COL215P Software Assignment 2

Divyansh Mittal - 2020CS10342
Avval Amil - 2020CS10330

October 2022

1 First Logic Used for Expansion

The algorithm used in this assignment for expansion iterates over all the given terms for which the Karnaugh map entry is 1, and uses the terms only for which the Karnaugh map entry is either 1 or dont care (‘x’).
For a particular term, all of the literals of the term are iterated over, and other ‘neighbour’ terms are searched for which will help to expand the given term (it is only checked if a particular neighbour is given to us in either of the input lists or not). A ‘neighbour’ term for a particular term is a term which has all the literals with the same value as that of the term except exactly one (i.e. their gray-code differs in exactly one place). The neighbour terms of a particular term are its neighbours on a Karnaugh map. A particular term is expanded as much as it can be by repeatedly finding its neighbours, and if found then reducing the term and again finding the new term’s neighbours and so on. This process terminates once the term cannot be reduced any further. The above is repeated for all of the terms whose Karnaugh map entry is True/1 and finally the reduced terms are returned. This of course does not give an optimal solution, as once we have expanded the region for a given literal, we never backtrack to check for the possibility of another region leading to overall larger expansion.

2 Second Logic similar to Quine McCluskey Method

Another approach which returns the optimal solution was similar to the one used by Quine McCluskey Method. Initially we have all the terms that have all the literals. We iterate over all pairs of these terms, trying to combine them. We now get a list of all reductions which have n-1 literals. We keep doing this iteratively, reducing them one literal at a time, till no more reduction of literals is possible. Now for each true term, we iterate through the list of reductions in order from maximal reduction to least reduction, to check if this term is part of any of these reductions. Thus we are able to get the maximal reduction in this case.

3 Time Complexity Analysis

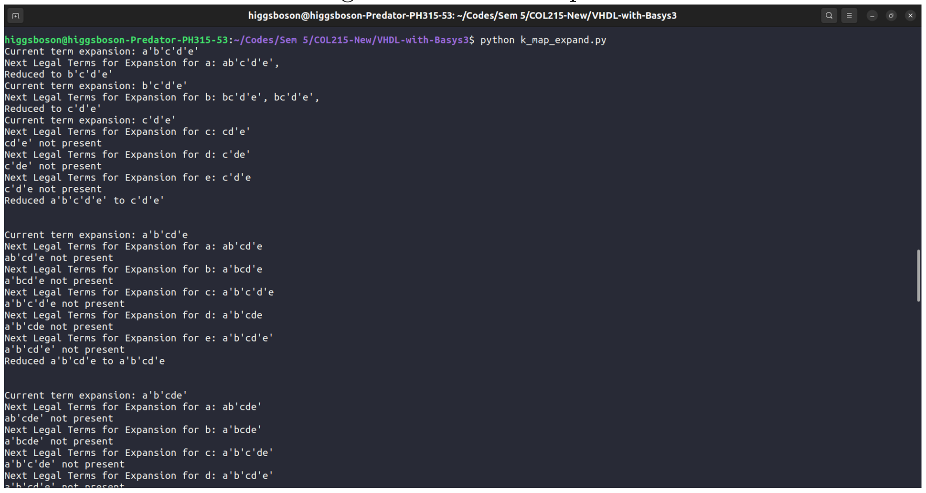
Let the number of variables in the terms given to us be n and the number of terms given to us (in both of the lists combined) be l. We can show that the time taken for the algorithm described in the previous section to run is polynomial in n and l.
First, observe that a particular term can have at most n neighbour terms. So, for a particular term, time taken to find a neighbour term in the list of terms given is $\mathcal{O}(nl)$ as we can iterate over all the elements of the list in the worst case not to find any neighbour. Also note that a particular term can be reduced only $\mathcal{O}(n)$ times, and hence the total time till a particular term is reduced completely is bounded by $\mathcal{O}(n * nl) = \mathcal{O}(n^2 l)$. Finally, there are $\mathcal{O}(l)$ terms in the input list 'func_TRUE' and hence time taken for all of them to be reduced completely is bound by $\mathcal{O}(n^2 l * l) = \mathcal{O}(n^2 l^2)$. Since this bound is polynomial in both n and l. We can say that the time taken for the running of the algorithm presented will not be exponential in the size of the input.

4 Test Cases

For the test cases, we generate them randomly, using brute-force. This helped us confirm that our reductions were correct. The test cases are attached in the submission as well.

We made test cases for literal size 3,4,6,8 and 10.

Figure 1: Code 1 Output

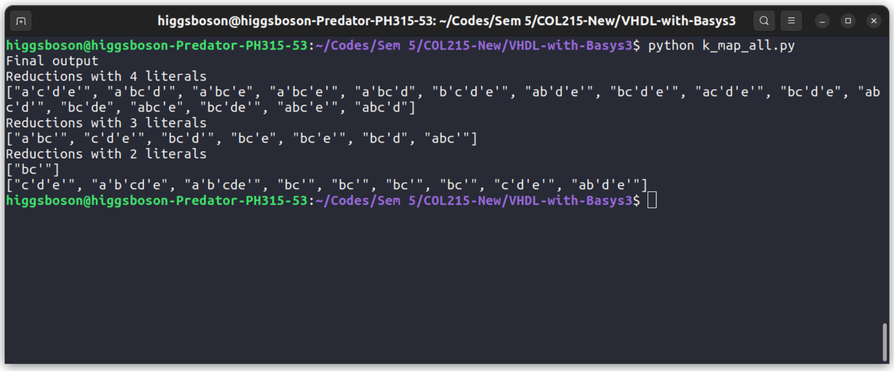


```
higgsboson@higgsboson-Predator-PH315-53: ~/Codes/Sem 5/COL215-New/VHDL-with-Basys3
higgsboson@higgsboson-Predator-PH315-53:~/Codes/Sem 5/COL215-New/VHDL-with-Basys3$ python k_map_expand.py
Current term expansion: a'b'c'd'e'
Next Legal Terms for Expansion for a: ab'c'd'e',
Reduced to b'c'd'e'
Current term expansion: b'c'd'e'
Next Legal Terms for Expansion for b: bc'd'e', bc'd'e',
Reduced to c'd'e'
Current term expansion: c'd'e'
Next Legal Terms for Expansion for c: cd'e'
cd'e' not present
Next Legal Terms for Expansion for d: c'de'
c'de' not present
Next Legal Terms for Expansion for e: c'd'e
c'd'e not present
Reduced a'b'c'd'e' to c'd'e'


Current term expansion: a'b'cd'e
Next Legal Terms for Expansion for a: ab'cd'e
ab'cd'e not present
Next Legal Terms for Expansion for b: a'bcd'e
a'bcd'e not present
Next Legal Terms for Expansion for c: a'b'c'd'e
a'b'c'd'e not present
Next Legal Terms for Expansion for d: a'b'cde
a'b'cde not present
Next Legal Terms for Expansion for e: a'b'cd'e'
a'b'cd'e' not present
Reduced a'b'cd'e to a'b'cd'e


Current term expansion: a'b'cde'
Next Legal Terms for Expansion for a: ab'cde'
ab'cde' not present
Next Legal Terms for Expansion for b: a'bcde'
a'bcde' not present
Next Legal Terms for Expansion for c: a'b'c'de'
a'b'c'de' not present
Next Legal Terms for Expansion for d: a'b'cd'e'
```

Figure 2: Code 2 Output



```
higgsboson@higgsboson-Predator-PH315-53:~/Codes/Sem 5/COL215-New/VHDL-with-Basys3$ python k_map_all.py
Final output
Reductions with 4 literals
["a'c'd'e'", "a'bc'd'", "a'bc'e", "a'bc'e'", "a'bc'd", "b'c'd'e'", "ab'd'e'", "bc'd'e'", "ac'd'e'", "bc'd'e", "ab
c'd'", "bc'de", "abc'e", "bc'de'", "abc'e'", "abc'd"]
Reductions with 3 literals
["a'bc'", "c'd'e'", "bc'd'", "bc'e", "bc'e'", "bc'd", "abc'"]
Reductions with 2 literals
["bc'"]
["c'd'e'", "a'b'cd'e", "a'b'cde'", "bc'", "bc'", "bc'", "bc'", "c'd'e'", "ab'd'e'"]
higgsboson@higgsboson-Predator-PH315-53:~/Codes/Sem 5/COL215-New/VHDL-with-Basys3$ 
```

5 Food for thought

5.1 Do all expansions result in an identical set of terms?

No, for the optimal solution, the count of number of literals will be the same, but there is no guarantee of the sets being identical. The simple contradiction is for a case of no of literals = 4, with all the minterms with a or c are 1 or 'x' and all others 0. In this case, the minterm ab'cd' all have the reduction a or c possible.

5.2 Are all expansions equally good, assuming that our objective is to maximally expand each term? Explain

No, all expansions are not equally good. We may expand a term which might not lead to the optimal expansion. One counter example for n = 4 literals, if we have all terms with b as 1, and also the term ab'cd as 1, so the term abcd can be reduced to acd or b both. If initally we expand with respect to eliminating b, we will get stuck and no more expansion would be possible.