KARADENİZ TEKNİK ÜNİVERSİTESİ

Bilgisayar Mühendisliği

Birinci Ödevi Raporu

Ad: Jamiu Oluwaseun

Soyad: Ojeleye

No: 359704

Ders: Ayrık Matematik

Ders Sorumlusu: Prof. Dr. Vasıf NABIYEV

Dili: İngilizce

Problem Statement:

A program that minimizes a boolean function inputted in Normal SOP(Sum of Product) form.

Graph theory and recursion concept were used to apply maximum minimization to inputted boolean functions.

Algorithm:

The function was minimized using the procedures below:

1. Finding relations between the minterms:

Assuming we have n-minterms in the function, the relationship between the minterms can be determined using a square matrix, M of order n. This matrix will contain terms that define the relationship between one minterm and another.

If the minterms differ by one digit position, we will combine them and store the result in a corresponding M_{ij} entry on the matrix. Otherwise, the corresponding entry on the matrix is "---".

Example: For minterms 110 and 111, they can be combined to form "11-" minterm. For minterms 101 and 110, they cannot be combined so we represent their corresponding entry on the matrix with "---".

2. Separation of minterms:

After the first step, the matrix is represented on a graph to separate the minterms into EI_{set} and PI_{set}.

EI_{set}: This set updates itself with new Essential Implicants(Mutlak Satir, EI) during every recursion.

EI: This is a minterm whose node has no edge.

PI_{set}: This set stores Prime Implicants(PI) when the minimization function is called in the program. It initializes to an empty set after every recursion.

PI: This is a minterm whose node has 1 or more edge.

After the separation of the minterms into EI_{set} and PI_{set}, the union of EI_{set} and PI_{set} which can be represented as F (that is $F=EI_{set} \cup PI_{set}$) is then printed as a more simplified version of the function during every recursion.

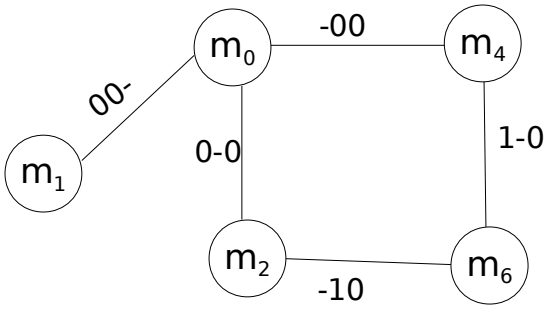
3. The process repeats itself again from step 1. with the new minterms=PI_{set}. It recurs until PI_{set} is empty.

Algorithm Illustration Using Graph Theory:

To minimize F=V(0,1,2,4,6):

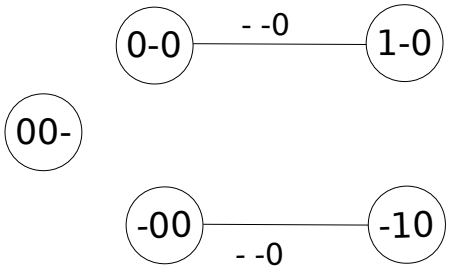
minterms={m_0, m_1, m_2, m_4, m_6,}

	m_0	m_1	m_2	m_4	m_6
m_0	---	00-	-00	-00	---
m_1	00-	---	---	---	---
m_2	0-0	---	---	---	-10
m_4	-00	---	---	---	1-0
m_6	---	---	-10	1-0	---



EI_{set}={}
PI_{set}={m_0, m_1, m_2, m_4, m_6}
F={m_0, m_1, m_2, m_4, m_6}

	00-	0-0	-00	-10	1-0
00-	---	---	---	---	---
0-0	---	---	---	---	--0
-00	---	---	---	--0	---
-10	---	---	--0	---	---
1-0	---	--0	---	---	---



EI_{set}={00-}
PI_{set}={0-0, 1-0, -00, -10}
F={00-, 0-0, 1-0, -00, -10}

	--0
--0	---

EI_{set}={00-, --0}
PI_{set}={}
F={00-, --0}

Finally, the minimiztion of F=V(0,1,2,4,6) is $\overline{A}\overline{B}+\overline{C}$

Pseudocode:

1. EI_{set}=[]
2. minterms=input
3. n=number of minterms
4. PI_{set}=[]
5. MinimizationTable=[][]
6. for(i=0; i<n; i++):
7. for(j=0; j<n; j++):
8. if(minterms[i] and minterm[j] have a relation):
9. MinimizationTable[i][j]=relation
10. else:
11. MinimizationTable[i][j]="---"
12. EI=getEI(MinimizationTable)
13. PI=getPI(MinimizationTable)
14. EI_{set} =EI_{set} +EI
15. PI_{set} =PI
16. F=EI_{set} +PI_{set}
17. print(F)
18. if (PI_{set} is not empty):
19. minterms=PI_{set}
20. goto line 3
21. else:
22. End

Implementation:

The algorithm was implemented in python. A class called **BooleanFunction** performs all the minimization operation. The class contains 15 methods and their functions are as follows:

1. Constructor: It initialize and assigns values to the global and private variables that will be used in the class.

2. getBooleanVar: It returns boolean variables in string form. That is, A,B,C,... or x1,x2,x3,... This is useful when we are outputting the simplified function, F with a special printing function.

3. BinConverter: It Converts decimal number to binary number which is returned in string form. Example: 2 is returned as "10".

4. AddZeroAtFront: It adds zeros string at the front of a binary number (in string form) if its number of bits is not complete. Example: "10" is returned as "010" for a 3-bit operation and "0010" for a 4-bit operation.

5. BinlistConverter: It converts a list with decimal numbers to a string list with equivalent binary numbers. It uses both BinConverter and AddZeroAtFront Methods to perform this operation. Example: [0,1,2,4,6] is converted to ["000", "001", "010", "100", "110"].

6. isGrayCode: It checks if two binary numbers differ by one digit position. It returns boolean variables.

7. ChangeComplements: If two binary numbers have a relation, it combines them together and returns a simplified term. Example: 000 and 001 will be simplified to "00-".

8. emptydashes: It generates n-bits of dashes. Example: for 3bits, it returns "---".

9. MinimizingTable: It computes the relations between minterms using matrix before they are represented on a graph.

10 & 11. getEssentialImplicants and getPrimeImplicants: They compute the EI and PI from the minterms respectively using the MinimizingTable's returned matrix.

12. Convert_minterm_variables: This method is used for converting binary numbers to alphabetical or alphanumeric form. Example: for 00-, $x_1'x_2'$ is returned. For 1011, $x_1x_2'x_3x_4$ is returned.

13. Special Print: This is a special print method for printing the binary minterms in F. These binary minterms are converted to alphabetical form before they are outputted.

14. SimplifywithRule: The minimization of the function and recursion is done here. In this method, the union of EI_{set} and PI_{set} is assigned to a variable, F, whose terms are then outputted using the SpecialPrint method. This action repeats itself until PI_{set} is empty.

15. Minimize: It combines all the major methods together for minimization of the Boolean function.

Code:

```
class BooleanFunction():
    def __init__(self,List):
        ##Getting maximum minterm(number) in list
        max_no=max(List)
        ##Getting no of binary bits/variables for greatest number
        self.variables=len("{0:b}".format(max_no))
        self.List=List
        self.EI=[]

    ## Main Method for minimization
    def minimize(self):
        L=self.BinlistConverter(self.List,self.variables)
        self.SpecialPrint(L)
        self.simplifywithRule(L)
        print()

    # getting boolean variables in string form
    def getBooleanVar(self):
        string=list()
        variables=["x1","x2","x3","x4","x5","x6","x7","x8"]
        for i in range(self.variables):
            string.append(variables[i])
        return variables

    #Binary number Converter
    def BinConverter(self,decimal):
        binary=""
        while(decimal!=0):
            r=decimal%2
            decimal=decimal//2
            binary=str(r)+binary
        return binary
```

```
## adding 0 at front of binary if it has less bits
def addZeroAtFront(self, binary,VarNo):
        for i in range (VarNo-len(binary)):
                binary="0"+binary
        return binary

## Method for converting a list with number to binary list
##Example F=[0,1,2,4,6] converted to F=["000","001","010","100","101"]
def BinlistConverter(self,decimallist,VarNo):
        binList=[]
        for i in range (len(decimallist)):
                b=self.addZeroAtFront(self.BinConverter(decimallist[i]),VarNo)
                binList.append(b)
        return binList

## Method for checking if two binaries differ in only one bit
def isGrayCode(self, binary1,binary2):
        count=0
        for i in range (self.variables):
                if(binary1[i]!=binary2[i]):
                        count=count+1
        if(count==1):
                return True
        else:
                return False

## Method for simplifying two binary numbers
## Example 000 and 001 will be simplified to 00-
def changeComplements(self, binary1, binary2):
        term=""
        for i in range (self.variables):
                if(binary1[i]!=binary2[i]):
                        term=term+"-"
```

```
                else:
                        term=term+binary1[i]
        return term

## Method for returning n-bit of dashes E.g "---"
def emptydashes(self):
        s=""
        for i in range (self.variables):
                s=s+"-"
        return s

##Minimization Of terms is done on tablehere
##It's Like a Table... It is filled with minterms
def MinimizingTable(self, List):
        MT=[]
        if(len(List)>1):
                for i in range(len(List)):
                        MT.append([])
                        for j in range(len(List)):
                                if(i!=j):
                                        if(self.isGrayCode(List[i],List[j])):

MT[i].append(self.changeComplements(List[i],List[j]))
                                        else:
                                                MT[i].append(self.emptydashes())
                                else:
                                        MT[i].append(self.emptydashes())
                return MT
        else:
                return List

## Computing Prime Implicants on an already filled Matrix MT
def getPrimeImplicants(self, MT):
        PI=[]
```

```
        if(len(MT)>1):
            for i in range (len(MT[0])):
                for j in range (len(MT[0])):
                    if(MT[i][j]!=self.emptydashes()):
                        PI.append(MT[i][j])

            ##removing duplicates of prime implicants.
            PI=list(set(PI))
            if(len(PI)==0):
                return 0
            else:
                return PI
        else:
            return 0

    ## Computing Essential Implicants on an already filled Matrix MT
    def getEssentialImplicants(self, List, MT):
        EI=[]
        if(len(MT)>1):
            for i in range (len(MT[0])):
                c=0
                for j in range (len(MT[0])):
                    if(MT[i][j]==self.emptydashes()):
                        c=c+1
                if(c==len(MT[0])):
                    EI.append(List[i])
            return EI
        else:
            return List

## This Is the method that do all the minimization using the above Methods...
## It uses recursion by simplifying and printing every newly minimized expression until it gets to the last one
    def simplifywithRule(self,mintermslist):
        #print("MinTerm is: ",mintermslist)
```

```
        F=[]
        T=self.MinimizingTable(mintermslist)
        self.EI=self.EI+self.getEssentialImplicants(mintermslist, T)##Essential Implicant
        PI=self.getPrimeImplicants(T)     ##Prime Implicants

        ## Keeping all minterms from EI and PI in a list called F
        if (PI!=0):
                F=self.EI+PI
        else:
                F=self.EI
        self.SpecialPrint(F)

        ## Continue recursion by simplifying PI until we have no PI anymore
        while(PI!=0):
                return self.simplifywithRule(PI)



## Special Method for printing
def SpecialPrint(self, F):
        print("\nF=", end=" ")
        for i in range (len(F)):
                if(F[i]!=self.emptydashes()):
                        print(self.Convert_minterm_variable(F[i]), end=" ")
                        #print(F[i])
                if(i!=(len(F)-1)):
                        print("+",end=" ")
        return 0

## Method for converting binary minterms to string form...
## Example (00- is A`B`)  and  (1011 is AB`CD)
def Convert_minterm_variable(self, boolVar):
        temp=""
        variables=self.getBooleanVar()
        for i in range (len(boolVar)):
```

```
                        if(boolVar[i]!="-"):
                                if(boolVar[i]=="0"):
                                        temp=temp+variables[i]+"\'"
                                else:
                                        temp=temp+variables[i]
                return temp


def main():
        ###     Testing...
        ##l=[0,1,4,5,8,9,12,13]

        print("Input Minterms: (Example- 0 1 4 5 8 9 12 13)")
        l=[int(i) for i in input().split()]


        F=BooleanFunction(l).minimize()
        print("Function Simplified.....")

main()
```

Result:

Assuming we want to minimize a boolean function F=V(0,1,2,4,6), we can minimize it by storing the minterms in a list before minimizing it with the BooleanFunction class.

Illustration1:

L=[0, 1, 2, 4, 6]

F=BooleanFunction(L).minimize()

Output1:

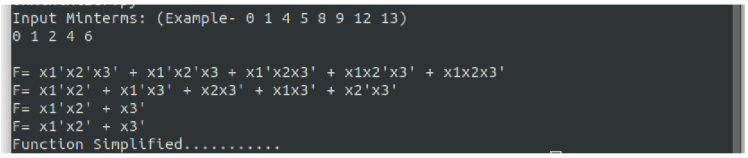


Illustration2:

L=[0, 1, 4, 5, 8, 9, 12, 13]

F=BooleanFunction(L).minimize()

Output2: