COL215 SW3

Shreyansh Singh Aman Verma

November 12, 2022

1 Function Description

get_literals: This function takes in a term and returns the set of literals in it. For e.g., input = ab'c, output = "a", "b' ", "c".

gray_code: This function returns the gray code for a given number.

get_coordinate: This function takes in a term and returns its location in kmap.

merge: This function takes in 2 terms and merges those terms if only one literal is different. For eg. ab'c' and ab'c will be merged to ab'.

end_points: This function takes in a kmap and a term and returns the start and end coordinates of the region representing that term.

draw_area: This function draws a region on the kmap GUI.

opt_function_reduce: This function determines the minimized set of terms corresponding to the given input.

2 Approach

We first add all the terms with 1s and x's into a set. Then we look at each pair and merge them, if possible, and then add them into a new set. We repeatedly check the pairs in the new set, thus decreasing the number of literals in every new set. Thus we have a list of n+1 sets (where n is the number of variables). The sets in the end might be empty. The terms of the i_{th} ($i \in [0, n]$) set denotes legal region of size 2^i.

We now combine all of the sets to get all of the legal regions. Clearly a set of t terms will never produce more that t^2 terms in any of the subsequent sets.

Next, we extract all maximal terms from this set. We claim that there will be $O(t)$ terms. This is done by removing any term which is a proper subterm of another term in this list. This can be done efficiently - either at time of generation of the sets by simply removing any term which was merged at any point with another term or by iterating over the set to find a super term for it. In any case this is a polynomial time step.

We assume that we have this set up from the previous assignment and thus we go ahead with the analysis from this point.

After this we order the regions based on the number of the numbers of 1s they contain and also their sizes. We finally use this ordering for running an elimination algorithm, in which we remove any region which is covered by the other terms(either those which have already been selected or those which haven't been processed yet - the ordering comes into play here).

To be clear, it seems that obtaining the optimal minimised set of terms is a "hard" problem. Therefore, we use this particular heuristic which essentially tries to greedily select terms with more 1s. In this manner, we also remove non-essential smaller terms and select essential smaller terms, which finally go on to eliminate the larger non-essential terms covered by these smaller essential terms.

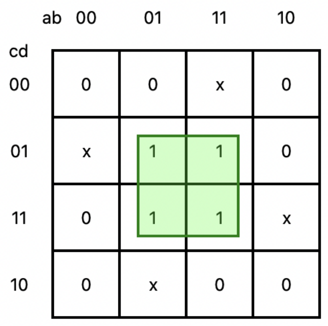
2.1 Time Complexity

We skip the analysis for the part which gets the maximal regions, which was done in the previous assignment. The time complexity for this was $O(t^4 n^2)$. Assuming that we have the set of all maximal regions.The time complexity for processing the maximal set of terms in order to get the reduced set(including sorting and counting 1's is $O(k^2 t)$, where k is the number of maximal terms, which as mentioned earlier was $O(t)$.

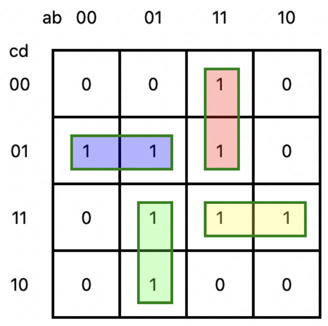
3 Testing strategy

- For testing, we thought of some test cases where things could go wrong and then tested our heuristic with those.
- We also created some cases to show where our heuristic performed well.
- After extensive testing, we could see that the algorithm we used gave optimal outputs for all small cases as well as for all of the cases that we could think of.
- We finally attempted to create a test case which would break our heuristic, but we could not come up with any test case with a small value of N for which our algorithm gave a bad result.

Let us take a look at the following 2 kmaps. The regions marked are the final set of terms selected by our algorithm as the minimised set:



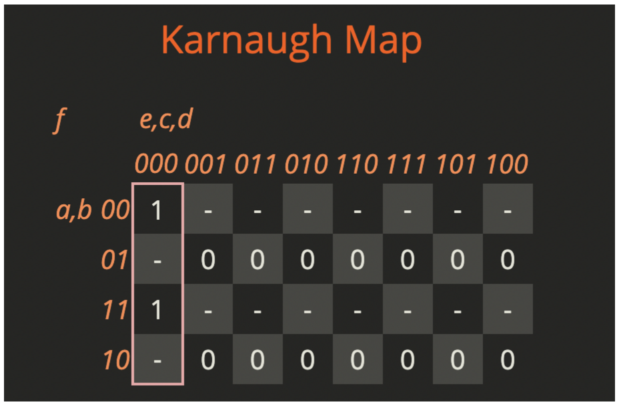
cd \ ab	00	01	11	10
00	0	0	x	0
01	x	1	1	0
11	0	1	1	x
10	0	x	0	0



cd \ ab	00	01	11	10
00	0	0	1	0
01	1	1	1	0
11	0	1	1	1
10	0	1	0	0

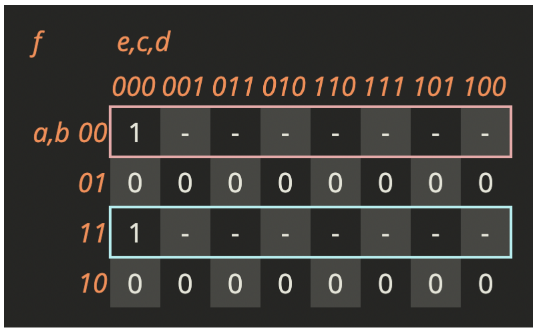
It may be interesting to note that for the above two cases, our algorithm works ideally. For the first case, with the don't cares, we would like to select the region with area 4. However, for the second case, that region ends up being a redundant one since the smaller 4 areas are essential and end up covering the larger area. These test-cases give us a lot of confidence in our heuristic.

The next test-case is a 5 variable case, our algorithm produces optimal outputs for both of the variants(external GUI is used):



Karnaugh Map

f *e,c,d*	000	001	011	010	110	111	101	100
a,b 00	1	-	-	-	-	-	-	-
01	-	0	0	0	0	0	0	0
11	1	-	-	-	-	-	-	-
10	-	0	0	0	0	0	0	0



f	e,c,d							
	000	001	011	010	110	111	101	100
a,b 00	1	-	-	-	-	-	-	-
01	0	0	0	0	0	0	0	0
11	1	-	-	-	-	-	-	-
10	0	0	0	0	0	0	0	0

Even though larger sized regions are detected, it selects the smaller region which covers both 1s and selects that.

We also tested on some larger sized(8-10 variable) test cases, which we have omitted in the report.