COMPUTER SCIENCE ENGINEERING
UG
01|01|2021

ARTIFICIAL INTELLIGENCE
20CS5PCAIP
LAB TEST 2

PRANAV JAGADEESH
1BM18CS071
5TH SEM 'B' SECTION
BATCH B2

Design a forward reasoning system to prove the query "Someone intelligent cannot read", using forward chaining. The Knowledge Base has following statements

1. whoever can read is literate
2. Dolphins are not literate
3. Some Dolphins are intelligent

Program Write-up

```
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())
    def splitExpression (self, expression):
          predicate = getPredicates (expression)[0]
      params = getPredicates( expression )[0].
          ~~paltern~~           strip('()').split(',')
       return [predicate, params]
```

Pranav

COMPUTER SCIENCE ENGINEERING
UG
01 | 01 | 2021

ARTIFICIAL INTELLIGENCE
20CS5PCAIP
LAB TEST 2

PRANAV JAGADEESH
1BM18CS071
5TH SEM 'B' SECTION
BATCH B2

```
def getResult (self ):
        return self.result
    def getConstants (self ):
        return [None if isVariable (c) else c for c in
            self. params ]
    def get Variables (self ):
        return [v if is Variable (v) else None for v
            in self. params ]
    def substitute (self, constants ):
        c = constants. copy ()
        f = f "{ self. predicate } ({', '.join(
            [constants. pop (0) if variable
    (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 l[0].
                split ('&')]
            self . rhs = Fact ( l [1])
        def evaluate (self, facts ):
            constants = {}
            new_lhs = [ ]
            for fact in facts :
                for val in self. lhs :
                    if val. predicate == fact. predicate :
                for i, v in enumerate (val. getVariables()
                    if v :
                        constants [v] = fact. get constants()
                                                        [i]
```

2

Pranav

```
new lhs . append( fact )
predicate , attributes = get Predicates (self . rhs . expression)
                              [0] , str
      (get Attributes (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. getResult () for f in n
           new_lhs ] ) else None

class KB:
    def _init_ (self ) :
        self . facts = set ( )
        self . implications = set ( )
        set . 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
        print ( f'Querying {e}:' )
        fo
```

Pranav

COMPUTER SCIENCE ENGINEERING
UG
01|01|2021

ARTIFICIAL INTELLIGENCE
20CS5PCAIP
LAB TEST 2

PRANAV JAGADEESH
1BM18CS071
5TH SEM 'B' SECTION
BATCH B2

```
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()
   print("Enter KB: (enter e to exit)")
     while True:
       t = input()
      if(t == 'e'):
        break
       Kb.tell(t)
      print("Enter Query:")
       q = input()
        Kb=input()
       Kb.query(q)
      Kb.display()
```

[illegible]