29-12-20

1BM18CS028
Dhruv Agrawal
5A
Computer Science and Engineering

# Artificial Intelligence in Python

## Lab Test -2

## Forward Reasoning

Forward reasoning in Artificial Intelligence is an inference system which is used to deduce some query or use to proove some query.

Logical Agents ~~have~~ are defined with knowledge Base. When asked with query, they ask themselves looking in their knowledge base, determine what needs to be done and outputs ~~as~~ a desired action

KB ⟶ Ask ⟶ Action
(Tell) (Reasoning ability) (Measuring Intelligence)

Forward chaining algorithms works with ~~horn~~ horn clauses as it is easy to draw conclusions from premices using implications.

1BM18CS028
Dhruv Agrawal

Knowledge Base

1 Rani Likes all finds of food

$\forall x\,[\text{Food}(x) \rightarrow \text{Likes}(\text{Rani}, x)]$

2 Peanut is food

~~$\forall y[\text{Food}(y$~~ $\forall y\,[\text{Peanut}(y) \rightarrow \text{Food}(y)]$

3 Mug is not food

$\neg \text{Food}(\text{Mug})$

Query: Rani Likes Peanuts

Likes(Rani, Peanuts)

# Three classes have been used to implement

```
class Fact: #Getting the facts
    def __init__(self, expression):
        self.expression = expression
        predicate, params = self.splitExpression(expression)
        self.predicate = predicate
        self.params = params
        self.result = any(self.getConstants())

    def split_expression(self, expression):
        #getting predicate and params by splitting

    def getResult()
    def getConstants()
def getVariables()

def substitute():
    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 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.getConstants()[i]

              new_lhs.append(fact)

      predicate, attributs = getPredicates(self.rhs.expression)[0], ~~str~~
                             str(getAttributs(self.rhs.expression)[0])
```

1BM18CS028
Dhruv Agrawal

```
    for key in constants:
        if constants[key]:
            attributes = attributes.replace(key, constans[key])

    expr = f'{predicate}{attributes}'

    return Fact(expr) if len(new_lhs) and all([f.getResult() 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.implications.add(Implication(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}:')
        for f in facts:
            if ~~ka~~ Fact(f).predicate == Fact(e).predicate:
                print(f'\t{i}. {f}')
```

1BM18CS028
Dhruv Agrawal

```
        i = i+1

    def display()



kb = KB()

n = int(input("Enter number of statements: "))
 for i in range(0,n):
        inp = input()
        kb.tell(inp)

 qry = input("Enter the query");

 kb.query(qry)

 kb.display()
```