KB ENTAILS QUERY

kb = []

def CLEAR():

global kb

kb = []

def TELL(sentence):

global kb

# If the sentence is a clause, insert directly.

if isClause(sentence):

kb.append(sentence)

# If not, convert to CNF, and then insert clauses one by one.

else:

sentenceCNF = convertCNF(sentence)

if not sentenceCNF:

print("Illegal input")

return

# Insert clauses one by one when there are multiple clauses

if isAndList(sentenceCNF):

for s in sentenceCNF[1:]:

kb.append(s)

else:

kb.append(sentenceCNF)

def ASK(sentence):

global kb

# Negate the sentence, and convert it to CNF accordingly.

if isClause(sentence):

neg = negation(sentence)

else:

sentenceCNF = convertCNF(sentence)

if not sentenceCNF:

print("Illegal input")

return

neg = convertCNF(negation(sentenceCNF))

# Insert individual clauses that we need to ask to ask\_list.

ask\_list = []

if isAndList(neg):

for n in neg[1:]:

nCNF = makeCNF(n)

if type(nCNF).\_\_name\_\_ == 'list':

ask\_list.insert(0, nCNF)

else:

ask\_list.insert(0, nCNF)

else:

ask\_list = [neg]

clauses = ask\_list + kb[:]

while True:

new\_clauses = []

for c1 in clauses:

for c2 in clauses:

if c1 is not c2:

resolved = resolve(c1, c2)

if resolved == False:

continue

if resolved == []:

return True

new\_clauses.append(resolved)

if len(new\_clauses) == 0:

return False

new\_in\_clauses = True

for n in new\_clauses:

if n not in clauses:

new\_in\_clauses = False

clauses.append(n)

if new\_in\_clauses:

return False

return False

def resolve(arg\_one, arg\_two):

resolved = False

s1 = make\_sentence(arg\_one)

s2 = make\_sentence(arg\_two)

resolve\_s1 = None

resolve\_s2 = None

# Two for loops that iterate through the two clauses.

for i in s1:

if isNotList(i):

a1 = i[1]

a1\_not = True

else:

a1 = i

a1\_not = False

for j in s2:

if isNotList(j):

a2 = j[1]

a2\_not = True

else:

a2 = j

a2\_not = False

# cancel out two literals such as 'a' $ ['not', 'a']

if a1 == a2:

if a1\_not != a2\_not:

# Return False if resolution already happend

# but contradiction still exists.

if resolved:

return False

else:

resolved = True

resolve\_s1 = i

resolve\_s2 = j

break

# Return False if not resolution happened

if not resolved:

return False

# Remove the literals that are canceled

s1.remove(resolve\_s1)

s2.remove(resolve\_s2)

# # Remove duplicates

result = clear\_duplicate(s1 + s2)

# Format the result.

if len(result) == 1:

return result[0]

elif len(result) > 1:

result.insert(0, 'or')

return result

def make\_sentence(arg):

if isLiteral(arg) or isNotList(arg):

return [arg]

if isOrList(arg):

return clear\_duplicate(arg[1:])

return

def clear\_duplicate(arg):

result = []

for i in range(0, len(arg)):

if arg[i] not in arg[i+1:]:

result.append(arg[i])

return result

def isClause(sentence):

if isLiteral(sentence):

return True

if isNotList(sentence):

if isLiteral(sentence[1]):

return True

else:

return False

if isOrList(sentence):

for i in range(1, len(sentence)):

if len(sentence[i]) > 2:

return False

elif not isClause(sentence[i]):

return False

return True

return False

def isCNF(sentence):

if isClause(sentence):

return True

elif isAndList(sentence):

for s in sentence[1:]:

if not isClause(s):

return False

return True

return False

def negation(sentence):

if isLiteral(sentence):

return ['not', sentence]

if isNotList(sentence):

return sentence[1]

# DeMorgan:

if isAndList(sentence):

result = ['or']

for i in sentence[1:]:

if isNotList(sentence):

result.append(i[1])

else:

result.append(['not', sentence])

return result

if isOrList(sentence):

result = ['and']

for i in sentence[:]:

if isNotList(sentence):

result.append(i[1])

else:

result.append(['not', i])

return result

return None

def convertCNF(sentence):

while not isCNF(sentence):

if sentence is None:

return None

sentence = makeCNF(sentence)

return sentence

def makeCNF(sentence):

if isLiteral(sentence):

return sentence

if (type(sentence).\_\_name\_\_ == 'list'):

operand = sentence[0]

if isNotList(sentence):

if isLiteral(sentence[1]):

return sentence

cnf = makeCNF(sentence[1])

if cnf[0] == 'not':

return makeCNF(cnf[1])

if cnf[0] == 'or':

result = ['and']

for i in range(1, len(cnf)):

result.append(makeCNF(['not', cnf[i]]))

return result

if cnf[0] == 'and':

result = ['or']

for i in range(1, len(cnf)):

result.append(makeCNF(['not', cnf[i]]))

return result

return "False: not"

if operand == 'implies' and len(sentence) == 3:

return makeCNF(['or', ['not', makeCNF(sentence[1])], makeCNF(sentence[2])])

if operand == 'biconditional' and len(sentence) == 3:

s1 = makeCNF(['implies', sentence[1], sentence[2]])

s2 = makeCNF(['implies', sentence[2], sentence[1]])

return makeCNF(['and', s1, s2])

if isAndList(sentence):

result = ['and']

for i in range(1, len(sentence)):

cnf = makeCNF(sentence[i])

# Distributivity:

if isAndList(cnf):

for i in range(1, len(cnf)):

result.append(makeCNF(cnf[i]))

continue

result.append(makeCNF(cnf))

return result

if isOrList(sentence):

result1 = ['or']

for i in range(1, len(sentence)):

cnf = makeCNF(sentence[i])

# Distributivity:

if isOrList(cnf):

for i in range(1, len(cnf)):

result1.append(makeCNF(cnf[i]))

continue

result1.append(makeCNF(cnf))

# Associativity:

while True:

result2 = ['and']

and\_clause = None

for r in result1:

if isAndList(r):

and\_clause = r

break

# Finish when there's no more 'and' lists

# inside of 'or' lists

if not and\_clause:

return result1

result1.remove(and\_clause)

for i in range(1, len(and\_clause)):

temp = ['or', and\_clause[i]]

for o in result1[1:]:

temp.append(makeCNF(o))

result2.append(makeCNF(temp))

result1 = makeCNF(result2)

return None

return None

def isLiteral(item):

if type(item).\_\_name\_\_ == 'str':

return True

return False

def isNotList(item):

if type(item).\_\_name\_\_ == 'list':

if len(item) == 2:

if item[0] == 'not':

return True

return False

def isAndList(item):

if type(item).\_\_name\_\_ == 'list':

if len(item) > 2:

if item[0] == 'and':

return True

return False

def isOrList(item):

if type(item).\_\_name\_\_ == 'list':

if len(item) > 2:

if item[0] == 'or':

return True

return False

CLEAR()

TELL('p')

TELL(['implies', ['and', 'p', 'q'], 'r'])

TELL(['implies', ['or', 's', 't'], 'q'])

TELL('t')

TELL('s')

print(ASK('r'))