COL215 Assignment 1

Dhruv Ahlawat, 2021CS10556
and
Mihir Kaskhedikar, 2021CS10551

August 20, 2022

1 Introduction

In this report, we will discuss how we implemented assignment 1 about Karnaugh maps in Python. Both of us were in batch B and were taught about Karnaugh maps before, so we directly started working on it. As was mentioned in the Assignment sheet, we have considered a legal region to be any region corresponding to a term that does not have any 0's in it. Note that we are NOT validating if the term is minimal or if we have taken the biggest possible region with 1s since the assignment did not explicitly mention that a non-optimal region is an illegal region.

2 Method

Firstly, we ran a loop to convert all the 'None' elements to 2 in terms, to make it easier to handle (otherwise we would need to use conditional statements to check if the term is None every time we use it). Then we thought of a way to store the starting and ending rows and columns corresponding to the 9(3*3) possible two term expressions **(i.e. if the starting row is s and the ending row is e then all the rows (s, $(s+1)$ mod 4, ... ,$(e-1)$ mod 4, e) satisfy the two term expression)**. We created 2 Lists to store these locations, **start** and **end** respectively.

2.1 For size of terms = 4

For size = 4, any expression is of the form $abcd$ where $0 \leq a, b, c, d \leq 2$. We needed a way to access the start and end rows and columns via 2 variables (since 2 variables are assigned row-wise and 2 column-wise. We could have used 2-dimensional lists of start and end to store this, but finally decided on using a linear list which, for a 2-sized term ij, we access by using the index $3i+j$. then we filled the lists start and end to correspond to the starting and ending rows where the term ij is satisfied.
So let the terms be a, b, c, d. Since column indexing is same as rows, we get the top left corner as (start$[3c+d]$, start$[3a+b]$) and bottom right corner as (end$[3c+d]$, end$[3a+b]$).
Since the coordinate system has row number as the first index, and cd is what determines the rows of our region, therefore $3c+d$ occurs in the first term. Then we loop over the coordinates firstly column-wise and then row-wise to check if there are any 0s, and if there are then we set a Boolean value storing the status of the region as 'False'. We note down the indices of the start and end lists as z1 = $3c+d$ and z2 = $3a+b$ to use later when we return a value from this function.

2.2 For size of terms = 2

For size = 2, any expression is of the form ab where $0 \leq a, b \leq 2$. Also, since each row and column only has 1 variable, we accessed the start and end rows/columns for a term i as start$[i]$ and end$[i]$. Again since column indexing is same as row indexing, we get the top left corner as (start$[b]$, start$[a]$) and (end$[b]$, end$[a]$).
Then we loop over the coordinates firstly column-wise and then row-wise to check if there are any 0s, and if there are then we set a Boolean value storing the status of the region as 'False'. We note down the indices of the start and end lists as z1 = b and z2 = a to use later when we return a value from this function.

2.3 For size of terms = 3

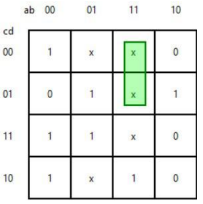
For size = 3, any expression is of the form abc where $0 \leq a, b, c \leq 2$. Here, the columns have 2 variables ab while the rows have only one c. Hence this case is a mixture of the previous 2 cases. We applied the same logic as the previous 2, and accessed the values of the starting and ending rows where c is satisfied as start[c] and end[c], and starting and ending columns where ab is satisfied as start[$3a + b$] and end[$3a + b$].
Therefore the top left corner is (start[c], start[$3a + b$]) and bottom right corner is (end[c], end[$3a + b$]).
Then we ran 2 nested loops to check for each element within this region, if there is a '0', we set the value of ans as 'False'. We note down the indices of the start and end lists as z1 = c and z2 = $3a + b$ to use later when we return a value from this function.

3 End

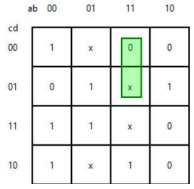
Finally, we return ((start[z1], start[z2]), (end[z1], end[z2]), ans).

4 Some Test Cases checked

Test Case 1: kmap function=[[1,'x','x',0],[0,1,'x',1],[1,1,'x',0],[1,'x',1,0]]
Expression=[1,1,0,None]
Output: ((0,2),(1,2),True)

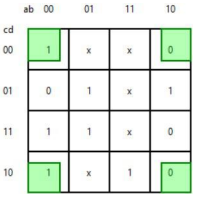


Test Case 2: kmap function=[[1,'x',0,0],[0,1,'x',1],[1,1,'x',0],[1,'x',1,0]]
Expression=[1,1,0,None]
Output: ((0,2),(1,2),False)

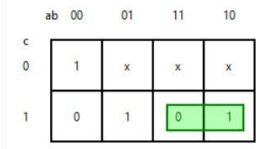


Test Case 3: kmap function= [[1,'x','x','x'],[0,1,0,1],[0,0,0,1],[1,'x',0,'x']]
Expression=[1,None,None,None]
Output: ((0,2),(3,3),False)

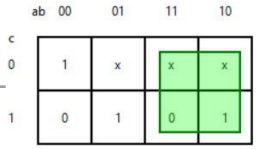
Test Case 4: kmap function= [[1,'x','x','x'],[0,1,0,1],[0,0,0,1],[1,'x',0,'x']]
Expression=[0,None,None,0]
Output: ((3,0),(0,1),True)



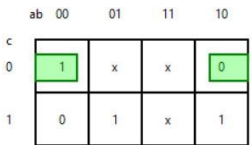
Test Case 5: kmap function= [[1,'x','x',0],[0,1,'x',1],[1,1,'x',0],[1,'x',1,0]]
Expression=[None,0,None,0]
Output: ((3,3),(0,0),False)



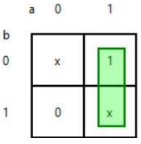
Test Case 6: kmap function= [[1,'x','x','x'],[0,1,0,1]]
Expression=[1,None,1]
Output: ((1,2),(1,3),False)



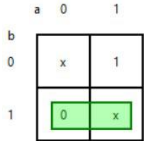
Test Case 7: kmap function=[[1,'x','x','x'],[0,1,0,1]]
Expression=[1,None,None]
Output: ((0,2),(1,3),False)



Test Case 8: kmap function=[[1,'x','x',0],[0,1,'x',1]]
Expression=[None,0,0]
Output: ((0, 3), (0, 0), False)



Test Case 9: kmap function=[['x',1],[0,'x']]
Expression=[1,None]
Output: ((0,1),(1,1),True)



Test Case 10: kmap function=[['x',1],[0,'x']]
Expression=[None,1]
Output: ((1,0),(1,1),False)