Introduction to Artificial Intelligence
Lecture 6: Logic Knowledge Representation

1. Introduction to pyDatalog

pyDatalog is a logic programming library for Python. Unlike standard programming where you tell the computer *how* to do something step-by-step (imperative), in pyDatalog you tell the computer *what* result you want (declarative), and the engine figures out how to find it. It is used to store knowledge and infer new information from that knowledge.

Installation

```
pip install pyDatalog
```

Importing

```
import pyDatalog. pyDatalog as py
```

---

2. Variables and Predicates

In logical programming, we use **Variables** to represent unknown values in Rules an Queries, and **Predicates** to establish facts. We must declare these variables and predicates before using them. The variables should be declared UPPERCASE, while the predicts have no constraints.

```
py.create_terms('X, Y,parent,brother')
```

---

3. Facts and Clauses (Rules)

This is the core of knowledge representation.

- **Facts:** Simple pieces of data that are known to be true (e.g., "John is the father of Bill"). In pyDatalog, we often use the + operator to add a fact.

  `Example: +parent('bill', 'john adams')`

- **Clauses (Rules):** Logic statements that define relationships. In pyDatalog, the operator <= represents "If".

  Result <= Condition means "The Result is true **if** the Condition is met."
  `Example: brother(X, Y) <= (parent(X, Z)) & (parent(Y, Z)) & (X != Y)`

---

4. Query:

The query is simply a question you ask the logic engine to discover facts that satisfy certain logical rules and relationships. The query looks like facts but it contains variable which we need the computer to find.

`Example: print(brother('bill', Y))`

Example:

We will define family relationships. First, we store facts about who is a parent. Then, we write a rule to determine who is a "brother". **For each query, the correct output is written below in a comment.**

```python
import pyDatalog.pyDatalog as py

# Initialize terms
py.create_terms('parent, brother, X, Y, Z')

# --- Facts ---
+parent('bill', 'john adams')
+parent('mark', 'john adams')


# Query: Who is the parent of Bill?
print(parent('bill', X))


# Output: john adams

# --- Rules ---

brother(X, Y) <= (parent(X, Z)) & (parent(Y, Z)) & (X != Y)

# Query: Who is Bill's brother?
print(brother('bill', Y))

|
# Output: mark
```

Exercise:
The following are the marks scored by 5 students:
ahmad 90 -- sami 45 -- khaled 60 -- carol 70 -- mohamad 80
Enter the above data using pyDatalog and write queries for the following:

1. print student name and mark of all students.
2. who has scored 80 marks?
3. what mark has been scored by sami?
4. write a rule for finding grade letters for marks 'grades' and use the rule to find the grade letter of a given mark.
5. use the previous rule to print all student who failed.

```
######solution
import pyDatalog.pyDatalog as py
py.create_terms('X,Y,marks,grades')
+marks('ahmad',90)
+marks('sami',45)
+marks('khaled',85)
+marks('carol',70)
+marks('mohammad',80)
#1--------------------------------
print(marks(X,Y))

#2--------------------------------
print(marks(X,80))

#3--------------------------------
print(marks('sami',X))

#4--------------------------------
grades(X,'A') <= marks(X,Y) & (Y>= 90) & (Y<= 100)
grades(X,'B') <= marks(X,Y) & (Y>= 80) & (Y< 90)
grades(X,'C') <= marks(X,Y) & (Y>= 70) & (Y< 80)
grades(X,'D') <= marks(X,Y) & (Y>= 60) & (Y< 70)
grades(X,'F') <= marks(X,Y) & (Y< 60)
print(grades(X,Y))

#5--------------------------------
print(grades(X,'F'))
```