4/1/23

Lab -7 - Proof by Resolution.

Aim: To create a knowledge base using propositional logic and prove the given query using resolu

Algorithm:

function PL-Resolution (KB, α) returns true or false
inputs: KB, the knowledge base, a sentence in propositional logic, α;
the query, a sentence in propositional logic

clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
new ← {}
loop do
for each pair of clauses C_i, C_j in clauses do
resolvents ← PL_RESOLVE (C_i, C_j)
if resolvents contains the empty clause
then return true
new ← new ∪ resolvents.
if new ⊆ clauses then return false
clauses ← clauses ∪ new

Code

code:

```
import re
def negate(term):
    return f'~{term}' if term[0] != '~' else
                                        term[1]

def reverse(clause):
    if len(clause) > 2:
        t = split_terms(clause)
        return f'{t[1]} v {t[0]}'
    return ''

def split_terms(rule):
    exp = '(~*[PQRS])'
    terms = re.findall(exp, rule)
    return terms

def contradiction(query, clause):
    contradictions = [f'{query} v {negate(query)}',
                      f'{negate(query)} v {query}']

    return clause in contradictions or reverse(
                clause) in contradictions

def resolve(kb, query):
    temp = kb.copy()
    temp += [negate(query)]
    steps = dict()
    for rule in temp:
        steps[rule] = 'Given'
    steps[negate(query)] = 'Negated conclusion'
    i = 0
    while(i < len(temp)):
        n = len(temp)
```

in while

```
j = (i+1)%n
clauses = []
while j != i:
    terms1 = split_terms(temp[i])
    terms2 = split_terms(temp[j])
    for c in terms1:
        if negate(c) in terms2:
            t1 = [t for t in terms1
                if t != c]
            t2 = [t for t in terms2
            if t != negate(c)]
            gen = t1 + t2
            if len(gen) == 2:
                if gen[0] != negate(gen[1]):
                    clauses += [f'{gen[0]} v {gen[1]}']
                else:
                    if contradiction(query,
f'{gen[0]} v {gen[1]}'):
                        temp.append(
f'{gen[0]} v {gen[1]}')
                        steps[i] = f"
Resolved {temp[i]} and {temp[j]} to {temp[-1]}
which is in turn null. \nA contradiction is
found when {negate(query)} is assumed as
true. Hence, {query} is true."
                        return steps
            elif len(gen) == 1:
                clauses += [f'{gen[0]}']
            else:
                if contradiction(query,f'
{terms1[0]} v {terms2[0]}'):
                    temp.append(f'{terms1
[0]} v {terms2[0]}')
```

under if contradiction

steps[' '] = f" resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in turn null. A contradiction is found when {negate(query)} is assumed as true. Hence, {query} is true".

return steps

under main while

for clause in clauses:
    if clause not in temp and clause != reverse(clause) not in temp:
        temp.append(clause)
        steps[clause] = f'resolved from {temp[i]} and {temp[j]}.'

under main while

j = (j+1) % n

i += 1

return steps

```
def resolution(kb, query):
    kb = kb.split(' ')
    steps = resolve(kb, query)
    print('\nStep\t|Clause\t|Derivation\t')
    print('-' * 30)
    i = 1
    for step in steps:
        print(f'{i}.\t|{step}\t|{steps[step]}\t')
        i += 1

def main():
    print("Enter the kb:")
    kb = input()
    print("Enter the query:")
    query = input()
    resolution(kb, query)
main()
```

output: Enter the KB:
~P v ~Q v R P ~S v T v Q S
Enter the query:
R.

step	Clause	Derivation.
1.	~P v ~Q R	Given.
	P	Given.
	~S v ~T v Q	Given.
	S	Given.
	~R	Negated conclusion.
	~Q v R	Resolved from ~P v ~Q v R and P.
	~P v ~Q	Resolved from ~P v Q v R and ~R.
	~Q	Resolved from P and ~P v ~Q.
	Q	Resolved from ~S v ~T v Q and S
	~S v R	Resolved from ~S v ~T v Q and ~P.
	~S	Resolved from ~S v ~T v Q ~~S v R~~ ~Q.
	R	Resolved from S and ~S v R
	~P	Resolved from S and ~S v ~P.
	)	Resolved ~R and R to R~R,

which is in turn null.
A contradiction is found when ~R is assumed as true. Hence R is true.