UNC CHARLOTTE
College of Computing and Informatics

FINAL PROJECT

ITCS-6150
INTELLIGENT SYSTEMS

Submitted by:
Sai Harsha Kanamarlapudi(801167706)
Amulya Prodduturi (801166255)
Krupa Chand Appikonda (801166358)
Tagore Pothuneedi (801164787)

Aim: Implement the verification method (called RS method) to determine if the given formula in the propositional calculus is a tautology.

Assume that L_0 is a language of order zero. Letter S will denote finite sequences ($\alpha_1, \alpha_2 \ldots. \alpha_m$) of formulas in L_0 .

If S1 = ($\alpha_1, \alpha_2 \ldots. \alpha_m$) and S2 = ($\beta_1, \beta_2 \ldots. B_n$) and α , β are any formulas, then S1, α, S2 and S1, α , β, S2 denote sequences ($\alpha_1, \alpha_2 \ldots. \alpha_m, \alpha$, $\beta_1, \beta_2 \ldots. B_n$) and ($\alpha_1, \alpha_2 \ldots. \alpha_m, \alpha$, β , $\beta_1, \beta_2 \ldots. B_n$).

A Formula is indecomposable if it is either a propositional variable or negation of propositional variable.

A sequence is indecomposable provided it is formed only of indecomposable formulas.

A sequence is fundamental if it simultaneously contains a formula α and its negation.

For this project, we consider two types of schema's: S1/S2 and S1/(S2;S3).

Here S1 is called a premise and S2,S3 are conclusions.

If a schema is of the form S1/(S2;S3) , then S2 is left conclusion and S3 is right conclusion.

We consider the following schema's:

Let D(α) denote the diagram tree built for α using the above schema's.

Tautology:

A tautology is a formula which is "always true" --- that is, it is true for every assignment of truth values to its simple components. You can think of a tautology as a rule of logic.

Example: (P -> Q) V (Q -> P)

P	Q	$P \to Q$	$Q \to P$	$(P \to Q) \vee (Q \to P)$
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

Propositional logic:

A proposition is a statement that can be either true or false; it must be one or the other, and it cannot be both.

A formula F is a propositional tautology if and only if all end sequences in the diagram D(α) are fundamental.

A propositional formula is a type of syntactic formula which is well formed and has a truth value. If the values of all variables in a propositional formula are given, it determines a unique truth value.

EXAMPLES. The following are propositions:

- The reactor is on
- The wing-flaps are up
- John Major is prime minister.

whereas the following are not:

- Are you going out somewhere?
- 2+3

Code Structure:

```
# Enter the input in variables of either a, b, c, d, p, q, r, s
# Reference for Propositional formula elements
# 'and' == &
# 'or' == |
# '-> == >>
# 'not' == ~
# <-> ('Iff') == <<


#Input - This program expects a propositional formula.
#Output - The expected output is whether the propositional formula is a tautology or not.

#Tautology -

class Expression:
   def __invert__(self):
      return Not(self)
```

```
def __and__(self, other):
   return And(self, other)

def __or__(self, other):
   return Or(self, other)

def __rshift__(self, other):
   return Implies(self, other)

def __lshift__(self, other):
   return Iff(self, other)

def __eq__(self, other):
   return self.__class__ == other.__class__ and self.eq(other)

def call_func(self, left, right):
   while True:
      found = True
      for item in left:
         if item in right:
            return None
         if not isinstance(item, Prop_Var):
            left.remove(item)
            tup = item._tleft(left, right)
            left, right = tup[0]
            if len(tup) > 1:
               v = self.call_func(*tup[1])
               if v is not None:
                  return v
            found = False
            break
      for item in right:
         if item in left:
            return None
         if not isinstance(item, Prop_Var):
            right.remove(item)
            tup = item._tright(left, right)
            left, right = tup[0]
            if len(tup) > 1:
               v = self.call_func(*tup[1])
               if v is not None:
```

```
                return v
            found = False
            break
        if found:
          return "The Given Propositional Formula is not a Tautology"

    def _call_func(self):
      return self.call_func([], [self])


class Bin_Op(Expression):
    def __init__(self, left_child, right_child):
      self.left_child = left_child
      self.right_child = right_child

    def __str__(self):
      return '(' + str(self.left_child) + ' ' + self.op + ' ' + str(self.right_child) + ')'

    def eq(self, other):
      return self.left_child == other.left_child and self.right_child == other.right_child


class And(Bin_Op):
    op = '^'
    st = ''
    def _tleft(self, left, right):
      print("Result of AND function", self.left_child, self.right_child)
      return (left + [self.left_child, self.right_child], right),

    def _tright(self, left, right):
      print("Result of AND function", self.left_child, self.right_child)
      return (left, right + [self.left_child]), (left, right + [self.right_child])


class Implies(Bin_Op):
    op = '->'

    def _tleft(self, left, right):
      print("Result of Implies function", self.left_child, self.right_child)
      return (left + [self.right_child], right), (left, right + [self.left_child])
```

```
    def _tright(self, left, right):
        print("Result of Implies function", self.left_child, self.right_child)
        return (left + [self.left_child], right + [self.right_child]),


class Iff(Bin_Op):
    op = '<->'

    def _tleft(self, left, right):
        print("Result of Iff function", self.left_child, self.right_child)
        return (left + [self.left_child, self.right_child], right), (left, right + [self.left_child, self.right_child])

    def _tright(self, left, right):
        print("Result of Iff function", self.left_child, self.right_child)
        return (left + [self.left_child], right + [self.right_child]), (left + [self.right_child], right + [self.left_child])


class Not(Expression):
    def __init__(self, child):
        self.child = child

    def __str__(self):
        return '~' + str(self.child)

    def eq(self, other):
        return self.child == other.child

    def _tleft(self, left, right):
        return (left, right + [self.child]),

    def _tright(self, left, right):
        return (left + [self.child], right),


class Or(Bin_Op):
    op = 'v'

    def _tleft(self, left, right):
        print("Result of Or function", self.left_child, self.right_child)
```

```
        return (left + [self.left_child], right), (left + [self.right_child], right)

    def _tright(self, left, right):
        print("Result of Or function", self.left_child, self.right_child)
        return (left, right + [self.left_child, self.right_child]),


class Prop_Var(Expression):
    def __init__(self, name):
        self.name = name

    def __hash__(self):
        return hash(self.name)

    def __str__(self):
        return str(self.name)

    __repr__ = __str__

    def eq(self, other):
        return self.name == other.name


a = Prop_Var('a')
b = Prop_Var('b')
c = Prop_Var('c')
d = Prop_Var('d')
p = Prop_Var('p')
q = Prop_Var('q')
r = Prop_Var('r')
s = Prop_Var('s')
treestruct = []

def rs_method(e):
    print("The given input in actual propositional formula terms ", e)
    result = e._call_func() # This function break down the given formula to evaluate if it's a
tautology or not
    #print(treestruct)
    if result == None:
        print("The Given Propositional Formula is a Tautology")
    else:
```

```
        print("The Given Propositional Formula is not a Tautology")


# input2 = 'a & b'
# input1 = '(a|b)|~b'
# input3 = "(((~a | (b >> p)) & (a | b)) >> b)"
while(True):
    inputString = input("Please enter the Propositional Formula. Press Ctrl + C to terminate:")
    #treestruct.clear()

    try:
        inputString = inputString.lower()
        e = eval(inputString)
        rs_method(e)
    except NameError:
        print("Invalid Propositional Formula. Please enter a valid one")
```

By definition, tautology is a propositional formula that is true under any possible Boolean valuation of its propositional variables. We can validate the correctness of the formula by drafting a truth table. If all the values of the last column are 1 / True, then we can conclude that the given formula is True for any given value and hence tautology can be obtained.

Results of the Experiment:

Input: (a|b)|~b

```
Please enter the Propositional Formula. Press Ctrl + C to terminate:(a|b)|~b
The given input in actual propositional formula terms  ((a v b) v ~b)
Result of Or function (a v b) ~b
Result of Or function a b
The Given Propositional Formula is a Tautology
Please enter the Propositional Formula. Press Ctrl + C to terminate:
```

$(a|b)|\sim b$ $[\ | = or = \cup \text{ (union)}\ ]$

a	b	$a \cup b$	$\sim b$	$(a \cup b) \cup \sim b$
0	0	0	1	1
0	1	1	0	1
1	0	1	1	1
1	1	1	0	1

This implies this is a tautology.

Input: "(((~a | (b >> p)) & (a | b)) >> b)":

```
Result of Implies function ((~a v (b -> p)) ^ (a v b)) b
Result of AND function (~a v (b -> p)) (a v b)
Result of Or function ~a (b -> p)
Result of Or function a b
Result of Implies function b p
Result of Implies function b p
The Given Propositional Formula is not a Tautology
Please enter the Propositional Formula. Press Ctrl + C to terminate:
```

$(\sim a \mid (b \to p)) \ \& \ (a \mid b) \to b$.

a	b	p	$\sim a$	$b \to p$	$\sim a \lor (b \to p)$	$a \lor b$
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	0	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	0	0	0	1
1	1	1	0	1	1	1

$a \lor b \to b$	$\sim a \lor (b \to p) \ \& \ (a \lor b) \to b$
1	1
1	1
1	1
1	1
0	0
0	0
1	0
1	1

This is not true for all the values. Hence, it's not a tautology.

Input: ((a -> b) V (b -> c))

```
================= RESTART: E:\IS\Final Project\tautology_rs.py =================
Please enter the Propositional Formula. Press Ctrl + C to terminate:(a>>b)|(b>>c)
The given input in actual propositional formula terms  ((a -> b) v (b -> c))
Result of Or function (a -> b) (b -> c)
Result of Implies function a b
Result of Implies function b c
The Given Propositional Formula is a Tautology
Please enter the Propositional Formula. Press Ctrl + C to terminate:
```

$(a >> b) \mid (b >> c)$

a	b	c	$b \to c$	$a \to b$	$(a \to b) \vee (b \to c)$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	0	1	1
0	1	1	1	1	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	1	1	1

This is a tautology

Here

0 can be construed as False
1 can be construed as True.