1BM18CS152
Shivam Vaish
AI Lab Test 2

(5) → Input of

```
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.param = params
        self.result = any(self.getConstants())

    def splitExpression(self, expression):
        predicate = getPredicate(expression)[0]
        params = getAttribute(expression)[0].
        return [predicate, params]
                                  strip('()').split(',')

    def getResult(self):
        return self.result
```

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

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.params]

def substitute(self, expression):

    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])
```

(2)

```
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. getvariable()):
                if v:
                    constants[v] =
                        fact. getconstants(
                            [i]
                    new_lhs. append(fact)
    predicate, attributes = get predicates
        (self. rhs. expression[0]
    str(getAttributes(self. rhs.
        expression)[0])

    for key in constants:
        if constants[key]
            attributes = attributes.
                replace(key, constants
    expr = f'{predicate}{attributes}' [key])
    return fact(expr) if fun(new_lhs)
        and all(f. getresult() for
            f in new_lhs]) else
            None

class KB:
    def __init__(self):
        self. facts = set()
        self. implications = set()
```

(3)

```
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 ('Querying {e}:')
    for f in facts:
        if Fact(f).predicate == Fact(e).
            .predicate:
            print (f'\t{i}. {f}')
            i = i+1

def display (self)
    print ("All facts")
    for i, f in enumerate(
        set([f.expression for
            f in self.facts
```

(9)

```
def main():
    kb = KB()
    print("Enter KB: (write e to exit)")
    while True:
        t = input()
        if (t == 'e'):
            break
        kb.tell(t)
    print("Enter query")
    q = input()
    kb.query(q)
    kb.display()

main()
```