COL 215 Assignment 2

Parth Thakur, 2021CS50615
Amish Kansal, 2021CS50622

Approach:

For expanding terms, the logic is that to drop one variable, i.e, double the size of the region in the k-map, you need to have terms that contain both the variable and its complement as 1s.
For example:
We can reduce ab’ and ab to a by dropping the variable b.

This logic is repeated recursively to facilitate maximally expanding the terms with this logic until no further expansion is possible.

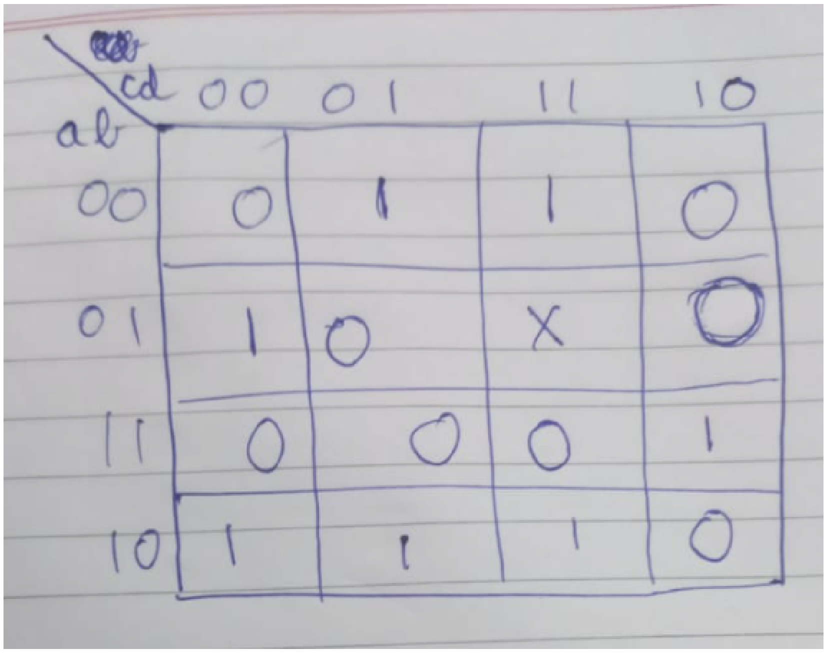
Here is a detailed explanation of what the code does:

- Find the number of variables we have to deal with
- Using the result from the previous step, construct a dictionary mapping the alphabets to indices, i.e., map a, b, c… to 0, 1, 2…
- Convert the func_TRUE and func_DC list of terms to boolean representation list of lists. E.g., “ab’c” is converted to [1, 0, 1]
- Expand the boolean list maximally:
    - Reduce one variable using func_TRUE and func_DC
    - Recursively keep reducing one term until no more variable can be reduced and add this result to the front of the list that the function will return.
    - Store these expanded terms in a list
    - By construction, the list is such that it is sorted in decreasing order of the number of terms present in the list.
    - Return this list
- Reduce each term of the boolean list corresponding to func_TRUE, taking elements from the list returned in the last step
    - For each term in func_TRUE
        - If an expansion of this term in present in the list of expanded regions, append it to the result list

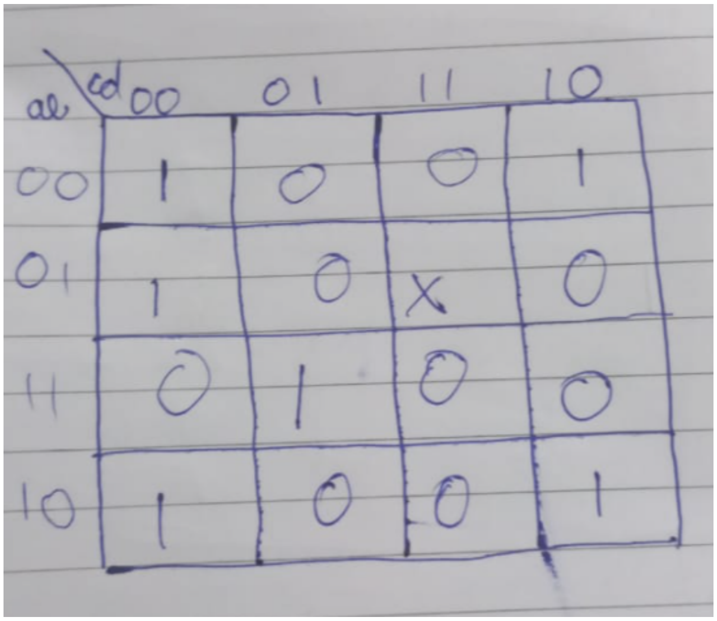
        - Otherwise, append the original term in the result list
    - Return the result list
- Convert the boolean terms list into variable terms list and return it

Test Cases:

Each Test Case output is verified with the help of K-map analysis.

- Test Case 1 (sample case 1 from assignment description)
    - Input:
        - func_TRUE = ["a'bc'd'", "abc'd'", "a'b'c'd", "a'bc'd", "a'b'cd"]
        - func_DC = ["abc'd"]
    - Output:
        - ["bc'", "bc'", "a'c'd", "bc'", "a'b'd"]
- Test Case 2
    - Input:
        - func_TRUE = ["a'b'c'd","a'b'cd","a'bc'd'","abcd'","ab'c'd'","ab'c'd","ab'cd"]
        - func_DC = ["a'bcd"]
    - Output:
        - ["b'd", "b'd", "a'bc'd'", "abcd'", "ab'c'", "b'd", "b'd"]
    - 

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

- Test Case 3
  - Input:
    - func_TRUE = ["a'b'c'd'","a'b'cd'","a'bc'd'","abc'd","ab'c'd'","ab'cd'"]
    - func_DC = ["a'bcd"]
  - Output:
    - ["b'd'", "b'd'", "a'c'd'", "abc'd", "b'd'", "b'd'"]
  - 

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

Some other cases tested:

- func_TRUE = ['abcdef'] func_DC = ["abcdef'"] Output: ['abcde']
- func_TRUE = ["a'b'cde", "a'b'c'd'e", "a'b'c'de", "a'bcde"] func_DC = ["abcd'e", 'abcde', "abcd'e'"] Output: ["a'cde", "a'b'c'e", "a'b'de", "a'cde"]
- func_TRUE = ["abcd'e", 'abcde', "abcd'e'"] func_DC = [] Output: ['abce', 'abce', "abcd'"]
- func_TRUE = ["a'bc'd'e", "a'bc'de", "a'bcde"] func_DC = ['abcde', "ab'c'de", "ab'cde"] Output: ["a'bc'e", "a'bc'e", "a'bde"]
- func_TRUE = ["ab'c'd'e'", "ab'c'de", 'abcde', "ab'c'd'e", "ab'cde"] func_DC = ["ab'cd'e'", "ab'cd'e"] Output: ["ab'd'", "ab'e", 'acde', "ab'd'", "ab'e"]
- func_TRUE = ["a'bcd"] func_DC = ["ab'c'd'", 'abcd', "ab'c'd", "ab'cd"] Output: ['bcd']
- func_TRUE = ["a'bcd", "a'b'cd'", "a'b'cd"] func_DC = ["ab'c'd'", 'abcd', "ab'c'd", "ab'cd"] Output: ['cd', "a'b'c", 'cd']