LAB 8:

Program: Create knowledge base consisting of FOL statements and prove given query using forward reasoning.

Code:

```
class Fact:

def __init__(self, predicate, args=None):

self.predicate = predicate

self.args = args if args else []

def __repr__(self):

return f"{self.predicate}({', '.join(map(str, self.args))})"

class Rule:

def __init__(self, premise, conclusion):

self.premise = premise

self.conclusion = conclusion

def __repr__(self):

return f"IF {' AND '.join(map(str, self.premise))} THEN {self.conclusion}"

class KnowledgeBase:

def __init__(self):

self.facts = set()

self.rules = []

def add_fact(self, fact):

self.facts.add(fact)

def add_rule(self, rule):

self.rules.append(rule)
```

```
def forward_reasoning(self, query):
new_facts = set(self.facts)
derived = set()
while True:
added = False
for rule in self.rules:
if all(premise in new_facts for premise in rule.premise):
if rule.conclusion not in new_facts:
new_facts.add(rule.conclusion)
derived.add(rule.conclusion)
added = True
if not added:
break
return query in new_facts
kb = KnowledgeBase()
kb.add_fact(Fact("Food", ["Banana"]))
kb.add_fact(Fact("Food", ["Pizza"]))
kb.add_fact(Fact("Eats", ["Sam", "Idli"]))
kb.add_fact(Fact("Alive", ["Sam"]))
kb.add_fact(Fact("Enjoys", ["Ravi", "Food"]))
kb.add_rule(Rule([Fact("Eats", ["X", "Y"]), Fact("Alive", ["X"])], Fact("Food", ["Y"])))
kb.add_rule(Rule([Fact("Eats", ["Bill", "X"])], Fact("Eats", ["Sam", "X"])))
query = Fact("Enjoys", ["Ravi", "Idli"])
result = kb.forward_reasoning(query)
```

```
print(f"Can we prove that Ravi likes Idli? {'Yes' if result else 'No'}")
```

26/11/24 Tuesday

Lab-8

Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

Algorithm:

```
class ForwardReasoning:
Initialize:
- knowledge_base = []
- facts = set()
method add_fact(fact):
  Add the given fact to the facts set.
method add_rule(premise, conclusion):
add a rule to the knowledge base.
A rule is an implication of the form:
if premise is true, then conclusion is true.
method forward_reasoning(query):
Initialize:
- new_facts = True
while new_facts is True:
  Set new_facts to False
For each rule in knowledge_base:
- Let premise and conclusion be the
rule's premise and conclusion
- if premise is a subset of facts:
- if conclusion is not in facts:
  - add Conclusion to facts
  - Set new_facts to True
- print "Derived new fact : conclusion"

If query is in facts:
  Return True
```

Else:
Return False
# Initialize Forward Reasoning Object
reasoner = ForwardReasoning()
# add facts to Knowledge base
reasoner.add_fact("Human(John)")
reasoner.add_fact("Human(mary)")
# add rules (Horn clauses) to the Knowledge base
reasoner.add_rule("Human(x)", "mortal(x)")
reasoner.add_rule("Loved(John, mary)", "Happy(John)")
# Set the query to prove
query = "Mortal(John)"
# perform forward reasoning
result = reasoner.forward_reasoning(query)

if result is True:
Print "Query 'mortal(John)' is proven."
Else:
Print "Query 'mortal(John)' could not be proven."

Output:
Query 'mortal(John)' could not be proven