Tabulation Method Online Simulator TECHNICAL MANUAL

Evangeline Louise Carandang
Joseph Niel Tuazon

University of the Philippines Manila
Department of Physical Sciences and Mathematics
BS Computer Science

February 2014

INTRODUCTION

This manual is a documentation of the algorithm used by the creators of the online simulator. It is divided into sections where in each sections, the functions or the source code is thoroughly explained. The authors made use of JavaServer Pages or JSP to build the webpage. The source code is based on the Tabulation method by Quine and McCluskey.

The Quine – McCluskey Method is also known as the Tabulation method or the method of prime implicants. It is developed by W.V. Quine and E.J. McCluskey in 1956. This method is used to minimize Boolean functions just like Karnaugh mapping; however, this method is more efficient to use in computer algorithms and it provides more accurate solutions to validate the simplified form of the Boolean function. The main goal is to find all prime implicants and use them to find the essential prime implicants that can be used in the possible simplified forms of the function.

TABLE OF CONTENTS

1.0 GETTING INPUT FROM THE USER

The user is given an example that his/her input must be separated by commas or spaces. The input is in type String, then it is tokenized and put into an ArrayList. As it is put into the array, it is converted to an integer. If the user inputted the same number more than once or if a number is found to be already in the array, the number of tokens is decreased and it is not added in the array. It is then sorted from smallest to largest.

```
String input = request.getParameter("input");

input = input.replaceAll(","," ");

StringTokenizer token = new StringTokenizer(input);

int tokenCount = token.countTokens();

ArrayList<Integer> inputArray = new ArrayList<Integer>();

int tempTokenCount = tokenCount;
for(int i = 0; i < tempTokenCount; i++){
      String current = token.nextToken();

      int currentNumber = Integer.parseInt(current);

      if(!inputArray.contains(currentNumber)){
            inputArray.add(currentNumber);
      }
      else{
            tokenCount = tokenCount - 1;
      }
}

Collections.sort(inputArray);
```

2.0 CONVERSION FROM DECIMAL TO BINARY

The decimal numbers in the user's input are converted to binary and assigned an element to the multi-dimensional array. The length of each binary is dependent on the length of the decimal number which has the most 1s. If the binary lacks length, 0s are appended to the beginning of the binary. As this is done, the number of 1s is being counted for the grouping later.

```
String[][] BinaryInputArray = new String[tokenCount][4];
	int maxLengthOfBinary = 0,
		numberOfGroups = 0;

ArrayList<Integer> numberOfOnesGroup = new ArrayList<Integer>();

for(int i = 0; i < tokenCount; i++){
	String binaryString = Integer.toBinaryString(inputArray.get(i));
	BinaryInputArray[i][0] = binaryString;
	BinaryInputArray[i][1] = String.valueOf(inputArray.get(i));

	int numberOfOnes = 0;

	for(int j = 0; j < binaryString.length(); j++){
		if(binaryString.substring(j,j+1).equals("1")){
			numberOfOnes++;
		}
	}

	if(!numberOfOnesGroup.contains(numberOfOnes)){
		numberOfOnesGroup.add(numberOfOnes);
	}

	BinaryInputArray[i][2] = String.valueOf(numberOfOnes);

	BinaryInputArray[i][3] = "0";

	if(maxLengthOfBinary < binaryString.length()){
		maxLengthOfBinary = binaryString.length();
	}
}

numberOfGroups = numberOfOnesGroup.size();

for(int i = 0; i < tokenCount; i++){
	while(BinaryInputArray[i][0].length() != maxLengthOfBinary){
		BinaryInputArray[i][0] = "0" + BinaryInputArray[i][0];
	}
}
```

3.0 GROUPING OF BINARIES ACCORDING TO NUMBER OF 1s

This part groups together the binary representations that have the same number of 1s into an ArrayList.

```
ArrayList<ArrayList> superArray = new ArrayList<ArrayList>();
ArrayList<ArrayList> mainArray = new ArrayList<ArrayList>();

for(int i = 0; i < numberOfGroups;i++){
      ArrayList<ArrayList> groupArray = new ArrayList<ArrayList>();
      for(int j = 0; j < tokenCount; j++){
            ArrayList<String> nodeArray = new ArrayList<String>();
            if(numberOfOnesGroup.get(i) == Integer.parseInt(BinaryInputArray[j][2])){
                  for(int k = 0; k < 4; k++){
                        nodeArray.add(BinaryInputArray[j][k]);
                  }
                  groupArray.add(nodeArray);
            }
      }
      mainArray.add(groupArray);
}

superArray.add(mainArray);
```

4.0 COMPARING TWO BINARIES FROM EACH GROUP

```
loopTerminator = 0;
while(loopTerminator == 0){
        int numberOfComparisons = numberOfGroups - 1;

        ArrayList<ArrayList> currentTable = superArray.get(superArray.size()-1);
        ArrayList<ArrayList> newTable = new ArrayList<ArrayList>();

        for(int i = 0; i < numberOfComparisons; i++){
                ArrayList<ArrayList> firstGroupToCompare = currentTable.get(i);
                ArrayList<ArrayList> secondGroupToCompare = currentTable.get(i+1);
                ArrayList<ArrayList> resultingGroup = new ArrayList<ArrayList>();

                for(int j = 0; j < firstGroupToCompare.size(); j++){
                        for(int k = 0; k < secondGroupToCompare.size(); k++){
                                ArrayList firstNumberToCompare = firstGroupToCompare.get(j);
                                ArrayList secondNumberToCompare = secondGroupToCompare.get(k);
                                ArrayList resultingNumber =
                                compareBinaryDigits(firstNumberToCompare,secondNumberToCompare);

                                if(resultingNumber != null){
                                        firstNumberToCompare.set(3,"1");
                                        secondNumberToCompare.set(3,"1");

                                        int repeatedNumberIndicator = 0;

                                        for(int l = 0; l < resultingGroup.size(); l++){
                                                ArrayList tempNode = resultingGroup.get(l);
                                                if(tempNode.contains(resultingNumber.get(0))){
                                                        repeatedNumberIndicator = 1;
                                                }
                                        }

                                        if(repeatedNumberIndicator == 0){
                                                resultingGroup.add(resultingNumber);
                                        }
                                }
                                firstGroupToCompare.set(j, firstNumberToCompare);
                                secondGroupToCompare.set(k, secondNumberToCompare);
                        }
                }

                if(resultingGroup.size()!=0){
                        newTable.add(resultingGroup);
                }
                else{
                        loopTerminator = 1;
                }

                currentTable.set(i, firstGroupToCompare);
                currentTable.set(i+1, secondGroupToCompare);
        }

        superArray.set(superArray.size()-1, currentTable);
        numberOfGroups = newTable.size();

        if(numberOfGroups == 0){
                loopTerminator = 1;
        }
        else{
                superArray.add(newTable);
        }
}
```

```java
public ArrayList compareBinaryDigits(ArrayList<String> first, ArrayList<String> second){
        ArrayList<String> returnArrayString = new ArrayList<String>();

        String firstBinary = first.get(0);
        String secondBinary = second.get(0);

        int repeatIndicator = 0;
        String tempReturnString = new String();

        for(int i = 0; i < firstBinary.length(); i++){
                if(!firstBinary.substring(i,i+1).equals(secondBinary.substring(i,i+1))){
                        repeatIndicator++;
                        tempReturnString = tempReturnString + "-";
                }
                else{
                        tempReturnString = tempReturnString + firstBinary.substring(i,i+1);
                }
        }

        if(repeatIndicator == 1){
                returnArrayString.add(tempReturnString);

                String formattedFirstDecimal =
                        first.get(1).replaceAll("\\(","").replaceAll("\\)","");
                String formattedSecondDecimal =
                        second.get(1).replaceAll("\\(","").replaceAll("\\)","");

                String formattedResultDecimal = "(" + formattedFirstDecimal + ", " +
                        formattedSecondDecimal + ")";

                returnArrayString.add(formattedResultDecimal);
                returnArrayString.add(first.get(2));
                returnArrayString.add("0");
        }
        else{
                return null;
        }

        return returnArrayString;
}
```

This part is responsible for comparing each binary representations for each group until none can be compared. Representations with exactly one different digit is replaced with “-“.

5.0 FUNCTIONS/METHODS

```java
public String convertToDecimalRepresentation(ArrayList<Integer> array){

		String returnString = new String();

		returnString = returnString + "(";
		for(int a : array){
			returnString = returnString + a + ", ";
		}
		returnString = returnString.substring(0,returnString.length()-2) + ")";

		return returnString;
}
```

This function converts the contents of the array to a decimal representation of type String that is used for comparing the decimal of the possible answers. If the String and the possible answer's decimal representation are equal, the decimal's variable representation or literals will be added to the list of possible solutions.

```java
public String convertToLetters(String binary, int maxLengthOfBinary){
		String resultingString = new String();
		for(int j = 0; j < maxLengthOfBinary; j++){
			String tempDigit = binary.substring(j,j+1);
			if(tempDigit.equals("1")){
				resultingString = resultingString + letters[j];
			}
			else if(tempDigit.equals("0")){
				resultingString = resultingString + letters[j] + "'";
			}
		}
		return resultingString;
	}
```

This function converts the binary representation of type String to the corresponding literal value.

```
public ArrayList<String> compareBinaryDigits(ArrayList<String> first, ArrayList<String> second){

		ArrayList<String> returnArrayString = new ArrayList<String>();

		String firstBinary = first.get(0);
		String secondBinary = second.get(0);

		int repeatIndicator = 0;
		String tempReturnString = new String();

		for(int i = 0; i < firstBinary.length(); i++){
			if(!firstBinary.substring(i,i+1).equals(secondBinary.substring(i,i+1))){
				repeatIndicator++;
				tempReturnString = tempReturnString + "-";
			}
			else{
				tempReturnString = tempReturnString + firstBinary.substring(i,i+1);
			}
		}

		if(repeatIndicator == 1){
			returnArrayString.add(tempReturnString);

			String formattedFirstDecimal =
				first.get(1).replaceAll("\\(","").replaceAll("\\)","");
			String formattedSecondDecimal =
				second.get(1).replaceAll("\\(","").replaceAll("\\)","");

			String formattedResultDecimal = "(" + formattedFirstDecimal + ", " +
				formattedSecondDecimal + ")";

			returnArrayString.add(formattedResultDecimal);
			returnArrayString.add("0");

		}
		else{
			return null;
		}

		return returnArrayString;
	}
```

This function compares two binary representations from different groups. If they exactly have one difference in terms of their binary digits, then the function will return the new binary. Else, it will return null.

The rest of the code used compares the implicants and groups them into two: prime implicants and essential prime implicants. The essential prime implicants are then converted to its variable representation that is later added to the final answer. On the other hand, the rest of the prime implicants are then compared to one another to find the rest of the possible answers.