COL215 Assignment 3

Manpreet Singh(2021CS10070) Om Dehlan(2021CS10076)

September 29, 2022

1 Introduction

From Assignment 2, we look at the list of the maximally expanded regions, and delete the terms which can be covered by the other regions. In other words, we have to cover all the '1's in the given Karnaugh map, using minimum number of legal regions. We made some small change to the code in assignment 2 to return the maximally expanded region which has maximum number of '1's included, which we didn't do before as we only had to output any one maximally expanded region.

2 Approach

We get a list `all_1_prev_terms`, which is the list of each expanded term we got from our modified assingment 2. the set `setform` is the set of all these expanded terms to get a non-repeated iteration of these regions. `The augmented_func_TRUE` is a list of tuples (number of regions in which min term occurs, index of the min term, min term) sorted by the number of regions. Now since in `all_1_prev_terms`, we have an expanded region repeated as many times as the number of min terms it corresponds to, we iterate through the `The augmented_func_TRUE` list with 'i' and in the inner loop using 'j', check for all min terms taken by 'j' (not equal to i) that are contained in expanded region corresponding to i, and removes these from further considerations by putting these 'j' in the set `rem`. By the end of these loops we have taken into account all the min terms in `func_TRUE` using minimum expanded regions that were added to the list `ans`. To calculate the time complexity, we see that we used two nested loops, to remove the redundant terms from the output of assignment 2, and we are iterating through the term to check if it is in this region, so this takes $\mathcal{O}(n^2m)$, where n is the number of inputs in the `func_TRUE` and `func_DC` lists, which is a polynomial. To calculate the maximally expanded terms for each term in `func_TRUE`, we looked through the possible expansions of any term and recursively selected the largest possible expansion. Thus checking the subsets of the term takes time $\mathcal{O}(\Sigma(\binom{m}{i}))$, the summation is from i = 0 to i = log(n) to check all the possible substrings of the term, we only need check upto $log(n)$ as any term with fewer than $m - log(n)$ literals cannot have all elements non 0, as we only have n non 0 elements. So,

in the case when $log(n)$ is much less than m, where m is number of literals, we get a polynomial bound on the time complexity, otherwise if $log(n)$ is itself comparable to m, or n is of exponential order in m, then we cannot do better than exponential time as we have to return exponential number of outputs.

3 Questions

3.1 Algorithm.

Explained in approach

3.2 Time Complexity

Explained in approach

3.3 Testing Strategy

Consider the following given test cases and their output given, checked to be correct by manual calculation that the minimum possible number of terms are given by dropping redundant terms. Test cases 1 and 2 are random test cases. Test case 3 checks for an edge case where no term can be expanded and S.O.P. cannot be minimized. Test case 4 checks for a trivial case, where all possible min terms are in `func_TRUE` and no minimization can be done. Test case 5 checks for a large number of literals in min term. Since these combined checks for trivial cases, random cases and large cases, and thus, the testing strategy can be said to be sufficient for validation of our implementation.

4 Test Cases

4.1 Test Case 1

```
func_TRUE = ["a'bc'd'", "abc'd'", "a'b'c'd", "a'bc'd", "a'b'cd"]
func_DC = ["abc'd"]
Output: ["bc'", "a'b'd"]
```

4.2 Test Case 2

```
func_TRUE = ["a'b'c", "a'bc", "a'bc'", "ab'c'"]
func_DC = ["abc'"]
Output: ["a'c", "a'b", "ac'"]
```

4.3 Test Case 3

```
func_TRUE = ["a'b'c'd'", "a'b'cd", "a'bc'd", "a'bcd'",
"abc'd'", "abcd", "ab'c'd", "ab'cd'"]
func_DC = []
Output: ["a'b'c'd'", "a'b'cd", "a'bc'd", "a'bcd'", "abc'd'", "abcd", "ab'c'd", "ab'cd'"]
```

4.4 Test Case 4

```
func_TRUE = ["a'b'", "a'b", "ab'", "ab"]
func_DC = []
Output: [None]
```

4.5 Test Case 5

```
func_TRUE = ["abcdefghijklmno", "abcdefghijklm'n'o'",
"abcdefghijklmn'o", "abcdefghijklm'n'o", "abcdefghijklm'no'"]
func_DC = ["abcdefghijklm'no", "abcdefghijklmn'o'", "abcdefghijklmno'"]
Output: ['abcdefghijkl']
```