QUINIFY

To make your life easier with Quine McCluskey Logic Minimization algorithm

TEAM

Abdelrahman Abo Ismaiel	Habeba Saad	Salma El-Hawary

CSCE2301

Supervisor Dr. Dina Mahmoud

Spring 2025

Table of Contents

Project description

The objective of the project was to minimize the logic expression using the Quine McCluskey method. The desired program was designed using C++ code, especially using the VS Code application. First, the data inputs are stored in text files to take the inputs from and generate the minimization process. Each text file has three lines, the first is for the number of variables, the second is for the minterms (m) or maxterms (M), and the third is for the don't cares (d). After reading the inputs from the file, the first process is to check whether the inputs are correct or not; there are different edge cases that are considered, which are checking whether the minterms / maxterms / don't cares' numbers are true or not. In other words, an n-variable has 2^n cells, and there will be an error for any number in the input that is out of range or if there is duplication between the number in the minterm/maxterm and in the don't care at the same time. Furthermore, the code will loop over all the input terms to make sure that all of them are minterms or maxterms, not a combination, otherwise it will generate an error.

After checking that the inputs are correct, the minimization process will be generated. There is a function for generating all prime implicants, and from it, the essential prime implicants (EPI) are generated. There is a function for detecting the remaining prime implicants after removing the EPIs. After generating the EPIs and removing them from the remaining PIs vector, the remaining PIs are detected by the function of processing the remaining PIs. Then, there is another function for simplifying the Boolean expression using the domination rules. The domination rule is to loop around the columns in the coverage chart to detect the dominating and dominated columns. As is known from the domination rules, the dominating columns are deleted after being detected. The process is the same for the rows, but with deleting the dominated rows. After looping and simplifying the expression, the remaining PIs are being simplified as taking the best and unique PI minterms after domination and using them in the final expression, besides having the EPI from the second stage. The remaining nonunique PIs are being selected from the Petrick method and expressing each of them in each line for the minimization process. In other words, each minimized expression consists of the EPIs + the unique PIs from the domination process + a PI from the Petrick method, which will be expressed as more than one simplified logic minimization solution. Moreover, there is a for loop to check whether the solution is correct or not by comparing the minimized expressions' answer with the output files and return true with the correct output file name if the two solutions match. Finally, generating the Verilog module for the function was performed. It generates the module for the minimized expression of the function. The algorithm has a time complexity of $O(2^n)$.

To build the program

To get started with the project, the first step is to clone the project repository to your local machine. You can do this by using the following command in your terminal:

```
git clone https://github.com/CSCE2301/dd1-s25-project1-quinify.git
```

This command will create a local copy of the project on your system, which you can modify and work with as needed. Once the cloning process is complete, navigate to the project directory.

This is important because the subsequent steps will require you to be within the correct project folder. To do this, use the command:

```
cd .\src\project
```

After navigating to the project directory, you'll need to ensure that the correct test case is included in the main function of the project. This step is crucial as it ensures that you are running the correct set of tests for your work, helping to verify that everything functions as expected. Once you've confirmed this, you can move on to the next steps.

Now, open your terminal window if you haven't done so already. You'll need it to compile the C++ files necessary for the project. The next step is to compile the project's source code into an executable file. To do this, you'll use the g++ compiler with the following command:

```
g++ -o initial.exe initial.cpp Term.cpp Expression.cpp Table.cpp
```

This command will compile four C++ files—`initial.cpp`, `Term.cpp`, `Expression.cpp`, and `Table.cpp`—and link them together into a single executable file named `initial.exe`. It's important to ensure all the cpp files are included in the compilation process to avoid any missing components in the final executable.

After the compilation is complete, you can run the solver. This is where the actual computation or problem-solving takes place, depending on the project's purpose. To run the solver, simply use the following command in your terminal:

```
.\initial.exe
```

By running this command, the compiled executable will start, and the program will begin its process, using the test cases you've prepared to perform the necessary tasks. If everything has been set up correctly, this should execute the program successfully and provide the expected output.

Data Structures Used & Algorithms

There are different classes used for making the logic seem easier while handling the code, besides using sets, vectors, maps, and other libraries.

Expression class

- Parses the Boolean function for the input and stores the variables from the input file.
- Does the suitable conversions if the inputs to minterms, if they are maxterms, and handles the errors.

Term class

It represents the individual terms in the Boolean expression through:

- Checking and combining terms in the columns to generate the prime implicants.
- Convert the binary expression into algebraic form.

Table class

This is the core class for implementing the algorithm.

- Generating the prime implicants by grouping the minterms and don’t cares and printing them.
- Generating the EPIs and storing them.
- Implementing the domination rules by generating the column Androw dominance to get the best PIs from the dominations after reducing the chart and remove the EPIs from it.
- Performing Petrick method for the remaining PIs.

bitset<20> To convert decimal values to binary strings.

fstream To handle the file input operations, starting from reading the file.

stringstream To parse comma-separated values from input strings.

Other algorithms

Using maps, vectors, and sets to store and retrieve variables. Each of them is used according to their proper function in the code.

Testing

The primary objective of testing was to ensure the program's correctness, reliability, and performance. The tests were designed to cover:

- Functionality: verifying that all components behave as expected under normal and edge case scenarios.
- Integration testing: testing the algorithm by comparing the values with the manually process.
- Edge cases: testing with empty term lists, with don’t cares only, with maxterms or minterms only, and with wrong inputs to test the handled error messages for that.
- Validation Methods: verifying that all original minterms are covered by the prime implicants and no prime implicant can be further simplified.

Challenges

- There were errors in handling the domination process since the for loops in the row and column dominance were not correctly handled.
    - The error was in the dominance since it was performing in the deletion of the dominated columns, not the dominating ones.

- After handling this error, there was an error in the row dominance since the function was not handled correctly.
- This challenge led to not removing the remaining minterms from the dominated rows correctly.
- These errors were fixed after fixing the function of row dominance and the generation of dominance rules.
- After correctly doing the domination, a problem of not generating all the best PIs after domination was figured out.
- The method was not generating the solutions for the best PIs in a proper way since only one PI was selected as the only best solution. The issue was fixed after determining an error in a for loop for saving the possible best PIs and removing them from the remaining PIs vector.
- After solving the issue, a problem in Petrick's method was figured.
- The issue was figured out to be in the "expandToPetricksSOP" function as extracting the possibilities of minimized expressions was not handled correctly.
- After correctly implementing the "expandToPetricksSOP" function, Petrick's method correctly worked.

Contributions

- All the team members contributed equally and collaborated together in building the project. In addition, they constructed meetings (online / offline) to follow what each of the members has done, besides making decisions about the remaining parts of the project and the ways to solve them. For more details about writing the code:
  - Abdelrahman: constructing the classes, generating PIs, building up 5 test files, Verilog, and readme.txt file.
  - Salma: Generating EPIs and helping in the domination process and the Petrick method, and building up the other 5 test files.
  - Habeba: Performing the row and column dominance rules, Petrick method, comparing the outputs, and reporting.

Acknowledgment of AI Usage

ChatGPT was used in helping to help make up the general ideas. Also, it was sometimes used to check whether there was a logical issue with a specific function when an unknown issue appeared, but it did not fix it. In addition, it was used to generate the minterms for the edge cases and to perform random inputs for testing.