COMP 5361: Discrete Structures and Formal Languages

Programming Assignment 1

Jenish Soni (40132415)

1. **Source Code:**

PART 1:

```
# N is the number of propositional variables in terms of a string

N = input("Please enter the number of variables: ")
n = int(N)

#we cast the string N to integer n
truthvalues = []

for x in range(1,n+1):
        t = input("Enter truth value: ")
        truthvalues.append(t)
#we input the truth values of the variables in terms of 0 and 1
#print(truthvalues)

e = "((P ∧ Q) ∨ (R ∧ True) ∨ ((¬P ∧ ¬R) ∧ Q)"
m = e


for x in e:
        if(x=="∧"):
                e = e.replace("∧", "&")
        if(x==" "):
                e = e.replace(" ", "")
        if(x=="∨"):
                e = e.replace("∨", "|")
        if(x=="¬"):
                e = e.replace("¬", "~")
        if(x=="→"):
                e = e.replace("→", "<")
        if(x=="⇔"):
                e = e.replace("⇔", ">")

e = e.replace("True", "1")
e = e.replace("False", "1")

#the above code strips the extra space and converts the and, or, implies and iff symbols to
machine readable format
#print("updated" + e)

variables = ""
```

```
for x in e:
	if(x!='(' and x!=')' and x!='1' and x!='0' and x!='&' and x!='|' and x!='~' and x!='<' and
x!='>'):
		if(x in variables):
			continue
		else:
			variables += x

# the variable, (variables) gets the distinct propositional variable from the expression sentence
string.
#print(variables)

for x in variables:
	for y in e:
		if(y=='P'):
			e = e.replace('P', truthvalues[0])
		if(y=='Q'):
			e = e.replace('Q', truthvalues[1])
		if(y=='R'):
			e = e.replace('R', truthvalues[2])

e = e.replace("~1", "0")
e = e.replace("~0", "1")

#print(e)

# we then convert the True and False to 1 and 0 respectively and we remove the not(~) symbol
by taking the complement of the current symbol

stack = []
parentheses = []
operators = []

def operation(t1, t2, t3):
	if(t2 == '&'):
		return int(t1) and int(t3)
	if(t2 == '|'):
		return int(t1) or int(t3)
	if(t2 == '<'):
		return not int(t1) or int(t3)
	if(t2 == '>'):
```

```
                return ((not int(t1) or int(t3)) and (not int(t3) or int(t1)))
# the function operations is for the basic operations and, or, implication and if and only if
function.
res = 0

for x in e:
        if x == ' ':
                continue
        if x == '(':
                parentheses.append(x)

        if x == '1' or x == '0':
                stack.append(x)

        if x == '&' or x == '|' or x == '<' or x == '>':
                operators.append(x)

        if x == ')':
                value2 = stack.pop()
                value1 = stack.pop()
                ope = operators.pop()
                result = operation(value1, ope, value2)
                stack.append(result)
                parentheses.pop()

while(len(operators) != 0):
        if(len(stack) >= 2 and len(operators) >= 1):
                value2 = stack.pop()
                value1 = stack.pop()
                ope = operators.pop()
                result = operation(value1, ope, value2)
                stack.append(result)
                #parentheses.pop()


# the above is the stack implementation of the code which takes the parentheses in the
parentheses stack, operators in the operators stack and values 1 and 0 in the stack named
"stack"


pin = stack.pop()

#pin is the top element of the stack-------->pin = stack[-1]
```

```
if(pin == 0):
        print("Truth Value of " + m  + ": False" )
else:
        print("Truth Value of " + m  + ": True" )

#coded by jenish soni (40132415)
```

PART 2:

```
#initialising variables to 0 for the count of truth values

taut = 0
contra = 0

print("=========================================================================
=========================================================================
========================")
print("    P1   |    P2    |    ~P1   |   P1 V P2   |   S(Output)")
print()

for P1 in [True,False]:
        for P2 in [True, False]:
                S = (not(not P1 and (P1 or P2)) or P2)
                if S==True:
                        taut = taut + 1

                else:
                        contra = contra + 1


                print("   " ,P1, "    |   " ,P2, "   |   "  ,not P1, "   |   " ,P1 or P2, "   |   " ,S)

if taut == 4:
        print("S is a TAUTOLOGY")
elif contra == 4:
        print("S is a CONTRADICTION")
else:
        print("S is a CONTINGENCY")
# as there are only 2 propositional variables here, the maximum values as true and false can be
4 (2^2 = 4)
```

```
print("=========================================================================
=========================================================================
========================")

taut = 0
contra = 0
# again changing their values to zeroes

print("     P1    |     P2    |    ~P1    |   ~P2  |    P1=>~P2  |  ~P1=>~P2  | S(Output)  ")
print()

for P1 in [True,False]:
        for P2 in [True, False]:
                S = P2 and (not P1 or not P2) and (P1 or not P2)
                if S==True:
                        taut = taut + 1

                else:
                        contra = contra + 1


                print("   " ,P1, "    |   " ,P2, "   |   "  ,not P1, "   |   " ,not P2, "   |   " ,not P1 or not P2,
"   |   " ,P1 or not P2, "   |   " ,S)


if taut == 4:
        print("S is a TAUTOLOGY")
elif contra == 4:
        print("S is a CONTRADICTION")
else:
        print("S is a CONTINGENCY")

# as there are only 2 propositional variables here, the maximum values as true and false can be
4 (2^2 = 4)


print("=========================================================================
=========================================================================
========================")
# again changing their values to zeroes
taut = 0
contra = 0
```

```
print("      P1    |    P2    |    P3    |    ~P1    |     ~P2     |    ~P3    |    P2=>P3
|P1=>(P2=>P3)|   P1=>P2   | P1=>(P2=>P3) |  S(Output)")
print()

for P1 in [True,False]:
        for P2 in [True, False]:
                for P3 in [True, False]:

                        S = (P1 and P2 and not P3) or ((P1 and not P2) or P3)
                        if S==True:
                                taut = taut + 1

                        else:
                                contra = contra + 1


                        print("   " ,P1, "    |   " ,P2, "   |   "  ,P3, "   |   " ,not P1, "   |   " ,not P2, "   |   "
,not P3, "   |   " ,not P2 or P3, "   |   " ,not P1 or(not P2 or P3), "   |   " ,not P1 or P2, "   |   " ,not P1
or (not P2 or P3), "   |   " ,S)


if taut == 8:
        print("S is a TAUTOLOGY")
elif contra == 8:
        print("S is a CONTRADICTION")
else:
        print("S is a CONTINGENCY")

# as there are 3 propositional variables here, the maximum values as true and false can be 8
(2^3 = 8)
# coded by jenish soni
```

2. **Explanation of the program: (PART 1)**
    - In the first part of the program, the implementation is made using the help of the stack data structure. There are 3 different lists which take, parentheses, operators, and variables as input as pops out as the value of the variables and the associated operations are performed. The final answer is always on the top of the stack list. Time Complexity for the program is O(n) - linear time.

Explanation of the program: (PART 2)

- Using nested for loops I have iterated through multiple lists to get all the possible combinations according to the number of variables inputted. There are two counters namely taut and contra which increments whenever a true and false values are found respectively. For n inputs, if the value of taut or contra is (n^2) then depending on the values of taut or contra it is a tautology, contingency or a contradiction.

3.Solutions(with images)

Part 1:

PART 2:

```
    P1    |     P2     |      ~P1    |    P1 V P2   |    S(Output)

   True     |    True    |    False    |    True    |    True
   True     |    False   |    False    |    True    |    True
   False    |    True    |    True     |    True    |    True
   False    |    False   |    True     |    False   |    True
S is a TAUTOLOGY
=============================================================================================================================
     P1     |      P2     |     ~P1     |    ~P2    |     P1=>~P2    |   ~P1=>~P2  |  S(Output)

   True     |    True    |    False    |    False    |    False    |    True    |    False
   True     |    False   |    False    |    True     |    True     |    True    |    False
   False    |    True    |    True     |    False    |    True     |    False   |    False
   False    |    False   |    True     |    True     |    True     |    True    |    False
S is a CONTRADICTION
=============================================================================================================================
       P1    |      P2     |     P3     |      ~P1     |       ~P2       |      ~P3      |    P2=>P3  |P1=>(P2=>P3)|   P1=>P2   | P1=>(P2=>P3) |  S(Output)

   True     |    True    |    True    |    False    |    False    |    False    |    True    |    True    |    True    |    True    |    True
   True     |    True    |    False   |    False    |    False    |    True     |    False   |    False   |    True    |    False   |    True
   True     |    False   |    True    |    False    |    True     |    False    |    True    |    True    |    False   |    True    |    True
   True     |    False   |    False   |    False    |    True     |    True     |    True    |    True    |    False   |    True    |    True
   False    |    True    |    True    |    True     |    False    |    False    |    True    |    True    |    True    |    True    |    True
   False    |    True    |    False   |    True     |    False    |    True     |    False   |    True    |    True    |    True    |    False
   False    |    False   |    True    |    True     |    True     |    False    |    True    |    True    |    True    |    True    |    True
   False    |    False   |    False   |    True     |    True     |    True     |    True    |    True    |    True    |    True    |    False
S is a CONTINGENCY
PS C:\Users\jenis\Desktop\T1\COMP 5361>
```