COL215 - Software Assignment 3- REPORT

The main function 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"] then group 0 contains 00, 1 contains 01, 10 and 2 contains 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_term*' consisting of terms which can't be further expanded.

Then we start another iteration to form a dictionary '*all_prime_implicants*' which will store key as the minterm and value as the region it is coming in. For this we use two functions :

- The function '*minterm_composed_of*' to find the minterms which combine to give a region '*merged_minterms*'.
- Another function '*only_ones((my_list,dc_list)*' removes *don't care* minterms from a given list of m*interms* representing a region when combined and returns the list of minterms which have *one*.

Then we use the function '*essential_pi*' to find essential prime implicants from the dictionary '*all_prime_implicants*'. The function '*remove_minterms*' removes the essential prime implicants term from *all_prime_implicants* .

Then we iterate over the modified *all_prime_implicants* . For each iteration we find the term representing the maximum region using the function '*findmax*' and append it in essential prime implicants.

Lastly we use the function '*convert_to_alpha*' to convert the terms in the essential prime implicants list from gray code to alphabetical form and finally return '*essential_prime_implicants*' as the output.

Hence, this is how the main function *opt_function_reduce(func_TRUE,func_DC)* gives the desired result

Time Complexity:

The main function runs with a time complexity O(n^3). The complexity of the helper functions has been stated in the code. We utilise them to analyse the complexity for the main function.

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'b'c'd","a'b'cd","a'bc'd","abc'd'","abc'd","ab'c'd'","ab'cd"]
  func_DC =["a'bc'd'","a'bcd","ab'c'd"]

  OUTPUT = [["b'd", "ac'", "c'd"]] (Correct)
- INPUT:
  func_True
  =["a'b'c'd","a'bc'd","a'bcd","a'bcd'","abc'd'","abc'd","abcd'","a
  b'cd"]
  func_DC =[]

  OUTPUT = [["abc'", "a'bc", "a'c'd", 'acd']] (Correct)
  The maximum region in prime implicants is bd, but we don't see it in output as it is a redundant region.
- INPUT:
  func_True
  =["a'b'cd","a'b'cd'","a'bcd","abc'd'","abc'd'","abc'd","ab'c'd'","ab'c
  'd"]
  func_DC =["a'bc'd'","a'bc'd","a'bcd'"]

  OUTPUT = ["ac'", "a'c"](Correct)
- INPUT:
  func_True
  =["a'b'c'd'e'f'","a'b'cd'e'f'","a'b'cd'e'f","a'b'cde'f'","a'bcde'
  f'","a'bcdef","a'bc'd'e'f'","a'bc'd'ef'","ab'c'd'e'f","ab'c'd'ef"
  ,"ab'c'd'ef'","ab'c'de'f'","ab'c'de'f","ab'c'd'ef","ab'cd'e'f'","
  ab'cde'f'","ab'cde'f","ab'cdef"]

  func_DC=
  ["a'b'c'd'e'f","a'b'c'de'f'","a'b'c'de'f","a'b'cde'f","a'bcd'e'f'
  ","a'bcdef'","a'bc'de'f'","ab'c'd'e'f'","ab'c'def","ab'c'def'","a
  b'cd'e'f","ab'cd'ef","ab'cd'ef'","ab'cdef'","abc'de'f"]

  OUTPUT = ["a'bc'd'f'", "b'e'", "ab'", "a'bcde", "a'e'f'"]
  (Correct)

The diversity and complexity of the test cases make them a good enough set for validation of our function. In the second test case , the maximum region in prime implicants is bd, but we don't see it in output as it is a redundant region. In the third test case, we see a good example of how 'x' is ignored or chosen which leads to output returning two regions which perfectly fit the criteria. In the fourth test case, we observe in the output how the regions are overlapping but maximally expanded and the output has no redundant implicants.

By :
Kanishka Gajbhiye
Entry no:2021CS50131
Piyush Chauhan
Entry no: 2021CS11010