COL215 Assignment II

Mihir Kaskhedikar, 2021CS10551
Dhruv Ahlawat, 2021CS10556

9 September 2022

1 Introduction

In this assignment we try to find the simplest form of a given sum of minterms (SoP) using the idea of dropping literals whenever possible. To simplify a SoP is to find the least number of products(which can contain any number of inputs) whose sum is logically equivalent to the SoP

To cite an example, suppose the given SoP is
a'b'c'd+a'b'c'd'+a'b'cd'+a'b'cd+a'bc'd+a'bcd.
Using Boolean algebra it can be simplified as,
(a'b'c'd+a'b'c'd')+(a'b'cd'+a'b'cd)+(a'bc'd+a'bcd)=a'b'c'+a'b'c=a'b'+a'bd
Clearly no further simplification is possible.

The main advantage in doing this is to understand how the final output varies on varying each of the input and also what maximum inputs we can alter so the output still remains unchanged. There is no known polynomial time algorithm in the number of inputs that can give the simplest form of every SOP in these inputs. So to simplify the problem we were asked to find the simplest expression into which a given minterm can be simplified such that all minterms that satisfy this expression belong to the SoP. So for the SoP above a'b' is a simplified form of the product a'b'c'd, but a' is not a simplified form as a'bcd' does not belong to the SoP.

We were able to find an algorithm of worst time complexity O($n4^n$) for this problem where **n is the number of input signals and not the number of minterms in the SoP**.

2 The Algorithm

We define the size of a product as the number of terms in it. We consider an array of lists **A** such that A[i] stores all those products of size i that can be obtained from addition of some of the minterms in the SoP. So if the number of input signals is n, A[n] stores the minterms of the SoP, and we inductively find A[i] as follows:

Step 1: Begin the iteration over the elements of A[i+1]
Step 2: For each such element, iterate over all the input signals that it contains
Step 3: Create a new product formed by replacing the input signal with its complement
Step 4: Check whether this product belongs in A[i+1] (this can be done using a dictionary)
Step 5: If true, remove this input signal from the product and append the product to A[i] (using dictionary it is ensured that A[i] only contains distinct terms), else continue with the next element of A[i+1]

Once **A** is created, we do the following:

Step 1: Begin the iteration over the minterms of the given SoP.
Step 2: For each such minterm, iterate over the elements of $A[i]$ in increasing order of i.
Step 3: If the element is a superset of the given minterm (i.e. our current minterm belongs to the set of minterms that satisfy this element), we set the answer corresponding to this minterm as this element and continue with the next minterm, else continue with the next element.

3 Complexity of the Algorithm

For our code, the worst case would be the one where all 2^n minterms are present in the SoP. So we analyze the time complexity of this case.
While creating A[i], we are iterating over all elements of A[i+1] first and then over all input signals in each term. The length of A[i+1] can be easily seen to be 2^{n-i+1} and the number of signals is $i+1$. So the number of operations performed is

$(i+1)\text{len}(A[i+1]) \leq (i+1)\binom{n}{i+1}2^{i+1} \leq n\binom{n}{i+1}2^{i+1}$

Here we have used the fact that checking the existence of a string in a Python dictionary takes O(1) time and thus is irrelevant to the algorithm complexity. So the number of operations to create **A** is atmost

$n\sum_{i=1}^{i=n-1} 2^{i+1}\binom{n}{i+1} \leq n4^n$

The second part of the algorithm, finding the answer for each minterm in the SoP would take atmost n*(total number of minterms)*(total products present in **A**) operations and is asymptotic to $n4^n$.

So the worst case complexity is O($n4^n$)

4 Analysis of the Algorithm

4.1 Do all expansions lead to identical set of terms?

No, it is not always necessary that we get the same simplified product for a minterm using two different methods. What can be said is that the sizes of the final products will be same. So if you consider the SoP:
a’bc’d+a’bcd+a’bcd’+abc’d+abcd+abcd’+ab’c’d+ab’cd+ab’cd’
then the simplified product corresponding to abcd can either be bd or bc or ad or ac each of size 2. No further simplification would be possible

4.2 Are all expansions equally good for maximally expanding each term?

Our final goal is finding the most simplified form of the given SoP. One simplification is taking the sum of the answers obtained for each minterm of the SoP in the above algorithm. So it would be better that the set of answers be as small as possible or it may contain several pairs in which one answer of the pair is a subset of the other answer. So it may happen that for a particular minterm of the SoP, replacing its answer with another valid simplification might optimise our set of answers. **Hence we think that not all expansions of a particular minterm are equally good and some are more optimal than others**.

5 Testcases

Input1:
func TRUE=[’abcde’, ”abc’de”, ”abc’d’e”]
func DC=[”a’bcd’e’”, ”a’bcd’e”, ”a’bcde”]
Output=[’bcde’, ’abde’, ”abc’e”]

Input 2:
func TRUE = [”a’bc’d’”, ”abc’d’”, ”a’b’c’d”, ”a’bc’d”, ”a’b’cd”]
func DC = [”abc’d”]
Output=[”bc’”, ”bc’”, ”a’c’d”, ”bc’”, ”a’b’d”]

Input 3:
func TRUE = [”a’b’c’d’e’”, ”a’b’cd’e”, ”a’b’cde’”, ”a’bc’d’e’”, ”a’bc’d’e”, ”a’bc’de”, ”a’bc’de’”, ”ab’c’d’e’”, ”ab’cd’e’”]
func DC = [”abc’d’e’”, ”abc’d’e”, ”abc’de”, ”abc’de’”]
Output=[”c’d’e’”, ”a’b’cd’e”, ”a’b’cde’”, ”bc’”, ”bc’”, ”bc’”, ”bc’”, ”c’d’e’”, ”ab’d’e’”]

6 Conclusion

In this assignment we learned how a maximal valid region containing a given minterm can be found by using the simple idea of dropping literals. The assignment also inspired us to learn more about what class of problems are solvable in polynomial time and what aren't. We also got to know what are NP- Hard problems and what challenges they pose to the researchers.