Unate Tautology

Nicholas Sica

I. Report

This program was run on a linux operating system with Python version 3.11.6. The program is invoked with "python main.py filename", where the filename is the name of the input file to be used. Most of the data structures are simple lists and flags to build up the cube list and the transposed cube list as well as whether the structure is unate or has both polarities. The basic flow of the program uses the pseudocode that was discussed in lecture as a baseline and follows its layout.

Most of the solutions were double checked with Karnaugh maps and when there was some confusion, walking through what should happen to debug the program. In Fig 1, the Karnaugh map for the unate problem is shown and as expected, it is filled with true outcomes. In Fig 2 the Karnaugh map for the non-unate problem is shown and as expected, it is missing boxes with true outcomes. The outputs for the previous two examples are in Listing 1 and Listing 2, respectively. The output in Listing 3 shows a run on the example problem that was provided with the project. The entire program is shown in Listing 4.

Listing 1: Unate output

```
Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '10', '10']
['11', '11', '01', '11', '11']
['11', '11', '10', '11', '01']
['11', '11', '11', '01', '11']
No rules apply, splitting
Splitting on c at index 2

Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '10', '10']
['11', '11', '11', '11', '11']
['11', '11', '11', '01', '11']
No rules apply, splitting
Splitting on d at index 3

Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '11', '11']
['11', '11', '11', '11', '11']
This is unate
Rule 1 applies, this equals 1
Returning 1

Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '11', '10']
['11', '11', '11', '11', '11']
This is unate
Rule 1 applies, this equals 1
Returning 1

Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '11', '10']
['11', '11', '11', '11', '01']
['11', '11', '11', '11', '11']
Rule 3 applies, one term has positive and complement
Returning 1
This entire SOP equals 1
```

Listing 2: Non-unate output

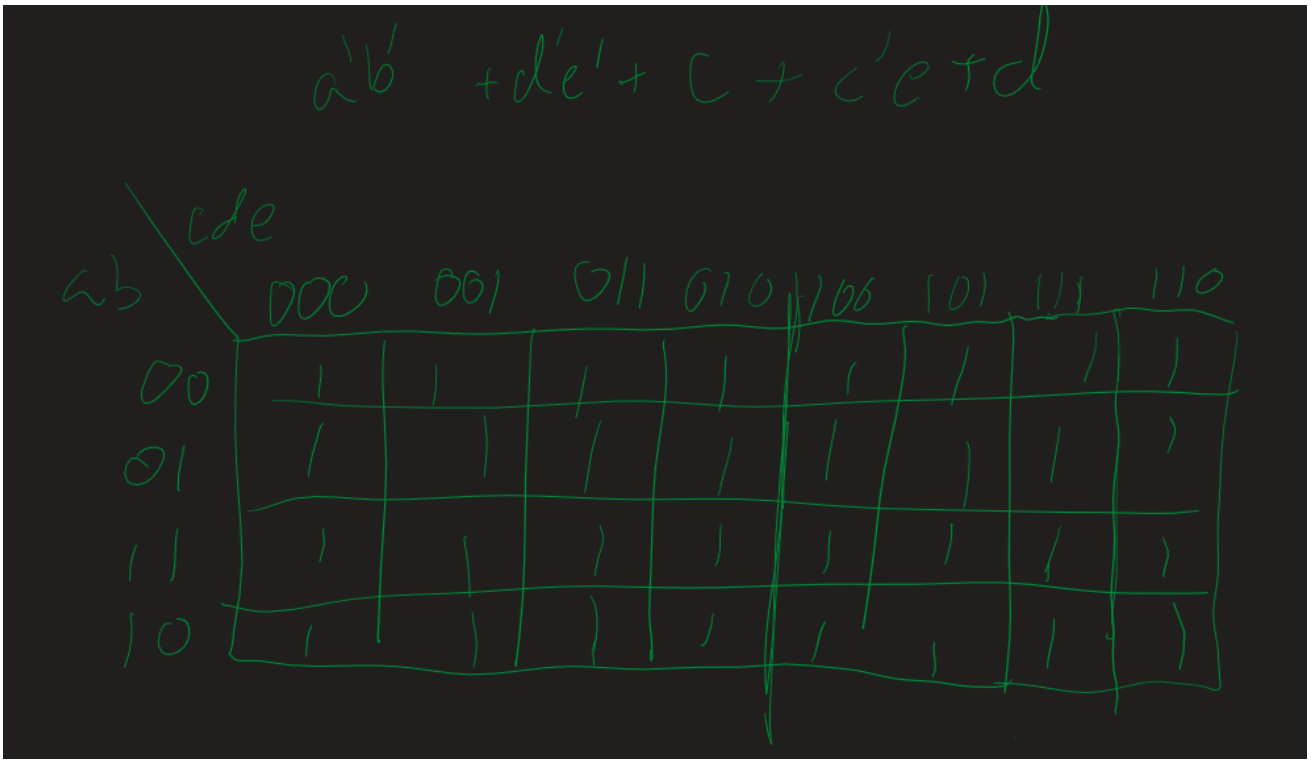

Fig. 1: A Karnaugh map of a unate problem

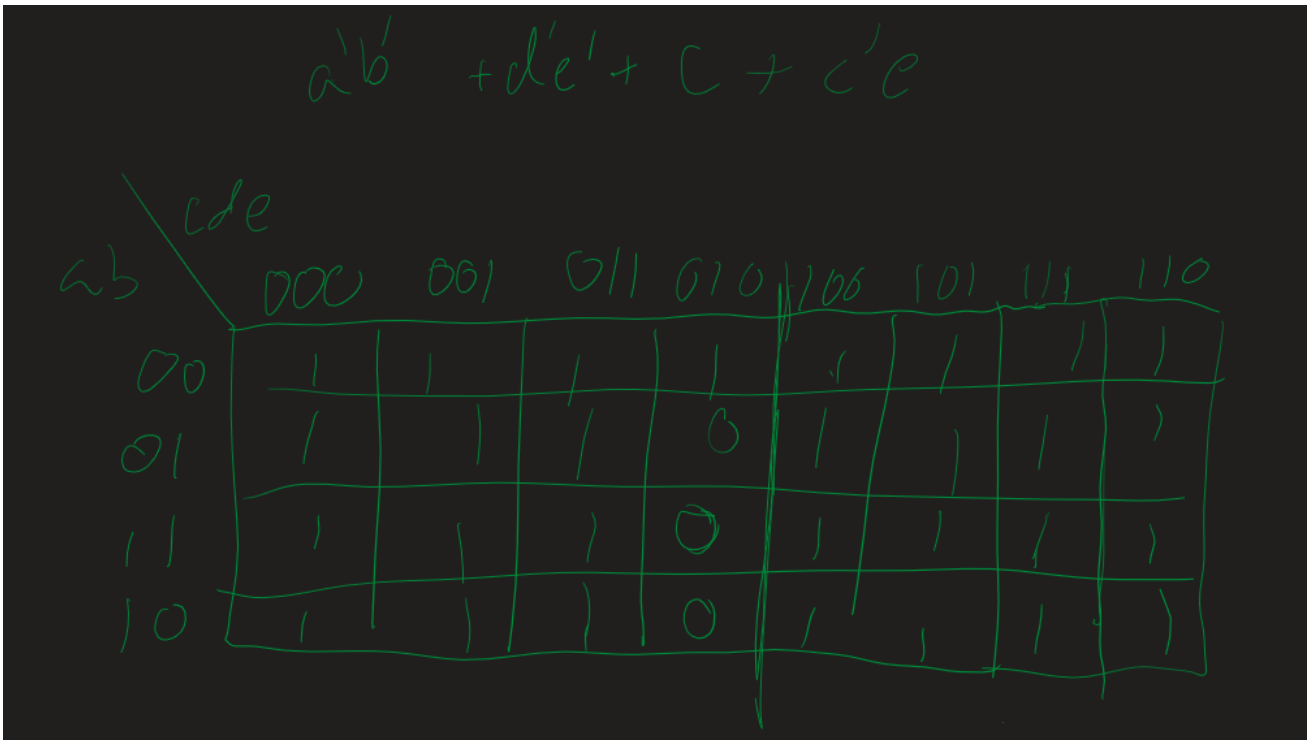

Fig. 2: A Karnaugh map of a non-unate problem

```
Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '10', '10']
['11', '11', '01', '11', '11']
['11', '11', '10', '11', '01']
No rules apply, splitting
Splitting on c at index 2

Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '10', '10']
['11', '11', '11', '11', '11']
This is unate
Rule 1 applies, this equals 1
Returning 1

Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '10', '10']
['11', '11', '11', '11', '01']
No rules apply, splitting
Splitting on e at index 4

Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '11', '11']
This is unate
Rule 1 applies, this equals 1
Returning 1

Cube List
['10', '10', '11', '11', '11']
['11', '11', '11', '10', '11']
This is unate
Rule 2 applies, this does not equal 1
Returning 0
This entire SOP does not equal 1
```

Listing 3: Project problem output

```
Cube List
['01', '11', '11', '11']
['11', '11', '10', '10']
['10', '10', '11', '11']
['01', '01', '01', '11']
['01', '10', '11', '01']
No rules apply, splitting
Splitting on a at index 0

Cube List
['11', '11', '11', '11']
['11', '11', '10', '10']
['11', '01', '01', '11']
['11', '10', '11', '01']
No rules apply, splitting
Splitting on b at index 1

Cube List
['11', '11', '11', '11']
['11', '11', '10', '10']
['11', '11', '01', '11']
No rules apply, splitting
Splitting on c at index 2

Cube List
['11', '11', '11', '11']
['11', '11', '11', '11']
This is unate
Rule 1 applies, this equals 1
Returning 1

Cube List
['11', '11', '11', '10']
['11', '11', '11', '11']
```

```
This is unate
Rule 1 applies, this equals 1
Returning 1

Cube List
['11', '11', '11', '11']
['11', '11', '11', '10']
['11', '11', '11', '01']
Rule 3 applies, one term has positive and complement
Returning 1

Cube List
['11', '11', '11', '10']
['11', '10', '11', '11']
This is unate
Rule 2 applies, this does not equal 1
Returning 0
This entire SOP does not equal 1
```

II. Program

Listing 4: Unate tautology code

```
import string
import argparse
import sys

verbose = 0

def print_cube_list(cube_list):
    print("Cube List")
    for i in cube_list:
        print([f"{x:02b}" for x in i])

def read_sops(file):
    lines = ""
    global var_num, letters
    var_num = 0
    sops = []
    var_locs = {}
    with open(file, 'r') as f:
        var_num = int(f.readline().strip())
        letters = list(string.ascii_lowercase)[0:var_num]
        while (line := f.readline().strip().split(" "))[0] != ";":
            sop = []
            i = 0
            letter_idx = 0
            while line[i] != ";":
                neg = line[i]
                term = line[i+1]
                if letters[letter_idx] != term:
                    sop_term = 0b11
                elif line[i] == '-':
                    sop_term = 0b10
                    i += 2
                elif line[i] == '+':
                    sop_term = 0b01
                    i += 2
                if verbose:
                    print(f"{letters[letter_idx]}: {sop_term:b}")
                sop.append(sop_term)
                letter_idx += 1


            # Fill out remaining values if there are any
            while letter_idx < len(letters):
                if verbose:
                    print(f"{letters[letter_idx]}: 11")
                sop.append(0b11)
                letter_idx += 1
            if verbose:
                print([f"{x:b}" for x in sop])
            sops.append(sop)
        return sops
```

```
def tautology(cube_list):
    print("")
    print_cube_list(cube_list)
    # Check unate
    idx = 0
    tran_cube_list = list(map(list, zip(*cube_list)))
    unate = True
    both_polarities = 0
    break_check = False
    single_cubes = []
    first_loop = True
    var_idx = 0
    for sop in tran_cube_list:
        pos = 0
        neg = 0
        idx = 0
        for term in sop:
            if term == 0b01:
                pos += 1
            elif term == 0b10:
                neg += 1

            only_pos = (cube_list[idx].count(0b01) == 1) and (cube_list[idx].count(0b10) == 0)
            only_neg = (cube_list[idx].count(0b10) == 1) and (cube_list[idx].count(0b01) == 0)
            if first_loop and (only_pos or only_neg):
                single_cubes.append(cube_list[idx])
            idx += 1

        first_loop = False
        if pos > 0 and neg > 0:
            unate = False
            and_reduce = 0b11
            for cube in single_cubes:
                and_reduce = and_reduce & cube[var_idx]

            if and_reduce == 0:
                both_polarities = 1
        var_idx += 1

    if unate == True:
        print("This is unate")
        dont_care = (cube_list.count([0b11] * var_num) > 0)# if all don't care cube return 1
        if dont_care == True:
            print("Rule 1 applies, this equals 1")
            print("Returning 1")
            return 1
        else:
            print("Rule 2 applies, this does not equal 1")
            print("Returning 0")
            return 0
    elif both_polarities == 1:
        print("Rule 3 applies, one term has positive and complement")
        print("Returning 1")
        return 1
    else:
        print("No rules apply, splitting")
        var_idx = 0
        binate_num = 0
        var_diff = 0
        i = 0
        for sop in tran_cube_list:
            term_list = [i for i in sop if i != 0b11 ]
            if len(term_list) > binate_num:
                var_idx = i
                binate_num = len(term_list)
            elif len(term_list) == binate_num:
                true = term_list.count(0b01)
                comp = term_list.count(0b10)
                temp_diff = abs(true - comp)
                if temp_diff < var_diff:
                    var_idx = i
                    var_diff = temp_diff
            i += 1
        print(f"Splitting on {letters[var_idx]} at index {var_idx}")
        new_cube_list = cube_list.copy()
```

```
        cube_list_comp = cube_list.copy()
        cube_list_i = 0
        comp_list_i = 0
        for sop in cube_list:
            if sop[var_idx] == 0b10:
                new_cube_list.remove(sop)
                cube_list_comp[comp_list_i][var_idx] = 0b11
                comp_list_i += 1
            elif sop[var_idx] == 0b01:
                new_cube_list[cube_list_i][var_idx] = 0b11
                cube_list_comp.remove(sop)
                cube_list_i += 1
            else:
                cube_list_i += 1
                comp_list_i += 1

        return tautology(new_cube_list) & tautology(cube_list_comp)

def main():
    if len(sys.argv) == 1:
        print("Please input a file")
        return 1
    file = sys.argv[1]
    sops = read_sops(file)
    taut = tautology(sops)
    if taut == 1:
        print("This entire SOP equals 1")
    else:
        print("This entire SOP does not equal 1")

if __name__ == "__main__":
    main()
```