CSCE 2301, 02 Project I Report Spring 2023

Quine-McCluskey Logic Minimization

Mohamed Abbas 900211252

Mohammad Alashkar 900214276

Omar Bahgat 900211747

CSCE 2301, 02

Department of Computer Science and Engineering, AUC

Table of Contents

Github Link: https://github.com/MohamedRagabAbbas/Digital-Design-Project-Final-Version-1

Program Design

I. Data Structures used:

A. Struct: It is a data structure used to represent all kinds of implicants such as the prime and the essential prime implicants. It also contains a vector, and this vector contains the minterms that this implicant covers.

B. Stack : In our project, we used the stack data structure to evaluate user-entered expression. Due to its ability to effectively manage parenthesis and prioritize operations, we were able to handle complicated and nested expressions.

C. Vectors: They played a key role in our project's data processing. Large data sets including user inputs, truth tables, PIs, EPIs, coverage charts, and any implicants that are not covered by essential prime implicants were stored and processed using vectors. We were able to effectively carry out computations and provide visuals that assisted in our program by utilizing the vector data format. Vectors also gave us the ability to simply resize and remove any element from our data processing, giving us important and extendable functionalities to use.

D. Maps: The map data structure is heavily utilized in our project to manage user-entered expressions. It aids in figuring out the literal's truth table index and value. Also, the Quine-McCluskey (QM) method is employed to construct the minimized function expression when using the map data structure, which first involves determining the dominating columns and dominated rows. The map data structure, in general, is essential for effectively arranging and updating the data necessary for these processes. Its usage enables the algorithms to operate more rapidly, reducing calculation time and enhancing output precision.

E. Set: One of our project's key components is the set data structure. Its main duty is to clear the data of information (implicants) that is duplicated and redundant. It is highly helpful when comparing functions since several implicants may combine to form other implicants that have previously been produced in those functions. The set data structure, which ensures that each implicant is only saved once, reduces the quantity of data. This method expedites implicant construction while cutting down on computation time as the ordered_set works in O(logn) time. As a result, the project's algorithms are much efficient.

II. Main Functions

A. Void generateTruthTable (num)

The "generateTruthTable" function generates a truth table with an additional column for the output and the same number of columns as the variables in the logical expression. The number of rows is calculated using the power of two for the number of variables. The function alters the ones and zeros in each column of the truth table in a certain order. The first column has half zeros and half ones, whereas the second column has a pattern of one-quarter zeros, one-quarter ones, one-quarter zeros, and one-quarter ones. More specifically, for each additional column, the number of succeeding zeros or ones in each block is reduced by half. The pattern will be 1/8 zeros, 1/8 ones, 1/8 zeros, and so on in the third column. The generated truth table is returned as a 2D vector of booleans.

B. These functions are designed to validate user input(must be in sop form), permit expression entry without the use of products, replace literals with truth table values, and evaluate expressions while taking priority concerns and parenthetical notation into account.

1- Function: Validation

we developed this function to verifies user input to make sure the expression is in the form of a sum of products. If the user-entered expression is a sum of products, this method returns true; otherwise, it returns false.

2- Function: Expression Conversion

The user can input expressions without using a product by utilizing the second function. For instance, this method will return "a*b" if the user inputs "ab." The expression conversion function makes it simpler for the user to enter expressions into the application and facilitates this procedure.

3- Function: Replace

The "replace" function is designed to take an expression and a truth table as inputs. It replaces each literal with its corresponding value in the truth table, considering the inversion of the literal. This ensures that the program interprets the expression correctly, taking into account the values assigned to each literal in the truth table.

4- Function: Evaluate

The "evaluate" function takes the modified expression as input, which is the expression where the literals have been replaced with their corresponding values from the truth table. It evaluates the expression while respecting priorities and handling parentheses. The function returns either 0 or 1 based on the output. This function is crucial as it calculates the output based on the input expression.

C. void SoP;

This function receives the truth table as a 2-D vector and generates the terms of the SOP and returns them as a vector of strings (which is passed by reference as an empty vector). In addition, it receives all the terms as a string called vnames. It loops over the last column of the truth table

and see where the function equal to 1, and then it loops over this row (where the last column equal to one), and it adds to an empty string char (it get this char from the vnames string) if it find one and char' if it find 0. It repeats this process and pushes these strings to the vector of strings that contains the the SOP terms.

D. void PoS;

It is basically very similar to the SoP function. It receives the truth table as a 2-D vector of boolean values, and all used characters as a string. It loops over the truth table similar to what SoP do but it tries to find the rows that contain 0 at the last column instead of trying to find the rows with 1. It also returns the PoS terms as a vector of strings that we pass to the function by reference (the vector should be empty when it is passed to the function).

E. vector<vector<Implicant>> compare

The function compare takes a 2D vector of Implicant objects and an integer combine as input. It iterates through the vector to combine the adjacent Implicant objects that differ in only one term. The function creates a new Implicant object with the differing term as a dash and merges the minterms of the two combined Implicants. It then adds the new Implicant object to a new 2D vector while keeping track of already added combinations using a set to avoid having the same prime implicant twice which would result in redundant comparisons. Finally, the function returns the new 2D vector of combined Implicants. The integer combine is passed by value and not by reference and is updated with the number of combinations made. The looping on the compare function terminates when the integer combine is equal to zero which means that no implicants have been combined together in the last iteration.

F. vector<string> epi;

1. The function takes two parameters:
    a) A vector of Implicant objects (prim) representing the prime implicants of the function. Each Implicant object contains a string representation of a prime implicant and a vector of integers representing the minterms covered by that implicant.
    b) A vector of integers (m) representing all the minterms . The function returns a vector of strings (EPI) representing the essential prime implicants of the function.

2. The function works by first constructing a coverage chart using the CoverageChart function. The coverage chart is a table that lists all of the minterms and the prime implicants that cover them. The function then iterates over each column of the table, looking for columns that have only one "x" in them. If a column has only one "x", then the corresponding prime implicant is an essential prime implicant and is added to the EPI1 set. For each essential prime implicant found, the function also identifies the minterms that it covers and adds them to the deletedM vector (the minterms that are covered by the essential prime implicants). After all essential prime implicants have been found, the function removes the minterms covered by those implicants from the m vector, and removes the corresponding prime implicants from the prim vector. Finally, the function adds the elements of the EPI1 set to the EPI vector and returns it. Overall, the epi function applies the Quine-McCluskey algorithm to find the essential prime implicants of a boolean function, by using a coverage chart to identify columns that have only one "x" and then removing the

corresponding implicants and minterms from consideration. (In short it returns the essential primes in addition to the minterms that are not covered by essential primes, and it removes the essential primes from the prime implicants vector.)

G. Minimizing Function:

The minimizing function is responsible for finding the minimum implicants that cover the chart, using a series of steps. Firstly, it takes in the implicants that are not covered by the essential prime implicants. Then, it utilizes two helper functions called "Rows" and "Columns" to determine the dominating columns and dominated rows. Each helper function contains a map that counts the number of "x" in each row or column. Next, the minimizing function identifies the most dominated row and stores the corresponding implicant in a vector. It then deletes this row and the columns that it covers. This process is repeated several times until the number of rows or columns is less than two. Finally, the function returns a vector of minimum implicants that cover the chart. These minimum implicants can be merged with the essential prime implicants to obtain a complete set of implicants.

Validating Input

1) In the case of a normal expression, we check whether the expression is written in SoP form or not and whether any invalid characters are inputted or not.

2) In the case of minterns and/or don’t cares, we check whether they are non-negative integers or not.

3) When choosing any printing option, we check whether a valid number is written or not.

How to Use the Program

H. First you have to determine the the type of the input :

Please enter your input type

1. Expression
2. Minterms
3. Minterms and Don't Care Terms
0. Exit

Enter Your Choice : Invalid input type. Exiting program

I. After that the program will ask you to enter the input according to the type that you choose. For example, if you choose to enter the input as minterms the program will ask you to enter the minterms (the numbers where there is 1 in the table of truth of the function.

III. Now, the program will ask you what do you want to do with this input and it will give you the following list:

A. 1. Truth table
2. Canonical SOP
3. Canonical POS
4. All PIs
5. Print Coverage Chart
6. All EPIs
7. All implicants that are not covered by EPIs
8. Minimized Function
9. Start from beginning
0. Exit

B. You can press any number from the ones written above and it will print it. The program will repeat these questions infinitely until you choose 0 which will exit the program or 9 which will cause you to enter a new input, whether a normal expression or an expression in the form of minterms and/or don’t cares.

IV. Each Individual Contribution

A. We worked together on all the functions and we decided what the logic that we wanted to use together, then each one of us wrote a part of what we discussed and after each one finished writing a function we debugged it together. And here is where each one on us worked the most

1. Mohammad Alashkar: Most of his contribution was in finding the SoP and the PoS and finding the essential primes and the minterms that are not covered in the essential primes. In addition he helped Omar Bahgat with reading and validating the input.

2. Mohamed Abbas: most of his work focused on evaluating the output of the truth table that Bahgat has created and on the minimization function and he worked with Bahgat on finding the primes implicants. Also, he worked on the function of handling the input (the one that includes a switch to let the user decide which function he/she wants to use).

3. Omar Bahgat : Most of his work focused on creating the truth table then reading and validating the input. In addition he worked with Abbas to find the prime implicants and did part of the reducing PI table to remove essential prime implicants and the minterms they cover from the coverage chart.

B. We did the rest of the work together (the report and the test cases).