UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”

FACULDADE DE ENGENHARIA

CAMPUS DE ILHA SOLTEIRA

EVANDRO CATELANI FERRAZ

# COMBINAÇÃO DE FUNÇÕES PRIMITIVAS PARA SÍNTESE DE FUNÇÕES MAJORITÁRIAS

Ilha Solteira

2018

EVANDRO CATELANI FERRAZ

# COMBINAÇÃO DE FUNÇÕES PRIMITIVAS PARA SÍNTESE DE FUNÇÕES MAJORITÁRIAS

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
2018

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

F381c

Ferraz, Evandro Catelani .
Combinação de funções primitivas para síntese de funções majoritárias / Evandro Catelani Ferraz. -- Ilha Solteira: [s.n.], 2018
65 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia. Área de conhecimento: Automação, 2018

Orientador: Alexandre César Rodrigues da Silva

Inclui bibliografia

1. Circuitos digitais. 2. Lógica majoritária. 3. Síntese. 4. Funções primitivas.

Sandra Maria Clemente de Souza
DSTBD/STRAUD
Bibliotecária
CRB 8-4740

UNIVERSIDADE ESTADUAL PAULISTA

**Câmpus de Ilha Solteira**

## CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Combinação de Funções Primitivas para Síntese de Funções Majoritárias

AUTOR: EVANDRO CATELANI FERRAZ
ORIENTADOR: ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovado como parte das exigências para obtenção do Título de Mestre 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

Prof. Dr. RICARDO TOKIO HIGUTI
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. ANDRE INACIO REIS
Instituto de Informática / Universidade Federa do Rio Grande do Sul

Ilha Solteira, 21 de dezembro de 2018

# AGRADECIMENTOS

Aos meus familiares, pela empatia e por apoiar minhas decisões.

Ao meu orientador, Professor Dr. Alexandre César Rodrigues da Silva, por ter possibilitado o desenvolvimento deste trabalho sob seus ensinamentos.

A todos meus amigos, pela cooperação e momentos de descontração. Em especial aos amigos do Laboratório de Processamento de Sinais e Sistemas Digitais.

A todos que contribuíram indiretamente para a realização deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

# RESUMO

Devido ao grande avanço da tecnologia e à miniaturização de circuitos, o estudo de sistemas lógicos que podem ser aplicados à nanotecnologia vem sendo realizado de forma abrangente. Para criação de circuitos nanoeletrônicos destacam-se a lógica reversível e a lógica majoritária. Neste trabalho é proposto o algoritmo $MPC$ (*Majority Primitives Combination*), utilizado para síntese de lógica majoritária. O algoritmo recebe uma tabela verdade como entrada e retorna uma função majoritária que cobre o mesmo conjunto de mintermos. A criação de uma função de saída válida é realizada a partir da combinação entre funções primitivas previamente otimizadas. Como critério de custo busca-se a geração de funções que tenham a menor quantidade de níveis, seguida da menor quantidade de operadores, inversores e literais. Nesse trabalho também é realizada a comparação do $MPC$ com o algoritmo *exact_mig*, considerado o melhor algoritmo para síntese de funções majoritárias atualmente. O *exact_mig* codifica a síntese exata de funções utilizando a quantidade de níveis e operadores como critério de custo. O $MPC$ utiliza dois critérios de custo adicionais, o número de inversores e o número de literais, com o objetivo de otimizar ainda mais os resultados gerados pelo *exact_mig*. Dessa forma, o $MPC$ busca a síntese de funções que possuam a mesma quantidade de níveis e de operadores, mas com uma quantidade menor de inversores e literais. Testes mostraram que ambos os algoritmos retornam soluções ótimas para todas as funções com 3 variáveis de entrada. Para funções com 4 variáveis, o $MPC$ consegue melhores resultados para 66% das funções e resultados iguais para 11%. Para funções com 5 variáveis de entrada, a partir de uma amostra de 1.000 funções geradas aleatoriamente, o $MPC$ gerou melhores resultados para 58% das funções e resultados iguais para 14%.

**Palavras-chave:** Síntese de funções majoritárias. Lógica majoritária. Funções primitivas.

# ABSTRACT

Due to technology advancements and circuits miniaturization, the study of logic systems that can be applied to nanotechnology has been progressing steadily. Among the creation of nanoeletronic circuits the reversible and majority logic stands out. This paper proposes the $MPC$ (Majority Primitives Combination) algorithm, used for majority logic synthesis. The algorithm receives a truth table as input and returns a majority function that covers the same set of minterms. The formulation of a valid output function is made with the combination of previously optimized functions. As cost criteria the algorithm searches for a function with the minimum number of levels, followed by the minimum number of gates, inverters, and literals. In this paper it's also presented a comparison between the $MPC$ and the *exact_mig*, currently considered the best algorithm for majority synthesis. The *exact_mig* encode the exact synthesis of majority functions using the number of levels and gates as cost criteria. The $MPC$ considers two additional cost criteria, the number of inverters and the number of literals, with the goal to further improve *exact_mig* results. Therefore, the $MPC$ aims to synthesize functions with the same amount of levels and gates, but with less inverters and literals. Tests have shown that both algorithms return optimal solutions for all functions with 3 input variables. For functions with 4 inputs, the $MPC$ is able to further improve 66% of all functions and achieves equal results for 11%. For functions with 5 input variables, out of a sample of 1.000 randomly generated functions, the $MPC$ further improved 58% of all functions and achieved equal results for 14%.

**Keywords:** Majority functions synthesis. Majority logic. Primitive functions.

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AIG | *And-Inverter Graphs* |
| B2M | *Boolean to Majority* |
| CMOS | *Complementary Metal-Oxide-Semiconductor* |
| MIG | *Majority Inverter Graph* |
| MLD | *Modified Levenshtein Distance* |
| MPC | *Majority Primitives Combination* |
| QCA | *Quantum-Dot Cellular Automata* |
| SMT | *Satisfiability Modulo Theories* |
| SOP | *Sum of Products* |
| XMG | *XOR-Majority Graph* |

# SUMÁRIO

# 1 INTRODUÇÃO

Com a tecnologia $CMOS$ (*Complementary Metal-Oxide-Semiconductor*) atingindo seus limites físicos (MISHRA; THAPLIYAL, 2017), a síntese de circuitos nanoeletrônicos vem sendo um grande foco de pesquisa. A lógica majoritária permite a criação de circuitos nanoeletrônicos para diferentes tecnologias, o que justifica a pesquisa por algoritmos de síntese majoritária que gerem circuitos cada vez mais otimizados. Como exemplos de aplicação da lógica majoritária tem-se as tecnologias $QCA$ (*Quantum-Dot Cellular Automata*) (KONG; SHANG; LU, 2010), *Spin-Wave* (ZOGRAFOS *et al.*, 2014) e $DNA$ *Logic* (GEORGE; SINGH, 2017).

Entre os primeiros trabalhos que abordam o conceito de lógica majoritária, destacam-se os trabalhos de Richard Lindaman (LINDAMAN, 1960), Marius Cohn (COHN; LINDAMAN, 1961) e Sheldon B. Akers (AKERS, 1961).

Lindaman (1960) propôs o primeiro teorema para aplicação da lógica majoritária em problemas de decisão binária, introduzindo um operador majoritário à álgebra booleana. O teorema propõe uma função booleana equivalente a uma operação majoritária. Na Equação 1, apresenta-se esse teorema, em que $M(A, B, C)$ representa a função majoritária entre as variáveis $A$, $B$ e $C$.

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

Posteriormente, Cohn e Lindaman (1961) propuseram um conjunto de axiomas que define a álgebra majoritária de forma independente da álgebra booleana. O conjunto proposto foi a base para a axiomatização atual da álgebra majoritária, representada pelo símbolo $\Omega$.

Akers (1961) propôs o primeiro algoritmo para síntese de funções majoritárias que utiliza como entrada somente uma tabela verdade, sem que seja necessário a conversão prévia de funções majoritárias em funções booleanas clássicas. Nesse algoritmo, a função de saída é construída com a aplicação de variáveis complementadas e constantes à tabela

verdade de entrada, com o objetivo de formular uma função lógica composta somente por operadores majoritários.

Zhang *et al.* (2004) propuseram um método que realiza o mapeamento de funções booleanas de 3 variáveis a partir de um cubo, em que cada um dos 8 vértices representa um mintermo ou maxtermo de uma função. Com a combinação de vértices tem-se a formação de arestas, que representam a interação entre mintermos, e faces do cubo, que representam uma das variáveis de entrada em sua forma complementada ou não. A partir da combinação de vértices, arestas e faces, 13 padrões de mapeamento podem ser obtidos. Cada padrão possui uma fórmula diferente para transcrever a função representada em uma função majoritária.

De forma semelhante, Wang, Niamat e Vemuru (2011) propuseram um método que utiliza um hipercubo de 4 dimensões para o mapeamento de funções com 4 variáveis de entrada, definindo um total de 143 padrões de representação. Os 143 padrões também possuem fórmulas específicas para encontrar suas funções majoritárias equivalentes.

Na álgebra majoritária, algoritmos de simplificação baseados em funções primitivas são utilizados de forma abrangente. Funções primitivas são funções que possuem no máximo um operador em sua forma otimizada. Walus *et al.* (2004) desenvolveram um algoritmo que efetua o mapeamento de cada uma das funções primitivas e utiliza a combinação dos mapas obtidos para geração de funções majoritárias. O mapeamento de funções é feito com a utilização de Mapas de Karnaugh, um método gráfico proposto por Maurice Karnaugh em 1953, que tem como objetivo a simplificação de uma função booleana clássica a partir do mapeamento de sua tabela verdade (KARNAUGH, 1953).

Mishra e Thapliyal (2017) desenvolveram um algoritmo de síntese semelhante, denominado $B2M$ (*Boolean to Majority*). O algoritmo $B2M$ recebe uma função booleana como entrada e gera uma função majoritária que cobre os mesmos mintermos. Para formar a função de saída, realiza-se a combinação de funções primitivas, selecionadas a partir do valor $MLD$ (*Modified Levenshtein Distance*) entre todas as funções primitivas e a função booleana de entrada. Na Equação 2 apresenta-se o cálculo do valor $MLD$ para $X$ e $Y$, sendo $X$ o número de mintermos cobertos pela função de entrada, $Y$ o número de mintermos cobertos por uma função primitiva qualquer e $X \cap Y$ o número de mintermos cobertos por ambas as funções. Nota-se que o valor $MLD$ é igual ao maior valor entre as duas subtrações.

$$MLD(X,Y) = max\left\{X - (X \cap Y), Y - (X \cap Y)\right\} \tag{2}$$

Chu *et al.* (2018) desenvolveram uma metodologia de decomposição que utiliza como base operadores *XOR* e majoritários, recebendo uma função booleana como entrada e a decompondo em funções mais simples. Para aplicação da metodologia, a função de entrada é transcrita no formato de representação $XMG$ (*XOR-Majority Graph*), também proposto pelos autores. O algoritmo de decomposição combina características da álgebra majoritária, expansão de Shannon (SHANNON, 1949) e $DSD$ (*Disjoint-support decomposition*) (BERTACCO; DAMIANI, 1997).

Soeken *et al.* (2017) desenvolveram o algoritmo $exact\_mig$, apresentado como o melhor algoritmo para síntese de funções majoritárias no estado atual da arte. O $exact\_mig$ recebe uma tabela verdade e retorna uma função que faz a cobertura de todos os mintermos dessa tabela, funcionando para até 6 variáveis de entrada. A principal característica desse algoritmo é a proposta de uma síntese exata para a geração de funções majoritárias, construída a partir de um conjunto de restrições ($K$) que moldam um determinado problema de acordo com as definições da álgebra majoritária. A função majoritária de saída é gerada com a aplicação de $K$ à um solucionador de satisfatibilidade (MOURA; BJØRNER, 2008). Como critério de custo o $exact\_mig$ leva em consideração a quantidade de operadores e de níveis na função de saída, possibilitando a escolha de qual desses critérios será priorizado.

Em Soeken *et al.* (2018), os autores propuseram adaptações da síntese exata presente no algoritmo $exact\_mig$, tendo como objetivo a formação de funções booleanas clássicas. Novas metodologias que utilizam como base a aplicação de restrições em um solucionador de satisfatibilidade foram apresentadas e comparadas.

Neste trabalho é proposto o algoritmo $MPC$ (*Majority Primitives Combination*), que utiliza a combinação de funções primitivas para construção de funções majoritárias a partir de uma tabela verdade de entrada. O algoritmo verifica todas as combinações possíveis entre funções primitivas e cria uma nova tabela para armazená-las. Como resultado tem-se a tabela $M_2$, que lista todas as funções majoritárias de 2 níveis em sua forma otimizada. Considera-se como critério de custo primeiramente a quantidade de níveis em uma função, seguida pela quantidade de operadores, inversores e literais.

O $MPC$ pode ser utilizado para síntese de funções majoritárias que possuem no máximo 5 variáveis de entrada e 4 níveis. Para 3 variáveis de entrada o algoritmo retorna uma solução ótima para todas as funções possíveis. Para 4 e 5 variáveis o algoritmo garante a solução ótima de funções primitivas ou cobertas por $M_2$, e utiliza a combinação dessas funções para a cobertura das funções restantes. Para 5 variáveis entretanto, o

algoritmo é viável apenas para funções que podem ser cobertas por no máximo 4 níveis.

O texto está organizado da seguinte forma. No capítulo 2, apresenta-se uma explicação sobre os elementos que constituem a álgebra majoritária, incluindo um resumo sobre a álgebra booleana convencional. No capítulo 3, apresenta-se uma explicação sobre o algoritmo $exact\_mig$. No capítulo 4, apresenta-se uma explicação sobre o algoritmo $MPC$. No capítulo 5, são apresentados os resultados obtidos na comparação entre o $MPC$ e o algoritmo $exact\_mig$. No capítulo 6, são apresentadas as conclusões do trabalho.

# 2 ÁLGEBRA MAJORITÁRIA

Neste capítulo apresenta-se a teoria sobre a álgebra majoritária, incluindo sua axiomatização e o conceito de funções primitivas. Na primeira seção, apresenta-se os principais conceitos da álgebra booleana convencional, com o objetivo de estabelecer uma base técnica para o entendimento da álgebra majoritária, abordada nas seções subsequentes.

## 2.1 ÁLGEBRA BOOLEANA

A álgebra booleana foi introduzida na literatura a partir do livro "*An Investigation Of The Laws of Thought*", escrito por George Boole e publicado em 1854. Em sua obra, Boole apresenta um novo tipo de álgebra que simula o pensamento humano, tendo como base a tomada de decisões lógicas. Uma decisão lógica é feita a partir de uma série de argumentos que podem ser verdadeiros ou falsos. Na álgebra booleana, argumentos são representados por variáveis de entrada e decisões por operadores lógicos (BOOLE, 1854).

Uma variável booleana pode assumir somente 2 valores: 0 ou 1. Esses valores são usados para expressar a relação existente entre as variáveis de entrada e a saída de um circuito, sendo o valor de saída determinado pelas variáveis de entrada. Deste modo, 0 e 1 representam o estado lógico de uma variável (TOCCI; WIDMER; MOSS, 2017). Define-se o valor 0 como o estado lógico FALSO e o valor 1 como o estado lógico VERDADEIRO.

A álgebra booleana é constituida por 3 operadores básicos: *AND* (E), *OR* (OU) e *NOT* (NÃO), que são referentes às operações de conjunção, disjunção e complemento, respectivamente. Todas as expressões booleanas podem ser representadas por essas 3 operações (FLOYD, 2015).

O operador de conjunção é equivalente à operação matemática de multiplicação e pode ser representado pelo símbolo $\wedge$. O operador de disjunção é equivalente à operação matemática de adição e pode ser representado pelo símbolo $\vee$. Por fim, o operador de complemento, também chamado de inversor, indica a inversão de um determinado valor

binário, sendo representado pelo símbolo $\neg$. Uma variável $X$ que esteja complementada é escrita como $\overline{X}$. Dessa forma, os elementos que constituem a álgebra booleana formam o conjunto $\{\mathbb{B}, \wedge, \vee, \neg\}$, sendo $\mathbb{B}$ o conjunto de números binários $\{0, 1\}$.

Na Tabela 1, apresenta-se um exemplo de uma operação *AND* entre as variáveis $A$ e $B$. Uma operação *AND* pode ser representada de forma alternativa pelo símbolo de multiplicação $(\cdot)$.

**Tabela 1** – Tabela verdade de uma operação *AND*.

| $\boldsymbol{A}$ | $\boldsymbol{B}$ | $\boldsymbol{A \cdot B}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Fonte: Próprio autor**

Pode-se observar que em uma função *AND* o valor de saída só será verdadeiro se todas as variáveis de entrada forem também verdadeiras. Para qualquer outro caso, em que pelo menos uma das entradas é falsa, o valor de saída será também falso.

Na Tabela 2, apresenta-se um exemplo de uma operação *OR* entre as variáveis $A$ e $B$. Uma operação *OR* pode ser representada de forma alternativa pelo símbolo de adição $(+)$.

**Tabela 2** – Tabela verdade de uma operação *OR*.

| $\boldsymbol{A}$ | $\boldsymbol{B}$ | $\boldsymbol{A + B}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Fonte: Próprio autor**

Pode-se observar que o valor de saída é verdadeiro sempre que pelo menos um dos valores de entrada é verdadeiro. O único caso em que a saída é falsa acontece quando todas as variáveis de entrada são também falsas. É importante apontar que, na álgebra booleana, uma adição nunca resultará em um valor maior do que 1. Isso acontece por que cada variável pode assumir somente um dos valores binários $\{0, 1\}$. Essa afirmação

continua sendo verdadeira independentemente do número de entradas em uma função.

Na Tabela 3, pode-se observar um exemplo de uma operação $NOT$ para a variável $A$, gerando a saída $\overline{A}$. Nota-se que $\overline{A}$ possui sempre um valor oposto ao valor de $A$.

**Tabela 3** – Tabela verdade de um operador $NOT$.

| $\boldsymbol{A}$ | $\boldsymbol{\overline{A}}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Fonte: Próprio autor**

## 2.2 CARACTERÍSTICAS DA ÁLGEBRA MAJORITÁRIA

Enquanto a álgebra booleana é definida pelo conjunto $\{\mathbb{B}, \wedge, \vee, \neg\}$, a álgebra majoritária é composta pelo conjunto $\{\mathbb{B}, \neg, M\}$. Os elementos $\mathbb{B}$ e $\neg$, assim como na algebra booleana, representam os valores binários $\{0, 1\}$ e o operador de complemento, respectivamente. O elemento restante do conjunto, $M$, representa o operador majoritário (CHATTOPADHYAY *et al.*, 2016).

Uma função majoritária retornará como saída o valor binário que for maioria entre suas variáveis de entrada. Dessa forma, um operador $M$ que tenha como entrada um total de 3 variáveis retornará como saída um valor verdadeiro apenas se duas ou mais entradas forem verdadeiras. Devido a essa lógica, uma função majoritária só pode ter como entrada uma quantidade ímpar de variáveis, tendo em vista que um número par pode gerar casos com quantidades iguais de entradas verdadeiras e falsas. Na tabela verdade apresentada na Tabela 4, exemplifica-se uma operação majoritária para as variáveis $A$, $B$ e $C$.

**Tabela 4** – Exemplo de uma operação majoritária.

| $\boldsymbol{A}$ | $\boldsymbol{B}$ | $\boldsymbol{C}$ | $\boldsymbol{M(A,B,C)}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**Fonte: Próprio autor**

A partir de uma operação majoritária também é possível a formação de funções $AND$

e *OR*, realizada com a fixação de uma das variáveis de entrada em um valor binário constante.

Como exemplo, considera-se a função $M(A, B, C)$. Fixando o valor de $A$ em 0 tem-se uma função *AND* entre $B$ e $C$. Fixando o valor de $A$ em 1 tem-se uma função *OR* entre $B$ e $C$. Tabelas verdade provando essa relação são apresentadas nas Tabelas 5 e 6, para as funções *AND* e *OR*, respectivamente.

**Tabela 5** – Formação de uma função *AND* a partir de $M$.

| $B$ | $C$ | $B \cdot C$ | $M(0, B, C)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

**Fonte: Próprio autor**

**Tabela 6** – Formação de uma função *OR* a partir de $M$.

| $B$ | $C$ | $B + C$ | $M(1, B, C)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 |

**Fonte: Próprio autor**

## 2.3 CLASSIFICAÇÕES DA FUNÇÃO MAJORITÁRIA

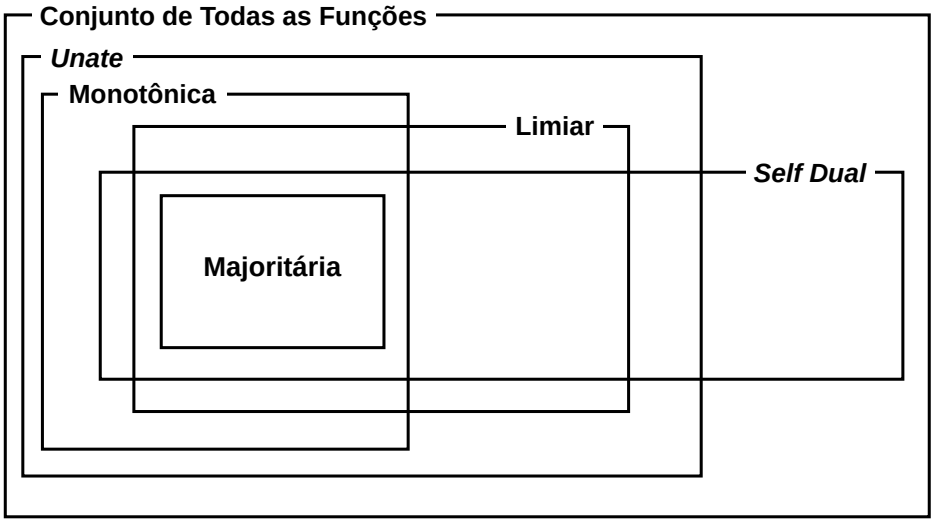
A função majoritária é composta por diferentes grupos de funções booleanas. Na Figura 1, é possível observar a relação hierárquica entre esses grupos.

Pode-se observar que uma função majoritária é, ao mesmo tempo, uma função monotônica, *unate*, limiar e *self dual*.

### 2.3.1 Funções monotônicas e *unates*

Para entender o conceito de funções monotônicas e *unates* é necessário que se entenda primeiramente o formato *SOP* (*Sum of Products*), ou Soma de Produtos. Esse formato é utilizado para a representação de funções booleanas para casos em que existem duas ou mais operações *AND* (Produto) que são conectadas por operadores *OR* (Soma). Como exemplo desse formato, tem-se a função: $(A \cdot B \cdot C) + (A \cdot \overline{B} \cdot C) + (A \cdot \overline{B} \cdot \overline{C})$.

**Figura 1** – Hierarquia entre as funções que compõem uma função majoritária.




**Fonte:** Adaptado de Sasao (2012).

Uma função é considerada monotônica se puder ser representada no formato $SOP$ com todas suas variáveis complementadas ou sem complemento. Se uma função monotônica não possui nenhuma variável complementada, define-se a função monotônica crescente. Uma função monotônica que possui todas as variáveis complementadas é uma função monotônica decrescente (SASAO, 2012).

As funções $(A \cdot B \cdot C) + (A \cdot C)$ e $(\overline{A} \cdot \overline{B} \cdot \overline{C}) + (\overline{A} \cdot \overline{C})$ são exemplos de funções monotônicas crescentes e decrescentes, respectivamente. Pode-se observar que ambas as funções possuem o formato $SOP$, mas, no primeiro exemplo, existem apenas variáveis sem complemento, enquanto no segundo todas as variáveis estão complementadas.

Uma função majoritária $M(A, B, C)$ pode ser escrita em formato monotônico crescente, como é apresentado na Equação 3.

$$M(A, B, C) = (A \cdot B) + (A \cdot C) + (B \cdot C) \tag{3}$$

Na Tabela 7, apresenta-se a tabela verdade dessa equação.

Funções *unates* representam uma classe de funções que inclui funções monotônicas crescentes e decrescentes. Uma função é considerada *unate* se estiver em formato $SOP$ e cada uma de suas variáveis de entrada possuir sempre a mesma polarização. Entende-se como polarização a existência ou não de complemento, ou seja, cada variável deve aparecer sempre complementada ou sempre sem complemento na função (SASAO, 2012).

Como exemplo, a função $(A \cdot \overline{C}) + (B \cdot \overline{C}) + (\overline{A} \cdot B \cdot \overline{C})$ não pode ser considerada

**Tabela 7** – Tabela verdade de $M(A, B, C)$ em formato monotônico crescente.

| $\boldsymbol{A}$ | $\boldsymbol{B}$ | $\boldsymbol{C}$ | $\boldsymbol{M(A,B,C)}$ | $\boldsymbol{(A \cdot B) + (A \cdot C) + (B \cdot C)}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Fonte: Próprio autor**

uma função *unate*, pois mesmo que ela esteja no formato $SOP$ a variável $A$ aparece sem complemento na primeira operação $AND$ e complementada na terceira operação. Já a função $(\overline{A} \cdot \overline{C}) + (B \cdot \overline{C}) + (\overline{A} \cdot B \cdot \overline{C})$ pode ser considerada uma função *unate* pois está em formato $SOP$ e todas suas variáveis possuem sempre a mesma polarização.

### 2.3.2 Funções limiares

Funções limiares retornam um valor de saída verdadeiro apenas se o valor resultante da soma de seus argumentos exceder um determinado limite $T$. Uma função limiar recebe como entrada um vetor de variáveis, sendo cada variável relacionada a um número real, que é chamado de peso.

Um valor de peso é utilizado para moldar a relevância de uma determinada variável na função. Os pesos e o limite são normalmente escritos em um único vetor: $[w1, w2, ..., wn; T]$ (AVEDILLO; QUINTANA; RUEDA, 1999). Por exemplo, o vetor $[2, 1, 1; 3]$ representa a função limiar $f(x1, x2, x3) = 2x1 + x2 + x3 \geq 3$, sendo $[x1, x2, x3]$ variáveis de entrada, $[2, 1, 1]$ seus respectivos pesos e o valor 3 o limite $T$. Na Tabela 8, pode-se observar a tabela verdade da função $f(x1, x2, x3) = 2x1 + x2 + x3 \geq 3$.

A Equação 4 apresenta a escrita matemática de uma função limiar.

$$Z \begin{cases} 1, \text{se} \sum_{i=1}^{n} x_i.w_i \geq T \\ 0, \text{se} \sum_{i=1}^{n} x_i.w_i < T \end{cases} \tag{4}$$

Pode-se observar que o valor de saída $Z$ será verdadeiro se o somatório de todos os produtos de uma variável com seu respectivo peso for maior ou igual a $T$. Para casos em que esse somatório é menor do que $T$, tem-se um valor de saída falso. A partir da

**Tabela 8** – Exemplo de uma função limiar.

| $x1$ | $x2$ | $x3$ | $2x1 + x2 + x3 \geq 3$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**Fonte: Próprio autor**

Equação 4 também nota-se que uma função limiar é sempre escrita em formato $SOP$.

Uma função majoritária é uma função limiar em que $T = n/2$ e $w1 = w2 = ...wn = 1$, sendo $n$ o número de variáveis de entrada. Dessa forma, a função limiar de uma função majoritária, com 3 variáveis de entrada, é escrita da seguinte forma: $M(x1, x2, x3) = x1 + x2 + x3 \geq 1,5$. Na Tabela 9, é possível observar a tabela verdade dessa função.

**Tabela 9** – Tabela verdade de uma função majoritária em formato limiar.

| $x1$ | $x2$ | $x3$ | $M(x1, x2, x3) = x1 + x2 + x3 \geq 1,5$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**Fonte: Próprio autor**

### 2.3.3 Funções *self dual*

Uma função é considerada *self dual* quando é equivalente a sua forma *dual*. A forma *dual* de uma função é obtida a partir da substituição de todas as variáveis de entrada por suas formas complementadas, seguida da inversão de suas operações e do valor de saída da função (SASAO, 2012). Por exemplo, a forma *dual* da função $(X \cdot Y) + (X \cdot Z)$ é igual a $\overline{(\overline{X} + \overline{Y}) \cdot (\overline{X} + \overline{Z})}$.

Qualquer função formada por operadores majoritários é considerada uma função *self dual*, tendo em vista que uma função majoritária sempre é equivalente à sua forma *dual*

(AMARU, 2017). Na Tabela 10, exemplifica-se a equivalência entre a função majoritária $M(A, B, C)$ e sua forma *dual* $\overline{M}(\overline{A}, \overline{B}, \overline{C})$ .

**Tabela 10** – Equivalência entre $M(A, B, C)$ e sua forma *dual*.

| $\boldsymbol{A}$ | $\boldsymbol{B}$ | $\boldsymbol{C}$ | $\boldsymbol{M(A,B,C)}$ | $\boldsymbol{\overline{M}(\overline{A},\overline{B},\overline{C})}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Fonte: Próprio autor**

## 2.4 AXIOMATIZAÇÃO DE FUNÇÕES MAJORITÁRIAS (Ω)

O conjunto de axiomas que compõe a álgebra majoritária é chamado de Ω, e pode ser dividido em axiomas de Comutatividade, Associatividade, Distribuição, Propagação de Inversão e Maioria. Todos os axiomas presentes em Ω podem ser provados por indução perfeita, com a listagem de todos os possíveis casos em uma tabela verdade (CHATTOPADHYAY *et al.*, 2016).

### 2.4.1 Comutatividade (Ω.C)

O axioma de Comutatividade, representado na Equação 5, determina que a ordem das entradas não influencia no valor de saída da função.

$$M(A, B, C) = M(A, C, B) = M(C, B, A) \tag{5}$$

Na Tabela 11, pode-se observar a tabela verdade desse axioma.

### 2.4.2 Associatividade (Ω.A)

O axioma Ω.A define que a troca de variáveis entre duas funções é possível, desde que elas estejam em níveis sequenciais e possuam uma variável em comum. Na Equação 6, tem-se um exemplo da aplicação desse axioma.

**Tabela 11** – Tabela verdade de Ω.C.

| $A$ | $B$ | $C$ | $M(A,B,C)$ | $M(A,C,B)$ | $M(C,B,A)$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | M(0,0,0) = 0 | M(0,0,0) = 0 | M(0,0,0) = 0 |
| 0 | 0 | 1 | M(0,0,1) = 0 | M(0,1,0) = 0 | M(1,0,0) = 0 |
| 0 | 1 | 0 | M(0,1,0) = 0 | M(0,0,1) = 0 | M(0,1,0) = 0 |
| 0 | 1 | 1 | M(0,1,1) = 1 | M(0,1,1) = 1 | M(1,1,0) = 1 |
| 1 | 0 | 0 | M(1,0,0) = 0 | M(1,0,0) = 0 | M(0,0,1) = 0 |
| 1 | 0 | 1 | M(1,0,1) = 1 | M(1,1,0) = 1 | M(1,0,1) = 1 |
| 1 | 1 | 0 | M(1,1,0) = 1 | M(1,0,1) = 1 | M(0,1,1) = 1 |
| 1 | 1 | 1 | M(1,1,1) = 1 | M(1,1,1) = 1 | M(1,1,1) = 1 |

**Fonte: Próprio autor**

$$M(A, D, M(B, D, C) = M(C, D, M(B, D, A)) \tag{6}$$

Nota-se que a variável compartilhada entre os 2 níveis é a variável $D$. Dessa forma, é possível trocar a variável restante no nível superior por uma das variáveis do nível subsequente. No exemplo apresentado, foi efetuada a troca entre as variáveis $A$ e $C$.

Na Tabela 12, pode-se observar a tabela verdade de Ω.A.

**Tabela 12** – Tabela verdade de Ω.A.

| $A$ | $B$ | $C$ | $D$ | $M(A,D,M(B,D,C)$ | $M(C,D,M(B,D,A))$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | M(0,0,M(0,0,0)) = 0 | M(0,0,M(0,0,0)) = 0 |
| 0 | 0 | 0 | 1 | M(0,1,M(0,1,0)) = 0 | M(0,1,M(0,1,0)) = 0 |
| 0 | 0 | 1 | 0 | M(0,0,M(0,0,1)) = 0 | M(1,0,M(0,0,0)) = 0 |
| 0 | 0 | 1 | 1 | M(0,1,M(0,1,1)) = 1 | M(1,1,M(0,1,0)) = 1 |
| 0 | 1 | 0 | 0 | M(0,0,M(1,0,0)) = 0 | M(0,0,M(1,0,0)) = 0 |
| 0 | 1 | 0 | 1 | M(0,1,M(1,1,0)) = 1 | M(0,1,M(1,1,0)) = 1 |
| 0 | 1 | 1 | 0 | M(0,0,M(1,0,1)) = 0 | M(1,0,M(1,0,0)) = 0 |
| 0 | 1 | 1 | 1 | M(0,1,M(1,1,1)) = 1 | M(1,1,M(1,1,0)) = 1 |
| 1 | 0 | 0 | 0 | M(1,0,M(0,0,0)) = 0 | M(0,0,M(0,0,1)) = 0 |
| 1 | 0 | 0 | 1 | M(1,0,M(0,0,1)) = 0 | M(1,1,M(0,1,0)) = 0 |
| 1 | 0 | 1 | 0 | M(1,0,M(0,1,0)) = 0 | M(1,0,M(1,0,1)) = 0 |
| 1 | 0 | 1 | 1 | M(1,1,M(0,1,1)) = 1 | M(1,1,M(0,1,1)) = 1 |
| 1 | 1 | 0 | 0 | M(1,0,M(1,0,0)) = 0 | M(0,0,M(1,0,1)) = 0 |
| 1 | 1 | 0 | 1 | M(1,1,M(1,1,0)) = 1 | M(0,1,M(1,1,1)) = 1 |
| 1 | 1 | 1 | 0 | M(1,0,M(1,0,1)) = 1 | M(1,0,M(1,0,1)) = 1 |
| 1 | 1 | 1 | 1 | M(1,1,M(1,1,1)) = 1 | M(1,1,M(1,1,1)) = 1 |

**Fonte: Próprio autor**

### 2.4.3 Distribuição (Ω.D)

O axioma de Distribuição (Ω.D) determina que é possível a distribuição de um conjunto de variáveis para funções que estejam em níveis sequenciais. Na Equação 7 apresenta-se um exemplo desse teorema, em que o conjunto $\{A, B\}$ foi distribuído para as variáveis no nível seguinte.

$$M(A, B, M(D, E, C)) = M(M(A, B, D), M(A, B, E), M(A, B, C)) \tag{7}$$

Na Tabela 13, pode-se observar a tabela verdade de Ω.D.

**Tabela 13** – Tabela verdade de Ω.D.

| $A$ | $B$ | $C$ | $D$ | $E$ | $M(A, B, M(D, E, C))$ | $M(M(A, B, D), M(A, B, E), M(A, B, C))$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | M(0,0,M(0,0,0)) = 0 | M(M(0,0,0), M(0,0,0), M(0,0,0)) = 0 |
| 0 | 0 | 0 | 0 | 1 | M(0,0,M(0,1,0)) = 0 | M(M(0,0,0), M(0,0,1), M(0,0,0)) = 0 |
| 0 | 0 | 0 | 1 | 0 | M(0,0,M(1,0,0)) = 0 | M(M(0,0,1), M(0,0,0), M(0,0,0)) = 0 |
| 0 | 0 | 0 | 1 | 1 | M(0,0,M(1,1,0)) = 0 | M(M(0,0,1), M(0,0,1), M(0,0,0)) = 0 |
| 0 | 0 | 1 | 0 | 0 | M(0,0,M(0,0,1)) = 0 | M(M(0,0,0), M(0,0,0), M(0,0,1)) = 0 |
| 0 | 0 | 1 | 0 | 1 | M(0,0,M(0,1,1)) = 0 | M(M(0,0,0), M(0,0,1), M(0,0,1)) = 0 |
| 0 | 0 | 1 | 1 | 0 | M(0,0,M(1,0,1)) = 0 | M(M(0,0,1), M(0,0,0), M(0,0,1)) = 0 |
| 0 | 0 | 1 | 1 | 1 | M(0,0,M(1,1,1)) = 0 | M(M(0,0,1), M(0,0,1), M(0,0,1)) = 0 |
| 0 | 1 | 0 | 0 | 0 | M(0,1,M(0,0,0)) = 0 | M(M(0,1,0), M(0,1,0), M(0,1,0)) = 0 |
| 0 | 1 | 0 | 0 | 1 | M(0,1,M(0,1,0)) = 0 | M(M(0,1,0), M(0,1,1), M(0,1,0)) = 0 |
| 0 | 1 | 0 | 1 | 0 | M(0,1,M(1,0,0)) = 0 | M(M(0,1,1), M(0,1,0), M(0,0,0)) = 0 |
| 0 | 1 | 0 | 1 | 1 | M(0,1,M(1,1,0)) = 1 | M(M(0,1,1), M(0,1,1), M(0,1,0)) = 1 |
| 0 | 1 | 1 | 0 | 0 | M(0,1,M(0,0,1)) = 0 | M(M(0,1,0), M(0,1,0), M(0,1,1)) = 0 |
| 0 | 1 | 1 | 0 | 1 | M(0,1,M(0,1,1)) = 1 | M(M(0,1,0), M(0,1,1), M(0,1,1)) = 1 |
| 0 | 1 | 1 | 1 | 0 | M(0,1,M(1,0,1)) = 1 | M(M(0,1,1), M(0,1,0), M(0,1,1)) = 1 |
| 0 | 1 | 1 | 1 | 1 | M(0,1,M(1,1,1)) = 1 | M(M(0,1,1), M(0,1,1), M(0,1,1)) = 1 |
| 1 | 0 | 0 | 0 | 0 | M(1,0,M(0,0,0)) = 0 | M(M(1,0,0), M(1,0,0), M(1,0,0)) = 0 |
| 1 | 0 | 0 | 0 | 1 | M(1,0,M(0,1,0)) = 0 | M(M(1,0,0), M(1,0,1), M(1,0,0)) = 0 |
| 1 | 0 | 0 | 1 | 0 | M(1,0,M(1,0,0)) = 0 | M(M(1,0,0), M(1,0,0), M(1,0,0)) = 0 |
| 1 | 0 | 0 | 1 | 1 | M(1,0,M(1,1,0)) = 1 | M(M(1,0,1), M(1,0,1), M(1,0,0)) = 1 |
| 1 | 0 | 1 | 0 | 0 | M(1,0,M(0,0,1)) = 0 | M(M(1,0,0), M(1,0,0), M(1,0,1)) = 0 |
| 1 | 0 | 1 | 0 | 1 | M(1,0,M(0,1,1)) = 1 | M(M(1,0,0), M(1,0,1), M(1,0,1)) = 1 |
| 1 | 0 | 1 | 1 | 0 | M(1,0,M(1,0,1)) = 1 | M(M(1,0,1), M(1,0,0), M(1,0,1)) = 1 |
| 1 | 0 | 1 | 1 | 1 | M(1,0,M(1,1,1)) = 1 | M(M(1,0,1), M(1,0,1), M(1,0,1)) = 1 |
| 1 | 1 | 0 | 0 | 0 | M(1,1,M(0,0,0)) = 1 | M(M(1,1,0), M(1,1,0), M(1,1,0)) = 1 |
| 1 | 1 | 0 | 0 | 1 | M(1,1,M(0,1,0)) = 1 | M(M(1,1,0), M(1,1,1), M(1,1,0)) = 1 |
| 1 | 1 | 0 | 1 | 0 | M(1,1,M(1,0,0)) = 1 | M(M(1,1,1), M(1,1,0), M(1,1,0)) = 1 |
| 1 | 1 | 0 | 1 | 1 | M(1,1,M(1,1,0)) = 1 | M(M(1,1,1), M(1,1,1), M(1,1,0)) = 1 |
| 1 | 1 | 1 | 0 | 0 | M(1,1,M(0,0,1)) = 1 | M(M(1,1,0), M(1,1,0), M(1,1,1)) = 1 |
| 1 | 1 | 1 | 0 | 1 | M(1,1,M(0,1,1)) = 1 | M(M(1,1,0), M(1,1,1), M(1,1,1)) = 1 |
| 1 | 1 | 1 | 1 | 0 | M(1,1,M(1,0,1)) = 1 | M(M(1,1,1), M(1,1,0), M(1,1,1)) = 1 |
| 1 | 1 | 1 | 1 | 1 | M(1,1,M(1,1,1)) = 1 | M(M(1,1,1), M(1,1,1), M(1,1,1)) = 1 |

**Fonte: Próprio autor**

### 2.4.4 Propagação de inversão (Ω.I)

O axioma Ω.I, representado na Equação 8, determina que os inversores aplicados nas entradas e na saída de uma função podem ser propagados para entradas sem complemento, desde que a propagação seja feita para a função como um todo. Essa relação é possível por que uma função majoritária é uma função *self dual*. Nota-se que $\overline{M}(A, B, C)$ e $M(\overline{A}, \overline{B}, \overline{C})$ são a forma *dual* uma da outra.

$$\overline{M}(A, B, C) = M(\overline{A}, \overline{B}, \overline{C}) \tag{8}$$

Na Tabela 14, pode-se observar a tabela verdade de Ω.I.

**Tabela 14** – Tabela verdade de Ω.I.

| $\boldsymbol{A}$ | $\boldsymbol{B}$ | $\boldsymbol{C}$ | $\boldsymbol{\overline{M}(A,B,C)}$ | $\boldsymbol{M(\overline{A},\overline{B},\overline{C})}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |

**Fonte: Próprio autor**

### 2.4.5 Maioria (Ω.M)

O axioma Ω.M pode ser dividido em duas equações. A Equação 9 define que a saída de uma função majoritária é igual ao valor que for maioria entre suas entradas. A Equação 10 define que o valor de saída será igual à variável de desempate em funções que possuem uma quantidade igual de valores verdadeiros e falsos.

$$M(A, A, B) = A \tag{9}$$

$$M(A, \overline{A}, B) = B \tag{10}$$

Na Tabela 15, pode-se observar a tabela verdade de Ω.M.

**Tabela 15** – Tabela verdade de Ω.M.

| **A** | **B** | $\boldsymbol{M(A, A, B) = A}$ | $\boldsymbol{M(A, \overline{A}, B) = B}$ |
|---|---|---|---|
| 0 | 0 | M(0,0,0)) = 0 | M(0,1,0) = 0 |
| 0 | 1 | M(0,0,1)) = 0 | M(0,1,1) = 1 |
| 1 | 0 | M(1,1,0) = 1 | M(1,0,0) = 0 |
| 1 | 1 | M(1,1,1) = 1 | M(1,0,1) = 1 |

**Fonte: Próprio autor**

## 2.5 FUNÇÕES MAJORITÁRIAS PRIMITIVAS

Funções majoritárias primitivas são funções formadas por um único, ou por nenhum, operador majoritário. Na álgebra majoritária, essas funções podem ser utilizadas como base para a construção de funções mais complexas. Todas as funções majoritárias primitivas podem ser obtidas a partir dos conjuntos $U$, $V$, $G$ e $T$, sendo cada conjunto correspondente a funções com uma quantidade específica de entradas. A quantidade total de funções majoritárias primitivas é obtida pela soma do número de funções em $U$, $V$, $G$ e $T$ (WANG *et al.*, 2015).

O conjunto $U$ representa funções que não possuem nenhuma variável de entrada, cobrindo as constantes 0 e 1.

O conjunto $V$ representa todas as funções formadas por uma única variável de entrada, em sua forma complementada ou não. Na Equação 11, tem-se o cálculo da quantidade de funções que a listagem de $V$ deve gerar.

$$|V| = (2 \cdot n) \tag{11}$$

Na Tabela 16, pode-se observar a listagem do conjunto $V$ para 3 variáveis de entrada ($n = 3$). Nota-se que as funções clássicas e suas correspondentes majoritárias são iguais porque o conjunto $V$ é composto apenas por funções sem operadores.

O conjunto $G$ é formado por funções que possuem duas variáveis de entrada, gerando um único operador *AND* ou *OR*. Na Equação 12, tem-se o cálculo da quantidade de funções geradas pela listagem de $G$. As variáveis $E$ e $O$ representam as possíveis combinações entre as variáveis de entrada, para operações *AND* e *OR* respectivamente. Para $n = 3$, tem-se $E = \{A \cdot B, A \cdot C, B \cdot C\}$ e $O = \{A + B, A + C, B + C\}$. Cada combinação possui 4 variações de inversão, a combinação $A + B$ por exemplo, possui as variações $\{A + B, \overline{A} + B, A + \overline{B}, \overline{A} + \overline{B}\}$.

**Tabela 16** – Listagem do conjunto $V$ para $n = 3$.

| **Função Clássica** | **Função Majoritária** |
|---|---|
| $A$ | $A$ |
| $B$ | $B$ |
| $C$ | $C$ |
| $\overline{A}$ | $\overline{A}$ |
| $\overline{B}$ | $\overline{B}$ |
| $\overline{C}$ | $\overline{C}$ |

$$|G| = (4 \cdot |E|) + (4 \cdot |O|) \tag{12}$$

Na Tabela 17, tem-se a listagem do conjunto $G$ para $n = 3$. Nota-se que funções majoritárias formam operadores $AND$ e $OR$ com a fixação de uma das entradas em um valor constante.

O conjunto $T$ representa funções com um único operador majoritário, que não possuam nenhum valor constante ou variável repetida como entrada. Na Equação 13, apresenta-se o cálculo da quantidade de funções geradas pela listagem de $T$. A variável $t$ representa a quantidade de combinações possíveis entre as variáveis de entrada, considerando 3 entradas por combinação. Nota-se que cada combinação possui 8 variações de polarização e, para $n = 3$, existe apenas uma combinação possível.

$$|T| = 8 \cdot t \tag{13}$$

Na Tabela 18, tem-se a listagem do conjunto $T$ para $n = 3$.

Na Tabela 19 tem-se a lista completa de funções primitivas para $n = 3$. Nota-se que $|U| + |V| + |G| + |T| = 40$.

Na Tabela 20 tem-se a lista completa de funções primitivas para $n = 4$. Nota-se que $|U| + |V| + |G| + |T| = 90$.

Neste capítulo foram apresentados os conceitos que definem a álgebra majoritária, com o objetivo de efetuar uma base para o entendimento dos algoritmos *exact_mig* e $MPC$, descritos nos capítulos seguintes. Abordou-se a lógica de uma função majoritária e sua classificação entre os tipos de funções booleanas existentes, o conceito de funções primitivas majoritárias e como essas funções podem ser obtidas, e o conjunto de axiomas que define a álgebra majoritária, sendo cada axioma explicado individualmente.

**Tabela 17** – Listagem do conjunto $G$ para $n = 3$.

| **Função Clássica** | **Função Majoritária** |
|---|---|
| $A \cdot B$ | $M(A, B, 0)$ |
| $\overline{A} \cdot B$ | $M(\overline{A}, B, 0)$ |
| $A \cdot \overline{B}$ | $M(A, \overline{B}, 0)$ |
| $\overline{A} \cdot \overline{B}$ | $\overline{M}(A, B, 1)$ |
| $A \cdot C$ | $M(A, 0, C)$ |
| $\overline{A} \cdot C$ | $M(\overline{A}, 0, C)$ |
| $A \cdot \overline{C}$ | $M(A, 0, \overline{C})$ |
| $\overline{A} \cdot \overline{C}$ | $\overline{M}(A, 1, C)$ |
| $B \cdot C$ | $M(0, B, C)$ |
| $\overline{B} \cdot C$ | $M(0, \overline{B}, C)$ |
| $B \cdot \overline{C}$ | $M(0, B, \overline{C})$ |
| $\overline{B} \cdot \overline{C}$ | $\overline{M}(1, B, C)$ |
| $A + B$ | $M(A, B, 1)$ |
| $\overline{A} + B$ | $M(\overline{A}, B, 1)$ |
| $A + \overline{B}$ | $M(A, \overline{B}, 1)$ |
| $\overline{A} + \overline{B}$ | $\overline{M}(A, B, 0)$ |
| $A + C$ | $M(A, 1, C)$ |
| $\overline{A} + C$ | $M(\overline{A}, 1, C)$ |
| $A + \overline{C}$ | $M(A, 1, \overline{C})$ |
| $\overline{A} + \overline{C}$ | $\overline{M}(A, 0, C)$ |
| $B + C$ | $M(1, B, C)$ |
| $\overline{B} + C$ | $M(1, \overline{B}, C)$ |
| $B + \overline{C}$ | $M(1, B, \overline{C})$ |
| $\overline{B} + \overline{C}$ | $\overline{M}(0, B, C)$ |

**Tabela 18** – Listagem do conjunto $T$ para $n = 3$.

| **Função Clássica** | **Função Majoritária** |
|---|---|
| $AB + AC + BC$ | $M(A, B, C)$ |
| $\overline{A} \cdot \overline{B} + \overline{A} \cdot \overline{C} + \overline{B} \cdot \overline{C}$ | $\overline{M}(A, B, C)$ |
| $\overline{A} \cdot B + \overline{A} \cdot C + B \cdot C$ | $M(\overline{A}, B, C)$ |
| $A \cdot \overline{B} + A \cdot \overline{C} + \overline{B} \cdot \overline{C}$ | $M(A, \overline{B}, \overline{C})$ |
| $A \cdot \overline{B} + A \cdot C + \overline{B} \cdot C$ | $M(A, \overline{B}, C)$ |
| $\overline{A} \cdot B + \overline{A} \cdot \overline{C} + B \cdot \overline{C}$ | $M(\overline{A}, B, \overline{C})$ |
| $A \cdot B + A \cdot \overline{C} + B \cdot \overline{C}$ | $M(A, B, \overline{C})$ |
| $\overline{A} \cdot \overline{B} + \overline{A} \cdot C + \overline{B} \cdot C$ | $M(\overline{A}, \overline{B}, C)$ |

**Tabela 19** – Tabela de funções majoritárias primitivas para $n = 3$.

| **Nº** | **Função Primitiva** | **Função Majoritária** | **Nº** | **Função Primitiva** | **Função Majoritária** |
|---|---|---|---|---|---|
| 1 | $0$ | $0$ | 21 | $A + B$ | $M(A, B, 1)$ |
| 2 | $1$ | $1$ | 22 | $\overline{A} + B$ | $M(\overline{A}, B, 1)$ |
| 3 | $A$ | $A$ | 23 | $A + \overline{B}$ | $M(A, \overline{B}, 1)$ |
| 4 | $B$ | $B$ | 24 | $\overline{A} + \overline{B}$ | $\overline{M}(A, B, 0)$ |
| 5 | $C$ | $C$ | 25 | $A + C$ | $M(A, 0, C)$ |
| 6 | $\overline{A}$ | $\overline{A}$ | 26 | $\overline{A} + C$ | $M(\overline{A}, 1, C)$ |
| 7 | $\overline{B}$ | $\overline{B}$ | 27 | $A + \overline{C}$ | $M(A, 1, \overline{C})$ |
| 8 | $\overline{C}$ | $\overline{C}$ | 28 | $\overline{A} + \overline{C}$ | $\overline{M}(A, 0, C)$ |
| 9 | $A \cdot B$ | $M(A, B, 0)$ | 29 | $B + C$ | $M(1, B, C)$ |
| 10 | $\overline{A} \cdot B$ | $M(\overline{A}, B, 0)$ | 30 | $\overline{B} + C$ | $M(1, \overline{B}, C)$ |
| 11 | $A \cdot \overline{B}$ | $M(A, \overline{B}, 0)$ | 31 | $B + \overline{C}$ | $M(1, B, \overline{C})$ |
| 12 | $\overline{A} \cdot \overline{B}$ | $\overline{M}(A, B, 1)$ | 32 | $\overline{B} + \overline{C}$ | $\overline{M}(0, B, C)$ |
| 13 | $A \cdot C$ | $M(A, 0, C)$ | 33 | $AB + AC + BC$ | $M(A, B, C)$ |
| 14 | $\overline{A} \cdot C$ | $M(\overline{A}, 0, C)$ | 34 | $\overline{A} \cdot B + \overline{A} \cdot C + B \cdot C$ | $M(\overline{A}, B, C)$ |
| 15 | $A \cdot \overline{C}$ | $M(A, 0, \overline{C})$ | 35 | $A \cdot \overline{B} + A \cdot C + \overline{B} \cdot C$ | $M(A, \overline{B}, C)$ |
| 16 | $\overline{A} \cdot \overline{C}$ | $\overline{M}(A, 1, C)$ | 36 | $A \cdot B + A \cdot \overline{C} + B \cdot \overline{C}$ | $M(A, B, \overline{C})$ |
| 17 | $B \cdot C$ | $M(0, B, C)$ | 37 | $\overline{A} \cdot \overline{B} + \overline{A} \cdot C + \overline{B} \cdot C$ | $M(\overline{A}, \overline{B}, C)$ |
| 18 | $\overline{B} \cdot C$ | $M(0, \overline{B}, C)$ | 38 | $\overline{A} \cdot \overline{B} + \overline{A} \cdot \overline{C} + \overline{B} \cdot \overline{C}$ | $\overline{M}(A, B, C)$ |
| 19 | $B \cdot \overline{C}$ | $M(0, B, \overline{C})$ | 39 | $A \cdot \overline{B} + A \cdot \overline{C} + \overline{B} \cdot \overline{C}$ | $M(A, \overline{B}, \overline{C})$ |
| 20 | $\overline{B} \cdot \overline{C}$ | $\overline{M}(1, B, C)$ | 40 | $\overline{A} \cdot B + \overline{A} \cdot \overline{C} + B \cdot \overline{C}$ | $M(\overline{A}, B, \overline{C})$ |

**Fonte: Próprio autor**

**Tabela 20** – Tabela de funções majoritárias primitivas para $n = 4$.

| Nº | Função Primitiva | Função Majoritária | Nº | Função Primitiva | Função Majoritária |
|---|---|---|---|---|---|
| 1 | $0$ | $0$ | 46 | $\overline{A} + \overline{D}$ | $\overline{M}(A, 0, D)$ |
| 2 | $1$ | $1$ | 47 | $B + C$ | $M(B, 1, C)$ |
| 3 | $A$ | $A$ | 48 | $\overline{B} + C$ | $M(\overline{B}, 1, C)$ |
| 4 | $B$ | $B$ | 49 | $B + \overline{C}$ | $M(B, 1, \overline{C})$ |
| 5 | $C$ | $C$ | 50 | $\overline{B} + \overline{C}$ | $\overline{M}(B, 0, C)$ |
| 6 | $D$ | $D$ | 51 | $B + D$ | $M(B, 1, D)$ |
| 7 | $\overline{A}$ | $\overline{A}$ | 52 | $\overline{B} + D$ | $M(\overline{B}, 1, D)$ |
| 8 | $\overline{B}$ | $\overline{B}$ | 53 | $B + \overline{D}$ | $M(B, 1, \overline{D})$ |
| 9 | $\overline{C}$ | $\overline{C}$ | 54 | $\overline{B} + \overline{D}$ | $\overline{M}(B, 0, D)$ |
| 10 | $\overline{D}$ | $\overline{D}$ | 55 | $D + C$ | $M(1, C, D)$ |
| 11 | $A \cdot B$ | $M(A, B, 0)$ | 56 | $\overline{D} + C$ | $M(1, C, \overline{D})$ |
| 12 | $\overline{A} \cdot B$ | $M(\overline{A}, B, 0)$ | 57 | $D + \overline{C}$ | $M(1, \overline{C}, D)$ |
| 13 | $A \cdot \overline{B}$ | $M(A, \overline{B}, 0)$ | 58 | $\overline{D} + \overline{C}$ | $\overline{M}(0, C, D)$ |
| 14 | $\overline{A} \cdot \overline{B}$ | $\overline{M}(A, B, 1)$ | 59 | $A \cdot B + A \cdot C + B \cdot C$ | $M(A, B, C)$ |
| 15 | $A \cdot C$ | $M(A, 0, C)$ | 60 | $\overline{A} \cdot \overline{B} + \overline{A} \cdot \overline{C} + \overline{B} \cdot \overline{C}$ | $\overline{M}(A, B, C)$ |
| 16 | $\overline{A} \cdot C$ | $M(\overline{A}, 0, C)$ | 61 | $\overline{A} \cdot B + \overline{A} \cdot C + B \cdot C$ | $M(\overline{A}, B, C)$ |
| 17 | $A \cdot \overline{C}$ | $M(A, 0, \overline{C})$ | 62 | $A \cdot \overline{B} + A \cdot \overline{C} + \overline{B} \cdot \overline{C}$ | $M(A, \overline{B}, \overline{C})$ |
| 18 | $\overline{A} \cdot \overline{C}$ | $\overline{M}(A, 1, C)$ | 63 | $A \cdot \overline{B} + A \cdot C + \overline{B} \cdot C$ | $M(A, \overline{B}, C)$ |
| 19 | $A \cdot D$ | $M(A, 0, D)$ | 64 | $\overline{A} \cdot B + \overline{A} \cdot \overline{C} + B \cdot \overline{C}$ | $M(\overline{A}, B, \overline{C})$ |
| 20 | $\overline{A} \cdot D$ | $M(\overline{A}, 0, D)$ | 65 | $A \cdot B + A \cdot \overline{C} + B \cdot \overline{C}$ | $M(A, B, \overline{C})$ |
| 21 | $A \cdot \overline{D}$ | $M(A, 0, \overline{D})$ | 66 | $\overline{A} \cdot \overline{B} + \overline{A}\dot{C} + \overline{B} \cdot C$ | $M(\overline{A}, \overline{B}, C)$ |
| 22 | $\overline{A} \cdot \overline{D}$ | $\overline{M}(A, 1, D)$ | 67 | $A \cdot B + A \cdot D + B \cdot D$ | $M(A, B, D)$ |
| 23 | $B \cdot C$ | $M(0, B, C)$ | 68 | $\overline{A} \cdot \overline{B} + \overline{A} \cdot \overline{D} + \overline{B} \cdot \overline{D}$ | $\overline{M}(A, B, D)$ |
| 24 | $\overline{B} \cdot C$ | $M(0, \overline{B}, C)$ | 69 | $\overline{A} \cdot B + \overline{A} \cdot D + B \cdot D$ | $M(\overline{A}, B, D)$ |
| 25 | $B \cdot \overline{C}$ | $M(0, B, \overline{C})$ | 70 | $A \cdot \overline{B} + A \cdot \overline{D} + \overline{B} \cdot \overline{D}$ | $M(A, \overline{B}, \overline{D})$ |
| 26 | $\overline{B} \cdot \overline{C}$ | $\overline{M}(1, B, C)$ | 71 | $A \cdot \overline{B} + A \cdot D + \overline{B} \cdot D$ | $M(A, \overline{B}, D)$ |
| 27 | $B \cdot D$ | $M(0, B, D)$ | 72 | $\overline{A} \cdot B + \overline{A} \cdot \overline{D} + B \cdot \overline{D}$ | $M(\overline{A}, B, \overline{D})$ |
| 28 | $\overline{B} \cdot D$ | $M(0, \overline{B}, D)$ | 73 | $A \cdot B + A \cdot \overline{D} + B \cdot \overline{D}$ | $M(A, B, \overline{D})$ |
| 29 | $B \cdot \overline{D}$ | $M(0, B, \overline{D})$ | 74 | $\overline{A} \cdot \overline{B} + \overline{A} \cdot D + \overline{B} \cdot D$ | $M(\overline{A}, \overline{B}, D)$ |
| 30 | $\overline{B} \cdot \overline{D}$ | $\overline{M}(1, B, D)$ | 75 | $A \cdot C + A \cdot D + C \cdot D$ | $M(A, C, D)$ |
| 31 | $D \cdot C$ | $M(0, D, C)$ | 76 | $\overline{A} \cdot \overline{C} + \overline{A} \cdot \overline{D} + \overline{C} \cdot D$ | $\overline{M}(A, C, D)$ |
| 32 | $\overline{D} \cdot C$ | $M(0, \overline{D}, C)$ | 77 | $\overline{A} \cdot C + \overline{A} \cdot D + C \cdot D$ | $M(\overline{A}, C, D)$ |
| 33 | $D \cdot \overline{C}$ | $M(0, D, \overline{C})$ | 78 | $A \cdot \overline{C} + A \cdot \overline{D} + \overline{C} \cdot \overline{D}$ | $M(A, \overline{C}, \overline{D})$ |
| 34 | $\overline{D} \cdot \overline{C}$ | $\overline{M}(1, D, C)$ | 79 | $A \cdot \overline{C} + A \cdot D + \overline{C} \cdot D$ | $M(A, \overline{C}, D)$ |
| 35 | $A + B$ | $M(A, B, 1)$ | 80 | $\overline{A} \cdot C + \overline{A} \cdot \overline{D} + C \cdot \overline{D}$ | $M(\overline{A}, C, \overline{D})$ |
| 36 | $\overline{A} + B$ | $M(\overline{A}, B, 1)$ | 81 | $A \cdot C + A \cdot \overline{D} + C \cdot \overline{D}$ | $M(A, C, \overline{D})$ |
| 37 | $A + \overline{B}$ | $M(A, \overline{B}, 1)$ | 82 | $A \cdot C + A \cdot D + C \cdot \overline{D}$ | $M(A, C, \overline{D})$ |
| 38 | $\overline{A} + \overline{B}$ | $\overline{M}(A, B, 0)$ | 83 | $B \cdot C + B \cdot D + C \cdot D$ | $M(B, C, D)$ |
| 39 | $A + C$ | $M(A, 1, C)$ | 84 | $\overline{B} \cdot \overline{C} + \overline{B} \cdot \overline{D} + \overline{C} \cdot \overline{D}$ | $\overline{M}(B, C, D)$ |
| 40 | $\overline{A} + C$ | $M(\overline{A}, 1, C)$ | 85 | $\overline{B} \cdot C + \overline{B} \cdot D + C \cdot D$ | $M(\overline{B}, C, D)$ |
| 41 | $A + \overline{C}$ | $M(A, 1, \overline{C})$ | 86 | $B \cdot \overline{C} + B \cdot \overline{D} + \overline{C} \cdot D$ | $M(B, \overline{C}, \overline{D})$ |
| 42 | $\overline{A} + \overline{C}$ | $\overline{M}(A, 0, C)$ | 87 | $B \cdot \overline{C} + B \cdot D + \overline{C} \cdot D$ | $M(B, \overline{C}, D)$ |
| 43 | $A + D$ | $M(A, 1, D)$ | 88 | $\overline{B} \cdot C + \overline{B} \cdot \overline{D} + C \cdot \overline{D}$ | $M(\overline{B}, C, \overline{D})$ |
| 44 | $\overline{A} + D$ | $M(\overline{A}, 1, D)$ | 89 | $B \cdot C + B \cdot \overline{D} + C \cdot \overline{D}$ | $M(B, C, \overline{D})$ |
| 45 | $A + \overline{D}$ | $M(A, 1, \overline{D})$ | 90 | $\overline{B} \cdot \overline{C} + \overline{B} \cdot D + \overline{C} \cdot D$ | $M(\overline{B}, \overline{C}, D)$ |

**Fonte: Próprio autor**

# 3 ALGORITMO *EXACT_MIG*

Neste capítulo apresenta-se o algoritmo *exact_mig*, proposto em Soeken *et al.* (2017) e apresentado como o melhor algoritmo para síntese de funções majoritárias no estado atual da arte. São abordados os critérios de custo utilizados pelo algoritmo, a forma como é feita a representação de funções e a codificação de variáveis e restrições que formulam sua lógica de síntese.

O algoritmo *exact_mig* possui como objetivo a síntese exata para formação de funções. Entende-se como síntese exata a tarefa de encontrar a função de menor custo possível que cubra um determinado conjunto de mintermos. O *exact_mig* recebe uma tabela verdade $f$ como entrada e retorna a função majoritária de menor custo que cubra os mesmos mintermos de $f$. Devido à complexidade da síntese exata, o algoritmo é viável para até 6 variáveis de entrada (SOEKEN *et al.*, 2017).

Para formação de funções, o *exact_mig* utiliza o conceito de $MIG$s (*Majority-Inverter Graphs*), uma forma de representar funções majoritárias a partir de redes lógicas compostas exclusivamente por operadores majoritários de 3 entradas e inversores.

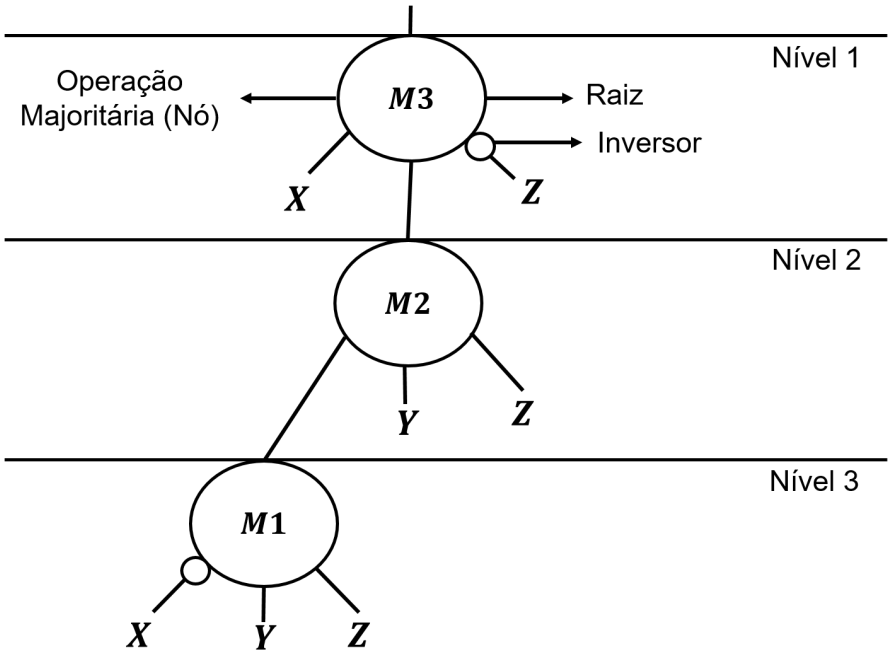
Na Figura 2, tem-se como exemplo a função $M(X, M(M(\overline{X}, Y, Z), Y, Z), \overline{Z})$ em formato $MIG$, e cada elemento de sua estrutura é apontado.

As variáveis existentes em um $MIG$ são representadas por ramos, que se conectam posteriormente a um nó. Cada nó representa uma função majoritária de 3 entradas e cada entrada pode ou não estar complementada. Inversores são representados por marcações circulares em ramos de entrada ou saída. O nó de maior índice é denominado raiz. A profundidade de um nó é igual à distância deste nó até a base da representação. Um conjunto de nós com a mesma profundidade é denominado nível (AMARU; GAILLARDON; MICHELI, 2014).

O algoritmo *exact_mig* foi implementado para o *framework Cirkit*, um conjunto de ferramentas utilizado para efetuar a síntese lógica de circuitos majoritários e reversíveis, disponibilizado por Mathias Soeken em (SOEKEN, 2017). Um manual para utilização do

algoritmo *exact_mig* está disponibilizado no Apêndice A.

**Figura 2** – Exemplo da estrutura de um $MIG$.




**Fonte:** Próprio autor

## 3.1 SOLUCIONADORES DE SATISFATIBILIDADE

No algoritmo *exact_mig*, os dados de entrada são codificados como um problema que pode ser resolvido por um solucionador de satisfatibilidade. Um solucionador de satisfatibilidade, também chamado de solucionador $SAT$, é um algoritmo que recebe como entrada um conjunto de restrições compostas por variáveis booleanas, e retorna como saída um modelo de valores que respeita todas as restrições impostas ao problema. Na literatura, um resultado válido é chamado de $SAT$ e um resultado inválido é chamado de $UNSAT$ (GANZINGER *et al.*, 2004).

Dada uma entrada $K$, sendo $K$ um conjunto de restrições, um solucionador associa valores aleatórios para cada uma das variáveis existentes e cada associação propaga consequências no conjunto de restrições como um todo. O objetivo desse algoritmo é gerar um modelo de valores que respeite inteiramente o conjunto $K$, gerando assim um resultado $SAT$ ao problema. Um problema de resultado $UNSAT$ acontece quando existe uma contradição (infactibilidade) em $K$ ou quando um resultado válido não pôde ser obtido após uma busca exaustiva. Entende-se como contradição casos em que duas restrições não podem ser verdadeiras ao mesmo tempo.

Para exemplificar um resultado $SAT$ considera-se $K = \{p + q; p \cdot (p + \overline{q})\}$. Um

modelo de valores que respeita as duas restrições em $K$ é $M = \{p = 1, q = 0\}$, logo $M$ é um resultado $SAT$ para $K$.

Para exemplificar um resultado $UNSAT$ considera-se $K = \left\{p + q; \overline{(p + q)}\right\}$. Pode-se observar que o resultado é $UNSAT$ por que as duas restrições não podem ser verdadeiras ao mesmo tempo, não existe um modelo válido de valores para $K$.

Um solucionador $SMT$ (Solucionador de Módulo e Teoria), é uma extensão de um solucionador de satisfatibilidade comum, sendo utilizado para casos em que o solucionador precisa respeitar uma teoria específica, representada pela variável $T$.

Em um solucionador $SAT$, todas as variáveis e restrições assumem somente valores booleanos. Já em um $SMT$, embora as restrições ainda representem funções booleanas, as variáveis que compõem essas restrições não estão limitadas somente a um único tipo de valor. Essa característica permite uma modelagem de restrições mais abrangente, possibilitando uma codificação que formule uma teoria $T$ (MOURA; DUTERTRE; SHANKAR, 2007).

Dessa forma, o solucionador $SMT$ representa a aplicação de um solucionador $SAT$ a um problema em que $K$ está codificado de forma a reproduzir os conceitos de uma teoria $T$. No *exact_mig*, por exemplo, a variável $T$ representa os conceitos da álgebra majoritária.

Como exemplo, tem-se $T$ representando um problema de programação linear. Em um problema de programação linear as restrições são formadas por equações ou inequações de variáveis inteiras. Sendo assim, o conjunto de restrições $K$ deve seguir esse formato. Respeitando $T$, tem-se $K = \{3x + 2y + z < 14; 5x + 3z > 10; y + z = 3\}$.

Um exemplo de um modelo de valores inválido ($UNSAT$) seria $M = \{x = 4, y = 2, z = 1\}$, pois a restrição $3x + 2y + z < 14$ não foi respeitada, tendo em vista que $(3 \cdot 4) + (2 \cdot 2) + 1 = 17$.

Um exemplo de um modelo de valores válido ($SAT$) seria $M = \{x = 2, y = 2, z = 1\}$, pois todas as restrições em $K$ foram respeitadas.

## 3.2 CODIFICAÇÃO DE VARIÁVEIS E RESTRIÇÕES

Nesta seção apresenta-se a codificação das variáveis e restrições que estruturam o algoritmo *exact_mig*. A codificação é dividida em duas partes. A primeira parte se baseia na codificação de um $MIG$ base, que será utilizado posteriormente para gerar um modelo

de valores válido. A segunda parte se baseia na codificação do conjunto de restrições $K$, que formula a teoria $T$, sendo $T$ os conceitos da álgebra majoritária. Um modelo de valores é considerado válido se respeitar todas as restrições existentes no conjuto $K$ (SOEKEN *et al.*, 2017).

### 3.2.1 Codificação de um $MIG$ base

Como passo inicial, tem-se a codificação das variáveis $f$, $r$ e $d$. A variável $f$ representa a tabela verdade de entrada, que deve ser coberta pela função de saída. Essa representação é feita com o uso de um vetor de $2^n$ elementos, sendo $n$ a quantidade de variáveis de entradas no problema. Cada posição do vetor $f$, que vai de 0 até $(2^n) - 1$, recebe um valor binário e as posições atribuidas ao valor 1 representam os mintermos cobertos pela função. Dessa forma, um vetor $f = [00010101]$ representa a cobertura dos mintermos $\{3, 5, 7\}$, pois os valores 1 estão nas posições 3, 5 e 7 do vetor $f$.

A variável $r$ representa a quantidade de nós utilizada na estruturação do $MIG$ base, sendo cada nó uma operação majoritária. A variável $d$ representa a profundidade máxima que um $MIG$ base pode atingir. Sendo assim, as variáveis $r$ e $d$ definem o critério de custo utilizado pelo $exact\_mig$, que busca a minimização da quantidade de operadores e de níveis em uma função, possibilitando a escolha de qual desses parâmetros será priorizado. Neste trabalho utiliza-se a síntese do $exact\_mig$ que prioriza a minimização de níveis.

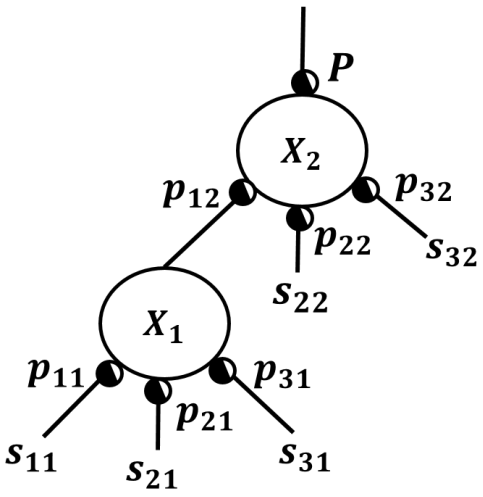
Para cada nó existente no $MIG$ base, representado por $X_i$, adiciona-se as variáveis $s_{ci}$ e $p_{ci}$. O índice $c$ representa as entradas de cada nó. Como em um $MIG$ existem apenas operadores majoritários de 3 entradas, tem-se $1 \leq c \leq 3$. O índice $i$ representa o nó à qual as variáveis pertencem, logo $1 \leq i \leq r$. Sendo assim, o nó raiz é representado por $X_r$.

As variáveis $s_{ci}$ representam as atribuições de valores para as variáveis de entrada de cada nó $X_i$. Para uma codificação em que as variáveis de entrada são $\{A, B, C\}$, as variáveis $s_{ci}$ podem receber os valores constantes (0 ou 1) ou as próprias variáveis de entrada $\{A, B, C\}$.

Para cada $MIG$ base, adiciona-se a variável $P$. As variáveis $p_{ci}$ e $P$ são valores binários que representam a polarização de $s_{ci}$ e de $X_r$ (nó raiz), respectivamente. Uma atribuição $p_{ci} = 0$ significa que a variável $s_{ci}$ está complementada, e uma atribuição $P = 0$ significa que a saída do $MIG$ base tem complemento.

Na Figura 3, tem-se um exemplo de um $MIG$ base codificado em que $d = 2$ e $r = 2$.

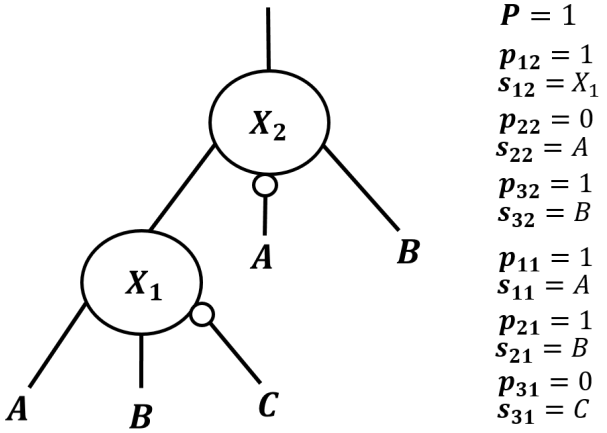
**Figura 3** – Codificação de um $MIG$ base.




**Fonte:** Adaptado de Soeken (2017).

Nota-se que variáveis de polarização sem um valor definido são representadas por um círculo de duas cores. Na Figura 4, tem-se um modelo de valores aplicado ao mesmo $MIG$ base da Figura 3. Para $X_1$, atribui-se $s_{c1} = \{A, B, C\}$ e $p_{c1} = \{1, 1, 0\}$. Para $X_2$, atribui-se para $s_{c2} = \{X_1, A, B\}$ e $p_{c2} = \{1, 0, 1\}$. Para a variável $P$, atribui-se $P = 1$, formando assim a função de saída $M(M(A, B, \overline{C}), \overline{A}, B)$.

**Figura 4** – Função $M(M(A, B, \overline{C}), \overline{A}, B)$ representada no formato $MIG$.




**Fonte:** Próprio autor

Para concluir a codificação de um problema, deve-se codificar, além do $MIG$ base, o conjunto de restrições $K$. No *exact_mig*, as restrições presentes em $K$ podem ser divididas em 3 categorias: restrições de operadores, restrições de profundidade e uma única restrição de satisfatibilidade.

### 3.2.2 Restrições de operadores

Restrições de operadores são restrições que se aplicam somente aos nós de um $MIG$ base. Na Equação 14, pode-se observar a restrição que garante a ordenação das variáveis $s_{ci}$. Essa ordenação é importante pois define o posicionamento das entradas em cada operador.

$$(s_{1i} < s_{2i}) \cdot (s_{2i} < s_{3i}) \tag{14}$$

Na Equação 15, apresenta-se a restrição que garante que no máximo uma entrada esteja complementada em cada operador. Essa restrição é importante por que a quantidade de inversores também possui custo na construção de um circuito majoritário. Nota-se que o *exact_mig* não busca uma solução ótima para a quantidade de inversores, o algoritmo apenas restringe a quantidade máxima de inversores por operador.

$$(p_{1i} + p_{2i}) \cdot (p_{1i} + p_{3i}) \cdot (p_{2i} + p_{3i}) \tag{15}$$

Para determinar se um modelo de valores é válido, todos os elementos da tabela verdade $f$ devem ser verificados. Dessa forma, deve-se adicionar as variáveis $a_{ci}^{(t)}$ e $b_i^{(t)}$. O índice $(t)$ representa um termo da tabela verdade $f$, logo $0 \leq (t) \leq (2^n) - 1$.

As variáveis $a_{ci}^{(t)}$ representam cada $s_{ci}$ aplicada à sua respectiva polarização $p_{ci}$ para um determinado termo $(t)$ de $f$. Na Equação 16, apresenta-se a restrição que formula essa relação, garantindo que os valores sejam propagados corretamente no $MIG$ base.

$$a_{ci}^{(t)} = (s_{ci} \oplus \overline{p_{ci}}) \tag{16}$$

As variáveis $b_i^{(t)}$ representam o valor de saída de cada nó $X_i$ para um termo $(t)$ específico. Sendo assim, $b_i^{(t)}$ representa uma operação majoritária entre as variáveis $a_{ci}^{(t)}$, como é possível observar na restrição apresentada na Equação 17.

$$b_i^{(t)} = M(a_{1i}^{(t)}, a_{2i}^{(t)}, a_{3i}^{(t)}) \tag{17}$$

### 3.2.3 Restrições de profundidade

Para formular as restrições de profundidade são adicionadas as variáveis $l_{ci}$ e $L_i$, para cada nó $X_i$. As variáveis $l_{ci}$ representam o nível de uma saída $c$ para cada nó $X_i$. A variável $L_i$ representa a profundidade de um nó $X_i$.

A restrição apresentada na Equação 18 garante a correta atribuição da profundidade $L_i$ para um nó $X_i$. Pode-se observar que a profundidade de $X_i$ é igual ao maior nível de suas variáveis $l_{ci}$ respectivas.

$$L_i = max\left\{l_{1i}, l_{2i}, l_{3i}\right\} \tag{18}$$

A restrição apresentada na Equação 19 garante que o limite de níveis $d$ não seja ultrapassado. Nota-se que $L_r$ se refere à profundidade do nó $X_r$, ou seja, o nó de maior índice (raiz).

$$L_r \leq d \tag{19}$$

### 3.2.4 Restrição de satisfatibilidade

A restrição de satisfatibilidade, apresentada na Equação 20, garante que o modelo de valores representado pelo $MIG$ base efetue a cobertura de $f$.

$$f^{(t)} = (b_r^{(t)} \oplus \overline{P}) \tag{20}$$

Pode-se observar que, para todos os mintermos $(t)$ de $f$, a variável de saída do nó $X_r$ (representada por $b_r^{(t)}$), aplicada à polarização geral $P$ deve ser igual ao valor de $f$ na posição de $(t)$. Sendo assim, um modelo de valores deve gerar uma tabela verdade idêntica à $f$.

Para exemplificar a aplicação da Equação 20, considera-se $f = [00101001]$ e $(t) = 2$, de forma que $f^{(2)} = 1$. Para codificação do $MIG$ base tem-se como exemplo $r = 1$, $d = 1$, $P = 0$ e as variáveis $s_{ci} = \{A, B, C\}$ e $p_{ci} = \{1, 0, 1\}$, gerando o modelo de valores $\overline{M}(A, \overline{B}, C)$.

O código binário do termo $(t) = 2$ é 010 (para $n = 3$), em que $A = 0$, $B = 1$ e $C = 0$. Dessa forma, $b_r^{(2)} = M(A, \overline{B}, C) = M(0, 0, 0) = 0$. Aplicando a Equação 20,

tem-se $(b_r^{(2)} \oplus \overline{P}) = (0 \oplus \overline{0}) = 1$, que é igual à $f^{(2)}$. Para o termo $(t) = 2$, o modelo de valores gerou um resultado igual à $f^{(2)}$, mas um modelo de valores só será válido se retornar um resultado igual à $f^{(t)}$ para todos os mintermos de $f$.

## 3.3 DESCRIÇÃO DO ALGORITMO $EXACT\_MIG$

Nesta seção apresenta-se a descrição do algoritmo *exact_mig*. Na Figura 5, é possível observar o *exact_mig* descrito em um pseudocódigo.

**Figura 5** – Descrição do algoritmo *exact_mig*.

```
Entrada: Tabela verdade (f)
Saída:   Função majoritária otimizada (Z)
1.  SAT = falso;
2.  d = 0;
3.  enquanto (SAT == falso) faça
4.  |   para (r = 0; r ≤ (3^d − 1)/2; r++) faça
5.  |   |   se (SMT(f, r, d) == verdadeiro) então
6.  |   |   |   retorne Z;
7.  |   |   |   SAT = verdadeiro;
8.  |   |   fim
9.  |   fim
10. |   d = d + 1;
11. |   SAT = falso;
12. fim
```

**Fonte:** Adaptado de Soeken (2017).

Como entrada tem-se uma função booleana $f$ representando os mintermos que devem ser cobertos. Como saída tem-se uma função majoritária otimizada que cobre $f$, representada por $Z$. O primeiro passo do algoritmo é a atribuição do valor 0 para a variável $d$, que representa a quantidade de níveis no $MIG$ base. Essa atribuição torna possível a cobertura de casos em que $f$ pode ser satisfeito com uma constante ou com uma única variável, sem a necessidade de um operador majoritário.

O algoritmo é composto por 2 laços de repetição. O primeiro deles efetua a incrementação da quantidade de níveis $d$ sempre que o segundo laço falhar em encontrar um resultado $SAT$. No segundo laço, a variável $r$ representa a quantidade de operadores que será distribuída em um $MIG$ de $d$ níveis. O processo SMT($f$,$r$,$d$) representa a aplicação do solucionador $SMT$ à codificação de restrições formulada pelo *exact_mig*. Para cada retorno $UNSAT$, a variável $r$ é incrementada e o algoritmo busca a resolução de SMT($f$,$r$,$d$) utilizando um $MIG$ com um nó adicional.

Para cada quantidade de níveis $d$ busca-se resoluções considerando $0 \leq r \leq ((3^d - 1)/2)$, em que $((3^d-1)/2)$ é a quantidade máxima de operadores possíveis em um $MIG$ de $d$ níveis. Caso não seja encontrado um resultado válido, o valor da variável $d$ é incrementado e o algoritmo volta ao primeiro laço de repetição considerando uma quantidade maior de níveis na formação da função de saída. Se um resultado $SAT$ for encontrado, então o algoritmo atingiu seu objetivo e o resultado é retornado como a saída $Z$.

Neste capítulo apresentou-se o *exact_mig*, visando explicar o funcionamento do algoritmo utilizado como parâmetro de comparação para o algoritmo $MPC$, abordado no próximo capítulo. Abordou-se o critério de custo utilizado, em que prioriza-se a quantidade de níveis seguida da quantidade de operadores, e o conceito de $MIG$s, utilizados para representar funções majoritárias na formulação de um modelo de valores válido. Abordou-se também as restrições que compõem o conjuto $K$, utilizado para validar o modelo de valores gerado pelo *exact_mig*.

# 4 ALGORITMO $MPC$

Neste capítulo apresenta-se o algoritmo $MPC$ (*Majority Primitives Combination*), proposto neste trabalho. Assim como o *exact_mig*, o $MPC$ recebe como entrada uma tabela verdade $f$ e retorna como saída uma função majoritária que cobre os mesmos mintermos. O critério de custo também é semelhante: em ambos busca-se a função majoritária com a menor quantidade de níveis, seguida da menor quantidade de operadores. No $MPC$, entretanto, tem-se a quantidade de inversores e de literais como terceiro e quarto critérios de custo, respectivamente.

Para gerar uma função de saída válida utiliza-se a expressão $M(X_1, X_2, X_3)$. Cada variável $X_c$, sendo $1 \leq c \leq 3$, representa uma função majoritária primitiva ou uma função majoritária de 2 níveis. Por se tratar de um método exaustivo, o algoritmo é viável apenas para funções com no máximo 5 variáveis de entrada e 4 níveis.

## 4.1 FORMAÇÃO DE TABELAS

O primeiro passo do $MPC$ é a fase de formação de tabelas, em que são construídas a tabela de funções primitivas e a tabela $M_2$, formada a partir da combinação de funções primitivas. O algoritmo recebe como entrada uma tabela verdade $f$, identifica a quantidade de entradas $n$, e gera a tabela de funções primitivas a partir dos conjuntos $U$, $V$, $G$ e $T$. Também são registrados os mintermos cobertos por cada função primitiva, sendo todas as funções primitivas a solução ótima para cobertura de suas respectivas tabelas verdade.

A tabela $M_2$ é formada pela aplicação de todas as combinações possíveis de funções primitivas à formula $M(X_1, X_2, X_3)$. Para cada função gerada também registra-se o conjunto de mintermos coberto. Caso existam duas funções que cubram os mesmos mintermos, a função de menor custo é mantida e a outra é descartada. Dessa forma, a tabela $M_2$ lista todos os conjuntos de mintermos que devem ser cobertos por uma função majoritária de 2 níveis. Como exemplo de uma função da tabela $M_2$, tem-se

$M(X_1, X_2, X_3) = M(A, M(A, \overline{B}, 0), \overline{M}(A, B, C))$, sendo $X_1 = A$, $X_2 = M(A, \overline{B}, 0)$ e $X_3 = \overline{M}(A, B, C)$.

Por serem obtidas com um método exaustivo, as funções da tabela $M_2$, assim como as funções primitivas, serão sempre a solução ótima para suas respectivas tabelas verdade.

Para garantir a minimização de inversores, as funções primitivas de um único operador seguem 4 possíveis padrões:

- $M(A, B, C)$, sem inversores;
- $M(\overline{A}, B, C)$, apenas uma das entradas com inversão;
- $\overline{M}(A, B, C)$, apenas um inversor aplicado ao valor de saída do operador;
- $\overline{M}(\overline{A}, B, C)$, uma das entradas e a saída do operador com inversão.

É importante destacar que nos casos em que apenas duas entradas possuem inversão, mesmo que a quantidade de inversores permaneça igual, é melhor negar a saída da operação e uma das entradas, sendo que $M(\overline{A}, B, \overline{C}) = \overline{M}(A, \overline{B}, C)$. Essa padronização permite a aplicação de $\Omega.I$ para minimizar a quantidade de inversores na formação de funções com 2 níveis ou mais. Como exemplo, considera-se $M(\overline{M}(A, \overline{B}, C), \overline{D}, 0)$, que possui 2 níveis, 2 operadores e 3 inversores. Com a aplicação de $\Omega.I$ tem-se $M(\overline{M}(A, \overline{B}, C), \overline{D}, 0) = \overline{M}(M(A, \overline{B}, C), D, 1)$, que possui a mesma quantidade de operadores e níveis, mas possui um inversor a menos.

É importante destacar também que operadores repetidos não são considerados no cálculo do custo. Na função $M(M(0, A, \overline{C}), \overline{M}(1, A, M(B, C, D)), M(1, C, M(B, C, D)))$ por exemplo, tendo em vista que o operador $M(B, C, D)$ aparece duas vezes, conta-se um total de 5 operadores apenas.

O total de conjuntos de mintermos, representado pela variável $S$, é calculado por $2^m$, sendo $m$ a quantidade de mintermos em $f$. Nota-se que $m = 2^n$.

Para $n = 3$, tem-se $S = 256$. A tabela de primitivos cobre 40 desses conjuntos. Os 216 restantes são cobertos por funções da tabela $M_2$. Dessa forma, $S$ pode ser coberto completamente por funções majoritárias de 2 ou menos níveis, o que torna a fase de formação de tabelas o suficiente para a obtenção de todas as soluções ótimas para $n = 3$.

Para $n = 4$, tem-se $S = 65.536$, sendo 90 desses conjuntos referentes à funções com um único ou nenhum operador majoritário, que são cobertas pela tabela de funções primitivas.

Na formação da tabela $M_2$, apenas 10.260 conjuntos podem ser cobertos por funções majoritárias de 2 níveis. Para os 55.186 conjuntos restantes, 55.184 podem ser cobertos por funções majoritárias de 3 níveis. Apenas 2 conjuntos precisam de uma função majoritária de 4 níveis. A cobertura de cada um desses casos é feita com a aplicação de uma síntese específica.

Para $n = 5$, $S = 4.294.967.296$ e apenas 172 desses conjuntos podem ser cobertos por funções primitivas. A tabela $M_2$ armazena os 253.560 conjuntos que podem ser cobertos por uma função majoritária de 2 níveis. Os conjuntos restantes precisam de 3 níveis ou mais para serem cobertos.

## 4.2 SÍNTESE PARA FUNÇÕES MAJORITÁRIAS DE 3 NÍVEIS

Nesta seção apresenta-se a síntese utilizada pelo algoritmo $MPC$ na construção de funções majoritárias de 3 níveis. O objetivo dessa síntese é montar uma função majoritária $M(X_1, X_2, X_3)$ a partir da combinação de funções primitivas ou da tabela $M_2$, que cubra os mesmos mintermos cobertos pela tabela verdade de entrada $f$.

A síntese está dividida em 2 laços de repetição. Se uma função válida não for encontrada no primeiro laço, o segundo tem início. Para $n = 4$, o primeiro laço é responsável por encontrar funções em que até 2 elementos de $X_c$ são funções primitivas, enquanto o segundo é responsável pela cobertura das demais funções de 3 níveis, em que todos os elementos de $X_c$ são funções de 2 níveis da tabela $M_2$.

Para $n = 5$, o primeiro laço é responsável pela cobertura de funções em que 2 dos elementos de $X_c$ são funções primitivas. As demais funções, em que necessita-se de pelo menos duas funções de $M_2$, são encontradas no segundo laço de repetição.

O primeiro laço de repetição é composto pelos seguintes passos:

1. Todas as funções primitivas ou da tabela $M_2$ que não cobrem pelo menos um dos mintermos cobertos por $f$ são descartadas;

2. Forma-se uma nova tabela $P$, que será utilizada para a seleção de $X_1$ e $X_2$. A tabela $P$ é formada a partir da combinação de funções primitivas ou da tabela $M_2$, sendo as variáveis $p_1$ e $p_2$ referentes à primeira e à segunda função de um par, respectivamente.

   Para $n = 5$, utiliza-se somente pares em que $p_1$ e $p_2$ são funções primitivas.

Para $n = 4$, forma-se primeiramente uma tabela $P$ em que $p_1$ e $p_2$ são funções primitivas. Entretanto, se todas as funções em $P$ já tiverem sido selecionadas e nenhuma função de saída válida pôde ser encontrada, o algoritmo cria uma nova tabela $P$, em que $p_1$ é uma função primitiva e $p_2$ é uma função de $M_2$.

Para formação de $P$ considera-se apenas pares de funções em que:

- Todos os mintermos cobertos por $f$ são cobertos pelo menos uma vez;
- Apenas mintermos cobertos por $f$ são cobertos duas vezes.

3. Seleciona-se o par de funções de menor custo na tabela $P$, que ainda não tenha sido selecionado, como $X_1$ e $X_2$. Para $n = 4$, se nenhuma função de saída válida foi encontrada sendo $X_1$ e $X_2$ funções primitivas, o algoritmo retorna ao passo 2 e constrói uma nova tabela $P$ considerando $p_1$ como uma função primitiva e $p_2$ como uma função da tabela $M_2$;

4. Cria-se um vetor $v$ de $2^n$ posições $i$, que será utilizado para construir a tabela verdade de $X_3$. Cada posição é referente a um mintermo de $f$, sendo $0 \leq i \leq (2^n - 1)$.

   O vetor $v$ é atualizado de acordo com os mintermos cobertos por $X_1$ e $X_2$. Se um mintermo $i$ é coberto por ambas as funções, $v_i = 2$. Se for coberto por apenas uma das funções, $v_i = 1$. E se não for coberto por nenhuma das funções, $v_i = 0$.

   Como exemplo, tem-se $n = 4$, $f = \{0, 1, 5, 8\}$, $X_1 = \{0, 1, 4, 5\}$ e $X_2 = \{0, 1, 2, 8, 10\}$. Após ser atualizado, o vetor $v$ possui os valores apresentados na Tabela 21:

5. Forma-se o padrão de tabela verdade para $X_3$, representado pelo vetor $X_3f$. Posições onde $v_i = 2$ ou $v_i = 0$ são consideradas *don't care states* e representadas por $x$.

   Para posições em que $v_i = 1$ e $i$ também é coberto por $f$, tem-se $X_3f_i = 1$. Se $v_i = 1$ e $i$ não é coberto por $f$, tem-se $X_3f_i = 0$. Dessa forma, para o exemplo apresentado na Tabela 21, tem-se $X_3f = [xx0x01xx1x0xxxxx]$;

6. Seleciona-se como $X_3$, a função de menor custo em $M_2$ que respeite o padrão formado por $X_3f$. Se não existir nenhuma função válida para $X_3$, o algoritmo retorna ao passo 3 e seleciona um novo par de funções em $P$;

7. Com a seleção de $X_3$, uma saída válida $M(X_1, X_2, X_3)$ é formada. Para minimizar a quantidade de inversores, aplica-se $\Omega.I$ em todos os níveis da função construída. Se a aplicação de $\Omega.I$ gerar uma função com menos inversores, a função anterior é substituída. A função encontrada é armazenada em uma tabela de possíveis resultados, representada por $Z$, e passa-se ao passo 8;

**Tabela 21** – Exemplo de atualização do vetor $v$.

| **Mintermos** | $f$ | $X_1$ | $X_2$ | $v$ |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 2 |
| 1 | 1 | 1 | 1 | 2 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 1 | 1 |
| 9 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 1 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 |
| 13 | 0 | 0 | 0 | 0 |
| 14 | 0 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 | 0 |

**Fonte: Próprio autor**

8. O algoritmo retorna ao passo 3 e seleciona um novo par em $P$. O laço de repetição termina quando todos os pares existentes em $P$ forem selecionados como $X_1$ e $X_2$, e todas as funções de saída válidas encontradas forem armazenadas em $Z$. Por fim, o algoritmo retorna a função de menor custo existente em $Z$ como saída.

Para exemplificar uma iteração do primeiro laço de repetição considere $n = 4$ e $f = \{4, 5, 6, 9, 15\}$. Uma função de saída válida pode ser encontrada na iteração em que $X_1 = M(A, D, 0)$ e $X_2 = M(\overline{A}, B, C)$, $X_1$ cobrindo os mintermos $\{9, 11, 13, 15\}$ e $X_2$ cobrindo os mintermos $\{2, 3, 4, 5, 6, 7, 14, 15\}$. A Tabela 22 apresenta o vetor $v$ atualizado a partir desse par de funções.

Os mintermos considerados *don't care states*, em que $v_i = 2$ ou $v_i = 0$, são $\{0, 1, 8, 10, 12, 15\}$. Os mintermos em que $v_i = 1$ e $f_i = 1$ são $\{4, 5, 6, 9\}$, e os mintermos em que $v_i = 1$ e $f_i = 0$ são $\{2, 3, 7, 11, 13, 14\}$. Sendo assim, $X_3 f = xx001110x1x0x00x$.

Para seleção de $X_3$ procura-se, na tabela $M_2$, a função de menor custo que se encaixe no modelo de tabela verdade definido em $X_3 f$. Seleciona-se $X3 = \overline{M}(C, M(\overline{B}, D, 1), M(A, B, 0))$, que cobre os mintermos $\{0, 1, 4, 5, 6, 8, 9, 12\}$, tendo 1100111011001000 como tabela verdade. Dessa forma, tem-se $M(X_1, X_2, X_3) = M(M(A, D, 0), M(\overline{A}, B, C), \overline{M}(C, M(\overline{B}, D, 1), M(A, B, 0)))$.

A forma *dual* de $M(M(A, D, 0), M(\overline{A}, B, C), \overline{M}(C, M(\overline{B}, D, 1), M(A, B, 0)))$ é igual a

**Tabela 22** – Exemplo do vetor $v$ atualizado a partir de $X_1$ e $X_2$.

| **Mintermos** | $f$ | $X_1$ | $X_2$ | $v$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 1 | 0 | 1 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 0 | 1 | 1 |
| 7 | 0 | 0 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 |
| 9 | 1 | 1 | 0 | 1 |
| 10 | 0 | 0 | 0 | 0 |
| 11 | 0 | 1 | 0 | 1 |
| 12 | 0 | 0 | 0 | 0 |
| 13 | 0 | 1 | 0 | 1 |
| 14 | 0 | 0 | 1 | 1 |
| 15 | 1 | 1 | 1 | 2 |

**Fonte: Próprio autor**

$\overline{M}(\overline{M}(\overline{A},\overline{D},1),\overline{M}(A,\overline{B},\overline{C}),M(\overline{C},\overline{M}(B,\overline{D},0),\overline{M}(\overline{A},\overline{B},1)))$, que possui uma quantidade maior de inversores. Sendo assim, para $f = \{4,5,6,9,15\}$, o algoritmo $MPC$ adiciona $M(M(A,D,0),M(\overline{A},B,C),\overline{M}(C,M(\overline{B},D,1),M(A,B,0)))$ à tabela de possíveis resultados $Z$. O laço de repetição se encerra apenas quando todos os pares em $P$ forem combinados com uma função de $M_2$, e retorna a função de menor custo em $Z$ como saída.

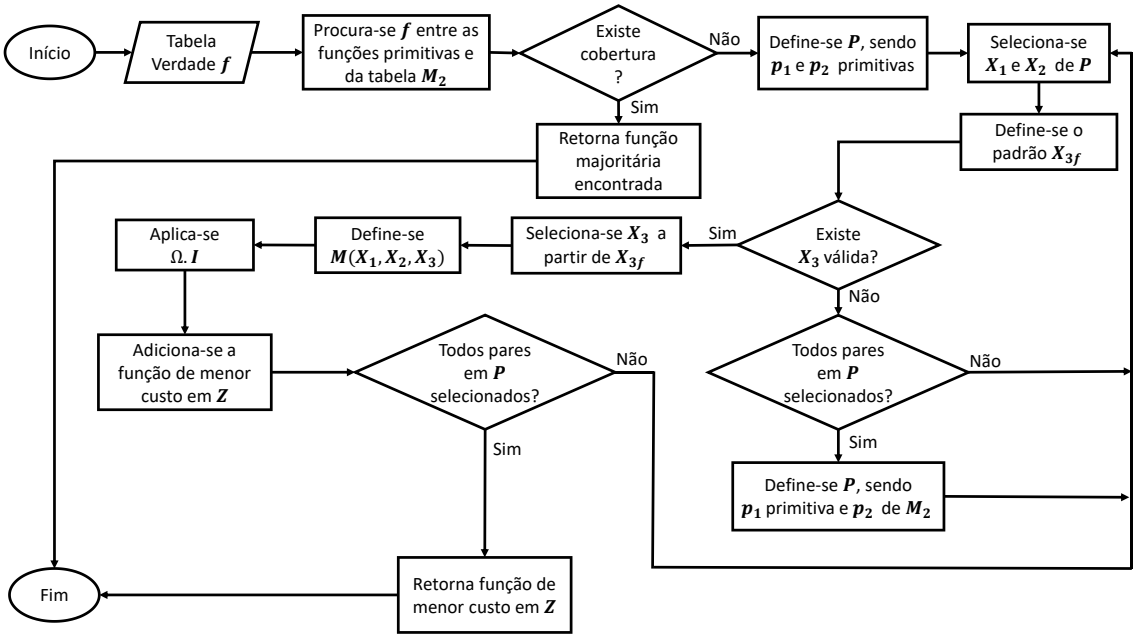
Nas Figuras 6 e 7, apresenta-se os fluxogramas do primeiro laço de repetição para $n = 4$ e $n = 5$, respectivamente.

Para $n = 4$, de todos os 55.184 conjuntos de mintermos que precisam de 3 níveis para serem cobertos, 50.016 podem ser cobertos por funções onde 2 elementos de $X_c$ são funções primitivas, e 5.056 podem ser cobertos por funções em que apenas um dos elementos de $X_c$ é uma função primitiva. Os outros 112 conjuntos podem ser cobertos apenas por funções em que todos os elementos de $X_c$ são funções de 2 níveis da tabela $M_2$. Esses casos são cobertos pelo segundo laço de repetição, que é composto pelos seguintes passos:

1. Seleciona-se como $X_1$ a função de menor custo, entre as funções primitivas ou da tabela $M_2$, que cubra pelo menos um mintermo de $f$. Para $n = 4$, entretanto, seleciona-se apenas funções da tabela $M_2$;

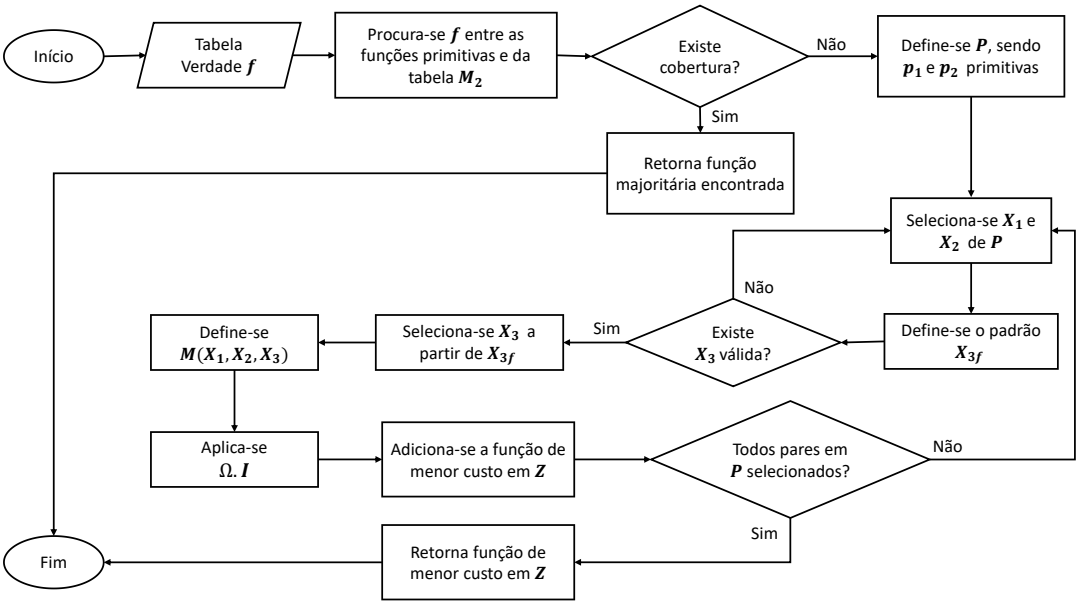
2. Cria-se um vetor $v$ de $2^n$ posições $i$, que será utilizado para criar os critérios para seleção de $X_2$ e $X_3$, a partir de $X_1$. Cada posição de $v$ é referente a um mintermo de $f$, sendo $0 \leq i \leq (2^n - 1)$, e começa com o valor 0. Se um mintermo $i$ for coberto

**Figura 6** – Fluxograma do primeiro laço de repetição para $n = 4$.




**Fonte:** Próprio autor.

**Figura 7** – Fluxograma do primeiro laço de repetição para $n = 5$.




**Fonte:** Próprio autor.

por $X_1$ e por $f$, tem-se $v_i = v_i + 1$. Se for coberto apenas por $X_1$, tem-se $v_i = -1$;

3. A partir de $v$ cria-se 2 novos vetores, $v_0$ e $v_{-1}$. O vetor $v_0$ guarda as posições referentes à mintermos cobertos por $f$ em que $v_i = 0$. O vetor $v_{-1}$ guarda todas as posições em que $v_i = -1$;

4. Seleciona-se como $X_2$ a função de menor custo da tabela $M_2$ que respeite os seguintes requisitos:

   - Cubra todos os mintermos em $v_0$;
   - Não cubra nenhum dos mintermos em $v_{-1}$.

   Se uma função $X_2$ válida não pôde ser encontrada, o algoritmo retorna ao passo 1 e seleciona uma nova função como $X_1$;

5. Atualiza-se o vetor $v$, tendo $X_2$ como base. Da mesma forma, para um mintermo $i$ coberto por $f$ e por $X_2$, tem-se $v_i = v_i + 1$. E para um mintermo $i$ coberto apenas por $X_2$, tem-se $v_i = -1$;

6. A partir de $v$ atualiza-se o vetor $v_{-1}$, que guarda todas as posições em que $v_i = -1$, e cria-se um novo vetor $v_1$, que armazena os mintermos onde $v_i = 1$;

7. Seleciona-se como $X_3$ a função de menor custo da tabela $M_2$ que respeite os seguintes requisitos:

   - Cubra todos os mintermos em $v_1$;
   - Não cubra nenhum dos mintermos em $v_{-1}$.

   Se uma função $X_3$ válida não pôde ser encontrada, o algoritmo retorna ao passo 4 e seleciona uma nova função como $X_2$. Com a seleção de $X_3$ tem-se uma função de saída válida;

8. Aplica-se $\Omega.I$ em todos os níveis de $M(X_1, X_2, X_3)$, gerando $\overline{M}(\overline{X_1}, \overline{X_2}, \overline{X_3})$. Para $n = 5$, o algoritmo retorna a função com a menor quantidade de inversores como saída e é concluído. Para $n = 4$, a função com a menor quantidade de inversores é adicionada à tabela de possíveis resultados $Z$, a quantidade de operadores em $X_1$ é armazenada em uma variável $r$ e passa-se ao passo 9;

9. O algoritmo retorna ao passo 1 e seleciona uma nova função $X_1$ em $M_2$. O laço de repetição termina quando todas as funções em $M_2$, com a mesma quantidade $r$ de operadores, tenham sido selecionadas como $X_1$. O algoritmo retorna a função de menor custo em $Z$ como saída.

Para exemplificar o segundo laço de repetição, considere $n = 5$ e $f = \{2, 4, 6, 7, 8, 11, 13, 14, 15\}$. Uma função de saída válida pode ser encontrada na iteração em que $X_1 = C$, cobrindo os mintermos $\{4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, 28, 29, 30, 31\}$.

Atualizando o vetor $v$ a partir de $X_1$ tem-se $v_0 = \{2, 8, 11\}$ e $v_{-1} = \{5, 12, 20, 21, 22, 23, 28, 29, 30, 31\}$. Na tabela 23, pode-se observar o vetor $v$ atualizado após a seleção de $X_1$.

**Tabela 23** – Vetor $v$ atualizado a partir de $X_1$, para o segundo laço de repetição.

| **Mintermos** | $\boldsymbol{f}$ | $\boldsymbol{X_1}$ | $\boldsymbol{v}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 |
| 3 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 |
| 5 | 0 | 1 | -1 |
| 6 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 |
| 9 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 |
| 11 | 1 | 0 | 0 |
| 12 | 0 | 1 | -1 |
| 13 | 1 | 1 | 1 |
| 14 | 1 | 1 | 1 |
| 15 | 1 | 1 | 1 |
| 16 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 |
| 20 | 0 | 1 | -1 |
| 21 | 0 | 1 | -1 |
| 22 | 0 | 1 | -1 |
| 23 | 0 | 1 | -1 |
| 24 | 0 | 0 | 0 |
| 25 | 0 | 0 | 0 |
| 26 | 0 | 0 | 0 |
| 27 | 0 | 0 | 0 |
| 28 | 0 | 1 | -1 |
| 29 | 0 | 1 | -1 |
| 30 | 0 | 1 | -1 |
| 31 | 0 | 1 | -1 |

**Fonte: Próprio autor**

Deve-se então selecionar a função de menor custo da tabela $M_2$ que cubra todos os mintermos em $v_0$ e não cubra nenhum dos mintermos em $v_{-1}$.

Seleciona-se $X_2 = \overline{M}(M(C, D, 1), M(B, \overline{E}, 0), M(A, \overline{B}, E))$, que cobre os mintermos $\{0, 1, 2, 4, 6, 8, 9, 11, 13, 15, 16, 17, 24, 25\}$.

A partir de $X_2$, atualiza-se novamente o vetor $v$, gerando $v_1 = \{2, 7, 8, 11, 14\}$ e $v_{-1} = \{0, 1, 5, 9, 12, 16, 17, 20, 21, 22, 23, 24, 25, 28, 29, 30, 31\}$. Na tabela 24, tem-se o vetor

$v$ atualizado após a seleção de $X_2$.

**Tabela 24** – Vetor $v$ atualizado a partir de $X_2$, para o segundo laço de repetição.

| **Mintermos** | $f$ | $X_2$ | $v$ |
|---|---|---|---|
| 0 | 0 | 1 | -1 |
| 1 | 0 | 1 | -1 |
| 2 | 1 | 1 | 1 |
| 3 | 0 | 0 | 0 |
| 4 | 1 | 1 | 2 |
| 5 | 0 | 0 | -1 |
| 6 | 1 | 1 | 2 |
| 7 | 1 | 0 | 1 |
| 8 | 1 | 1 | 1 |
| 9 | 0 | 1 | -1 |
| 10 | 0 | 0 | 0 |
| 11 | 1 | 1 | 1 |
| 12 | 0 | 0 | -1 |
| 13 | 1 | 1 | 2 |
| 14 | 1 | 0 | 1 |
| 15 | 1 | 1 | 2 |
| 16 | 0 | 1 | -1 |
| 17 | 0 | 1 | -1 |
| 18 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 |
| 20 | 0 | 0 | -1 |
| 21 | 0 | 0 | -1 |
| 22 | 0 | 0 | -1 |
| 23 | 0 | 0 | -1 |
| 24 | 0 | 1 | -1 |
| 25 | 0 | 1 | -1 |
| 26 | 0 | 0 | 0 |
| 27 | 0 | 0 | 0 |
| 28 | 0 | 0 | -1 |
| 29 | 0 | 0 | -1 |
| 30 | 0 | 0 | -1 |
| 31 | 0 | 0 | -1 |

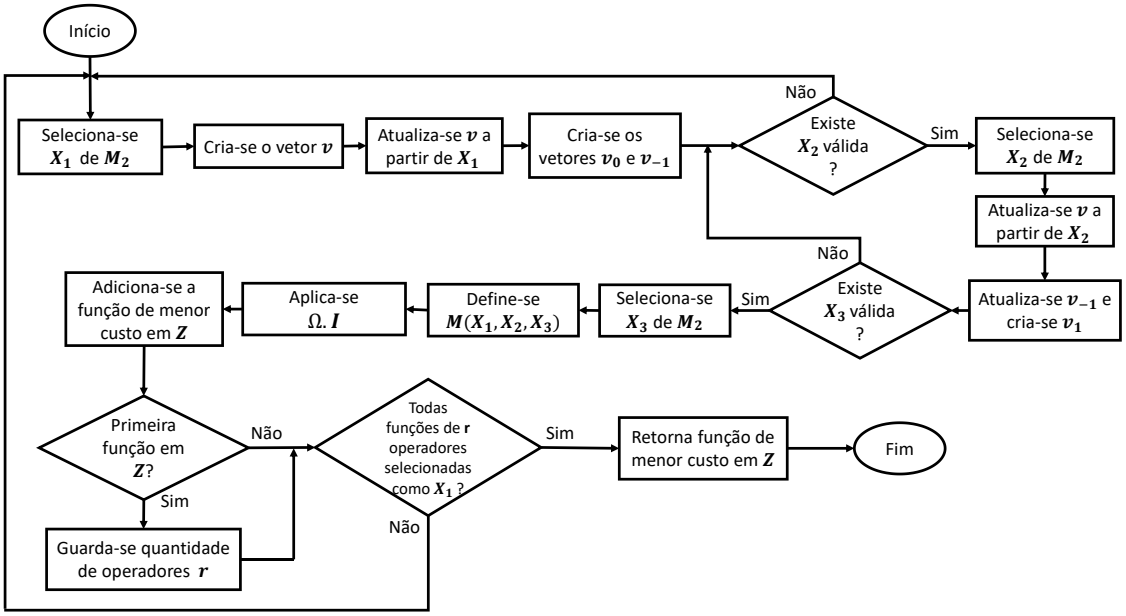
**Fonte: Próprio autor**

Como $X_3$, deve-se selecionar a função de menor custo da tabela $M_2$ que não cubra nenhum dos mintermos em $v_{-1}$ e que cubra todos os mintermos em $v_1$. Tem-se $X_3 = M(M(\overline{A}, D, 0), M(B, \overline{C}, 0), \overline{M}(A, B, E))$, que cobre os mintermos $\{2, 3, 6, 7, 8, 10, 11, 14\}$.

Com a seleção de $X_3$, o algoritmo $MPC$ retorna como saída $M(X_1, X_2, X_3) = M(C,\overline{M}(M(C, D, 1), M(B, \overline{E}, 0), M(A, \overline{B}, E)), M(M(\overline{A}, D, 0), M(B, \overline{C}, 0), \overline{M}(A, B, E)))$, que possui menos inversores que sua forma *dual*.

Na Figura 8 apresenta-se o fluxograma do segundo laço de repetição para $n = 4$.

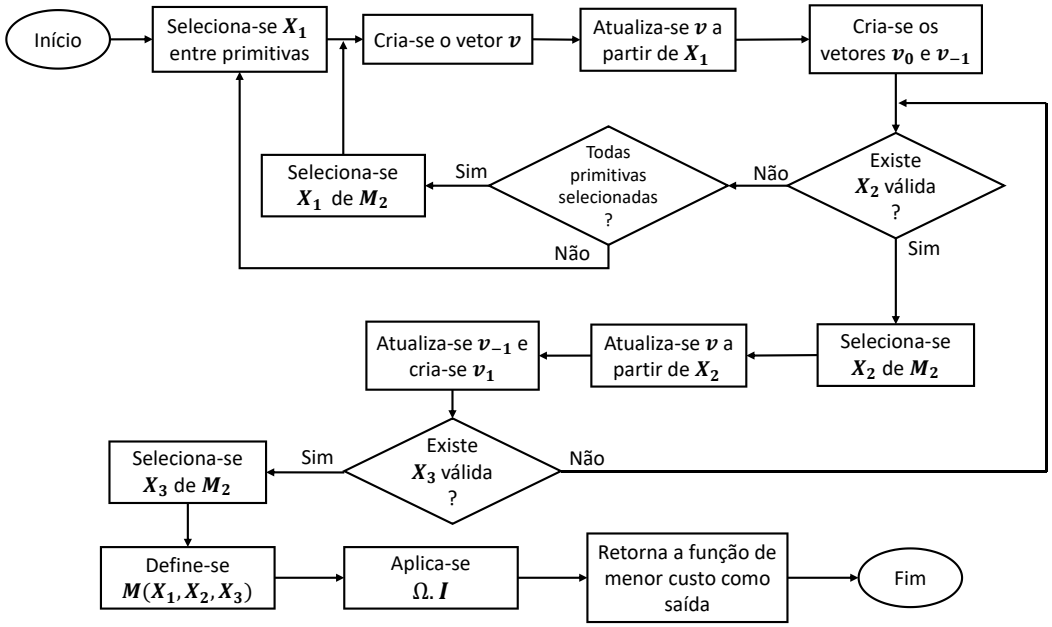
**Figura 8** – Fluxograma do segundo laço de repetição para $n = 4$.




**Fonte:** Próprio autor.

Na Figura 9 apresenta-se o fluxograma do segundo laço de repetição para $n = 5$.

**Figura 9** – Fluxograma do segundo laço de repetição para $n = 5$.




**Fonte:** Próprio autor.

## 4.3 SÍNTESE PARA FUNÇÕES MAJORITÁRIAS DE 4 NÍVEIS

Nesta seção apresenta-se a síntese utilizada pelo $MPC$ para formação de funções, sendo $n = 4$ ou $n = 5$, que podem ser cobertas apenas por funções majoritárias de 4 níveis. Para $n = 4$, esta síntese se aplica somente a 2 conjuntos de mintermos, os conjuntos correspondentes à função booleana clássica em que todas as variáveis de entrada estão conectadas a um operador $XOR$ e a essa função complementada. Na Tabela 25, é possível observar as duas funções clássicas e suas respectivas tabelas verdade.

**Tabela 25** – Funções clássicas que podem ser cobertas apenas por uma função majoritária de 4 níveis, sendo $n = 4$.

| **Função Clássica** | **Tabela Verdade** |
|---|---|
| $A \oplus B \oplus C \oplus D$ | [0110100110010110] |
| $\overline{A \oplus B \oplus C \oplus D}$ | [1001011001101001] |

**Fonte: Próprio autor**

O objetivo dessa síntese é formular $M(B, X_2, X_3)$. A atribuição $X_1 = B$ é justificada porque a tabela verdade de $B$ pode ser utilizada para definir o conjunto de mintermos que deve ser coberto pelas funções $X_2$ e $X_3$. A tabela verdade de $B$ é [0000111100001111].

Como é definido pelo axioma $\Omega.M$, cada mintermo deve ser coberto por no mínimo duas das funções $X_c$. Tendo $\Omega.M$ e a tabela verdade de $B$ como base, define-se a estrutura apresentada na Tabela 26. Nota-se que a estrutura divide cada tabela verdade em 4 subvetores, representados pela variável $Q_j$, em que $1 \leq j \leq 4$.

**Tabela 26** – Estrutura para formação da tabela verdade de $X_2$ e $X_3$.

| | $\boldsymbol{Q_1}$ | $\boldsymbol{Q_2}$ | $\boldsymbol{Q_3}$ | $\boldsymbol{Q_4}$ |
|---|---|---|---|---|
| $\boldsymbol{B}$ | 0000 | 1111 | 0000 | 1111 |
| $\boldsymbol{X_2}$ | 1111 | $----$ | $----$ | 0000 |
| $\boldsymbol{X_3}$ | $----$ | 0000 | 1111 | $----$ |

**Fonte: Próprio autor**

Pode-se observar que, a partir de $B$ e de $\Omega.M$, foi possível definir 2 dos 4 subvetores da tabela verdade de $X_2$ e $X_3$. A outra metade da tabela é definida pelos subvetores da tabela verdade de entrada $f$.

Como exemplo considera-se a função $A \oplus B \oplus C \oplus D$, em que $f = [0110100110010110]$ e seus subvetores são $Q_1 = [0110]$, $Q_2 = [1001]$, $Q_3 = [1001]$ e $Q_4 = [0110]$.

A tabela verdade de $X_2$, representada pela variável $X_2f$, é encontrada a partir da combinação de seus quadrantes predefinidos, $Q_1$ e $Q_4$, e dos quadrantes $Q_2$ e $Q_3$ de $f$.

Sendo assim, $X_2f = [1111100110010000]$.

De forma semelhante, a tabela verdade de $X_3$, representada por $X_3f$, pode ser obtida com a combinação de seus quadrantes predefinidos, $Q_2$ e $Q_3$, e dos quadrantes $Q_1$ e $Q_4$ de $f$, gerando $X_3f = [0110000011110110]$.

A seleção de $X_2$ e $X_3$ é feita com a síntese para funções de 3 níveis, selecionada de acordo com o valor de $n$, aplicada à sua respectiva tabela verdade ($X_2f$ e $X_3f$).

Neste capítulo apresentou-se a síntese do algoritmo $MPC$, visando o entendimento do algoritmo desenvolvido neste trabalho e concluindo a explicação dos algoritmos que serão comparados no capítulo seguinte. Abordou-se a lógica utilizada pelo $MPC$ na síntese de funções de 3 níveis, e abordou-se também a lógica utilizada na síntese de funções de 4 níveis, destacando as únicas duas funções de 4 entradas que precisam de 4 níveis para sua cobertura.

# 5 TESTES E RESULTADOS

Neste capítulo apresenta-se os resultados obtidos com a comparação entre os algoritmos $MPC$ e *exact_mig*. Os testes foram feitos a partir de um arquivo *.bash* que executou ambos os algoritmos para todos os conjuntos de mintermos desejados. O critério de custo utilizado nessa comparação é o mesmo utilizado pelo $MPC$, que prioriza a quantidade de níveis, seguida da quantidade de operadores, inversores e literais.

Para $n = 4$, os algoritmos foram executados para todas as 65.536 possíveis funções. Na Tabela 27 tem-se em cada coluna a quantidade de conjuntos pertencentes a um grupo $S_i$ específico. Cada $S_i$, sendo $0 \leq i \leq (2^n - 1)$, representa um grupo de funções que cobre uma quantidade específica de mintermos. $S_4$, por exemplo, representa todas as funções que cobrem 4 mintermos.

Na Equação 21, apresenta-se o cálculo de $S_i$. Nota-se que $m = 2^n$, representando o total de mintermos possíveis para uma determinada quantidade de entradas. Para $n = 4$, tem-se $m = 16$.

$$S_i = \frac{m!}{i! \cdot (m-i)!} \tag{21}$$

Para cada $S_i$, a tabela mostra a quantidade de conjuntos em que o $MPC$ gerou resultados com custo menor, maior e igual ao *exact_mig*.

Dessa forma, o algoritmo $MPC$ melhorou a cobertura de 42.987 (66%) funções, gerou resultados inferiores para 15.351 (23%) funções e resultados iguais para as outras 7.198 (11%), atingindo um total de 50.185 (77%) funções de custo menor ou equivalente às funções encontradas pelo algoritmo *exact_mig*.

É importante apontar que o $MPC$ consegue gerar melhores resultados porque o *exact_mig* não busca a otimização de inversores e literais, em sua síntese exata considera-se apenas a quantidade de níveis e operadores como critério de custo. Nessa comparação, as funções do *exact_mig* foram geradas com a priorização da minimização de níveis seguida da

**Tabela 27** – Comparação de custo entre $MPC$ e *exact_mig*.

| $i$ | $S_i$ | $MPC < exact\_mig$ | $MPC > exact\_mig$ | $MPC = exact\_mig$ |
|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 1 |
| **1** | 16 | 16 | 0 | 0 |
| **2** | 120 | 41 | 8 | 71 |
| **3** | 560 | 324 | 60 | 176 |
| **4** | 1.820 | 808 | 708 | 304 |
| **5** | 4.368 | 2.906 | 583 | 879 |
| **6** | 8.008 | 4.493 | 2.276 | 1.239 |
| **7** | 11.440 | 7.188 | 3.300 | 952 |
| **8** | 12.870 | 8.108 | 3.474 | 1.288 |
| **9** | 11.440 | 7.536 | 3.022 | 882 |
| **10** | 8.008 | 6.273 | 1.121 | 614 |
| **11** | 4.368 | 3.334 | 512 | 522 |
| **12** | 1.820 | 1.373 | 279 | 168 |
| **13** | 560 | 482 | 0 | 78 |
| **14** | 120 | 93 | 8 | 19 |
| **15** | 16 | 12 | 0 | 4 |
| **16** | 1 | 0 | 0 | 1 |
| **TOTAL** | **65.536** | **42.987** | **15.351** | **7.198** |

**Fonte: Próprio autor**

minimização de operadores, diferindo do critério de custo do $MPC$ apenas pela quantidade de inversores e literais como terceiro e quarto critérios.

É importante apontar também que as funções em que o $MPC$ retorna um resultado pior que o algoritmo *exact_mig* acontecem por que o $MPC$ preenche a expressão $M(X_1, X_2, X_3)$ priorizando $X_1$ e $X_2$ como funções primitivas, só atribuindo funções da tabela $M_2$ se necessário. Essa regra é essencial para a formação de funções minimizadas na grande maioria dos casos, porém existem casos em que a priorização da seleção de funções de 2 níveis pode gerar um custo menor. Como exemplo, considere $f = 1000000000000001$.

Para $f$, o algoritmo $MPC$ gera a função $M(\overline{M}(B, C, 1), M(B, D, 0), M(1, M(A, C, 0), \overline{M}(A, D, 1)))$, que possui 3 níveis, 6 operadores, 2 inversores e 8 literais. O termo literais refere-se ao número de vezes em que as variáveis de entrada aparecem na função, em suas formas complementadas ou não.

O algoritmo *exact_mig* gera a função $\overline{M}(M(0, C, \overline{D}), M(1, D, M(A, B, \overline{D})), \overline{M}(0, C, M(A, B, \overline{D})))$, que possui 3 níveis, 5 operadores, 5 inversores e 7 literais.

Nesse caso, o $MPC$ gerou um resultado priorizando o uso de duas funções primitivas ($X_1 = \overline{M}(B, C, 1)$ e $X_2 = M(B, D, 0)$), utilizando apenas uma função de 2 níveis ($X_3 =$

$M(1, M(A, C, 0), \overline{M}(A, D, 1)))$, enquanto o *exact_mig* conseguiu gerar melhores resultados utilizando uma única função primitiva e duas funções de 2 níveis, tendo em vista que a combinação dessas duas funções usa o operador $M(A, B, \overline{D})$ duas vezes, e desconsidera-se o custo de operadores repetidos.

Como exemplo de uma função em que o $MPC$ conseguiu melhores resultados, tem-se $f = 0000001001000100$, em que o algoritmo $MPC$ gera a função $M(M(A, \overline{C}, 0), M(C, D, 1), M(0, B, \overline{M}(A, D, 1)))$, que possui 3 níveis, 5 operadores, 2 inversores e 7 literais.

O algoritmo *exact_mig* gera a função $M(\overline{M}(0, \overline{B}, C), M(0, D, \overline{M}(\overline{A}, C, D)), M(\overline{D}, M(\overline{A}, C, D), M(0, \overline{B}, C))))$, que possui 3 níveis, 5 operadores, 5 inversores e 7 literais. Nota-se que o $MPC$ foi capaz de gerar uma função com o mesmo número de níveis, operadores e literais, porém com menos inversores.

Na Tabela 28, apresenta-se o tempo computacional médio e total de cada grupo de funções $S_i$ para ambos os algoritmos.

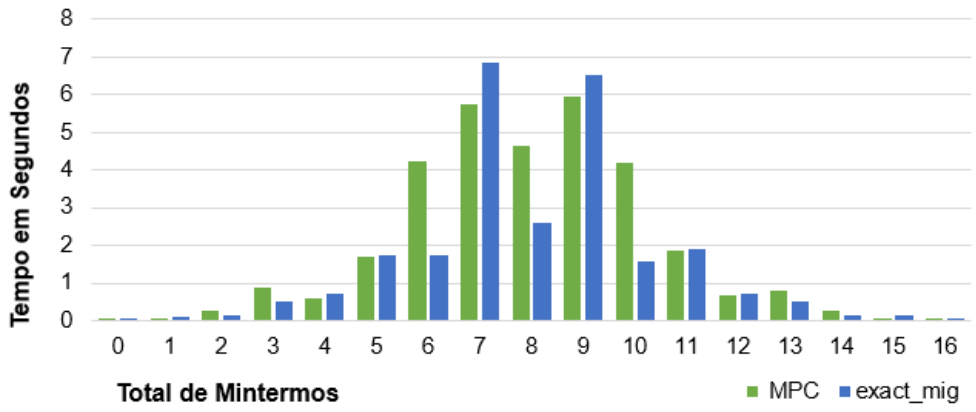
**Tabela 28** – Comparação de tempo computacional entre $MPC$ e *exact_mig*.

| | | ***MPC*** | | ***exact_mig*** | |
|---|---|---|---|---|---|
| $i$ | $S_i$ | Total | Média | Total | Média |
| 0 | 1 | 0,01 s | 0,01 s | 0,01 s | 0,01 s |
| 1 | 16 | 1,10 s | 0,06 s | 1,65 s | 0,10 s |
| 2 | 120 | 30,02 s | 0,25 s | 17,85 s | 0,14 s |
| 3 | 560 | 8,29 m | 0,88 s | 4,57 m | 0,49 s |
| 4 | 1.820 | 17,58 m | 0,58 s | 21,44 m | 0,70 s |
| 5 | 4.368 | 2,07 h | 1,70 s | 2,09 h | 1,72 s |
| 6 | 8.008 | 9,46 h | 4,25 s | 3,84 h | 1,73 s |
| 7 | 11.440 | 18,32 h | 5,76 s | 21,74 h | 6,84 s |
| 8 | 12.870 | 16,58 h | 4,63 s | 9,36 h | 2,61 s |
| 9 | 11.440 | 18,94 h | 5,96 s | 20,76 h | 6,53 s |
| 10 | 8.008 | 9,31 h | 4,18 s | 3,54 h | 1,59 s |
| 11 | 4.368 | 2,24 h | 1,84 s | 2,27 h | 1,89 s |
| 12 | 1.820 | 20,75 m | 0,68 s | 22,12 m | 0,73 s |
| 13 | 560 | 7,42 m | 0,79 s | 4,81 m | 0,51 s |
| 14 | 120 | 31,56 s | 0,26 s | 18,68 s | 0,15 s |
| 15 | 16 | 0,99 s | 0,06 s | 2,24 s | 0,14 s |
| 16 | 1 | 0,01 s | 0,01 s | 0,01 s | 0,01 s |
| **TOTAL** | **65.536** | **77,82 h** | **4,27 s** | **64,49 h** | **3,54 s** |

**Fonte: Próprio autor**

Nota-se que o $MPC$ possui um tempo computacional médio de 4,27 segundos, enquanto o *exact_mig* possui uma média de 3,54 segundos. Na Figura 10, tem-se o tempo computacional médio em formato de gráfico para cada grupo $S_i$.

**Figura 10** – Tempo computacional médio (em segundos) para $n = 4$.




**Fonte:** Próprio autor

Na Tabela 29, tem-se a utilização média de memória na síntese de cada grupo $S_i$, em *megabytes* (MB).

**Tabela 29** – Comparação de utilização média de memória entre $MPC$ e *exact_mig*.

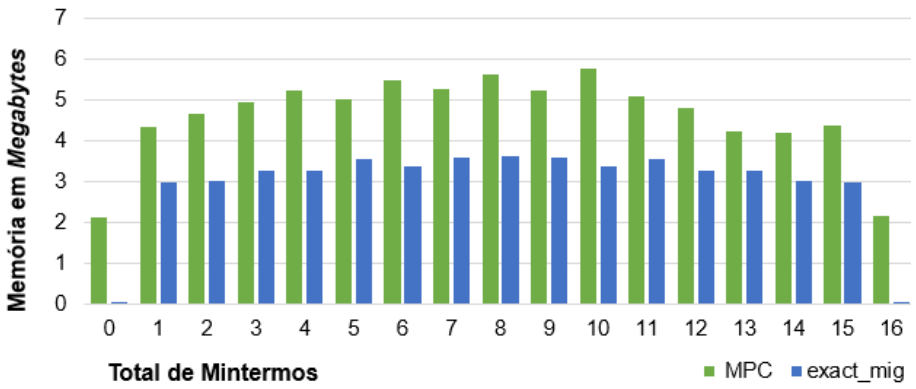
| $i$ | $S_i$ | $MPC$ | *exact_mig* |
|---|---|---|---|
| **0** | 1 | 3,36 MB | 0,01 MB |
| **1** | 16 | 4,33 MB | 3,00 MB |
| **2** | 120 | 4,67 MB | 3,01 MB |
| **3** | 560 | 4,97 MB | 3,28 MB |
| **4** | 1.820 | 5,25 MB | 3,26 MB |
| **5** | 4.368 | 5,03 MB | 3,55 MB |
| **6** | 8.008 | 5,49 MB | 3,39 MB |
| **7** | 11.440 | 5,28 MB | 3,61 MB |
| **8** | 12.870 | 5,62 MB | 3,63 MB |
| **9** | 11.440 | 5,23 MB | 3,61 MB |
| **10** | 8.008 | 5,79 MB | 3,38 MB |
| **11** | 4.368 | 5,10 MB | 3,55 MB |
| **12** | 1.820 | 4,81 MB | 3,27 MB |
| **13** | 560 | 4,23 MB | 3,28 MB |
| **14** | 120 | 4,19 MB | 3,02 MB |
| **15** | 16 | 4,38 MB | 3,00 MB |
| **16** | 1 | 3,37 MB | 0,01 MB |
| **TOTAL** | **65.536** | **5,36 MB** | **3,52 MB** |

**Fonte: Próprio autor**

Nota-se que o $MPC$ possui uma média de utilização igual à 5,36 MB, enquanto o *exact_mig* apresenta uma média de 3,52 MB. Na Figura 11, tem-se a utilização média de

memória em formato de gráfico para cada grupo $S_i$.

**Figura 11** – Memória média utilizada (em *megabytes*) para $n = 4$.

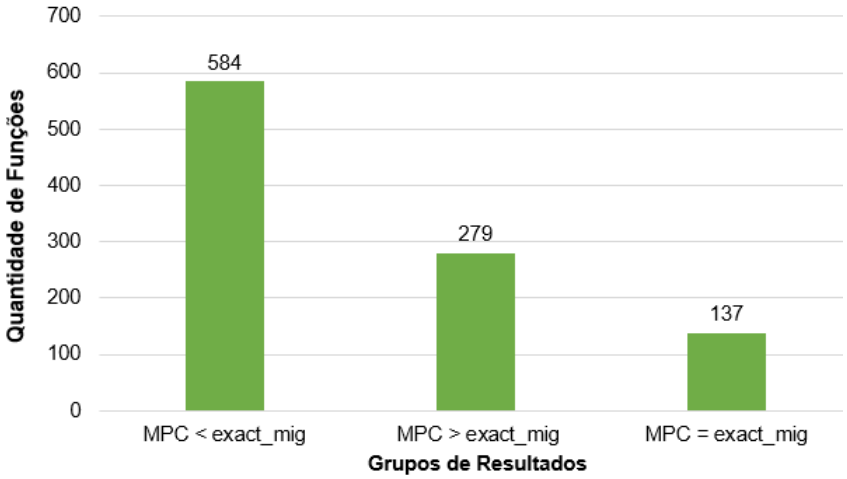



**Fonte:** Próprio autor

Para $n = 5$ utilizou-se uma amostra de 1.000 funções geradas aleatoriamente. O algoritmo $MPC$ gerou um custo menor para 584 (58%) funções e um custo equivalente para 137 (14%), atigindo um total de 721 (72%) funções de custo menor ou equivalente em relação às funções encontradas pelo *exact_mig*.

Na Figura 12, apresenta-se a comparação de custo em formato de gráfico.

**Figura 12** – Comparação de custo para $n = 5$.




**Fonte:** Próprio autor

O tempo computacional total do $MPC$ foi de 23,16 horas, com um tempo computacional médio de 1,38 minutos. O tempo computacional total do *exact_mig* foi de 59,33 horas e seu tempo médio foi de 3,55 minutos. Dessa forma, o algoritmo $MPC$ conseguiu um tempo computacional 56% menor em relação ao *exact_mig*.

A média de memória utilizada pelo $MPC$ foi de 45,41 MB, enquanto o *exact_mig* utilizou uma média menor, de apenas 5,05 MB. Todos os testes foram realizados em um

computador com 8 GB de RAM e um processador de 1.7 GHZ.

# 6 CONCLUSÕES

Neste trabalho desenvolveu-se o algoritmo $MPC$ (*Majority Primitives Combination*), que tem como objetivo a geração de funções majoritárias a partir de uma tabela verdade. O $MPC$ foi então comparado ao algoritmo *exact_mig*, tido como estado da arte na síntese de funções majoritárias.

Para funções com 4 variáveis, de um total de 65.536 funções possíveis, o algoritmo $MPC$ gerou um custo menor para 42.987 (66%) funções e um custo equivalente para 7.198 (11%), alcançando um total de 50.185 (77%) funções de custo menor ou equivalente em relação ao *exact_mig*. O $MPC$ teve um tempo computacional médio de 4,27 segundos e utiliza uma média de 5,36 MB de memória, enquanto o *exact_mig* teve um tempo computacional médio de 3,54 segundos e utiliza em média 3,52 MB de memória.

Para funções com 5 variáveis, de uma amostra de 1.000 funções, o algoritmo $MPC$ gerou um total de 721 (72%) funções com custo menor ou equivalente ao *exact_mig*, sendo 584 (58%) funções de menor custo e 137 (14%) de custo equivalente. O tempo computacional médio do $MPC$ foi de 1,38 minutos, sendo 56% mais rápido que o *exact_mig*, que teve um tempo computacional médio de 3,55 minutos. No quesito utilização de memória, os algoritmos $MPC$ e *exact_mig* tiveram uma média de 45,41 MB e 5,05 MB, respectivamente.

Como proposta para continuação deste trabalho tem-se a implementação de um novo algoritmo de síntese semelhante ao $MPC$, mas que também utilize operadores majoritários de 5 entradas na formação de funções. Trata-se de uma grande contribuição pois a utilização de operadores de 5 entradas ainda não foi abordada de forma abrangente no estado atual da arte, e espera-se que essa utilização possa reduzir a quantidade de níveis necessários para formação de uma função majoritária.

# REFERÊNCIAS

AKERS, S. B. A truth table method for the synthesis of combinational logic. **IRE Transactions on Electronic Computers**, Nova York, EC-10, n. 4, p. 604–615, 1961.

AMARU, L.; GAILLARDON, P.-E.; MICHELI, G. D. Majority-inverter graph: a novel data-structure and algorithms for efficient logic optimization. In: ANNUAL DESIGN AUTOMATION CONFERENCE, 51., 2014, São Franscisco. **Proceedings...** São Franscisco: IEEE. 2014. p. 1–6.

AMARU, L. G. **New data structures and algorithms for logic synthesis and verification**. Lausana: Springer, 2017. 60-61 p.

AVEDILLO, M. J.; QUINTANA, J. M.; RUEDA, A. **Threshold logic**. New York: Wiley Online Library, 1999. 3–5 p.

BERTACCO, V.; DAMIANI, M. The disjunctive decomposition of logic functions. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN - ICCAD, 15., 1997, San Jose. **Proceedings...** San Jose: IEEE. 1997. p. 78–82.

BOOLE, G. **An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities**. Cork: Dover Publications, 1854. 24-38 p.

CHATTOPADHYAY, A. *et al.* Notes on majority boolean algebra. In: MULTIPLE-VALUED LOGIC - ISMVL, 46., 2016, Sapporo. **Proceedings...** Sapporo: IEEE. 2016. p. 50–55.

CHU, Z. *et al.* Functional decomposition using majority. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, 23., 2018, Jeju. **Proceedings...** Jeju: IEEE. 2018. p. 676–681.

COHN, M.; LINDAMAN, R. Axiomatic majority-decision logic. **IRE Transactions on Electronic Computers**, Nova York, EC-10, n. 1, p. 17–21, 1961.

FLOYD, T. L. **Digital fundamentals**. 11. ed. Dallas: Pearson Education, 2015. 125–149 p.

GANZINGER, H. *et al.* Dpll (t): fast decision procedures. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, 16., 2004, Boston. **Proceedings...** Boston: Springer. 2004. p. 175–188.

GEORGE, A. K.; SINGH, H. Design of computing circuits using spatially localized dna majority logic gates. In: INTERNATIONAL CONFERENCE ON REBOOTING COMPUTING - ICRC, 1., 2017, Washington. **Proceedings...** Washington: IEEE. 2017. p. 1–7.

KARNAUGH, M. The map method for synthesis of combinational logic circuits. **Transactions of the American Institute of Electrical Engineers**, Texas, v. 72, n. 5, p. 593–599, 1953.

KONG, K.; SHANG, Y.; LU, R. An optimized majority logic synthesis methodology for quantum-dot cellular automata. **IEEE Transactions on Nanotechnology**, v. 9, n. 2, p. 170–183, 2010.

LINDAMAN, R. A theorem for deriving majority-logic networks within an augmented boolean algebra. **IRE Transactions on electronic computers**, EC-9, n. 3, p. 338–342, 1960.

MISHRA, V. K.; THAPLIYAL, H. Heuristic based majority/minority logic synthesis for emerging technologies. In: INTERNATIONAL CONFERENCE ON EMBEDDED SYSTEMS, 16., 2017, Hyderabad. **Proceedings...** Hyderabad: IEEE. 2017. p. 295–300.

MOURA, L. D.; BJØRNER, N. Z3: An efficient smt solver. In: INTERNATIONAL CONFERENCE ON TOOLS AND ALGORITHMS FOR THE CONSTRUCTION AND ANALYSIS OF SYSTEMS, 14., 2008, Budapeste. **Proceedings...** Budapeste: Springer. 2008. p. 337–340.

MOURA, L. de; DUTERTRE, B.; SHANKAR, N. A tutorial on satisfiability modulo theories. In: INTERNATIONAL CONFERENCE ON COMPUTER AIDED VERIFICATION, 19., 2007, Berlim. **Proceedings...** Berlim: IEEE. 2007. p. 20–36.

SASAO, T. **Switching theory for logic synthesis**. Berlim: Springer Science & Business Media, 2012. 93-106 p.

SHANNON, C. E. The synthesis of two-terminal switching circuits. **Bell System Technical Journal**, Kansas, v. 28, n. 1, p. 59–98, 1949.

SOEKEN, M. **CirKit**: documentation. [*S.l.: s.n.*], 2017. Disponível em: http://cirkit.readthedocs.io/en/latest/. Acesso em: 12 dez 2017.

SOEKEN, M. *et al.* Exact synthesis of majority-inverter graphs and its applications. **IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems**, La Jolla, v. 36, n. 11, p. 1842–1855, 2017.

SOEKEN, M. *et al.* Practical exact synthesis. In: 2018 DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION - DATE, 1., 2018, Dresden. **Proceedings...** Dresden: IEEE. 2018. p. 309–314.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Digital systems: principles and applications**. 12. ed. Dallas: Pearson Education, 2017. 57–58 p.

WALUS, K. *et al.* Circuit design based on majority gates for applications with quantum-dot cellular automata. In: ASILOMAR CONFERENCE ON SIGNALS, SYSTEMS AND COMPUTERS, 38., 2004, Pacific Grove. **Proceedings...** Pacific Grove: IEEE. 2004. p. 1354–1357.

WANG, P.; NIAMAT, M.; VEMURU, S. Minimal majority gate mapping of 4-variable functions for quantum cellular automata. In: IEEE CONFERENCE ON NANOTECHNOLOGY, 11., 2011, Portland. **Proceedings...** Portland: IEEE. 2011. p. 1307–1312.

WANG, P. *et al.* Synthesis of majority/minority logic networks. **IEEE Transactions on Nanotechnology**, Boston, v. 14, n. 3, p. 473–483, 2015.

ZHANG, R. *et al.* A method of majority logic reduction for quantum cellular automata. **IEEE Transactions on Nanotechnology**, Boston, v. 3, n. 4, p. 443–450, 2004.

ZOGRAFOS, O. *et al.* Majority logic synthesis for spin wave technology. In: EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN, 17., 2014, Verona. **Proceedings...** Verona: IEEE. 2014. p. 691–694.

# APÊNDICE A -- ETAPAS PARA UTILIZAÇÃO DO ALGORITMO *EXACT_MIG*

Neste apêndice, apresenta-se um passo a passo de como instalar e utilizar o algoritmo *exact_mig* no sistema operacional Ubuntu.

O algoritmo *exact_mig* está disponibilizado no *framework Cirkit*, um conjunto de ferramentas de síntese para a minimização de circuitos lógicos. Sendo assim, o primeiro passo é acessar o terminal e digitar o comando de *download* do *Cirkit*:

- *git clone --recursive https://github.com/msoeken/cirkit.git*

Em seguida, deve-se criar uma pasta chamada *build* dentro da pasta em que o *Cirkit* foi instalado, e deve-se executar o arquivo de compilação do algoritmo:

- *mkdir build*
- *cd build*
- *cmake ..*
- *make cirkit*

Com a instalação concluída, o *Cirkit* pode ser acessado pelo terminal a partir do comando *./cirkit*, digitado no diretório *cirkit/build/programs*, onde se encontra o executável do programa.

O algoritmo *exact_mig* é acessado no *Cirkit* a partir do comando *exact_mig -o x*, sendo $x$ o parâmetro que define o critério de custo utilizado na formação da função de saída. Se $x = 1$, prioriza-se a redução da quantidade de operadores seguida da quantidade de níveis. Se $x = 2$, prioriza-se a redução da quantidade de níveis seguida da quantidade de operadores.

Para execução do algoritmo *exact_mig*, é necessário que se informe primeiramente uma tabela verdade de entrada, a partir do comando *tt* seguido pela tabela verdade desejada, que deve ser inserida de forma que os mintermos estejam ordenados do maior para o menor. A tabela verdade para o conjunto de mintermos $\{3, 8, 12, 15\}$, por exemplo, é inserida pelo comando *tt 1001000100001000*.

Para exemplificar o conjunto completo de comandos necessários na execução do algoritmo *exact_mig*, considere os mintermos $\{0, 3, 6, 7, 9, 10, 11, 13, 14\}$. A sequência de comandos necessária é:

- *tt 0110111011001001*
- *convert --tt_to_aig*
- *convert --aig_to_mig*
- *exact_mig -o 2*
- *convert --mig_to_expr*
- *print -e*

Nota-se que duas conversões de formato são necessárias para gerar um formato de leitura válido para o algoritmo *exact_mig*, que deve estar em formato *MIG*. Primeiro converte-se o formato *tt* em *AIG* (*And-Inverter Graphs*), a partir do comando *convert --tt_to_aig*. E em seguida, converte-se o formato *AIG* em *MIG* a partir do comando *convert --aig_to_mig*.

O formato *AIG* é semelhante ao *MIG*, enquanto no *MIG* as funções são representadas por operadores majoritários e inversores, no formato *AIG* essa representação é feita com a utilização de operadores *AND* e inversores. Essas duas conversões são necessárias porque não existe nenhum comando de leitura direta para o formato *MIG* no *Cirkit*. Além disso, também não existe nenhum comando que permita a visualização da função minimizada em formato *MIG*, tornando necessária a conversão do *MIG* obtido em uma expressão lógica. Essa conversão é realizada com o comando *convert --mig_to_expr*.

Por fim, o comando *print -e* imprime a expressão armazenada. Para o conjunto de mintermos $\{0, 3, 6, 7, 9, 10, 11, 13, 14\}$, utilizado no exemplo, a expressão gerada pelo algoritmo *exact_mig* é:

- *!<<a!bc><!abd>!<bd<1ac>>>*

Operações majoritárias são representadas por <>, inversores são representados por *!* e a variável $d$ representa o bit mais significativo da tabela verdade de entrada ($tt$). Nota-se que nesse exemplo o algoritmo *exact_mig* foi executado com o parâmetro $x = 2$, que prioriza a redução da quantidade de níveis.