Implementation of the Quine-McCluskey Boolean simplification algorithm in an interactive smartphone game.

Oleksii Levkovskyi, and Kevin Lopez
Florida Atlantic University, United States,
{*olevkovskyi2015, klopez2013*}*@fau.edu*

Abstract- Logic Design is a compulsory prerequisite course in most computer science, computer engineering and electrical engineering bachelor degree programs in the United States. The most popular and/or desirable outcome for undergraduate students in computer science is to secure an industrial position where programming is required as a skill in one form or another. While the theoretical and practical aspects of "Introduction to Logic Design" are tailored to moderately expose the student to a broader body of knowledge, the learning curve for a programming-oriented individual persists and may pose an academic performance problem. We have designed a game for mobile devices designed to aid students taking the Logic Design class in understanding the practical side of the material, namely the techniques associated with using Karnaugh Maps. The objective is to make the process of simplifying Boolean expressions using Karnaugh Maps both a challenging and rewarding experience, reduced to the compact and approachable format of a smartphone game.

Keywords: Karnaugh Maps, Quine-McCluskey Algorithm, Engineering Education

I. Introduction

One subset of Boolean algebra is the collection of various methods and techniques that describe minimizing equations which combine Boolean variables, resulting in a value that is either true (1) or false (0) (given that the realm of operation is active high logic), or vice versa. In the process of designing a logical (and/or electrical, if implemented in hardware) circuit, one typically aims to obtain the simplest (finitely minimized) Boolean expression, since its complexity and demand for resources will ultimately determine the cost and respective complexity of the implementation, as a non-physical prototype, hardware prototype, or otherwise. One such minimization technique is the use of Karnaugh Maps [1], used primarily for small design implementation, e.g., logical expressions combining up to 6 unique Boolean variables in total. A combination of greater magnitude than six variables increases the complexity of manual minimization using Karnaugh Maps to such an extent that the solution is no longer practical [2].

As a result, other industry-standard methods have emerged. The Quine-McCluskey minimization algorithm, designed to be a programmable replacement for solving Karnaugh Maps that can handle solving expressions with a number of variables greater than six [3]. As a key component of our interactive mobile Karnaugh Map solver, we have implemented the Quine-McCluskey algorithm in the Swift programming language [4] and demonstrated our implementation's capacity to produce multiple optimal solutions.

II. Implementation

A. *Quine-McCluskey as a Programmable Algorithm*

Our implementation was written using the object-oriented Swift programming language, which is native to the iOS mobile platform, our target for the Karnaugh Map solver. The implementation nests three recursive function calls at its core:

1. *DERIVE-NTH-ORDER*
2. *REDUCE-PRIMES*
3. *PETRICKIFY*

The first call is used to reduce the table of minimum terms (combinations of Boolean values) of the logical expression to a set of prime implicants or, in other words, a set of minimum terms such that its subset covers the entire function. The second call in the list reduces the set of prime implicants to its subset, or a set of essential prime implicants, whose values are guaranteed to cover the entire function. This subset will in effect be the solution: a collection of essential prime products, the total logical sum (OR) of which produces all expected ON (1) outputs of the function. The last and final call is necessary when, even after an attempted reduction of the prime implicant set, there are no essential prime implicants. In this special (and quite frequent) case, the minimization problem for a given expression has several solutions. For example, if a minimum term array for a 5-variable expression is given by T = <0, 1, 2, 3, 5, 10, 11, 14, 15, 16, 17, 19, 22, 24, 26, 27, 28, 30, 31>, then the set of optimum solutions contains all of the following equations:

C'DE + A'B'D'E + ABE' + BD + ACDE' + A'B'C' + B'C'D'
B'C'E + A'B'D'E + ABE' + BD + ACDE' + A'B'C' + B'C'D'
B'C'E + A'B'D'E + ABE' + BD + ACDE' + A'C'D + B'C'D'
C'DE + A'B'D'E + ABE' + BD + ACDE' + A'C'D + B'C'D'

This ability to obtain multiple solutions for a minimization problem opens up certain possibilities in game design, such as prompting the user to find all possible optimal solutions, or ask to find a given number of such, as a function of the difficulty level set in the game.

B. *Quine-McCluskey and Performance*

The main unfortunate drawback with the programmability of Quine-McCluskey is exponential time complexity. The upper bound for the worst-case running time of the algorithm is given by the following:

$$O\left(\frac{3^n}{n}\right) \tag{1}$$

where *n* is the number of unique Boolean variables present in the logical expression [3]. This compares poorly to the running time of the Espresso heuristic logic minimizer [5], upper bounded by:

$$O(n.p^2 + n^2.p) \tag{2}$$

where *n* is the number of unique variables and p is the number of given minimum terms [6]. However, it must be taken into account that for our purposes, given the design on the Karnaugh Map solver, logical expressions with a magnitude of no more than 5 variables will require a correctness check using the algorithm. Hence, the exponential time complexity does not pose a problem with user experience or otherwise. In addition, and following the same logic, there is not reason to use a heuristic logic minimizer, because such an algorithm is designed to handle logic expressions of large magnitudes (i.e. 50 variables or higher), and the end result is a *near-optimal solution*, obtained using a heuristic approach [7]. For our purposes, a near-optimal solution is not the goal; rather, it is to obtain all confidently optimal solutions of a given non-large logical expression, because they are all required for error checking. For a more complex solver implementation primarily concerned with trading accuracy for speed, the Espresso logic minimizer would be the best option.

III. Challenges

A significant challenge in implementation was the branching of the algorithm after the DERIVE-NTH-ORDER routine. It was not immediately obvious that certain permutations of minimum term array T predict multiple confidently optimal solutions, which are all correct solutions, while others permit only one. The implementation of Petrick's method [8] solved this problem, but such a solution may significantly increase the running time as added overhead, for example, if the magnitude is set to 7 or 8 variables, and all optimal solutions must be computed.

Another limitation of the algorithm lies in the cases when certain minimum term combinations producing very large prime implicant charts, yet yielding no essential prime implicants for the function, known as the cyclic covering problem [9]. This problem is usually circumvented using the Petrick's method, but such a solution is unrealistic when the prime implicant chart is too large. In this case, the running time increases so rapidly that it is no longer practical to use the algorithm to obtain an optimal solution for this particular set of minimum terms

We developed a special technique for creating a comparator that would check if two minimum terms, expressed in the binary system, differed in only one bit at an arbitrary position. For a binary representation of a term stored as a string type, it was merely a question of string manipulation. However, when it was necessary to compare unsigned integer values, a different approach was used:

$$x = m_1 \; XOR \; m_2 \qquad c = log_2 x \tag{3}$$

where m_1 and m_2 are minimum terms to be compared. If $c \in \mathbb{N}$ then the two terms differ by one bit.

IV. Prototype

The user experience of the Karnaugh Map solver will give the user freedom to arbitrarily set the challenge with regards to the Karnaugh Map he or she is about to solve. The user interface will prompt the user with a magnitude selection dial (with values ranging from 2 to 5) and a difficulty slider, the value of which will determine the quantified "difficulty" of the minimization problem. This property is defined as follows: suppose that the problem is given as an array T of terms with size *n* containing random (non-repeating) values in the range $[0 \ldots 2^m - 1]$ where m is the magnitude of the expression. Then *n* will be calculated as the floor of (m . (SLIDER_VALUE)) / 1.7), where m is the magnitude and the SLIDER_VALUE is can take value in the range [1…6]. Based on these properties set by the user, a pseudorandom array of minimum terms is generated, which will populate the Karnaugh Map to be solved with ON (1) terms.

Acknowledgment

O.L. and K.L. thank Dr. Maria M. Larrondo-Petrie for her contribution to this small research project, for the resources she provided to render the effort complete, for logistical assistance and academic guidance.

References

[1] Karnaugh, Maurice, "The Map Method for Synthesis of Combinational Logic Circuits". Transactions of the American Institute of Electrical Engineers part I, November 1953.

[2] Lewin, Douglas, "Design of Logic Systems", Van Nostrand (UK), 1985

[3] Banerji, Sourangsu, "Computer Simulation Codes for the Quine-McCluskey Method of Logic Minimization", Department of Electronics & Communication Engineering, RCC-Institute of Information Technology

[4] "Swift." Accessed on March 1, 2016 at https://developer.apple.com/swift/

[5] Eigen, David, "Minimizing Boolean Sum of Products Functions," accessed March 1, 2016 at https://www.techhouse.org/~deigen/older-projects/min-bool-funcs.pdf

[6] Petr Fišer, David Toman, "A Fast SOP Minimizer for Logic Functions Described by Many Product Terms", Czech Technical University in Prague Department of Computer Science and Engineering, 2009.

[7] Roman Lysecky, Frank Vahid, "On-Chip Logic Minimization", Department of Computer Science and Engineering University of California, Riverside, 2003.

[8] Arslan, Nescati and Ahmet Sertbaş, "An Educational Computer Tool for Simplification of Boolean Function's via Petrick Method," *Journal of Electrical & Electronics,* 2(2), 2002.

[9] Noswick, Steven. CSEE E6861y Handout 5: The Quine-McCluskey Method, January 21, 2016. Accessed March 1, 2016 at http://www.cs.columbia.edu/~cs6861/handouts/quine-mccluskey-handout.pdf