**ASSIGNMENT HELP**

**MANUAL**



SUBMITTED

TO

VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE

FOR THE SKILL AND COMPETENCY EVALUATION OF

ARTIFICIAL INTELLIGENCE [CAUA31201]

IN

**CSE AI DEPARTMENT**

BY

**Vedant Rakesh Mukhekar**

**Class: T.Y. BTech Division: A Batch: A2**

**Batch Teacher**

**Dr. ANURADHA YENKIKAR.**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **CONTENTS** | **PAGE NO.** |
| **1** | **PROBLEM STATEMENT** | **4-5** |
| **2** | **LIBRARY USED** | **5-6** |
| **3** | **THEORY** | **6-7** |
| **4** | **METHODOLOGY** | **7-14** |
| **5** | **ADVANTAGES & DISADVANTAGES** | **14-15** |
| **6** | **WORKING** | **16-17** |
| **7** | **DIAGRAM** | **17-18** |
| **8** | **CONCLUSION** | **18-19** |

### ****Problem Statement****

The goal is to parse and infer family relationships using a **knowledge-base**. Given facts about family members (e.g., parent-child relationships), the system should deduce additional relationships such as siblings, grandparents, and cousins. We will use logic programming to represent the family relationships, define rules, and allow for querying the knowledge-base to infer new relationships dynamically.

The family tree structure will be used to define who is related to whom, and the program will allow for parsing the tree to answer questions like:

* Who are the grandparents of a given individual?
* Who are the siblings or cousins of an individual?
* What are the parent-child relationships in the tree?

### ****Libraries Used****

* **Python Libraries**:
  + **pyDatalog**: A logic programming library that allows us to define relationships (facts) and rules, and query them in a way similar to a knowledge-base.
  + **Optional**: networkx or other graph libraries for visualization of the family tree.

### ****Theory****

In a **family tree**, individuals are connected by relationships like parent-child, sibling, and grandparent-grandchild. Using **logic programming**, we define basic facts such as "John is the parent of Mary," and then use rules to infer other relationships like "Mary and Jake are siblings because they share a parent."

We can formalize the relationships as:

* **Parent-Child**: Direct relationship between a parent and a child.
* **Siblings**: Two individuals are siblings if they share the same parent.
* **Grandparents**: A person is the grandparent of another if their child is the parent of that person.
* **Cousins**: Two individuals are cousins if their parents are siblings.

### ****Methodology****

1. **Define Facts**: Store basic parent-child relationships in the knowledge-base.
2. **Define Rules**: Use logic rules to infer more complex relationships (sibling, grandparent, cousin).
3. **Parse and Query**: Query the knowledge-base to get relationships like siblings, grandparents, etc.

### ****Advantages and Disadvantages****

* **Advantages**:
  + Flexible querying of relationships.
  + Easy extension with new facts and rules.
  + Automated inference of relationships.
* **Disadvantages**:
  + Can be inefficient for very large family trees due to exhaustive rule checking.
  + Complex queries might be harder to optimize.

### ****Working****

Here’s how to use **pyDatalog** for parsing the family tree:

1. **Create the facts** representing known relationships (e.g., parent-child).
2. **Define rules** to infer other relationships based on these facts.
3. **Query the knowledge-base** for relationships like siblings, grandparents, or cousins.

### ****Code Implementation****

python

Copy code

from pyDatalog import pyDatalog

# Clear previous facts and rules in pyDatalog

pyDatalog.clear()

# Define terms (relationships and individuals)

pyDatalog.create\_terms('parent, sibling, grandparent, cousin, X, Y, Z')

# Facts: parent(Parent, Child)

+parent('John', 'Mary')

+parent('Alice', 'John')

+parent('Bob', 'John')

+parent('John', 'Jake')

+parent('Alice', 'Tom')

+parent('Bob', 'Tom')

+parent('Jake', 'Mike')

# Rule for siblings: Two people are siblings if they share the same parent

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

# Rule for grandparents: X is a grandparent of Y if X is a parent of Z and Z is a parent of Y

grandparent(X, Y) <= (parent(X, Z)) & (parent(Z, Y))

# Rule for cousins: Two people are cousins if their parents are siblings

cousin(X, Y) <= (parent(Z, X)) & (parent(W, Y)) & sibling(Z, W)

# Queries

print("Siblings of Mary: ", sibling(X, 'Mary')) # Returns siblings of Mary

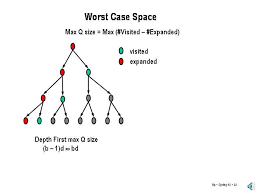
print("Grandparents of Mike: ", grandparent(X, 'Mike')) # Returns grandparents of Mike

print("Cousins of Mike: ", cousin(X, 'Mike')) # Returns cousins of Mike

### ****Explanation of Code****

1. **Define Facts**:
   * The basic relationships are defined as parent-child facts using +parent('John', 'Mary'), meaning John is the parent of Mary.
2. **Define Rules**:
   * **Siblings**: The rule for siblings checks if two individuals share the same parent and are not the same person.
   * **Grandparents**: The grandparent rule checks if an individual is the parent of a parent.
   * **Cousins**: The cousin rule checks if the parents of two individuals are siblings.
3. **Querying the Knowledge-Base**:
   * The program can now query relationships based on the facts and rules. For example, querying sibling(X, 'Mary') will return Mary's siblings, and querying grandparent(X, 'Mike') will return Mike's grandparents.

### ****Diagram for Family Tree Example****



### ****Queries and Outputs****

1. **Siblings of Mary**:

sql

Copy code

sibling(X, 'Mary')

Output: John (since John is a parent of both Mary and Jake, they are siblings)

1. **Grandparents of Mike**:

vbnet

Copy code

grandparent(X, 'Mike')

Output: Alice, Bob (because Alice and Bob are parents of Jake, and Jake is a parent of Mike)

1. **Cousins of Mike**:

sql

Copy code

cousin(X, 'Mike')

Output: None (since Mike does not have any first cousins in the given family tree)

### ****Conclusion****

Parsing a **family tree** using a **knowledge-base** with a tool like **pyDatalog** allows us to infer complex family relationships (siblings, grandparents, cousins) from basic facts (like parent-child relationships). The power of this approach lies in its ability to infer new information automatically through well-defined rules, making it a useful tool for reasoning about structured data like family trees.