UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”
FACULDADE DE ENGENHARIA DO CÂMPUS DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RICHARD WILLIAM PRETI

OTIMIZAÇÃO DE CIRCUITOS DIGITAIS: ESTUDO DOS PRINCIPAIS MÉTODOS E DESENVOLVIMENTO DE ALGORITMO BASEADO NO TEOREMA DO CONSENSO ITERATIVO

ILHA SOLTEIRA
2023

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

RICHARD WILLIAM PRETI

OTIMIZAÇÃO DE CIRCUITOS DIGITAIS: ESTUDO DOS PRINCIPAIS MÉTODOS E DESENVOLVIMENTO DE ALGORITMO BASEADO NO TEOREMA DO CONSENSO ITERATIVO

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia de Ilha Solteira – Unesp como parte dos requisitos para obtenção do título de engenheiro eletricista.

Orientador:
Prof. Dr. Alexandre César Rodrigues da Silva

ILHA SOLTEIRA
2023

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Preti, Richard William.
P942o Otimização de circuitos digitais: estudo dos principais métodos e desenvolvimento de algoritmo baseado no teorema do consenso iterativo / Richard William Preti. -- Ilha Solteira: [s.n.], 2023
63 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Elétrica) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2023

Orientador: Alexandre César Rodrigues da Silva

Inclui bibliografia

1. Estudo da minimização digital. 2. Método da iteração única. 3. Algoritmo.

Amanda Sertori dos Santos
Bibliotecária - CRB/8-9061
Seção Técnica de Referência, Atendimento ao Usuário e Documentação
Diretoria Técnica de Biblioteca e Documentação

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Ilha Solteira

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

Aos quinze dias do mês de dezembro do ano de dois mil e vinte e três, o discente ***Richard William Pret,*** matriculado sob o nº 161051022, tendo como banca examinadora o seu orientador, *o Prof. Dr. Alexandre Cesar Rodrigues da Silva, o Prof. Dr. Carlos Antonio Alves e o Eng. Eletricista Leonardo Cesar Toneti Teixeira,* apresentou o Trabalho de Graduação intitulado **"OTIMIZAÇÃO DE CIRCUITOS DIGITAIS: ESTUDO DOS PRINCIPAIS MÉTODOS E DESENVOLVIMENTO DE ALGORITMO BASEADO NO TEOREMA DO CONSENSO ITERATIVO".** Obtendo a nota 9,0 (NOVE) e conceito APROVADO.

Prof. Dr. Alexandre Cesar Rodrigues da Silva	Richard William Pret
- Orientador-	- Discente -
Prof. Dr. Carlos Antonio Alves	Eng. Eletric. Leonardo Cesar Toneti Teixeira
- Membro da Banca -	- Membro da Banca -

Faculdade de Engenharia – Departamento de Engenharia Elétrica
Avenida Professor José Carlos Rossi, 1370 - CEP 15.385-000, Ilha Solteira - SP - Brasil
Tel (18) 37431150 - Fax (18) 3742-2735 - www.feis.unesp.br

DEDICATÓRIA

Dedico este trabalho à minha família, por todo o suporte prestado durante toda a minha trajetória, aos meus amigos, que fizeram parte dessa história e me ajudaram a construir este sonho e a todos os professores da graduação, sem os quais chegar até aqui não seria possível.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Regis e Elisângela, pela vida e por todo o apoio durante estes 25 anos. Aos meus irmãos, David e Amanda, pela convivência e companhia. À todos os meus familiares, por todo o suporte e amizade. À minha namorada, Maria Antônia, por todo o incentivo, paciência e apoio. À todos os meus amigos que fiz durante a graduação na UNESP de Ilha Solteira, os amigos que fiz na Siemens Infraestrutura e Indústria e àqueles que, antes de todo esse sonho começar, já faziam parte da minha vida. Um agradecimento especial ao professor Alexandre, que com muita atenção e profissionalismo, me orientou durante este trabalho de graduação e muitas vezes durante a trajetória na Unesp. À todos vocês, meu muito obrigado.

RESUMO

A minimização de funções booleanas já é pauta há muito tempo no meio acadêmico. Desde a década de 50, cientistas das mais diversas formações ligadas às ciências exatas vem buscando uma forma ideal, ou perto da ideal, de minimizar funções lógicas. Minimizar funções lógicas, significa diminuir a quantidade de variáveis e portas lógicas do circuito, no entanto, sem perder funcionalidade. Este é um desafio que persiste há décadas. Tendo em vista que circuitos digitais estão presentes em itens praticamente indispensáveis pela população nos dias atuais, como telefones celulares e televisores. Dessa forma, fica evidente compreender a necessidade de se minimizar, uma vez que, apesar de esses equipamentos estarem cada vez mais potentes, os fabricantes pretendem mantê-los cada vez mais compactos no que diz respeito à exposição de *hardware.* O trabalho de graduação teve como objetivo o estudo dos principais métodos de minimização de funções booleanas, em ordem cronológica, a fim de evidenciar as principais características, vantagens e desvantagens de cada um e trazer uma visão geral do tema. Para o desenvolvimento de um algoritmo que utiliza-se da teoria do consenso iterativo, estudou-se a linguagem de programação *python* e técnicas de programação. O algoritmo desenvolvido tem como característica a obtenção de um mínimo local, entretanto, para os testes realizados, na maioria das vezes, obteve-se o mínimo global. Para a avaliação do desempenho computacional, utilizou-se o tempo de execução e o consumo de memória. Foram utilizadas funções com até 5 variáveis e obtidos tempo de execução inferior a 0,01 segundos e um uso de memória de até 0,1 Mega Byte, parâmetros aceitáveis para este tipo de processamento.

Palavras chave: Estudo da minimização digital. Método da Iteração Única. Algoritmo.

ABSTRACT

The minimization of Boolean functions has been on the academic agenda for a long time. Since the 1950s, scientists from a wide range of backgrounds in the exact sciences have been searching for an ideal, or close to ideal, way of minimizing logic functions. Minimizing logic functions means reducing the number of variables and logic gates in the circuit, but without losing functionality. This is a challenge that has persisted for decades. Given that digital circuits are present in items that are practically indispensable to the population these days, such as cell phones and televisions. In this way, it is clear to understand the need to minimize, since, although these devices are increasingly powerful, manufacturers want to keep them increasingly compact in terms of hardware exposure. The goal of this undergraduate project is to study the main methods for minimizing Boolean functions, in chronological order, in order to highlight the major characteristics, advantages and disadvantages of each method and provide an overview of the subject. In order to develop an algorithm that uses iterative consensus theory, the phython programming language and programming techniques were studied. The algorithm developed has the characteristic of obtaining a local minimum, however, for the tests carried out, the global minimum was obtained most of the time. Execution time and memory consumption were used to evaluate computational performance. Have been used functions with up to 5 variables and obtained execution times of less than 0.01 seconds and memory usage of up to 0.1 Mega Byte, which are acceptable parameters for this type of processing.

Key words: Digital minimization study. Single Iteration Method. Algorithm.

Lista de Figuras

Lista de Tabelas

SUMÁRIO

1. INTRODUÇÃO

Circuitos eletrônicos que têm como base de seu funcionamento as operações de lógica binária são denominados circuitos digitais. A lógica binária utiliza-se apenas dos valores 0 e 1 em suas operações, 0 significando faixa de tensão que representa nível baixo, ou falso, e 1, faixa de tensão que representa nível alto, ou verdadeiro, por exemplo. A lógica binária não trabalha com os valores exatos da grandeza, e sim, com esses valores discretos, diferenciando assim, os circuitos digitais dos circuitos analógicos. Esse tipo de circuito apresenta diversas vantagens em relação aos circuitos analógicos, tais como: a facilidade no armazenamento de informações, maior precisão e exatidão, a possibilidade de utilização de programação nas suas operações e o fato de serem menos afetados por ruídos.

Devido as características citadas e outras vantagens, esse tipo de tecnologia está cada vez mais presente em nosso dia a dia. Alguns equipamentos, que hoje podem parecer indispensáveis por uma grande parcela da população, apresentam circuitos digitais em sua composição, como por exemplo: computadores, aparelhos celulares, televisões, aparelhos de DVD/ Blu-Ray e muitos outros. Além disso, os circuitos digitais se fazem presentes na telecomunicação, visto que sinais analógicos são convertidos em sinais digitais para uma comunicação mais avançada, em empresas automobilísticas, uma vez que os automóveis apresentam cada vez mais tecnologia e conforto, e, principalmente na indústria de forma geral, pois a mesma foi completamente modernizada com o uso de microprocessadores, microcontroladores e controladores lógicos programáveis, fazendo com que as máquinas possam realizar inúmeras tarefas em um tempo de execução muito curto. Pode-se afirmar, que o avanço da tecnologia e da indústria está completamente ligado e conectado ao avanço da eletrônica digital, no entanto, com o passar do tempo os desafios começaram a aparecer. Com uma maior complexidade exigida pelos grandes projetos de automação, controle e outras áreas da engenharia, os circuitos ficaram cada vez maiores, mais robustos e mais caros. Sendo assim, despertou-se, cada vez mais, um grande interesse na otimização desses circuitos, que fez-se assim necessária, até os dias atuais.

Por esse motivo, diversas pesquisas em todo o mundo, desde a década de 50, vêm sendo realizadas a fim de aperfeiçoar métodos de otimização de sistemas digitais por meio de minimizações de funções booleanas, isto é, minimizar a quantidade de variáveis e portas

lógicas, com o intuito de diminuir o custo de implementação dos circuitos digitais. No entanto, a tarefa não é simples. Hoje em dia, a minimização é realizada majoritariamente por meio de algoritmos com bons desempenhos de tempo de execução e memória utilizada, porém, para se chegar ao nível atual, esbarrou-se em diversos desafios ao longo do tempo. Na década de 50, início dos estudos, os custos de porta lógica eram bem altos e esse foi o impulso dado para se começar a pensar em minimizar os circuitos. Posteriormente, nas décadas de 60 e 70, os custos das portas lógicas diminuíram, fazendo com que a minimização não tivesse mais a mesma importância e relevância do passado. Todavia, o assunto ganhou corpo, realmente, após a década de 70, com o surgimento dos PLA's (*Programmable logic arrays*) na implementação das funções lógicas, uma vez que cada termo obtido, ocuparia uma linha do PLA. Dessa forma, diversos métodos de minimização surgiram após a década de 70, e grande parte destes, utilizados ainda.

Este trabalho de Graduação teve como objetivo o estudo dos principais métodos de otimização de circuitos digitais, abordando, de forma geral, suas principais características, vantagens e desvantagens, buscando assim, discorrer sobre a história da minimização de circuitos digitais. Desenvolveu-se um algoritmo, baseado na teoria do consenso e em conceitos já empregados em alguns métodos clássicos para a minimização de funções lógicas descritos na literatura. Na fase final, realizou-se testes, a fim de determinar a qualidade das soluções geradas pelo algoritmo, além de uma análise de desempenho, no quis diz respeito à quantidade de memória utilizada e velocidade de execução.

Um objetivo adicional do trabalho diz respeito ao estudo da programação. Para o desenvolvimento do algoritmo proposto, foi utilizada a linguagem de programação Python e um software de desenvolvimento adequado para a codificação do programa e os testes, conforme a necessidade.

1.1. O Mapa de Karnaugh

O Mapa de Karnaugh (KARNAUGH, 1953) foi elaborado inicialmente por Edward Veitch (VEITCH, 1952) no início da década de 50 e posteriormente aperfeiçoado pelo físico, matemático e engenheiro de telecomunicações americano Maurice Karnaugh. Esse método, consiste em trazer uma representação gráfica da tabela verdade de uma função lógica, e

minimizá-la, a fim de reduzir custos de implementação. Esse artifício ganhou muita popularidade ao longo dos anos e hoje em dia está ilustrado em praticamente todos os livros de sistemas digitais. O método é simples, eficaz e minimiza o erro de simplificações quando comparado a manipulações algébricas, no entanto, pode-se tornar muito complexo para funções com mais de 5 entradas. As simplificações podem ser feitas por mintermos ou maxtermos, cujas definições são:

Mintermo: Produto contendo todas as variáveis na sua forma complementada ou não;

Maxtermo: Soma contendo todas as variáveis na sua forma complementada ou não.

1.1.1. Montagem do Mapa de Karnaugh para 4 variáveis

A minimização de funções booleanas utilizando o mapa de Karnaugh é feita da seguinte forma:

1) Montagem do mapa: O mapa é montado sempre da mesma forma (para mintermos), buscando preencher todos os valores possíveis para o número de variáveis da função. Lembrando que A remete-se ao bit 1 (verdadeiro) e A’ se remete ao bit 0 (falso), portanto, para quatro variáveis, 0000, ou 0 em decimal, será A'B'C'D'. A ilustração do mapa pode ser vista na Figura 1.

Figura 1- Estrutura do Mapa de Karnaugh

	C’D’	C’D	CD	CD'
A’B’				
A’B				
AB				
AB’				

Fonte: Elaborado pelo autor

2) Uma vez obtendo-se a tabela-verdade da função, deve-se preencher os valores das saídas (com 0 ou 1) em sua posição correta no mapa, como podemos visualizar em um exemplo hipotético, na Figura 2.

Figura 2 - Preenchimento do Mapa de Karnaugh

	C'D'	C'D	CD	CD'
A'B'	0	0	1	0
A'B	0	X	0	0
AB	0	1	1	0
AB'	0	0	0	0

Fonte: Elaborado pelo autor

Os "x"são os chamados *don't care states*. Nesses casos, o valor ser zero ou um é irrelevante para a minimização, logo deve ser escolhido o valor mais conveniente como veremos mais adiante.

3) Minimização por agrupamento: A minimização no mapa de Karnaugh é feita por agrupamentos e pode ser realizada em agrupamentos de 1, 2, 4 ou 8 casas adjacentes que tenham valor de saída "1", no caso de minimização por mintermos. Exemplos serão ilustrados nas Figuras 3, 4 e 5.

Figura 3 - Agrupamentos de unitários e pares

	C'D'	C'D	CD	CD'
A'B'	0	0	1	0
A'B	1	X	0	0
AB	0	1	1	0
AB'	0	0	0	0

Fonte: Elaborado pelo autor

Figura 4 - Agrupamento por quartetos e por pares laterais

	C'D'	C'D	CD	CD'
A'B'	1	0	1	0
A'B	0	X	1	0
AB	0	1	1	0
AB'	1	0	1	0

Fonte: Elaborado pelo autor

Figura 5 - Agrupamento por octetos

	C'D'	C'D	CD	CD'
A'B'	1	1	1	1
A'B	1	X	1	1
AB	0	1	1	0
AB'	0	1	1	0

Fonte: Elaborado pelo autor

É importante lembrar que o mesmo valor de saída 1, pode ser agrupado mais de uma vez. Para realizar a minimização correta, é essencial certificar-se de que todos os valores 1 (para minimização por mintermos) foram agrupados.

4) Ao término dos agrupamentos, deve-se analisar os mesmos (cada um simboliza um implicante primo) e montar a função minimizada. Na Figura 5, podemos perceber que 2 implicantes primos foram estabelecidos. São dois octetos. Um deles tem A, B e C variando, portanto esse implicante é igual a D. O outro por sua vez, tem B, C e D variando, logo tem-se A' como sendo o outro implicante primo. Sendo assim, a resposta que representa a função minimizada é: F(ABCD) = D + A'.

1.1.2. Mapa de Karnaugh de 4 variáveis com simplificação por mintermos

Considere a função: $F(A,B,C,D) = \sum (1,2,5,6,9,12,15)$ **(1).**

Logo, tem-se a seguinte Tabela-verdade:

Tabela 1 - Tabela-Verdade referente à função 1

A	B	C	D	Saída	A	B	C	D	Saída
0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	1	0	0	1	1
0	0	1	0	1	1	0	1	0	0
0	0	1	1	0	1	0	1	1	0
0	1	0	0	0	1	1	0	0	1
0	1	0	1	1	1	1	0	1	0
0	1	1	0	1	1	1	1	0	0
0	1	1	1	0	1	1	1	1	1

Fonte: Elaborado pelo autor

O Mapa de Karnaugh que representa essa função lógica é ilustrado na Figura 6:

Figura 6 - Mapa de Karnaugh referente à função 1

	C’D’	C’D	CD	CD'
A’B’	0	1	0	1
A’B	0	1	0	1
AB	1	0	1	0
AB’	0	1	0	0

Fonte: Elaborado pelo autor

Logo, realizando os agrupamentos, obtém-se a minimização na Figura 7:

Figura 7 - Implicantes agrupados referentes à minimização da função 1

	C’D’	C’D	CD	CD'
A’B’	0	1	0	1
A’B	0	1	0	1
AB	1	0	1	0
AB’	0	1	0	0

Fonte: Elaborado pelo autor

Sendo assim, tem-se como resultado: F(ABCD) = ABC'D' + ABCD + B'C'D + A'C'D + A'CD'. É importante destacar que o resultado obtido na minimização por mapa de Karnaugh, assim como as demais minimizações ao longo deste trabalho, dão uma das minimizações possíveis da função, uma vez que, em alguns casos, pode-se obter mais de um resultado correto.

1.1.3. Mapa de Karnaugh de 5 variáveis com simplificação por mintermos

No Mapa de Karnaugh para 5 variáveis, deve-se fixar a variável A em zero e em um, montando assim, 2 mapas. Os agrupamentos devem ser feitos no mesmo mapa, igualmente

aos agrupamentos feitos para 4 variáveis, e também, entre valores de saídas 1 que estiverem na mesma posição no outro mapa. Considere a seguinte função:

$$F(A,B,C,D,E) = \sum (0,1,4,5,8,16,17,20,21,22,24,26,27,30,31) \quad (2)$$

O Mapa de Karnaugh que representa essa função lógica pode ser visto na Figura 8:

Figura 8 - Mapa de Karnaugh referente à função 2

A = 0

	D'E'	D'E	DE	DE'
B'C'	1	1	0	0
B'C	1	1	0	0
BC	0	0	0	0
BC'	1	0	0	0

A = 1

	D'E'	D'E	DE	DE'
B'C'	1	1	0	0
B'C	1	1	0	1
BC	0	0	1	1
BC'	1	0	1	1

Fonte: Elaborado pelo autor

Realizando os agrupamentos, obtém-se a Figura 9:

Figura 9 - Implicantes agrupados referentes à minimização da função 2

A = 0

	D'E'	D'E	DE	DE'
B'C'	1	1	0	0
B'C	1	1	0	0
BC	0	0	0	0
BC'	1	0	0	0

A = 1

	D'E'	D'E	DE	DE'
B'C'	1	1	0	0
B'C	1	1	0	1
BC	0	0	1	1
BC'	1	0	1	1

Fonte: Elaborado pelo autor

Assim tem-se a função minimizada: F(ABCDE) = ABD + ACDE' + B'D'+ C'D'E'

A Tabela-verdade que representa a função 2, pode ser verificada na Tabela 2:

Tabela 2 - Tabela-Verdade referente à função 2

A	B	C	D	E	Saída
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	1
1	0	0	1	0	0
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	1
1	1	1	1	1	1

Fonte: Elaborado pelo autor

1.1.4 Mapa de Karnaugh de 4 variáveis com simplificação por maxtermos

Utilizando o exemplo do item 1.1.2, realizaremos a simplificação utilizando maxtermos. Considere a função: $F(A, B, C, D) = \sum (1,2,5,6,9,12,15)$ (3)

O Mapa de Karnaugh que representa essa função lógica é ilustrado na Figura 10:

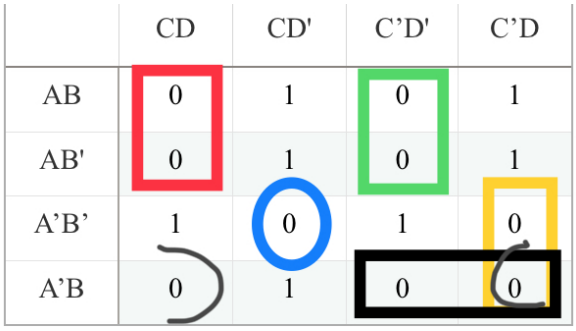
Figura 10 - Mapa de Karnaugh referente à função 3

	CD	CD'	C'D'	C'D
AB	0	1	0	1
AB'	0	1	0	1
A'B'	1	0	1	0
A'B	0	1	0	0

Fonte: Elaborado pelo autor

Na minimização utilizando maxtermos, deve-se agrupar os 0's adjacentes e deve-se atentar a montagem do mapa de Karnaugh, uma vez que agora para a entrada 0000 não tem-se mais a representação A'B'C'D' e sim, A+B+C+D. Sendo assim, logo chega-se a Figura 11:

Figura 11 - Implicantes agrupados referentes à função 3



Fonte: Elaborado pelo autor

Portanto, F(ABCD) = (A+ C+ D) . (A + C'+ D') . (A'+ B'+ C + D') . (A'+ C'+ D) . (A'+ B + C') . (A'+ B + D)

1.2. O Algoritmo de Quine-McCluskey

Com o passar do tempo, os métodos foram sendo aprimorados, uma vez que o mapa de Karnaugh tinha algumas limitações no quesito quantidade de variáveis e ainda era suscetível ao erro humano. Sendo assim, no ano de 1952, o matemático americano Willard Van Orman Quine, propôs um novo método de simplificação de funções booleanas (QUINE, 1952), que 4 anos depois seria aperfeiçoado pelo engenheiro Edward McCluskey (MCCLUSKEY, 1956) e anos mais tarde daria corpo ao que se conhece hoje como algoritmo de Quine-McCluskey (MCCLUSKEY, 1986). Esse novo modelo consiste em um método tabular e iterativo, exato e que realmente trás a solução ótima após a realização em duas etapas: a geração de implicantes primos e implicantes primos essenciais e, em seguida, a cobertura dos mintermos da função e a obtenção da função minimizada. A grande vantagem desse método é a possibilidade da implementação computacional, eliminando qualquer possibilidade de erro humano. Todavia, apesar de muito interessante e revolucionário para a época, o método apresenta alguns pontos negativos, sendo o principal a exaustão em sua realização, o que leva a enormes tempos de execução do programa e o alto consumo de memória.

Para ilustrar os passos do algoritmo, apresenta-se um exemplo de minimização com quatro variáveis empregando o método citado.

Considere a função: $F(A,B,C,D) = \sum (1,2,5,6,9,12,15)$ (4)

1) O primeiro passo consiste em listar todos os mintermos e *don't care states* e agrupá-los pelo número de 1s que eles contenham (CESARKALLAS, "s.d"). Cada agrupamento é chamado de "caixa" e é numerada de acordo com o número de 1's presentes no mintermo. Cada passo do algoritmo, por sua vez, diz respeito à um "grupo". Sendo assim, temos o grupo 0 na Tabela 3.

Tabela 3 - Agrupamento dos mintermos por quantidade de variáveis 1 (Grupo 0)

Caixa 1	0001(1)
	0010(2)
Caixa 2	0101(5)
	0110(6)
	1001(9)
	1100(12) *
Caixa 4	1111(15) *

Fonte: Elaborado pelo autor

2) Os termos com asterisco representam os implicantes primos da função que não podem ser agrupados novamente e os numerais entre parênteses representam, em decimal, os valores cobertos por aquele implicante. Sendo assim, o passo seguinte é comparar as caixas e formar pares de termos que diferem por uma variável, criando um novo termo com uma variável a menos (a variável que difere) e a chamando de X. O grupo 1 está ilustrado na Tabela 4:

Tabela 4 - Agrupamento de pares que diferem de uma variável (Grupo 1)

Caixa 1	0X01(1,5) *
	X001(1,9) *
	0X10(2,6) *

Fonte: Elaborado pelo autor

3) A terceira etapa consiste em repetir o passo 2 até não existir um novo termo a ser formado. Como os implicantes do grupo 1 não podem mais ser agrupados, eles se juntam aos demais implicantes primos. O resultado é um conjunto de implicantes primos da função, que pode ser notado na Tabela 5:

Tabela 5 - Implicantes primos da função 4 (Grupo 2)

Caixa 1	0X01(1,5) *
	X001(1,9) *
	0X10(2,6) *
Caixa 2	1100(12)*
Caixa 4	1111(15)*

Fonte: Elaborado pelo autor

4) O quarto passo do algoritmo é formar uma tabela onde os mintermos originais definem as colunas e os implicantes primos definem as linhas. A relação entre cada mintermo e um dado implicante primo, chamado também de cobertura, é indicada por um X no cruzamento da linha e coluna na tabela. A Tabela 6 ilustra a cobertura completa dos implicantes da função.

Tabela 6 - Tabela de cobertura dos implicantes primos da função 4

1	2	5	6	9	12	15	A,B,C,D
					X		1100
						X	1111
X		X					0X01
X				X			X001
	X		X				0X10

Fonte: Elaborado pelo autor

Como pode-se verificar, todos os implicantes primos são essenciais, uma vez que cada um cobre pelo menos um mintermo que não seja coberto por nenhum outro implicante primo. Sendo assim, tem-se a seguinte função minimizada: F(A,B,C,D) = ABC'D'+ ABCD + B'C'D + A'C'D + A'CD'.

É importante frisar que a resposta obtida nessa minimização é exatamente a mesma obtida na aplicação da minimização pelo mapa de Karnaugh, realizada na seção 1.1.2.

Realizando o exemplo para 5 variáveis, tem-se:

$$F(A,B,C,D,E) = \sum (0,1,4,5,8,16,17,20,21,22,24,26,27,30,31) \quad (5)$$

Agrupando os mintermos em seções de acordo com o número de variáveis 1, obtém-se a Tabela 7:

Tabela 7 - Agrupamento de mintermos de acordo com o número de variáveis 1 (Grupo 0)

Caixa 0	00000(0)
Caixa 1	00001(1)
	01000(8)
	10000(16)
	00100(4)
Caixa 2	00101(5)
	10001(17)
	10100(20)
	11000(24)

Caixa 3	10101(21)
	10110(22)
	11010(26)
Caixa 4	11011(27)
	11110(30)
Caixa 5	11111(31)

Fonte: Elaborado pelo Autor

Dando continuidade na execução do algoritmo, deve-se agrupar os pares de grupos adjacentes. Desse modo, chega-se até a Tabela 8:

Tabela 8 - Agrupamento de pares de implicantes que diferem de uma variável (Grupo 1)

Caixa 0	0000X(0,1)
	00X00(0,4)
	0X000(0,8)
	X0000(0,16)
Caixa 1	00X01(1,5)
	X0001(1,17)
	X1000(8,24)
	1000X(16,17)
	10X00(16,20)
	1X000(16,24)
	0010X(4,5)
	X0100(4,20)
Caixa 2	X0101(5,21)
	10X01(17,21)
	1010X(20,21)
	101X0(20,22)
	110X0(24,26)
Caixa 3	1X110(22,30) *
	1101X(26,27)
	11X10(26,30)
Caixa 4	11X11(27,31)
	1111X(30,31)

Fonte: Elaborado pelo autor

Aplicando mais uma vez o agrupamento de pares adjacentes, tem-se o resultado ilustrado na Tabela 9:

Tabela 9 - Agrupamento de pares de implicantes que diferem de uma variável (Grupo 2)

Caixa 0	X000X (0,1,16,17)	Caixa 1	X0X01 (1,5,17,21)
	00X0X (0,1,4,5)		10X0X (16,17,20,21)
	X0X00 (0,4,16,20)		X010X (4,5,20,21)
	XX000 (0,8,16,24)*	Caixa 3	11X1X (26,27,30,31)*

Fonte: Elaborado pelo autor

Dando prosseguimento na resolução, chega-se a Tabela 10:

Tabela 10 - Agrupamento de pares de implicantes que diferem de uma variável (Grupo 3)

Caixa 0	X0X0X (0,1,4,5,16,17,20,21)*

Fonte: Elaborado pelo autor

Montando a tabela de cobertura, tem-se o resultado da Tabela 11:

$$F(A,B,C,D,E) = \sum (0,1,4,5,8,16,17,20,21,22,24,26,27,30,31)$$

Tabela 11 - Tabela de cobertura dos implicantes primos da função 5

0	1	4	5	8	16	17	20	21	22	24	26	27	30	31	A,B,C,D,E
									X				X		1X110
X				X	X					X					XX000
											X	X	X	X	11X1X
X	X	X	X		X	X	X	X							X0X0X

Fonte: Elaborado pelo autor

Sendo assim, temos como solução: F(ABCDE) = ABD + ACDE' + B'D'+ C'D'E'.

Mais uma vez pode-se observar que a minimização é exatamente a mesma obtida pela resolução utilizando o Mapa de Karnaugh, comprovando a eficácia do método.

1.3. Os métodos heurísticos

Com o passar do tempo, novos desafios foram surgindo e a otimização de circuitos digitais foi tornando-se mais essencial e robusta. Apesar do algoritmo de Quine-McCluskey trazer a enorme vantagem da possibilidade de implementação computacional, eliminando eventuais erros humanos, o método está longe de ser o mais viável para otimizar-se um circuito digital. O algoritmo apresenta longos e exaustivos passos, o que leva a um tempo de processamento alto e a um elevado consumo de memória. Tendo isso em vista, diversos métodos surgiram nos anos subsequentes, aprimorando cada vez mais a velocidade de execução e trazendo soluções aceitáveis com uma utilização de memória razoável.

Estes métodos são conhecidos como métodos heurísticos. Uma técnica heurística é qualquer abordagem para solução de problemas ou autodescoberta que emprega um método

prático que não é garantido como ideal, perfeito ou racional, mas é, no entanto, suficiente para alcançar um objetivo ou aproximação imediata e de curto prazo. Ou seja, como já abordado, este artifício tem como objetivo trazer a melhor solução possível em um curto espaço de tempo. Os principais métodos heurísticos, referindo-se à minimização de funções booleanas são: O MINI (CAIN;HONG;OSTAPKO,1974), o PRESTO (BROWN, 1981), o ESPRESSO-I (BRAYTON, 1982) e o ESPRESSO-II (BRAYTON, 1984).

Segundo Fernandes (FERNANDES, 1991), as simplificações introduzidas por esses métodos, têm como princípio a expansão dos implicantes que representam uma função lógica, e, uma vez tendo realizado esta etapa, na remoção dos mintermos já cobertos pelo implicante expandido. Sendo assim, os métodos heurísticos, ao realizar a expansão dos implicantes e posteriormente a eliminação de alguns que já se apresentam cobertos, conseguiram contornar o principal ponto negativo dos métodos anteriores: a geração e armazenamento exaustivos de todos os implicantes.

1.3.1. O MINI

Desenvolvido pela IBM, em meados da década de 1970, o programa MINI (CAIN;HONG;OSTAPKO, 1974) é utilizado para minimização de funções de muitas variáveis. Em sua entrada tem-se uma especificação de lógica booleana expressa como uma tabela de entrada-saída, evitando assim uma lista com diversos mintermos. O MINI busca uma solução com o menor número de implicantes, sem gerar todos os implicantes primos da função, para isso, subprocessos são iterados até que não haja mais redução na solução.

A abordagem de dois níveis para a minimização da lógica booleana, desenvolvida por Quine e McCluskey parte de duas etapas. A primeira tem como objetivo gerar todos os implicantes primos da função, e a segunda, procura obter a cobertura mínima, como visto na seção 1.2. Segundo o artigo, *"MINI: A Heuristic Approach for Logic Minimization"* (CAIN;HONG;OSTAPKO, 1974) esta abordagem, é uma melhoria considerável na construção e comparação de todas as soluções possíveis. O texto cita também que a geração de implicantes primos evoluiu para um processo relativamente simples com o resultado dos esforços de Roth (ROTH, 1968), Morreale (MORREALE, 1970), Slagel (CHANG; LEE; SLAGEL, 1970) e muitos outros. Contudo, o texto deixa claro um empecilho: o número de

implicantes primos de uma função, que, segundo Miller (MILLER, 1965), em uma função de n-variáveis é proporcional a $3^n/n$. Portanto, para funções de mais de 4 variáveis, o número de implicantes primos pode ser muito grande, além de a etapa de cobertura ser um grande desafio devido à sua alta complexidade computacional.

Partindo desse ponto, o MINI foi desenvolvido com a filosofia de que a minimização tem como primeiros passos a cobertura inicial F e a cobertura DC, respectivamente cobertura de saídas 1 e cobertura de *don't cares*, que trata-se de listas de cubos (teoria que pode ser verificada no Apêndice A), o que seriam análogos à implicantes, onde cada cubo tem P partes. O objetivo é mesclar os cubos de alguma forma em direção ao número mínimo, através de três subprocessos. O primeiro é a expansão, que tem como proposta agrupar dois ou mais cubos maiores que contenham outros cubos menores encontrados nas primeiras coberturas, F e DC. O segundo subprocesso é a redução. A redução objetiva aparar todos os cubos da solução e evitar que vértices, que seriam análogos à mintermos, sejam cobertos por mais de um cubo. Todos os cubos redundantes também são eliminados. Os cubos aparados na redução, seguem para a remodelação. Esse processo localiza todos os pares de cubos que podem ser remodelados em outros pares de cubos desconexos, ou seja, sem nenhuma interseção, cobrindo os mesmos vértices de antes. Esta etapa finaliza a preparação da solução para outra aplicação de uma nova iteração. Os três subprocessos são aplicados de modo iterativo até obter-se a solução mínima do problema.

Brayton (BRAYTON, 1984), cita o MINI diversas vezes em seu livro. Em um trecho o intitula o primeiro método heurístico com sucesso. Fato é que o MINI abriu diversas portas para o aperfeiçoamento dos métodos heurísticos, como pode-se visualizar ao decorrer do trabalho.

1.3.2. O PRESTO

No ano de 1981, D. Brown apresentou um novo método: O PRESTO (BROWN, 1981). O algoritmo que tem como objetivo utilizar um conjunto mínimo de operações rápidas e que ocupem pouca memória, encontra iterativamente uma nova descrição do termo do produto a partir do anterior. Para cada uma de sua iterações, temos a expansão e a redução, onde, expandir significa verificar cada variável de entrada para ver se uma expansão produz

um termo coberto pela função, e, reduzir, significa testar cada termo do produto para uma possível eliminação, verificando se ele é redundante em cada variável de saída (MARTINEZ-CARBALLIDO; POWERS, 1983). O PRESTO, expande a parte de entrada de cada implicante ao seu tamanho máximo, mas reduz ao máximo as partes de saída dos implicantes, removendo os implicantes do maior número possível de espaços de saída.

Apesar de semelhanças, o PRESTO e o MINI apresentam algumas diferenças. Uma dessas diferenças está na questão da ordenação. Enquanto o MINI, faz uso de heurísticas para fins de ordenação em todos os seus subprocessos, ou seja, na expansão, redução e remodelação, o PRESTO não realiza nenhuma alteração nas especificações originais (MARTINEZ CARBALLIDO; POWERS, 1983). A segunda diferença, e talvez a mais evidente delas, é que o PRESTO não apresenta uma etapa de remodelação, e o MINI sim. O PRESTO, por sua vez, possui um procedimento que verifica se um termo produto faz parte da função. O MINI também diverge do PRESTO na forma como é tratada a expansão. O MINI, a fim de verificar se a expansão de um implicante não altera a cobertura da função, gera o complemento da função lógica, já o PRESTO, por sua vez, apesar de não apresentar os custos iniciais para computar o complemento, precisa realizar, posteriormente, uma verificação para identificar se todos os mintermos cobertos pelo implicante expandido são cobertos por algum outro implicante presente na cobertura (BRAYTON, 1984).

Em um de seus textos, explicitando o algoritmo PRONTO (MARTINEZ-CARBALLIDO; POWERS, 1983), os autores deixam claro que o MINI geralmente entrega melhores soluções, no entanto, necessita de um esforço computacional e uma capacidade de memória muito maior que o programa PRESTO, de Brown.

1.3.3. O Programa ESPRESSO

O ESPRESSO-I (BRAYTON, 1982) é um programa processador de dados que utiliza heurística em conjunto com algoritmos específicos para reduzir de forma eficiente a complexidade de circuitos digitais. O programa em questão foi desenvolvido por Robert K. Brayton, em 1982, sendo que o projeto original está disponível como código-fonte C no site da Universidade da Califórnia, Berkeley. O ESPRESSO-I é tido como um algoritmo eficiente em seu desempenho, que apresenta a minimização de lógica em dois níveis, assim como

alguns métodos anteriores. O algoritmo nos permite minimizar e calcular funções lógicas através de suas tabelas verdades e apresenta 3 passos fundamentais de execução: Expansão, cobertura irredutível e redução. O ESPRESSO-I têm como base a notação do cubo, uma notação utilizada para representar funções lógicas que consiste na ideia de que uma expressão booleana com n entradas, pode ocupar 2^n vértices em um cubo de dimensão n. Sendo assim, ao invés de expandir uma função lógica em mintermos, o programa manipula "cubos", representando os termos do produto nas coberturas ON-, DC- e OFF- de forma iterativa.

A entrada do ESPRESSO é uma tabela de funções da funcionalidade desejada e seu resultado é uma tabela minimizada, descrevendo a ON-Cobertura ou a OFF-Cobertura da função, o que varia de acordo com as opções selecionadas. Por padrão, os termos produto serão compartilhados o máximo possível pelas várias funções de saída, mas o programa pode ser instruído a lidar com cada uma das funções de saída separadamente. Isso permite a implementação eficiente em matrizes lógicas de dois níveis, como PLA (*Programmable Logic Array*) ou PAL (*Programmable Array Logic*).

Embora o resultado da minimização não seja garantido como o mínimo global, na prática, isso é muito aproximado, enquanto a solução está sempre livre de redundância.

Comparado aos outros métodos antecessores, este é essencialmente mais eficiente, reduzindo o uso de memória e o tempo de computação em várias ordens de grandeza. O algoritmo ESPRESSO obteve tanto êxito que foi incorporado como uma etapa específica de minimização de função lógica em diversas ferramentas sofisticadas de síntese lógica posteriores (LAWLESS, 2002).

Como já descrito anteriormente o programa ESPRESSO-I apresenta três principais etapas. Na expansão um cubo é expandido até ser primo. Essa operação envolve o complemento de cada entrada, para testar se o novo vértice é membro da off-cobertura ou on-cobertura da expressão booleana. No final deste processo de expansão, temos uma cobertura de primos da função tal que nenhum cubo primo está contido em outro. No entanto, um subconjunto adequado desses cubos primos também pode fornecer uma cobertura, o que nos deixa claro, que o programa não necessariamente obtém a cobertura mínima global.

Na cobertura irredutível procura-se reduzir o número de cubos primos ao mínimo para que não haja cubos primos redundantes cobrindo a função booleana. Os cubos primos são

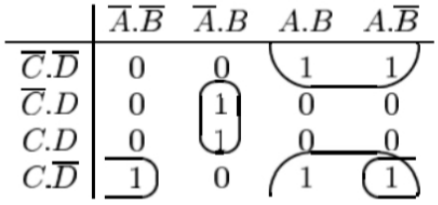
classificados em cubos essenciais que não podem ser omitidos e cubos redundantes que podem ser removidos sem destruir a propriedade de cobertura.

A última etapa é a redução. Nesta etapa, a cobertura obtida até o momento é transformada em uma nova, que substitui cada cubo encontrado por um menor contido nele.

Os três procedimentos descritos, são iterados com diferentes pontos de partida até que não haja mais melhorias na otimização da redução. O comando ESPRESSO é então completado pela preparação de um arquivo que contém a configuração de saída ideal para a função booleana.

Agora, considera-se a operação real do comando ESPRESSO no programa SIS e tem-se como exemplo a minimização do sistema lógico para o qual o mapa de Karnaugh é mostrado na Figura 12 (LAWLESS, 2002).

Figura 12 - Mapa de Karnaugh - Programa Espresso



Fonte: (LAWLESS, 2002)

Este mapa de Karnaugh pode ser expresso como um arquivo de entrada para o programa SIS, colocando-o no formato PLA (*Programmable Logic Array*), conforme mostrado na descrição do arquivo, chamado blexptl. O arquivo de entrada é semelhante à uma tabela verdade, utilizada em métodos anteriores. Observe que os comentários dentro do arquivo foram adicionados à lista posteriormente e não fazem parte dos dados de entrada do programa (LAWLESS, 2002).

```
.i. 4     ;número de variáveis de entrada do programa
.o. 1.    ;número de variáveis de saída do programa
1100 1
1000 1
0101 1
```

0111 1

0010 1

1110 1

1010 1

O sistema SIS é então iniciado e este arquivo de entrada é lido no computador usando a instrução read_pla. O comando ESPRESSO é dado para minimizar o sistema lógico. O resultado da minimização é lido usando o comando write_blif para gravar a saída em um arquivo Berkeley Logic Interchange Format (blif) chamado blexptlo. A sequência de instruções é descrita da seguinte forma:

sis

read_pla blexpt1

espresso

write_blif blexpt1o

quit

A saída lógica minimizada está então disponível no arquivo chamado blexptlo. Novamente, os comentários foram inseridos no arquivo posteriormente(LAWLESS, 2002).

```
.model blexpt1          ;Nome do arquivo de entrada
.inputs v0 v1 v2 v3     ;Nomes SIS para A, B, C, D
.outputs v4.0.          ;Nome SIS para a única saída
.names v0 v1 v3 [1]     ;Primeiro termo produto, ABD
 011   1                 ; ABD = 011 fornece saída 1
.names v0 v3      [2]   ;Segundo termo produto, AD
 10    1                 ; AD = 10 fornece saída 1
 .names v1 v2 v3 [3]     ;Terceiro termo produto, BCD
 010    1                ; BCD = 010 fornece saída 1
 .names [1] [2] [3] [4]  ;A saída é produto de 3 complementos
  000    1               ;Valor 000 = 1
.names [4] v4.0.         ;Completou-se para fornecer a soma dos produtos
   0    1                ; Pelo teorema de De Morgan
.end
```

Interpretações:
A’B D está representado nas linhas 4 e 5
A D’está representado nas linhas 6 e 7
B’C D’está representado nas linhas 8 e 9
o que nos dá a expressão booleana nas linhas 10 a 13:
Sendo assim: Q = A’BD + AD’+ B’CD’

1.3.4. O ESPRESSO-II

O algoritmo ESPRESSO-II (BRAYTON,1984), proposto por Brayton, em 1984, como o próprio nome sugere, é uma evolução do ESPRESSO-I. O programa em questão, assim como o ESPRESSO-I, busca por meio de heurísticas de alta eficiência, encontrar uma solução de minimização de funções lógicas, fazendo uso de poucos recursos computacionais e gastando um tempo de execução cada vez menor.

A estrutura do programa traz em suas entradas, F (ON - Cobertura) e D (*don't care* - Cobertura) de uma função booleana. Outra alternativa possível, é as entradas serem F e R, esta última referindo-se a OFF - Cobertura da função. A saída do programa nos traz uma cobertura minimizada. O ESPRESSO-II, assim como métodos anteriores, utiliza a estrutura de manipulação de cubos ao decorrer de sua teoria.

O processo de minimização inicia-se com a criação de uma função vetorial de 3 variáveis e, após isso, diversos ciclos de simplificação são executados até que nenhuma das 3 variáveis da função tenha sido minimizada por duas vezes em seguida, ou seja, em duas passagens adjacentes do ciclo. As variáveis da função são: O número de cubos que formam a cobertura da função, o número de literais existentes no cubo de entradas da cobertura e o número de literais no cubo de saída.

Antes do início da execução do programa, o procedimento *UNRWAP* é executado, buscando analisar quais cubos de entrada correspondem a mais do que uma saída. Uma vez analisados, estes cubos são separados, ou seja, se um cubo correspondia a n saídas, ele é fragmentado em n cubos, onde cada um diz respeito a uma só saída.

O programa ESPRESSO-II apresenta diversas rotinas ao longo de sua execução. A primeira delas é a "Complemento", na qual procura-se calcular R (OFF-Cobertura da função) caso as entradas sejam F e D. Tendo em vista que um cubo que apresente, na sua interseção com o R da função, um conjunto vazio, é denominado um implicante, o complemento de R é utilizado em rotinas posteriores, para se determinar se um cubo é implicante ou não.

A segunda rotina denomina-se "Expandir". Esta etapa substitui os cubos de F (ON - Cobertura) por implicantes primos. Isso ocorre de forma sequencial, onde cada cubo implicante é substituído por um cubo primo que o contenha ou seja igual ao cubo implicante. Após isso, os cubos cobertos são excluídos da cobertura F.

Dando prosseguimento, temos a rotina “Primos Essenciais”. Esse momento do algoritmo, como o próprio título já sugere, objetiva localizar os implicantes primos essenciais, isto é, definir os implicantes primos que obrigatoriamente precisam estar na cobertura F (ON - Cobertura), uma vez que possuem mintermos existentes somente nestes cubos. O próximo passo da rotina, retira os implicantes primos encontrados da cobertura F e os adiciona na cobertura D (*don't care* Cobertura), de modo que evite que estes implicantes primos essenciais sejam alterados durante o ciclo de minimização. É importante lembrar, que esta rotina em questão, é apenas executada, na primeira vez em que o programa percorre o ciclo de minimização.

O seguinte procedimento, denomina-se “Cobertura Irredundante” e visa particionar a cobertura obtida em passos anteriores em três subcoberturas: a relativamente essencial, a parcialmente redundante e a totalmente redundante. Os cubos que se fazem presentes na subcobertura totalmente redundante, são excluídos da cobertura, sendo assim, podemos concluir que a cobertura de todos os mintermos de F é dada por a união de D com a subcobertura relativamente essencial e uma porção da subcobertura parcialmente redundante. Logo, F (ON - Cobertura), apresenta uma cobertura minimizada de implicantes primos.

O procedimento “Redução”, tem como propósito, reduzir cada cubo pertencente a cobertura F (chamado de c), ao menor cubo (chamado de c-) que contém todos os mintermos de c não pertencentes a [(F - {c}) U D], realizando em seguida F = [F - {c} U {c-}]. Ao término deste procedimento, obtém-se uma cobertura que não é prima, com cubos menores do que os obtidos anteriormente e esta ideia parte de dois propósitos: Cubos menores podem ser expandidos em um número maior de direções, e, quanto menor o cubo, maior a chance de ele ser coberto por expansões de outros cubos.

A etapa que sucede pode ser traduzida como “Último Suspiro”. Esta fase do algoritmo é muito semelhante à etapa “Redução”e também a última tentativa de redução do número de cubos na cobertura. A rotina calcula c^i- para cada cubo c^i pertencente à cobertura F. Isto significa, o menor cubo contendo a parte de c que não é coberta por [(F - $\{c^i\}$) U D]. Diferentemente de etapas anteriores, na rotina “último suspiro”, a redução dos cubos é realizada individualmente, e não em sequência, independente da ordenação dos cubos. Dando sequência, utiliza-se de uma variante da rotina “expansão” para determinar uma nova cobertura, chamada de H. Esta cobertura, busca primos que cubram, no mínimo, um dos

cubos c^i. Após isso, F é substituído por (F U H), fazendo com que o módulo de F seja reduzido após cada iteração.

O último procedimento realizado pelo algoritmo é o “Tornar Escasso”. Nesta parcela do programa os implicantes primos essenciais são retirados da cobertura D e reposicionados na cobertura F, fazendo com que o número de cubos presentes em F torne-se o menor possível. Ao reduzir o número de literais, obtém-se a cobertura F mínima da função.

Em suma, podemos sintetizar o programa da seguinte forma: O procedimento *UNRWAP* (desembrulhar) e a rotina “complemento” realizam as coberturas da função, sendo elas, F, R e D. Após a execução de quatro etapas (“Expansão”, “Cobertura Irredundante”, “Redução”e “Último Suspiro”) analisa-se a qualidade da cobertura. Ao final de cada uma das três primeiras etapas, é monitorado se houve redução na cobertura, caso a resposta seja negativa, o procedimento “Último Suspiro” é acionado a fim de encontrar algum implicante primo que possa-se usar para reduzir a cobertura. Caso seja encontrado algum primo nessa condição, o programa é executado novamente, caso contrário, “Tornar Escasso” é executado e assim temos uma cobertura mínima da função (FERNANDES, 1991).

A porcentagem em relação ao tempo total de execução, de cada uma das etapas pode ser verificada a seguir, na Tabela 12. O tempo restante para completar os 100% fica por conta do procedimento *“Unrwap”*

Tabela 12 - Porcentagem de cada rotina em relação ao tempo total de execução

Complemento	14%
Expansão	29%
Cobertura irredundante	12%
Primos Essenciais	13%
Redução	8%
Último Vão	12%
Tornar Escasso	10%

Fonte: (FERNANDES, 1991)

O programa ESPRESSO-II inspirou diversos outros trabalhos posteriores, uma vez que obteve enorme sucesso e cumpriu com seus objetivos, trazendo um algoritmo robusto em conjunto com uma minimização eficaz em um tempo de execução interessante. Pode-se

compreender também, que o sucesso do programa ESPRESSO-II, está relacionado com a eficácia de cada uma de suas rotinas, bem organizadas e bem estruturadas, o que faz com que o algoritmo torne-se mais exitoso.

1.4. O Método do Consenso Iterativo

O método do consenso iterativo (MOTT, 1960), objetiva executar as operações de consenso e eliminação, sucessivamente, até o momento em que não se possa mais aplicá-las. A definição do consenso entre dois termos, sugere que havendo dois termos produtos com um único literal, que apareça em ambos os termos, sendo em um de forma negada, e em outro, de forma não negada, o resultado da operação se dará por um novo termo, que contenha os literais restantes dos dois termos iniciais combinados. A Tabela 13, explicita alguns exemplos de consensos entre dois termos produtos (FRANCISCANI, 2016).

Tabela 13 - Exemplos de aplicação do consenso

Termos produto	Consenso
A’B e AB	B
A’B’D e A’C D’	A’B’C
AB e AB’DE	ADE
B’C’E e AC	AB’E
ACD’ e A’B C'	Não há consenso entre os termos

Fonte: Elaborado pelo autor

No último caso não houve consenso entre os termos, isso se deu pois mais de um literal apresentava formas negada e não negada em ambos os termos.

O método do consenso iterativo, visa aplicar o consenso e a eliminação até que não seja mais possível fazer uso destas técnicas. A eliminação consiste na ideia de eliminar, passagem a passagem, qualquer termo produto, que já esteja sendo coberto por outro. A seguir, tem-se um exemplo:

Considerando os termos produto A’B’CD + AB’CD’ + ABCD’, tem-se:

1) O consenso entre AB’CD’ e ABCD’ é ACD’;
2) Como ACD’ cobre AB’CD’e ABCD’, temos: A’B CD + ACD’

3) A aplicação do consenso entre os dois termos restantes não é mais possível, uma vez que há dois literais (A e D) presentes em ambos os termos, de forma negada e não negada. Logo a minimização da função é : A'B'CD + ACD'

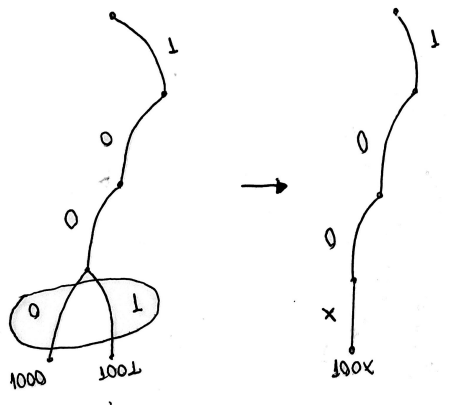
Neste outro exemplo, com os termos produto ABC'D'+ ABC'D + A'BC'D, temos:

1) O consenso entre ABC'D e A'BC'D é BC'D;
2) Como BC'D cobre ABC'D e A'BC'D, temos: ABC'D' + BC'D;
3) O consenso entre ABC'D' e BC'D é ABC', portanto o resultado da minimização é ABC' + BC'D, uma vez que as operações de consenso e eliminação não podem ser mais aplicadas e ABC' não cobre BC'D, o que impossibilita a eliminação deste último termo citado.

O método do consenso iterativo utiliza-se da árvore binária para ilustrar as operações. Esta é uma forma equivalente a tabela verdade, utilizada para representar os mintermos de uma função booleana. Esta representação é extremamente conveniente para a implementação computacional, uma vez que um mesmo ramo pode ser utilizado por mais de um mintermo, o que significa economia de memória. Algumas situações remetem à alterações na árvore durante a minimização. A estrutura da árvore binária e mais detalhes sobre essas situações serão ilustradas a seguir:

A primeira situação possível é a fusão. Nesse tópico, dois ramos adjacentes, são substituídos por um novo, que cubra os dois anteriores. Na Figura 13, temos a ilustração de um exemplo, onde dois ramos adjacentes (1001 e 1000), que representam dois mintermos da função, sofrem uma fusão e são substituídos pelo implicante 100X.

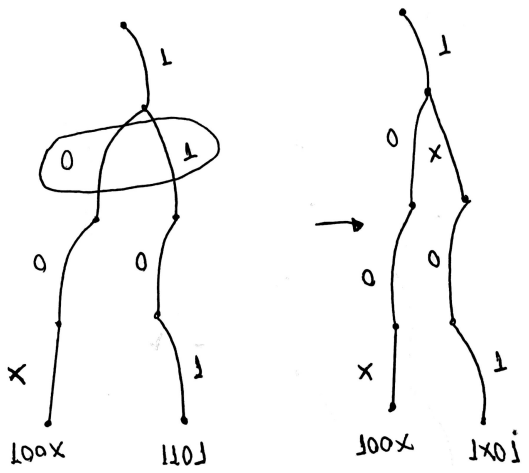
Figura 13 - Aplicação da fusão entre dois ramos



Fonte: Elaborado pelo autor

A segunda situação possível de ocorrer no diagrama de árvore binária é o deslocamento. O deslocamento acontece quando dois ramos adjacentes geram um terceiro, assim como na fusão. No entanto, a diferença entre estas duas situações é que no deslocamento o ramo gerado cobre apenas um dos anteriores, portanto, obtemos a eliminação de um dos ramos anteriores e não dos dois. A Figura 14, ilustra esse procedimento. Na Figura 14, tem-se 100X e 1101, um em cada ramo. Sendo assim, aplicando-se o consenso, surge um terceiro ramo, 1X01. Como o implicante 1X01 não cobre 100X, este último citado permanece na estrutura da árvore.

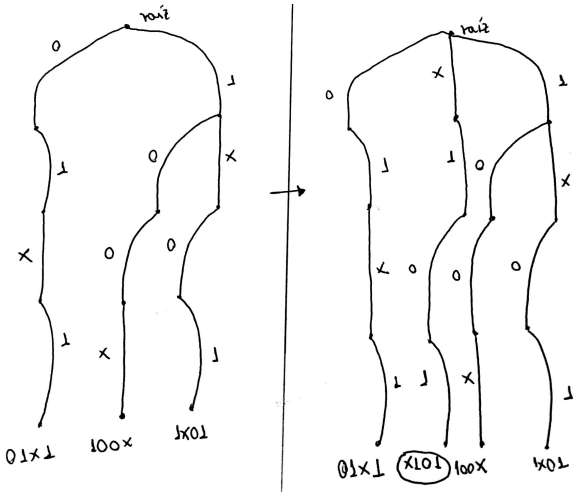
Figura 14 - Aplicação do deslocamento entre dois ramos



Fonte: Elaborado pelo autor

A última situação possível de ocorrer em um diagrama binário após a aplicação do consenso é a expansão. O processo de expansão implica na criação de um novo implicante a partir de dois outros. Na expansão, o termo criado não cobre nenhum dos anteriores, portanto, os 3 permanecem na estrutura da árvore. A Figura 15 ilustra uma expansão causada pelos termos 01X1 e 1X01, resultando no termo X101.

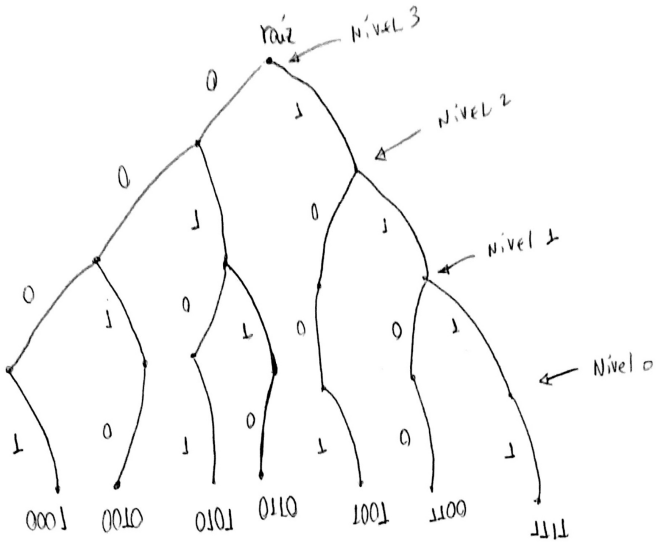
Figura 15 - Aplicação da expansão entre dois ramos



Fonte: Elaborado pelo autor

Nesta etapa do trabalho, repetiremos o exemplo para 4 variáveis executado na seção 1.1.2 para ilustrar método do consenso iterativo. A Figura 16 ilustra a estrutura de árvore binária equivalente à tabela verdade do exemplo.

Figura 16 - Estrutura de diagrama binário equivalente ao exemplo

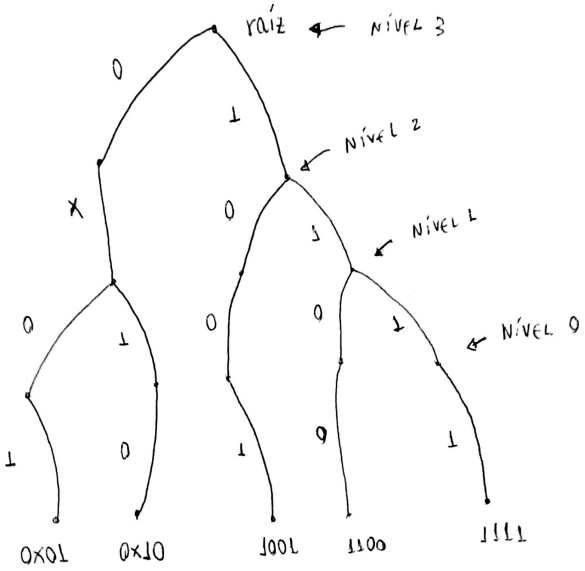


Fonte : Elaborado pelo autor

Pode-se verificar que nos níveis 0 e 1, não há possibilidade da aplicação do consenso entre os mintermos da função. No nível 2, por sua vez, verifica-se a possibilidade da aplicação do consenso entre dois termos, 0001 e 1001 que tornam-se X001 após uma fusão e 0010 e 0110, que tornam-se 0X10, após outra fusão, conforme pode-se visualizar na Figura 17. Por último, observamos, no nível 3, um deslocamento, provocado pelos termos 0X01 e 1001, e gerando o implicante X001, conforme ilustrado na Figura 18. Dessa forma, através do método

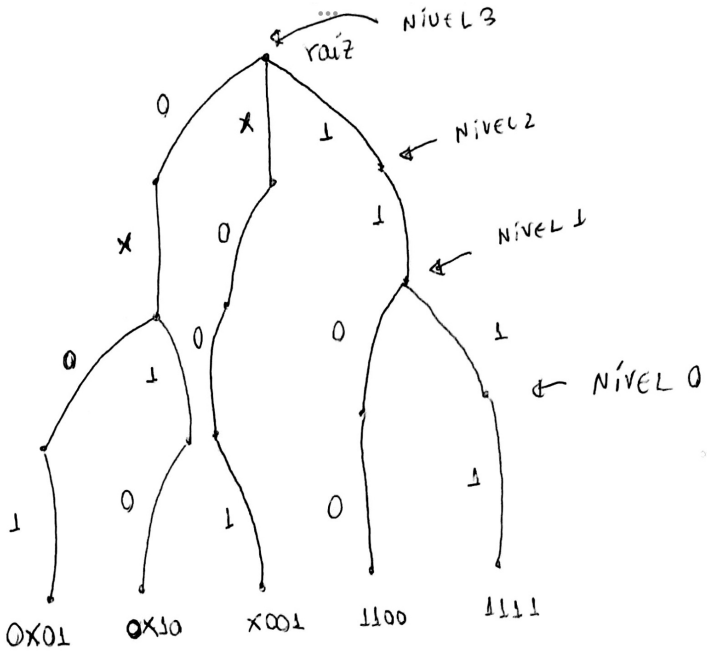
do consenso iterativo, obtivemos a mesma minimização encontrada anteriormente com o emprego do mapa de Karnaugh e com o método de Quine-McCluskey.

Figura 17 - Aplicação do consenso no nível 2



Fonte : Elaborado pelo autor

Figura 18 - Aplicação do consenso no nível 3



Fonte : Elaborado pelo autor

2. MATERIAIS E MÉTODOS

Para a execução deste trabalho, o material necessário está por conta de um software para codificação de linguagem Python. O software escolhido foi a plataforma de codificação online programiz. A linguagem Python foi escolhida por ser moderna, de alto nível e por apresentar diversas bibliotecas. Outro motivo para a escolha do Python, foi o interesse do autor pela linguagem, fazendo com que o estudo dessa ferramenta ao longo desse projeto fizesse parte do plano do trabalho.

Em relação aos métodos, primeiramente foi feito um estudo das principais técnicas de otimização de sistemas digitais já propostos, com a finalidade de absorver suas mais notórias características e expor suas ideias de execução, além de aprofundar o estudo sobre a história e a evolução do assunto abordado no projeto. Esse estudo inicial foi feito por meio da leitura de artigos e outros trabalhos já realizados, além de pesquisas em bases de dados científicas.

Posteriormente, como já citado, utilizou-se a ferramenta Python para desenvolver um programa que tem como base um dos métodos estudados, o método do consenso iterativo. Na parte final do projeto, o algoritmo desenvolvido foi testado, trazendo à tona suas principais características, êxitos e contratempos, além de resultados de execução, no que diz respeito à quantidade de memória e velocidade de execução.

2.1 A Linguagem Python

A linguagem de desenvolvimento Python, lançada em 1991 pelo matemático holandês Guido Van Rossum, é uma linguagem de alto nível, interpretada, ou seja, cada linha é executada uma por vez. Este último ponto é visto por muitos como uma desvantagem, uma vez que leva mais tempo do que compactar tudo em um compilador. Entretanto, a ferramenta apresenta diversos benefícios tais como a facilidade em aprender, utilizar, entender e interagir com a plataforma. Adicionalmente, tem-se a questão de poder ser utilizada em diversos sistemas operacionais ou até mesmo na web e principalmente a vantagem de ser um software livre, o que significa que não é necessário pagar para utilizar de seus recursos.

O Python apresenta uma lógica e linha de raciocínio bem simples e é bastante direto, o que rapidamente fez com que a linguagem se popularizasse, chegando a estar entre as cinco

linguagens mais utilizadas, de acordo com uma pesquisa realizada pela empresa RedMonk. O trecho retirado do site “didática tech”, traz uma boa visão de algumas das principais virtudes da linguagem:

> *Como Python é uma das linguagens mais ativas em termos de comunidade, a cada dia novas bibliotecas são construídas e aprimoradas. Existem funções e módulos prontos para se executar de tudo, desde manipulações em imagens até algoritmos de inteligência artificial. Isso é muito conveniente porque um programador iniciante acaba conseguindo obter recursos e resultados avançados apenas importando e utilizando módulos prontos, sem precisar criar tudo do zero. Em outras palavras, a programação python é diferenciada pela riqueza de bibliotecas e frameworks prontos para utilização, bem como pelo suporte da comunidade. O fato de existirem bibliotecas robustas também permite que um programador se especialize em uma tarefa específica, por exemplo: “manipulação de tabelas e datasets” para ciências de dados. Nesse caso, bastaria estudar e dominar a biblioteca Pandas.*
>
> (DIDÁTICA TECH, 2022).

Uma das principais diferenças do Python é que apesar de várias partes possuírem padrões e especificações formais, a linguagem, como um todo, não é formalmente especificada. É uma linguagem livre, onde os desenvolvedores podem modificar seu código, o que permite que seja utilizada para todos os tipos de complexidade e mesmo assim não deixar de lado a clareza e permitir a fácil leitura do código, exigindo poucas linhas de programação se comparado ao mesmo programa em outras linguagens, que é seu principal objetivo (SILVA, 2018). Hoje em dia, a linguagem tem aplicações nas mais diversas áreas, como podemos visualizar neste trecho, retirado de uma edição de 2018 da revista portuguesa “Programar”:

> *algumas empresas e softwares livres que utilizam Python em suas aplicações como: Zope (em seu gerenciador de conteúdo); Air Canada (em suas reservas de passagens e assentos aéreos); BitTorrent (totalmente elaborada nesta linguagem); Globo (em seus streamings); Google (em seus crawlers utilizados para buscas na internet); Youtube (totalmente feito nesta linguagem); NASA; Industrial Light & Magic (distribuidora do Star Wars, utiliza a linguagem em suas aplicações gráficas); Autodesk (na aplicação Maya e Softimage para aplicações de efeitos visuais e animações em 3D); Blender e Gimp (programas de manipulação de imagens e animações dinâmicas em 3D)* (SILVA, 2018).

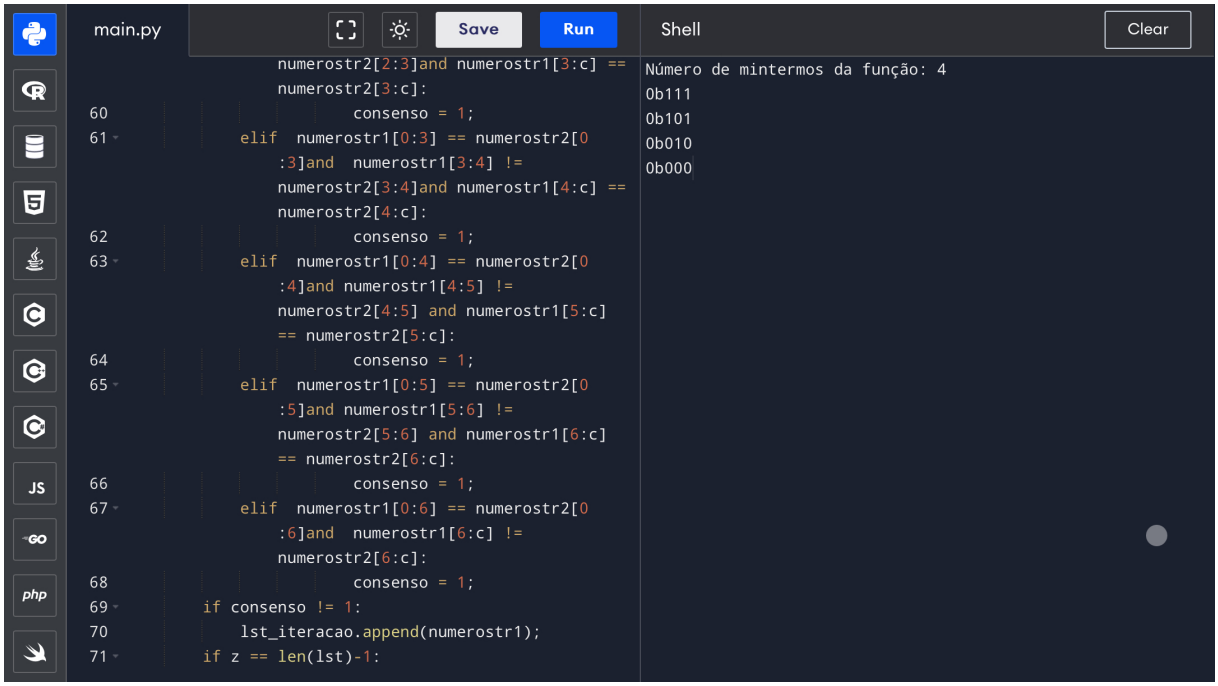
Levando todos estes fatos em consideração, a linguagem foi a escolhida pelo autor para o desenvolvimento do projeto proposto neste trabalho de graduação.

3. MÉTODO DA ITERAÇÃO ÚNICA

O Método da Iteração Única é um método computacional, desenvolvido neste trabalho de graduação, baseado, principalmente, na teoria do consenso iterativo. O método foi desenvolvido na linguagem Python, em uma plataforma online de programação chamada “programiz”. O nome, deve-se ao fato de o programa ser executado por completo apenas uma vez, e assim, gerar uma minimização da função booleana que se deseja.

Inicialmente, o número de mintermos da função deve ser escrito para que o programa inicie. Após isso, o usuário apresenta como entrada os mintermos da função, listados de maneira aleatória, na notação binária utilizada na linguagem phyton ‘0b’, sendo que o prefixo ‘0b’ significa que o valor é binário. Dessa forma, o valor 7, em decimal, por exemplo, seria representado por ‘0b111’. A Figura 19 ilustra a entrada do programa.

Figura 19 - Entrada do programa



Fonte: Elaborado pelo autor

Na sequência, os mintermos são armazenados em uma lista e transformados em números com 7 dígitos binários, apenas por padronização. Com a lista completa, o programa realiza a aplicação da teoria do consenso, da seguinte forma: O primeiro mintermo é atribuído a uma variável e comparado com os demais, um por vez, em busca de verificar se há consenso entre eles, partindo do princípio que, apenas uma das variáveis pode divergir entre os

mintermos para que haja o consenso. Se a condição do consenso for respeitada, ocorre a aplicação do consenso entre os termos e o resultado é armazenado em uma nova lista, que contemplará então, parte dos implicantes da função. Sendo assim, este passo é realizado, até que todos os mintermos da lista sejam varridos. Um detalhe importante, é que, neste primeiro momento, o primeiro mintermo é comparado com todos os demais, no entanto, o segundo é comparado apenas com o terceiro e seus sucessores, e assim por diante.

Na próxima etapa do programa, a lista inicial é novamente utilizada. Desta vez, para determinar quais mintermos não apresentam consenso com nenhum outro, sendo assim, um implicante essencial da função. Dando continuidade, novamente, um mintermo por vez é atrelado à uma variável, todavia, desta vez, todos os mintermos são comparados com cada um dos demais integrantes da lista, sem exceção. Após a comparação, caso a condição de consenso não tenha sido respeitada na verificação com nenhum outro mintermo da lista, este mintermo será movido para a mesma lista de implicantes que já contém os termos provenientes da aplicação do consenso. Deste modo, apenas com uma iteração, consegue-se visualizar uma nova lista, que contém implicantes da função. Um ponto a ser esclarecido, é que nem sempre, os implicantes trazidos após a iteração serão implicantes essenciais, o que quer dizer que o método não garante uma solução mínima global, e, caso o usuário deseje obtê-la, deve realizar uma cobertura após a execução do método.

A primeira etapa do método da iteração única pode-se dizer análogo à fusão do método do consenso iterativo, buscando trazer implicantes que cubram ambos os mintermos selecionados. Após o armazenamento destes mintermos em uma nova lista, a segunda parte do método é acionada, verificando quais termos podem não ter sido cobertos pelos demais implicantes, portanto, esta fase do programa, pode-se dizer análoga ao deslocamento e a expansão, do método do consenso iterativo.

O software apresenta algumas limitações como o fato de não aceitar valores de *don't cares states* na minimização e de os mesmo não ser aplicável para mais de 7 variáveis.

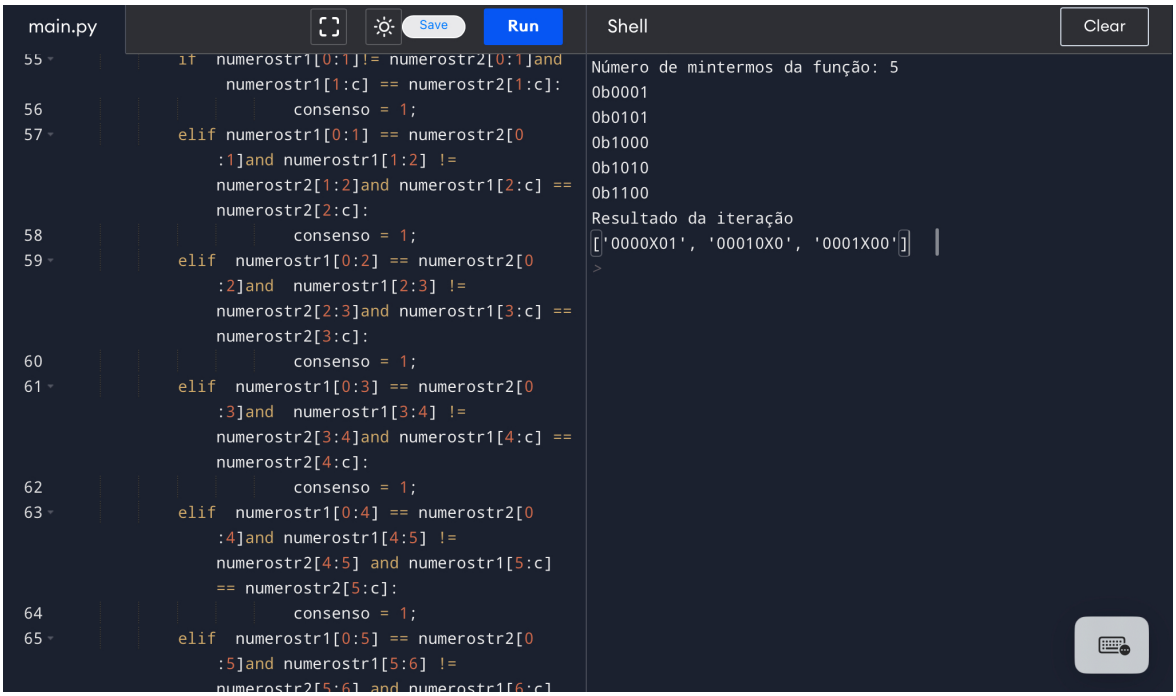
Em suma, o propósito do método da iteração única é ser simples, e, utilizando a teoria do consenso, gerar em pouco tempo, alguma minimização para funções booleanas. No Apêndice B, podemos verificar um fluxograma para o método. No Apêndice C, há o código completo do algoritmo e comentários sobre cada linha. Testes serão descritos tos na seção 3.1.

3.1. Testes e Resultados

Após a elaboração e codificação do programa, testes foram realizados, utilizando a plataforma online programiz. Pode-se constatar que o algoritmo cumpriu sua função no que diz respeito à minimização de funções booleanas, no entanto, isso ocorre em diferentes níveis, de acordo com cada função.

Tomando como exemplo uma função com 5 mintermos de 4 variáveis: $F(A,B,C,D) = \sum(1,5,8,10,12)$ (6). Como podemos visualizar na Figura 20, para este caso, a solução trazida pelo programa foi a minimização completa, contendo todos os implicantes primos essenciais: 0X01 (A'C'D), 10X0 (AB'D') e 1X00 (AC'D'). Isso ocorreu, pois neste caso, uma iteração é a suficiente para a minimização ocorrer de forma completa.

Figura 20 - Resultado da minimização da função 6

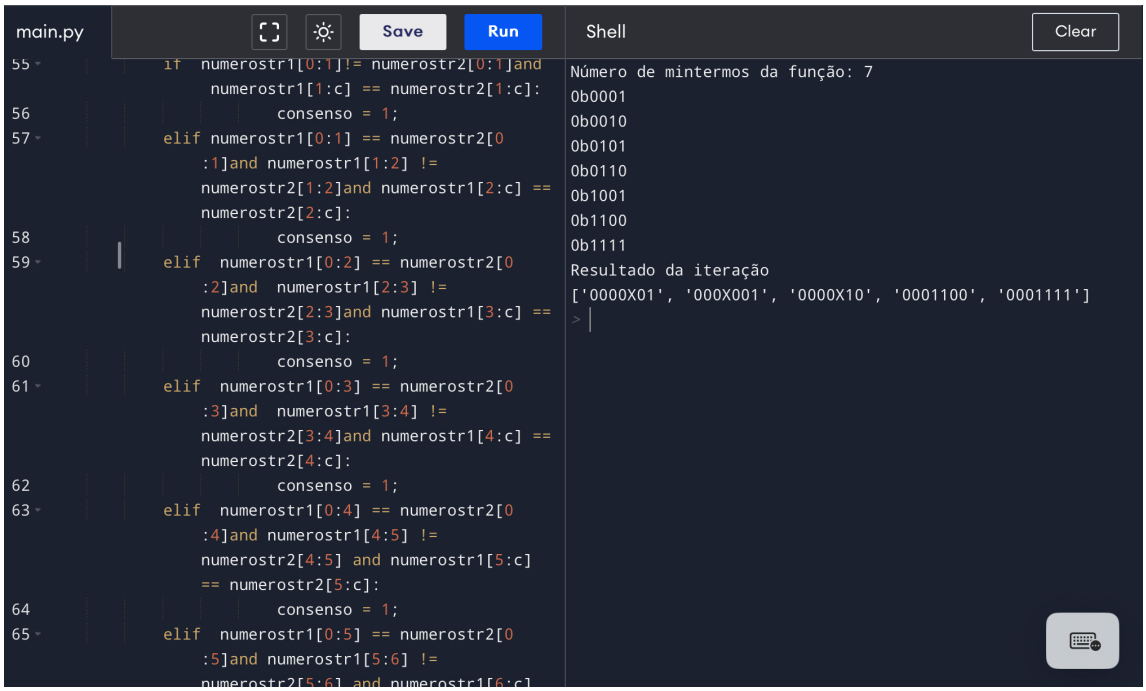


Fonte: Elaborado pelo autor

Em seguida, toma-se como exemplo a função de 4 variáveis minimizada neste trabalho nas seções 1.2 e 1.1.2 . Considere $F(A,B,C,D) = \sum(1,2,5,6,9,12,15)$ (7). Na Figura 21, apresenta-se mais uma vez que a minimização ocorreu de forma completa, trazendo todos os implicantes primos essenciais. Novamente, isso ocorreu pois apenas uma iteração foi

suficiente para minimizar a função booleana. Os implicantes são: 0X01 (A'C'D), X001 (B'C'D), 0X10 (A'CD'), 1100 (ABC'D') e 1111 (ABCD).

Figura 21 - Resultado da minimização da função 7



Fonte: Elaborado pelo autor

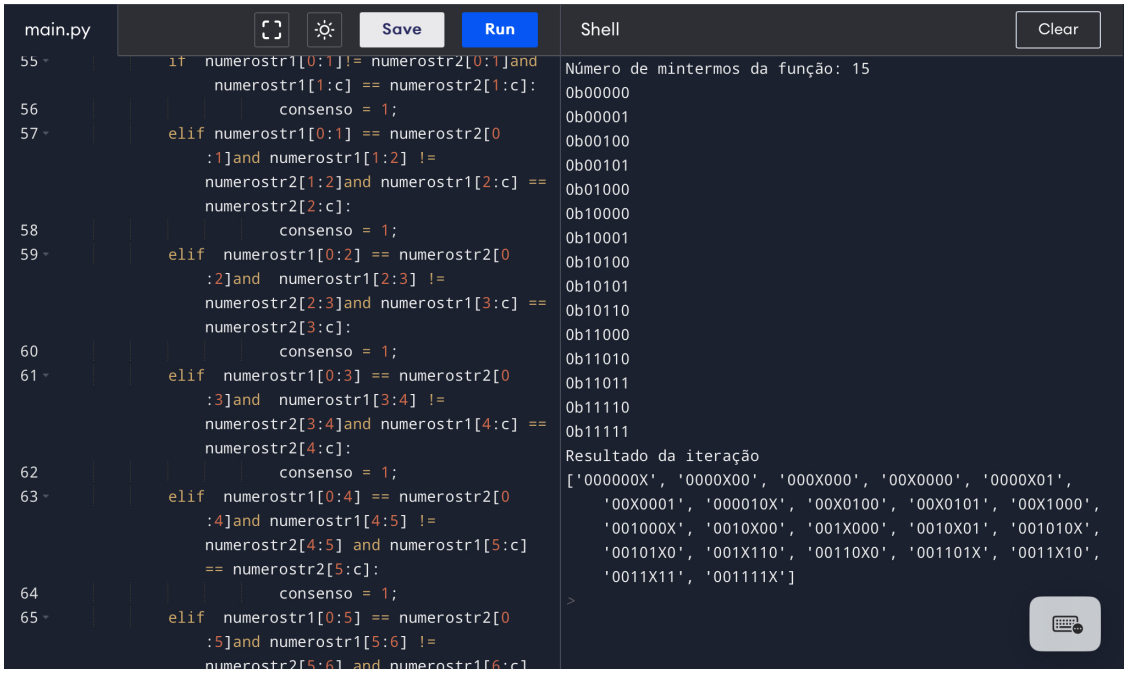
Algumas funções, no entanto, necessitam de mais de uma iteração para serem minimizadas de uma forma completa, ou até mesmo de uma cobertura. A função de 5 variáveis $F(A,B,C,D,E) = \sum(0,1,4,5,8,16,17,20,21,22,24,26,27,30,31)$ (8), utilizada como exemplo nas seções 1.1.3 e 1.2 é uma evidencia deste ponto. Ao usar os mintermos da função como entrada no algoritmo construído neste trabalho, fica claro que os implicantes gerados não são necessariamente os essenciais e que uma cobertura deve ser aplicada para que a solução mínima global seja obtida. A Figura 22 apresenta o resultado da minimização.

Em um novo exemplo, ainda com 4 variáveis, pode-se reafirmar que nem sempre uma iteração é suficiente para chegarmos até a minimização global.

Considere $F(A,B,C,D) = \sum(0,2,5,6,10,11)$ (9). Pode-se observar que o implicante X010 não é essencial, visto que cobre os valores 2 e 10, em decimal, e os mesmos já são cobertos, por exemplo, por 0X10 e 101X, respectivamente. Sendo assim, apenas uma cobertura poderia entregar a minimização com somente implicantes primos essenciais.

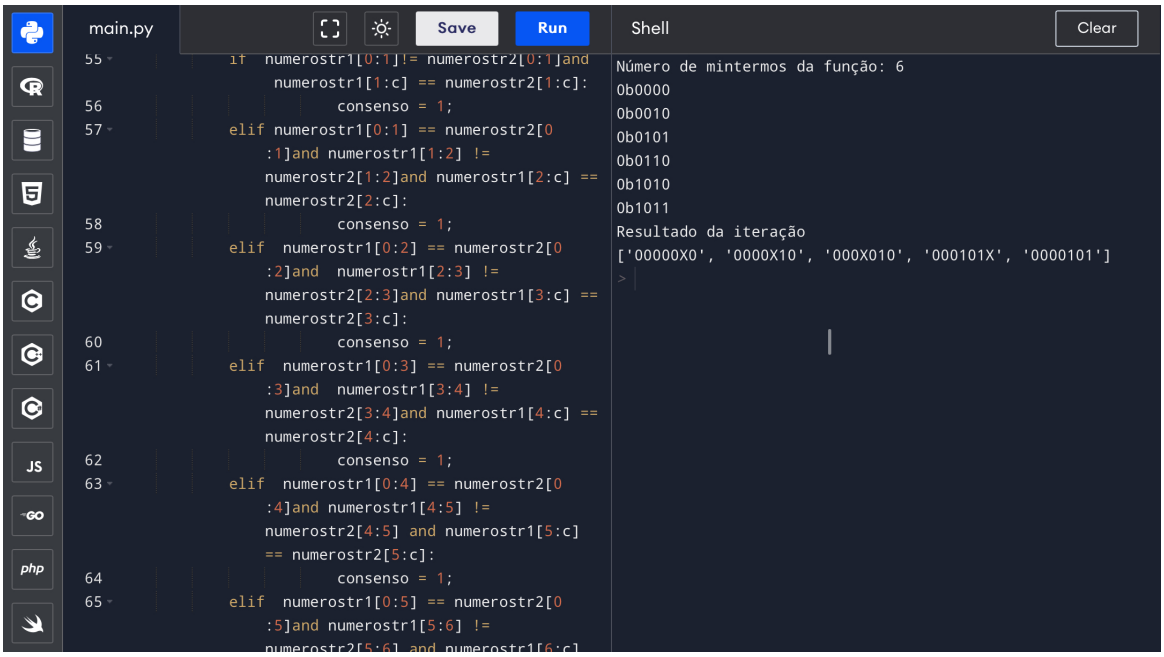
Todavia, o método da iteração única se mostrou eficiente, trazendo rapidamente os implicantes da função. A Figura 23 evidencia a minimização.

Figura 22 - Resultado da minimização da função 8



Fonte: Elaborado pelo autor

Figura 23 - Resultado da minimização da função 9

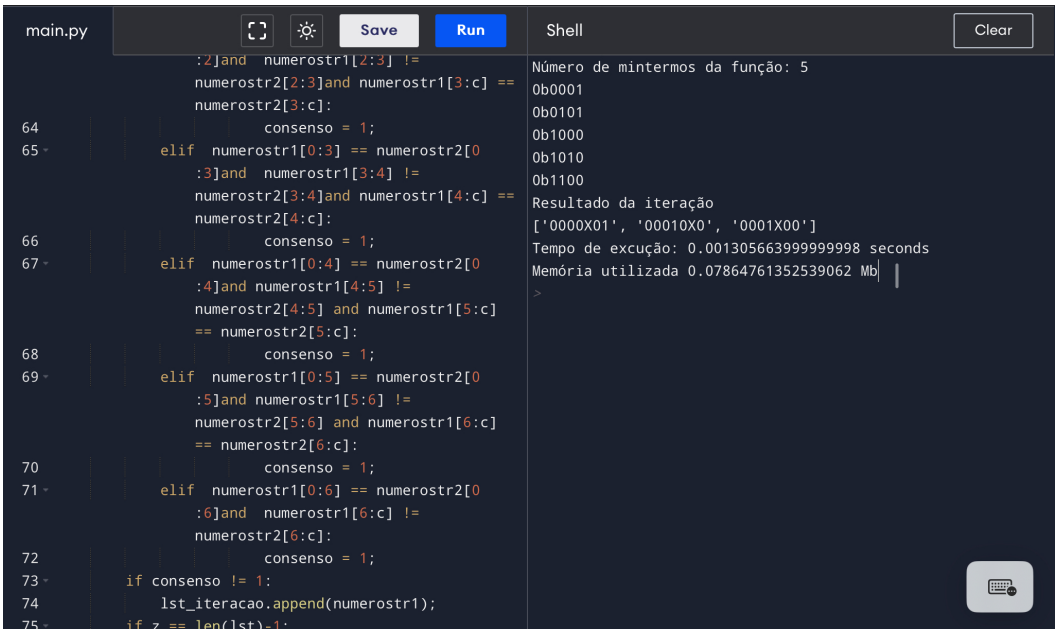


Fonte: Elaborado pelo autor

3.1.1. Desempenho de Tempo e Memória

Na última etapa do trabalho, o algoritmo foi testado quanto ao tempo de execução e quanto à memória utilizada. Para tal análise, duas bibliotecas foram importadas: a *time* e a *resource*. Algumas linhas foram adicionadas ao código. A versão final em texto estruturado pode ser verificada no Apêndice C. A partir disso, foi possível realizar os testes com relação a tempo e memória utilizada. Os resultados estão expostos nas Figuras de 24 a 27 e tomaram como base os mesmos exemplos utilizados para a demonstração do algoritmo na Seção 3.1.

Figura 24 - Resultado do teste de desempenho da função 6

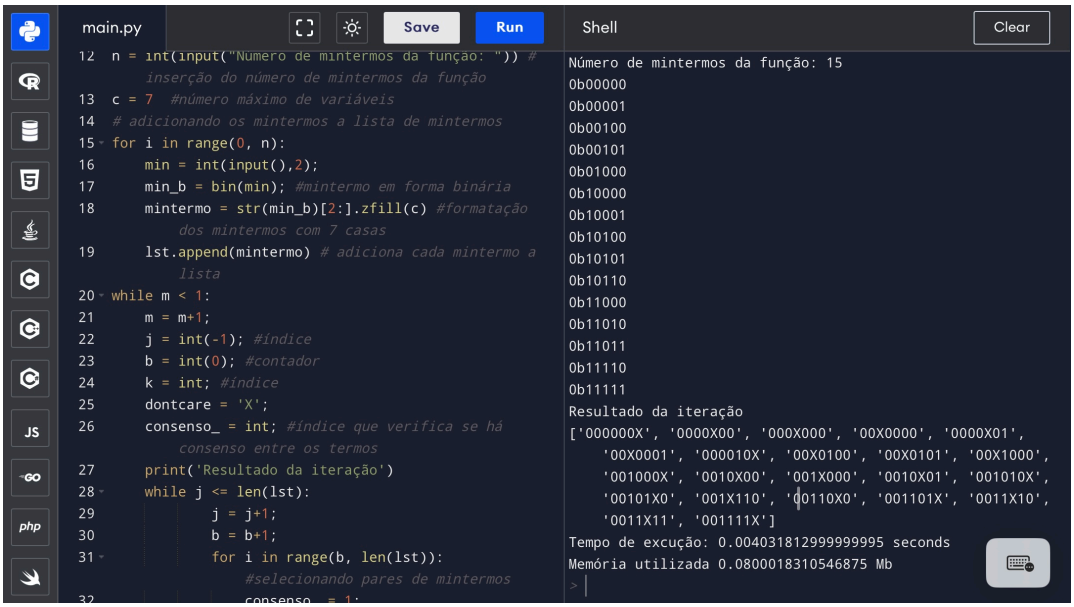


Fonte: Elaborado pelo autor

Figura 25 - Resultado do teste de desempenho da função 7

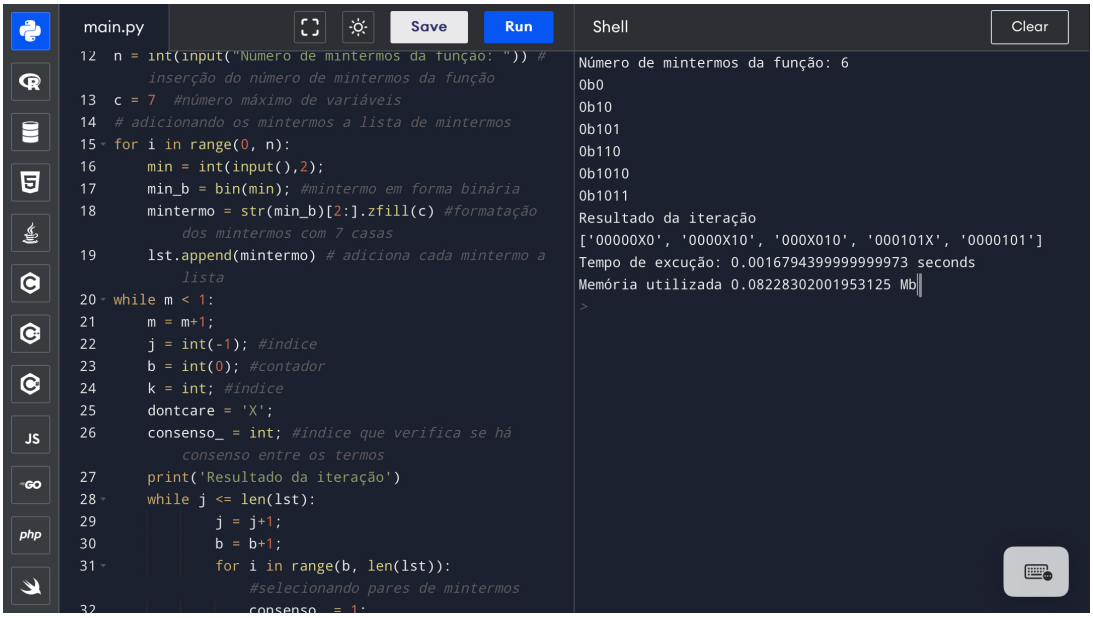
Fonte: Elaborado pelo autor

Figura 26 - Resultado do teste de desempenho da função 8



Fonte: Elaborado pelo autor

Figura 27 - Resultado do teste de desempenho da função 9



Fonte: Elaborado pelo autor

Sendo assim, os testes previstos foram realizados e os devidos resultados apresentados. Como um novo método de minimização foi desenvolvido na plataforma python, conclui-se que o trabalho cumpriu seus objetivos. Quanto aos testes de performance, considera-se o resultado adequado e satisfatório, tendo em vista que o método traz implicantes de uma função de 4 ou 5 variáveis em um tempo inferior à 0.01 segundo e com uma memória inferior à 0.1 mega byte.

4. CONCLUSÕES

Ao longo deste trabalho de graduação foi realizado um estudo dos principais métodos de minimização de sistemas digitais, procurando trazer à tona suas principais características, vantagens e desvantagens, em uma ordem cronológica.

O texto procurou, dessa forma, ressaltar a importância da minimização de funções booleanas e como os circuitos digitais estão presentes e impactam no nosso cotidiano. A cronologia fez-se importante para compreender a evolução constante dos métodos, que vão desde os mais mecânicos, como o mapa de Karnaugh, que tornou-se uma referência de alta relevância em diversos livros didáticos e materiais sobre o tema, até algoritmos computacionais complexos, como é o caso dos programas ESPRESSO I e II.

Após os estudos dos principais métodos e tomado conhecimento sobre a história da minimização digital, elaborou-se um algoritmo que objetiva a minimização de funções booleanas. O programa foi escrito na linguagem de programação Phyton, fazendo com que o estudo da linguagem fizesse parte dos objetivos do trabalho. Posteriormente à codificação do algoritmo, testes foram realizados, a fim de evidenciar as características do programa e verificar pontos importantes, como velocidade de execução e quantidade de memória utilizada. O programa, que tomou por inspiração o método do consenso iterativo, foi desenvolvido com o auxílio da plataforma online “programiz”. Após isso, testes foram realizados e expostos na seção 4 do trabalho. Os testes evidenciaram que o algoritmo (denominado “Método da Iteração Única”) gera em diversos casos a minimização contendo os implicantes primos essenciais, apenas. No entanto, para outros casos, mais de uma iteração é necessária, ou até mesmo uma cobertura para que se chegue ao mínimo global, o que significa que algoritmo minimiza parcialmente a função, ou seja, gera implicantes, mas não necessariamente apenas primos essenciais.

Nos testes de desempenho, o algoritmo trouxe resultados satisfatórios do ponto de vista do autor, trazendo implicantes de uma função de 4 ou 5 variáveis em um tempo inferior à 0.01 segundo e com uma memória inferior à 0.1 mega byte.

Dessa forma, conclui-se que os objetivos do projeto foram alcançados e deseja-se que este trabalho inspire futuros estudantes a proporem novos métodos e técnicas de minimização de funções booleanas.

5. REFERÊNCIAS BIBLIOGRÁFICAS

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

BRAYTON, R. K,; HACHTEL, G. D.; HEMACHANDRA, L. A.; NEWTON, A. R.; SANGIOVANNI-VINCENTELLI, A. L. M.. (1982). **A comparison of logic minimization strategies using ESPRESSO: an APL program package for partitioned logic simulation**. *Proceedings of the IEEE International Symposium on Circuits and Systems, 1982*. New York, New York, USA: IEEE: 42–48.

BROWN, D.W. **A State-machine Synthesizer –SMS**. Proc 18th Design Automation Conference, Nashville, 1981.

CAIN R. G.; HONG S. J.; OSTAPKO, D. L. **MINI: A Heuristic Approach for Logic Minimization.** in IBM Journal of Research and Development, vol. 18, no. 5, pp. 443-458, Sept. 1974, doi: 10.1147/rd.185.0443.

CESARKALLAS. **Algoritmo de Quine-McCluskey.** Disponível em: http://cesarkallas.net/arquivos/faculdade/circuitos_logicos/teoria/diversos/QUINE_McCLUSKEY.pdf. Acesso em: 11.09.2022 às 10:30.

CHANG , C. L.; LEE, R. C. T.; SLAGLE, . J. R. **A New Algorithm for Generating Prime Implicants.** IEEE Trans. Cornput.C-19,304 (1970).

DIDÁTICA TECH. **A Linguagem Python.** Disponível em: https://didatica.tech/a-linguagem-python/. Acesso em: 28.09.2022 às 20 horas.

FERNANDES, M.I.G.M.A.S. **Estudo de Algoritmos de Minimização de Funções Booleanas Utilizando Diagramas de Decisão Binária**. Dissertação (Mestrado emEngenharia Eletrotécnica e de Computadores), Universidade do Porto, Porto, 1991.

FRANCISCANI, J.D.F. **Consenso iterativo: Geração de implicantes primos para minimização de funções booleanas com múltiplas saídas.** Dissertação (Mestrado em Engenharia Elétrica), UNESP, Ilha Solteira, 2016.

KARNAUGH, M. **The map method for synthesis of combinational logic circuits," in Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics**, vol. 72, no. 5, pp. 593-599, Nov. 1953, doi: 10.1109/TCE.1953.6371932.

KUNGL TEKNISKA HOGSKOLAN. **L3: Representations of functions.** Disponível em: https://people.kth.se/~dubrova/LScourse/LECTURES/lecture3.pdf. Acesso em: 07.10.2023 às 09:30.

LAWLESS, B. **Fundamental Digital Eletronics**. Lecture Notes, 2002.

MARTINEZ-CARBALLIDO, J.F; POWERS, V.M . **PRONTO: QUICK PLA PRODUCT REDUCTION.** Oregon State University, 1983

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

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

MILLER, R. E. **Switching Theory. Vol. I : Combinatorial Circuits.** JohnWiley&Sons,Inc.,New York, 1965.

MORREALE, E. **Recursive Operators for Prime Implicant and Irredundant normal Form Determination.** IEEE Trans.Cornput.C-19,504 (1970).

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

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

ROTH, J.P. **A Calculus and An Algorithm for the Multiple-Output 2 LevelMinimizationProblem.** Research Report RC 2007, IBM ThomasJ.WatsonResearchCenter, Yorktown Heights, New York, February 1968.

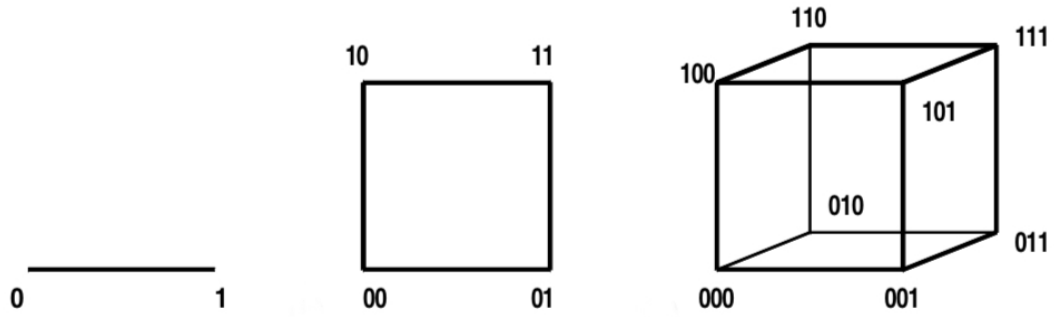
SILVA, D. M.. **Python: História e Ascendência.** Revista Programar, ed. 59, p. 96-97, Portugal, Fevereiro de 2018.

VEITCH, E. W. **A chart method for simplifying truth functions**. Proceedings of the Association for Computing Machinery, Jointly sponsored by the Association for Computing Machinery and the Mellon Institute, Pittsburgh, Pa., May 2 and 3, 1952, pp. 127–133.

APÊNDICE A - A Notação do Cubo

Ao longo deste trabalho, explicitou-se diversos métodos de minimização de circuitos digitais que utilizam a notação do cubo para realizar suas operações. A notação do cubo considera cada valor de entrada de uma função booleana como sendo um vértice de um cubo, sendo que, a dimensão de um cubo é dada por *n,* que representa o número de entradas que se faz presente, ou seja, se uma função apresenta apenas a entrada A, a notação é dada por um cubo de uma dimensão, ou, uma reta, com vértices 0 e 1. Expandindo a afirmação para uma função de duas entradas, A e B, tem-se um cubo bidimensional, ou, um quadrado, com 4 vértices, sendo eles: 00, 01, 10 e 11. Assim, chega-se a outra afirmação: O número de vértices de um cubo é dado por 2^n, sendo n o número de variáveis de entrada. A Figura 28 ilustra cubos de uma, duas e três dimensões.

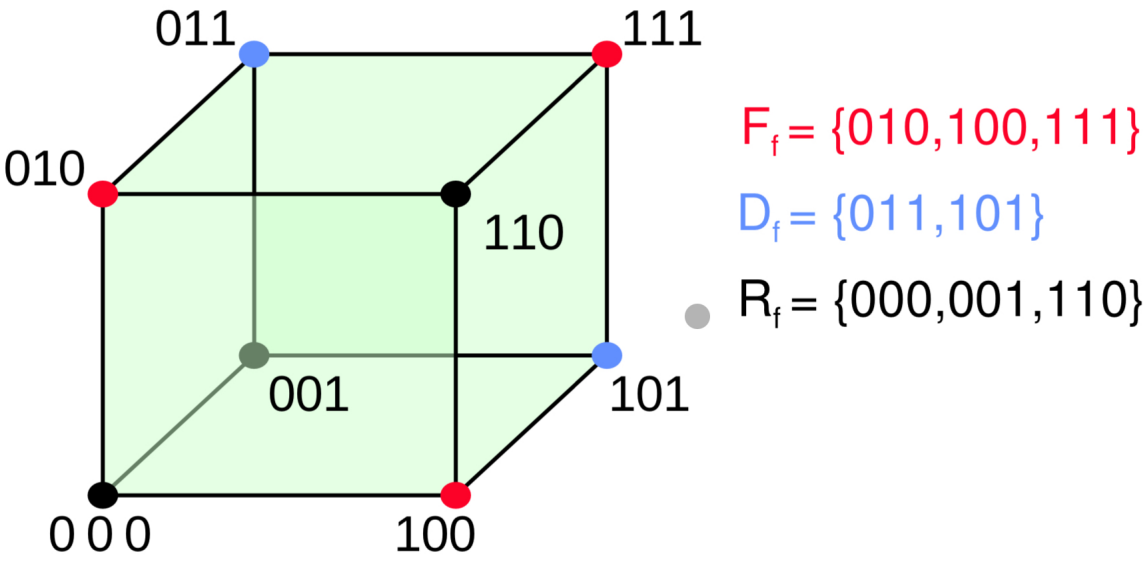
Figura 28 - Cubos de uma, duas e três dimensões



Fonte: (LAWLESS,2002)

Outra definição comumente utilizada na notação do cubo diz respeito aos *"sets",* ou, coberturas. A cobertura F diz respeito aos vértices do cubo que apresentam como saída o valor 1, a cobertura R é a cobertura dos vértices que apresentam como saída o valor 0 e a cobertura D, representa os vértices do cubo que apresentam como saída o valor de *don't care state*. A partir destas definições, os métodos realizam manipulações de cubos, por meio da álgebra de conjuntos para realizar a minimização de funções booleanas. A Figura 29 ilustra esta representação em exemplo hipotético.

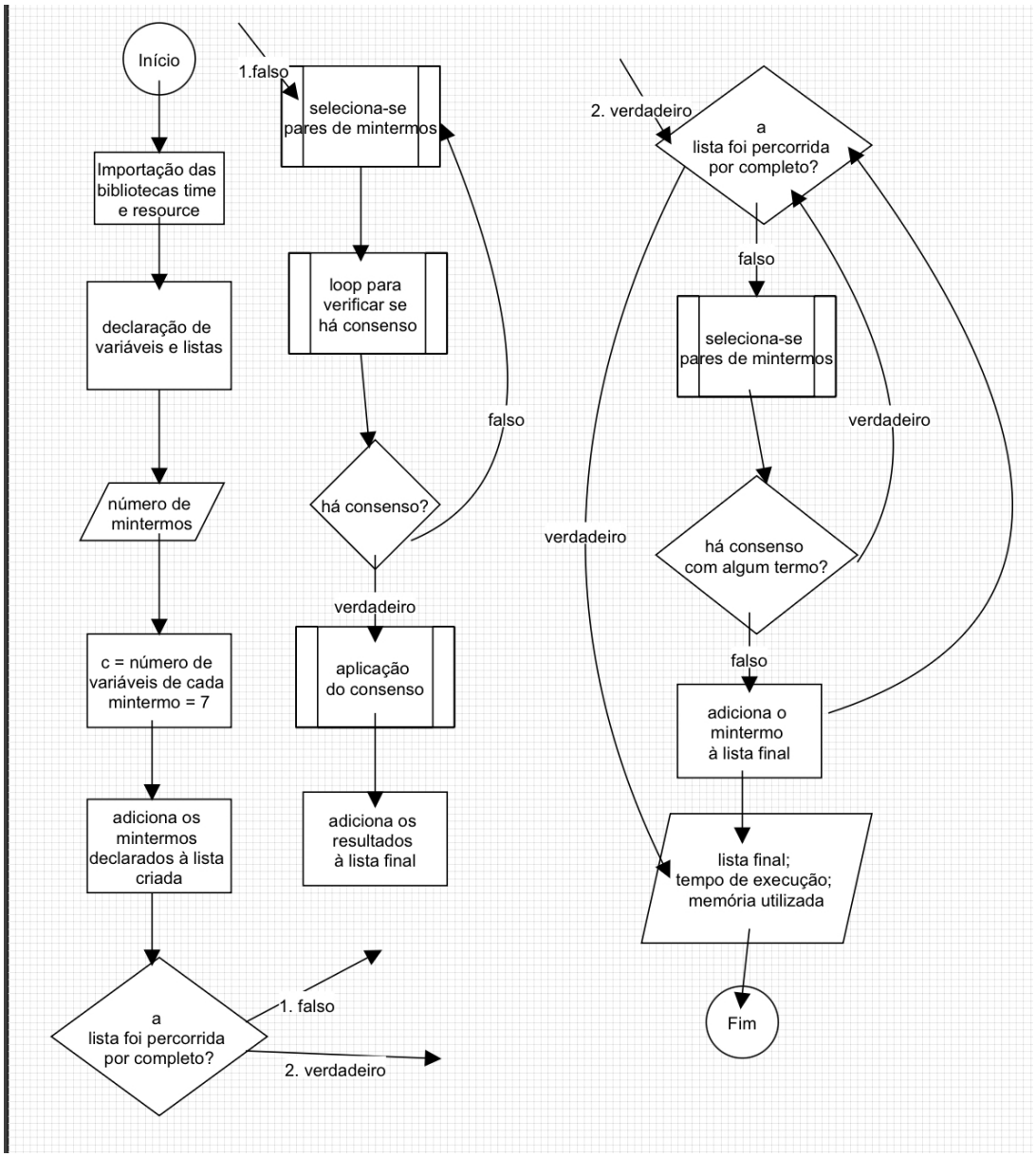
Figura 29 - Representação das coberturas



Fonte: (KUNGL TEKNISKA HOGSKOLAN, “s.d”)

APÊNDICE B - Fluxograma do Algoritmo

A Figura 30, ilustra o fluxograma simplificado do algoritmo.



APÊNDICE C - Código do Programa e Comentários de Cada Linha

O programa começa importando duas bibliotecas, ‘time’e ‘resource’, e definindo ti como tempo de processamento.

Linha 1: lst = [] , A primeira linha declara uma lista, a mesma tem a finalidade de armazenar os mintermos da função booleana que o usuário irá definir

Linha 2: lst_consenso = [], A segunda linha declara uma lista que armazenará o resultado de cada operação de consenso

Linha 3: lst_iteracao = [], A terceira linha declara outra lista, esta, por sua vez, armazenará os mintermos após a execução do programa ser concluída. Será a lista que trás o resultado final

Linha 4: m = 0

Linha 5: n = int(input("Número de mintermos da função: ")) A quinta linha refere-se ao ponto do programa em que o usuário irá indicar o número de mintermos que a função booleana a ser minimizada apresenta

Linha 6: c = 7 , A sexta linha declara a variável c, que nos indica o número máximo de variáveis que podem conter os mintermos da função a ser minimizada.

Linha 7: for i in range(0, n):, A sétima linha dá início a um loop, que irá de zero até o número máximo de mintermos

Linha 8: min = int(input(),2); , A oitava linha declara a variável min, que indica cada minterno da função a ser declarado pelo usuário

Linha 9: min_b = bin(min); , A nona linha declara a variável min_b, que é indicado por cada mintermo na forma binária

Linha 10: mintermo = str(min_b)[2:].zfill(c) , A décima linha do programa declara a variável mintermo, que diz respeito à formatação dos mintermos com 7 variáveis

Linha 11: lst.append(mintermo), A décima primeira linha adiciona cada mintermo a lista lst

Linha 12: while m < 1:, A linha de número 12 trás uma condição que sugere a execução da sequência do programa, apenas se a variável m for menor que 1

Linha 13: m = m+1; A linha 13 soma 1 à variável m

Linha 14: j = int(-1); , A linha 14 declara a variável inteira j, que funcionará como um índice, e, atribui o valor inicial -1 para j

Linha 15: b = int(0); A linha 15 declara a variável inteira b, que funcionará como um contador, e, atribui o valor inicial 0 para b

Linha 16: k = int; A linha 16 declara a variável inteira k, que funcionará como um índice

Linha 17: dontcare = 'X'; A linha 17 declara a variável dontacre e atribui a *string* X à variável

Linha 18: consenso_ = int; A décima oitava linha, declara um índice que verifica se há consenso entre os termos. O índice, chamado de consenso_, é um inteiro.

Linha 19: print('Resultado da iteração'), Na linha 19 é executado um comando de escrita, em que a frase 'Resultado da iteração' é solicitada a ser escrita na tela do programa

Linha 20: while j <= len(lst): A linha de número 20 trás uma condição que sugere a execução da sequência do programa, apenas se a variável j for menor ou igual ao tamanho da lista lst

Linha 21: j = j+1; , Na linha 21, o valor 1 é somado ao valor de momento da variável j

Linha 22: b = b+1; , Na linha 22, o valor 1 é somado ao valor de momento da variável b

Linha 23: for i in range(b, len(lst)): , Na linha 23, a fim de começar a selecionar os pares de mintermos, uma condição "for" é chamada, selecionando pares de b até o tamanho da lista

Linha 24: consenso_ = 1; , Na linha 24, é atribuído o valor 1 para a variável consenso_

Linha 25: numerostr1 = lst[j]; , Na linha 25, uma variável é criada para carregar o valor do mintermo que está na posição "j"da lista lst

Linha 26: numerostr2 = lst[i]; , Na linha 26, uma variável é criada para carregar o valor do mintermo que está na posição "i"da lista lst

Linha 27: for k in range(0,c): , Na linha 27, a fim de verificar se há consenso entre os mintermos uma condição "for" é chamada, verificando de 0 até o valor de c, os bits de cada mintermo selecionado

Linha 28: lst_consenso = [];, Na linha 28, a lista lst_consenso é limpa, para armazenar novos valores

Linha 29: if numerostr1[k] == numerostr2[k] or numerostr1[k] == 'X' or numerostr2[k] == 'X':, Na linha 29, uma condição "if" é estabelecida, nela, caso o bit de mesma posição nos dois mintermos seja igual ou um dos dois seja igual a dontcare a linha abaixo será executada

Linha 30: consenso_ = consenso_; Na linha 30, a variável consenso_ permanecerá com o mesmo valor, se a condição "if"for respeitada

Linha 31: else:, Na linha 31, a condição "else" é chamada. A mesma significa que caso a condição "if"não seja verdadeira, o comando a seguir deverá ser executado

Linha 32: consenso_ = consenso_-1;, Na linha 32, a variável consenso_ será subtraída no valor 1, se a condição “if” não for respeitada

Linha 33: if consenso_ == 0: , Na linha 33 uma condição “if” é estabelecida, nela, caso a variável consenso_ seja igual a zero, a linha baixo será executada, e assim por diante

Linha 34: for k in range(0,c): , Na linha 34, começa a aplicação da operação de consenso. Uma condição “for” é chamada, verificando de 0 até o valor de c, os bits de cada mintermo selecionado

Linha 35: if numerostr1[k]=='0' and numerostr2[k]=='1'or numerostr1[k]=='1' and numerostr2[k]=='0'or numerostr1[k]=='X' and numerostr2[k]=='X':, Na linha 35 uma condição “if” é estabelecida, nela, caso o bit selecionado do mintermo numerostr1 for igual a zero e o bit selecionado do mintermo numerostr2 for igual a 1ou o bit selecionado do mintermo numerostr1 for igual a zero e o bit selecionado do mintermo numerostr2 for igual a 0 ou ainda se os dois bits forem iguais a dontcare, o comando da linha abaixo será executado

Linha 36: lst_consenso.append(dontcare); , Na linha 36, dontcare será adicionado à lista lst_consenso, se e somente se, a condição da linha 35 for verdadeira

Linha 37: elif numerostr1[k]=='0' and numerostr2[k]=='0' or numerostr1[k]=='0' and numerostr2[k]=='X' or numerostr1[k]=='X' and numerostr2[k]=='0':, Na linha 37 uma condição “if” é estabelecida, nela, caso o bit selecionado do mintermo numerostr1 for igual a dontcare e o bit selecionado do mintermo numerostr2 for igual a zero ou o bit selecionado do mintermo numerostr1 for igual a zero e o bit selecionado do mintermo numerostr2 for igual a dontcare ou ainda se os dois bits forem iguais a zero, o comando da linha abaixo será executado

Linha 38: lst_consenso.append('0'); , Na linha 38, 0 será adicionado à lista lst_consenso, se e somente se, a condição da linha 37 for verdadeira

Linha 39: elif numerostr1[k]=='1' and numerostr2[k]=='1'or numerostr1[k]=='1' and numerostr2[k]=='X'or numerostr1[k]=='X' and numerostr2[k]=='1': , Na linha 39 uma condição “if” é estabelecida, nela, caso o bit selecionado do mintermo numerostr1 for igual a um e o bit selecionado do mintermo numerostr2 for igual a dontcare ou o bit selecionado do mintermo numerostr1 for igual a dontcare e o bit selecionado do mintermo numerostr2 for igual a um ou ainda se os dois bits forem iguais a um, o comando da linha abaixo será executado

Linha 40: lst_consenso.append('1'); , Na linha 40, 1 será adicionado à lista lst_consenso, se e somente se, a condição da linha 39 for verdadeira

Linha 41: numerostr3 = str((''.join(lst_consenso))); , Na linha 41, a variável numerostr3 é declarada e a ela é atribuída uma string, sendo esta, a transformação da lst_consenso do formato lista para string

Linha 42: lst_iteracao.append(numerostr3); , Na linha 42, a string numerostr3 com o resultado da operação é adicionada à lista de resultados, chamada de lst_iteracao. Ou seja, ao final de cada operação de consenso, o resultado é armazenado na lst_iteracao

Linha 43: z = int(-1); , Na linha 43, a variável z é criada e a ela é atribuído o valor inteiro -1

Linha 44: while z <= len(lst): , A linha de número 44 nos trás uma condição que sugere a execução da sequência do programa, apenas enquanto a variável z for menor ou igual ao tamanho da lista lst

Linha 45: z = z+1; , Caso a condição acima seja respeitada, na linha 45, a variável z receberá o valor de z somado ao número 1

Linha 46: consenso = 0; , Na linha 46, a variável consenso é declarada e a ela é atribuído o valor 0

Linha 47: for i in range(0, len(lst)):, Na linha 47 a fim de selecionar os pares de mintermos, uma condição "for" é chamada, selecionando pares de 0 até o tamanho da lista

Linha 48: numerostr1 = lst[z]; , Na linha de número 48, a variável numerostr1 recebe um mintermo. Este mintermo ocupa a posição do valor atual de z, na lista lst

Linha 49: numerostr2 = lst[i]; , Na linha de número 49, a variável numerostr2 recebe um mintermo. Este mintermo ocupa a posição do valor atual de i, na lista lst

Linha 50: if numerostr1[0:1]!= numerostr2[0:1]and numerostr1[1:c] == numerostr2[1:c]: , Na linha 50, começa uma nova verificação de consenso entre os termos, dessa vez a fim de determinar os termos que não apresentam consenso com nenhum outro, e, portanto, são implicantes primos essenciais. A primeira condição, sugere que se apenas o primeiro bit, da esquerda para a direita, for diferente entre os pares de mintermos selecionados, o comando da linha abaixo será executado

Linha 51: consenso = 1; , Na linha 51, se e somente se, a condição da linha 50 for respeitada, a variável consenso será igual a 1

Linha 52: elif numerostr1[0:1] == numerostr2[0:1]and numerostr1[1:2] != numerostr2[1:2]and numerostr1[2:c] == numerostr2[2:c]: , A linha 52 sugere que se apenas o segundo bit, da esquerda para a direita, for diferente entre os pares de mintermos selecionados, a condição da linha abaixo será executada

Linha 53: consenso = 1; , Na linha 53, se e somente se, a condição da linha 52 for respeitada, a variável consenso será igual a 1

Linha 54: elif numerostr1[0:2] == numerostr2[0:2]and numerostr1[2:3] != numerostr2[2:3]and numerostr1[3:c] == numerostr2[3:c]: , A linha 54 sugere que se apenas o terceiro bit, da esquerda para a direita, for diferente entre os pares de mintermos selecionados, a condição da linha abaixo será executada

Linha 55: consenso = 1; , Na linha 55, se e somente se, a condição da linha 54 for respeitada, a variável consenso será igual a 1

Linha 56: elif numerostr1[0:3] == numerostr2[0:3]and numerostr1[3:4] != numerostr2[3:4]and numerostr1[4:c] == numerostr2[4:c]: , A linha 56 sugere que se apenas o quarto bit, da esquerda para a direita for diferente entre os pares de mintermos selecionados, a condição da linha abaixo será executada

Linha 57: consenso = 1; , Na linha 57, se e somente se, a condição da linha 56 for respeitada, a variável consenso será igual a 1

Linha 58: elif numerostr1[0:4] == numerostr2[0:4]and numerostr1[4:5] != numerostr2[4:5] and numerostr1[5:c] == numerostr2[5:c]: , A linha 58 sugere que se apenas o quinto bit, da esquerda para a direita, for diferente entre os pares de mintermos selecionados, a condição da linha abaixo será executada

Linha 59: consenso = 1; , Na linha 59, se e somente se, a condição da linha 58 for respeitada, a variável consenso será igual a 1

Linha 60: elif numerostr1[0:5] == numerostr2[0:5]and numerostr1[5:6] != numerostr2[5:6] and numerostr1[6:c] == numerostr2[6:c]: , A linha 60 sugere que se apenas o sexto bit, da esquerda para a direita, for diferente entre os pares de mintermos selecionados, a condição da linha abaixo será executada

Linha 61: consenso = 1; , Na linha 61, se e somente se, a condição da linha 60 for respeitada, a variável consenso será igual a 1

Linha 62: elif numerostr1[0:6] == numerostr2[0:6]and numerostr1[6:c] != numerostr2[6:c]:, A linha 62 sugere que se apenas o sétimo bit, da esquerda para a direita, for diferente entre os pares de mintermos selecionados, a condição da linha abaixo será executada

Linha 63: consenso = 1; , Na linha 63, se e somente se, a condição da linha 62 for respeitada, a variável consenso será igual a 1

Linha 64: if consenso != 1:, Na linha 64 uma condição “if” é estabelecida, nela, caso a variável consenso seja diferente de 1, a condição da linha abaixo será executada

Linha 65: lst_iteracao.append(numerostr1); Na linha 65, se e somente se, a condição da linha 64 for respeitada, o mintermo correspondente ao valor atual de numerostr1 é adicionado à lista lst_iteracao

Linha 66: if z == len(lst)-1:, Na linha 66 uma condição “if” é estabelecida, nela, caso a variável z seja igual a o tamanho da lista lst subtraído de1, a condição da linha abaixo será executada

Linha 67: break, Na linha 67, caso a condição da linha 66 seja respeitada, o programa deve parar

Linha 68: print(lst_iteracao); , Na linha 68, a lista que contém os mintermos após a operação é apresentada na tela do usuário

Linha 69: tf = time.process_time(); , Na linha 69 é definido o tempo final de execução tf;

Linha 70: ttotal = tf - ti; , Na linha 70 é definido ttotal como sendo a diferença entre o tempo final e o inicial, de execução

Linha 71: print('Tempo de excução:', ttotal, 'seconds'), A linha 71 exibe na tela do usuário o tempo de execução total do programa, em segundos.

Linha 72: memMb=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0, A linha 72 define a memória utilizada pelo programa

Linha 73: print ('Memória utilizada', memMb,'Mb'), A linha 73 exibe na tela do usuário a memória utilizada pelo programa, em Mbyte

A seguir, temos o código completo, em formado de texto estruturado:

```
import time
import resource
ti = time.process_time() #tempo inicial
# iteração
lst = [] #lista que armazena os primeiros mintermos
lst_consenso = [] #lista que armazena o resultado de cada operação de consenso
```

```
lst_iteracao = [] #lista que armazena os mintermos após a operação de consenso ser concluída
m = 0

n = int(input("Número de mintermos da função: ")) # inserção do número de mintermos da
função
c = 7  #número máximo de variáveis
# adicionando os mintermos a lista de mintermos
for i in range(0, n):
   min = int(input(),2);
   min_b = bin(min); #mintermo em forma binária
   mintermo = str(min_b)[2:].zfill(c) #formatação dos mintermos com 7 casas
   lst.append(mintermo) # adiciona cada mintermo a lista
while m < 1:
   m = m+1;
   j = int(-1); #índice
   b = int(0); #contador
   k = int; #índice
   dontcare = 'X';
   consenso_ = int; #índice que verifica se há consenso entre os termos
   print('Resultado da iteração')
   while j <= len(lst):
         j = j+1;
         b = b+1;
         for i in range(b, len(lst)): #selecionando pares de mintermos
            consenso_ = 1;
            numerostr1 = lst[j];
            numerostr2 = lst[i];
            for k in range(0,c): # verificando se há consenso entre os mintermos
               lst_consenso = [];
               if  numerostr1[k] == numerostr2[k] or numerostr1[k] == 'X' or numerostr2[k] ==
'X':
                  consenso_ = consenso_;
               else:
                  consenso_ = consenso_-1;
            if consenso_ == 0:
               for k in range(0,c): #aplicação da operação de consenso
                  if numerostr1[k]=='0' and numerostr2[k]=='1'or numerostr1[k]=='1' and
numerostr2[k]=='0'or numerostr1[k]=='X' and numerostr2[k]=='X':
                     lst_consenso.append(dontcare);
                  elif numerostr1[k]=='0' and numerostr2[k]=='0' or numerostr1[k]=='0' and
numerostr2[k]=='X' or numerostr1[k]=='X' and numerostr2[k]=='0':
                     lst_consenso.append('0');
                  elif numerostr1[k]=='1' and numerostr2[k]=='1'or numerostr1[k]=='1' and
numerostr2[k]=='X'or numerostr1[k]=='X' and numerostr2[k]=='1':
                     lst_consenso.append('1');
```

```
                numerostr3 = str((''.join(lst_consenso)));#transformação de formato lista para
string
                lst_iteracao.append(numerostr3); #adição do resultado do consenso a lista de
resultados
    #verificação dos termos que não apresentam consenso
    z = int(-1);
    while z <= len(lst):
        z = z+1;
        consenso = 0;
        for i in range(0, len(lst)):
            numerostr1 = lst[z];
            numerostr2 = lst[i];
            if  numerostr1[0:1]!= numerostr2[0:1]and  numerostr1[1:c] == numerostr2[1:c]:
                    consenso = 1;
            elif numerostr1[0:1] == numerostr2[0:1]and numerostr1[1:2] != numerostr2[1:2]and
numerostr1[2:c] == numerostr2[2:c]:
                    consenso = 1;
            elif  numerostr1[0:2] == numerostr2[0:2]and  numerostr1[2:3] != numerostr2[2:3]and
numerostr1[3:c] == numerostr2[3:c]:
                    consenso = 1;
            elif  numerostr1[0:3] == numerostr2[0:3]and  numerostr1[3:4] != numerostr2[3:4]and
numerostr1[4:c] == numerostr2[4:c]:
                    consenso = 1;
            elif  numerostr1[0:4] == numerostr2[0:4]and numerostr1[4:5] != numerostr2[4:5] and
numerostr1[5:c] == numerostr2[5:c]:
                    consenso = 1;
            elif  numerostr1[0:5] == numerostr2[0:5]and numerostr1[5:6] != numerostr2[5:6] and
numerostr1[6:c] == numerostr2[6:c]:
                    consenso = 1;
            elif  numerostr1[0:6] == numerostr2[0:6]and  numerostr1[6:c] != numerostr2[6:c]:
                    consenso = 1;
        if consenso != 1:
            lst_iteracao.append(numerostr1);
        if z == len(lst)-1:
          break
    print(lst_iteracao); #lista após a aplicação do consenso entre os mintermos
#tempo final
tf = time.process_time()

#tempo total
ttotal = tf - ti
print('Tempo de excução:', ttotal, 'seconds')
memMb=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024.0/1024.0
print ('Memória utilizada', memMb,'Mb')
```