03-12-2024 WEEK-8 KNOWLEDGE BASE USING RESOLUTION

ALGORITHM-

```
ALGORITHM:

REJOLUTION (KB, R):

Negate the avery and add it as a new clause to tB

Instalize an empty set of new clauses

While True:

Instalize an empty set of resolved clauses

For each pair of clauses (C1, (2) in KB:

RESOlvent: Resolve (U, (2))

If Resolvent is None:

Remain "Resolve"

If enoivent is nor empty:

add resolvent to out of resolved clauses

If no new clauses are generated

Remain "Not pronen"
```

```
Add all resolved leaves to EB.

Resolve LCI. (?):

tor each lincl:

If a comprimentary lieral (nlorl)exists

in C2:

Remove L from CI and its complement

prom C?.

combine remaining lierals from CI and C2 into

a new clause.

If new clause is empty:

Return none

else:

Return Mone.
```

CODE-

```
def resolve(clause1, clause2):
  resolved = set()
  for literal in clause1:
     if f"~{literal}" in clause2 or (literal.startswith("~") and literal[1:] in clause2):
        temp1 = clause1 - {literal}
        temp2 = clause2 - \{f''\sim\{literal\}'' \text{ if not literal.startswith}("\sim") \text{ else literal}[1:]\}
        resolved = temp1.union(temp2)
        if not resolved:
          return None return resolved
  return None
def resolution(kb, query):
  negated query = \{f'' \sim \{query\}''\}
  kb.append(negated query)
  new clauses = set()
  while True:
     pairs = combinations(kb, 2)
     for clause1, clause2 in pairs:
        resolvent = resolve(clause1, clause2)
```

```
if resolvent is None:
         return True # Query proven
       if resolvent:
         new clauses.add(frozenset(resolvent))
    if new clauses.issubset(set(kb)):
       return False # No progress, query not proven
    kb.extend([set(clause) for clause in new clauses])
# Define the KB in clausal form
kb = [
  {"~Food(x)", "Likes(John,x)"}, # John likes all food
  {"Food(Apple)"},
                              # Apple is food
  {"Food(Vegetables)"},
                               # Vegetables are food
  {\text{"-Eats}(x,y)", "Killed(x)", "Food(y)"}, # Eats and not killed -> food
                               # Anil eats peanuts
  {"Eats(Anil,Peanuts)"},
                             # Anil is not killed
  {"~Killed(Anil)"},
  {"~Eats(Anil,y)", "Eats(Harry,y)"}, # Anil eats -> Harry eats
query = "Likes(John,Peanuts)"
result = resolution(kb, query)
if result:
  print(f"The query '{query}' is PROVEN.")
else:
  print(f"The query '{query}' is NOT PROVEN.")
```

OUTPUT-

The query 'Likes(John, Peanuts)' is PROVEN.

PROOF TREE-

	PROOF TREE:
	- Uker (John, Peanuts) - 100(x) V (ker (John, x)
	The state of the s
	- Food (Peanur) - ear (4,2) v killed (4) V bood (2)
2	the second second of the second second second
	7 eatr (4, Peanute) v willed (4) care (Anse, Peanute)
	Mains & Transmission 199
	uelled (Anie) ralive (K) V- willed (K)
	ralive (Anie) odise (Anil)
	to the same was a second of the same a second of th
	d usua (John) peanuts) peance proved.
	- rugrug