QUINE MCCLUSKEY SIMULATOR

TECHNICAL MANUAL

Authored by:

Barcellano, John Derick

Coscolluela, Jan Federico

TABLE OF CONTENT

General Information	3
Project Overview	3
Organization of the Manual	3
Code Structure	4
Discussion	6
Limitations	17

Quine-McCluskey Simulator

1.0 Organization of the Manual

The technical manual is divided into five (5) sections: General Information, Project Overview, Code Structure, Discussion, and Limitations.

General Information section explains the project and its general purpose.

Project Overview section provides a brief summary of the project as well as the language of choice used in the implementation.

Code Structure section talks about the number of classes involved, as well as essential things to consider with respect to external jars within the source code.

Discussion section explains each class and method that makes the simulator possible and provides an overview of how it behaves.

Limitations section tells restrictions of the program and things to consider.

1.1 General Information

This technical manual is intended to go through all the classes used in the implementation of the simulator. This includes the discussion of methods, logic, and FXML employed throughout the project.

1.2 Project Overview

The Quine-McCluskey Simulator implements such Boolean simplification method using Java and returns the corresponding output after scanning the input from user. This is made possible through multiple Java classes and usage of FXML to display a GUI for a user-friendly interface.

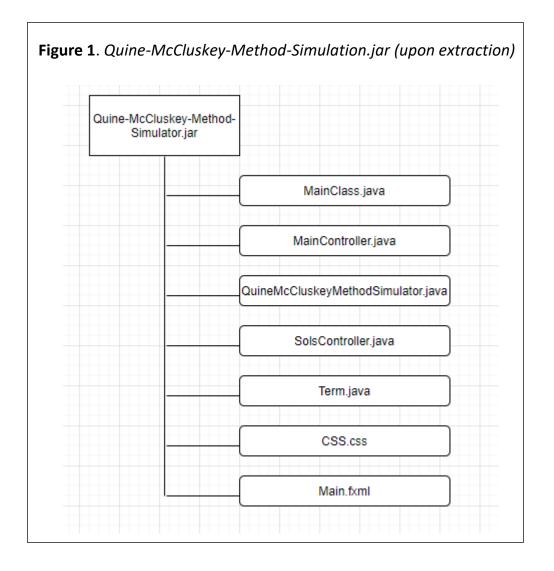
1.3 Code Structure

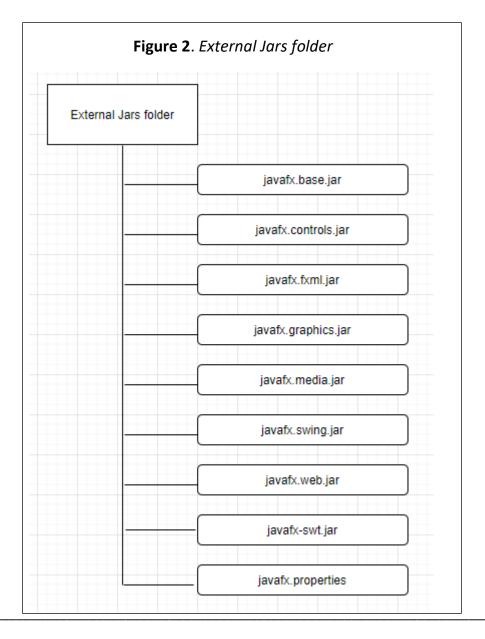


This section discusses the structure of the source code submitted. Throughout this part, each component will be discussed separately.

The code submission comprises the following (aside from the manuals):

- Quine-McCluskey-Method-Simulation.jar
- External Jars
- ➤ MP1_Simulator (source code)





The MP1_Simulator has exactly the same structure as that of Figure 1. This is because they are essentially identical, and the jar file from Figure 1 is simply a compressed version of MP1_Simulator which will be used as reference in the codes. The only observable difference is that the components of MP1_Simulator are within a package, namely \quine\mccluskey\method\simulator.

1.4 Discussion



This section would explain, one by one, the classes with their methods that make the simulator fully functional.

I. CODE IMPLEMENTATION

- a) MainClass.java
 - Initialization

```
private ArrayList<Term> terms = new ArrayList<Term>();
private ArrayList<ArrayList<Term> > termsGroups = new ArrayList<ArrayList<Term>>();
private ArrayList<ArrayList<ArrayList<Term>>> newGroup = new ArrayList<ArrayList<ArrayList<Term>>>();
private ArrayList<Integer> minterms = new ArrayList<Integer>();
private int maxLength = -100;
  ivate ArrayList<Term> finalPrimeTerms = new ArrayList<Term>();
private ArrayList<Term> essentials = new ArrayList<Term>();
   ivate StringBuilder outPut = new StringBuilder();
int [][] tempChart;
```

terms – array for every term input, either minterm or don't care termsGroups – group of term based on number of 1's **newGroup** – successive new groups based on number of 1's minterms – array of integers corresponding to minterms maxLength – highest count of digits among all input's binary values **finalPrimeTerms** – list of terms categorized as final prime implicants essentials – grouped minterms that satisfy only one difference of digits **restPrimes** – prime implicants left after choosing the essential ones outPut – final String output that contains the answer and solution tempChart – 2d chart that shows which term pairs are selected

Tabulation and Grouping

String toBinary (int num)

- Returns a binary representation of passed integer

void addMinterm (int value)

- adds an input in minterm field to the array terms

void addDontCare (int value)

- adds an input in don't cares field to the array terms

void getMaximumTerm ()

 counts the number of digits of every input's binary value and selects the maximum count and sets it to maxLength

void modifyTerms()

- left pads 0's in case there are missing digits at left side

void groupingTerms

 groups all term entries based on number of 1's and add them to termsGroups

int countOnes (String t)

increments int variable count based on number of 1's in the passed binary value String t

void callQuine()

invokes methods groupingTerms and RecursiveQuine at a specific level

void RecursiveQuine (int level)

 goes through level of comparison of differences and checks if two binary numbers only have one (1) different digit, then pairs such number for further comparison with other potential pairs/group

boolean canBeReducedTogether (String a, String b)

simulates the comparison of two binary numbers and returns
 true if only one different digit is observed

String getReducedExpression (String a, String b)

replaces the lone different digit between two binary values
 with a dash (-)

void reduceThem (int level, int i, int j, int k)

 invokes the getReducedExpression method to group binary numbers with only one different digit, and replaces such digit with a dash (-)

String getReducedExpression (String a, String b)

replaces the lone different digit between two binary values
 with a dash (-)

boolean isExist (ArrayList<Term> arr, String expr)

 prevents duplication while adding a term to new groups by checking if a term is already added in the new array of groups

void printIt()

 appends the output (including list of all entry of minterm and/or don't cares, if any) to the String output and shows the grouping of binary conversion of these terms based on number of 1's, per level of comparison

void printPrime()

- appends the essential terms to String *outPut* to be printed

void printSols()

appends the complete list of terms, and shows all possible
 options of answers based on the prime implicants

String printFormattedSols()

Finally displays the formatted solution, that is, in the form of
 F = AB', etc

String toAlpha (String s)

replaces digit 0 with an apostrophe (') denoting a prime, and
 does no appending if digit is 1

String finalString()

Typecasts outPut (which contains the entire solution) then
 prints it out

ArrayList<Term> processChart (int[][] primeChart)

returns all prime implicants from the passed 2d primeChart

ArrayList<String> processMultiplication (ArrayList<String> temp)

 Returns an array of String which denote all terms that are possible to consider for alternative answers (employs Petrick's method)

ArrayList<String> checkPossible (boolean[termsDeleted],

boolean[] primeDeleted, int[][] primeChart)

 Employs the Petrick's method which looks for other alternative solution by considering other prime implicants in the *primeChart*

void getPrimeImplicants()

- adds prime implicants to the array of terms *finalPrimeTerms*

void createChart()

- fills in the initialized *tempChart* with 2d table that shows 0's and 1's (i.e., the pair (2, 6) shows 1 on both 2 and 6 meaning they are checked in the chart, while all other terms are not)

void printChart()

 prints the *primeChart* that simulates the last step for Quine-McCluskey method, which is determining the final terms to include in the answer through a table

b) MainController.java

void addSol(String s)

- clears the text area for solution in the GUI and sets its content to the passed parameter String s

void mintermsListAdd()

 adds the minterms to the list and shows it to the minterm content section of GUI, else shows warning of no input is detected at all

void dontCaresListAdd()

 adds the don't cares (if any) to the list and shows it to the don't'cares content section of GUI, else shows warning of no input is detected at all

void applyButton()

- finally pushes all entry to the list of terms within MainClass.java and invokes all necessary methods as shown in the code

void answerButton()

shows the final answer/s (with no solution) to the solution
 section of GUI by invoking addSol method from
 SolsController.java

void stepsButton()

shows the entire solution to the solution section of the GUI
 by invoking addSol method from SolsController.java

void clearButton()

- deletes all entry and showed solutions

void exitButton()

- closes the GUI and ends the session itself

void saveFile()

 saves the solutions steps in *Output.txt* and downloads it to the user's workspace.

c) QuineMcCluskeyMethodSimulator.java

void start (Stage primaryStage)

 calls the Scene (essentially the GUI) and sets it to the stage in preparation for GUI display

void main (String[] args)

- launches the stage and displays the GUI

d) SolsController.java

void addSol (String s)

- sets the solution section of GUI to contain the solution itself

void initialize (URL url, ResourceBundle rb)

- built-in requirement to launch the controller

e) Term.java

Term (String value, boolean isDontCare, int num)

- Contructor that sets the value of the three passed parameters to their corresponding initialized variables in the class
- Adds int num to the group of terms where it belongs based on number of 1

Term (String value, boolean isDontCare)

 Constructor that sets the value of the two passed parameters to their corresponding initialized variables in the class

String getValue ()

Getter of value

void setValue (String value)

sets the value of a term to the value of the passed parameter
 String

void setDeleted (boolean deleted)

accepts boolean parameter *deleted* that deletes the term if
 such passed parameter is *true*, and otherwise if false

boolean isDontCare ()

- returns either true or false if a term is a don't care

boolean isDeleted ()

- returns either true or false if a term is deleted or not

ArrayList<Integer> getNumbersItCovers ()

 Returns an array of integers corresponding to terms with the same number of 1's, hence belong to the same group

void addNumbersItCovers (ArrayList<Integer> newCovered)

- adds the array of integers that belong to the same group (same number of 1's) to the array of integers numbersItCovers

void setEssential ()

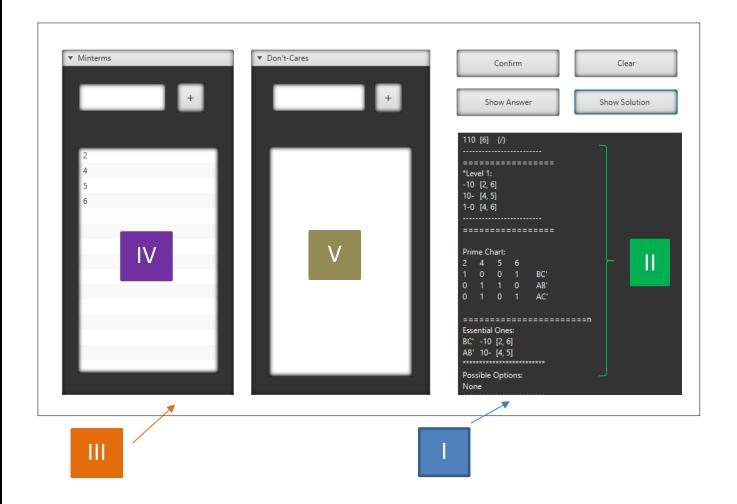
- declares a term (or specifically a prime implicant) as essential

II. GUI IMPLEMENTATION

a) CSS.css

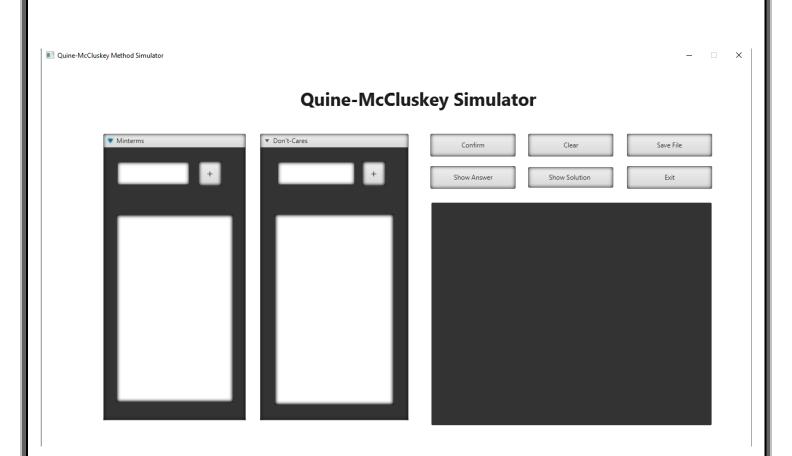
- provides the general styling of the interface, i.e., color and content border

```
1
2 #solutionsArea{
3     -fx-border-radius: 6px;
4     -fx-border-color: #fff;
5     -fx-background-color: #fff;
6     -fx-text-fill: #fff;
7 }
8 #solutionsArea .content {
9     -fx-background-color: #333333;
10 }
11 #AnchorPane {
12     -fx-background-color: white;
13 }
14 #minTerms .content{
15     -fx-background-color: #333;
16 }
17 #dontCares .content{
18     -fx-background-color: #333;
19 }
```



b) Main.fxml

- serves as the primary frontend output that displays the GUI itself. This is done using Scenebuilder and connecting it to Java using FXML. An illustration is provided below.



1.5 Limitations



This section mentions some instances and considerations that impose limitations on the simulator project.

- 1. The simulator requires **ALL EXTERNAL JARS** provided along the source codes.
- 2. It is not designed to accept inputs other than integers, hence the code provides no proper handling of such instances.
- 3. Given time complexity NP Complete, the simulator may take a while (and even potentially crash) if input quantity is too large.

Feedback or Concerns

For any queries, feedback, or concerns regarding the project,

feel free to contact us =)

jebarcellano@up.edu.ph

jpcoscolluela@up.edu.ph