COL215P Assignment 1 - SW

Divyansh Mittal - 2020CS10342
Avval Amil - 2020CS10330

October 15, 2022

1 Working

Nothing in the assignment is hardcoded. The way our algorithm works is as follows:

1. Construct all terms involved from the term provided: The algorithm creates a list of all the min-terms that are in the region. These are a string of 0s and 1s. It does that by recursively appending 0 or 1 to the elements of list, if the variable is 0 or 1 and both 0 and 1 if the variable is None. Eg a term a'd gets expanded as ['0001', '0011', '0101', '0111']

2. Next we construct the gray code list for the column size($\lceil \frac{n}{2} \rceil$) and row size($\lfloor \frac{n}{2} \rfloor$). This too is done recursively. The *get_grey_code(size)* returns empty string list if size = 0, else it recursively first gets a list for size-1 and prepends 0 to the list for size-1 and 1 to the reverse of list for size-1 and concatenate the 2 and returns.

3. For all the terms in the list generated in point(1), we get the index of row and column element by checking the index of first $\lceil \frac{n}{2} \rceil$ characters in the column gray list and rest of the characters in the row column index.

4. If the element in the K-Map at these indices is 0, we know this is not a legal term. Further we keep a set of all the row and column indices.

5. Sort all the column and row indices. If these form a continuous sequence like [0,1,2,3], we know there is no wrap around, but if it is of the form [0,3], then it implies a wrap-around is involved and the first row or first column index needs to be swapped.

6. Finally we return the 2 tuple coordinates, and the legality of the term.

2 PseudoCode

Algorithm 1 PsueduoCode

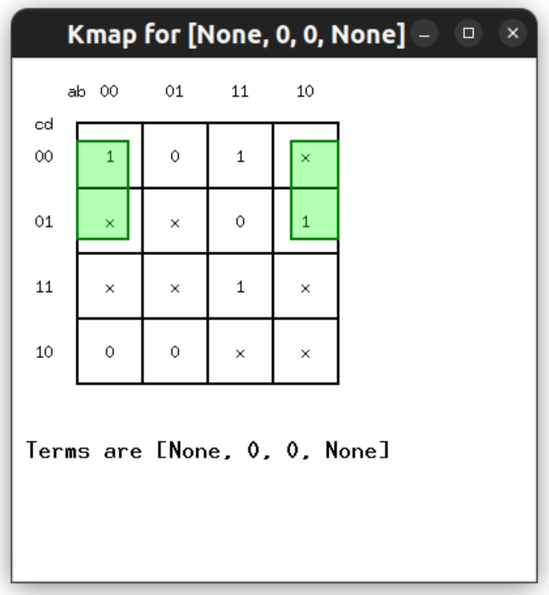
```
procedure IS_LEGAL_REGION($KMap$,$term$)
    allTerms ← $expand(term)$        ▷ Gets all minterms whose sum is term
    colGreyCode ← $greyCode(ceil(n/2))$        ▷ Gets greycode list for column
    rowGreyCode ← $greyCode(floor(n/2))$        ▷ Gets greycode list for row
    isLegal ← $True$
    for mterm in allTerms do
        Get column and row index using colGreyCode and rowGreyCode
        Check if Kmap has a 0 at the index and if yes isLegal = False
        Add row index and col index to 2 sets
    end for
    Sort all the row indices and the column indices
    If row indices contiguous, no wrap around, else rowWrapAround = True
    If col indices contiguous, no wrap around, else colWrapAround = True
    Choose first and last row index, based on rowWrapAround
    Choose first and last col index, based on colWrapAround
    return First-Cordinate, Second-Cordinate,isLegal
end procedure
procedure GREYCODE($size$)
    if $size$ is 0 then
        return ['']
    else
        $greyCodeSmall \leftarrow greyCode(size - 1)$
        $greyCodeList \leftarrow ['0' + term$ for term in $greyCodeSmall] + ['1' +$
$term$ for term in $reverse(greyCodeSmall)]$
        return greyCodeList
    end if
end procedure
```

3 Testing

A testing.py file helped generate random K-Maps of various sizes and different terms to test the program thoroughly. The output of our function was verified using the gui file provided. The 2 coordinates that our function returns create the region with the help of gui and coloring it green to indicate returned value is true and red region to indicate returned false.

To run these tests, use the slightly tweaked gui file in the submission.[1]

[1] In the submission .py is removed, so that autograder works properly

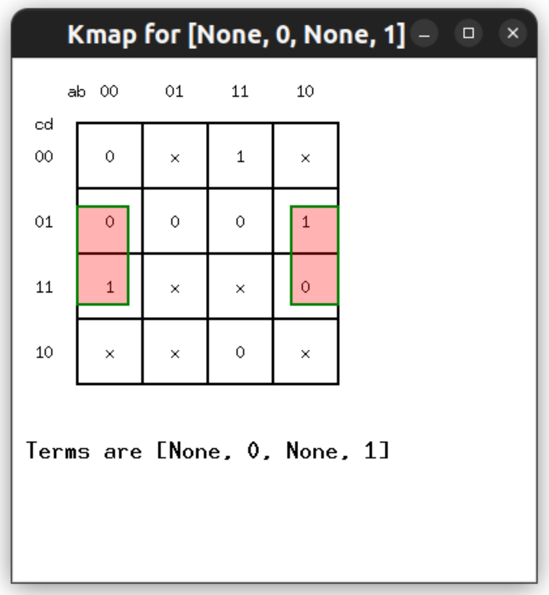


Kmap for [None, 0, 0, None]

ab	00	01	11	10
cd				
00	1	0	1	×
01	×	×	0	1
11	×	×	1	×
10	0	0	×	×

```
Terms are [None, 0, 0, None]
```

Figure 1: KMap for b'c'

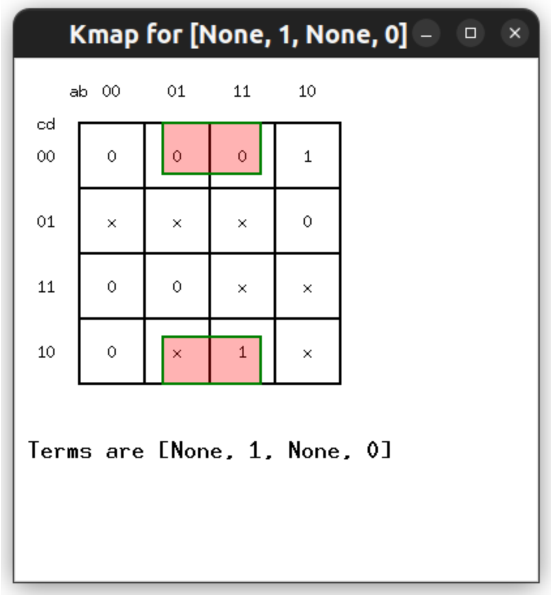


Kmap for [None, 0, None, 1]

ab	00	01	11	10
cd				
00	0	×	1	×
01	0	0	0	1
11	1	×	×	0
10	×	×	0	×

```
Terms are [None, 0, None, 1]
```

Figure 2: KMap for b'd

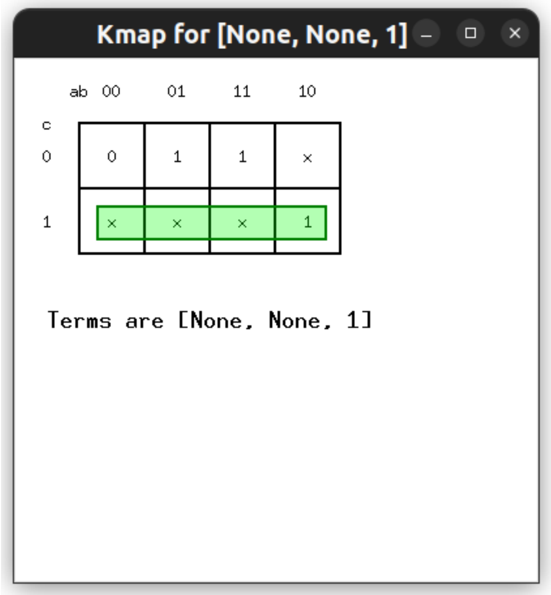


Kmap for [None, 1, None, 0]

cd \ ab	00	01	11	10
00	0	0	0	1
01	×	×	×	0
11	0	0	×	×
10	0	×	1	×

```
Terms are [None, 1, None, 0]
```

Figure 3: KMap for bd’

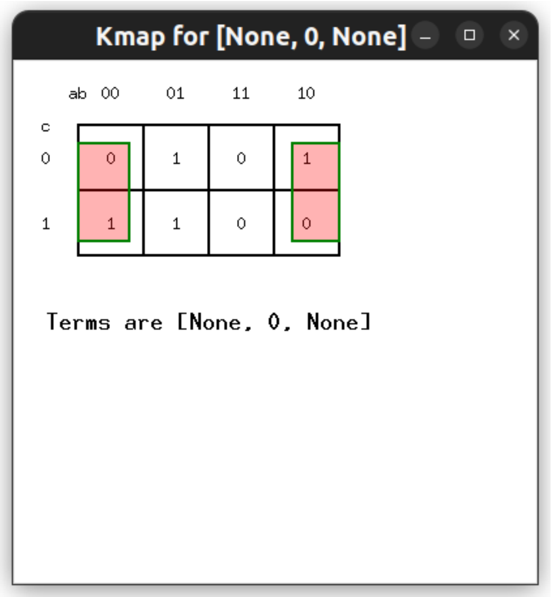


Kmap for [None, None, 1]

c \ ab	00	01	11	10
0	0	1	1	×
1	×	×	×	1

```
Terms are [None, None, 1]
```

Figure 4: KMap for c

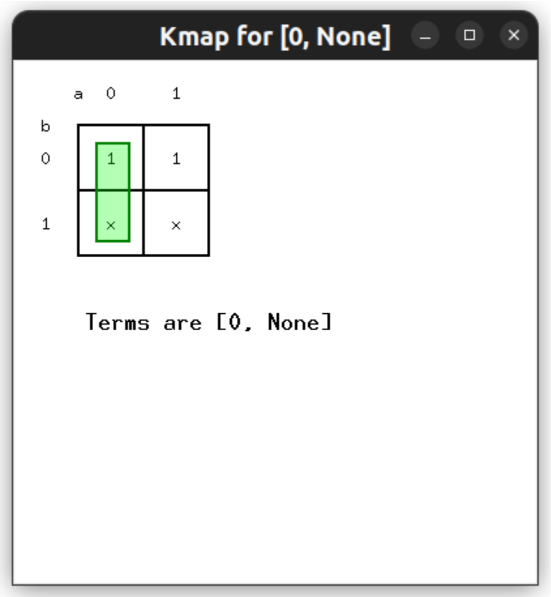


Kmap for [None, 0, None]

ab c	00	01	11	10
0	0	1	0	1
1	1	1	0	0

```
Terms are [None, 0, None]
```

Figure 5: KMap for b'

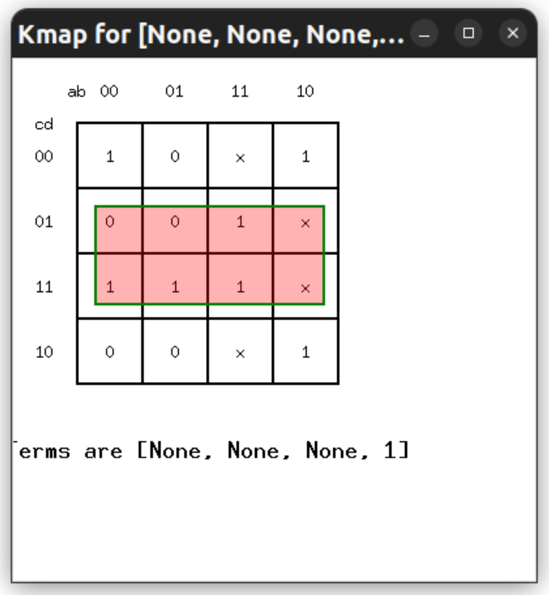


Kmap for [0, None]

a b	0	1
0	1	1
1	×	×

```
Terms are [0, None]
```

Figure 6: KMap for a'

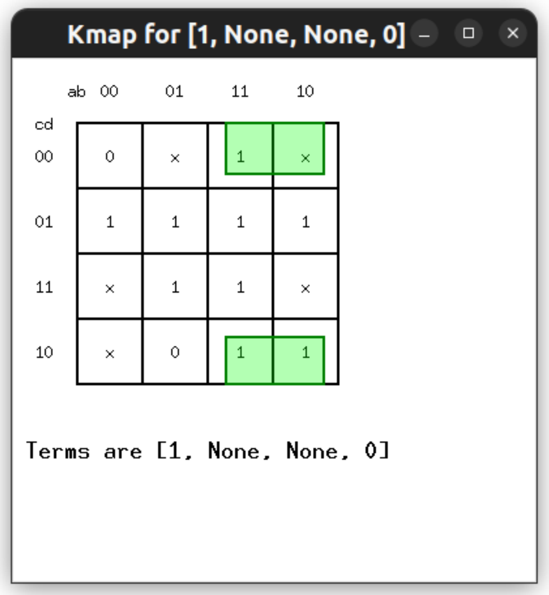


Kmap for [None, None, None,...

ab cd	00	01	11	10
00	1	0	×	1
01	0	0	1	×
11	1	1	1	×
10	0	0	×	1

erms are [None, None, None, 1]

Figure 7: KMap for d

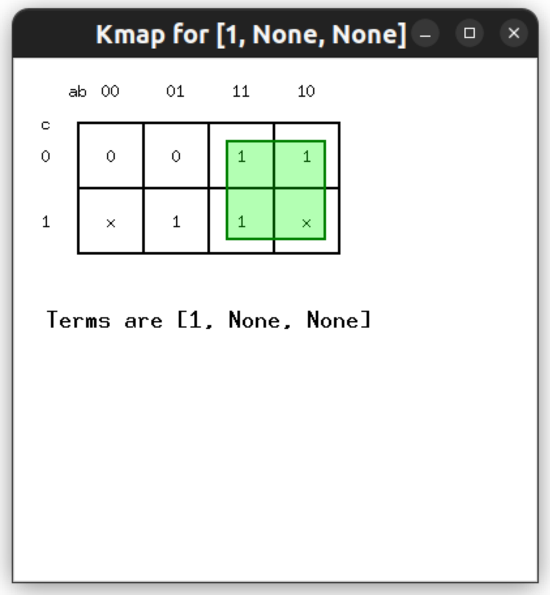


Kmap for [1, None, None, 0]

ab cd	00	01	11	10
00	0	×	1	×
01	1	1	1	1
11	×	1	1	×
10	×	0	1	1

Terms are [1, None, None, 0]

Figure 8: KMap for ad’



c ab	00	01	11	10
0	0	0	1	1
1	×	1	1	×

```
Terms are [1, None, None]
```

Figure 9: KMap for a