Artificial Intelligence

Shreyas M
1BM18CS105

```
import re

def  isVariable(x):
  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 Fact:
  def __init__(self, expression):
  self.expression = expression
  predicate, params = self.splitExpression(expression)
  self.predicate = predicate
  self.params = params
  self.result = any(self.getConstants())
```

[illegible]

Shreyas M
1BM18CS105

```
def splitExpression(self, expression):
predicate = getPredicate(expression)[0]
params = getAttributes(expression)[0]
.strip('()').split(',')

return[predicate, params]

def getResult(self):
return self.result

def getConstants(self):
return[None if isVariable(c)
 else c for c in self.params]
 def getVariable(self):
 return[v if isVariable(v) else None for v in self.params]
 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)

class Implication:
 def __init__(self, expression):
     self.expression = expression
     l = expression.split("=>")
     self.lhs = [Fact(f) for f in
```

Shreyas m

1BM18CS165

```
l[0].split('f')]
self.rhs=Fact(l[i])

def evaluate(self, facts):
 constants={}
 new_lhs=[]
 for facts in facts
 for val . prediate = fact. predicate:
      for i,v . inenumerate (val.get variables()):
     if v:
      constante[v]= fact. getconstants()[i]
     new_lhs. append (fact)

predidates, attributes =get Predicate(self .rhs. expression)[0].str
 (get attributes(self.rhs.expression)[0])
for key inconstants:
 if constants [key]: attributes =attributes .replace
  (key, constants[key])
expr = f' {predicate} {attributes}
return Fact(expr) if len(new_lhs) and
all ([f. getResult() for fin new_lhs])
 else None
```

Scanned with CamScanner

Shreyas.M
IBM18CS105

```
class KB:

 def _init__(self):

   self.facts =self()
    self.implications =set()

 def tell(self,e):

  if "=>" in e:
   self.implications.add (Implications(e))
   else:
      self-facts.add (Fact(e))

    for i in self.implications:
     res = i-evaluate (self.facts)
     if res :
        self.facts.add (res)

    def query(self,e):
     facts =set ([f .expression for f in self-facts])
     i=1

    def display (self):
    for i if f in enumerate (set([f.expression for f in self
    facts])):
      print(f'\t{i+1},{f}')
```

Shreyas.M