COL215 - Software Assignment 2- REPORT

The code first converts the two inputs ‘ func\_True’ and ‘func\_DC’ into a list of integers from the list of terms, where terms are represented in alphabetical form of gray code using the auxiliary function *list\_conv\_int(list\_str).* This function further uses the auxiliary function *convert\_to\_gray(str)* to convert alphabetical form to its corresponding gray code form.

Then we combine the two new lists into one *‘minterms’*. and start grouping them. Initially we group based on the no. of 1s present in the gray code. For example, if minterms=[“00”,“001,“11”,“10”]

| Group | 0 | 1 | 2 |
| --- | --- | --- | --- |
| Terms | 00 | 01,10 | 11 |

Then we go on to group the terms of a group with the terms of its corresponding next group since grouping can only be done if two terms differ by 1 bit only. We use *compare(a,b)* function which checks for that 1 bit difference and if true returns the index at which the bit differs. We combine the terms and try to assign them to correct groups. In this manner we combine terms representing one element to form a term representing 2 elements, terms representing 2 elements to form a term representing 4 elements, terms representing 4 elements to form a term representing 8 elements and so on. If a term of any size can’t be further expanded it goes to *‘not\_common’.* When no further terms can be expanded the grouping ends and we are left with the set *‘mini\_terms*’ consisting of terms which can’t be further expanded. Here, the set property helps us ensure that no term is repeated. We convert this set into a list ‘*t*’ and use *sorter(li)* function which arranges the term representing regions from the largest to smallest. For the region/term having the same no. of elements, arrangement is done based on increasing order of gray code form.

The terms in the list ‘*t*’ are then converted into alphabetical form of grey code by using *convert\_to\_alpha(st)* on the elements individually.

Now we iterate over *‘func\_True’*, to find the region in the list *‘t*’ in which the term from *‘func\_True’* is contained .Once found, we append that region into ‘*final\_list*’ . We use the ‘*is\_inside(in1,in2)*’ function to check if the 1 lies in that region. This function uses the auxiliary function ‘*splitter\_base(k)’ which* takes the combined string (alphabetical form of gray code) and splits it into its corresponding parts.

Eg. a’b’cde to [a’,b’,c,d,e]

We split *in1 and in2* and check for common elements. If the set of common elements is equal to the smaller set (larger region) then we return the larger set to our main function. If the two sets have the same size then we can return any. If the region returned is not *None* then we append the unsplit form of the smaller set and *None* otherwise into ‘*final\_list*’ . Once the iteration ends we are left with ‘*final\_list*’ which consists of maximally expanded terms in the form of boolean literals.

Hence, this is how the main function comb\_function\_expansion(func\_True,func\_DC) gives the desired result.

• Do all expansions result in an identical set of terms?

-No, all expansions don’t result in identical lists of terms. Because if we have an input which can be expanded in two(or more) regions such that the two regions are of equal dimension. Then, expansions set can include any of those region for that term.

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

-Yes, since all expansions are maximally done. Hence, all expansions are equally good.

**Test Cases:**

We ran our code on test cases for K-map of 2\*2, 2\*4, 4\*4, 4\*8 and 8\*8. A few of them are:

* INPUT:

func\_True =["a'bcd","abc'd","abcd","abcd'","ab'cd"]

func\_DC =[]

OUTPUT = ['bcd', 'abd', 'acd', 'abc', 'acd'] (Correct)

* INPUT:

func\_True =["a'b'","a'b"]

func\_DC =["ab'","ab"]

OUTPUT = [None,None] (Correct)

* INPUT:

func\_True =["a'b'c'de'","a'b'cd'e'","a'bc'd'e","a'bc'de","a'bc'de'","ab'c'de'"]

func\_DC =["a'bcde","a'bcde'"]

OUTPUT = ["b'c'de'", "a'b'cd'e'", "a'bc'e", "a'bd", "a'bd", "b'c'de'"] (Correct)

* INPUT:

func\_True =["a'b'c'de'","a'b'cd'e'","a'bc'd'e","a'bc'de","a'bc'de'","ab'c'de'"]

func\_DC =["a'bcde","a'bcde'"]

OUTPUT = ["b'c'de'", "a'b'cd'e'", "a'bc'e", "a'bd", "a'bd", "b'c'de'"] (Correct)

-------------------------------------------------------------------------------------------------------

By :

Kanishka Gajbhiye

Entry no:2021CS50131

Piyush Chauhan

Entry no: 2021CS11010