Discrete Structures
Assignment 02

Submitted by:

Name: Ali Husnain
Roll No: P17-6071
Section: B

To:

Dr. Nauman Azam

Example 06:

```
 x  -  ▫  sasuke@sasuke-uchiha: ~/Desktop/python
INSTRUCTONS:
 ----------------------------
 | Operators |  Functions   |
 ----------------------------
 |    <->    | biconditional|
 |    ->     | implies      |
 |     ^     | conjunction  |
 | ,(comma)  | disjunction  |
 |     ~     | negation     |
 ----------------------------
1. This code works when the input premises are '.'(dot) seperated
   and the last one will be the conclusion.
2. Only those logical operators will use which are mentioned in above table.
   Otherwise, it will give you the wrong answer(s).
3. Run this file on python 3.X kernel.
------------------------------------------------------------------


------------------------------------------------------------------
Question 06:

    1. ~p^q
    2. r->p
    3. ~r->s
    4. s->t
    5. t

Solution:

    C1 : q              By simplification
    C2 : ~p             By simplification
    C3 : ~r,p           By simplification
    C4 : r,s            By simplification
    C5 : ~s,t           By simplification
    C6 : ~t             By simplification

    C7 : ~r             By solving (~p) and (~r,p)
    C8 : r,t            By solving (r,s) and (~s,t)
    C9 : t              By solving (~r) and (r,t)
    C10:                By solving (t) and (~t)
    C11: q              By solving (q).

        There is an independent clause 'q'. So, this system is invalid.
              >-----------------------------------<
```

Example 07:

```
 ×  -  ▫  sasuke@sasuke-uchiha: ~/Desktop/python
INSTRUCTONS:
 ---------------------------
 | Operators |  Functions    |
 ---------------------------
 |    <->    | biconditional|
 |    ->     | implies      |
 |     ^     | conjunction  |
 | ,(comma)  | disjunction  |
 |     ~     | negation     |
 ---------------------------
1. This code works when the input premises are '.'(dot) seperated
   and the last one will be the conclusion.
2. Only those logical operators will use which are mentioned in above table.
   Otherwise, it will give you the wrong answer(s).
3. Run this file on python 3.X kernel.
------------------------------------------------------------------


------------------------------------------------------------------
Question 07:

    1. p->q
    2. ~p->r
    3. r->s
    4. ~q->s

Solution:

    C1 : ~p,q           By simplification
    C2 : p,r            By simplification
    C3 : ~r,s           By simplification
    C4 : ~q             By simplification
    C5 : ~s             By simplification

    C6 : q,r            By solving (~p,q) and (p,r)
    C7 : ~r             By solving (~r,s) and (~s)
    C8 : q              By solving (q,r) and (~r)
    C9 :                By solving (q) and (~q)

         This system is valid.
       >----------------------<

➜ python
```

Example 08:

```
 x  -  ▫  sasuke@sasuke-uchiha: ~/Desktop/python
INSTRUCTONS:
 ---------------------------
 | Operators |  Functions    |
 ---------------------------
 |    <->    | biconditional|
 |    ->     | implies      |
 |     ^     | conjunction  |
 | ,(comma)  | disjunction  |
 |     ~     | negation     |
 ---------------------------
1. This code works when the input premises are '.'(dot) seperated
   and the last one will be the conclusion.
2. Only those logical operators will use which are mentioned in above table.
   Otherwise, it will give you the wrong answer(s).
3. Run this file on python 3.X kernel.
------------------------------------------------------------------------


------------------------------------------------------------------------
Question 08:

     1. T->(M,E)
     2. S->~E
     3. T^S
     4. M

Solution:

     C1 : ~T,M,E          By simplification
     C2 : ~S,~E           By simplification
     C3 : S               By simplification
     C4 : T               By simplification
     C5 : ~M              By simplification

     C6 : ~T,M,~S         By solving (~T,M,E) and (~S,~E)
     C7 : ~T,M            By solving (~T,M,~S) and (S)
     C8 : M               By solving (~T,M) and (T)
     C9 :                 By solving (M) and (~M)

          This system is valid.
        >------------------------<

➜ python
```

Example 09:

```
 ×  -  ▫  sasuke@sasuke-uchiha: ~/Desktop/python
INSTRUCTONS:
 ----------------------------
 | Operators |  Functions   |
 ----------------------------
 |    <->    | biconditional|
 |    ->     | implies      |
 |     ^     | conjunction  |
 | ,(comma)  | disjunction  |
 |     ~     | negation     |
 ----------------------------
1. This code works when the input premises are '.'(dot) seperated
   and the last one will be the conclusion.
2. Only those logical operators will use which are mentioned in above table.
   Otherwise, it will give you the wrong answer(s).
3. Run this file on python 3.X kernel.
------------------------------------------------------------------

------------------------------------------------------------------
Question 08:

    1. L->A
    2. E->~I
    3. A->E
    4. L->~I

Solution:

    C1 : ~L,A           By simplification
    C2 : ~E,~I          By simplification
    C3 : ~A,E           By simplification
    C4 : L              By simplification
    C5 : I              By simplification

    C6 : ~L,E           By solving (~L,A) and (~A,E)
    C7 : ~E             By solving (~E,~I) and (I)
    C8 : ~L             By solving (~L,E) and (~E)
    C9 :                By solving (~L) and (L)

        This system is valid.
       >----------------------<

➜ python
```

Code:

If you can verify my code from this website by downloading the file; ***https://github.com/linxnerd/principle-of-resolution***

OR

Copy and paste following code;

```
try:
    input = raw_input
except:
    pass

def And(p):
    return p.split('^')

def biconditional(p):
    if not "<->" in p:
        return p
    p = p.split("<->")
    p = '('+p[0] + "->" +p[1]+')^(' +p[1]+"->"+p[0]+')'
    p = p.split('^')
    p[0] = implies_To_disjunction(p[0])
    p[1] = implies_To_disjunction(p[1])
    p = '^'.join(p)
    return p
# # biconditional("p<->q")

def implies_To_disjunction(p):
    if  not "->" in p: return p
    s = p.split(',')
    for i in range(len(s)):
        if '->' in s[i]:
            s[i] = s[i].split('->', 1)
            if s[i][0][0] == '(':
                if ')' in s[i][0]:
                    s[i][0] = '~' + s[i][0]
                else:
                    s[i][0] = s[i][0].replace("(","")
                    s[i][0] = '(~' + s[i][0]
                    if '~~' in s[i][0]:
                        s[i][0] = s[i][0].replace("~~","")
            else:
                s[i][0] = '~' + s[i][0]
            s[i] = ','.join(s[i])
            if '~~' in s[i]:
                s[i] = s[i].replace("~~", "")
            break;
    p = ','.join(s)
    return implies_To_disjunction(p)
# implies_To_disjunction("(~p->~q)")
def negation(p,q,o):
```

```
    if o == '->':
        return '(~' + p +',' +q+')'
    if o == '^':
        return '(~' + p +',~' +q+')'
    if o == ',':
        return '(~' + p +'^~' +q+')'
    if o == '<->':
        p = '('+p +'<->~' +q + ')'
        p = biconditional(p)
        return
    else:
        return '~' + p
def removeExtra(s = "xyz"): # remove extra brackets
    for i in range(len(s)):
        if (s[i] == "(" == s[i+1] or s[i] == "(" and '~' == s[i+1] )and (s[len(s)-1-i] == s[len(s)-1-(i+1)] == ')'):
            s = s[1:-1]
            return s
        return s
def removeBrackets(s):   # remove useless brackets
    a = 0
    b = 0
    for i in s:
        if i == '(':
            a+=1
        if i == ')':
            b+=1
    if a > b:
        s = s[1:]
        return s
    elif a < b:
        s = s[:-1]
        return s
    else:
        return s
def simple(s):
    x = ""
    for i in range(len(s)):
        if (s[i] == '~' and s[i+1]== '('):
            for j in range(i+2, len(s)):
                temp = ""
                if s[j] == '~' and s[j+1]== '(':
                    y  = s[s.index('(')+1:s.index(')')+1]
                    z = simple(y)
                    s = s.replace(y,z)
                if s[j] == '(':
                    p = s[s.index('(')+1:s.index(')')+1]
                    temp = s
                    temp = temp.replace(p,"")
                    if temp[j] == '-' and temp[j+1] == '>':
                        o = temp[j] + temp[j+1]
                        q = temp[j+2:s.index(')')]
                        q = removeBrackets(q)
                    elif temp[j] == '<' and temp[j+1] == '-' and temp[j+2] == '>':
                        o = temp[j] + temp[j+1] + temp[j+2]
                        q = temp[j+3:]
                    else:
                        o = temp[j]
                        q = temp[j+1:]
```

```
                q = removeBrackets(q)
            temp = negation(p,q,o)
            s = s.replace(s, temp)
        else:
            if s[j] == '~':
                p = s[j] + s[j+1]
                o = s[j+2]
                q = s[j+3:s.index(')')]
            else:
                p = s[j]
                o = s[j+1]
                q = s[j+2:s.index(')')]
            temp = '~(' + p + o + q + ')'
            temp = removeBrackets(temp)
            p = negation(p,q,o)
            s = s.replace(temp,p)
        if '~~' in s:
            s = s.replace('~~',"")
        return s
    else:
        x += s[i]


def removeNegation(s):
    while '~(' in s:
        s = simple(s)
        s = removeExtra(s)
    return s
removeNegation("~(p,q)")
def backwardBracketMatch(s):
    a ,b = 1 ,0
    for i in range(len(s)-2,-1,-1):
        if s[i] == ')':
            a+=1
        if s[i] == '(':
            b+=1
        if a == b:
            return i
    return s

# print(backwardBracketMatch("^s(p->q)"))
def forwarbracketMatch(s,index,i):
    a, b = 1,0
    for j in range(i+1, len(s)):
        if s[j] == '(':
            a+=1
        if s[j] == ')':
            b+=1
        if a == b:
            return j
# print(forwarbracketMatch('((s->p)->r)->x','(',0))

def removeImplies(p):    # p = premis
    temp = ""
    if not '->' in p: return p
    i = 0
    l = len(p)
    while( i < l):
```

```
        if p[i] == '(':
            crtl = forwarbracketMatch(p,p[i], i)
            x = p[i+1:crtl]
            i = crtl
            temp = p[:i+1]
            if '->' in x:
                y = removeImplies(x)
                p = p.replace(x, y)
            else:
                temp = p[:i]
                continue
            if '~~' in p:
                p = p.replace('~~',"")
            return removeImplies(p)
        if p[i] == '-' and p[i+1] == '>':
            p = p.split('->',1)
            p = '~' + p[0] + ',' + p[1]
            if '~~' in p:
                p = p.replace('~~',"")
            return removeImplies(p)
        if((p[i] >= chr(65) <= chr(90)) or (p[i] >= chr(97) <= chr(122))) and (p[i+2] == '>'):
            p = p.split('->',1)
            p = '~' + p[0] + ',' + p[1]
            if '~~' in p:
                p = p.replace('~~',"")
            return removeImplies(p)
        if p[i] == '~' and ((p[i+1] >= chr(65) <= chr(90)) or (p[i+1] >= chr(97) <= chr(122))) and (p[i+3] == '>'):
            p = p.split('->',1)
            p =  p[0][1] + ',' + p[1]
            if '~~' in p:
                p = p.replace('~~',"")
            return removeImplies(p)
        else:
            temp += p[i]
        i+=1
    return p

# print(removeImplies("(~p->q)->(s->r)"))
def resolution(a, b):
    a = a.split(',')
    b = b.split(',')
    x = ""
    y = ""
    for i in range(len(a)):
        for j in range(len(b)):          # checking '~p' in a and 'p' in b (principle of resolution)
            if (a[i][0] == '~' and b[j][0] == a[i][1]) or (b[j][0] == '~' and b[j][1] == a[i][0]):
                a.pop(i)
                b.pop(j)
                if a == []:
                    x = ','.join(b)
                    return x
                if b == []:
                    x = ','.join(a)
                    return x
                x = ','.join(a)
                y = ','.join(b)
                return x + ',' +y
    return None                     #if not found any premis to apply resolution
```

```
# print(resolution('T,R,w',"~T,~S"))

def checkClauses(s):
    i = 1
    n = len(s) + 1
    c = 0
    while s != ['']:                # checking for empty clause
        for i in range(c+1, len(s)):
            x = resolution(s[c],s[i])
            if x != None:
                print( '{:5}C{:2}: {:15} By solving ({}) and ({})'.format("",str(n),x,s[c],s[i]))
                s[c] = x
                s.pop(i)
                n+=1
                break;
        if len(s) == 2 and s[0] == "":
            print( '{:5}C{:2}: {:15} By solving ({}).'.format("",str(n),s[1],s[1]))
            print("\n\t There is an independent clause '{}'. So, this system is invalid.".format(s[0]))
            print('\t\t>-----------------------------------<\n')
            return 0
        if len(s) == 2 and s[1] == "":
            print( '{:5}C{:2}: {:15} By solving ({}).'.format("",str(n),s[0],s[0]))
            print("\n\t There is an independent clause '{}'. So, this system is invalid.".format(s[0]))
            print('\t\t>-----------------------------------<\n')
            return 0
        if len(s) == 2 and s[0] == s[1]:
            print( '{:5}C{:2}: {:15} By solving ({}) and ({})'.format("",str(n),s[0],s[0],s[1]))
            print("\n\t There is an independent clause '{}'. So, this system is invalid.".format(s[0]))
            print('\t\t>-----------------------------------<\n')
            return 0
        if c != len(s) and len(s) != 2:
            c+=1
        else:
            c = 0


    print("\n\t  This system is valid.\n\t>-----------------------<\n")
    return 1
# s = ['~p,q','~r,s','p,r','~q','~s']
# checkClauses(s)


def discrete():
    info  = """INSTRUCTONS:\n ----------------------------\n | Operators |  Functions   |\n ----------------------------
 |    <->    | biconditional|\n |    ->     | implies      |\n |     ^     | conjunction  |\n | ,(comma)  | disjunction  |\n |     ~     | negation
|
 ----------------------------\n1. This code works when the input premises are '.'(dot) seperated
   and the last one will be the conclusion.\n2. Only those logical operators will use which are mentioned in above table.
   Otherwise, it will give you the wrong answer(s).\n3. Run this file on python 3.X kernel.\
n-----------------------------------------------------------------"""
    print(info)
#     s = input("Enter string: ")     # s = string
    s = "L->A.E->~I.A->E.L->~I"
    s = s.replace(" ","")
    for i in s:                 #checking for invalid premises
        if (i >= chr(65) <= chr(90)) or (i >= chr(97) <= chr(122)) or (i in ",().<->~^"):
            if i.upper() in s and i.lower() in s and i not in ",().<->~^" :
                print("( {} ) Next time please enter the same alphabetic case in premises!".format(i.lower()))
```

```
            s = s.upper()
        else:
            print("\nYou put this '{}' character in premises which is unknown for this programme.\n".format(i))
            getData()
    s = s.split('.')     # seperating the premises and conclusion
    for i in range(len(s)):
        if(s[i] == ''):
            s.pop(i)
    print("\n----------------------------------------------------------------------\nQuestion 08:\n")
    for i in range (len(s)):
        print("{:5}{}. {}".format("",str(i+1),s[i]))
    for i in range(len(s)):           # changing premises into clauses
        s[i] = removeNegation(s[i])
        s[i] = removeImplies(s[i])
        s[i] = removeNegation(s[i])
        s[i] = s[i].replace('(',"")
        s[i] = s[i].replace(')',"")
        if '^' in s[i]:
            lst = And(s[i])
            s.pop(i)
            for j in lst:
                s.insert(i, j)
    print("\nSolution:\n")
    c = s[-1]                 # taking negation of c = conclusion
    if c[0] == '(':
        c = '~' + c
    else:
        c = '~(' + c + ')'
    c = removeNegation(c)
    c = c.replace('(','')
    c = c.replace(')','')
    s.pop(-1)

    if '^' in c:
        c = And(c)
        s.append(c[0])
        s.append(c[1])
    else:
        s.append(c)

    for i in range(len(s)):     # print simplified clauses
        print("{:5}C{:2}: {:15} By simplification".format("",str(i+1),s[i]), )
    print('')

    #  implimentation of resolution principle
    checkClauses(s)

discrete()
```

END