EAST WEST UNIVERSITY

Lab Report-06

Course Title: Artificial Intelligence

Course Code: CSE 366

Semester: Fall 2021

Section No: 01

Submitted By

Name: Syeda Tamanna Sheme

ID:2018-2-60-010

Submitted To

Md Al-Imran

Lecturer

Department of Computer Science & Engineering

Date of Submission: 11 December, 2021

Lab-06: Logic Programming

Theory

Logic Programming talks about the study of principles that orbit across the establishment of reasoning within tasks. It is the analysis of present rules, using which future outcomes can be derived. For instance, if three statements are resulting in a 'True' answer, the program can infer the output of a fourth related statement.

The logic here details the facts and rules that the programming structure needs to understand. A logical approach to programming requires a set of input rules that the code learns and then infers an output on a new related fact it has not seen before. This can also be viewed as a form of a learning algorithm with explicit instruction of understanding.

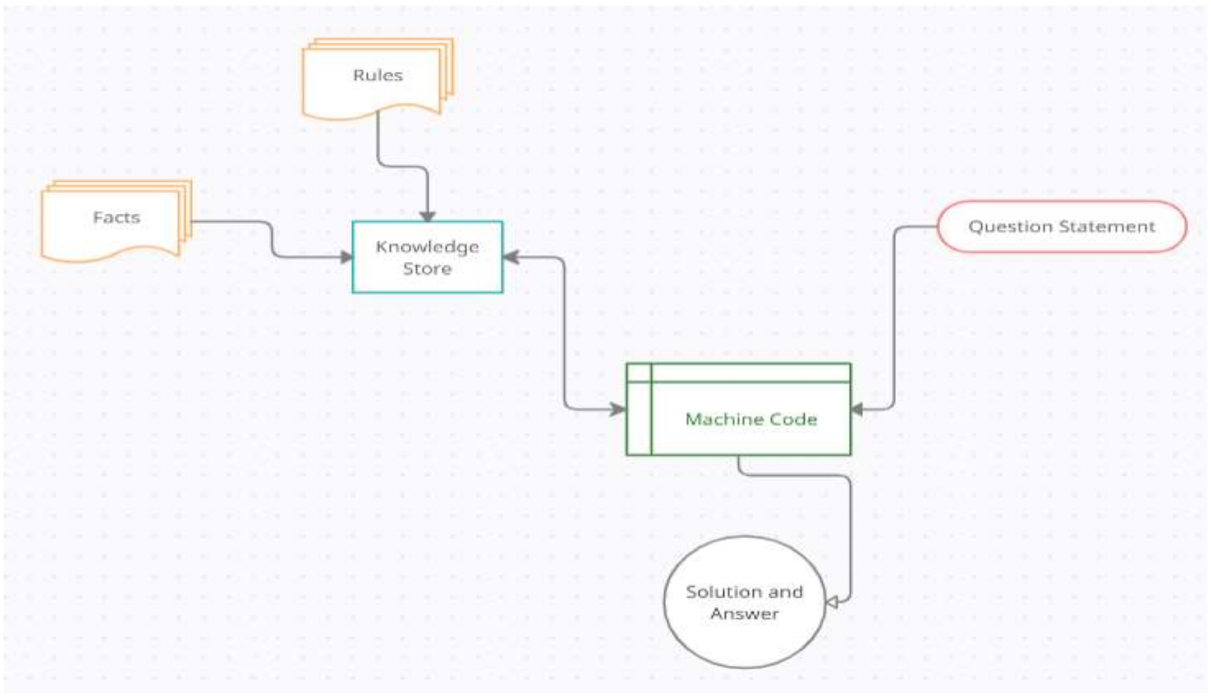

Figure: Logic Programming Visualization

Logic Programming is just another programming paradigm that works on relationships. These relationships are built using facts and rules and are stored as a database of relations. It is a programming methodology that works on formal and explicit logic of events.

Relation: Relations are the basis of logic programming. A relation can be defined as a fact that follows a certain rule. For example, a relation given by [A → B] is read as "if A is true, then B occurs". In language terms, this can be read as, "If you are an Engineer, then you are a Graduate" and infers that, "Engineers are Graduates". In programming languages, the semantics of writing relations change based on the language's syntax, but this is the overall rationality behind what relations mean.

Facts: Every program that is built on logic needs facts. To achieve a defined goal, facts need to be provided to the program. As the name suggests in general English, facts are merely the truth. True statements that represent the program and the data. For instance, Washington is the capital of the United States of America.

Rules: Rules, like programming syntax, are the constraints that help in drawing conclusions from a domain. These are logical clauses that the program or the fact needs to follow to build a relation. You can think of it like this, the fact is that Raman is a man. Now, gender can be a singular entity, that is a rule. A man cannot be a woman. Therefore, the relations we build here are that, since Raman is a man, he cannot be a woman. This is how rules are built:

For example:

```
predecessor(A,B) :- parent(A,B).
predecessor(A,C) :- parent(A,B), predecessor(B,C).
```

This can be read as, for every A and B, if A is the parent of B and B is a predecessor of C, A is the predecessor of C. For every A and B, A is the predecessor of C, if A is the parent of B and B is a predecessor of C.

These relationships are specified in a file called **relationships.json** provided for you. The file looks like the following:

Code:

```
{

  "father":

 [
    {"John": "William"},
    {"John": "David"},
    {"John": "Adam"},
    {"William": "Chris"},
    {"William": "Stephanie"},
    {"David": "Wayne"},
    {"David": "Tiffany"},
    {"David": "Julie"},
    {"David": "Neil"},
    {"David": "Peter"},
    {"Adam": "Sophia"}
 ],
 "mother":

 [
   {"Megan": "William"},
   {"Megan": "David"},
   {"Megan": "Adam"},
   {"Emma": "Stephanie"},
   {"Emma": "Chris"},
   {"Olivia": "Tiffany"},
   {"Olivia": "Julie"},
   {"Olivia": "Neil"},
   {"Olivia": "Peter"},
   {"Lily": "Sophia"}
 ]
}
```

It is a simple JSON file that specifies the father and mother relationships.

Create a new Python file and import the following packages:

```
import json

from kanren import Relation, facts, run, conde, var, eq
```

Load the data from the relationships.json file:

```
with open('relationships.json') as f:

    d = json.loads(f.read())
```

Define a function to check if x is the parent of y. We will use the logic that if x is the parent of y, then x is either the father or the mother. We have already defined "father" and "mother" in the fact base:

```
# Check if 'x' is the parent of 'y'

def parent(x, y):

 return conde([father(x, y)], [mother(x, y)])
```

Define a function to check if x is the grandparent of y. We will use the logic that if x is the grandparent of y, then the offspring of x will be the parent of y:

```
# Check if 'x' is the grandparent of 'y'

def grandparent(x, y):

 temp = var()

 return conde((parent(x, temp), parent(temp, y)))
```

Define a function to check if x is the sibling of y. We will use the logic that if x is the sibling of y, then x and y will have the same parents. Notice that there is a slight modification needed here because when we list out all the siblings of x, x will be listed as well because x satisfies these conditions. So, when we print the output, we will have to remove x from the list. We will discuss this in the main function:

```
# Check for sibling relationship between 'a' and 'b'

def sibling(x, y):

 temp = var()

 return conde((parent(temp, x), parent(temp, y)))
```

Define a function to check if x is y's uncle. We will use the logic that if x is y's uncle, then x grandparents will be the same as y's parents. Notice that there is a slight modification needed here because when we list out all the uncles of x, x's father will be listed as well because x's father satisfies these conditions. So, when we print the output, we will have to remove x's father from the list. We will discuss this in the main function:

```
# Check if x is y's uncle

def uncle(x, y):

 temp = var()

 return conde((father(temp, x), grandparent(temp, y)))
```

Define the main function and initialize the relations father and mother:

```
if __name__=='__main__':

 father = Relation()

 mother = Relation()
```

Read the data and add it to the fact base:

```
for item in d['father']:

    facts(father, (list(item.keys())[0], list(item.values())[0]))

for item in d['mother']:

    facts(mother, (list(item.keys())[0], list(item.values())[0]))
```

Define the variable x:

```
     x = var()
```

We are now ready to ask some questions and see if the solver can come up with the right answers. Let's ask who John's children are:

```
# John's children

 name = 'John'

 output = run(0, x, father(name, x))

 print("\nList of " + name + "'s children:")

 for item in output:

    print(item)
```

Output:

```
List of John's children:
David
Adam
William
```

Who is William's mother?

```
# William's mother

name = 'William'

output = run(0, x, mother(x, name))[0]

print("\n" + name + "'s mother:\n" + output)
```

Output:

```
William's mother:
Megan
```

Who are Adam's parents?

```
# Adam's parents name = 'Adam'

 output = run(0, x, parent(x, name))

 print("\nList of " + name + "'s parents:")

 for item in output:

    print(item)
```

Output:

```
List of William's parents:
John
Megan
```

Who are Wayne's grandparents?

```
# Wayne's grandparents name = 'Wayne'

 output = run(0, x, grandparent(x, name))

 print("\nList of " + name + "'s grandparents:")

 for item in output:

    print(item)
```

Output:

```
List of Tiffany's grandparents:
John
Megan
```

Who are Megan's grandchildren?

```
# Megan's grandchildren

 name = 'Megan'

 output = run(0, x, grandparent(name, x))

 print("\nList of " + name + "'s grandchildren:")

 for item in output:

    print(item)
```

Output:

```
List of Megan's grandchildren:
Neil
Sophia
Stephanie
Peter
Chris
Julie
Tiffany
Wayne
```

Who are David's siblings?

```
# David's siblings

 name = 'David'

 output = run(0, x, sibling(x, name))

 siblings = [x for x in output if x != name]

 print("\nList of " + name + "'s siblings:")

 for item in siblings:

    print(item)
```

Output:

```
List of David's siblings:
William
Adam
```

Who are Tiffany's uncles?

```
# Tiffany's uncles

 name = 'Tiffany'

 name_father = run(0, x, father(x, name))[0]

 output = run(0, x, uncle(x, name))

 output = [x for x in output if x != name_father]

 print("\nList of " + name + "'s uncles:")

 for item in output:

    print(item)
```

Output:

```
List of Tiffany's uncles:
William
Adam
```

List all of the spouses in the family:

```
# All spouses
 a, b, c = var(), var(), var()
 output = run(0, (a, b), (father, a, c), (mother, b, c))
 print("\nList of all spouses:")
 for item in output:
    print('Husband:', item[0], '<==> Wife:', item[1])
```

Output:

```
List of all spouses:
Husband: David <==> Wife: Olivia
Husband: John <==> Wife: Megan
Husband: William <==> Wife: Emma
Husband: Adam <==> Wife: Lily
```

Results

We learn Logic Programming after completing the lab task and lab activity. We also learn JSON, how to read and write files, and Python Logic Programming. We are now able to produce shows on these subjects.

Discussion

Python is a general-purpose, versatile and popular programming language. It's great as a first language because it is concise and easy to read, and it is also a good language to have in any programmer's stack as it can be used for everything from web development to software development and data science applications.

This lab task is a great introduction to both fundamental programming concepts and the Python programming language. Python 3 is the most up-to-date version of the language with many improvements made to increase the efficiency and simplicity of the code that we write. Day by-day, python new version are realising and all the new versions have new features and these new features are better than previous one. So, finally I can say that python will be more user friendly in future.