**Minimizing Boolean Functions: A Quine-McCluskey Method**

Group members:

Kirollos Ehab Zikry

Mohammad Yahya Hammoudeh

Rawad Batrawi.

On October 31, 2023

Report on Digital Design Project 1

Instructor: Dr. Muhammad Shaalan.

**Abstract**

The code uses the Quine-McCluskey algorithm to minimize Boolean functions. It accepts as input a Boolean function in the Product of Sums (PoS) or Sum of Products (SoP) form, and outputs the minimized Boolean expression. In addition, the code creates a truth table, finds prime implicants, and divides them into essential and non-essential categories. It also creates HTML files that can be used to visualize logic circuits and Karnaugh Maps.

**Introduction**

In many areas of computer science and engineering, especially in the design and optimization of digital circuits, boolean function minimization is a crucial task. The main goal is to reduce a given Boolean function to its most basic form while maintaining its original logic. Boolean functions can be made simpler to create more effective digital circuits. By requiring fewer gates and connections, these circuits can operate faster, consume less power, and require less money to produce.

The Quine-McCluskey algorithm, which bears the names of Edward J. McCluskey and Willard V. Quine, is among the best techniques for minimizing Boolean functions. Since the algorithm is exact, the best answer will always be found. But for functions with a lot of variables, it can be computationally costly, which makes it less appropriate for really big problems.

This report describes a program that uses the Quine-McCluskey algorithm to minimize Boolean functions. The program, which is written in C++, finds the minimized Boolean expression and also creates logic circuits and Karnaugh Maps as HTML files. This gives professionals and students alike a comprehensive tool for understanding and using Boolean minimization in practical situations.

The report will go into detail about the program's design, going over the data structures and algorithms that were employed. It will also list all current issues and restrictions, describe the testing procedures used, and offer comprehensive guidelines for compiling and executing the program.

**Program Structure**

2.1 Architectural Overview

Both extensibility and modularity were considered in the program's design. It is divided into multiple main parts, each in charge of a particular step in the minimization of the Boolean function. The HTML/JavaScript Code Generator, the Quine-McCluskey Algorithm Engine, the Karnaugh Map Generator, and the Input Parser are some of these components.

2.2 Input Parser

The user-provided Boolean function must be read and validated by the input parser. It transforms the function into a format that the Quine-McCluskey Algorithm Engine can understand. The parser verifies that the function follows the permitted Boolean expressions and looks for syntax errors.

2.3 Quine-McCluskey Algorithm Engine

This is the program's central component, where the real minimization happens. The engine utilizes the Quine-McCluskey algorithm on the parsed Boolean function to determine the simplified expression. It efficiently stores and manages minterms, prime implicants, and essential prime implicants using a variety of data structures, including vectors and hash maps.

2.4  Karnaugh Map Generator

Following minimization, a Karnaugh Map is produced by the Karnaugh Map Generator using the simplified function. This graphic representation confirms the outcome and helps to understand how the minimization was accomplished.

2.5 JavaScript and HTML Code Generator

This part is in charge of producing JavaScript and HTML code that can render the logic circuit diagram and the Karnaugh map. This feature serves both professional and educational purposes by making it easier for users to visualize the minimized function and its corresponding circuit.

2.6  Code Organization

For easier reading and maintenance, the code is divided into several files and classes. Logically related functions are grouped together, and related functionalities are encapsulated in classes. For instance, the QuineMcCluskey class manages the algorithmic aspects of the Karnaugh Map generation process, while the KarnaughMap class handles all other related tasks.

2.7 Efficiency Considerations

In order to efficiently handle Boolean functions with a reasonable number of variables, the program is optimized for speed and memory usage. However, functions with a lot of variables may cause the program to perform less well due to the Quine-McCluskey algorithm's inherent complexity.

2.8 Variable Naming and Type Safety

Whenever possible, type-safe constructs are employed to avoid type-related errors. The use of descriptive variable names improves readability and maintainability by making the code self-explanatory.

2.9 Future Improvements

The current architecture makes extensibility simple. Future iterations may incorporate a graphical user interface, support other minimization algorithms, and don't-care conditions, among other features.

The program aims to be a dependable, effective, and user-friendly tool for Boolean function minimization by following best practices in software engineering.

**Issues and Restrictions**

3.1 Scalability

The program's scalability is among its most important drawbacks. The central component of the minimization procedure, the Quine-McCluskey algorithm, has exponential time complexity. The time needed for minimization grows exponentially with the number of variables in the Boolean function. Because of this, the program is less appropriate for functions involving a lot of variables.

3.2 Error Handling

Although the program has some basic error handling—particularly for syntax errors in the Boolean function used as input—it does not have extensive mechanisms for handling more complex problems. It cannot handle edge cases such as self-contradictory functions or circular dependencies between variables, for instance.

3.3 Code Duplication

There are some code blocks in the current version of the program that are repeated with only slight modifications. This increases the size of the codebase and creates possible points of failure because every modification must be applied to every section.

3.4 Use of Global Variables

For convenience, the program uses global variables in certain places. While this does make some aspects of writing code simpler, in the long run, it makes the program more difficult to debug and maintain.

3.5 Inconsistencies in Code Style

While the style of the code is generally consistent, there are a few small deviations, like using both cout and std::cout. While not compromising the functionality of the program, such inconsistencies can make the code more difficult to read and update.

3.6 Lack of User Interface

Right now, the program runs via a command-line interface, which might not be the most intuitive method for all users. Although it hasn't been used yet, a graphical user interface would increase the program's accessibility.

3.7 Restricted Formats of Output

As of right now, the program can produce logic circuit diagrams and Karnaugh Maps in HTML and JavaScript code. It does not, however, support other formats, such as JSON, XML, or plain text, which might be helpful in other contexts.

3.8 Type Safety

The program still uses some less type-safe components, such as representing minterms with character vectors, despite efforts to employ type-safe constructs. Future development may experience type-related errors as a result of this.

3.9 Testing Limitations

The software has been put through some basic testing to make sure it works, but neither formal verification techniques nor extensive, rigorous testing have been applied to it. This leaves open the chance of unidentified flaws or restrictions.

3.10 Documentation

The extant documentation, encompassing external guides and inline comments, is not comprehensive. This could present difficulties for users attempting to comprehend the inner workings of the program as well as for future development.

The development team wants to improve the program's robustness, efficiency, and user-friendliness in the future by addressing these issues and limitations.

**Program Testing**

4.1 Unit Testing

Each function was tested individually with a variety of inputs.

4.2 Integration Testing

The entire program was tested with complex Boolean functions to ensure that all components work in harmony.

4.3 Test Cases

Test cases included both synthetic and real-world examples.

**Build and Run Instructions**

Clone the Repository: git clone https://github.com/example

Navigate to Directory: cd project\_directory

Compile: g++ main.cpp -o main

Run: ./main

**In summary**

All things considered, the application functions as a strong tool for Boolean function minimization and provides a number of features, such as the ability to generate Karnaugh Maps and render logic circuit diagrams. Based on the Quine-McCluskey algorithm, it offers a precise and effective method for minimizing Boolean functions in educational and professional settings. However, as the earlier sections have shown, there are some shortcomings in the program and room for development.

Task allocation:

Task 1: Kirollos

Task 2: Kirollos

Task 3: Rawad (The first draft of the code was written by Kirollos)

Task 4: Rawad

Task 5: Rawad

Task 6: Mohammad Yahya

Task 7: Mohammad Yahya, Kirollos Zikry (the translation to HTML was by Kirollos, the older code is commented out was written by Yahya)

Task 8: Mohammad Yahya (Kirollos fixed an issue with the HTML and the NOT gate)

Report written by Mohammad Yahya Hammoudeh