COL215 : Assignment 3 Report

Tejas Anand, Kashish Goel

29 September 2022

1 Problem Description

Given a function with 0s, 1s, and x's for each input combination, print the minimised sum of product terms, where these terms are obtained by deleting as many as terms as possible from the expanded function of the input combination, so that the number of terms thus obtained, is minimised.

- Expanded terms set should cover all the cells with '1'. No such condition for cells with 'x'.

2 Approach

We take the list of all prime implicants obtained from obtained from the algorithm used in assignment 2.
We take each prime implicant, and we check that if it is essential with respect to th function. If yes, we keep it in the function. If no, we remove it from the function and check the same for the next implicant until all implicants have been checked.
How do we check if a prime implicant is essential or not? We know that if an implicant is essential, it will contain atleast one minterm for which if we evaluate the rest of the function (by removing the prime implicant) we get an output of 0. But we must note that this minterm cannot be a *Don't Care* minterm.
So, we iterate over all the *Ones's* minterms and evalauate the value of the rest of the function. If we get an output of 0 for any minterm we conclude that the prime implicant is essential.
We then output the list of prime impicants.

3 Test Cases

```
func_TRUE = ["a'bc'd'", "abc'd'", "ab'c'd'", 'abcd', "ab'cd", "ab'cd'"]
func_DC = ["ab'c'd", "abcd'"]
Expected Output is  ["bc'd'", "ab'", "ac'"]  or  ["bc'd'", "ad'", 'ac']
Output = ["bc'd'", "ab'", 'ac']
This test shows that our algorithm gives one of the possible correct answers
```

```
func_TRUE = ["ab'c'd", "ab'cd"]
func_DC = ["a'bc'd", "abc'd", "a'bcd", 'abcd']
Expected Output is  ['ad']
Output =  ['ad']
This test checks that  Don't Care minterms aren't being considered unnecessarily
Only those minterms which result in literal count reduction are being taken
```

```
func_TRUE = ["a'bc'd'", "abc'd", "ab'c'd", "ab'cd", "a'bcd'", "ab'cd'"]
func_DC = ['abcd', "a'b'cd'"]
Expected Output is  ["a'bd'", 'ad', "ab'c"]  or  ["a'bd'", 'ad', "b'cd'"]
Output =  ["a'bd'", 'ad', "b'cd'"]
This test checks that  Don't Care minterms are being considered when necessary
Only those minterms which result in literal count reduction are being taken
```

```
func_TRUE = ["a'b'c'd'", "a'bc'd'", "abc'd'", "a'b'c'd", "abc'd", "a'b'cd", "a'b
func_DC = []
Expected Output is  ["a'b'c'", "bc'd'", 'abd', "a'cd"]  or
["a'c'd'", "abc'", 'bcd', "a'b'd"]
Output =  ["a'b'c'", "bc'd'", 'abd', "a'cd"]
Output contains all cells in one of the ways
```

4 Comments

4.1 Time Complexity

Let n be the number of input variables. The time complexity of generating all prime implicants was $\mathcal{O}(n^2 4^n)$, from assignment 2. For checking if a particular implicant is essential or not, we generate all its minterms, which takes $\mathcal{O}(2^n)$ at worst and then we compute the value of the function for that minterm , for each minterm we need to do $\mathcal{O}(2^n)$ computations in the worst case, so for a particular implicant we do $\mathcal{O}(2^n * 2^n) = \mathcal{O}(4^n)$ computations at worst, and at worst there are $\mathcal{O}(2^n)$ implicants when we have a checker-board pattern in our function. So we do in the absolute worst case less than $\mathcal{O}(2^n * 4^n)$=$\mathcal{O}(8^n)$ computations. The time complexity then of the complete algorithm is $\mathcal{O}(8^n) + \mathcal{O}(n4^n) = \mathcal{O}(8^n)$

4.2 Testing Strategy

Our testing strategy was to develop test-cases for which the code was likely to fail. This could happen when *Don't Care* terms are being taken unnecessarily in the final answer because during the implementation they were treated identically to the *Ones*. We tested on a lot of such cases, one of them being the 2^{nd} test case given in the report.
Also, since there can be multiple optimal expansions, we made sure that our output matched atleast one of the optimal expansions if there were multiple possible answers. There also is a possibility that there are implicants which are not essential but become essential when we remove the other. We tested this in the last test case.