# Batch: B4 Experiment Number: 5

**Roll Number: 16010421119 Name: Aarya Tiwari**

# Aim of the Experiment:

Write a program for implementation of family tree in PROLOG using condition-action rules based agent.

# Program/ Steps:

This approach follows a table for lookup of condition-action pairs defining all possible condition-action rules necessary to interact in an environment.

Create a family tree program to include following rules:

* M is the mother of P, if she is a parent of P and is female
* F is Father of P, if he is parent of P and is male
* X is Sibling of Y, if they have same parent
* Then add rules for sister, brother, grandfather, grandmother, uncle, aunty, cousins etc (consider 3
* generations of your own family and build the family tree)

Based on the facts, define goals to answer questions related to family tree.

# Output/Result:

**Code:**

male(james). female(lily). male(sirius). male(snape). male(hary). female(hermoine). male(ron).

brave(harry). loyal(harry). loyal(ron). clever(hermoine).

friend(harry,ron). friend(harry,hermoine). enemy(harry,malfoy). friend(james, sirius).

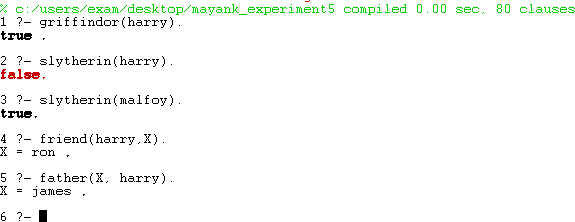
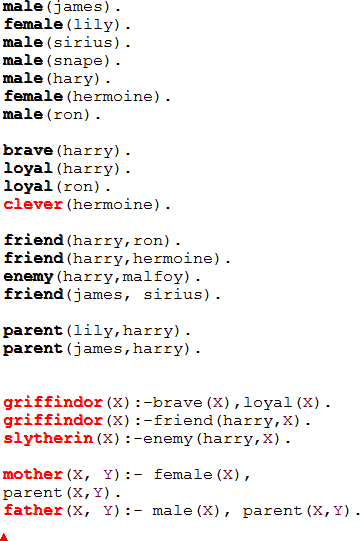
parent(lily,harry). parent(james,harry).

griffindor(X):-brave(X),loyal(X). griffindor(X):-friend(harry,X). slytherin(X):-enemy(harry,X).

mother(X, Y):- female(X), parent(X,Y).

father(X, Y):- male(X), parent(X,Y).

# Output:



**Post Lab Questions:**

# The PROLOG suit is based on

* 1. **Interpreter**

# Compiler

* 1. **None of the above**

# Both

Ans: d. Both

# State true of false

**There must be at least one fact pertaining to each predicate written in the PROLOG program.**

The given statement is TRUE.

# State true of false

**In PROLOG program the variable declaration is a compulsory part.**

The given statement is FALSE.

# Differentiate between a fact and a predicate with syntax. Facts:

A fact is an expression that makes a declarative statement about the problem domain. Whenever a variable occurs in a Prolog expression, it is assumed to be universally quantified. Note that all Prolog sentences must end with a period.

# Syntax:

likes(john, susie). /\* John likes Susie \*/ likes(X, susie). /\* Everyone likes Susie \*/ likes(john, Y). /\* John likes everybody \*/

likes(john, Y), likes(Y, john). /\* John likes everybody and everybody likes John \*/ likes(john, susie); likes(john,mary). /\* John likes Susie or John likes Mary \*/ not(likes(john,pizza)). /\* John does not like pizza \*/

likes(john,susie) :- likes(john,mary)./\* John likes Susie if John likes Mary.

# Predicates:

A predicate expression uses logical implication (:-) to describe a relationship among facts. Thus a Prolog predicate takes the syntax:

left\_hand\_side :- right\_hand\_side .

This sentence is interpreted as: left\_hand\_side if right\_hand\_side. The left\_hand\_side is restricted to a single, positive, literal, which means it must consist of a positive atomic expression. It cannot be negated

and it cannot contain logical connectives. Examples of valid predicates:

friends(X,Y) :- likes(X,Y),likes(Y,X). /\* X and Y are friends if they like each other \*/ hates(X,Y) :- not(likes(X,Y)). /\* X hates Y if X does not like Y. \*/

enemies(X,Y) :- not(likes(X,Y)),not(likes(Y,X)). /\* X and Y are enemies if they don't like each other \*/ Examples of invalid predicates:

left\_of(X,Y) :- right\_of(Y,X) /\* Missing a period \*/ likes(X,Y),likes(Y,X) :- friends(X,Y). /\* LHS is not a single literal \*/ not(likes(X,Y)) :- hates(X,Y). /\* LHS cannot be negated \*/

# s5. Differentiate between knowledge base and Rule base approach.

* Rule-based systems process data and output information, but they also process rules

and make decisions. They are good at processing lots of simple business rules with broad logic. They are commonly used for real-time decision systems, straight-thru processing (STP) systems, and compliance systems.

* Knowledge-based systems also process data and rules to output information and make decisions. In addition, they also process expert knowledge to output answers, recommendations, and expert advice. They are good at processing deep logic and very complex business rules. They are commonly used for advising systems, expert systems, and knowledge automation.

# Outcomes:

CO3: Ability to formally state the problem and develop the appropriate proof for given a logical deduction problem.

# Conclusion (based on the Results and outcomes achieved):

Through this experiment, we successfully understood and implemented a family tree in PROLOG using condition-action rules based agent.

# References:

* + Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Second Edition, Pearson Publication
  + Luger, George F. Artificial Intelligence: Structures and strategies for complex problem solving, 2009 ,6th Edition, Pearson Education
  + <https://www.101computing.net/prolog-family-tree/>