AI-Lab-Test-2

29/12/2020
JS. Meghana
1BM18CS039
Section-A
Sem-5
Batch-2

5) Design a forward reasoning System to prove the query "Rani likes Peanuts", using forward chaining. The Knowledge base has the following Statements.

1) Rani likes all Kinds of food.
2) Peanut is a food
3) Mug is not food
4) Assume that the reasoning System takes horn clauses as the input.

```
import re
class fact:
    def __init__(self, expression)
        self.expression = expression
        predicate, params = self.SplitExpression(expression)
        self.predicate = predicate
        self.params = params
```

Meg

```
Self.result = any (Self.getconstants())
```

J.S. Meghana
1BM18CS039

```
def SplitExpression (Self, expression):
    predicate = getPredicates (expression) [0]
    Params = getAttributes (expression) [0].Strip('()').
                                        Split (',')
    return [Predicate, Params]

def getResults (Self):
    return Self.result

def getconstants (Self):
    return [None if isVariable (c) else c for c in Self.
                                        params].

def getVariables (Self):
    return [v if isVariable (v) else None for v in Self.param
def Substitute (Self, constants):
    c = constants.copy()
    f = f"{Self.predicate}({','.join([constants.pop(0) if
                isVariable (p) else p for p in Self.params])})"
    return Fact(f).

def isVariable (x):
```


Meg

J.S. Meghana
1BM18CS039

```
return len(x) == 1 and x.islower() and
                        x.isalpha()

def getAttributes(string):
expr = '\([^)]+\)'
matches = re.findall(expr, string)
return matches.
def getPredicates(string):
expr = '([a-z~]+)\([^&|]+\)'
return re.findall(expr, string).
class Implication:
def __init__(self, expression):
self.expression = expression
l = expression.split('=>')
self.lhs = [Fact(f) for f in l[0].split('&')]
self.rhs = Fact(l[1])
def evaluate(self, facts):
constants = {}
new_lhs = []
for fact in facts:
```


Meg.

J.S. Meghana
1BM18CS039

```
for Val in Self.lhs:
if Val.predicate == fact.Predicate:
for i,v in enumerate (Val.getVariables()):
 if v:
   constants [v] = fact.getconstants () [i]
new_lhs. append (fact).
predicate, attributes = getPredicates (Self. rhs. expression)
                [0], str(getAttributes(Self. rhs. expression) [0])

for Key in constants:
if constants [Key]:
attributes = attributes. replace (Key, constants[Key])
expr = f'{predicate}{attributes}'
return Fact (expr) if len (new_lhs) and all([f.getResults()
                for f in new_lhs]) else none.

class KB
def __init__(Self):
Self. facts = Set()
Self. implications = Set()
def tell (Self, e):
 if '=>' in e:
  Self. implication. add (Implication (e))
else:
```

Meg.

J.S. Meghana
1BM18CS039

```
    self.facts.add(Fact(e))
    for i in self.implications:
        res = i.evaluate(self.facts)
        if res:
            self.facts.add(res)
def ask(self, e):
    facts = set([f.expression for f in self.facts])
    i=1
    print(f'Querying {e}:')
    for f in facts:
        if Fact(f).predicate == Fact(e).predicate:
            print(f'\t{i}. {f}')
            i +=1
def display(self):
    print("All facts: ")
    for i, f in enumerate(set([f.expression for f in
                                self.facts])):
        print(f'\t{i+1}. {f}')
def main():
    kb = KB()
```

JS. Meghana
1BM18CS039

```
print ("Enter the number of FOL
        expressions:")
n = int (input())
print ("Enter expressions to be performed:")
for i in range(n):
fact = input()
Kb. tell (fact)
print ("Enter the query:")
query = input()
Kb. ask (query)
Kb. display

main().
```