COL215 Assignment 2 Report

In the assignment, we were asked to define a function which takes two arguments as input, 'func_True ' (list of cells in K-map which contain 1) and 'func_DC' (list of cells in K-map which contain 'x' and returns the maximum region to which every term in func_True can be expanded.

Approach:

Two terms of K-map can be combined if they vary in a single literal and have all the remaining literals same. For eg : Consider terms abcd and abcd' , they both contain the same literals for a,b and c but vary in d.

In order to find such strings we defined a function str_num which takes two binary strings as input, compares literal at each index of both the strings. If the literal is same '0' is added to string s and if the literal varies then '1' is added to srtring s. Finally, string s is returned. Now, if the such a string s contains a single '1' in it means that the two strings vary in a single literal.

While accessing the elements of the input strings using string indexing, the literal and " ' " indicating complement were considered to be separate terms instead of a single term " a' ". To solve this problem, we defined a function dic which takes a string as input and converts all the terms of the form complement of literal e.g." a' " to the corresponding uppercase literal e.g. 'A' as mapped in the dictionary.

The next function comb_function takes two arguments func_True and func_DC . It forms a list bny_comb containing binary form of all strings in both the arguments. The combined list of arguments 'comb' is updated according to the dictionary. Next, we traverse along the list bny_comb using two for loops and find str-num of every element of comb with every other element occurring after it. If the return value of str_num contains a single '1', we append the corresponding string in comb by replacing the literal to be dropped by '*', in the list new_in. To keep a track of elements whose str_num is being appended in new_in, we append these elements in list sec. At the end of inner loop, if there is a element which isn't appended in sec, we directly append it in new_in.

Next we have function recur_sol which works similar to comb_function. It takes input a list similar to new_in (during first call it takes input as new_in). Then it simplifies new_in by dropping the possible literals everytime it is recursively called. It is called until the input argument and the output argument are equal indicating that the list can't be simplified further. At this instant, it returns the simplified list new_out.

Next, we traverse along the list new_out and append the strings without asterisk into the list 'list_out'.

The next function min_str takes two strings s1 and s2 and checks if the elements of s2 are contained in s1 and returns a Boolean value.

Next, we traverse the list func_True and it contains almost all the possible expanded regions corresponding to each term so we append the maximum region(smallest literal containing string) into a list 'list_k'.

Next we have function final_str which converts the literals in capital letters back to a' ,b' i.e the corresponding original literal in all the terms of list_k.

So we finally have our final_ans list containing the final expanded regions corresponding to every term given in func_True.

Testcases we have run:

```
#----------------------------------------TEST CASES FOR 2
VARIABLES-----------------------------------------------------
# print(comb_function_expansion(["a'b'"],[]))
# print(comb_function_expansion(["a'b'"],["a'b"]))
# print(comb_function_expansion(["a'b'","ab'"],["a'b"]))
# print(comb_function_expansion(["a'b","a'b'"],[]))
#ALL ARE ONE
# print(comb_function_expansion(["a'b'","a'b","ab'","ab"],[]))
#----------------------------------------TEST CASES FOR 3
VARIABLES-----------------------------------------------------------
# print(comb_function_expansion (["a'b'c'","a'b'c","a'bc","a'bc'"],["ab'c'","ab'c"]))
# print(comb_function_expansion(["a'b'c'","a'b'c"],["ab'c'","ab'c"]))
#----------------------------------------TEST CASES FOR 4
VARIABLES-----------------------------------------------------------
# print(comb_function_expansion( ["a'bc'd'", "abc'd'", "a'b'c'd", "a'bc'd", "a'b'cd"], ["abc'd"]))
#----------------------------------------TEST CASES FOR 5
VARIABLES-----------------------------------------------------------
# print(comb_function_expansion(["a'b'c'd'e'", "a'b'cd'e", "a'b'cde'", "a'bc'd'e'","a'bc'd'e", "a'bc'de",
"a'bc'de'", "ab'c'd'e'", "ab'cd'e'"],["abc'd'e'", "abc'd'e", "abc'de", "abc'de'"]))
#----------------------------------------TEST CASES FOR 6
VARIABLES-----------------------------------------------------------
# print(comb_function_expansion(["a'b'c'd'e'f'","a'b'c'd'e'f","a'b'cd'e'f'","a'b'cd'e'f","a'b'cd'ef'",
"a'b'cd'ef","a'b'cde'f'","a'b'cde'f","a'b'cdef'","a'b'cdef","a'bc'd'e'f'","a'bcd'e'f'","ab'c'd'ef","abc'de'f'",
"abc'de'f","abc'def'","abc'def"],["abcde'f'","abcde'f","abcdef'","abcdef"]))
```

Q1) Do all expansions result in identical set of terms?

Ans) No, all the expansions do not contain identical set of terms and the set of terms may vary because the way in which a region surrounding a term is expanded depends on which cells adjacent to the given term contain 1 or x.

Q2) Are all expansions equally good, assuming that our objective to maximally expand each term? Explain.

Ans) Yes, the final list returned contains the most expanded region possible for the corresponding term in func_True. The intermediate results may not be equally good but the final result is.

Made By : Disha Shiraskar and Srushti Junghare