29/12/2020
Tuesday

Lab Test - 2
Artificial Intelligence

Arjun A.S
1BM18CS019

5. Forward reasoning Program

```
import re

def isVariable (char):
    return len(char) ==1 and char.islower () and char.isalpha()

def getAttributes (string) :
    expr = '\([^)]+\)'
    matches = re.findall (expr, string)
    return matches

def getPredicate (string):
    expr = '([a-z~]+)\([^&|]+\)'
    matches = re findall (expr, string )
    return matches

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 =getPredicate (expression)[0]
        params = getAttributes (expression)[0]. strip ('()').split (',')
        return [predicate, params]
```

Aryan R.S
1BM19CS019

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

def getresult (self):
    return self. result.


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

def dosubstitution (self, constants):
    cop = constants. copy()
    fa = f'{self.predicate}({', 'join ([constants.pop(0) if
                   is_variable (p) else p for
                         p in self. params ])"

    return Fact (f)

class Implication:
  def __init__ (self, expression):
      self.expression = expression
      spl = expression. split ('=>')
      self. lhs = [ Fact (f) for f in spl [0]. split ('&') ]
      self. rhs = Fact (spl[1])

  def evaluation_implication(self, facts):
      constants = {}
      new_lhs = []
      for fact in facts
          for clause in self.lhs:
              if clause.predicate == fact. predicate:
```

Arjun A.S
1BM18CS019

```
for i, v in enumerate (clause.getVariables()):
    if v:
        constants [v] = fact.getConstants () [i]
    new_lhs.append (fact)

predicate, = getPredicate (self.rhs.expression)[0],
attributes = 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 KnowledgeBase:
    def __init__ (self):
            self.facts = set()
            self.implications = set()     # implications can be
                                            resolved as Horn clauses

    def tell (self, exp):
        if "=>" in exp:
            self.implications.add (Implication (exp))
        else: self.facts.add (Fact (exp))

        for i in self.implications:
            res = i.evaluation_implication (self.facts)
            if res:
                self.facts.add (res)
```

Arjun AS

Arjun A.S
1BM18CS019

```
def ask(self, exp):   # Querying the KB
    facts = set([f.expression for f in self.facts])
    i=1
    for f in facts:
        if Fact(f).predicate == Fact(exp).predicate:
            print(f '\t {i}. {f}")
            i+=1

def display(self):      # utility function to display all the facts
    print("All facts in Knowledge Base: ")
    for i, f in enumerate(set([f.expression for f in self.facts])):
        print (f'\t {i+1}. {f}')


if __name__ == '__main__":
    kb= KnowledgeBase()          # creates knowledgebase
    print(" Enter Clauses for knowledge Base: (Enter exit to stop):")

    while True:
        clause = input()
        if clause == 'exit':
            break
        kb.tell(clause)

    query= input("Enter Query: ")
    kb.ask(query)
    kb.display()
```

~~The given~~

we know that Horn clauses are the clauses with atmost one positive literal

Suppose given an implication $(P \land Q) \Rightarrow R$

This can be written as

$\neg P \lor \neg Q \lor R$ in this there is only one positive literal 'R'

Hence this is a horn clause.

in Question, given

1. Rani likes all kinds of food:

$\forall X \;\; food(X) \Rightarrow likes(Rani, X)$

Converting CNF $\neg food(X) \lor likes(Rani, X)$

2. Peanut is food

food (Peanut) # fact

3. Mug is not food

¬food (Mug) # fact.

Arjun AS