Indian Institute of Technology Delhi

COL 215 - Digital Logic and System Design

Assignment 1

Check the legality of a region in Karnaugh Maps

Aryan Sharma, Garv Nagori
2021CS10553, 2021CS10549

Contents

1 Introduction

The objective of this Assignment was to check the legality of a group given in Karnuagh Map. Given a term in the K Map we aim to highlight the corresponding region and report whether the region is LEGAL. A legal region consists of only 1s and x's, but no 0s.

2 Algorithm

2.1 Definitions

In this section we define some useful terms which helps to develop and implement our code and helps us to generalise things for 2,3 and 4 variable Karnaugh Maps.

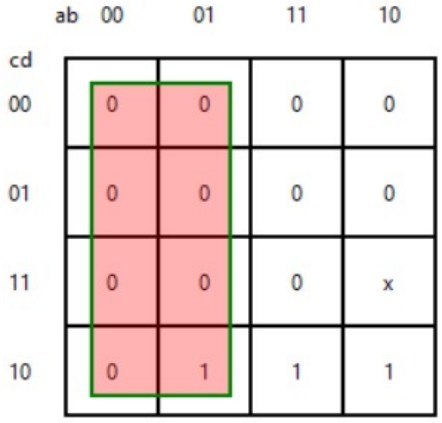
1. **ROW TERMS AND COLUMN TERMS**
   Row terms are the variable terms which dictate what the valid rows will be, similarly for valid columns.
   For a 2 variable K-map the variables are "a,b".
   The row terms will have only "b" as a variable and column terms will have "a" as the variable.
   For a 3 variable K-map the variables are "a,b,c".
   The row terms will have only "c" as a variable and column terms will have "a,b" as the variables.
   Similarly for 4 variable K-map,
   Row terms will be "c,d"
   Column terms will be "a,b"

2. **VALID ROW-COLUMN REPRESENTATION**
   We can completely express a region in terms of the region's constituent rows and columns
   In figure 1(a), the region highlighted below,the region can be completely represented without losing any information about the region by saying that the valid rows are {0,1,2,3} and valid columns are {0,1}.

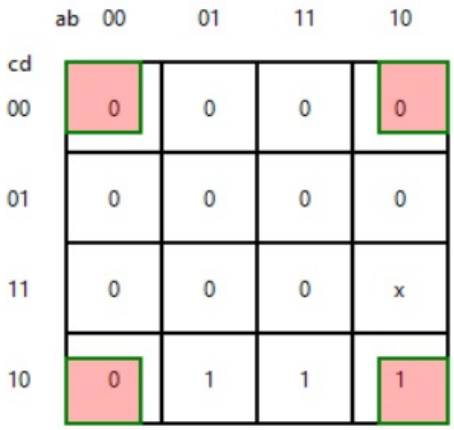
Similarly in Figure 1(b), in this case valid rows will be {3,0} and valid columns will be {3,0}
In this case, it is to be noted that we don't represent valid rows as {0,3} but instead as {3,0}, the convention that we will use is that we want to represent the valid row or valid column list in such a way that the following property holds,

$$list[i+1] = (list[i]+1)\ modulo\ 4\ \forall\ i \tag{1}$$

The region will be the cartesian product of valid row and valid column lists(or sets)



(a) Valid Rows = {0,1,2,3}
Valid Columns = {0,1}



(b) Valid Rows = {0,3}
Valid Columns = {0,3}

Figure 1: Two Regions highlighted in K Map

2.2 THE IDEA

We can visualise the solution to the given problem in the following steps

1. Breaking the list "terms" into lists "row terms" and "column terms".

2. Finding the valid rows corresponding to "row terms" and valid columns corresponding to "column terms"

3. Converting valid rows and valid columns representation to top-left co-ordinate and bottom-right co-ordinate representation of the region.

4. Iterating through the region using valid row and valid column representation.

2.3 ROW-COLUMN TERM

To break the input list "term" we simply used the split function in python to split about the point (len(terms) - 1)/2
The python code of the same is written below

```
slice = int((len(terms)-1)/2)
column = term[:slice+1]
row = term[slice+1:]
```

In the case of 3-variable K-map,
in the first line of the program,
len(terms) = 3
slice = int((3-1)/2) = 1
thus, columns = terms[:2], this will take the elements indexed 0 and 1, i.e. values corresponding to "a" and "b"
rows = terms[2:], this will take the element indexed 2 i.e the value corresponding to "c"

2.4 VALID ROW-COLUMN REPRESENTATION

We elaborate on the procedure to convert row term into valid row(and the same for column) here.
For this task we can use one of the best data structure in python - Set
We discuss everything in this section in terms of row terms but the same is valid for column terms too.
We divide this into two cases,

1. When length of row terms is 2

We define a universal set and a list of list of sets named gawd.
The python code for the same is written below

```
universal = {0,1,2,3}
gawd = [[{0,1},{2,3}],[{3,0},{1,2}]]
```

As the name universal suggests, this set represents valid rows when we don't have any prior information about the contents of row terms

The list gawd is made in the following way,

The term gawd[i][j] represents the valid rows for value j of i^{th} term in the row terms list

For example in the case row terms = [0,None]

This means the valid row will be, 0th term is "c", thus gawd[0][0] would represent valid rows for c having value 0, which can be seen from the K-map(as shown in Figure 2) are {0,1}

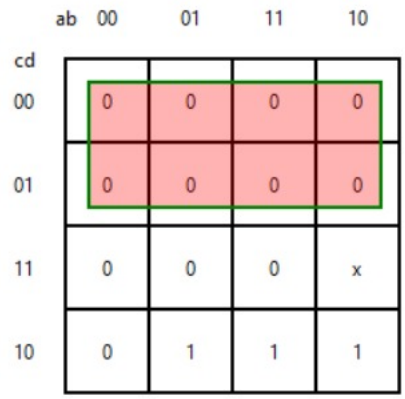


Figure 2: gawd[0][0] corresponds to valid rows = {0,1}

We declare a set = universal

Now we iterate through the list row terms and if row term[i] is not equal to None then that term does not affect our valid rows set.

If we get value j (0 or 1) for the i^{th} term the valid rows corresponding to that are gawd[i][j], thus we take intersection of set with gawd[i][term[i]] to only include rows which are valid for the i^{th} term.

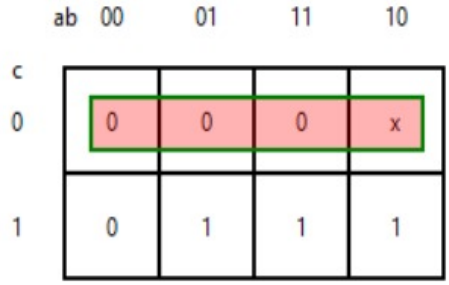
Thus at the end of this iteration, the set has all the valid rows for all the variables in the term.

The data structure set in python has the property that it is unordered, but as it has been noted before that if the valid rows are {0,3} then we want the list of valid rows to be [3,0] so that it obeys property (1)

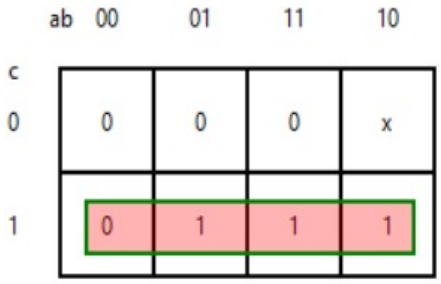
2. **When length of row terms is equal to 1**
Since we only need to check for 1 value, we use simple logic statements to perform this task.
The possible cases are that row term can be either None,0 or 1.

If row term is None, then the valid rows are {0,1}
If row term is 1, then the valid rows are {1}
If row term is 0, then the valid rows are {0}

The above is shown clearly in the Figure below.



(a) Valid Rows = {0}



(b) Valid Rows = {1}
Fully Highlighted K-Map

Figure 3: Two Regions highlighted in K Map

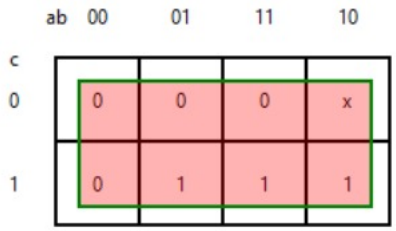


Figure 4: Valid rows = {0,1}

The python code for above algorithm is shown below, it has been implemented as a function valid.

```
def valid(term):
    universal = {0,1,2,3}
    gawd = [[{0,1},{2,3}],[{3,0},{1,2}]]
```

```
set = universal
if(len(term) == 2) :
        for i in range(len(term)):
                if (term[i] != None):
                        set =set.intersection(gawd[i][term[i]])
        if(set == 0,3):
                return [3,0]
        return list(set)
else:
        if(term[0] == None):
                set = {0,1}
        else:
                set = {term[0]}
        return list(set)
```

2.5 CONVERSION

To convert valid row and valid column representation to top_left co-ordinate and bottom_right co-ordinate representation, we can observe that the first element of valid row and valid column gives the top_left co-ordinate and the last elements of valid row and valid column gives the bottom_right co-ordinate.The significance of reversing the set {0,3} to [3,0] list in our program becomes clear here, as we need to output the co-ordinates of top left and bottom right we can easily extract the co-ordinates, also property (1) is significant because in a Karnaugh map on moving from co-ordinate [i,j] to $[(i+1)\ modulo\ 4\, j]$ or $[i, (j+1)\ modulo\ 4]$ only one bit changes. The Python Code is as follows:

```
top_left = (valid_row[0],valid_column[0])
bottom_right = (valid_row[len(row)-1],valid_column[len(column)-1])
```

2.6 ITERATION

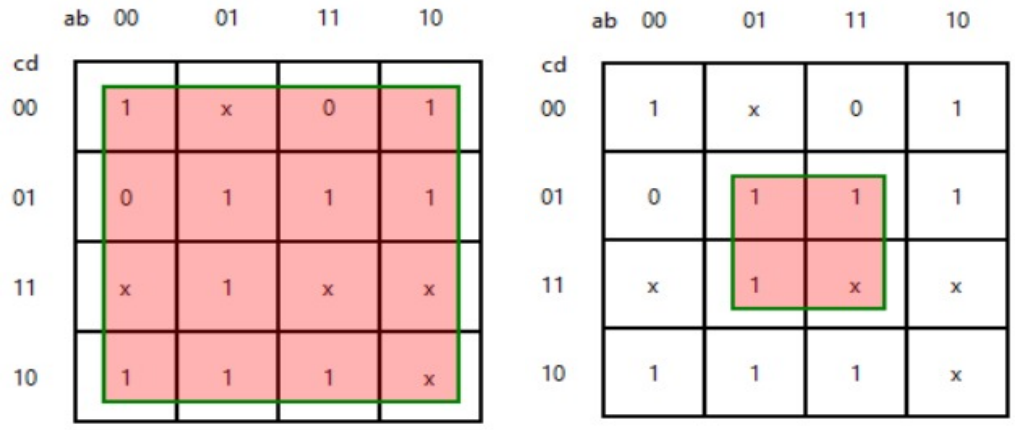
For checking the legality of a region we iterate through each square in the region and check if the region has a zero, if it does then we return (top_left,bottom_right,False) if the loop completes then we return (top_left,bottom_right,True), we iterate through the lists valid_row and

valid_column in a nested loop, thus covering each element of the cartesian product between valid_row and valid_column.
The python code for above is:

```
for i in valid_row:
        for j in valid_column:
            if(kmap_function[i][j] == 0):
                return (top_left,bottom_right,False)
return (top_left,bottom_right,True)
```

3 Test Cases

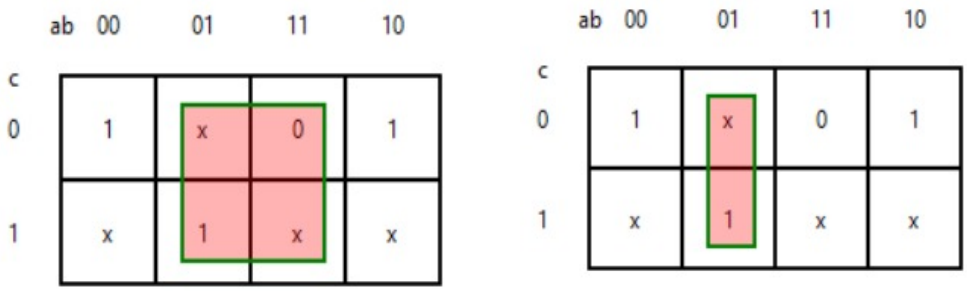
1. **GENERIC CASES**
   After implementing our algorithm in Python and dry running through some preliminary checks, we decided to check against some general cases of K Maps with 2,3 and 4 variables.



(a) Input:term=[None,None,None,None]
Output: ((0,0),(3,3),False)

(b) Input: term=[None,1,None,1]
Output: ((1,1),(2,2),True)

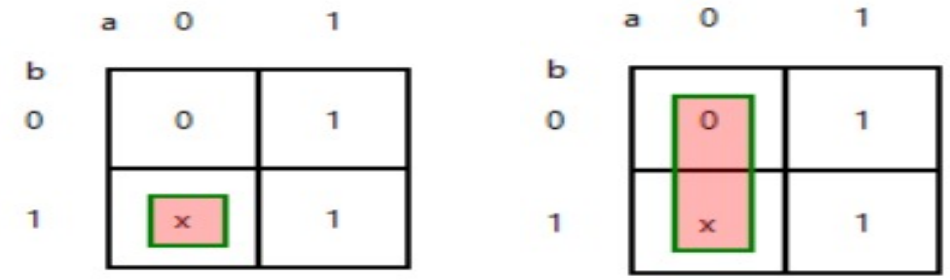
Figure 5: K_map = [[1,'x',0,1],[0,1,1,1],['x',1,'x','x'],[1,1,1,'x']]
General Test Cases for 4 Variable K Maps



(a) Input: term = [None,1,None]
Output: ((0,1),(1,2),False)

(b) Input: term = [0,1,None]
Output: ((0,1),(1,1),True)

Figure 6: K_map = [[1,'x',0,1],['x',1,'x','x']]
General Test Cases for 3 Variable K Maps

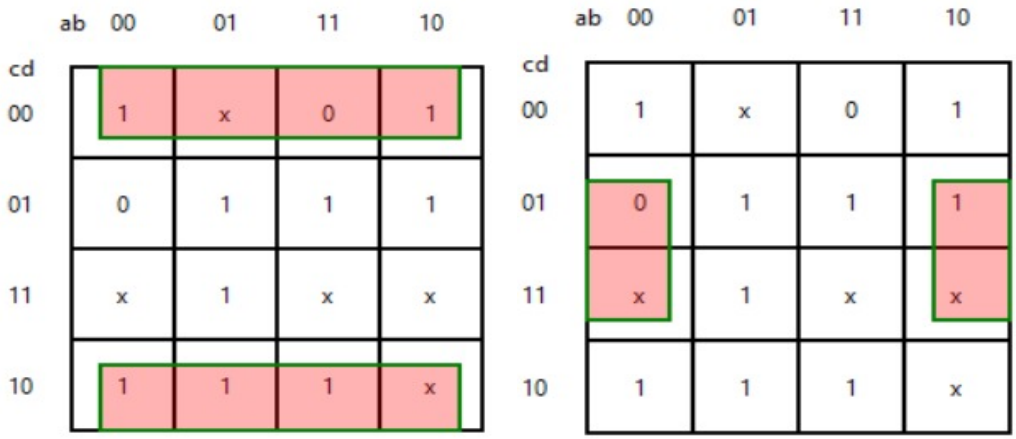


(a) Input: term = [0,1]
Output: ((1,0),(1,0),True)

(b) Input: term = [0,None]
Output: ((0,1),(1,1),False)

Figure 7: K_map = [[0,1],['x',1]]
General Test Cases for 2 Variable K Maps

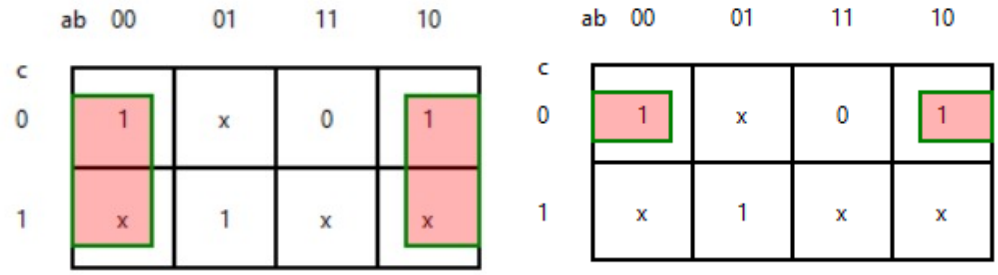
2. **EDGE CASES**
When our program passed the general test cases for all types of K Maps, we decided to test it against some edge cases. The 2 variable K Maps do not have any edge cases so we only checked the 3 variable and 4 variable edge cases.



(a) Input: term = [None,None,None,0]
Output: ((3,0),(0,3),False)

(b) Input: term = [None,0,None,1]
Output: ((1,3),(2,0),False)

Figure 8: K_map = [[1,'x',0,1],[0,1,1,1],['x',1,'x','x'],[1,1,1,'x']]
Edge Test Cases for 4 Variable K Maps

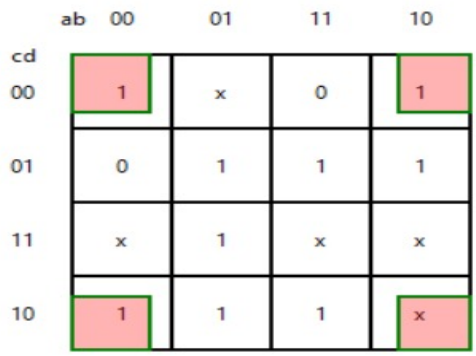


(a) Input: term = [None,0,None]
Output: ((0,3),(1,0),True)

(b) Input: term = [None,0,0]
Output: ((0,3),(0,0),True)

Figure 9: K_map = [[1,'x',0,1],['x',1,'x','x']]]
Edge Test Cases for 3 Variable K Maps

The most annoying test case to solve was the "Four Corners" of the 4 variable Karnaugh Map since not one of the cells in the group touch each other but instead are in the opposite corners of the map.



(a) Input: term = [None,0,None,0]]
Output: ((3,3),(0,0),True)

Figure 10: K_map = [[1,'x',0,1],[0,1,1,1],['x',1,'x','x'],[1,1,1,'x']]
Most annoying test case

4 Conclusion

In this Assignment we used set data structure in python to locate regions representing a boolean expression in a a Karnaugh map, we also checked whether the given term represents a valid region in the K-map, finally we ran some test-cases to check the functioning of the program. The program functions correctly on the tried test-cases.