AI LAB TEST - 2 29/12/20

Adith Pavi Pai 1BM18CS005 Adith.

papergrid
Date: / /

Program 5

Design a forward reasoning system to prove the query "Rani likes Peanuts" using forward chaining. The knowledge base has the following statements :-

(1) Rani likes all kinds of food
(2) Peanut is food
(3) Mug is not food.

Assume reasoning takes horn clause as the input

```
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-zn]+)\([^&|]+\)'
    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.getConsonants())
```

AI lab test 2 IBM18CS005 29/12/20

papergrid
Date: / /

```
def splitExpression(self, expression):
    predicate = getPredicates(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.param]

def substitute(self, constants):
    c = constants.copy()
    f = f"{self.predicate}({','.join([constant.pop(0)
        if isVariable(p) else p for p
                                    in self.param])})"
    return Fact(f)

class Implication:
def __init__(self, expression):
    self.expression = expression
    l = expression.split('=>')

    self.lhs = [Facts(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:
```

AI Lab Test 2 IBM18CS005 Adith Rai

papergrid

Date: / /

```
if val.predicate == fact.predicate:
    for i, v in enumerate(val.getVariables()):
        if v:
            ~~consonants~~ constants[v] = facts.getConstants()[i]
    new_lhs.append(fact)
predicate, attributes = getPredicates(self.rhs.expression)[0],
    str(getAttributes(self.rhs.expression)[0])
for key in constants:
    if constants[key]:
        attributes = attributes.replace(key, constants[key])
expr = f'{predicate}{attributes}'
return ~~f~~ Fact(expr) if len(new_lhs) and all
    ([f.getResult() for f in new_lhs])
    else None

class KB:
    def __init__(self):
        self.facts = ~~f~~ set();
        self.implications = set();

    def tell(self, e):
        if '=>' in e:
            self.implications.add(Implication(e))
        else:
            self.facts.add(Fact(e))
            ~~for~~ for i in self.implications:
                res = ~~f~~ i.evaluate(self.facts)
                i = 1
            print(f'Querying {e}:')
            for f* in facts:
                ~~if fact~~ if Facts(f).predicate == fact(e)
                                .predicate:
                    ~~predi~~ print(f'\t{i}. {f}')
                    ~~i~~ i += 1
```

AI LAB 2

29/12/20

papergrid
Date: / /

```
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 no of FOL expressions in KB:")
    n = int(input())
    print("Enter the expressions:")
    for i in range(n):
        fact = input()
        kb.tell(fact)
    print("Enter the query:")
    query = input()
    kb.ask(query)
    kb.display()
main()
```

Adith Pai
1BM18CS005

Adith