

24-1-24

Forward Chaining

27.

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]

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])

```

28

```

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])
```

29

```
i = 1
```

```
print(f'Querying {c}')
```

```
for f in facts:
```

```
if Fact(f).predicate == Fact(c).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(ulst)')
```

```
Kb.tell('enemy(Nono, America)')
```

```
Kb.tell('owns(Nono, M1)')
```

```
Kb.tell('missile(x) & owns(Nono, x) => sells(ulst, x, Nono)')
```

```
Kb.tell('american(x) & weapon(y) & sells(x, y, z) &
```

```
hostile(z) => criminal(x)')
```

```
Kb.query('criminal(x)')
```

```
Kb.display()
```

Output:

Querying criminal(x):

1) criminal(ulst)

All facts:

1. american(ulst)
2. sells(ulst, M1, Nono)
3. missile(M1)
4. enemy(Nono, America)
5. criminal(ulst)
6. weapon(M1)
7. owns(Nono, M1)
8. hostile(Nono)

Forward chaining:

Starts with the base state and uses the inference rules and available knowledge in the forward direction till it reaches the end state. The process is iterated till the final state is reached

$A \wedge B \Rightarrow C$

A

B

Query: C

Shake
24/11/24

```

95 kb = KB()
96 kb.tell('missile(x)=>weapon(x)')
97 kb.tell('missile(M1)')
98 kb.tell('enemy(x,America)=>hostile(x)')
99 kb.tell('american(West)')
100 kb.tell('enemy(Nono,America)')
101 kb.tell('owns(Nono,M1)')
102 kb.tell('missile(x)&owns(Nono,x)=>sells(West,x,Nono)')
103 kb.tell('american(x)&weapon(y)&sells(x,y,z)&hostile(z)=>criminal(x)')
104 kb.query('criminal(x)')
105 kb.display()

```

PROBLEMS ● OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\neha2\OneDrive\Documents\NehaKamath_18M21CS113_AILab> python -u "c:\V

Querying criminal(x):

1. criminal(West)

All facts:

1. missile(M1)

2. weapon(M1)

3. enemy(Nono,America)

4. owns(Nono,M1)

5. hostile(Nono)

6. criminal(West)

7. american(West)

8. sells(West,M1,Nono)

