WEEK 10

Create a knowledgebase consisting of first order logic statements and prove the given  
query using forward reasoning.

CODE:

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 = 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 getVariables(self):

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

    def substitute(self, constants):

        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, 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 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'\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}')

kb = KB()

kb.tell('missile(x)=>weapon(x)')

kb.tell('missile(M1)')

kb.tell('enemy(x,America)=>hostile(x)')

kb.tell('american(West)')

kb.tell('enemy(Nono,America)')

kb.tell('owns(Nono,M1)')

kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')

kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')

kb.query('criminal(x)')

kb.display()

OUTPUT:

