbf{001}, 010\}$ e $R_2 = \{\mathbf{10X}, 110\}$. O resultado é

Figura 26 – Aplicação do Método GPMultiplo no nível 2: F_{M_4}



Fonte: Elaborado pelo Autor

Figura 27 – Aplicação do Método GPMultiplo no nível 1: F_{M_4}



Fonte: Elaborado pelo Autor

um **Deslocamento**, t_1 é eliminado, e o novo caminho inserido na árvore composto por $R_g = \{\mathbf{X01}, 010\}$, Figura 28(a). Após a eliminação do termo, a representação da árvore pode ser observada na Figura 28(b).

Outros dois ramos são analisados $R_1 = \{\mathbf{0X0}, 001\}$ e $R_2 = \{\mathbf{100}, 111\}$, para esses, ocorre um **Pseudo Deslocamento**, e nesse caso não há eliminação. O novo caminho inserido na árvore é $R_g = \{\mathbf{X00}, 001\}$, Figura 28(c).

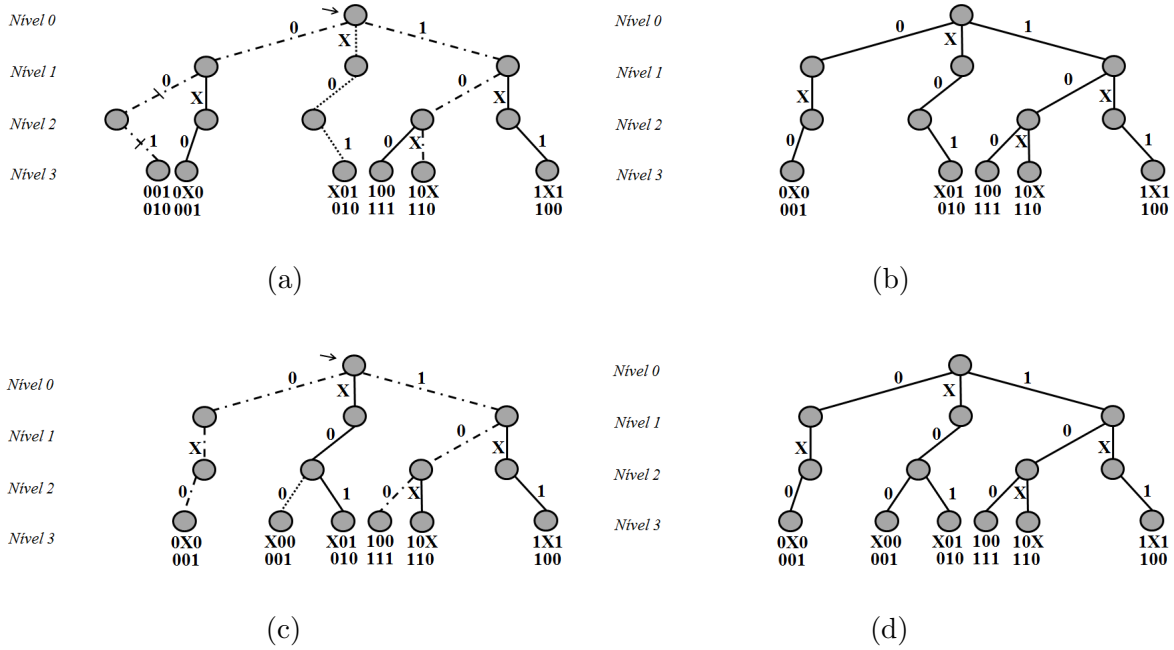
O resultado final é a árvore da Figura 28(d). Os caminhos existentes do nó raiz até os nós folhas da árvore representam os implicantes primos obtidos: $\{0X0, 001\}$, $\{X00, 001\}$, $\{X01, 010\}$, $\{100, 111\}$, $\{10X, 110\}$ e $\{1X1, 100\}$. O algoritmo GPMultiplo gera todos os implicantes primos das funções e da combinação entre as funções.

De posse dos implicantes primos gerados, a cobertura múltipla é realizada através de PLI 0 e 1. Considere $x_1 = 0X0$, $x_2 = X00$, $x_3 = X01$, $x_4 = 100$, $x_5 = 10X$ e $x_6 = 1X1$, a formulação do problema é descrita como apresentada na Tabela 13.

Após eliminar as linhas em duplicidade, gera-se a cobertura múltipla para F_{M_4}, de acordo com a Tabela 14.

Os implicantes primos obtidos após a cobertura múltipla são: $x_1 = 0X0$, $x_3 = X01$,

Figura 28 – Aplicação do Método GPMultiplo no nível 0: F_{M_4}



Fonte: Elaborado pelo Autor

Tabela 13 – Cobertura Múltipla de F_{M_4}

$MIN\ 3x_1 + 3x_2 + 3x_3 + 4x_4 + 3x_5 + 3x_6$	
f_{15}	$x_4 + x_5 \geq 1$ $x_5 + x_6 \geq 1$ $x_6 \geq 1$
f_{16}	$x_3 \geq 1$ $x_4 + x_5 \geq 1$ $x_3 + x_5 \geq 1$
f_{17}	$x_1 + x_2 \geq 1$ $x_1 \geq 1$ $x_2 + x_4 \geq 1$
$x_1, x_2, x_3, x_4, x_5, x_6 \in \{0, 1\}$	

Fonte: Elaborado pelo Autor.

$x_4 = 100$ e $x_6 = 1X1$. A partir dos implicantes múltiplos, deve-se fazer a cobertura singular (cada função deve ser analisada separadamente), conforme apresentado na Tabela 15

Após a cobertura singular, tem-se a solução mínima para cada função. O resultado obtido para as funções de F_{M_4} são:

$f_{15} = 1X1 + 100$;

$f_{16} = X01 + 100$;

Tabela 14 – Cobertura Múltipla sem restrições em duplicidade para F_{M_4}

$MIN\ 3x_1 + 3x_2 + 3x_3 + 4x_4 + 3x_5 + 3x_6$
$x_4 + x_5 \geq 1$
$x_5 + x_6 \geq 1$
$x_6 \geq 1$
$x_3 \geq 1$
$x_3 + x_5 \geq 1$
$x_1 + x_2 \geq 1$
$x_1 \geq 1$
$x_2 + x_4 \geq 1$
$x_1, x_2, x_3, x_4, x_5, x_6\ \epsilon\ \{0, 1\}$

Fonte: Elaborado pelo Autor.

Tabela 15 – Cobertura Singular para as Funções de F_{M_4}

Cobertura Singular f_{15}	Cobertura Singular f_{16}	Cobertura Singular f_{17}
$MIN\ 4x_4 + 3x_6$	$MIN\ 3x_3 + 4x_4$	$MIN\ 3x_1 + 4x_4$
$x_4 \geq 1$	$x_3 \geq 1$	$x_1 \geq 1$
$x_6 \geq 1$	$x_4 \geq 1$	$x_4 \geq 1$
$x_4,\ x_6\ \epsilon\ \{0, 1\}$	$x_3,\ x_4\ \epsilon\ \{0, 1\}$	$x_1,\ x_4\ \epsilon\ \{0, 1\}$

Fonte: Elaborado pelo Autor.

$f_{17} = 0X0 + 100$.

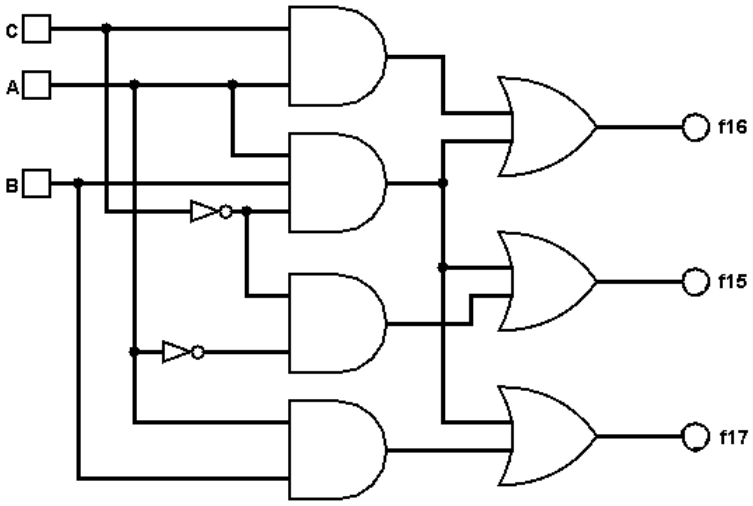
O implicante primo compartilhado $\{100, 111\}$ tem o seu custo relacionado as entradas da porta AND contabilizado apenas uma vez, logo o $custo_{F_{M_4}} = 15$. O circuito implementado para a função mínima de F_{M_4} pode ser observado na Figura 29.

Para demonstrar a vantagem de se utilizar funções com múltiplas saídas, considerando o exemplo supracitado, minimizando as funções separadamente, os implicantes primos gerados são:
$f_{15} = 10X + 1X1$;
$f_{16} = X01 + 10X$;
$f_{17} = 0X0 + X00$.

O custo total para a execução das três funções, separadamente, é composto pela soma dos custos de entradas nas portas AND e OR de cada uma das funções. O $custo_{total} = 18$.

Os circuitos lógicos para cada função minimizada (f_{15}, f_{16} e f_{17}), podem ser visualizados nas Figuras 30, 31 e 32, respectivamente. Observe que para múltiplas saídas,

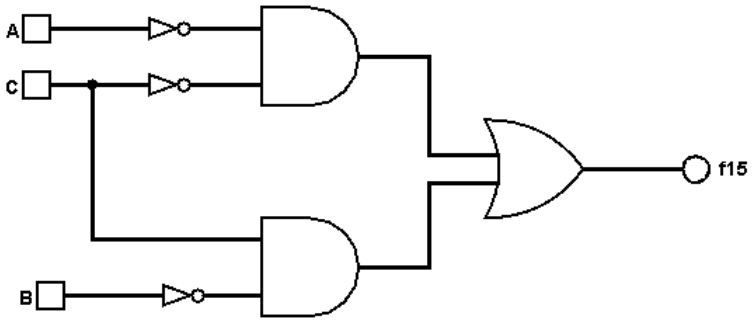
Figura 29 – Circuito Lógico da função mínima de F_{M_4}



Fonte: Elaborado pelo Autor.

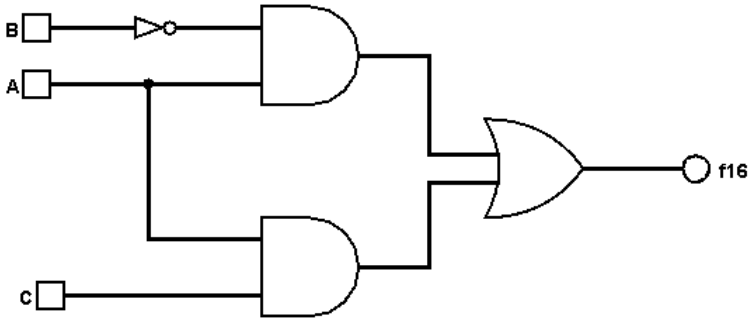
Figura 29, o número de entradas para as portas lógicas diminuiu.

Figura 30 – Circuito Lógico da função mínima de f_{15}



Fonte: Elaborado pelo Autor.

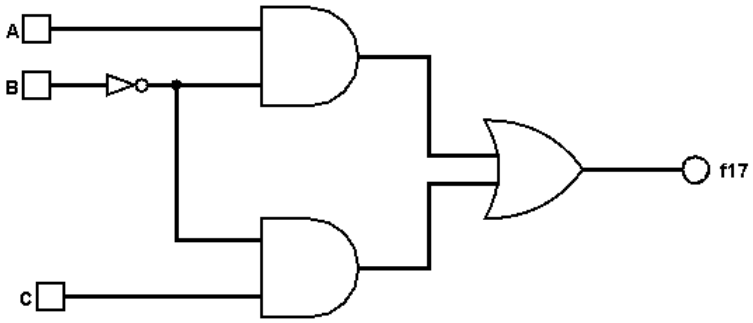
Figura 31 – Circuito Lógico da função mínima de f_{16}



Fonte: Elaborado pelo Autor.

A seguir, o algoritmo MultiGeraPlex é apresentado. Se difere do GPMultiplo pois não gera todos os implicantes primos da função, e existe uma atualização da *tag* de função do termo dependendo do tipo de situação que ocorre ao aplicar a operação do consenso.

Figura 32 – Circuito Lógico da função mínima de f_{17}



Fonte: Elaborado pelo Autor.

4.3 MULTIGERAPLEX - MGP

O MultiGeraPlex foi desenvolvido para geração de implicantes primos de funções booleanas com múltiplas saídas. O algoritmo foi baseado na filosofia do método GPMultiplo, porém, existe a possibilidade de atualização entre as *tags* de funções, quando a operação de consenso é aplicada entre dois termos.

Assim como no GPMultiplo, denomina-se F o conjunto de funções a serem minimizadas, e $\{f_1, f_2, \ldots, f_n\}$, as funções dadas, em que n representa o número máximo de funções. Seja T o conjunto de termos a serem analisados pertencentes a cada função, este pode ser composto por mintermos, *don't care states* ou a combinação destes, e é definido como t. Seja R a representação utilizada para definir o termo t e as funções que este pertence F_t, $R = \{t, F_t\}$.

O conjunto F é representado por uma sequência binária, de tamanho n (número total de funções a serem analisadas), em que 1 sinaliza a presença do termo na função, e 0 a ausência deste.

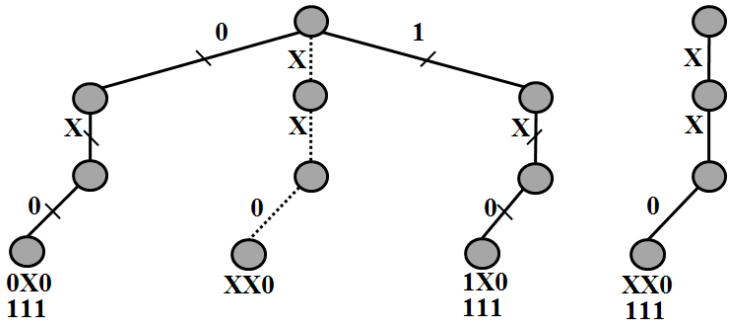
As situações que ocorrem quando aplica-se o consenso entre dois termos são similares as do GPMultiplo. As *tags* relacionadas as funções de cada termo (F_{t_1} e F_{t_2}) podem ser atualizadas, aplicando um XOR entre a *tag* do termo e a do termo gerado. Para as situações em que as funções são as mesmas para os termos (Fusão, Deslocamento e Expansão) o procedimento não altera em relação ao outro algoritmo.

Assim como no método anterior, as situações que podem ocorrer com a aplicação da operação de consenso no método MultiGeraPlex são: Fusão, Fusão Parcial, Pseudo Fusão, Deslocamento, Pseudo Deslocamento e Expansão. Porém, quando algumas situações ocorrem, pode-se atualizar as *tags* de funções. Devido a essa atualização, nem todos os implicantes primos da função são gerados e em alguns casos a solução mínima não é

garantida.

- **Fusão**: $F_{t_1} = F_{t_2}$ e, ao aplicar a operação de consenso entre (t_1, t_2), o termo gerado, t_g, cobre os de origem. Os termos analisados são eliminados, reduzindo, consequentemente, o número total de caminhos na árvore. $R_g = \{t_g, F_{t_g}\}$, $F_{t_g} = F_{t_1} = F_{t_2}$. (Figura 33).

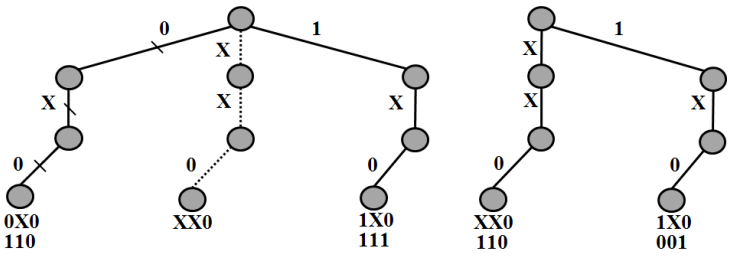
Figura 33 – MultiGeraPlex: exemplo de Fusão



Fonte: Elaborado pelo Autor.

- **Fusão Parcial**: $F_{t_1} \subset F_{t_2}$, ou seja, F_{t_1} é um subconjunto próprio de F_{t_2}. O termo gerado, t_g, cobre t_1 e t_2, porém, apenas t_1 é eliminado, e F_{t_2} deve ser atualizada (Figura 34).

Figura 34 – MultiGeraPlex: exemplo de Fusão Parcial

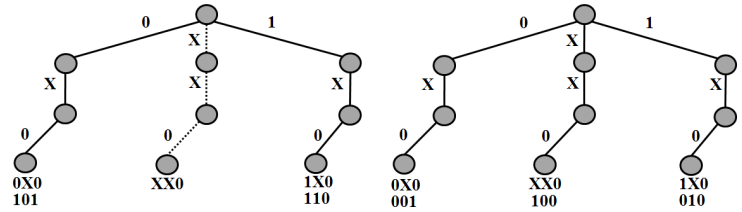


Fonte: Elaborado pelo Autor.

- **Pseudo Fusão**:$F_{t_1} \not\subset F_{t_2}$ $F_{t_1} \cap F_{t_2} \neq \emptyset$, ou seja, t_1 e t_2 estão presentes em pelo menos uma mesma função, e existe pelo menos uma função distinta em cada um dos F_t. O consenso entre t_1 e t_2, resulta em t_g, que cobre t_1 e t_2, porém, eles não são eliminados. O conjunto de função de t_g é formado por $F_{t_1} E F_{t_2}$ (Figura 35).

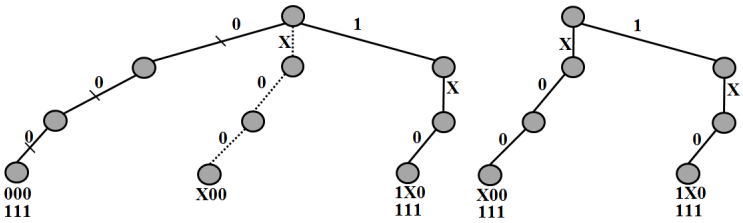
- **Deslocamento**: $F_{t_1} = F_{t_2}$ ou $F_{t_1} \subset F_{t_2}$, consenso é aplicado entre t_1 e t_2; obtem-se t_g, que cobre apenas t_1. Este é eliminado e o conjunto de funções de t_g, $F_{t_g} = F_{t_1}$. A Figura 36(a) representa o Deslocamento quando o conjunto de funções é igual, e a Figura 36(b) quando é sub-conjunto.

Figura 35 – MultiGeraPlex: exemplo de Pseudo Fusão

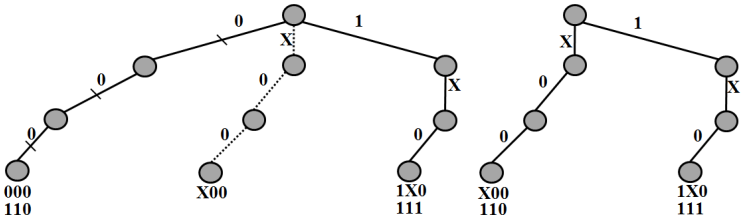


Fonte: Elaborado pelo Autor.

Figura 36 – MultiGeraPlex: exemplo de Deslocamento



(a) Deslocamento, $F_{t_1} = F_{t_2}$

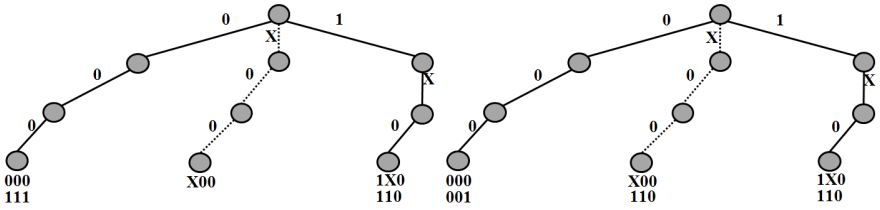


(b) Deslocamento, $F_{t_1} \subseteq F_{t_2}$

Fonte: Elaborado pelo Autor.

- **Pseudo Deslocamento**: $F_{t_1} \subset F_{t_2}$, consenso aplicado entre t_1 e t_2, gera t_g, que cobre apenas o t_2. Para esse caso, não há eliminação. O conjunto de funções F_{t_2} deve ser atualizado (Figura 37).

Figura 37 – MultiGeraPlex: exemplo de Pseudo Deslocamento

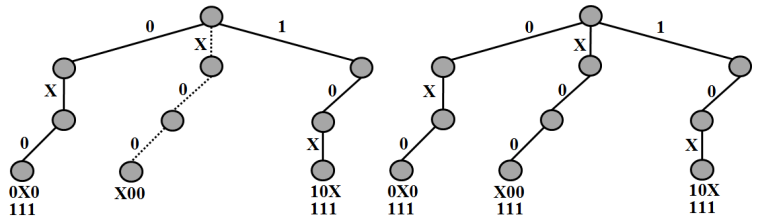


Fonte: Elaborado pelo Autor.

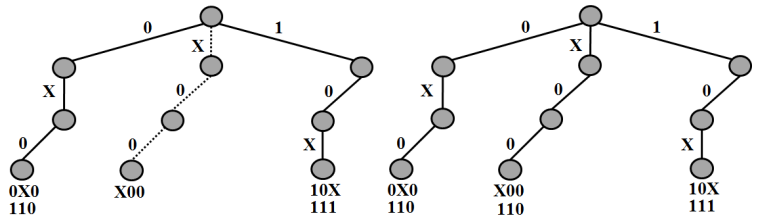
- **Expansão**: será aplicada quando existir função em comum entre F_{t_1} e F_{t_2}; ao aplicar o consenso, o termo gerado t_g, não cobre t_1 e t_2. Deve ser feita a análise de F_{t_1} e F_{t_2}

para a composição de F_{t_g}. As Figuras 38(a), 38(b) e 38(c), representam Expansão quando $F_{t_1} = F_{t_2}$, $F_{t_1} \subseteq F_{t_2}$ e $F_{t_1} \nsubseteq F_{t_2}$).

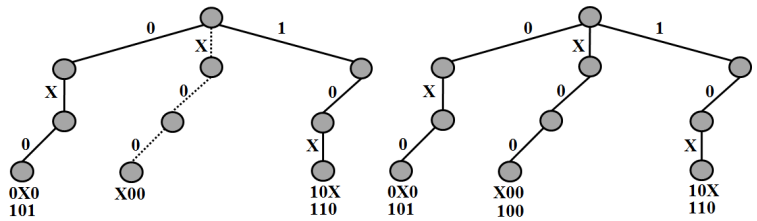
Figura 38 – MultiGeraPlex: exemplo de possíveis ocorrências de Expansão



(a) Expansão, $F_{t_1} = F_{t_2}$



(b) Expansão, $F_{t_1} \subseteq F_{t_2}$



(c) Expansão, $F_{t_1} \not\subset F_{t_2}$, $F_{t_1} \cap F_{t_2} \neq \emptyset$

Fonte: Elaborado pelo Autor.

4.3.1 Algoritmo MultiGeraPlex e fluxograma

Uma das etapas do MultiGeraPlex pode ser visualizada conforme descrito no Algoritmo 3. Similar ao GPMultiplo, proposto anteriormente, difere na composição das *tags* relacionadas a função após cada operação de consenso. Pode haver atualização nas *tags*, esta atualização pode diminuir o número de implicantes primos gerados, e consequentemente o número de comparações realizadas.

Algoritmo 3: Parte do Algoritmo MultiGeraPlex

Entrada: Arquivo contendo os mintermos e *don't care* da função, sub-árvore Esquerda otimizada

```
1  início
2      Percorrer Arquivo de Entrada;
3      Inserir na árvore cada mintermo e don't care;
4      se Sub-árvore direita não é nula então
5          para i = nivel − 1 até i = 1 faça
6              repita
7                  Encontrar nó pai;
8                  Verificar funções que cada termo pertence;
9                  se Existir função em comum então
10                     Aplicar consenso entre ramos;
11                     se (Consenso pode ser aplicado) então
12                         Gerar t_g e Inserir t_g na árvore;
13                         se Funções Iguais então
14                             Atualizar F_{t_g}, se necessário;
15                             se Fusão então
16                                 Eliminar t_1 e t_2; Atualizar número de filhos do nó pai;
17                             senão se Deslocamento então
18                                 Eliminar t_i coberto por t_g;
19                             fim
20                             senão se Expansão então
21                                 Atualizar número de filhos do nó pai;
22                             fim
23                         senão se F_{t_1} ⊂ F_{t_2} então
24                             Atualizar F_{t_g}, se necessário;
25                             se Fusão Parcial então
26                                 Eliminar t_1; Atualizar F_{t_2};
27                             senão se Deslocamento então
28                                 Eliminar t_1;
29                             fim
30                             senão se Pseudo Deslocamento então
31                                 Atualizar F_{t_2}; Atualizar número de filhos do nó pai;
32                             fim
33                             senão se Expansão então
34                                 Atualizar número de filhos do nó pai;
35                             fim
36                         fim
37                         senão se :F_{t_1} ⊄ F_{t_2} e F_{t_1} ∩ F_{t_2} ≠ ∅ então
38                             se Pseudo Fusão então
39                                 Atualizar F_{t_1} e F_{t_2}; Atualizar número de filhos do nó pai;
40                             senão se Pseudo Deslocamento então
41                                 Atualizar F_t referente ao termo coberto por t_g;
42                                 Atualizar número de filhos do nó pai;
43                             fim
44                             senão se Expansão então
45                                 Atualizar número de filhos do nó pai;
46                             fim
47                         fim
48                     fim
49                 fim
50                 Atualizar ponteiros de comparação;
51             até (Existir ramo a ser comparado no nível);
52             Atualizar o nível da árvore;
53         fim
54     fim
55 fim
```

A diminuição do número de implicantes primos, pode ser considerada uma desvantagem quando o implicante primo não gerado seria parte da função mínima. Ou seja, nem todos os implicantes primos da função mínima estão contidos no conjunto de implicantes primos gerados. Isso ocorre, quando existe implicante primo múltiplo que é um subcubo de implicantes primo de uma das funções, essa atualização gera prejuízo para o resultado final, não garantindo a solução mínima, porém o resultado é melhor ou igual se comparado a simplificação individual.

Para os casos em que existem implicantes primos compartilhados entre as funções, a diminuição do número gerado, é vantajosa, pois todos os implicantes primos da função mínima estão contidos no conjunto gerado.

Como o desenvolvimento do algoritmo MultiGeraPlex é muito semelhante ao GPMultiplo, o fluxograma principal é o mesmo que o apresentado na Figura 23, diferenciando apenas na atualização de *tags* de função quando o consenso é aplicado. Na Figura 39 pode ser visualizado a função que representa a atualização das *tags* que ocorre no MultiGeraPlex.

4.3.2 Exemplos de minimização utilizando MultiGeraPlex e PLI 0 e 1

Para exemplificar as situações que podem ocorrer com a aplicação do consenso iterativo, considere as funções f_{18} e f_{19} que pertencem ao conjunto F_{M_5}, descrita na expressão 16.

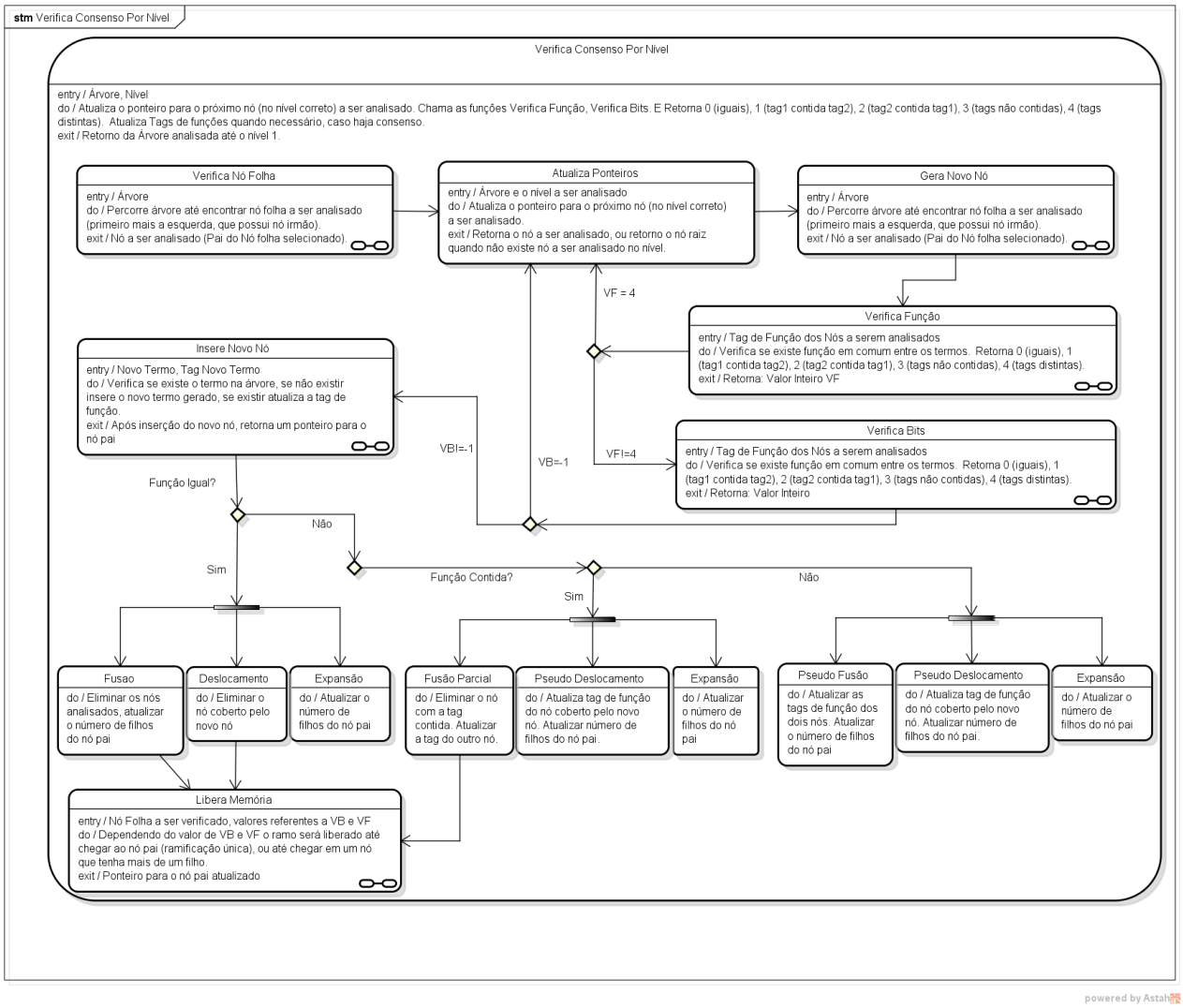
$$\begin{aligned} f_{18}(A,B,C) &= \sum m(0,1,2,5,7) \\ f_{19}(A,B,C) &= \sum m(0,1,2,4,5,7) \end{aligned} \tag{16}$$

Os mintermos pertencentes as funções estão representados na árvore, conforme Figura 40. Os passos da aplicação do consenso em cada nível da árvore são apresentados, conforme Figuras 41, 42 e 44.

Para funções com múltiplas saídas além da representação de cada mintermo no nó folha da árvore, a *tag* de funções também deve ser listada e analisada. Cada caminho na árvore é a representação de um termo; os nós folhas armazenam o termo e também a lista de função deste termo. Para o exemplo, o ramo 000, está contido na função f_{18} e também na f_{19}, logo sua representação é $\{000, 11\}$, $R = \{t, F_t\}$.

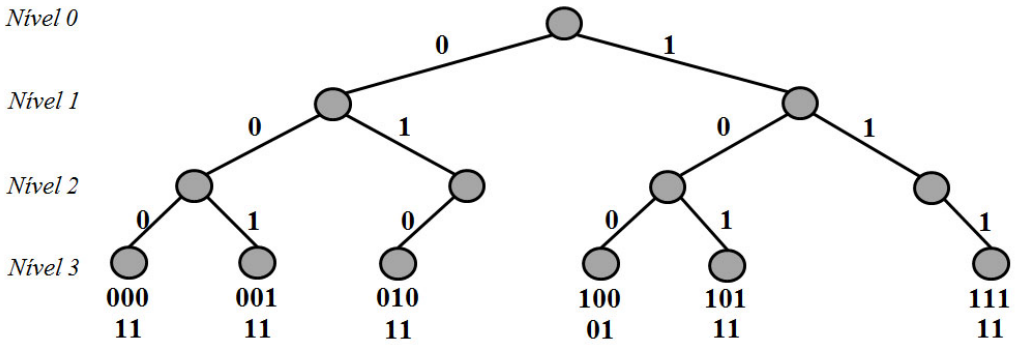
A análise se inicia no nível 3, onde estão presentes os nós folhas, e a partir destes nós verifica-se o nó pai, presente no nível 2. Considere, $t_1 = \{000\}$, $F_{t_1} = \{11\}$, $R_1 = \{000, 11\}$

Figura 39 – Fluxograma que Representa a Atualização das *Tags* de Função do Algoritmo MultiGeraPlex



Fonte: Elaborado pelo Autor.

Figura 40 – Representação dos mintermos das funções de F_{M_5} em árvore



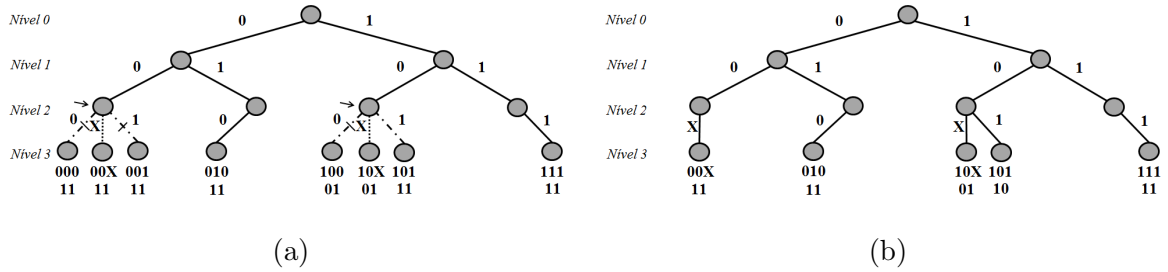
Fonte: Elaborado pelo Autor.

e $t_2 = \{001\}$, $F_{t_2} = \{11\}$, $R_2 = \{001, 11\}$, como os termos estão presentes em funções em comum, pode-se analisá-los. Ocorre a situação de **Fusão**. Ao aplicar o consenso entre t_1

e t_2, o termo gerado é o $t_g = 00X$ e a *tag* de função $F_{t_g} = \{11\}$, logo, $R_g = \{00X, 11\}$. Os ramos de origem são eliminados.

Seguindo a análise no nível, considere $t_1 = \{100\}$, $F_{t_1} = \{01\}$, $R_1 = \{100, 01\}$ e $t_2 = \{101\}$, $F_{t_2} = \{11\}$, $R_2 = \{101, 11\}$. Ocorre uma **Fusão parcial**, pois F_{t_1} está contida em F_{t_2}, e t_g, cobre os termos de origem, porém apenas t_1 é eliminado. O termo gerado é o $t_g = \{10X\}$, e $F_{t_g} = \{01\}$, $R_g = \{10X, 01\}$, a *tag* de função de R_2, F_{t_2}, é atualizada, resultando em $R_2 = \{101, 10\}$. A Figura 41(a) representa as duas situações descritas e, o resultado obtido após as eliminações e atualização na árvore pode ser observado na Figura 41(b).

Figura 41 – Aplicação do Método MultiGeraPlex no nível 2 da árvore: F_{M_5}



(a) (b)

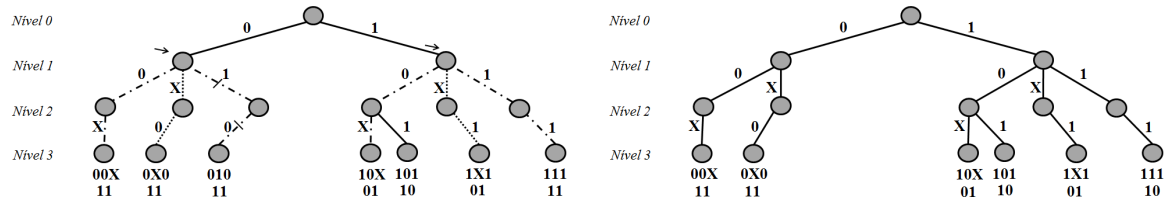
Fonte: Elaborado pelo Autor

Como não há mais nós a serem analisados no nível 2, (os nós filhos do nível 3 já foram todos visitados), atualiza-se para os nós pais pertencentes ao nível 1. Considerando $t_1 = \{00X\}$, $F_{t_1} = \{11\}$, $R_1 = \{00X, 11\}$ e $t_2 = \{010\}$, $F_{t_2} = \{11\}$, $R_2 = \{010, 11\}$, como, o conjunto de funções é igual e analisando o segundo *bit* de cada termo (pertencentes ao nível 2), a situação obtida é o **Deslocamento**. Um novo caminho é inserido na árvore, o ramo gerado é o $R_g = \{0X0, 11\}$, e R_2 é eliminado.

No mesmo nível, seja $t_1 = \{10X\}$ e $F_{t_1} = \{01\}$, $R_1 = \{10X, 01\}$ e $t_2 = \{111\}$, $F_{t_2} = \{11\}$, $R_2 = \{111, 11\}$, como o conjunto de funções do primeiro termo, F_{t_1}, está contido em F_{t_2}, após aplicar o consenso, verifica-se que ocorre o **Pseudo Deslocamento**, pois $t_g = \{1X1\}$ cobre t_2. O termo t_2 não é eliminado e, ocorre apenas uma atualização no conjunto de funções, F_{t_2}, resultando em $R_2 = \{111, 10\}$; o novo caminho gerado é inserido na árvore, $t_g = \{1X1\}$ e $F_{t_g} = \{01\}$, $R_g = \{1X1, 01\}$. As duas situações (Deslocamento e Pseudo Deslocamento) e o resultado, após eliminação, estão representados nas Figuras 42(a) e 42(b), respectivamente.

Ainda no nível 1, considere $t_1 = \{101\}$ e $t_2 = \{111\}$, $R_1 = \{101, 10\}$ e $R_2 = \{111, 10\}$. A **Fusão** é obtida, já que $F_{t_1} = F_{t_2}$ e o termo gerado, $t_g = \{1X1\}$, cobre os dois termos de origem, estes são eliminados, conforme Figura 43(a). Como t_g já existe na árvore, verifica-

Figura 42 – Aplicação do Método MultiGeraPlex no nível 1 da árvore: F_{M_5}

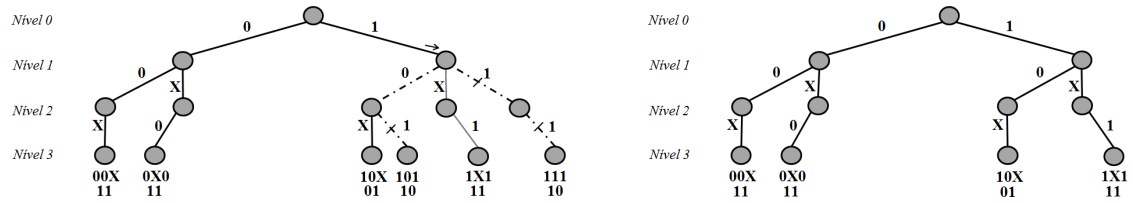


(a) (b)

Fonte: Elaborado pelo Autor

se e atualiza o conjunto de funções que este pertence, logo, $F_{t_g} = \{11\}$, $R_g = \{1X1, 11\}$, como representado na Figura 43(b).

Figura 43 – Aplicação do Método MultiGeraPlex no nível 1 da árvore: F_{M_5}

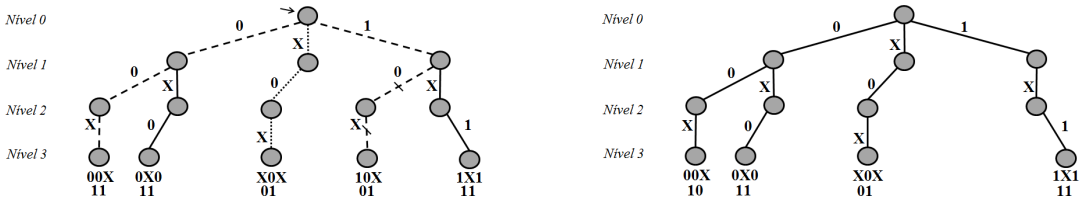


(a) (b)

Fonte: Elaborado pelo Autor

As comparações possíveis no nível 1 foram realizadas, a análise deve ser iniciada no nível 0. Considerando $t_1 = \{00X\}$, $F_{t_1} = \{11\}$, $R_1 = \{00X, 11\}$ e $t_2 = \{1X0\}$, $F_{t_2} = \{01\}$, $R_2 = \{1X0, 01\}$, o resultado é uma **Fusão parcial**, e o t_2 é eliminado, Figura 44(a). O novo caminho a ser inserido na árvore é representado por $R_g = \{X0X, 01\}$. A Figura 44(b) apresenta o resultado da comparação.

Figura 44 – Aplicação do Método MultiGeraPlex no nível 0 da árvore: F_{M_5}



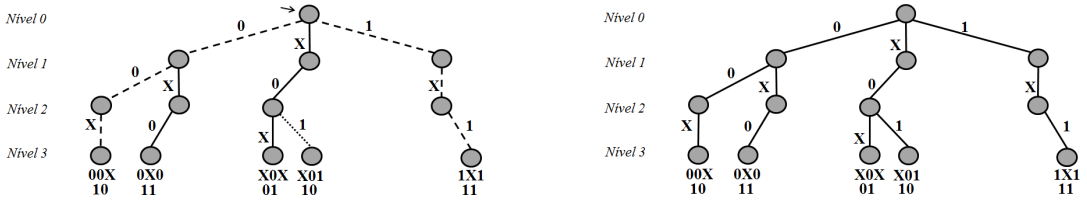
(a) (b)

Fonte: Elaborado pelo Autor

Considerando o próximo nó a ser analisado no nível, seja $t_1 = \{00X\}$, $F_{t_1} = \{10\}$, $R_1 = \{00X, 10\}$ e $t_2 = \{1X1\}$, $F_{t_2} = \{11\}$, $R_2 = \{1X1, 11\}$, uma **Expansão** pode ser

verificada. O novo caminho a ser inserido na árvore, $R_g = \{X01, 10\}$, no qual o termo t_g não cobre nenhum dos ramos de origem, como pode ser visualizado na Figura 45(a). O resultado final está representado na Figura 45(b).

Figura 45 – Aplicação do Método MultiGeraPlex no nível 0 da árvore: F_{M_5}



(a) (b)

Fonte: Elaborado pelo Autor

O problema de cobertura pode ser formulado, a partir dos implicantes primos obtidos e das funções nos quais os mintermos cobertos pertencem. Os implicantes primos de F_{M_5} são: $\{00X, 10\}$, $\{0X0, 11\}$, $\{X0X, 01\}$, $\{X01, 10\}$ e $\{1X1, 11\}$.

Considere $x_1 = 00X$, $x_2 = 0X0$, $x_3 = X0X$, $x_4 = X01$ e $x_5 = 1X1$ para a representação na tabela de cobertura.

Formula-se o problema para a cobertura múltipla, conforme Tabela 16, considerando o conjunto de implicantes primos gerados.

Tabela 16 – Cobertura Múltipla para F_{M_5}

$MIN\ 3x_1 + 3x_2 + 1x_3 + 3x_4 + 3x_5$	
f_{18}	$x_1 + x_2 \geq 1$
	$x_1 + x_4 \geq 1$
	$x_2 \geq 1$
	$x_4 + x_5 \geq 1$
	$x_5 \geq 1$
f_{19}	$x_2 + x_3 \geq 1$
	$x_3 \geq 1$
	$x_2 \geq 1$
	$x_3 \geq 1$
	$x_3 + x_5 \geq 1$
	$x_5 \geq 1$
$x_1, x_2, x_3, x_4, x_5 \epsilon \{0, 1\}$	

Fonte: Elaborado pelo Autor.

Elimina-se as restrições redundantes e o resultado da formulação pode ser visualizado na Tabela 17.

Tabela 17 – Cobertura Múltipla sem restrições em duplicidade para F_{M_5}

$MIN\ 3x_1 + 3x_2 + 1x_3 + 3x_4 + 3x_5$
$x_1 + x_2 \geq 1$
$x_1 + x_4 \geq 1$
$x_2 \geq 1$
$x_4 + x_5 \geq 1$
$x_5 \geq 1$
$x_2 + x_3 \geq 1$
$x_3 \geq 1$
$x_3 + x_5 \geq 1$
$x_1, x_2, x_3, x_4, x_5 \epsilon \{0, 1\}$

Fonte: Elaborado pelo Autor.

Os implicantes primos obtidos após a cobertura múltipla: x_1, x_2, x_3 e x_5. O implicante primo x_4 não faz mais parte da solução.

Formula-se a cobertura singular para cada uma das funções que compõe F_{M_5}, a partir do conjunto de implicantes primos gerados na cobertura múltipla. As restrições redundantes podem ser eliminadas. A cobertura singular está formulada conforme descrito na Tabela 18.

Tabela 18 – Cobertura Singular para as Funções de F_{M_5}

Cobertura Singular f_{18}	Cobertura Singular f_{19}
$MIN\ 3x_1 + 3x_2 + 3x_5$	$MIN\ 3x_2 + x_3 + 3x_5$
$x_1 + x_2 \geq 1$	$x_2 + x_3 \geq 1$
$x_1 \geq 1$	$x_3 \geq 1$
$x_2 \geq 1$	$x_2 \geq 1$
$x_5 \geq 1$	$x_3 + x_5 \geq 1$
$x_1, x_2, x_5\ \epsilon\ \{0, 1\}$	$x_5 \geq 1$
	$x_2, x_3,\ x_5\ \epsilon\ \{0, 1\}$

Fonte: Elaborado pelo Autor.

A cobertura singular para essas funções de múltiplas saídas são:
$f_{18} = x_1 + x_2 + x_5 \rightarrow f_{18} = 00X + 0X0 + 1X1$;
$f_{19} = x_2 + x_3 + x_5 \rightarrow f_{19} = 0X0 + X0X + 1X1$.

O custo total para F_{M_5} é 12, pois os implicantes $0X0$ e $1X1$ são compartilhados pelas funções, por isso, são contabilizados (entrada portas AND) uma única vez.

Ao minimizar as funções f_{18} e f_{19} individualmente, a função mínima é a mesma

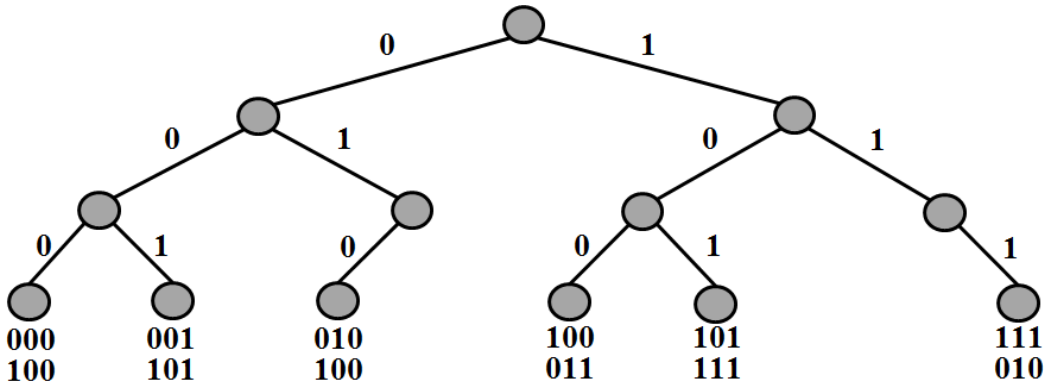
obtida anteriormente, porém o custo mínimo relacionado as funções é de $custo_{f_{18}} = 9$ e $custo_{f_{19}} = 7$, $custo_{total} = 16$.

Outro exemplo de minimização pode ser observado através do conjunto de funções F_{M_6}, expressão 17, composto pelas funções f_{20} , f_{15} e f_{16}.

$$\begin{aligned} f_{20} &= \sum m(0,1,2,5) \\ f_{15} &= \sum m(4,5,7) \\ f_{16} &= \sum m(1,4,5) \end{aligned} \tag{17}$$

Na Figura 46, pode-se observar a representação dos mintermos na árvore, e está descrito em cada nó folha o termo t e o conjunto de funções que pertence F_t.

Figura 46 – Representação dos termos das funções de F_{M_6} em árvore



Fonte: Elaborado pelo Autor.

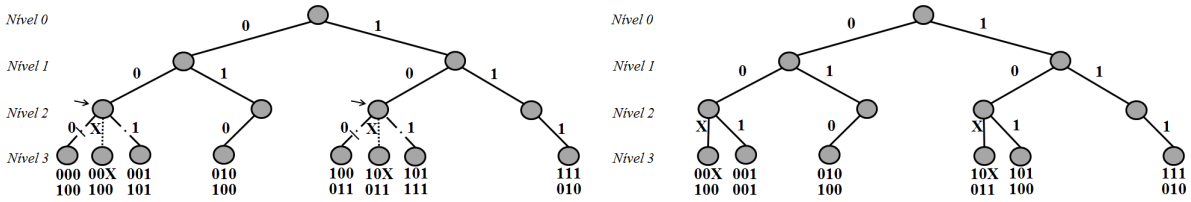
Os passos da aplicação do MultiGeraPlex são apresentados através das árvores das Figuras 47, 48, 49, nas quais cada situação é descrita e os resultados apresentados.

Inicialmente, o nó pai de $\{000\}$ e $\{001\}$ deve ser analisado. Considere $t_1 = \{000\}$, $F_{t_1} = \{100\}$, $R_1 = \{000, 100\}$ e $t_2 = \{001\}$, $F_{t_2} = \{101\}$, $R_2 = \{001, 101\}$ os dois nós filhos, do nível 3 que deverão ser analisados. Uma **Fusão Parcial** ocorre, já que $F_{t_1} \subset F_{t_2}$ e, o termo gerado $t_g = 00X$ cobre t_1 e t_2, porém apenas t_1 é eliminado. Deve-se atualizar as funções de t_2, o que resulta em $F_{t_2} = \{001\}$. O novo caminho a ser inserido na árvore é $R_g = \{00X, 100\}$.

Ao analisar os nós do nível 2, encontra-se outro em que seus filhos podem ser combinados. Seja, $t_1 = \{100\}$, $F_{t_1} = \{011\}$, $R_1 = \{100, 011\}$ e $t_2 = \{101\}$, $F_{t_2} = \{111\}$, $R_2 = \{101, 111\}$. Para este caso também ocorre uma **Fusão Parcial** e o termo t_1 é eliminado. O novo ramo inserido na árvore é $R_g = \{10X, 011\}$. As operações realizadas no nível 2 podem ser visualizadas na Figura 47(a) e o resultado na Figura 47(b). Não há

mais nós para serem analisados neste nível, por isso, o processo é realizado no nível 1.

Figura 47 – Aplicação do Método MultiGeraPlex nível 2 da árvore: F_{M_6}



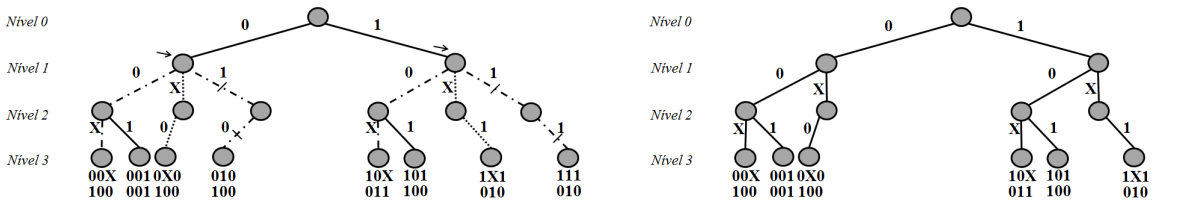
(a) (b)

Fonte: Elaborado pelo Autor

Dois nós pais podem ser analisados no nível 1, considere inicialmente o nó mais a esquerda da árvore (sub-árvore esquerda), nos quais os filhos são: $t_1 = \{00X\}$, $F_{t_1} = \{100\}$, $R_1 = \{00X, 100\}$ e $t_2 = \{010\}$, $F_{t_2} = \{100\}$, $R_2 = \{010, 100\}$. Um **Deslocamento** é obtido, logo, t_2 é eliminado e o termo resultante t_g é composto por $R_g = \{0X0, 100\}$.

Ainda no nível 1, o nó da sub-árvore direita é analisado. Considere $t_1 = \{10X\}$, $F_{t_1} = \{011\}$, $R_1 = \{10X, 011\}$ e $t_2 = \{111\}$, $F_{t_2} = \{010\}$, $R_2 = \{111, 010\}$, ocorre a situação de **Deslocamento**, o novo caminho a ser inserido na árvore é $R_g = \{1X1, 010\}$ e há a eliminação de t_2. As situações ocorridas no nível 1, supracitadas, podem ser observadas na Figura 48(a), e o resultado do processo na Figura 48(b).

Figura 48 – Aplicação do Método MultiGeraPlex no nível 1 da árvore: F_{M_6}



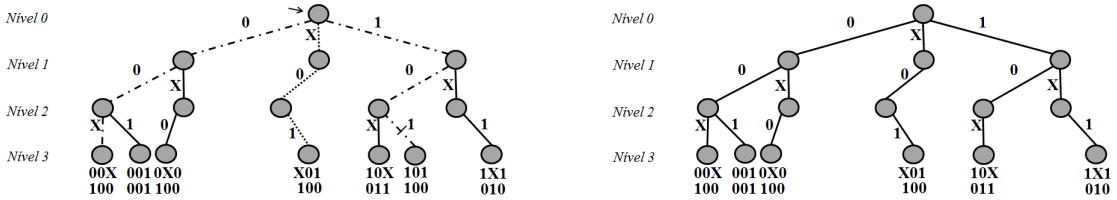
(a) (b)

Fonte: Elaborado pelo Autor

Após feita a análise dos nós do nível 1, o nível 0 deve ser analisado. Considere $t_1 = \{00X\}$, $F_{t_1} = \{100\}$, $R_1 = \{00X, 100\}$ e $t_2 = \{101\}$, $F_{t_2} = \{100\}$, $R_2 = \{101, 100\}$, um **Deslocamento** ocorre, e t_2 é eliminado. O novo caminho inserido na árvore é $R_g = \{X01, 100\}$, como pode ser observado na Figura 49(a). Após a eliminação do termo, a árvore está representada na Figura 49(b).

Ainda no mesmo nível, outros dois ramos devem ser analisados, e também um **Deslocamento** pode ser obtido. Seja, $t_1 = \{001\}$, $F_{t_1} = \{001\}$, $R_1 = \{001, 001\}$ e

Figura 49 – Aplicação do Método MultiGeraPlex no nível 0 da árvore: F_{M_6}

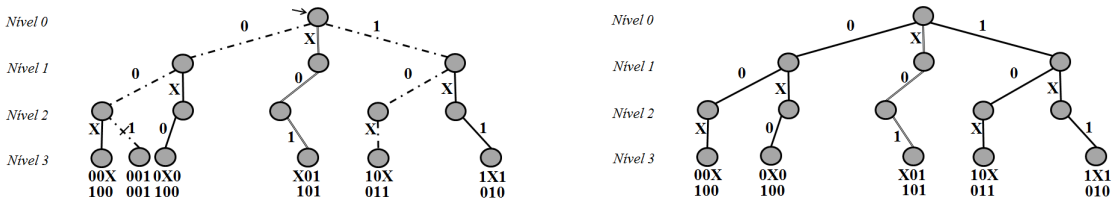


(a) (b)

Fonte: Elaborado pelo Autor

$t_2 = \{10X\}$, $F_{t_2} = \{011\}$, $R_2 = \{10X, 011\}$. O mintermo t_1 é eliminado, e como o ramo gerado, $t_g = \{X01\}$ já existe na árvore, verifica-se apenas a *tag* de função, F_{t_g}. Atualiza-se F_{t_g}, resultando em $R_g = \{X01, 101\}$. A Figura 50(a) representa a comparação realizada. Após a eliminação do mintermo, a árvore resultante pode ser observada na Figura 50(b).

Figura 50 – Aplicação do Método MultiGeraPlex no nível 0 da árvore: F_{M_6}



(a) (b)

Fonte: Elaborado pelo Autor

O resultado final é a árvore da Figura 50(b), no qual os caminhos da árvore representam os implicantes primos obtidos com a aplicação do Algoritmo MultiGeraPlex. Esses podem ser descritos como: $\{00X, 100\}$, $\{0X0, 100\}$, $\{X01, 101\}$, $\{10X, 011\}$ e $\{1X1, 010\}$.

A cobertura para F_{M_6}, expressão 17, é representada a partir dos implicantes primos obtidos na primeira etapa do problema, que são: $\{00X, 100\}$, $\{0X0, 100\}$, $\{X01, 101\}$, $\{10X, 011\}$ e $\{1X1, 010\}$.

Seja, $x_1 = 00X$, $x_2 = 0X0$, $x_3 = X01$, $x_4 = 10X$ e $x_5 = 1X1$, o problema de cobertura pode ser formulado como descrito na Tabela 19.

Exclui-se as restrições redundantes, obtém-se a cobertura múltipla para F_{M_6}, Tabela 20.

O conjunto de implicantes primos obtidos após a cobertura múltipla é: $x_2 = 0X0$,

Tabela 19 – Cobertura Múltipla para F_{M_6}

$MIN\ 3x_1 + 3x_2 + 3x_3 + 3x_4 + 3x_5$	
f_{20}	$x_1 + x_2 \geq 1$
	$x_1 + x_3 \geq 1$
	$x_2 \geq 1$
	$x_3 \geq 1$
f_{15}	$x_4 \geq 1$
	$x_4 + x_5 \geq 1$
	$x_5 \geq 1$
f_{16}	$x_3 \geq 1$
	$x_4 \geq 1$
	$x_3 + x_4 \geq 1$
$x_1, x_2, x_3, x_4, x_5\ \epsilon\ \{0, 1\}$	

Fonte: Elaborado pelo Autor.

Tabela 20 – Cobertura Múltipla para F_{M_6} sem restrições em duplicidade

$MIN\ 3x_1 + 3x_2 + 3x_3 + 3x_4 + 3x_5$
$x_1 + x_2 \geq 1$
$x_1 + x_3 \geq 1$
$x_2 \geq 1$
$x_3 \geq 1$
$x_4 \geq 1$
$x_4 + x_5 \geq 1$
$x_5 \geq 1$
$x_3 + x_4 \geq 1$
$x_1, x_2, x_3, x_4, x_5\ \epsilon\ \{0, 1\}$

Fonte: Elaborado pelo Autor.

$x_3 = X01$, $x_4 = 10X$ e $x_5 = 1X1$.

A cobertura singular para cada uma das funções que compõe F_{M_6} pode ser visualizada na Tabela 21.

Após a cobertura singular, o conjunto mínimo de implicantes primos que compõe cada função é:
$f_{20} = 0X0 + X01$;
$f_{15} = 10X + 1X1$;
$f_{16} = X01 + 10X$.

Como os implicantes primos $X01$ e $10X$ são compartilhados entre duas funções, o

Tabela 21 – Cobertura Singular para as Funções de F_{M_6}

Cobertura Singular f_{20}	Cobertura Singular f_{15}	Cobertura Singular f_{16}
$MIN\ 3x_2 + 3x_3$	$MIN\ 3x_4 + 3x_5$	$MIN\ 3x_3 + 3x_4$
$x_2 \geq 1$	$x_4 \geq 1$	$x_3 \geq 1$
$x_3 \geq 1$	$x_4 + x_5 \geq 1$	$x_4 \geq 1$
$x_2, x_3\ \epsilon\ \{0,1\}$	$x_5 \geq 1$	$x_3 + x_4 \geq 1$
	$x_4,\ x_5\ \epsilon\ \{0,1\}$	$x_3,\ x_4\ \epsilon\ \{0,1\}$

Fonte: Elaborado pelo Autor.

custo mínimo de $F_{M_6} = 8 + 6 = 14$.

Os implicantes primos gerados, para cada função separadamente, executada de forma simples, são os mesmos, porém o custo total difere, pois os implicantes $10X$ e $X01$ são contabilizados duas vezes (em relação a entradas na porta AND), por isso, tem-se um valor total de 18.

Considere o exemplo apresentado na seção 4.2.2, função F_{M_4}, função já minimizada pelo algoritmo GPMultiplo. A seguir é descrito a aplicação do algoritmo MultiGeraPlex para este caso. A disposição dos mintermos da função na árvore, já foram apresentados na Figura 25, seção 4.2.2.

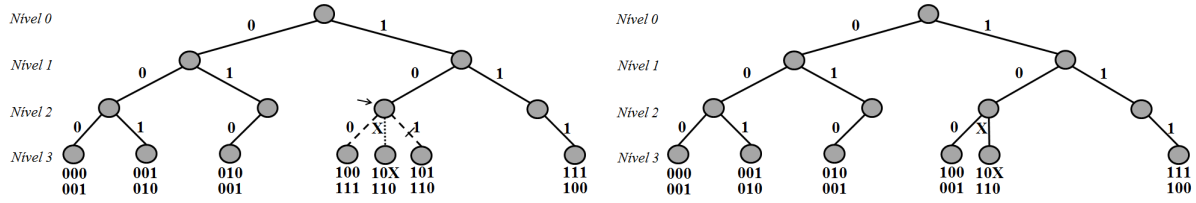
A execução de F_{M_4} utilizando o algoritmo MultiGeraPlex pode ser visualizada nas árvores das Figuras 51, 52, 53, nas quais cada situação de operação de consenso é descrita e os resultados obtidos apresentados.

Considere, $t_1 = \{100\}$, $F_{t_1} = \{111\}$, $R_1 = \{100, 111\}$ e $t_2 = \{101\}$, $F_{t_2} = \{110\}$, $R_2 = \{101, 110\}$, a partir da análise dos nós folhas que representam estes termos, e observando que existem funções em comum entre eles, a operação de consenso pode ser aplicada no nó pertencente ao nível 2. Uma **Fusão Parcial** ocorre, já que $F_{t_2} \subset F_{t_1}$, t_2 é eliminado e uma atualização em F_{t_1} é feita, resultando em $R_1 = \{100, 001\}$. O termo gerado $t_g = 10X$ é inserido na árvore, como mostrado na Figura 51(a).

O resultado após o consenso entre os termos está representado na Figura 51(b). A representação do novo nó folha criado é $R_g = \{10X, 110\}$. Como não há mais nós para serem analisados no nível 2, o processo se inicia no nível 1.

Considere inicialmente o nó da sub-árvore esquerda, no qual os nós filhos são: $R_1 = \{000, 001\}$ ($t_1 = \{000\}$ e $F_{t_1} = |001$) e $R_2 = \{010, 001\}$ ($t_2 = \{010\}$ e $F_{t_2} = |001$). Uma **Fusão** é obtida, logo, t_1 e t_2 são eliminados e o termo resultante t_g é composto por $R_g = \{0X0, 001\}$.

Figura 51 – Aplicação do Método MultiGeraPlex no nível 2 da árvore: F_{M_4}

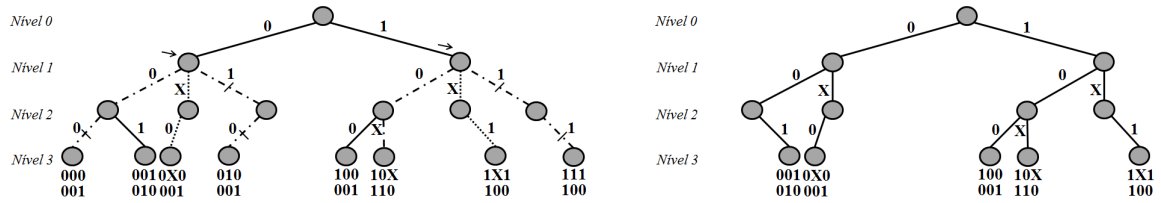


(a) (b)

Fonte: Elaborado pelo Autor

Ainda no nível 1, a análise é feita no nó da sub-árvore direita, $R_1 = \{10X, 110\}$ ($t_1 = \{10X\}$ e $F_{t_1} = |110$) e $R_2 = \{111, 100\}$ ($t_2 = \{111\}$ e $F_{t_2} = |100$), o consenso aplicado gera um **Deslocamento**, logo, o novo ramo, $R_g = \{1X1, 100\}$, é incluído na árvore e elimina-se t_2. As situações ocorridas no nível 1, supracitadas, podem ser visualizadas na Figura 52(a), e o resultado na Figura 52(b).

Figura 52 – Aplicação do Método MultiGeraPlex no nível 1 da árvore: F_{M_4}



(a) (b)

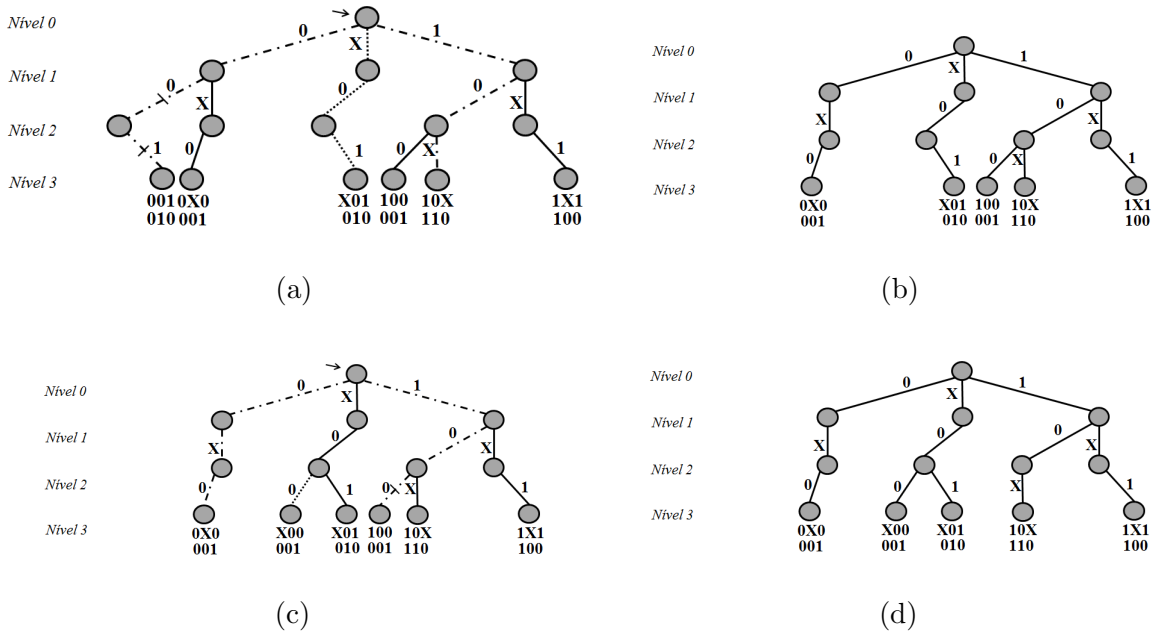
Fonte: Elaborado pelo Autor

Inicia-se a verificação no nível 0, considere $R_1 = \{001, 010\}$ ($t_1 = \{001\}$ e $F_{t_1} = |010$) e $R_2 = \{10X, 110\}$ ($t_2 = \{10X\}$ e $F_{t_2} = |110$). O resultado é um **Deslocamento**, t_1 é eliminado, e o novo caminho, $R_g = \{\mathbf{X01}, 010\}$, é inserido na árvore, conforme apresentado na Figura 53(a). Após a eliminação do termo, a representação da árvore pode ser observada na Figura 53(b).

Ainda considerando o nível 0, seja $R_1 = \{0X0, 001\}$ ($t_1 = \{0X0\}$ e $F_{t_1} = |001$) e $R_2 = \{\mathbf{100}, 001\}$ ($t_2 = \{100\}$ e $F_{t_2} = |001$), ocorre um **Deslocamento**, t_2 é eliminado e o novo caminho inserido na árvore, $R_g = \{X00, 001\}$, conforme representado na Figura 53(c).

O resultado final é a árvore da Figura 53(d). Os caminhos existentes do nó raiz até os nós folhas da árvore representam os implicantes primos obtidos: $\{0X0, 001\}$, $\{X00, 001\}$, $\{X01, 010\}$, $\{10X, 110\}$ e $\{1X1, 100\}$.

Figura 53 – Aplicação do Método MultiGeraPlexo no nível 0 da árvore: F_{M_4}



Fonte: Elaborado pelo Autor

De posse dos implicantes primos gerados, a cobertura múltipla é realizada através de PLI 0 e 1. Considere $x_1 = 0X0$, $x_2 = X00$, $x_3 = X01$, $x_4 = 10X$ e $x_5 = 1X1$, a formulação do problema é descrita como apresentada na Tabela 22.

Tabela 22 – Cobertura Múltipla de F_{M_4}

$MIN\ 3x_1 + 3x_2 + 3x_3 + 3x_4 + 3x_5$	
f_{15}	$x_4 \geq 1$
	$x_4 + x_5 \geq 1$
	$x_5 \geq 1$
f_{16}	$x_3 \geq 1$
	$x_4 \geq 1$
	$x_3 + x_4 \geq 1$
f_{17}	$x_1 + x_2 \geq 1$
	$x_1 \geq 1$
	$x_2 \geq 1$
$x_1, x_2, x_3, x_4, x_5\ \epsilon\ \{0, 1\}$	

Fonte: Elaborado pelo Autor.

Após eliminar as linhas em duplicidade, gera-se a cobertura múltipla para F_{M_4}, de acordo com a Tabela 23.

O conjunto de implicantes primos obtidos após a cobertura múltipla é: $x_1 = 0X0$, $x_2 = X00$, $x_3 = X01$, $x_4 = 10X$ e $x_5 = 1X1$.

Tabela 23 – Cobertura Múltipla sem restrições em duplicidade para F_{M_4}

$MIN\ 3x_1 + 3x_2 + 3x_3 + 4x_4 + 3x_5$
$x_4 \geq 1$
$x_4 + x_5 \geq 1$
$x_5 \geq 1$
$x_3 \geq 1$
$x_3 + x_4 \geq 1$
$x_1 + x_2 \geq 1$
$x_1 \geq 1$
$x_2 \geq 1$
$x_1, x_2, x_3, x_4, x_5\ \epsilon\ \{0, 1\}$

Fonte: Elaborado pelo Autor.

A cobertura singular para cada uma das funções que compõe F_{M_4} pode ser visualizada na Tabela 24.

Tabela 24 – Cobertura Singular para as Funções de F_{M_4}

Cobertura Singular f_{20}	Cobertura Singular f_{15}	Cobertura Singular f_{16}
$MIN\ 3x_4 + 3x_5$	$MIN\ 3x_3 + 3x_4$	$MIN\ 3x_1 + 3x_2$
$x_4 \geq 1$	$x_3 \geq 1$	$x_1 + x_2 \geq 1$
$x_4 + x_5 \geq 1$	$x_4 \geq 1$	$x_1 \geq 1$
$x_5 \geq 1$	$x_3 + x_4 \geq 1$	$x_2 \geq 1$
$x_4,\ x_5\ \epsilon\ \{0, 1\}$	$x_3,\ x_4\ \epsilon\ \{0, 1\}$	$x_1,\ x_2\ \epsilon\ \{0, 1\}$

Fonte: Elaborado pelo Autor.

Após a cobertura singular, a solução mínima para cada função de F_{M_4} é:
$f_{15} = 10X + 1X1$;
$f_{16} = X01 + 10X$;
$f_{17} = 0X0 + X00$.

Como o implicante primo $10X$ é compartilhado pelas funções f_{15} e f_{16} seu custo relacionado a porta AND é contabilizado apenas uma vez. O custo total para a execução de F_{M_4} é 16.

O resultado obtido com a aplicação do algoritmo GPMultiplo, apresentado na seção 4.2.2, para F_{M_4}, é diferente do obtido pelo MultiGeraPlex. Observa-se a existência de um implicante primo múltiplo, que não é primo de nenhuma das funções separadamente, o $A\bar{B}\bar{C}$, em binário o 100, que pertence às três funções de F_{M_4}, no resultado do GPMultiplo.

Para casos como esse, o MultiGeraPlex não gera o implicante primo múltiplo que está contido em um subcubo menor. No MultiGeraPlex, quando um termo múltiplo (que está presente em mais de uma função) é combinado com outro, e gera um terceiro que cobre o termo múltiplo, este é eliminado. Para alguns casos, essa eliminação acarreta a não geração da função de custo mínimo.

Para esse exemplo, com a aplicação do algoritmo MultiGeraPlex, o resultado final não contempla o implicante primo $\{100, 111\}$. Isso porque o algoritmo varre a árvore combinando os termos e atualizando as *tags* das funções para cada combinação. E como há uma combinação entre os mintermos 4 e 5, uma **Fusão Parcial** ocorre, o mintermo 5 é excluído e o mintermo 4 tem a *tag* de função atualizada, gera-se $10X$ e atualiza-se a *tag* do mintermo 4, $100, 001$. Entre os termos 100 e $0X0$, ocorre um **Deslocamento** e o mintermo 100 é excluído. O novo termo é o $X00$.

O conjunto de implicantes primos obtidos com a aplicação do MultiGeraPlex é: $\{0X0, 001\}$, $\{X00, 001\}$, $\{X01, 010\}$, $\{10X, 110\}$ e $\{1X1, 100\}$, e o custo para implementar o circuito lógico é 16.

No Capítulo 5 são apresentados os resultados obtidos através dos algoritmos propostos e também uma comparação desses com a primeira fase do método de Quine-McCluskey para Funções com Múltiplas Saídas.

5 RESULTADOS E DISCUSSÕES

Os algoritmos foram implementados em Linguagem de Programação C e a alocação de memória da maior parte das estruturas foi realizada de forma dinâmica. Os programas foram executados na plataforma Windows, computador com 8 *Gigabytes* de memória, processador core i7, 1.6 *Gigahertz*.

Para o cálculo do custo da função mínima, considera-se o custo associado a quantidade de entradas das portas lógicas envolvidas (AND e OR).

Uma base de dados de entrada, denominada Tudo Um foi criada, contendo todos os termos possíveis relacionado a quantidade de variáveis de entrada. Cada arquivo de entrada, foi nomeado conforme descrito *T1_n*, em que n está associado ao número de variáveis. Gerou-se arquivos em que n variou de 3 até 23, arquivo contendo 8.388.608 termos.

A base Tudo Um gera uma constante, isto é, pode assumir o valor em VCC[1] ou em GND[2]. Esta base foi criada para se analisar e avaliar a performance dos métodos apresentados. Para esta base, o comportamento do Método de Quine-McCluskey para Funções Booleanas com Múltiplas Saídas se comporta como o pior caso, pois realiza todas as comparações possíveis entre os termos. Já o comportamento dos métodos propostos para esta base é o melhor caso, pois a cada comparação uma fusão é realizada, sempre diminuindo o número de nó folhas da árvore.

Também foi criada uma base, em que o posicionamento dos termos, representados no Mapa de Karnaugh, baseia-se em um tabuleiro de xadrez. Os arquivos foram nomeados *TX_n*. Nestes casos não são gerados implicantes, e o conjunto de implicantes primos é composto pelos mintermos da função. Para esta base o Método de Quine-McCluskey tem seu melhor caso, pois não realiza comparações entre os termos, mas há uma comparação preliminar entre as caixas. Esse comportamento se dá pois os termos são classificados em caixas, e a combinação de um termo pertencente a uma caixa com outro termo pertencente

[1]VCC - tensão de corrente contínua, isto é, carga positiva, ou nível lógico 1.
[2]GND é abreviação de GROUND ou terra, isto é, ausência de tensão, ou nível lógico 0

a próxima caixa só é realizada entre caixas com numeração seqüencial.

As demais funções apresentadas, denominadas de *C_i* (*i* é um número inteiro), foram obtidas de artigos, teses, livros e também elaboradas pela autora.

A Tabela 25 representa o número de implicantes e o número de comparações realizadas pelos métodos propostos (GPM e MGP) e pela Primeira Fase do Método de QMM, com a execução dos arquivos da base Tudo Um. Salienta-se que o resultado final para os arquivos da Base Tudo Um é uma constante, isto é, pode ser representado por VCC ou GND. Por isso, não foi especificado o número de implicantes primos, pois, os três métodos geraram 1 implicante primo como resultado final para todos os arquivos da base.

Tabela 25 – Número de Implicantes e Número de Comparações obtidos com a aplicação da Primeira Fase do Método de Quine-McCluskey para Múltiplas Saídas (QMM) e dos Métodos Propostos GPMultiplo (GPM) e MultiGeraPlex (MGP) utilizando a Base Tudo Um

Caso	*Bit*	Termo	Implicante		Comparação	
			QMM	GPM/MGP	QMM	GPM/MGP
$T1_3$	3	8	19	7	60	7
$T1_4$	4	16	65	15	456	15
$T1_5$	5	32	211	31	3535	31
$T1_6$	6	64	665	63	27888	63
$T1_7$	7	128	2059	127	223209	127
$T1_8$	8	256	6305	255	1807760	255
$T1_9$	9	512	19171	511	14784759	511
$T1_{10}$	10	1024	58025	1023	121909800	1023
$T1_{11}$	11	2048	–	2047	–	2047
$T1_{12}$	12	4096	–	4095	–	4095
$T1_{13}$	13	8192	–	8191	–	8191
$T1_{14}$	14	16384	–	16383	–	16383
$T1_{15}$	15	32768	–	32767	–	32767
$T1_{16}$	16	65536	–	65535	–	65535
$T1_{17}$	17	131072	–	131071	–	131071
$T1_{18}$	18	262144	–	262143	–	262143
$T1_{19}$	19	524288	–	524287	–	524287
$T1_{20}$	20	1048576	–	1048575	–	1048575
$T1_{21}$	21	2097152	–	2097151	–	2097151
$T1_{22}$	22	4194304	–	4194303	–	4194303

Fonte: Elaborado pelo Autor

Como pode-se observar na Tabela 25, os resultados obtidos em relação ao número de implicantes gerados pelos métodos propostos em relação ao método de QMM, para o arquivo com 3 variáveis o QMM gera mais que o dobro de implicantes quando comparado com as propostas; e para o arquivo com 10 variáveis essa diferença é superior aumenta consideravelmente, sendo o número de implicantes gerado pelo Método de QMM 56 vezes

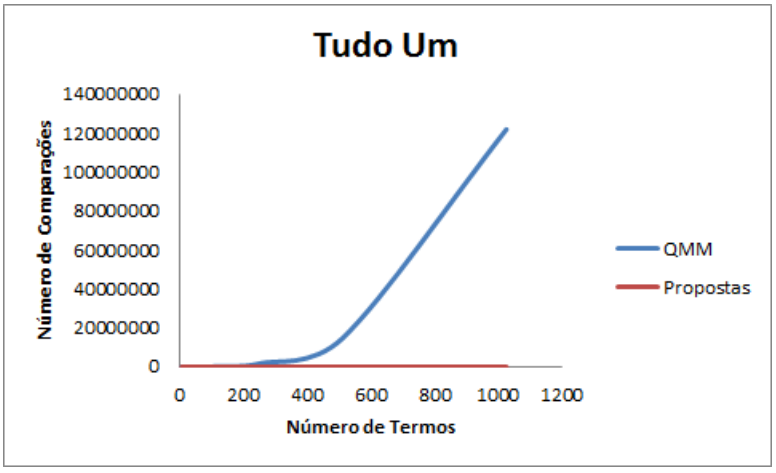
maior que o número de implicantes gerado pelas propostas. Na Figura 54 pode-se observar o gráfico que representa o comportamento do QMM e das Propostas (GPM e MGP), quando se trata do número de implicantes gerados em relação ao número de termos.

Figura 54 – Número de Termos *versus* Número de Implicantes, resultados obtidos com a execução da Primeira Fase do Método de QMM e das Propostas (GPM e MGP)

Fonte: Elaborado pelo Autor.

Em relação ao número de comparações, a superioridade das propostas também pode ser observada, de acordo com a Tabela 25. As comparações entre os termos realizadas pelos Métodos Propostos em relação ao Método de QMM é significativamente menor, para o arquivo inicial (3 variáveis) a diferença já é cerca de 8 vezes menor, atingindo para o arquivo com 10 variáveis uma proporção de 119.000 vezes. O Gráfico que representa o comportamento do Método de QMM e das Propostas (GPM e MGP) está representado na Figura 55.

Figura 55 – Número de Termos *versus* Número de Comparações da Primeira Fase do Método de QMM e das Propostas (GPM e MGP)



Fonte: Elaborado pelo Autor.

Para o GPM e para o MGP, os resultados (apresentados na Tabela 25) obtidos para a Base Tudo Um são iguais, pois ao aplicar o consenso entre dois termos, a única situação que ocorre é a Fusão, não há atualizações de *tags* de funções, por isso o comportamento é similar.

Na Tabela 26 estão dispostas as informações a respeito do Tempo de Execução e utilização de Memórias (Virtual e Física) pelos Métodos QMM, GPM e MGP com a execução da Base Tudo Um.

Ao comparar o tempo de execução gasto pelo QMM para execução dos arquivos da base Tudo Um, pode-se observar (Tabela 26) que para cada incremento no número de variáveis, o tempo de execução do QMM aumenta consideravelmente, inicialmente em $0,0167$ segundos para arquivo de 3 variáveis e para 10 variáveis atingindo $76,10$ segundos. Vale ressaltar, que os algoritmos propostos (GPM e MGP) obtiveram um tempo de execução para o arquivo de 10 variáveis menor que o tempo de execução do QMM para o arquivo de 3 variáveis.

Comparando a memória utilizada pelo QMM, GPM e MGP com a execução do arquivo $T1_{10}$, apresentado na Tabela 26, o Método de QMM consumiu 10^5 vezes mais memória que as Propostas.

Como pode-se observar, para os algoritmos propostos, foi executado o arquivo $T1_{22}$, contendo 22 variáveis, isto é, 4.194.304 termos, com um tempo de execução aceitável, sendo para o MGP inferior a 53 segundos e para o GPM inferior a 37 segundos. O QMM gerou resultados apenas até 10 variáveis, não sendo possível a execução de arquivos maiores.

Com a execução da Base Xadrez, que é o melhor caso para o Quine-McCluskey, pois não há comparações entre os termos, e sim apenas comparação entre uma caixa e a próxima caixa, os resultados obtidos com a execução para os três métodos podem ser observados na Tabela 27. Não há gasto de armazenamento relativo a memória virtual por isso não foi apresentado, assim com não há geração de implicantes, e para o resultado final são considerados os mintermos da função (implicantes primos).

Ao analisar os resultados obtidos com a Base Xadrez, mesmo sendo o melhor caso para o Método de Quine-McCluskey, pode-se notar que em relação a memória utilizada e tempo de execução os métodos propostos foram superiores. Essa diferença se deve pois para a implementação do Método QMM utilizou-se lista de lista e um arquivo para ordenar a entrada, e para as Propostas utilizou-se estrutura de dados árvore e não há geração de arquivo auxiliar para ordenação. Como era de se esperar, ao considerar o

Tabela 26 – Informações sobre Tempo de Execução e Memórias Virtual e Física gastos a partir da execução da Base Tudo Um com a aplicação da Primeira Fase do Método de Quine-McCluskey para Múltiplas Saídas (QMM) e dos Métodos Propostos GPMultiplo (GPM) e MultiGeraPlex (MGP)

Caso	Tempo (μs)			Memória Virtual (*Bytes*)			Memória Física (*Bytes*)		
	QMM	GPM	MGP	QMM	GPM	MGP	QMM	GPM	MGP
$T1_3$	16734	1593	1629	0	0	0	1,58 E+7	5,27 E+6	1,58 E+7
$T1_4$	34312	1658	1710	0	0	0	1,75 E+7	5,27 E+6	1,58 E+7
$T1_5$	35645	1758	1815	0	0	0	1,40 E+7	5,27 E+6	1,58 E+7
$T1_6$	53633	2031	2052	0	0	0	1,93 E+7	5,27 E+6	1,58 E+7
$T1_7$	160091	2531	2693	0	0	0	1,58 E+7	5,27 E+6	1,75 E+7
$T1_8$	1114491	3426	3898	1,32 E+9	0	0	2,00 E+8	5,27 E+6	1,58 E+7
$T1_9$	8723962	5178	6349	4,20 E+9	0	0	2,71 E+9	7,03 E+6	2,63 E+7
$T1_{10}$	76106390	8669	10590	1,54 E+10	0	0	1,41 E+10	1,05 E+7	2,11 E+7
$T1_{11}$	–	16165	20249	–	0	0	–	1,75 E+7	3,16 E+7
$T1_{12}$	–	30253	40216	–	4,59 E+8	4,66 E+8	–	3,87 E+7	4,74 E+7
$T1_{13}$	–	60420	80182	–	8,63 E+8	8,32 E+8	–	1,26 E+8	8,97 E+7
$T1_{14}$	–	120067	162448	–	1,64 E+9	1,63 E+9	–	2,04 E+9	2,15 E+9
$T1_{15}$	–	243914	333413	–	3,37 E+9	3,41 E+9	–	3,90 E+9	3,90 E+9
$T1_{16}$	–	500710	687635	–	6,81 E+9	6,66 E+9	–	7,21 E+9	7,28 E+9
$T1_{17}$	–	1008854	1411716	–	1,37 E+10	1,36 E+10	–	1,42 E+10	1,45 E+10
$T1_{18}$	–	2092888	2880509	–	2,70 E+10	2,65 E+10	–	2,70 E+10	2,65 E+10
$T1_{19}$	–	4178915	5971077	–	5,42 E+10	5,51 E+10	–	5,40 E+10	5,42 E+10
$T1_{20}$	–	8468697	12202319	–	1,09 E+11	1,11 E+11	–	1,05 E+11	1,06 E+11
$T1_{21}$	–	17300151	24973737	–	2,20 E+11	2,19 E+11	–	2,12 E+11	2,11 E+11
$T1_{22}$	–	36304764	52615635	–	4,43 E+11	4,52 E+11	–	4,26 E+11	4,43 E+11

Fonte: Elaborado pelo Autor

Tabela 27 – Resultados obtidos a partir da Execução da Base Xadrez através da Primeira Fase do Método de Quine-McCluskey para Múltiplas Saídas e dos Métodos Propostos GPMultiplo e MultiGeraPlex

Caso	*Bit*	Termo	Comparação		Tempo (μs)			Memória Física (*Bytes*)		
			QMM	GPM/MGP	QMM	GPM	MGP	QMM	GPM	MGP
X_3	3	4	0	6	28783	1629	1596	1,58 E+7	8,79E+6	1,055 E+7
X_4	4	8	0	28	34472	1699	1748	4,92 E+7	8,79E+6	1,05 E+7
X_5	5	16	0	120	36196	1991	2031	1,23 E+7	8,79E+6	1,05 E+7
X_6	6	32	0	496	48812	2825	2872	6,15 E+7	8,79E+6	1,05 E+7
X_7	7	64	0	2016	58076	5869	5966	1,40 E+7	8,79E+6	1,05 E+7
X_8	8	128	0	8128	78856	17992	18265	7,56 E+7	8,79E+6	1,05 E+7
X_9	9	144	0	10296	89722	30014	30628	3,87 E+7	3,69 E+7	3,68 E+7

Fonte: Elaborado pelo Autor

número de comparações, o QMM é superior aos métodos propostos, pois para o QMM não há comparação entre os termos. Mesmo não sendo comparado os termos, no QMM, há uma comparação entre as caixas sequenciais pertencentes ao grupo 0.

Neste trabalho, seis conjuntos de funções com múltiplas saídas foram apresentadas ao longo do texto, na Tabela 28 estão representados os custos relacionados a solução mínima dessas funções. A solução mínima foi obtida através da formulação do problema utilizando PLI 0 e 1, a partir dos implicantes primos gerados pelo QMM, GPM e MGP. Para comparar os custos obtidos, as funções foram executadas no Programa Espresso, e após os resultados (solução mínima) calculou-se os custos destas. A fim de um comparativo e para reforçar a importância de minimizar funções com múltiplas saídas, o custo total (soma dos custos individuais de cada função simplificada de maneira simples) das funções também foi calculado e descrito.

Nas Tabelas 29 e 30 estão representados os resultados obtidos através da aplicação da Primeira Fase do Método de Quine-McCluskey para Funções com Múltiplas Saídas, do GPMultiplo e do MultiGeraPlex para as funções apresentadas ao longo deste trabalho (F_{M_i}, i varia de 1 a 6).

O MultiGeraPlex não gera todos os implicantes primos, sendo esta, uma das características deste método, enquanto os Métodos QMM e GPM geram todos os implicantes primos da função. A partir dos resultados obtidos na Tabela 29 pode-se observar que o número de implicantes gerados pelos QMM quando comparado com as propostas ou foi equivalente ou maior. Em relação ao número de comparações, o QMM apresentou para os casos apresentados maior número de comparações. Ao comparar os resultados das Propostas, pode-se observar que o MGP gerou menos implicantes e menor

Tabela 28 – Custo relacionado a função mínima (minimização simples e múltipla) das funções executadas (F_{M_i}) pelo QMM, GPM, MGP e Espresso

F_{M_i}	Função f_i	Custo Função Mínima				
		$custo_{f_i}$	$custo_{QMM}$	$custo_{GPM}$	$custo_{MGP}$	$custo_{Espresso}$
F_{M_1}	$f_7 = \sum m(0,1,5,10,13,14)$ $f_8 = \sum m(4,5,6,7,13)$ $f_9 = \sum m(5,10,13,14)$	12 7 8 $T = 27$	18	18	18	18
F_{M_2}	$f_{10} = \sum m(0,2,6)$ $f_{11} = \sum m(0,1,3)$	6 6 $T = 12$	11	11	12	11
F_{M_3}	$f_{12} = \sum m(0,1,2,3,6,7)$ $f_{13} = \sum m(6,7,8,9,10,14)$ $f_{14} = \sum m(0,1,2,3,8,9)$	7 12 7 $T = 26$	18	18	22	18
F_{M_4}	$f_{15} = \sum m(4,5,7)$ $f_{16} = \sum m(1,4,5)$ $f_{17} = \sum m(0,2,4)$	6 6 6 18	15	15	16	16
F_{M_5}	$f_{18} = \sum m(0,1,2,5,7)$ $f_{19} = \sum m(0,1,2,4,5,7)$	9 7 $T = 16$	12	12	12	12
F_{M_6}	$f_{15} = \sum m(0,1,2,5)$ $f_{16} = \sum m(4,5,7)$ $f_{17} = \sum m(1,4,5)$	6 6 6 $T = 18$	14	14	14	14

Fonte: Elaborado pelo Autor

Tabela 29 – Número de Implicantes Primos e Implicantes Gerados e Número de Comparações realizadas com a Execução da Primeira Fase do Método de Quine-McCluskey para Múltiplas Saídas (QMM) e dos Métodos Propostos GPMultiplo (GPM) e MultiGeraPlex (MGP) para as Funções com Múltiplas Saídas F_{M_i}

Caso	*Bit*	Termo	Imp. Primo			Implicante			Comparação		
			QMM	GPM	MGP	QMM	GPM	MGP	QMM	GPM	MGP
F_{M_1}	4	9	5	5	5	9	7	7	32	16	17
F_{M_2}	3	5	5	5	4	4	4	4	10	8	6
F_{M_3}	4	10	8	8	8	16	10	10	67	18	18
F_{M_4}	3	6	6	6	5	5	5	5	13	10	10
F_{M_5}	3	6	5	5	5	7	6	6	15	8	8
F_{M_6}	3	6	6	6	5	5	5	5	13	12	10

Fonte: Elaborado pelo Autor

Tabela 30 – Tempo de Execução e Memórias (Virtual e Física) utilizadas com a Execução da Primeira Fase do Método de Quine-McCluskey para Múltiplas Saídas (QMM) e dos Métodos Propostos GPMultiplo (GPM) e MultiGeraPlex (MGP) para as Funções com Múltiplas Saídas F_{M_i}

Caso	Termo	Tempo (μs)			Memória Virtual (*Bytes*)		Memória Física (*Bytes*)		
		QMM	GPM	MGP	QMM	GPM/MGP	QMM	GPM	MGP
FM_1	9	43394	2743	2918	$9,04$ E+11	0	$1,41$ E+10	$1,76$ E+10	$1,94$ E+10
FM_2	5	25912	1647	3120	$4,52$ E+11	0	$1,58$ E+10	$1,76$ E+10	$1,94$ E+10
FM_3	10	39571	1744	3113	0	0	$1,78$ E+11	$1,23$ E+10	$2,81$E+10
FM_4	6	26607	1732	2642	0	0	$1,94$ E+10	$1,58$ E+10	$2,81$ E+10
FM_5	6	25554	2261	2648	0	0	$9,68$ E+10	$1,58$ E+10	$2,64$ E+10
FM_6	6	26686	2646	2389	$4,49$ E+11	0	$4,57$ E+11	$1,76$ E+10	$7,21$ E+10

Fonte: Elaborado pelo Autor

número de comparações para três das funções apresentadas, porém obteve a solução mínima apenas para um desses três casos (Tabelas 28 e 29).

O tempo de execução do QMM, assim como nos outros casos apresentados, também é maior em relação às Propostas, pelo menos 10 vezes maior quando comparado com o tempo de execução dos arquivos pelas Propostas.

Na Tabela 31 são apresentadas as funções utilizadas para compor a Base C_i, que são utilizadas para a execução dos Métodos de QMM, GPM e MGP. Essa base também foi executada com o *software* Espresso.

Nas Tabelas 32 e 33, são apresentados os resultados obtidos com a execução de várias funções com múltiplas saídas pelo Método de QMM, GPM e MGP. O número de implicantes primos e implicantes gerados e o número de comparações realizadas estão contidos na primeira Tabela e o Tempo de Execução e Memória Física apresentados na segunda.

Como pode-se observar, na Tabela 32, os métodos propostos (para a maior parte dos casos) geram menos comparações entre os termos e menos implicantes que o Método de QMM. Além disso, quando analisado o desempenho (Tempo de Execução e Memória Utilizadas) as propostas para os casos apresentados também apresentaram melhor resultado, conforme apresentado na Tabela 33.

A partir dos Implicantes primos obtidos pelos Métodos de QMM, GPM e MGP (com a execução dos arquivos da Base C_i) foi formulada, utilizando PLI 0 e 1, a cobertura dos mintermos, e calculado o custo da função mínima resultante. Os arquivos da Base também foram executados no Espresso. Na Tabela 34 são apresentados os resultados obtidos.

Tabela 31 – Relação de Funções com Múltiplas Saídas Executadas

C_i	f_x
C_1	$f_a = \sum m(2,3,7)$ e $f_b = \sum m(1,2,3)$
C_2	$f_a = \sum m(0,1,2,3)$ e $f_b = \sum m(1,2,3)$
C_3	$f_a = \sum m(1,2,3,5)$ e $f_b = \sum m(1,2,3)$ e $f_c = \sum m(1,2,5)$
C_4	$f_a = \sum m(0,2,6)$ e $f_b = \sum m(0,1,3)$
C_5	$f_a = \sum m(1,3,7)$ e $f_b = \sum m(2,6,7)$
C_6	$f_a = \sum m(0,1,2)$ e $f_b = \sum m(2,6,7)$
C_7	$f_a = \sum m(0,2,3)$ e $f_b = \sum m(0,1,3)$
C_8	$f_a = \sum m(4,5,7)$, $f_b = \sum m(1,4,5)$ e $f_c = \sum m(0,2,4)$
C_9	$f_a = \sum m(0,1,2,5,7)$ e $f_b = \sum m(0,1,2,4,5,7)$
C_{10}	$f_a = \sum m(0,1,2,5)$, $f_b = \sum m(5,7)$ e $f_c = \sum m(1,4,5)$
C_{11}	$f_a = \sum m(2,3,6,7)$ e $f_b = \sum m(0,1,2,3)$
C_{12}	$f_a = \sum m(0,4,7)$, $f_b = \sum m(0,3,4)$ e $f_c = \sum m(3,4,5,6,7)$
C_{13}	$f_a = \sum m(0,1,3,5)$, $f_b = \sum m(2,3,5,6)$ e $f_c = \sum m(0,1,6)$
C_{14}	$f_a = \sum m(0,2,4,6)$ e $f_b = \sum m(0,1,2,3)$
C_{15}	$f_a = \sum m(0,1,5,6,7)$, $f_b = \sum m(0,2,5,6,7)$ e $f_c = \sum m(0,1,6,7)$
C_{16}	$f_a = \sum m(2,3,6,7)$ e $f_b = \sum m(0,1,2,3)$
C_{17}	$f_a = \sum m(0,1,2,5,6,7)$ e $f_b = \sum m(0,1,2,3,5,6,7)$
C_{18}	$f_a = \sum m(2,3,4,6,7)$ e $f_b = \sum m(0,1,2,3,6)$
C_{19}	$f_{24} = \sum m(0,1,2,5,6,7)$ e $f_{22} = \sum m(0,1,2,3,5,6,7)$
C_{20}	$f_a = \sum m(0,2,5,6,13)$, $f_b = \sum m(0,5,11,13,15)$ e $f_c = \sum m(0,4)$
C_{21}	$f_a = \sum m(0,1,3,4,5,6,7)$ e $f_b = \sum m(0,1,2,4,6,7)$
C_{22}	$f_a = \sum m(3,4,7,9,15)$ e $f_b = \sum m(4,5,9,13,14,15)$
C_{23}	$f_a = \sum m(0,1,2,4,5,7)$ e $f_b = \sum m(0,1,2,3,5,6)$
C_{24}	$f_a = \sum m(0,2,4,5,7)$, $f_b = \sum m(1,2,3,5,6)$ e $f_c = \sum m(0,1,2,3,4,5,6)$
C_{25}	$f_a = \sum m(0,1,3,4,5,6,7)$ e $f_b = \sum m(0,1,2,4,6,7)$
C_{26}	$f_a = \sum m(0,1,2,4,5,7)$ e $f_b = \sum m(0,1,2,3,5,6)$
C_{27}	$f_a = \sum m(0,1,5,10,13,15)$, $f_b = \sum m(4,5,6,7,13)$ e $f_c = \sum m(5,10,13,15)$
C_{28}	$f_a = \sum m(0,2,3,5,6,8,9)$, $f_b = \sum m(0,2,4,6,7)$ e $f_c = \sum m(0,2,6,8,9)$
C_{29}	$f_a = \sum m(0,2,5,6,13)$, $f_b = \sum m(0,5,11,13,15)$ e $f_c = \sum m(0,8,12)$
C_{30}	$f_a = \sum m(0,1,5,10,13,14)$, $f_b = \sum m(4,5,6,7,13,14)$ e $f_c = \sum m(5,10,13,14)$
C_{31}	$f_a = \sum m(0,1,2,3,6,7)$, $f_b = \sum m(0,1,6,7,14,15)$ e $f_c = \sum m(0,1,2,3,8,9)$
C_{32}	$f_a = \sum m(0,1,3,4,5,7,11,15)$ e $f_b = \sum m(2,3,6,7,11,15)$
C_{33}	$f_a = \sum m(0,4,5,8,12,13)$ e $f_b = \sum m(3,5,7,11,13,15)$
C_{34}	$f_a = \sum m(0,1,4,5,6,7)$ e $f_b = \sum m(0,1,3,4,7,12,13,15)$
C_{35}	$f_a = \sum m(0,1,2,3,6,7)$, $f_b = \sum m(0,1,6,7,14,15)$ e $f_c = \sum m(0,1,2,3,8,9)$
C_{36}	$f_a = \sum m(0,2,5,6,7,8,9,12,13,15)$ e $f_b = \sum m(2,6,7,8,13)$
C_{37}	$f_a = \sum m(0,1,2,3,5)$, $f_b = \sum m(8,9,10,11,13)$ e $f_c = \sum m(0,1,2,3,5,6,8,9,10,11,13)$
C_{38}	$f_a = \sum m(0,4,5,8,12,13)$ e $f_b = \sum m(1,2,3,5,6,8,13,14,15)$
C_{39}	$f_a = \sum m(2,6,7,10,14,15)$ e $f_b = \sum m(4,5,7,8,9,11,12,15)$
C_{40}	$f_a = \sum m(0,4,5,8,12,13,15)$ e $f_b = \sum m(1,2,3,5,6,13,14)$
C_{41}	$f_a = \sum m(0,4,5,8,12,13)$ e $f_b = \sum m(1,2,3,5,6,8,13,14,15)$
C_{42}	$f_a = \sum m(0,2,4,5,6,7,8,12,13,15)$ e $f_b = \sum m(2,3,5,6,7,9,11,13,15)$
C_{43}	$f_a = \sum m(0,1,2,3,5,6,7,8,9,11,12,13,14)$ e $f_b = \sum m(0,1,3,4,5,6,9,12,13,14)$
C_{44}	$f_a = \sum m(1,2,3,4,6,7,8,9,10,11,12,13,15)$ e $f_b = \sum m(3,4,6,7,9,10,11,12,14,15)$
C_{45}	$f_a = \sum m(0,1,2,3,5,6,7,8,9,11,12,13,14)$ e $f_b = \sum m(0,1,3,4,5,6,9,12,13,14)$
C_{46}	$f_a = \sum m(0,1,2,3,8,9,10,11)$, $f_b = \sum m(8,9,10,12,13,14)$ e $f_c = \sum m(4,5,6,7)$
C_{47}	$f_a = \sum m(0,2,6,10,11,14,15)$, $f_b = \sum m(0,3,6,7,8,9,12,13,14,15)$ e $f_c = \sum m(0,3,4,5,7,10,11,12,13,14,15)$
C_{48}	$f_a = \sum m(0,1,4,5,6,7,8,9,10,11,12,14,15)$ e $f_b = \sum m(0,1,2,3,4,5,6,10,11,12,13)$

Fonte: Elaborado pelo Autor

Tabela 32 – Número de Implicantes Primos e Implicantes Gerados, e Número de Comparações realizadas com a Execução da Primeira Fase do Método de Quine-McCluskey para Funções com Múltiplas Saídas (QMM) e dos Métodos Propostos GPMultiplo (GPM) e MultiGeraPlex (MGP) para Funções com Múltiplas Saídas (C_i)

Caso	*Bit*	Termo	Imp. Primo			Implicante			Comparação		
			QMM	GPM	MGP	QMM	GPM	MGP	QMM	GPM	MGP
C_1	3	4	3	3	3	3	3	3	5	4	4
C_2	2	4	3	3	3	5	4	4	8	4	4
C_3	3	4	5	5	5	3	3	3	4	4	5
C_4	3	5	5	5	4	4	4	4	10	8	6
C_5	3	5	5	5	4	4	4	4	10	7	7
C_6	3	5	5	5	4	4	4	4	8	6	6
C_7	3	5	5	5	4	4	4	4	10	8	6
C_8	3	6	6	6	5	5	5	5	13	10	10
C_9	3	6	5	5	5	7	6	6	15	8	8
C_{10}	3	6	6	6	5	5	5	5	13	13	11
C_{11}	3	6	3	3	2	9	5	5	21	6	6
C_{12}	3	6	6	6	5	7	6	5	15	11	9
C_{13}	3	6	8	8	7	5	5	5	12	13	11
C_{14}	3	6	3	3	2	9	6	6	21	6	6
C_{15}	3	6	7	7	6	6	6	6	16	10	8
C_{16}	3	6	3	3	2	9	5	5	21	6	6
C_{17}	3	7	9	9	9	12	10	10	33	12	12
C_{18}	3	7	5	5	4	10	7	7	28	11	11
C_{19}	3	7	9	9	9	12	10	10	33	12	12
C_{20}	4	8	7	7	7	6	6	6	17	26	27
C_{21}	3	8	8	8	6	15	11	11	49	14	13
C_{22}	4	8	10	10	9	7	7	7	27	26	24
C_{23}	3	8	7	7	6	11	9	9	35	16	16
C_{24}	3	8	13	13	11	13	13	13	39	32	27
C_{25}	3	8	8	8	6	15	11	11	49	14	13
C_{26}	3	8	7	7	6	11	9	9	35	16	16
C_{27}	4	9	6	6	6	9	7	7	31	19	21
C_{28}	4	9	8	8	8	9	8	8	35	20	22
C_{29}	4	9	8	8	8	7	7	7	24	28	29
C_{30}	4	9	7	7	6	10	8	8	35	19	20
C_{31}	4	10	6	6	5	17	9	9	59	14	15
C_{32}	4	10	4	4	4	17	12	11	62	17	15
C_{33}	4	10	5	5	4	17	13	12	59	21	18
C_{34}	4	10	11	11	10	15	13	12	59	32	26
C_{35}	4	10	6	6	5	17	9	9	59	14	15
C_{36}	4	10	8	8	8	14	12	11	52	33	29
C_{37}	4	11	7	7	7	24	13	11	104	16	16
C_{38}	4	12	11	11	11	16	15	14	72	50	42
C_{39}	4	12	10	10	10	16	14	14	79	46	39
C_{40}	4	12	9	9	9	15	13	13	69	41	38
C_{41}	4	12	11	11	11	16	15	14	72	50	42
C_{42}	4	13	10	10	8	28	21	19	141	39	31
C_{43}	4	14	15	15	14	34	29	25	205	69	41
C_{44}	4	14	15	15	14	34	31	29	208	83	64
C_{45}	4	14	15	15	14	34	29	25	205	69	41
C_{46}	4	15	6	6	4	31	18	15	178	24	18
C_{47}	4	15	16	16	13	32	24	23	192	63	49
C_{48}	4	16	16	16	13	42	27	23	296	49	39

Fonte: Elaborado pelo Autor

Tabela 33 – Tempo de Execução e Memórias Física utilizada com a Execução da Primeira Fase do Método de Quine-McCluskey para Funções com Múltiplas Saídas (QMM) e dos Métodos Propostos GPMultiplo (GPM) e MultiGeraPlex (MGP) para Funções com Múltiplas Saídas (C_i)

Caso	*Bit*	Termo	Tempo (μs)			Memória Física (*Bytes*)		
			QMM	GPM	MGP	QMM	GPM	MGP
C_1	3	4	29803	2142	2267	4,74E+10	1,05E+10	1,05E+10
C_2	2	4	38249	3142	2336	3,51E+9	5,27E+9	1,05E+10
C_3	3	4	79001	2694	2639	3,51E+9	8,79E+9	1,05E+10
C_4	3	5	31606	2871	2636	2,99E+10	8,79E+9	1,05E+10
C_5	3	5	42699	1834	2346	8,79E+9	7,03E+9	8,79E+9
C_6	3	5	37817	2714	2654	1,75E+9	8,79E+9	1,05E+10
C_7	3	5	38659	2745	2971	3,51E+9	8,79E+9	1,05E+10
C_8	3	6	37630	2821	2833	2,81E+10	7,03E+9	1,05E+10
C_9	3	6	38670	2684	2830	4,57E+10	8,79E+9	1,05E+10
C_{10}	3	6	30697	3202	2763	1,26E+11	8,79E+9	1,05E+10
C_{11}	3	6	31382	2083	1760	5,27E+9	8,79E+9	7,03E+10
C_{12}	3	6	59055	2792	2786	1,75E+9	8,79E+9	1,05E+10
C_{13}	3	6	50620	3052	80863	1,58E+10	8,79E+9	1,05E+10
C_{14}	3	6	86833	2400	2341	3,51E+9	7,03E+9	7,03E+10
C_{15}	3	6	24589	1607	1670	8,79E+9	1,58E+10	1,93E+10
C_{16}	3	6	24244	1599	1585	1,58E+10	2,11E+10	1,58E+10
C_{17}	3	7	24477	1646	1719	5,27E+9	7,03E+9	8,79E+10
C_{18}	3	7	38502	2687	2455	3,51E+9	8,79E+9	8,79E+10
C_{19}	3	7	38153	1827	1981	1,40E+10	1,93E+10	1,93E+10
C_{20}	4	8	33057	3509	3050	5,27E+9	8,79E+9	1,05E+10
C_{21}	3	8	11590	3138	2953	3,51E+9	8,79E+9	1,05E+10
C_{22}	4	8	39080	3895	3395	5,27E+9	8,79E+9	1,05E+10
C_{23}	3	8	19262	2953	2903	3,51E+9	8,79E+9	1,05E+10
C_{24}	3	8	45525	3895	3535	1,75E+9	8,79E+9	1,05E+10
C_{25}	3	8	38994	1877	2881	1,40E+10	1,75E+10	1,93E+10
C_{26}	3	8	39044	1863	3273	1,05E+10	1,75E+10	1,93E+10
C_{27}	4	9	48767	2743	2761	7,03E+9	7,03E+9	1,05E+10
C_{28}	4	9	49101	3168	3280	4,04E+10	8,79E+9	1,05E+10
C_{29}	4	9	39720	3224	3071	3,51E+9	8,79E+9	1,40E+10
C_{30}	4	9	127037	2699	1993	2,99E+10	1,75E+10	1,93E+10
C_{31}	4	10	39016	2737	3922	1,75E+9	8,79E+9	8,79E+10
C_{32}	4	10	42555	1934	1930	6,68E+10	8,79E+9	1,05E+10
C_{33}	4	10	48416	3384	2793	3,51E+9	8,79E+9	1,05E+10
C_{34}	4	10	55995	3637	3573	5,27E+9	8,79E+9	1,05E+10
C_{35}	4	10	19364	1862	1917	1,40E+10	1,75E+10	1,93E+10
C_{36}	4	10	55044	1953	1900	2,11E+10	1,75E+10	1,93E+10
C_{37}	4	11	50762	2775	2009	2,81E+10	1,75E+10	1,93E+10
C_{38}	4	12	20628	87307	3754	3,51E+9	8,79E+9	1,05E+10
C_{39}	4	12	47156	1974	2118	3,16E+10	1,75E+10	1,9E+10
C_{40}	4	12	53394	2001	2062	1,58E+10	1,75E+10	1,93E+10
C_{41}	4	12	116897	1982	4001	1,75E+10	1,75E+10	2,81E+10
C_{42}	4	13	46185	2785	4030	1,58E+10	1,75E+10	1,93E+10
C_{43}	4	14	19992	4873	4182	3,51E+9	8,79E+9	1,05E+10
C_{44}	4	14	46532	2165	2170	2,28E+10	1,75E+10	1,93E+10
C_{45}	4	14	33449	2084	2130	1,75E+10	1,75E+10	1,93E+10
C_{46}	4	15	50413	2989	3142	3,51E+9	8,79E+9	1,05E+10
C_{47}	4	15	42224	2030	4328	1,75E+10	1,75E+10	1,93E+10
C_{48}	4	16	46122	6578	4530	3,51E+10	8,79E+9	1,05E+10

Fonte: Elaborado pelo Autor

Tabela 34 – Custo relacionado a função mínima (minimização simples e múltipla) das funções executadas (C_i) pelo QMM, GPM, MGP e Espresso

C_i	Custo Função Mínima				
	$custoT_{Simples}$	$custo_{QMM}$	$custo_{GPM}$	$custo_{MGP}$	$custo_{Espresso}$
C_1	12	10	10	10	10
C_2	2	2	2	2	2
C_3	21	16	16	16	17
C_4	12	11	11	12	11
C_5	12	11	11	12	11
C_6	12	11	11	12	11
C_7	12	11	11	12	11
C_8	18	15	15	16	15
C_9	16	12	12	12	12
C_{10}	15	13	13	13	13
C_{11}	2	2	2	2	2
C_{12}	18	14	14	16	14
C_{13}	26	20	20	24	20
C_{14}	2	2	2	2	2
C_{15}	24	16	16	16	16
C_{16}	2	2	2	2	2
C_{17}	12	12	12	12	12
C_{18}	8	8	8	8	8
C_{19}	12	12	12	12	12
C_{20}	29	23	23	26	23
C_{21}	10	10	10	10	10
C_{22}	30	28	28	30	28
C_{23}	14	14	14	14	14
C_{24}	21	21	21	21	22
C_{25}	10	10	10	10	10
C_{26}	14	14	14	14	14
C_{27}	37	24	24	24	24
C_{28}	45	31	31	31	31
C_{29}	33	24	24	30	24
C_{30}	31	22	22	22	22
C_{31}	19	17	17	17	17
C_{32}	12	10	10	10	10
C_{33}	12	11	11	12	11
C_{34}	22	20	20	22	20
C_{35}	19	17	17	17	17
C_{36}	32	29	29	29	29
C_{37}	22	22	22	22	22
C_{38}	27	25	25	27	27
C_{39}	22	20	20	22	22
C_{40}	26	24	24	26	24
C_{41}	27	25	25	27	27
C_{42}	18	16	16	16	18
C_{43}	29	27	27	27	27
C_{44}	32	27	27	27	27
C_{45}	32	27	27	27	27
C_{46}	10	10	10	10	10
C_{47}	37	28	28	28	28
C_{48}	27	25	25	25	26

Fonte: Elaborado pelo Autor

6 CONCLUSÕES E TRABALHOS FUTUROS

A implementação de várias funções booleanas, que contenham as mesmas variáveis, é muito importante quando se fala em economia relativa ao menor termo produto e menos portas, assim como acontece em funções booleanas simples. Porém, ao trabalhar com múltiplas funções, pode-se reduzir ainda mais o custo decorrente do uso comum de um termo produto por mais de uma função analisada.

Um breve histórico de trabalhos na área de minimização de funções booleanas foi realizado, com o intuito de contextualizar o assunto e mostrar a importância desse tema para a elaboração de circuitos mais otimizados e menores.

Dois algoritmos para trabalhar com funções com múltiplas saídas foram desenvolvidos, GPMultiplo e MultiGeraPlex; ambos baseados na filosofia do algoritmo GeraPlex, o primeiro algoritmo proposto gera todos os implicantes primos, enquanto o outro não.

Ao analisar o número de comparações realizadas pelos métodos (QMM, GPM e MGP), pode-se concluir que os métodos propostos obtiveram melhores resultados em 97% dos casos apresentados, em relação ao QMM (não considerando a Base Xadrez). Ao considerar o tempo de execução, para todos os casos apresentados, as propostas apresentaram melhores resultados que o QMM.

Pode-se concluir que a estrutura de árvore ternária, utilizada nos algoritmos propostos e desenvolvidos neste trabalho, é uma opção interessante para armazenamento e comparações entre os termos. Visto que, com a forma de representação dos termos em árvore, o espaço de armazenamento utilizado é menor ao comparado com o armazenamento do QMM.

Para que os métodos propostos possam ser utilizados de forma mais completa, as duas fases da minimização são necessárias. Por isso a implementação da cobertura dos mintermos ou de uma saída que integre diretamente a outro método que realize essa cobertura é um tópico relevante a ser considerado para trabalhos futuros.

Em se tratando de funções com múltiplas saídas, e principalmente com o advento de

lógica reversível, o estudo e aplicação da minimização de funções booleanas reversíveis é de grande importância.

Para trabalhos futuros pretende-se aprimorar e desenvolver um método, baseado nos métodos propostos, para funções reversíveis.

REFERÊNCIAS

AGRAWAL, P.; AGRAWAL, V. D.; BISWAS, N. N. Multiple output minimization. In: DESIGN AUTOMATION CONFERENCE, 22., 1985, Las Vegas. **Conference...** United States: IEEE Press, 1985. p. 674–680.

AKERS, S. B. Binary decision diagrams. **IEEE Transactions on Computers**, Washington, DC, USA, C–27, n. 6, p. 509–516, 1978.

AL-SHIHABI, S. A hybrid of max-min ant system and linear programming for the k-covering problem. **Computer & Operations Research**, Elsevier Science Ltd., Oxford, UK, UK, v. 76, n. C, p. 1–11, 2016.

AL-SHIHABI, S.; ARAFEH, M.; BARGHASH, M. An improved hybrid algorithm for the set covering problem. **Computers & Industrial Engineering**, Pergamon Press, Tarrytown, NY, USA, v. 85, n. C, p. 328 – 334, 2015.

ALEXE, G. et al. Consensus algorithms for the generation of all maximal bicliques. **Discrete Applied Mathematics**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 145, n. 1, p. 11–21, 2004.

ASTHANA, R.; VERMA, N.; RATAN, R. Generation of boolean functions using genetic algorithm for cryptographic applications. In: ADVANCE COMPUTING, 2014, Gurgaon. **Conference...** India: IEEE, 2014. p. 1361–1366.

BAHAR, A. N.; WAHEED, S.; HABIB, M. A. A novel presentation of reversible logic gate in Quantum-dot Cellular Automata (QCA). In: ELECTRICAL ENGINEERING AND INFORMATION COMMUNICATION TECHNOLOGY, 2014, Dhaka. **Conference...** Bangladesh: IEEE, 2014. p. 1–6.

BAJPAYEE, K. et al. Mining frequent itemset using Quine–McCluskey algorithm. In: SOFT COMPUTING FOR PROBLEM SOLVING, 5., 2016, Singapore. **Conference...** Singapore: Springer Singapore, 2016. p. 763–769.

BANSAL, M.; AGARWAL, A. Genetic algorithm for ordering and reduction of BDDs for MIMO circuits. In: INNOVATIVE COMPUTING TECHNOLOGY, 3., 2013, London. **Conference...** United Kingdom: IEEE, 2013. p. 411–414.

BARTEE, T. C. Computer design of multiple-output logical networks. **IRE Transactions on Electronic Computers**, IEEE, United States, EC-10, n. 1, p. 21–30, mar 1961.

BENEDIKTSSON, J. A.; SWAIN, P. H. Consensus theoretic classification methods. **IEEE Transactions on Systems, Man and Cybernetics**, IEEE, United States, v. 22, n. 4, p. 688–704, 1992.

BOUTE, R. The binary decision machine as programmable controller. **Euromicro Newsletter**, Netherlands, v. 2, n. 1, p. 16 – 22, 1976.

BRAYTON, R. K. **Logic minimization algorithms for VLSI synthesis**. United States: Springer Science & Business Media, 1984. v. 2. 54–138 p.

BRAYTON, R. K.; KHATRI, S. P. Multi-valued logic synthesis. In: VLSI DESIGN - VLSI FOR THE INFORMATION APPLIANCE, 12., 1999, Goa. **Conference...** United States: IEEE Computer Society, 1999. p. 196–205.

BRYANT, R. E. Graph-based algorithms for boolean function manipulation. **IEEE Transactions on computers**, IEEE, United States, v. 100, n. 8, p. 677–691, 1986.

CERNY, E.; MANGE, D.; SANCHEZ, E. Synthesis of minimal binary decision trees. **IEEE Transactions on Computers**, United States, v. 7, n. C-28, p. 472–482, 1979.

CHEN, Y.; HUO, B.; CHEN, X. Consensus method for group decision making based on multigranularity interval linguistic. In: INTELLIGENT SYSTEMS AND APPLICATIONS, 2009, Wuhan. **Workshop...** China: IEEE, 2009. p. 1–4.

CHOWDHURY, A. R.; NAZMUL, R.; BABU, H. M. H. A new approach to synthesize multiple-output functions using reversible programmable logic array. In: VLSI DESIGN, 19., 2006, Hyderabad. **Conference...** United States: IEEE Computer Society, 2006. p. 311–316.

CORMEN, T. H. et al. **Introduction to Algorithms**. Third. [S.l.]: The MIT Press, 2009. ISBN 9788560031931.

COUDERT, O. Two-level logic minimization: an overview. **Integration, the VLSI journal**, Elsevier, Netherlands, v. 17, n. 2, p. 97–140, 1994.

CRAWFORD, B. et al. Binary bat algorithms for the set covering problem. In: IBERIAN CONFERENCE ON INFORMATION SYSTEMS AND TECHNOLOGIES, 10., 2015, Aveiro. **Conference...** United States: IEEE, 2015. p. 1–4.

CRAWFORD, B. et al. A binary firefly algorithm for the set covering problem. In: MODERN TRENDS AND TECHNIQUES IN COMPUTER SCIENCE, 3., 2014. **Online Conference...** Switzerland: Springer, 2014. p. 65–73.

DAGENAIS, M.; AGARWAL, V. K.; RUMIN, N. C. The mcboole logic minimizer. In: DESIGN AUTOMATION, 22., 1985, Las Vegas. **Conference...** United States: IEEE, 1985. p. 667–673.

DAGENAIS, M. R.; AGARWAL, V. K.; RUMIN, N. C. Mcboole: a new procedure for exact logic minimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, United States, v. 5, n. 1, p. 229–238, January 1986.

DAHI, Z. A. E. M.; MEZIOUD, C.; DRAA, A. Binary bat algorithm: on the efficiency of mapping functions when handling binary problems using continuous-variable-based metaheuristics. In: COMPUTER SCIENCE AND ITS APPLICATIONS, 5., 2015, Algeria. **Conference...** Algeria: Springer, 2015. p. 3–14.

DANTZIG, G. B. et al. The generalized simplex method for minimizing a linear form under linear inequality restraints. **Pacific Journal of Mathematics**, United States, v. 5, n. 2, p. 183–195, 1955.

DAS, S. R. Comments on "a new algorithm for generating prime implicants". **IEEE Transactions on Computers**, IEEE, United States, v. 100, n. 12, p. 1614–1615, 1971.

DAS, S. R.; CHOUDHURY, A. K. Maxterm type expressions of switching functions and their prime implicants. **IEEE Transactions on Electronic Computers**, IEEE, United States, EC-14, n. 6, p. 920–923, Dec 1965.

DAS, S. R.; KHABRA, N. S. Clause-column table approach for generating all the prime implicants of switching functions. **IEEE Transaction Computation**, IEEE, United States, v. 21, n. 11, p. 1239–1246, 1972.

DEB, A. et al. An efficient reduction of common control lines for reversible circuit optimization. In: MULTIPLE-VALUED LOGIC, 2015, Waterloo. **Symposium...** United States: IEEE Computer Society, 2015. p. 14–19.

DRECHSLER, R.; WILLE, R. Synthesis of reversible circuits using decision diagrams. In: INTERNATIONAL SYMPOSIUM ON ELECTRONIC SYSTEM DESIGN, 2012, Kokalta. **Symposium...** United States: IEEE Computer Society, 2012. p. 1–5.

FEN, L. et al. Research on technology of mini-terms optimization for logic function. In: WORLD CONGRESS ON SOFTWARE ENGINEERING, 2009, Xiamen. **Congress...** United States: IEEE Computer Society, 2009. v. 2, p. 451–455.

FISER, P.; HLAVICKA, J. Implicant expansion methods used in the BOOM minimizer. In: DESIGN AND DIAGNOSTICS OF ELECTRONIC CIRCUITS AND SYSTEMS, 2001, Györ. **Workshop...** United States: IEEE, 2001. p. 291–298.

FISER, P.; HLAVICKA, J.; KUBATOVA, H. FC-Min: a fast multi-output boolean minimizer. In: EUROMICRO SYMPOSIUM ON DIGITAL SYSTEM DESIGN, 2003, Belek. **Symposium...** United States: IEEE Computer Society, 2003. p. 451–454.

FISER, P.; RUCKY, P.; VANOVA, I. Fast boolean minimizer for completely specified functions. In: DESIGN AND DIAGNOSTICS OF ELECTRONIC CIRCUITS AND SYSTEMS, 11., 2008, Bratislava. **Workshop...** United States: IEEE Computer Society, 2008. p. 1–6.

FISER, P.; TOMAN, D. A fast SOP minimizer for logic funcions described by many product terms. In: EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN, ARCHITECTURES, METHODS AND TOOLS, 12., 2009, Patras. **Conference...** United States: IEEE Computer Society, 2009. p. 757–764.

FLOYD, T. **Sistemas digitais**: fundamentos e aplicações. [S.l.]: Bookman Companhia, 2007. ISBN 9788560031931.

FRIEDMAN, S. J.; SUPOWIT, K. J. Finding the optimal variable ordering for binary decision diagrams. In: DESIGN AUTOMATION, 24., 1987, Miami Beach. **Conference...** United States: ACM, 1987. p. 348–356.

GURUNATH, B.; BISWAS, N. N. An algorithm for multiple output minimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, Canada, v. 8, n. 9, p. 1007–1013, 1989.

HELAL, A.; BHARGAVA, B. Performance evaluation of the quorum consensus replication method. In: INTERNATIONAL COMPUTER PERFORMANCE AND DEPENDABILITY, 1995, Erlangen. **Symposium...** United States: IEEE Computer Society, 1995. p. 165–172.

HLAVICKA, J.; FISER, P. Boom - a heuristic boolean minimizer. In: COMPUTER AIDED DESIGN, 2001, San Jose. **Conference...** United States: IEEE Press, 2001. p. 439–442.

HULME, B. L.; WORRELL, R. B. A prime implicant algorithm with factoring. **IEEE Transactions on Computers**, IEEE Computer Society, United States, C–24, n. 11, p. 1129–1131, Nov 1975.

HUNG, W. N. et al. Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, United States, v. 25, n. 9, p. 1652–1663, 2006.

ISHIURA, N.; SAWADA, H.; YAJIMA, S. Minimazation of binary decision diagrams based on exchanges of variables. In: COMPUTER-AIDED DESIGN, 1991, Santa Clara. **Conference...** United States: IEEE, 1991. v. 91, p. 472–475.

JIANLIN, Q. et al. Logic functions minimization algorithm based on recognition of essential prime implicants. In: NATURAL COMPUTATION, 6., 2010, Yantai. **Conference...** United States: IEEE, 2010. v. 1, p. 367–371.

JIANLIN, Q. et al. Minimization algorithm of unate logic functions. In: COMPUTER AND INFORMATION TECHNOLOGY, 10., 2010, Bradford. **Conference...** United States: IEEE Computer Society, 2010. p. 2940–2945.

KARNAUGH, M. The map method for synthesis of combinational logic circuits. **Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics**, IEEE, United States, v. 72, n. 5, p. 593–599, Nov 1953.

KAZIMIROV, A. S.; REIMEROV, S. Y. On genetic algorithms and neural networks for boolean functions minimization. In: SOFT COMPUTING AND MEASUREMENTS, 19., 2016, Saint Petersburg. **Conference...** United States: IEEE, 2016. p. 260–261.

KEAN, A.; TSIKNIS, G. An incremental method for generating prime implicants/implicates. **Journal of Symbolic Computation**, Elsevier, United States, v. 9, n. 2, p. 185–206, 1990.

KUO, Y.-S.; CHOU, W. Generating essential primes for a boolean function with multiple-valued inputs. In: DESIGN AUTOMATION CONFERENCE, 23., 1986, Las Vegas. **Conference...** United States: IEEE Computer Society, 1986. p. 193–199.

LEE, C.-Y. Representation of switching circuits by binary-decision programs. **Bell System Technical Journal**, Blackwell Publishing Ltd, United States, v. 38, n. 4, p. 985–999, 1959.

LUBA, T. Decomposition of multiple-valued functions. In: MULTIPLE-VALUED LOGIC, 25., 1995, Bloomington. **Symposium...** United States: IEEE Computer Society, 1995. p. 256–261.

MARQUIS, P. Knowledge compilation using theory prime implicates. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 14., 1995, Montreal. **Conference...** United States: Morgan Kaufmann Publishers Inc., 1995. p. 837–843.

MCCLUSKEY, E. J. Minimization of boolean functions. **The Bell System Technical Journal**, Blackwell Publishing Ltd, United States, v. 35, n. 6, p. 1417–1444, 1956.

MCCLUSKEY, E. J. Minimal sums for boolean functions having many unspecified fundamental products. In: SWITCHING CIRCUIT THEORY AND LOGICAL DESIGN, 1961, Chicago. **Symposium...** United States: IEEE Computer Society, 1961. p. 10–17.

MCCLUSKEY, E. J. **Logic design principles**: with emphasis on testable semicustom circuits. [S.l.]: Prentice Hall, 1986. ISBN 978-0135397848.

MENDELSON, E. **Theory and problems of boolean algebra and switching circuits**. [S.l.]: McGraw-Hill Book Company, 1970. (Shaum's outline series). ISBN 0070414602.

MILLER, D. M.; MASLOV, D.; DUECK, G. W. A transformation based algorithm for reversible logic synthesis. In: DESIGN AUTOMATION, 40., 2003, Anaheim. **Conference...** United States: ACM, 2003. p. 318–323.

MINATO, S.; ISHIURA, N.; YAJIMA, S. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In: DESIGN AUTOMATION, 27., 1990, Orlando. **Conference...** United States: ACM/IEEE, 1990. p. 52–57.

MORRISON, M.; RANGANATHAN, N. A novel optimization method for reversible logic circuit minimization. In: ANNUAL SYMPOSIUM ON VLSI, 2013, Natal. **Symposium...** United States: IEEE Computer Society, 2013. p. 182–187.

MOTT, T. H. Determination of the irredundant normal forms of a truth function by iterated consensus of the prime implicants. **IRE Transactions on Electronic Computers**, IEEE, United States, n. 2, p. 245–252, 1960.

NELSON, R. J. Simplest normal truth functions. **The Journal of Symbolic Logic**, Cambridge University Press, United States, v. 20, n. 2, p. 105–108, 1955.

NELSON, R. J. Weak simplest normal truth functions. **The Journal of Symbolic Logic**, Cambridge University Press, United States, v. 20, n. 03, p. 232–234, 1955.

OGUNBIYI, E. I.; HENLEY, E. J. Irredundant forms and prime implicants of a function with multistate variables. **IEEE Transactions on Reliability**, IEEE Reliability Society, United States, v. 30, n. 1, p. 39–42, 1981.

PERKINS, S. R.; RHYNE, T. An algorithm for identifying and selecting the primed implicants of a multiple-output boolean function. **IEEE transactions on computer-aided design of integrated circuits and systems**, Canada, v. 7, n. 11, p. 1215–1218, 1988.

QUINE, W. V. The problem of simplifying truth functions. **The American Mathematical Monthly**, Mathematical Association of America, United States, v. 59, n. 8, p. 521–531, 1952.

QUINE, W. V. A way to simplify truth functions. **The American Mathematical Monthly**, Mathematical Association of America, United States, v. 62, n. 9, p. 627–631, 1955.

RAWSKI, M.; SZOTKOWSKI, P. Reversible logic synthesis of boolean functions using functional decomposition. In: MIXED DESIGN OF INTEGRATED CIRCUITS & SYSTEMS, 22., 2015, Torun. **Conference...** Poland: IEEE, 2015. p. 380–385.

RHYNE, V. T. et al. A new technique for the fast minimization of switching functions. **IEEE Transactions on Computers**, IEEE, United States, C–26, n. 8, p. 757–764, Aug 1977.

RUDELL, R. L.; SANGIOVANNI-VINCENTELLI, A. Multiple-valued minimization for PLA optimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, United States, v. 6, n. 5, p. 727–750, 1987.

SCHEINMAN, A. H. A method for simplifying boolean functions. **The Bell System Technical Journal**, Wiley Online Library, United States, v. 41, n. 4, p. 1337–1346, July 1962.

SHIRINZADEH, S.; SOEKEN, M.; DRECHSLER, R. Multi-objective BDD optimization with evolutionary algorithms. In: GENETIC AND EVOLUTIONARY COMPUTATION, 2015, Madrid. **Conference...** United States: ACM, 2015. p. 751–758.

SILVA, A. C. R. da. **Contribuição a minimização e simulação de circuitos lógicos**. 1989. 138 p. Dissertação (Mestrado em Engenharia Elétrica) — Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, UNICAMP, Campinas, 1989.

SILVA, A. C. R. da. **Contribuição a síntese de circuitos digitais utilizando programação linear inteira 0 e 1.** 1993. 119 p. Tese (Doutorado em Engenharia Elétrica) — Faculdade de Engenharia Elétrica e de Computaçãoo, Universidade Estadual de Campinas, UNICAMP, Campinas, 1993.

SLAGLE, J. R.; CHANG, C.-L.; LEE, R. C. A new algorithm for generating prime implicants. **IEEE transactions on Computers**, IEEE, United States, v. 100, n. 4, p. 304–310, 1970.

SOTO, R. et al. The set covering problem solved by the black hole algorithm. In: INFORMATION SYSTEMS AND TECHNOLOGIES, 11., 2016, Gran Canaria. **Conference...** Spain: IEEE, 2016. p. 1–4.

STRZEMECKI, T. Polynomial-time algorithms for generation of prime implicants. **Journal of Complexity**, Academic Press, United States, v. 8, n. 1, p. 37–63, 1992.

THAYSE, A.; DAVIO, M.; DESCHAMPS, J. P. Optimization of multivalued decision algorithms. In: MULTIPLE-VALUED LOGIC, 8., 1978, Rosemont. **Symposium...** United States: IEEE Computer Society Press, 1978. p. 171–178.

TISON, P. Generalization of consensus theory and application to the minimization of boolean functions. **IEEE Transactions on Electronic Computers**, IEEE, United States, EC–16, n. 4, p. 446–456, 1967.

TODA, T. Dualization of boolean functions using ternary decision diagrams. **Annals of Mathematics and Artificial Intelligence**, Springer, Switzerland, p. 1–16, 2016.

WILLE, R.; LYE, A.; NIEMANN, P. Checking reversibility of boolean functions. In: REVERSIBLE COMPUTATION, 8., 2016, Bologna. **Conference...** Germany: Springer International Publishing, 2016. p. 322–337.

WILLE, R. et al. Circuit line minimization in the HDL-based synthesis of reversible logic. In: ANNUAL SYMPOSIUM ON VLSI, 2012, Amherst. **Symposium...** United States: IEEE, 2012. p. 213–218.