Software educacional para ensino de minimização de expressões lógicas

Educational software for teaching logic circuit minimization

Luana Cristina Guerreiro Campos (orientado) *, Gustavo Henrique Paetzold (orientador)†

RESUMO

Circuitos digitais podem ser representados como funções lógicas. Nesses circuitos as variáveis são representadas pela presença ou ausência de tensão. Com o intuito de reduzir a quantidade de portas lógicas utilizadas é feita a minimização da expressão lógica. A minimização pode ser feita com o Mapa de Karnaugh ou o algoritmo de Quine-McCluskey, o último mais apropriado para solução computacional. Esse trabalho desenvolveu um software educacional que realiza a minimização de expressões lógicas utilizando ambos os métodos mencionados. Foram utilizadas duas técnicas para computar a resposta do algoritmo de Quine-McCluskey. A primeira técnica faz uso de um algoritmo polinomial guloso que obtém apenas uma aproximação. A segunda técnica faz uso de um algoritmo exponencial recursivo, otimizado com programação dinâmica. Tanto o algoritmo quanto as técnicas foram implementadas utilizando a linguagem de programação *Python*. Uma interface gráfica, desenvolvida utilizando a *framework Qt*, acompanha o programa, que está disponível para *download*.

Palavras-chave: Quine-McCluskey. Mapa de Karnaugh. Programação dinâmica. Cobertura de conjuntos

ABSTRACT

Digital circuits can be represented as logic functions. In these circuits, variables are represented by the presence or absence of voltage. In order to reduce the amount of logic gates used, the logic expression is minimized. The minimization can be done with the Karnaugh Map or the Quine-McCluskey algorithm, the last one more suitable for computational solutions. This work developed an educational software that performs the minimization of logical expressions using both methods mentioned. Two techniques were used to compute the response of the Quine-McCluskey algorithm. The first technique uses a greedy polynomial algorithm that obtains only an approximation. The second technique uses a recursive exponential algorithm, optimized with dynamic programming. Both the algorithm and the techniques were implemented using the *Python* programming language. A graphical interface, developed using the *Qt framework*, accompanies the program, which is available for download.

Keywords: Quine-McCluskey. Karnaugh Map. Dynamic programming. Set cover

1 INTRODUÇÃO

Circuitos digitais podem ser descritos como expressões lógicas, que são compostas por variáveis e operadores lógicos. Nesses circuitos, o valor de uma variável é indicada pela tensão, ou a ausência dela e os operadores são representados por portas lógicas.

* Departamento de Engenharia de Computação; luanacampos@alunos.utfpr.edu.br; https://orcid.org/0000-0002-9761-5794.
† Departamento de Engenharia de Computação; ghpaetzold@utfpr.edu.br; https://orcid.org/0000-0001-9951-050X.

Com o crescimento rápido da quantidade de portas lógicas a serem utilizadas em um circuito, há a necessidade de otimização no processo de fabricação. Um dos principais objetivos da fabricação de circuitos digitais é manter o menor número de portas lógicas possível, o que irá reduzir o custo de produção desses sistemas.

O método mais conhecido de simplificação é o Mapa de Karnaugh que fornece um procedimento simples e direto para minimização de expressões lógicas, entretanto, o método de mapeamento não é conveniente quando excede cinco variáveis. Para minimizar expressões lógicas com muitas variáveis, é utilizado o algoritmo de Quine-McCluskey, sua forma tabular o faz mais eficiente para uso computacional. (ZHAO; WANG; SUN, 2017)

Foi desenvolvido uma aplicação *Desktop* implementada na linguagem de programação *Python* que exibe a execução passo-a-passo do algoritmo Quine-McCluskey. A framework *Qt* foi utilizada para a criação da interface gráfica do projeto. Em um trabalho de pesquisa prévio foi desenvolvido uma interface para o Mapa de Karnaugh que foi incorporado no projeto.

2 MATERIAIS E MÉTODOS

A tabela-verdade possui todas as combinações de entradas e suas respectivas saídas. O Quadro 1 mostra um exemplo de tabela-verdade de três variáveis. As colunas A, B e C representam as variáveis de entrada, e a coluna Y representa a saída. Cada entrada possui sua representação decimal, também descrita no quadro apresentado. São chamados de *mintermos* o conjunto de entradas que possuem saída verdadeira. A função da Eq. (1) descreve a tabela-verdade do Quadro 2 (COUDERT, 1994).

Quadro 1 – Tabela verdade.

Decimal	A	B	C	Y
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Fonte: Autoria própria (2021).

$$F(A,B,C) = \sum m(1,2,3,4,5,6) \tag{1}$$

O objetivo do algoritmo de Quine-McCluskey é realizar a minimização de funções lógicas, atividade crucial para otimização do processo de fabricação de circuitos digitais (PRASAD; BEG; SINGH, 2009). Nas seguintes seções são descritas as etapas do algoritmo.

2.1 Cálculo dos implicantes primos

Um implicante de um função lógica é qualquer produto entre os mintermos, Quine define um implicante primo como um implicante que é mínimo, ou seja, não é possível ser minimizado (QUINE, 1952). Inicialmente os mintermos são ordenados pela quantidade de números '1' na notação binária. No Quadro 2, a coluna Caixa representa a quantidade de números '1' dos mintermos, e na coluna adjacente está a representação binária e

decimal dos mintermos.

Quadro 2 – Primeira iteração

Caixa	Mintermos	
	Binário	Decimal
1	001	1
	010	2
	100	4
2	011	3
	101	5
	110	6

Fonte: Autoria própria (2021).

A combinação entre dois implicantes é a aplicação da propriedade da álgebra booleana apresentada na Eq. (2), ou seja, implicantes que possuem apenas uma variável distinta serão combinados (PAULA SANCHES, 2017). Haverá um tentativa de combinação entre os mintermos de caixas adjacentes e, se for possível, a variável distinta será substituída por 'X' e seus decimais serão unidos em um conjunto. O Quadro 3 exemplifica o processo.

$$AB + A\bar{B} = A \tag{2}$$

Quadro 3 – Segunda iteração

Caixa	Implicantes	
	Binário	Conjunto
1	0X1	{1, 3}
	X01	{1, 5}
	01X	{2, 3}
	X10	{2, 6}
	10X	{4, 5}
	1X0	{4, 6}

Fonte: Autoria própria (2021).

2.2 Construção da matriz de cobertura

A matriz de cobertura consiste em linhas rotuladas com os implicantes primos na forma binária e em colunas rotuladas com os mintermos. Em cada linha serão marcados os decimais que compõem o conjunto do implicante primo. O objetivo final é escolher subconjuntos da coleção S de implicantes primos que irá cobrir todos os mintermos U (PAULA SANCHES, 2017).

Quadro 4 – Matriz de cobertura

	1	**2**	**3**	**4**	**5**	**6**
1X0				X		X
01X		X	X			
X10		X				X
X01	X				X	
10X				X	X	
0X1	X		X			

Fonte: Autoria própria (2021).

2.3 Cobertura de conjuntos

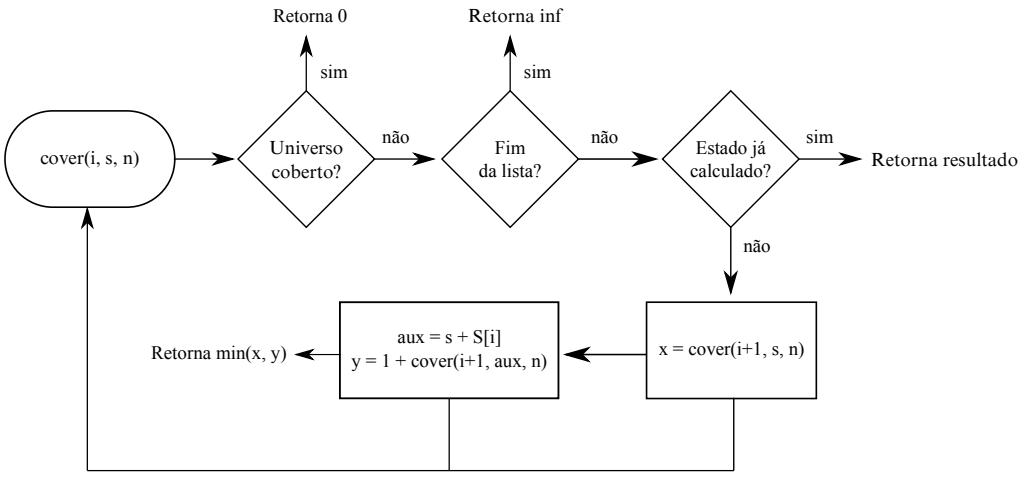
O problema de cobertura de conjuntos consiste em encontrar o menor número de subconjuntos de uma coleção S que cobre o universo U de termos, resolver esse problema em particular é NP-difícil (COUDERT, 1994). Existe um algoritmo guloso polinomial que obtém uma aproximação para o problema (SLAVIK, 1997). Nesse trabalho, além do algoritmo guloso, foi desenvolvido uma abordagem utilizando programação dinâmica que produz a resposta mínima, porém, é um algoritmo exponencial.

Um algoritmo guloso sempre faz a escolha que parece ser a melhor no momento, ou seja, faz uma escolha localmente ótima, esperando que essa escolha leve a uma solução globalmente ótima (CORMEN, 2012). A cada passo o algoritmo escolhe um conjunto não utilizado que cobre o maior número de termos restantes até cobrir o universo U de termos.

Programação dinâmica é uma técnica de otimização utilizada para resolver, usualmente, problemas recursivos. A ideia é armazenar os resultados dos subproblemas para que não seja necessário recalcula-los (CORMEN, 2012). A cada passo o algoritmo testa as possibilidades de cada conjunto de S pertencer ou não á resposta final. A função *cover* recebe o índice atual da lista de conjunto S, e um *set s* que contém os termos já cobertos. A função retorna a menor quantidade de conjuntos para cobrir o universo U.

Inicialmente é verificado se o *set s* recebido já cobre todos os termos de U, se sim, retorna zero já que não será necessário utilizar mais conjuntos. Em seguida, é verificado se o índice atual está fora do limite de S, se for o caso, é retornado um valor alto para simbolizar que não foi possível a cobertura de U. Então, é verificado se o estado atual já foi calculado, assim, será retornado o valor armazenado. Por último, o algoritmo testa duas hipóteses, utilizar ou não o conjunto atual, e retornará a hipótese que tem a menor quantidade de conjuntos utilizados. Assim, é possível refazer o caminho ótimo calculado pelo algoritmo, e armazenar os conjuntos pertencentes a resposta. O fluxograma da Figura 1 mostra o algoritmo mencionado.

Figura 1 – Fluxograma

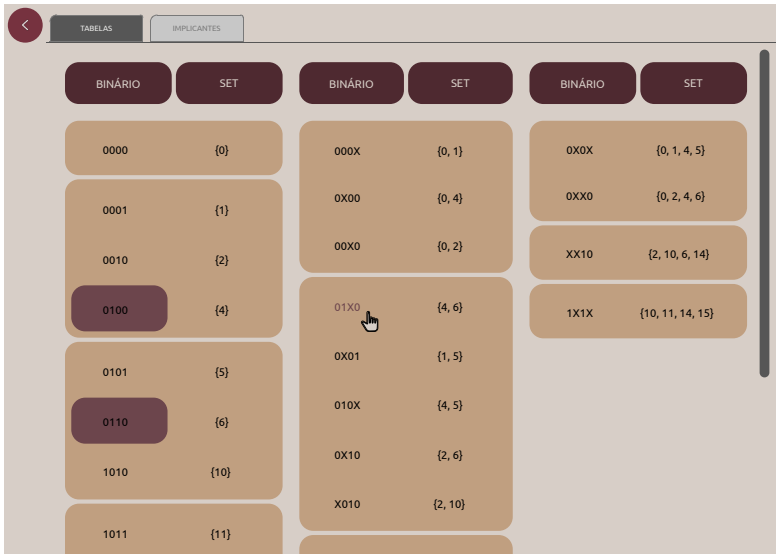


Fonte: Autoria própria (2021).

3 RESULTADOS

O programa suporta a entrada de tabelas-verdades em formato *csv* e *tsv*. A seguir são apresentadas imagens da interface gráfica do programa desenvolvido. A Figura 2 apresenta os quadros da execução passo-a-passo do algoritmo. A informação exibida nessa janela é similar aos Quadros 2 e 3. A fim de deixar o software mais didático, ao passar o cursor do mouse sobre um implicante, é possível ver quais implicantes foram combinados para sua criação.

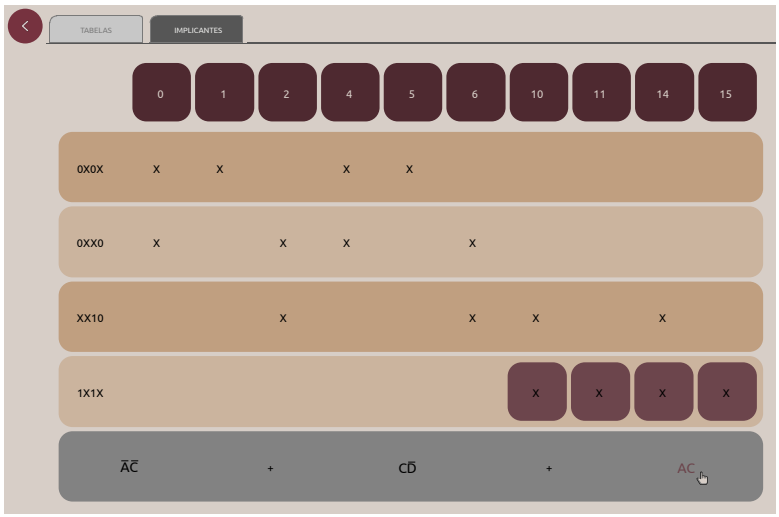
Figura 2 – Aba das tabelas



Fonte: Autoria própria (2021).

A Figura 3 mostra a resposta e a matriz de cobertura, de forma análoga ao Quadro 4. Outro recurso que deixa o software mais didático é o realce dos implicantes ao passar o cursor do mouse sobre um termo da resposta.

Figura 3 – Aba dos implicantes primos



Fonte: Autoria própria (2021).

4 CONCLUSÕES

Circuitos digitais podem ser representados como expressões lógicas. Para reduzir o número de portas lógicas é utilizado a minimização de funções lógicas. A minimização pode ser feita com o Mapa de Karnaugh ou o algoritmo de Quine-McCluskey, o último é mais adequado para métodos computacionais.

Nesse trabalho foi implementado o algoritmo de Quine-McCluskey com o intuito de minimizar funções lógicas. Foram implementados dois algoritmos para a resolução do problema de cobertura de conjuntos, um guloso, que não garante a solução ótima, e um recursivo otimizado com programação dinâmica que garante a solução mínima. Uma aplicação *Desktop* acompanha o programa e está disponível para *download*.

AGRADECIMENTOS

Agradeço a Fundação Araucária por fomentar essa pesquisa com o fornecimento de uma bolsa PIBITI durante o período de um ano. Também agradeço ao professor Ricardo por sanar minhas dúvidas e ao professor Gustavo por me orientar.

REFERÊNCIAS

CORMEN, Thomas. **Algoritmos - Teoria e Prática**. [S.l.]: Elsevier, 2012. v. 3.

COUDERT, Olivier. Two-level logic minimization: an overview. **Integration**, v. 17, n. 2, p. 97–140, 1994. ISSN 0167-9260. DOI: https://doi.org/10.1016/0167-9260(94)00007-7. Disponível em: .

PAULA SANCHES, Aline de. **EMPREGO DO MÉTODO DE QUINE-MCCLUSKEY ESTENDIDO PARA GERAR CIRCUITO MÍNIMO COM ESTRUTURAS ESOP (XOR- XNOR)**. Jul. 2017. Diss. (Mestrado) – Universidade Estadual Paulista.

PRASAD, P. W. Chandana; BEG, Azam; SINGH, Ashutosh Kumar. Effect of Quine-McCluskey simplification on Boolean space complexity. In: 2009 Innovative Technologies in Intelligent Systems and Industrial Applications. [S.l.: s.n.], 2009. P. 165–170. DOI: 10.1109/CITISIA.2009.5224219.

QUINE, W. V. The Problem of Simplifying Truth Functions. **The American Mathematical Monthly**, Taylor I& Francis, v. 59, n. 8, p. 521–531, 1952. DOI: 10.1080/00029890.1952.11988183. eprint: https://doi.org/10.1080/00029890.1952.11988183. Disponível em: .

SLAVIK, Petr. A Tight Analysis of the Greedy Algorithm for Set Cover. **Journal of Algorithms**, v. 25, n. 2, p. 237–254, 1997. ISSN 0196-6774. DOI: https://doi.org/10.1006/jagm.1997.0887. Disponível em: .

ZHAO, Shuguang; WANG, Chaozheng; SUN, Junling. Improvement and implementation of quine-McCluskey algorithm for synthesis of reversible logic circuits. In: 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC). [S.l.: s.n.], 2017. P. 640–642. DOI: 10.1109/IAEAC.2017.8054094.