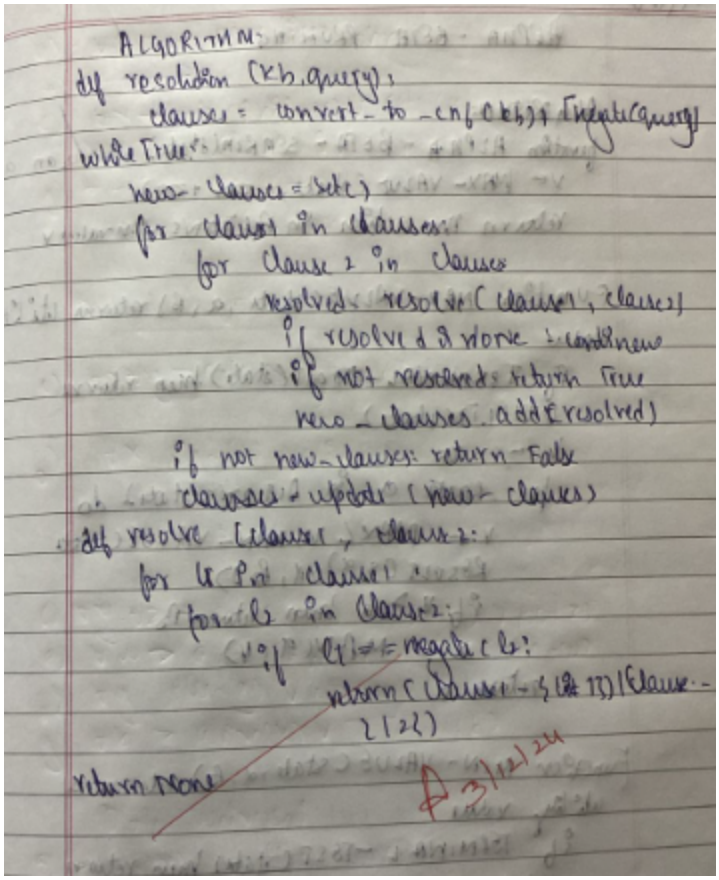
Date:3/12/24

Program Title: Create a knowledge base consisting of first order logic statements and prove the given query using Resolution

Algorithm:



ALGORITHM:

```
def resolution (kb, query):
    clauses = convert_to_cnf(kb) + [negate(query)]
while True:
    new_clauses = set()
    for clause1 in clauses:
        for clause2 in clauses
            resolvent = resolve(clause1, clause2)
                if resolvent == [] : continue
                if not resolvent: return True
                new_clauses.add(resolvent)
        if not new_clauses: return False
        clauses = update(new_clauses)
def resolve(clause1, clause2):
    for l1 in clause1:
        for l2 in clause2:
            if l1 == negate(l2):
                return (clause1 - {l1}) | (clause2 - {l2})
return None
```

3/12/24

5/12/24

For resolution

a) John likes all kind of food
$\forall x\ food(x) \rightarrow likes(John, x)$

b) Apple and vegetables are food
$food(Apple) \wedge food(vegetables)$

c) Anything anyone eats and not killed is food
$\forall x \forall y\ eats(x, y) \wedge \neg killed(x) \rightarrow food(y)$

d) Anil eats peanuts and still alive
$eats(Anil, peanuts) \wedge alive(Anil)$

e) Harry eats everything that Anil eats
$\forall x\ eats(Anil, x) \rightarrow eats(Harry, x)$

f) Anyone who is alive implies not killed
$\forall x\ g \rightarrow killed(x) \rightarrow alive(x)$

g) $\forall x\ alive(x) \rightarrow \neg killed(x)$ Anyone who is not killed implies alive

h) Prove John likes peanuts likes(John, Peanuts)

a) $\forall x\ \neg food(x) \vee likes(John, x)$
~~b) food(A)~~
c) $\forall x \rightarrow eats(Anil, x) \vee eats(Harry, x)$
f) $\forall x\ \neg[\neg killed(x)] \vee alive(x)$
g) $\forall \rightarrow alive(x) \vee \neg killed(x)$
rest remain same

Inward negation

e) $\forall x \forall y\ \neg eats(x, y) \vee killed(x) \vee food(y)$

f) $\forall x\ killed(x) \vee alive(x)$

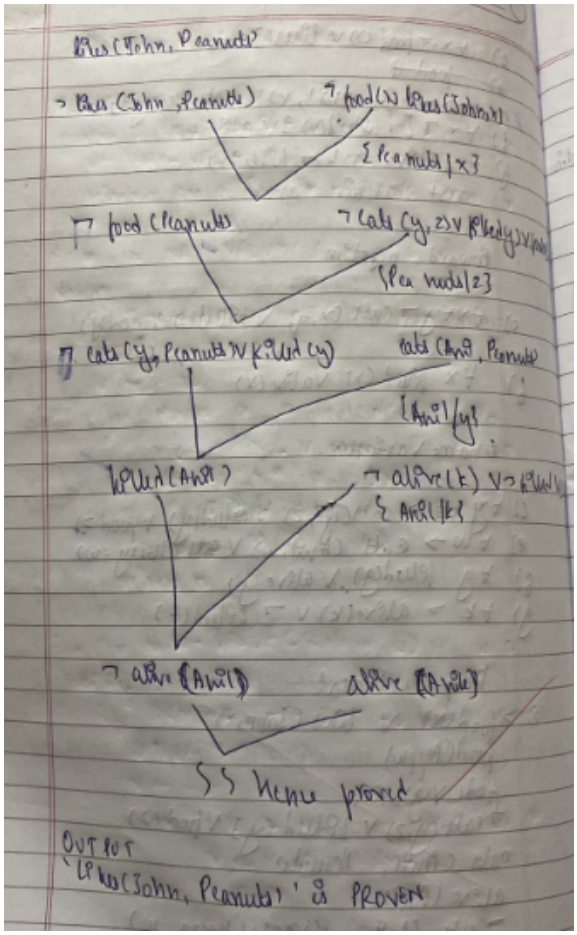
rename variables.

e) $\forall y\ \forall z \rightarrow eats(y, z) \vee killed(y) \vee food(z)$
e) $\forall w \rightarrow eats(Anil, w) \vee eats(Harry, w)$
f) $\forall g\ killed(g) \vee alive(g)$
g) $\forall k \rightarrow alive(k) \vee \neg killed(k)$

Drop universal

a) $\rightarrow food(x) \vee likes(John, x)$
$food(Apple)$
$food(vegetables)$
$\rightarrow eats(y, z) \vee killed(y) \vee food(z)$
$eats(Anil, Peanuts)$
$alive(Anil)$
$\rightarrow eats(Anil, w) \vee eats(Harry, w)$
$killed(g) \vee alive(k)$

Resolution Tree:



Code:

```python
from itertools import combinations
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"~{literal}" if not literal.startswith("~") else literal[1:]}
            resolved = temp1.union(temp2)
            if not resolved:
                return None
            return resolved
    return None
def resolution(kb, query):
    negated_query = {f"~{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
            if resolvent:
                new_clauses.add(frozenset(resolvent))
        if new_clauses.issubset(set(map(frozenset, kb))):
            return False
        kb.extend([set(clause) for clause in new_clauses])
kb = [
    {"~Food(x)", "Likes(John,x)"},
    {"Food(Apple)"},
    {"Food(Vegetables)"},
    {"~Eats(x,y)", "Killed(x)", "Food(y)"},
    {"Eats(Anil,Peanuts)"},
    {"~Killed(Anil)"},
    {"~Eats(Anil,y)", "Eats(Harry,y)"},
]
query = "Likes(John,Peanuts)"
result = resolution(kb, query)
if result:
    print(f"'{query}' is PROVEN.")
else:
    print(f"'{query}' is NOT PROVEN.")
```


from itertools import combinations

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"~{literal}" if not literal.startswith("~") else literal[1:]}

resolved = temp1.union(temp2)

if not resolved:

return None

return resolved

```
    return None
def resolution(kb, query):
    negated_query = {f"~{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
            if resolvent:
                new_clauses.add(frozenset(resolvent))
        if new_clauses.issubset(set(map(frozenset, kb))):
            return False
        kb.extend([set(clause) for clause in new_clauses])
kb = [
    {"~Food(x)", "Likes(John,x)"},
    {"Food(Apple)"},
    {"Food(Vegetables)"},
    {"~Eats(x,y)", "Killed(x)", "Food(y)"},
    {"Eats(Anil,Peanuts)"},
    {"~Killed(Anil)"},
    {"~Eats(Anil,y)", "Eats(Harry,y)"},
]
query = "Likes(John,Peanuts)"
result = resolution(kb, query)
if result:
```

```
    print(f"'{query}' is PROVEN.")
else:
    print(f"'{query}' is NOT PROVEN.")
```

Output Snapshot:

```
'Likes(John,Peanuts)' is PROVEN.
```