

School of Sciences and Engineering

**CSCE 2301, Project I Report Fall 2023**

**Prof. Mohamed Shaalan**

**Quine-McCluskey Logic Minimization**

Mohamed Sabry

Hamza Abouheiba

**Algorithm Explanation:**

The Quine-McCluskey algorithm is a method for simplifying Boolean algebra expressions. It takes a set of minterms (binary representations of specific input combinations) as input and systematically reduces them to a minimal set of prime implicants, which are essential components for the expression. The algorithm does this by iteratively comparing and combining minterms with similar patterns and removing redundant terms. It aims to find the simplest possible representation of a Boolean function, helping to reduce the number of logic gates required for implementation.

**Program Design:**

1. **Data Structures Used:**

* **Vectors:** They played a key role in our project's data processing as large data sets 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 resize and remove any element from our data processing, giving us important functionalities to use.
* **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. The map data structure, in general, is essential for effectively arranging and updating the data necessary for these processes as it works quickly in O(logn) time.
* **Sets:** One of our project's key components is the set data structure. It was highly used with the map data structure to store large amounts of information and was widely used in the helper functions.

1. **Helper Functions:** Several functions are defined to handle different tasks, such as printing containers (‘OnesMapPrint’, ‘SVectorPrint’, ‘SSetPrint’, ‘SSetPrint2’, ‘CSetPrint’, ‘ISetPrint’), grouping implicants by the number of ones in their binary representation (‘GroupByOnesIMP’, ‘GroupByOnes’), comparing and finding prime implicants (‘ImplicantGroupComparison’), checking the number of variables in a PoS or SoP expression (‘PoSValidation’, ‘SoPValidation2’), converting SoP expressions to binary strings (‘SoPtoBinaryString, SoPtoBinaryString2’), and converting PoS expressions to binary strings (‘PoStoBinaryString, PoStoBinaryString2’).
2. **Truth Table Functions:** ‘TTableBuild’ and ‘TTableBuild2’ are used to construct truth tables based on the input SoP or PoS expressions, respectively. They are based on the number of variables in the expression.
3. **Expression Cleanup Functions:** ‘SoPCleanUp’ and ‘PoSCleanUp’ are used to process SoP and PoS expressions, respectively. They extract minterms or maxterms and create truth tables for further processing.
4. **Implicant Comparison Functions:** ‘ImplicantGroupComparison’ is used to compare the different groups of implicants to groups beneath them in the ascending order of ones.
5. **Validation Functions:** ‘SoPValidation1’, ‘SoPValidation2’, and ‘PoSValidation’ are used to validate the inputted PoS or SoP for a number of variables more than the max or incorrect variables.

**How to build the program:**

* Take the cpp file in the DIGITALDESIGNPROJONE file and open it any C++ compiler.

**How our program works:**

1. Takes input as POS or SOP.
2. Does validation on the format. "abc + ad'c" or "(a + b + c)(c + d + e')".
3. SoPValidation1() -Takes the SoP and checks for brackets.
4. SoPValidation2() and PoSValidation() Validate that the number of variables is not more than 10.
5. SoPCleanUp() or PoSCleanUp() take the input string and divide it into parts using the functions and stores it in vectors.
6. We then take the vectors and turn the strings into binary strings. SoPtoBinaryString() or PoStoBinaryString().
7. Take the binary string to the TTableBuild() or TTableBuild2() functions to build the truth table.
8. After the table is built, the final column is filled according to the function provided. FTTColFillSOP() or FTTColFillPOS()
9. The PrintMinMaxterms() function is then used to derive the canonical min and max terms in boolean form.
10. It then calls the SoPtoBinaryString2() or PoStoBinaryString2() to turn the boolean terms into their binary string form and then into their binary string form.
11. The binary strings are then given to the GroupByOnes() to group them according to the number of ones.
12. The ImplicantGroupComparison() function derives the prime implicants and the essential prime implicants by comparing each group with the ones underneath it. The process is repeated until all essential prime implicants are derived it uses the GroupByOnesIMP() function as well.

**Problems faced:**

* One of the problems that faced us in the beginning was deciding how to deal with each character in the string individually. So, we decided to divide the string into vectors and turn the strings into binary strings. The general problem we faced was that we knew how to solve the problems physically, but not through coding.
* The validation is limited to checking if there are brackets in the SoP or missing brackets in the PoS. It also checks if there are less than 10 variables. We also didn’t have time to check if the PoS parts work or not, so we derive the implicants using SoP only. Additionally, the complexity is quite high as there are 5 for loops in one function.

**Contributions:**

**Mohamed Sabry:**

1. **Worked on the main and provided the 10 test cases.**
2. **Created the following functions: ‘**SoPValidation1’ , ‘SoPCleanUp’, ‘PoSCleanUp’, ‘SoPValidation2’ , ‘SoPtoBinaryString’, ‘TTableBuild’, ‘FTTColFillSOP’ , ‘PrintMinMaxterms’ , ‘SoPtoBinaryString2’ , ‘GroupByOnes’ , ‘GroupByOnesIMP’ , and ‘ImplicantGroupComparison’.
3. **Created all of the helper functions.**
4. **Worked on the Readme.md**
5. **Edited the Report.**

**Hamza AbuHeiba:**

1. **Created the Report.**
2. **Created/ Edited the following functions: ‘**TTableBuild2’, ‘FTTColFillPOS’, ‘PrintCanonicalForms’ , ‘PoStoBinaryString2’ , and ‘PoStoBinaryString’.