# REPORT FOR COL215 ASSIGNMENT-1

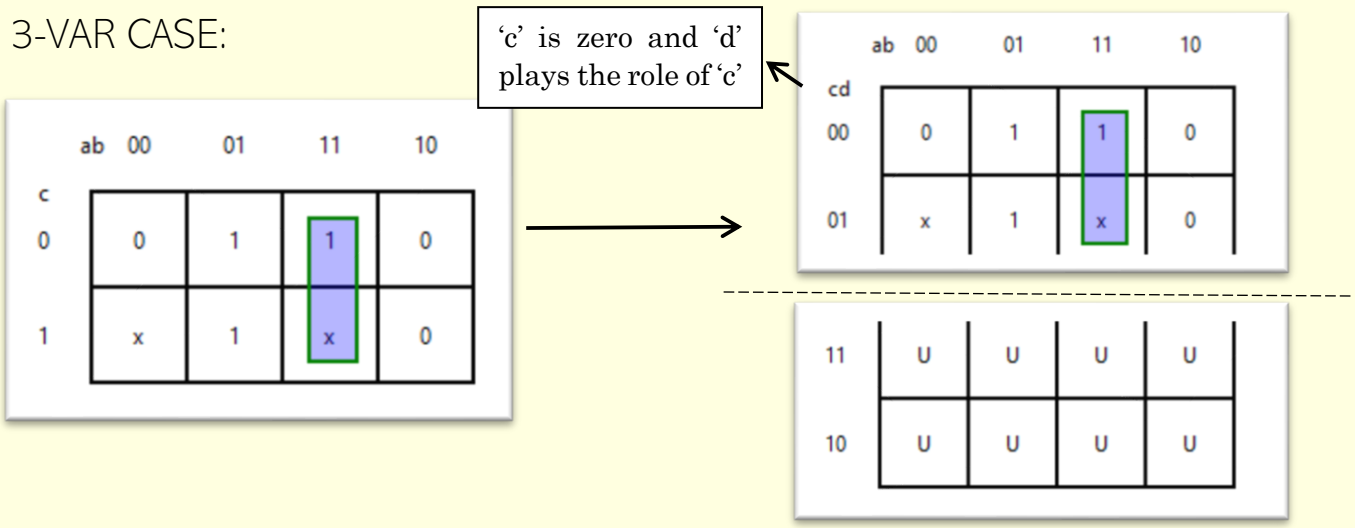
## Approach Summary:

The primary objective of the assignment is to write a function in python which takes a kmap_function and a term to return a 3-tuple. The main difference in the type of inputs the function will be provided is in the number of variables which can be either 2,3 or 4. This problem has been addressed by catering to the logic for a 4-variable kmap. The remaining inputs are taken care of by first converting them into 4-variable inputs and then solving them. Thus, the function is generalized in that sense.

## 4-VAR CASE:

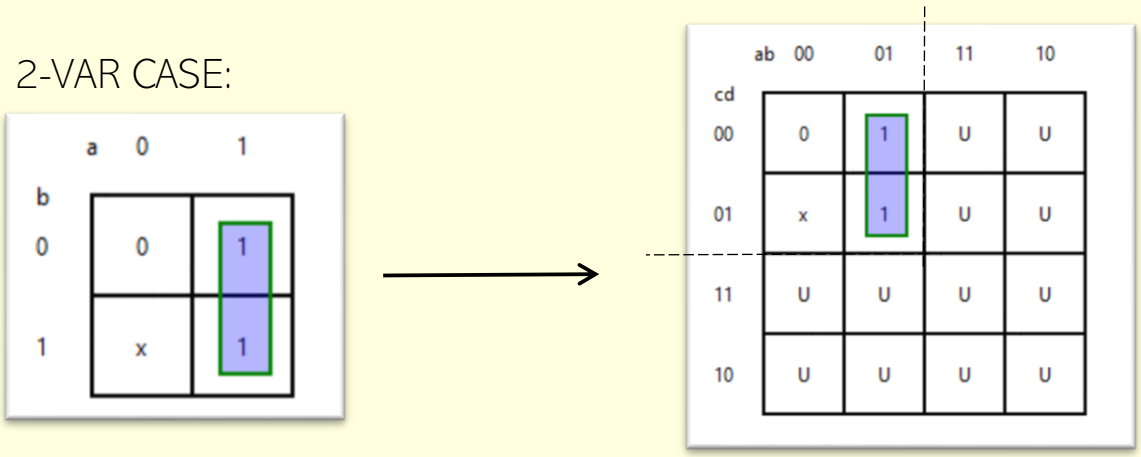
We clearly see that for a 4-variable kmap, the columns of region are decided by values corresponding to a, b and rows of region are decided by values corresponding to c, d. Two dictionaries are created namely 'acdict' for a, c and 'bddict' for b, d. Each dictionary contains key-value pairs mapping the value of a variable (0,1 or None) to the corresponding index of rows/columns. The final sorted lists of possible rows and columns are found by taking an intersection of a, b for the columns and c, d for the rows. This allows us to 'LEGAL' check each point in the region. We also get the general values of top-left and bottom-right by considering the first and last elements of the lists.

## 3-VAR CASE:




Let us understand the 3-variable case through an example. Let us say we had to find the highlighted region above given ab or [1, 1, None]. Observe that it would be same as finding the region for a 4-variable kmap with term abc' or [1, 1, 0, None]. In other words, 'None' was transferred from 'c' to 'd'. 'c' becomes '0' which is what removes the 'U' cases. Once we have done this transfer, the input can be handled similar to the 4-variable case.

## 2-VAR CASE:




Similar to the 3-variables case, we can clearly observe through example. Let us say the given term is a or [1, None]. It would be same as finding the region for a 4-variable kmap with term a'bc' or [0, 1, 0, None]. In other words, '1' was transferred from 'a' to 'b' and 'None' was transferred from 'c' to 'd'. After this 'a' and 'c' were made '0' to remove the 'U' cases. After this, it can be handled in the 4-variable case.

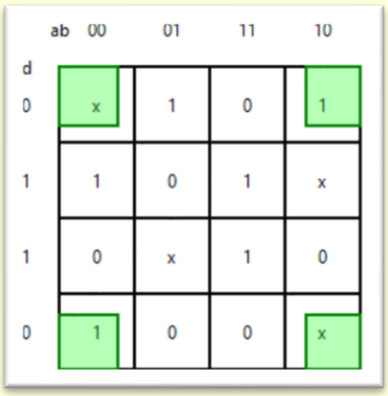
## BORDER CASES:

Certain border cases arise in the 3 and 4 variable kmaps. In the 3-variable case, it occurs when the value of 'b' becomes '0' as the region crosses over the column border. In the 4-variable case, this not only happens as earlier but also when 'd' is '0' because the row border gets crossed. Further, for 4-variables if both 'b' and 'd' are '0' then the corners of the kmap form a region. We introduce certain checks in the program to take care of these cases.

## TESTCASES:

1. kmap_function = [['x', 1, 0, 1], [1, 0, 1, 'x'], [0, 'x', 1, 0], [1, 'x', 0, x]]

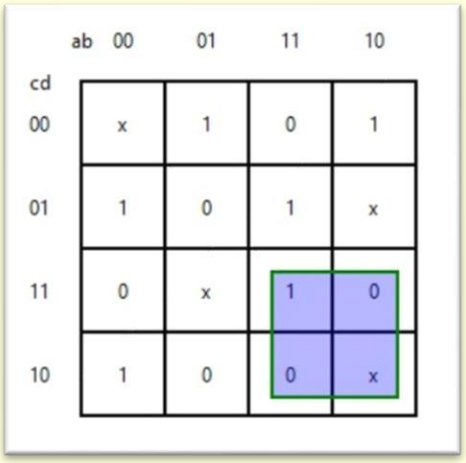
a. Term = [None, 0, None, 0]




For this 4*4 input, column list is obtained by intersection of acdict[None](={0,1,2,3}) and bddict[0](={0,3}), which results in {0,3} and ultimately [0,3]. Similarly, the row list is [0,3]. In general, we assign top-left and bottom-right to be min and max of co-ordinates obtained from row and column lists respectively. As term[1]=0, this is a special case in which the region is partitioned by the vertical side border of the square, so here row index of top-left will be max and of bottom-right will be min. Similarly, as term[3]=0, the region is partitioned by the horizontal side border of the square. So, top-left will be assigned to (3,3) and bottom-right (0,0). Now, checking all elements in the region with loop results in the Boolean value being true.

b. Term = [1,None,1,None]

Here, column list will be [2,3](union of {2,3} and {0,1,2,3}) and row list will be [2,3] as before. Here, term[1] and term[3] are both not zero. So, top-left coordinates=(2,2) and bottom-right =(3,3),which are min and max respectively. As, term[2][3] of region is zero, term is illegal. So, Boolean results to be false.




2. kmap_function = [['x',1,0,1][1,'x',0,1]]

a. Term = [None,0,1]

Here, for the sake of converting it to a 4*4 kmap, we append 0 and swap with input for c, which results in [None,0,0,1]. Applying 4*4 logic, we get column list to be [0,3] and row list=[1]. At first, top-left and bottom-right will be assigned to (1,0) and (1,3). Here, as term[1]=0, this belongs to special case of vertical border crossing, so column index of top-left and bottom-right will be max and min respectively. Hence, top-left and bottom-right will be (1,3) and (1,0) respectively and Boolean turns out to be True.