Birla Institute of Technology and Science Pilani, Hyderabad Campus
CS F214 Logic in Computer Science
I Semester 2020-2021
Homework 4
Total weightage: 15% of final grade

Preface

In this preface, we first informally describe the problem and outline some requisite background. After this, the problem statements for both parts of the homework are succinctly described.

Brief outline of the Problem:

This homework is a coding assignment, where you will write code that will take in a propositional logic formula, and determine if it is satisfiable and valid and print an equivalent propositional logic formula in Conjunctive Normal Form. The broad strategy is to implicitly construct a truth table and use it to determine the above things.

In order to do so, you will first need to write code that given a propositional logic formula and valuation evaluates the truth value of the formula for that valuation. While this seems straightforward to do manually, automating it is somewhat trickier. The background in this preface will walk you through some of the key steps. This is what Part 1 of the homework will accomplish. Part 2 of the homework will use this code to implicitly construct the truth table and determine satisfiability, validity and an equivalent CNF formula.

Definition of Propositional Logic Formula:

- The connective symbols you will use for this assignment is as follows.
  1. ~ for negation
  2. V for OR
  3. ^ for AND
  4. > for implication.
- Here is the formal definition of a fully parenthesized propositional logic formula in BNF:
  <statement> ::= p | (~p) |(~(<statement>)) | (<statement> ^ <statement>) | (<statement>V<statement>) |(<statement> > <statement>)

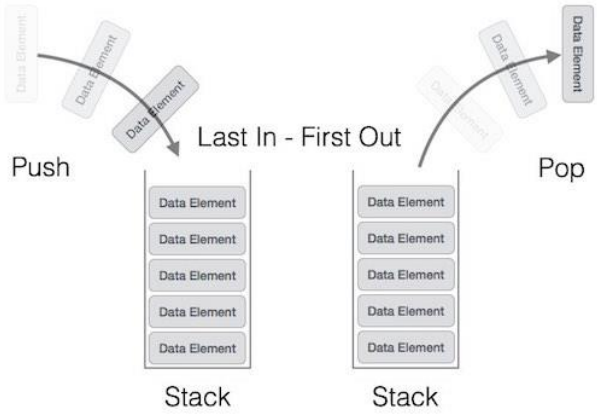
Definition of a Stack:

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only.

Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack. This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last is accessed first.

In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.



A stack is used for the following two primary operations −

- **push()** − Pushing (storing) an element on the top of the stack.
- **pop()** − Removing (accessing) an element from the top of the stack.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named as **top**.

For more see https://en.wikipedia.org/wiki/Stack_(abstract_data_type).

Evaluation of the truth value of a propositional logic formula at a valuation:

We will use two stacks

- Operand stack: This stack will be used to keep statements truth values.
- Operator stack: This stack will be used to keep operators

Let's first define the ***Process*** :(which will be used in the Main algorithm that appears afterward)

1. If operator on the top of the operator stack is *a binary operator* then :
    a. Pop-out two truth values from the operand stack; let's say it is **T** and **F**.
    b. Pop-out operation from operator stack. Let's say it is **V**.
    c. Perform **T V F** and push the result to the operand stack.
2. If operator on the top of the operator stack is *a unary operator* then :
    a. Pop-out one truth value from the operand stack
    b. Pop-out operation from operator stack.
    c. Perform the operation on the operand and push the result back to the operand stack.

Main algorithm:

Scan the given expression from left to right, one character at a time

1. If the character is an operand, push it onto the operand stack.
2. If the character is an operator, then push it onto the operator stack
3. If the character is "(", then push it onto the operator stack.
4. If the character is ")", then do ***Process*** (as explained above) until the corresponding "(" is encountered in operator stack. Now just pop out the "(".

Once the expression iteration is completed and the operator stack is not empty, do the ***Process*** until the operator stack is empty. The value left in the operand stack is our final result.

Example: ((p V q) > ((~r) ^ s)) and truth values **p: T, q: F, r: T and s: T** then the expression need to be evaluated is **((T V F) > ((~T) ^ T))** as given below table and result is **F**.

Token	Action	Operand stack	Operator stack
-	-	Empty	Empty
(	Push onto operator stack	Empty	(
(	Push onto operator stack	Empty	((
T	Push onto operand stack	T	((
V	Push onto operator stack	T	((V
F	Push onto operand stack	TF	((V
)	As top of operator stack is Binary Pop two values from operand stack	Empty	((V
	Pop operator from operator stack	Empty	((
	Push result T V F onto operand stack	T	((
	Just pop out "(" from operand stack	T	(
>	Push onto operator stack	T	(>
(	Push onto operator stack	T	(>(
(	Push onto operator stack	T	(>((
~	Push onto operator stack	T	(>((~
T	Push onto operand stack	TT	(>((~
)	As top of operator stack is Unary Pop one value from operand stack	T	(>((~
	Pop operator from operator stack	T	(>((
	Push result ~T onto operand stack	TF	(>((
	Just pop out "(" from operand stack	TF	(>(
^	Push onto operator stack	TF	(>(^
T	Push onto operand stack	TFT	(>(^
)	As top of operator stack is Binary Pop two values from operand stack	T	(>(^
	Pop operator from operator stack	T	(>(
	Push result F ^ T onto operand stack	TF	(>(
	Just pop out "(" from operand stack	TF	(>
)	As top of operator stack is Binary Pop two values from operand stack	Empty	(>
	Pop operator from operator stack	Empty	(

	Push result T > F onto operand stack	F	(
	Just pop out "(" from operand stack	F	Empty
The value left in the operand stack is our final result: F			

General Instructions:

- You are only allowed to use the C Programming language. Your file should compile with gcc and run on a unix/linux system.
- Please work as a team. There should be **only one submission per team**, which is a single C file of the form TeamX_PartY.c where X is your team number as entered in the Google Sheet that was shared previously and Y is either 1 or 2 referring to whether the submission is that of Part 1 or Part 2.
- The first few lines of your C file should have as comments your team number and the names and ID numbers of all the team members.
- Code corresponding to Part 1 and Part 2 should be submitted separately on the corresponding CMS pages. Part 1 and Part 2 have separate weightage and submission deadlines as noted in the following pages.
- Skeleton code for Part 1 with a stack implementation is provided to you with detailed comments. You should start from this code to complete Part 1. Build Part 2 starting from the code you developed for Part 1.
- You should comment your code properly.
- You will not be able to submit after the deadline. Submit what you have, to possibly receive partial credit.
- If you fail to submit Part 1, you can still submit Part 2 on the subsequent day and receive credit for Part2, but you won't receive marks for Part 1.

- **Do not share code with other teams. Copied code will be awarded zero marks for the entire assignment. Expecting all of you to be honest and take pride in your own work.**

Homework 3, Part 1.

Weightage: 8% of final grade

Submission deadline: Monday, November 23, 11pm

Implement the following tasks in C in light of previously described information:

Given a fully parenthesized propositional logic formula and a valuation (which is an assignment of truth values to the atomic propositions (which are the variables)), evaluate the formula with the given valuation and return either T or F.

For input consider the following simplifications:

Atoms - p1, p2, p3, p4, p5,…are equivalent to integers 1, 2, 3, 4, 5 … respectively. Your code will be expected to handle at most 9 atoms.

i.e, ((p1 V p2) > ((~p3) ^ p4)) is equivalent to ((1 V 2) > ((~3) ^ 4))

For storing operands, formula and operators, use character arrays.

You can assume that the formula given is a correctly-formed formula. You don't need to do error handling for incorrectly formatted formulas.

Input:

Line 1: Number of atoms

Line 2: a fully parenthesized propositional logic formula

Line 3: Assignment of truth values for literals

Output:

Line 4: The result of the evaluated formula.

Example1:

Input:

4

((1 V 2) > ((~3) ^ 4))

TFTT

Output:

F

Example2:

Input:

3

((1 V 2) ^ (1 V 3))

TFT

Output:

T

Homework 3, Part 2.

Weightage: 7% of final grade

Submission deadline: Tuesday, November 24, 11pm

Use code developed in Part 1 to go over all the truth value assignments to determine satisfiability, validity and to construct an equivalent CNF formula.

Input:

Line 1: Number of atoms

Line2: A fully parenthesized propositional logic formula.

Output:

Line3: Formula is satisfiable / not satisfiable.

Line4: Formula is valid / not valid.

Line5: Equivalent CNF formula

Example:

Input:

3

((1 > 2) > ((~3) ^ 2))

Output:

Formula is satisfiable.

Formula is not valid.

((1V2V3)^(1V2V(~3))^(1V(~2)V(~3))^((~1)V(~2)V(~3)))