Title: Introduction to PROLOG Programming

Theory:

Prolog is founded on predicate logic, a branch of formal logic used to model relationships and facts. Its core structure is built using Horn clauses, which simplify logical expressions into facts (parent(john, mary).) and rules (ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).). Prolog uses backward chaining to answer queries by starting from a goal and working backward to find matching facts or rules. The process of unification allows Prolog to match patterns by binding variables to specific values. Additionally, Prolog assumes a closed-world assumption, using negation as failure to treat any unprovable statements as false. These theoretical principles make Prolog powerful for applications in reasoning, artificial intelligence, and problem-solving.

PROLOG Code:

```
% Facts
likes(ram, mango).
girl(sheema).
likes(bill, candy).
red(rose).
owns(john, gold).

% Queries Examples
?- likes(ram, X).
% Output: X = mango.

?- girl(G).
% Output: G = sheema.

?- likes(bill, N).
% Output: N = candy.

?- likes(D, U).
% Output:
% D = ram, U = mango;
% D = bill, U = candy.
```

```
% Additional Facts and Relationships
female(pam).
female(liz).
female(ann).
male(tom).
male(bob).
male(pat).
male(jim).

% Parent-Child
Relationships parent(pam,
bob).  parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).

% Rules
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
grandmother(X, Y) :- grandparent(X, Y), female(X).
grandfather(X, Y) :- grandparent(X, Y), male(X).
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
siblings(X, Y) :- X \== Y, parent(Z, X), parent(Z, Y).

% New Rules Added
uncle(X, Y) :- male(X), parent(Z, Y), siblings(X, Z).
aunt(X, Y) :- female(X), parent(Z, Y), siblings(X, Z).
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

Outputs of Parent-Child Relationship:

```
?- parent(pam, bob).
```

% Output: true.

?- female(pam).
% Output: true.

?- male(jim).
% Output: true.

?- grandparent(X, jim).
% Output: X = bob.

?- grandfather(X, ann).
% Output: X = tom.

?- grandmother(X, ann).
% Output: X = pam.

?- uncle(X, jim).
% Output: X = bob.

?- aunt(X, pat).
% Output: X = liz.

?- ancestor(X, jim).
% Output:
% X = pat;
% X = bob;
% X = tom;
% X = pam.

Discussion:

In this lab, we explored Prolog programming by creating a database of facts and rules to represent relationships and properties. Starting with simple facts like likes(ram, mango) and girl(sheema), we expanded to complex relationships such as parent-child and sibling connections. Using queries like ?- likes(ram, X) and ?- grandparent(X, jim), we extracted information and validated relationships. We also added rules like uncle and ancestor to enhance functionality.

This exercise demonstrated Prolog’s declarative nature, which simplifies knowledge representation by focusing on what to solve rather than how. Debugging and refining rules improved our understanding, and the successful execution of queries highlighted Prolog’s utility for reasoning and AI applications.

Conclusion:

This lab provided a comprehensive introduction to Prolog programming, where we learned how to define facts and rules to model complex relationships. By working with queries, we were able to explore and validate these relationships, gaining valuable experience in how Prolog handles logical reasoning. The exercise helped us understand how this declarative language can be used in AI and related domains. All queries and rules were tested successfully, and the lab was completed with a solid understanding of Prolog's capabilities.