COL215 Assignment III

Dhruv Ahlawat, 2021CS10556
and
Mihir Kaskhedikar, 2021CS10551

20 September 2022

1 Introduction

Following from Assignment II, in this Assignment we attempt to delete unnecessary terms from our boolean expression to reduce the overall literal count. Using Assignment II's function, we already determined the maximally reduced term corresponding to each minterm, thus we need to only consider these final reduced terms and delete some of them if they are fully contained in the other terms.

2 The Algorithm

We had a set of outputs from our Assignment 2, which gave the final reduced terms corresponding to each minterm that was in func_True. We created a set out of these reduced terms, which removes the duplicates, and then convert it back to list. This is the first step in removing unnecessary terms from our expansion

Next we created a list "MintermFrequency" to store the frequency of minterms, that is, the number of time each minterm gets covered from the list of unique expanded terms. For example, for n = 3, a term "ab" would cover the terms "abc'" and "abc" and thus we increase their frequency by 1 in the list. The list accesses the frequency value corresponding to a minterm by using an index which corresponds to the binary representation of the minterm. For example, frequency of "abc" is stored in the list at the index of "111" in binary, or at the 7th index, while frequency of "abc'" is stored at 110 or index of 6. We traverse our list of unique expanded terms, and update the list MintermFrequency sequentially.

Finally, we traverse the expanded terms list again and check if all of its minterms that were in the input list funcTRUE have frequency more than 1. If all of them do, then we can safely remove this expanded term from our final answer. We do this "removing" by storing a boolean list where the i'th element stores if we want to consider the i'th term in the expanded terms list as the final answer. If we remove a particular term, we also decrement the frequencies of all of its minterms in the list MintermFrequency by 1.

After going through the entire expanded terms list, we have determined some of the terms that are redundant and can be removed without affecting any of the minterms which are 1 (the minterms given in the input funcTRUE). Note that we do not check for the frequencies of the "DON'T CARE" terms as they don't matter for deletions.

(Note that for $n = 15$, the size of the MintermFrequency list is only around 300Kb, and for the maximum case of $n = 26$ the size is ∼500Mb, which can be easily stored within a computer's memory which is why we use a list for quick accessing)

3 Complexity of the Algorithm

Our Assignment 2's worst case time complexity was $O(n4^n)$, which finds the maximally expanded term corresponding to each minterm in input list funcTRUE. We also find all the minterms corresponding to each expanded term in this step.

Our first step converts the answer list to a set to remove duplicates, and then back to a list. Considering that the initial answer list has a loose upper bound of 2^n, converting it to a set and

then back to a list takes at most $O(2^n)$ time, considering $O(1)$ amortised "add" operation for python set.

Then we initialise a list "MintermFrequency" of size 2^n and loop through all the minterms of the expanded terms list. the time complexity of this step would be (total number of minterms)*(n). we multiply by n because finding the index corresponding to a minterm takes $O(n)$ time. The total number of minterms that we find, including duplicates would always be lesser than 2^n * (number of unique expanded terms) $\leq 2^n * 2^n \leq 4^n$. Hence the time complexity of this step would be $O(n4^n)$.

In the next step, we again loop through the unique maximally expanded terms list and check if all of its minterms(that are in funcTRUE) have frequency stored in MintermFrequency more than 1. If true, We then decrement the frequency of all the minterms and also set its value in the "allowed" boolean list as false. This step is analogous to the previous one and also works in at most $O(n4^n)$.

After these steps, our boolean list "Allowed" has been formed and then we just create a new list "finalList" and loop through the expanded terms, adding the i'th expanded term to finalList if Allowed[i] is True. This step takes at most $O(2^n)$ time, and finalList is our final answer that we return.

Thus a loose bound on our time complexity is $O(n4^n)$

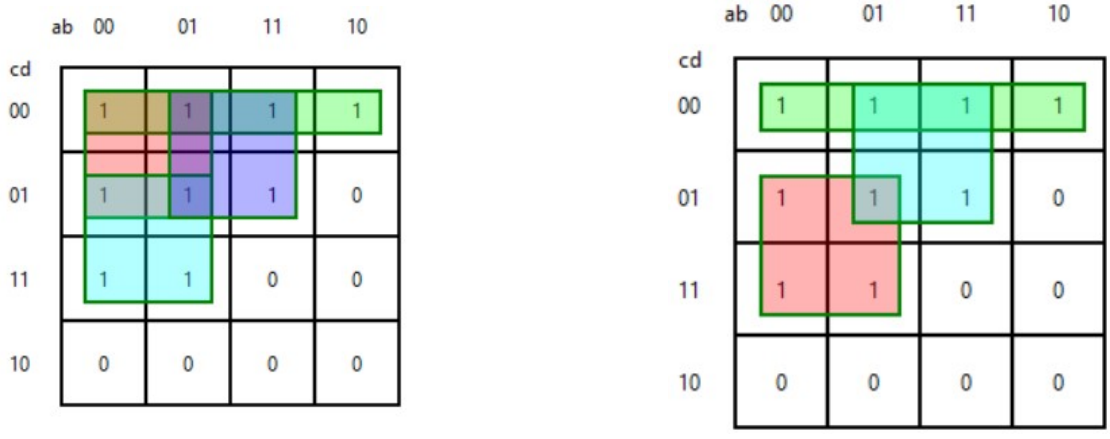
4 Testcases

We want our output list to satisfy 2 conditions:
1. No term in it must appear more than once
2. For every term t appearing in it, there must be atleast one minterm m which is covered only by t in our output list.

Condition 1 is easily taken care of by converting the output list to a set.
Now we want to remove as many terms from our output of **Assignment 2** such that condition 2 gets satisfied. So we can remove any term t satisfies one of these 2 conditions:
1. Every minterm m that is a part of t is covered by some other term t'
2. There exists non-essential(that contain 'x') minterms m that are part of t but are not covered by any other term t' but every essential minterm (that contains '1') that is a part of t is covered by some other term t'.
Thus we need to make such testcases such that both conditions 1 and 2 of deletion are tested.

First, we present 3 testcases in which the removed terms t satisfy condition 1.

Test Case 1 func TRUE = ["a'b'c'd","a'b'cd","a'bc'd","a'bcd","abc'd'","abc'd","a'b'c'd'","a'bc'd'","ab'c'd'"]
func DC=[]



(a) Original maximized terms

(b) after deletion

Assignment 2 output: ["a'd", "a'd", "a'd", "a'd", "bc'", "bc'", "a'c'", "a'c'", "c'd'"]
Assignment 3 output: ["bc'", "c'd'", "a'd"]

Test Case 2

func TRUE = [”a’b’c’d”,”a’b’cd”,”a’bc’d”,”a’bcd”,”abc’d’”,”abc’d”]
func DC=[]
Assignment 2 output: [”a’d”, ”a’d”, ”a’d”, ”a’d”, ”abc’”, ”bc’d”]
Assignment 3 output: [”abc’”, ”a’d”]

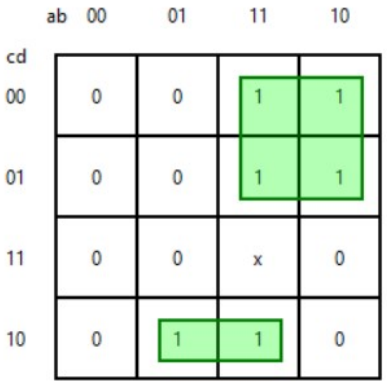
Test Case 3
func TRUE = [”abc’d’”,”abc’d”,”ab’c’d’”,”ab’c’d”,”abcd”,”abcd’”,”a’bcd”,”a’bcd’”]

func DC=[]
Assignment 2 output: [”ac’”, ”ac’”, ”ac’”, ”ac’”, ’ab’, ’ab’, ’bc’, ’bc’]
Assignment 3 output: [”ac’”, ’bc’]

Now we present 3 testcases in which the removed terms t satisfy condition 2.

Test Case 1
func TRUE=[”abc’d’”,”abc’d”,”ab’c’d’”,”ab’c’d”,”a’bcd’”,”abcd’”]
func DC=[”abcd”]

(a) Original maximized terms



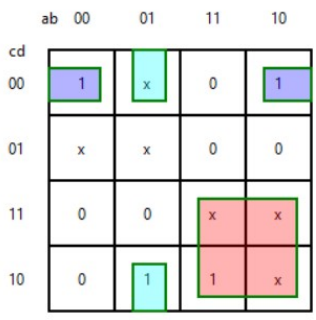
(b) after deletion

Assignment 2 output:[”ac’”, ”ac’”, ”ac’”, ”ac’”, ”bcd’”, ’ab’]
Assignment 3 output:[”ac’”, ”bcd’”]

Test Case 2
func TRUE=[”a’bc’d”,”a’bcd”,”a’bcd’”,”a’b’c’d”]
func DC=[”abc’d”,”abcd”]
Assignment 2 output:[’bd’, ’bd’, ”a’bc”, ”a’c’d”]
Assignment 3 output:[”a’c’d”, ”a’bc”]

Test Case 3
func TRUE=[”a’b’c’d’”,”a’b’cd’”,”ab’c’d”,”ab’cd”]
func DC=[”a’b’c’d”,”a’bc’d’”,”a’bc’d”,”abcd”,”abcd’”,”ab’cd’”]

(a) Original maximized terms



(b) after deletion

Assignment 2 output:[”a’c’”, ”a’b’d’”, ”b’c’d”, ’ac’]
Assignment 3 output:[’ac’, ”b’c’d”, ”a’b’d’”]

5 Conclusion

In this assignment we understood how to check if our code’s output contains some unnecessary terms, thus enabling us a step further in trying to solve the minimum number of minterms problem, which has a great significance in the field of electronics and computer science.