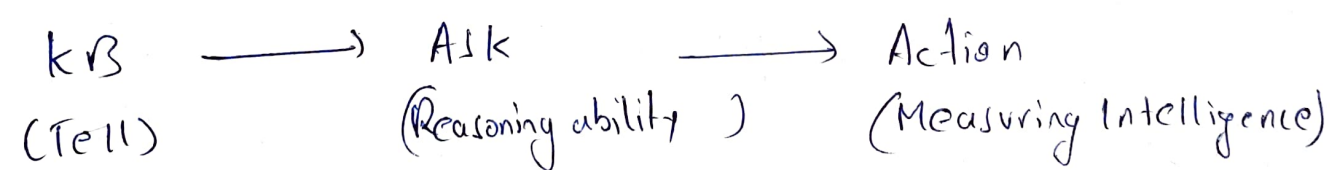


Forward Reasoning

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



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

Knowledge Base

IBM18CS028
Ohrav Agrawal

1 Rani Likes all kinds of food

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

2 Peanut is food

$$\forall x [\text{Peanut}(x) \rightarrow \text{Food}(x)]$$

3 Mug is not food

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

Query: Rani Likes Peanuts

$$\text{Likes}(\text{Rani}, \text{Peanut})$$

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():
```

IBM18CS028
Dhruv Agrawal

```
def substitute():
```

```
    c = constants.copy()
```

```
    t = t + "{ self.predicate } ( { ' , '.join ([ constants.pop() if v isVariable(p)  
        else p for p in self.params ]) } )"
```

```
    return Fact(t)
```

```
class Implication:
```

```
    def __init__(self, expression):
```

```
        self.expression = expression
```

```
        l = expression.split('=>')
```

```
        self.lhs = [ Fact(t) for t 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, attributes = getPredicates(self.rhs.expression)[0], str  
                                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.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 ~~Fact(f)~~.predicate == Fact(e).predicate:

print(f' {i}. {f}')

$i = j + 1$

13M18CS028
Dhruv Agrawal

def display(c)

~~kb~~ = kb(c)

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()