Discrete Sturctures
Assignment 02

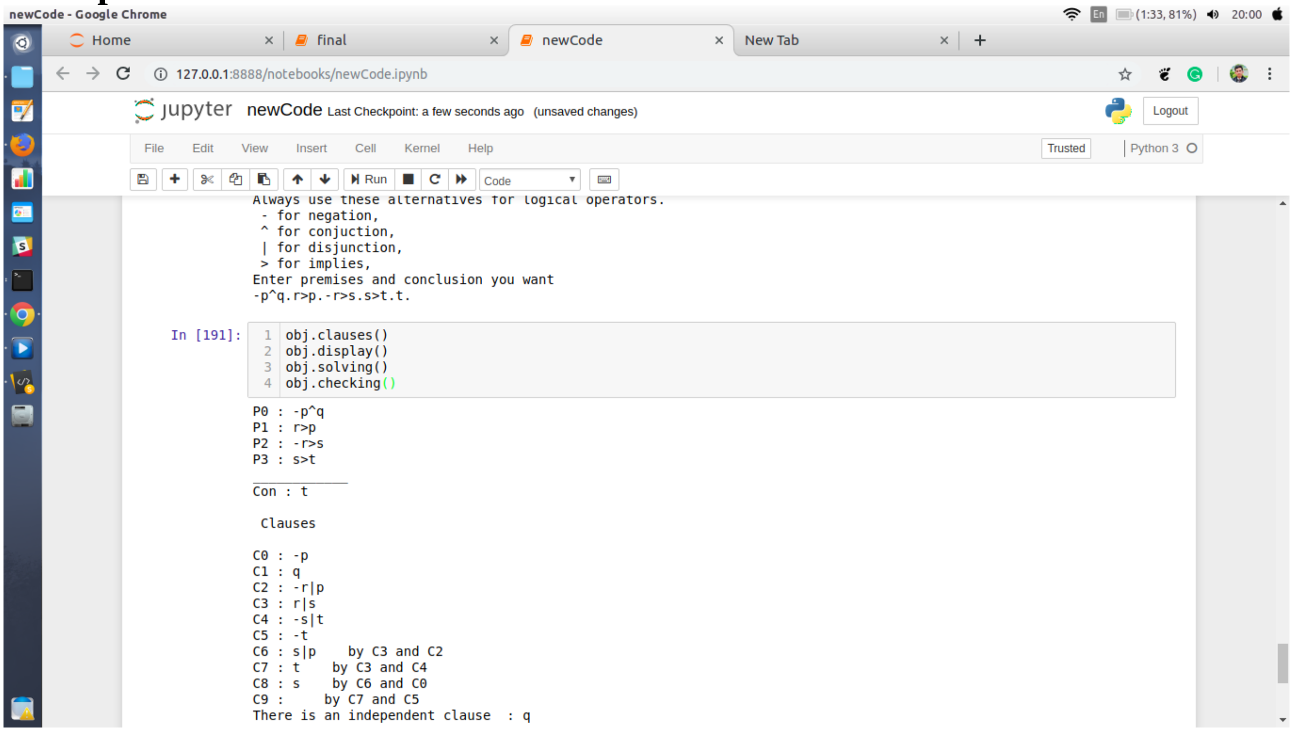
From:

H . M. Mehmood Munir
P17-6075 (B)

To:

Dr. Nauman Azam

Example 6:


newCode - Google Chrome
Home
final
newCode
New Tab
127.0.0.1:8888/notebooks/newCode.ipynb
jupyter newCode Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code

```
Always use these alternatives for logical operators.
 - for negation,
 ^ for conjuction,
 | for disjunction,
 > for implies,
Enter premises and conclusion you want
-p^q.r>p.-r>s.s>t.t.
```

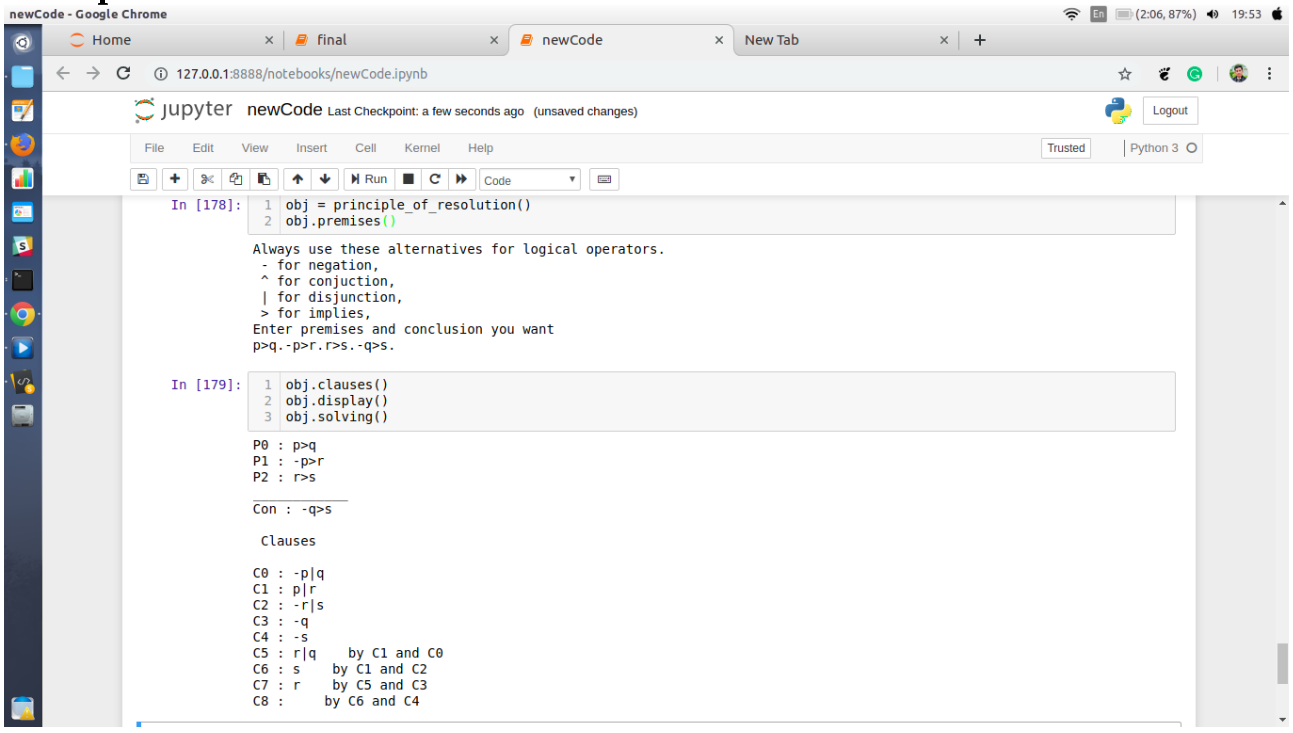
In [191]:
```
obj.clauses()
obj.display()
obj.solving()
obj.checking()
```

```
P0 : -p^q
P1 : r>p
P2 : -r>s
P3 : s>t
____________
Con : t

 Clauses

C0 : -p
C1 : q
C2 : -r|p
C3 : r|s
C4 : -s|t
C5 : -t
C6 : s|p    by C3 and C2
C7 : t    by C3 and C4
C8 : s    by C6 and C0
C9 :     by C7 and C5
There is an independent clause  : q
```

Example 07:


newCode - Google Chrome
Home
final
newCode
New Tab
127.0.0.1:8888/notebooks/newCode.ipynb
jupyter newCode Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code

In [178]:
```
obj = principle_of_resolution()
obj.premises()
```

```
Always use these alternatives for logical operators.
 - for negation,
 ^ for conjuction,
 | for disjunction,
 > for implies,
Enter premises and conclusion you want
p>q.-p>r.r>s.-q>s.
```

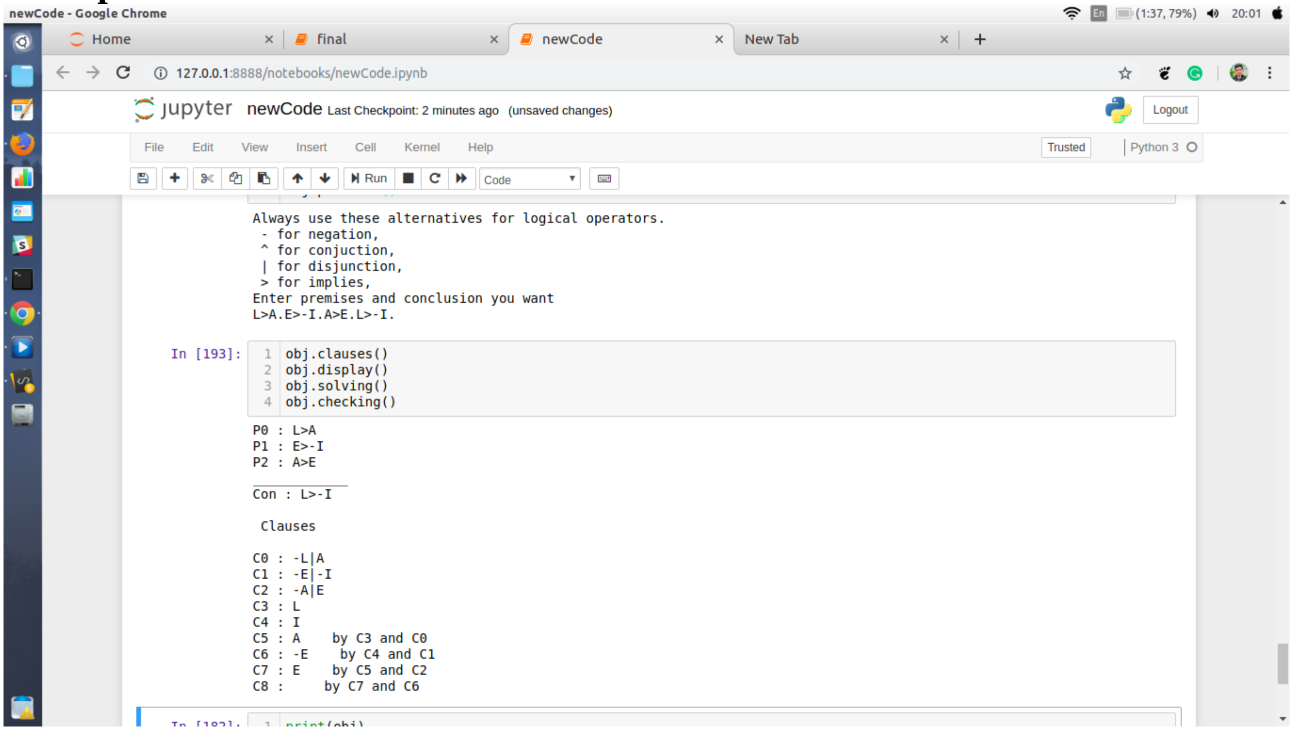
In [179]:
```
obj.clauses()
obj.display()
obj.solving()
```

```
P0 : p>q
P1 : -p>r
P2 : r>s
____________
Con : -q>s

 Clauses

C0 : -p|q
C1 : p|r
C2 : -r|s
C3 : -q
C4 : -s
C5 : r|q    by C1 and C0
C6 : s    by C1 and C2
C7 : r    by C5 and C3
C8 :     by C6 and C4
```

Example 09:


newCode - Google Chrome
Home
final
newCode
New Tab
127.0.0.1:8888/notebooks/newCode.ipynb
jupyter newCode Last Checkpoint: 2 minutes ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3
Run
Code

```
Always use these alternatives for logical operators.
 - for negation,
 ^ for conjuction,
 | for disjunction,
 > for implies,
Enter premises and conclusion you want
L>A.E>-I.A>E.L>-I.
```

```
In [193]: obj.clauses()
          obj.display()
          obj.solving()
          obj.checking()
```

```
P0 : L>A
P1 : E>-I
P2 : A>E
____________
Con : L>-I

 Clauses

C0 : -L|A
C1 : -E|-I
C2 : -A|E
C3 : L
C4 : I
C5 : A    by C3 and C0
C6 : -E    by C4 and C1
C7 : E    by C5 and C2
C8 :     by C7 and C6
```

Code:

```
class principle_of_resolution:
    def __init__(self):
        self.premiseList = []
        self.clauseList = []
        self.independentClause = []
        self.preConclusion = ''
        self.conclusion = ''

    def __str__(self):
        return str("Premises : "+str(self.premiseList)+ '\n'+"Clauses : "+str(self.clauseList) +'\n'+"conclusion : "+str(self.conclusion))



    def display(self):
        for i in range(len(self.premiseList)):
            print("P"+str(i)+" : "+self.premiseList[i])
        print("____________")
        print("Con : "+self.preConclusion)


        print("\n Clauses\n")
        for i in range(len(self.clauseList)):
            print("C"+str(i)+" : "+self.clauseList[i])


    #function to take input from user and convert it to premise and conclusion
    def premises(self):
        print("Always use these alternatives for logical operators.\n - for negation, \n ^ for conjuction,\n | for
disjunction, \n > for implies, ")
        print("Enter premises and conclusion you want ")
        n = input()[:-1]                                    #taking input
        self.premiseList = n.split('.')                     #getting premises into list form
        self.preConclusion =  self.premiseList[-1]   #taking out conclusion from premises
        self.conclusion = "-(" +self.preConclusion + ")"
        del self.premiseList[-1]                            #delete the conclusion



    #function that resolve the premises into clauses
    def clauses(self):
        for i in self.premiseList:
            if ">" in i:
                prem = self.implies(i)
                self.clauseList.append(prem)
            elif "^" in i:
                self.conjuction(i)
            else:
                self.clauseList.append(prem)
        self.conclusion = self.negation(self.conclusion)
        if len(self.premiseList) != 0:
            if ">" in self.conclusion:
                prem = self.implies(self.conclusion)
                self.clauseList.append(prem)
            elif "^" in self.conclusion:
```

```
            self.conjuction(self.conclusion)
        else:
            self.clauseList.append(self.conclusion)
    else:
        return "you have no conclusion"

#function that applies the conjunction method
def conjuction(self, prem):
    self.clauseList += prem.split('^')



#function that applies the implication method
def implies(self, prem):
    if prem[0] == "-":
        prem = prem.replace('-', "",1)
        prem = prem.replace('>', "|")
        return prem
    prem = '-' + prem.replace('>','|')
    return prem



#function that take premise as input and find negation of this premise and output will be produced
def negation(self,prem):
    prem = prem.replace('-', '',1)
    if '>' in prem:



        #applying p>q = -p|q
        if prem[prem.index('(')+1] == '-':
            prem = prem.replace('-', '',1)
        elif prem[prem.index('(')+1] != '-':
            prem = self.insert_char(prem,prem.index('(')+1)
        prem = prem.replace('(',"")
        prem = prem.replace(')',"")
        prem = prem.replace('>','|')
        #applying demorgan Law in case of implies
        if prem[0] == '-':
            prem = prem.replace('-', '',1)
        elif prem[0] != '-':
            prem = self.insert_char(prem,0)
        prem = prem.replace('|','^')
        if prem[prem.index('^')+1] == '-':
            prem = self.del_char(prem, prem.index("^")+1 )
        elif prem[prem.index('^')+1] != '-':
            prem = self.insert_char(prem,prem.index('^')+1)
        return prem

    elif '^' in prem:

        #applying demorgan Law in case of conjunction
        if prem[(prem.index('(')+1)] == '-':
            prem = prem.replace('-', '',1)
        elif prem[prem.index('(')+1] != '-':
            prem = self.insert_char(prem,prem.index('(')+1)
        prem = prem.replace('(',"")
        prem = prem.replace(')',"")
        prem = prem.replace('^','|')
        if prem[(prem.index('|'))+1] == '-':
            prem = self.del_char(prem, prem.index("|")+1 )
        elif prem[prem.index('|')+1] != '-':
            prem = self.insert_char(prem,prem.index('|')+1)
        return prem

    elif '|' in prem:
        #applying demorgan Law in case of disjunction
        if prem[prem.index('(')+1] == '-':
            prem =prem.replace('-', '', 1)
        elif prem[prem.index('(')+1] != '-':
                prem = self.insert_char(prem,prem.index('(')+1)
            prem =prem.replace('(',"")
            prem =prem.replace(')',"")
            prem =prem.replace('|','^')
            if prem[(prem.index('^'))+1] == '-':
                prem = self.del_char(prem, prem.index("^")+1 )
            elif prem[(prem.index('^')+1)] != '-':
                prem = self.insert_char(prem,prem.index('^')+1)
            return prem
        else:
            prem = "-" + prem
            if "(" in prem:
                prem =prem.replace('(',"")
                prem =prem.replace(')',"")
            return prem



    #function to insert a character in a string you have to give index and  string  in argument and character as well if you
want by default function inserts hash('-')
    def insert_char(self,string, index, char = '-'):
        return string[:index] + char + string[index:]


    #function to delete a character from a string you have to give index and string as input in the function parameters it
will return
    def del_char(self,string, index):
        return string[:index] + string[index+1:]





    def solving(self):
        counterClause = len(self.clauseList)
        for i in range(counterClause+3):
            lst = self.clauseList[i].split('|')
            for l in lst:
                if len(l) == 1:
                    for j in range(len(self.clauseList)):
```

```
if i == j:
    continue
elif l in self.clauseList[j]:
    for k in range(len(self.clauseList[j])):
        if self.clauseList[j][k] == l:
            if k-1 >= 0 and self.clauseList[j][k-1] == '-' :
                self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j][k-1]+self.clauseList[j][k],'')
                self.clauseList[i] = self.clauseList[i].replace(l,'')
                if len(lst) == 2:
                    if l == lst[0]:
                        clause = lst[1] + self.clauseList[j]
                        print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                        self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],'')
                        self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],'')
                        self.clauseList.append(clause)
                        counterClause += 1

                        lst = []
                        break
                    else:
                        clause = lst[0] + self.clauseList[j]
                        print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                        self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],'')
                        self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],'')
                        self.clauseList.append(clause)
                        counterClause += 1
                        lst = []
                        break
                else:
                    self.clauseList[j] = self.clauseList[j].replace('|','')
                    clause = self.clauseList[j]
                    print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                    self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],'')
                    self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],'')
                    self.clauseList.append(clause)
        counterClause += 1
        lst = []

        break
else:
    if len(self.clauseList[j]) == 1:
        self.clauseList[i].replace(l,")
        if len(lst) == 2:
            if l == lst[0]:
                clause = lst[1] +"|"+self.clauseList[j]
                print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],")
                self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],")
                self.clauseList.append(clause)
                counterClause += 1
                lst = []
                break
            else:
                clause = lst[0] +"|"+self.clauseList[j]
                print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],")
                self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],")
                self.clauseList.append(clause)
                counterClause += 1
                lst = []

                break
        else:
            clause = self.clauseList[j]
            print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
            self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],")
            self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],")
            self.clauseList.append(clause)
            counterClause += 1
            lst = []
```

```
                                break
                        elif len(self.clauseList[j]) > 2 and len(lst) == 2:
                            lst = []
                            break
                        elif len(self.clauseList[j]) > 2 and len(lst) == 1:
                            clause = self.clauseList[j]
                            print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                            self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],'')
                            self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],'')
                            self.clauseList.append(clause)
                            counterClause += 1
                            lst = []

                            break
        else:
            continue

elif len(l) == 2:
    for j in range(len(self.clauseList)):
        if i == j:
            continue
        elif l in self.clauseList[j]:
            for k in range(len(self.clauseList[j])):
                if self.clauseList[j][k] == l[1]:
                    if k-1 >= 0 and self.clauseList[j][k-1] == '-' :
                        self.clauseList[i] = self.clauseList[i].replace(l,'')
                        if len(lst) == 1 and len(self.clauseList[j]) == 2:
                            clause = lst[0]
                            print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                            self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],'')
                            self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],'')
                            self.clauseList.append(clause)
                            counterClause += 1
                            lst = []
        break
    elif len(lst) == 2 and len(self.clauseList[j]) == 2:
            clause = lst[0] +'|'+lst[1]
            print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
            self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],")
            self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],")
            self.clauseList.append(clause)
            counterClause += 1
            lst = []

            break
    else:
        lst = []
        break
else:
    if len(self.clauseList[j]) == 1:
        self.clauseList[i] = self.clauseList[i].replace(l,")
        if len(lst) == 2:
            if l == lst[0]:
                clause = lst[1]
                print(clause)
                print('C'+str(clauseCounter)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],")
                self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],")
                self.clauseList.append(clause)
                counterClause += 1
                lst = []

                break
            else:
                clause = lst[0]
                print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],")
                self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],")
            self.clauseList.append(clause)
            counterClause += 1
            lst = []

            break
    else:
        clause = ''
        print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
        self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],'')
        self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],'')
        self.clauseList.append(clause)
        counterClause += 1
        lst = []

        break
elif len(self.clauseList[j]) > 2 and len(lst) == 2:
    self.clauseList[i] = self.clauseList[i].replace(l,'')
    self.clauseList[j] = self.clauseList[j].replace(l[1],'')
    self.clauseList[j] = self.clauseList[j].replace("|",'')
    if l == lst[0]:
        clause = lst[1]+"|"+self.clauseList[j]
        print('C'+str(clauseCounter)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
        self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],'')
        self.clauseList.append(clause)
        counterClause += 1
        lst = []

        break
    else:
        clause = lst[0]+"|"+self.clauseList[j]
        print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
        self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],'')
        self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],'')
        self.clauseList.append(clause)
        counterClause += 1
```

```
                                            lst = []
                                            break

                                    elif len(self.clauseList[j]) > 2 and len(lst) == 1:
                                        clause = self.clauseList[j]
                                        print('C'+str(counterClause)+" : "+clause + "    by C"+str(i)+" and C"+str(j))
                                        self.clauseList[i] = self.clauseList[i].replace(self.clauseList[i],")
                                        self.clauseList[j] = self.clauseList[j].replace(self.clauseList[j],")
                                        self.clauseList.append(clause)
                                        counterClause += 1
                                        lst = []

                                        break
                    else:
                        continue
                break
            else:
                continue



def checking(self):
    for i in self.clauseList:
        if i != ":
            print("There is an independent clause  : "+i)
            break
```