QUINE MCCLUSKEY SOLVER:

TECHNICAL MANUAL

KLOWEE PO [1], BENJAMIN GABRIEL VELARDE [2]

CONTENTS

CS 130: Logic Gates and Circuit Design

1 INTRODUCTION

1.1 The Quine McCluskey Method

The Quine-McCluskey algorithm is an algorithm that is used to simplify Boolean functions and expressions along with algebraic manipulation, Karnaugh mapping et cetera. This algorithm is best used for simplifying Boolean functions with six variables or more (the other two can also be used but it can be a daunting task to do if these are the methods applied). The developers created a program that will simplify Boolean expressions using the algorithm, and the source codes are briefly explained here.

---

2 THE FXML CLASS

The FXML class contains the source code that will be used for the Graphic User Interface (GUI) of the program. This code seeks to attain a more user friendly version of the application. The developers used SceneBuilder to develop the GUI of the program.

a) `submit`:This method will process the user input in the GUI version of the program.

b) `openSolution`:This will display the full solution in the GUI version of the program.

Method/Constructor 1: Driver Class Method submit()

```
1   @FXML
2    public void submit(ActionEvent event) {
3
4     String input = numbers.getText();
5     input = input.replaceAll("\\s","");
6
7     String dc = numbers2.getText();
8     dc = dc.replaceAll("\\s","");
9
10    ArrayList<String> mintermsRaw = new ArrayList<String>();
11    Collections.addAll(mintermsRaw, input.split(","));
12    ArrayList<String> dcRaw = new ArrayList<String>();
13    if(!dc.isEmpty())
14         Collections.addAll(dcRaw, dc.split(","));
15
16    try {
17         openSolution();
18         InputHandler handler = new InputHandler();
19         handler.generateInput(mintermsRaw, dcRaw);
20         QuineMcCluskey qm = new QuineMcCluskey(handler);
21         qm.solve();
22         solution = new Text(qm.toPrint.toString());
23         solution.setFill(Color.WHITE);
24         solution.setWrappingWidth(500);;
25         scrollpane.setContent(solution);
26
27    } catch (Exception e) {
28         System.out.print("Invalid input!");
```

```
29          solution = new Text("ERROR! There must be a problem with your
                input.");
30          solution.setFill(Color.RED);
31          scrollpane.setContent(solution);
32         }
33     }
```

Method/Constructor 2: Driver Class Method openSolution()

```
1   @FXML
2   public void openSolution() throws IOException {
3     try {
4           FXMLLoader loader = new FXMLLoader();
5           loader.setLocation(getClass().getResource("solution.fxml"));
6           loader.setController(this);
7           Parent parent = loader.load();
8           ((Stage)submit.getScene().getWindow()).setScene(new Scene(parent,
                600,680));
9
10          back.setOnAction(new EventHandler<ActionEvent>() {
11              @Override public void handle(ActionEvent e) {
12                  try {
13                     FXMLLoader loader = new FXMLLoader();
14                      loader.setLocation(getClass().
15                          getResource("main1.fxml"));
16                      Parent parent = loader.load();
17                      ((Stage)back.getScene().getWindow()).setScene(new
                            Scene(parent, 600,680));
18                  } catch (IOException eox) {
19                       eox.printStackTrace();
20                  }
21              });
22          }
23     }
24     catch (IOException eox) {
25        eox.printStackTrace();
26     }
27   }
```

3 THE INPUTHANDLER CLASS

The InputHandler class handles the user inputted variables (the minterms) that will be processed and simplified by the program.

a) `generateInput`: this method will sort the user input in increasing (numerical) order, and it will convert the user input into binary numbers. This will also left-pad zeros to some converted binary numbers in order to have an equal number of digits. It will also handle repeating minterms and determine the number of variables needed.

b) `getNumberOfVar`: returns the number of variables in the user input

c) `getListOfNumbers`: returns the list of minterms in integer form.

d) `getListDontCaresRemoved`: returns the list of don't-care minterms.

e) `getMinterms`: returns the list of minterms in binary form.

Method/Constructor 3: InputHandler Class Method generateInput()

```
public void generateInput(ArrayList<String> mintermsRaw, ArrayList<String>
    dcRaw){
      if(!dcRaw.isEmpty()) {
           mintermsRaw.addAll(dcRaw);
           for(String s: dcRaw) {
                dcInt.add(Integer.parseInt(s));
           }
      }

      Collections.sort(mintermsRaw, new Comparator<String>() {
         @Override
         public int compare(String o1, String o2) throws RuntimeException {
             return Integer.valueOf(o1).compareTo(Integer.valueOf(o2));
         }
     });

      //The last element (largest number) was accessed to determine the length
         or number of variables needed.
      String lastDigit = Integer.toBinaryString(Integer.parseInt(mintermsRaw.
                         get(mintermsRaw.size()-1)));
      numVariables = lastDigit.length();

      for (int i=0; i < mintermsRaw.size(); i++) {
           int number = Integer.parseInt(mintermsRaw.get(i));
           // to avoid repeating inputs in integer form minterms
           if (!mintermsInt.contains(number)) {
                mintermsInt.add(number);
           }

           String binary = Integer.toBinaryString(number);
           // left-padding zeroes so that each binary string are of the same
              length.
           if (binary.length() < numVariables) {
                int index = binary.length();
                while(index != numVariables) {
                     binary = "0" + binary;
                     index++;
                }
           }
           // avoid repeating inputs in binary form minterms
           if(!minterms.contains(binary))
                minterms.add(binary);
      }
      ndcInt.addAll(mintermsInt);
      ndcInt.removeAll(dcInt);
 }
```

4 THE COMPARE CLASS

The Compare class handles the comparison between the minterms inputted by the user. This is an important step in the algorithm, as it is used repetitively.

4.1 getNumberOfOnes()

This method returns the number of ones in a minterm that is converted into a binary string.

Method/Constructor 4: Compare Class Method getNumberOfOnes()

```
public int getNumberOfOnes(String binary) {
     int one = 0;
      for (int i = 0; i < binary.length(); i++) {
            if (binary.charAt(i) == '1')
                one++;
        }
     return one;
}
```

4.2 compare()

This compares the number of ones between two binary strings (which is two minterms that are converted into a binary string).

Method/Constructor 5: Compare Class Method compare()

```
public int compare(String a, String b) {
     return getNumberOfOnes(a) - getNumberOfOnes(b);
 }
```

4.3 indexOfDifference()

This method will find what position the two compared binary strings differ in number.

Method/Constructor 6: Compare Class Method indexOfDifference()

```
public static int indexOfDifference(CharSequence cs1, CharSequence cs2) {
    int i;
    for (i = 0; i < cs1.length(); ++i) {
        if (cs1.charAt(i) != cs2.charAt(i)) {
            break;
        }
    }
    return i;
}
```

4.4 onlyOneDifference()

This method will confirm if there is indeed only one difference between the two compared binary strings.

Method/Constructor 7: Compare Class Method onlyOneDifference()

```java
boolean onlyOneDifference(String t1, String t2) {
   // if dashes not in same place or hamming dist != 1, return false
   int k = 0;
   for (int i = 0; i < t1.length(); i++) {
       if (t1.charAt(i) == '-' && t2.charAt(i) != '-')
           return false;
       else if (t1.charAt(i) != '-' && t2.charAt(i) == '-')
           return false;
     else if (t1.charAt(i) != t2.charAt(i))
           k++;
       else
           continue;
   }
   if (k != 1)
       return false;
   else
       return true;
}
```

4.5 difference()

This method will replace the position (index) with a differing number using a dash. (e.g. '1001' and '1000' - the only differing number is the last digit. This method will "merge" the two compared binary strings and the last digit will be replaced by a dash, thus the result will be '100-')

Method/Constructor 8: Compare Class Method difference()

```java
public String difference(String str1, String str2) {
    int at = indexOfDifference(str1, str2);
    StringBuilder temp = new StringBuilder(str1);
    temp.setCharAt(at, '-');
    String newString = temp.toString();
    return newString;
}
```

5 THE QUINEMCCLUSKEY CLASS

This class is the most important class in the source codes as it does the whole Quine-McCluskey algorithm. This class will process and simplify the Boolean expression based on the user input.

5.1 Constructor

The constructor will handle the user input based on the InputHandler class.

Method/Constructor 9: QuineMcCluskey Class Constructor

```
public QuineMcCluskey(InputHandler inputs) {
     this.inputs = inputs;
}
```

5.2 groupByOnes()

This method will group the minterms according to how many ones do they have on their binary forms. This is the second step in the algorithm.

5.2.1 *Visualization:*

Sample minterms: 2, 5, 9, 13, 17, 20.

$$2 - 00010$$
$$5 - 00101$$
$$9 - 01001$$
$$13 - 01101$$
$$17 - 10001$$
$$20 - 10100$$

Note: Zeros are left-padded in order to have equal numbers of digits in their binary forms. This left-padding of zeros is done by the InputHandler class. This method will group the binary numbers according to the number of ones they have. This will result to:

$$\text{One ones} - 2(00010)$$
$$\text{Two ones} - 5(00101), 9(01001), 17(10001), 20(10100)$$
$$\text{Three ones} - 13(01101)$$

Method/Constructor 10: QuineMcCluskey Class Method groupByOnes()

```
 HashMap<ArrayList<Integer>, String>[] data = new
    HashMap[inputs.getNumberOfVar()+1];
  for (int i = 0; i < data.length; i++) {
      data[i] = new HashMap<>();
  }

  /* k represents the group which is the number of ones
   * every integer is put into an individually separated ArrayList<Integer>
   * this is for grouping in pairs for later. */
  for (int i = 0; i < inputs.getMinterms().size(); i++) {
      int k = compare.getNumberOfOnes(inputs.getMinterms().get(i));
      ArrayList<Integer> temp = new ArrayList<Integer>();
      temp.add(inputs.getListOfNumbers().get(i));
      data[k].put(temp, inputs.getMinterms().get(i));
  }
```

5.3 solve()

The elimination() method is called in this method to do the final step of the algorithm. (Explanation for the elimination() method is explained below).

Method/Constructor 11: QuineMcCluskey Class Method solve()

```
1   public void solve() {
2     /* @dataContent will contain the pre-filtered implicants (keys are the
         pair of integers, values are the binary form.) */
3     HashMap<ArrayList<Integer>, String>[] groupedInitially = groupByOnes();
4     HashMap<ArrayList<Integer>, String>[] results;
5
6     /* @marked returns true if it is taken(has a partner with only one
         difference in their binary string),
7      * false if not taken. */
8     boolean marked;
9     do {
10         marked = false;
11         /* @results contains the initial filtered for every merging, it
              changes contents every iteration.
12          * its length is reduced at every iteration. At every filter, the
               groups are reduced by one. */
13         results = new HashMap[groupedInitially.length - 1];
14
15         /* @taken contains pairs with one difference */
16         HashSet<Object> taken = new HashSet<>();
17         /* @temp is for checking so that no pair would repeat in results. */
18         ArrayList<Object> temp;
19
20         /* iterate at every group. */
21        for (int i = 0; i < groupedInitially.length - 1; i++) {
22          results[i] = new HashMap<>();
23          temp = new ArrayList<>();
24          /* iterate every element in the preceding group */
25          for (ArrayList<Integer> keyPair : groupedInitially[i].keySet()) {
26              ArrayList<Integer> pair = new ArrayList<Integer>();
27              /* iterate every element in the succeeding group */
28              for (ArrayList<Integer> keyPair2 :
29                   groupedInitially[i+1].keySet()) {
30                   ArrayList<Integer> tempPair = new
                        ArrayList<Integer>();
31                   /* add the first half of the pair */
32                   tempPair.addAll(keyPair);
33                   /* if there is only one difference, add it to the taken
                        HashSet */
34                   if (compare.onlyOneDifference(groupedInitially[i]
35                        .get(keyPair), groupedInitially[i+1].get(keyPair2))){
36                      taken.add(groupedInitially[i].get(keyPair));
37                      taken.add(groupedInitially[i + 1].get(keyPair2));
38                      String diff =
                           compare.difference(groupedInitially[i].get(keyPair),
                           groupedInitially[i+1].get(keyPair2));
```

```
39                          taken.add(diff);
40                          tempPair.addAll(keyPair2);
41
42                          /* if temp does not have the pair and binary, add it
                               to the results, this way
43                           * there would be no repetitions. */
44                          if (!temp.contains(diff) && !temp.contains(tempPair)) {
45                              results[i].put(tempPair, diff);
46                              marked = true;
47                          }
48                          temp.add(diff);
49                          temp.add(tempPair);
50                      }
51                       /* pair may contain a pair(keyPair, keyPair2) if it is
                            taken, it may only contain keyPair if it is
52                        * not taken (NOTE: keyPair may contain a single number
                             if from the first merging, it has no partner.)*/
53                      pair = tempPair;
54                  }
55              }
56          }
57
58          /* the results contain the filtered, replace contents of
               groupedInitially then repeat the filtering until
59           * filtering becomes impossible to conduct. */
60          if (marked) {
61              /* if it is not taken, add it immediately to the list of finalists
                   */
62               for (HashMap<ArrayList<Integer>, String> hash : groupedInitially) {
63                    for (ArrayList<Integer> keypair: hash.keySet()) {
64                        if (!taken.contains(hash.get(keypair))) {
65                            dataContent.put(keypair, hash.get(keypair));
66                        }
67                    }
68                }
69              groupedInitially = results;
70          }
71
72      } while(marked && (groupedInitially.length > 1));
73      /* end of do-while loop */
74      /* after filtering everything, add the remaining implicants */
75      for (HashMap<ArrayList<Integer>, String> filtered : groupedInitially) {
76          dataContent.putAll(filtered);
77      }
78
79      if (dataContent.size() > 1)
80        elimination();
81      else
82          /* print answer using convertToLetters() */
83  }
```

5.4 elimination()

this method will filter out the essential prime implicants. This is the final step in the algorithm. In the manual set-up, this is the time wherein the groups without a check mark is tabulated to find out the essential prime implicants.

Method/Constructor 12: QuineMcCluskey Class Method elimination()

```
1  public void elimination() {
2       HashSet<ArrayList<Integer>> NprimeImplicants =  new
           HashSet<ArrayList<Integer>>();
3       ArrayList<Integer> primes = new ArrayList<Integer>();
4       ArrayList<Integer> tempMinterms = inputs.getListDontCaresRemoved();
5
6       for (Integer minterm: tempMinterms) {
7            int count = 0;
8            for (ArrayList<Integer> keyPair : dataContent.keySet()) {
9                 if(keyPair.contains(minterm)) {
10                     count++;
11                     if(count > 1)
12                          break;
13                }
14           }
15           if (count == 1)
16                primes.add(minterm);
17      }
18
19      Set<Integer> forCheck = new HashSet<Integer>();
20      for (ArrayList<Integer> keyPair : dataContent.keySet()) {
21           for(Integer prime : primes)
22                if(keyPair.contains(prime)) {
23                     essentialPrimeImplicants.add(keyPair);
24                     forCheck.addAll(keyPair);
25                }
26           if(!essentialPrimeImplicants.contains(keyPair))
27                NprimeImplicants.add(keyPair);
28      }
29
30      for (ArrayList<Integer> keys : essentialPrimeImplicants)
31            tempMinterms.removeAll(keys);
32
33      while(!forCheck.containsAll(inputs.getListDontCaresRemoved())) {
34           ArrayList<Integer> temp = new ArrayList<Integer>();
35           for (ArrayList<Integer> k : NprimeImplicants) {
36                Set<Integer> intersection = new HashSet<Integer>(k);
37                Set<Integer> intersection2 = new HashSet<Integer>(temp);
38                intersection.retainAll(tempMinterms);
39                intersection2.retainAll(tempMinterms);
40                if(intersection.size() > intersection2.size()) {
41                     temp = k;
42                }
43           }
44           tempMinterms.removeAll(temp);
```

```
45              essentialPrimeImplicants.add(temp);
46              forCheck.addAll(temp);
47          }
48      }
```

5.5 convertToLetters()

This method will convert the final answer into letters. The conversion will be as follows:
First digit - A, second digit - B, third digit - C and so on. If the digit is 0, then it is unprimed (no apostrophe) and if the digit is 1, it is primed (with apostrophe).

Method/Constructor 13: QuineMcCluskey Class Method convertToLetters()

```
1   private String convertToLetters(String binary) {
2       String letters = "";
3       char var = 'A';
4       for (int i = 0; i < binary.length(); i++) {
5           if(binary.charAt(i) == '1') {
6               letters += var;
7           }
8           else if(binary.charAt(i) == '0')
9               letters += var + "'";
10          var++;
11      }
12      return letters;
13  }
```

6 THE MAIN CLASS

The Main class is the class that starts the program. It also launches the GUI version of the program.

Method/Constructor 14: Main Class Method start()

```
1   public class Main extends Application {
2       @Override
3       public void start(Stage primaryStage) {
4           try {
5               Parent root =
                    FXMLLoader.load(getClass().getResource("main1.fxml"));
6           Scene scene = new Scene(root,600,680);
7               primaryStage.setTitle("Quine McCluskey App");
8               primaryStage.setScene(scene);
9               primaryStage.show();
10          } catch(Exception e) {
11              e.printStackTrace();
12          }
13      }
14      public static void main(String[] args) {
15          launch(args);
16      }
17  }
```