**Installation**

Installing Prolog varies depending on your operating system. Here are general steps:

1. **Windows:**
   * Download SWI-Prolog from SWI-Prolog website.
   * Run the installer and follow the installation instructions.
2. **Mac OS:**
   * SWI-Prolog can be installed via Homebrew: brew install swi-prolog.
   * Alternatively, download from the SWI-Prolog website and follow the instructions.
3. **Linux:**
   * Use your package manager (e.g., apt for Ubuntu/Debian, yum for Fedora) to install SWI-Prolog.
   * Alternatively, download from the SWI-Prolog website and follow the instructions.

**Syntax**

Prolog programs consist of **facts**, **rules**, and **queries**. Here’s a brief overview:

* **Facts:** Facts are assertions about the domain you're working with. They are typically written as predicates. For example:

likes(john, pizza).

likes(mary, chocolate).

* **Rules:** Rules define relationships between facts. They are written using a syntax similar to implications. For example:

friend(X, Y) :- likes(X, Z), likes(Y, Z).

* **Queries:** Queries are used to ask Prolog to find solutions based on the defined facts and rules. For example:

?- likes(john, pizza).

**Semantics**

Prolog operates on the basis of formal logic, specifically a form of logic called Horn clauses. The semantics of Prolog are based on the following principles:

* **Logical Inference:** Prolog uses a form of logical deduction called **backward chaining.** It tries to prove a goal (query) by matching it against the available facts and rules.
* **Unification:** This is a key concept in Prolog where terms are matched against each other to find substitutions that make them identical. This is crucial for resolving variables in rules and queries.
* **Resolution:** Prolog resolves queries by recursively trying to unify goals with the known facts and rules, using backtracking to explore different possibilities until a solution (or all solutions) are found.

In summary, Prolog is a declarative language where you define what is true (facts) and how truths are connected (rules), and then query the system to find solutions based on these declarations using logical inference and unification.

1.Create a file with extension “.pl”

2.specify the path

**consult('C:/Users/Rajee/OneDrive/Desktop/AVIT/8 july 2024 -AI/Prolog/1.Basic.pl').**

3.Ask queries

Or

1.command promt (move to the path) – [cd C:\Program Files\swipl\bin]

2.swipl

3.[filename].

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**1.Write a program in Prolog to create simple knowledge base and query over it.**

% Facts: parent(X, Y) means X is a parent of Y

parent(john, mary).

parent(john, luke).

parent(sarah, mary).

parent(sarah, luke).

% Rules: sibling(X, Y) means X and Y are siblings if they have the same parent

sibling(X, Y) :-

parent(Z, X),

parent(Z, Y),

X \= Y.

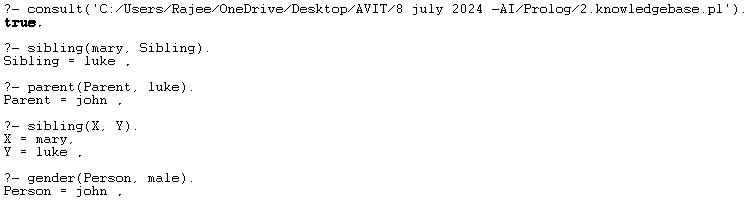
% Additional facts (optional): gender(X, G) specifies the gender of person X

gender(john, male).

gender(luke, male).

gender(sarah, female).

gender(mary, female).



**2. Write program for expert systems inferences using Prolog**

**Expert systems** are a branch of artificial intelligence (AI) designed to mimic the decision-making ability of a human expert. These systems use a combination of knowledge and inference procedures to solve problems that normally require human expertise. Here are some key aspects of how inferences are made in expert systems:

**Components of Expert Systems**

1. **Knowledge Base**: This is a repository of factual and heuristic knowledge. It contains rules, facts, and relationships about the specific domain of expertise.
2. **Inference Engine**: This component applies logical rules to the knowledge base to deduce new information or make decisions. It's the "brain" of the expert system.
3. **User Interface**: The interface through which users interact with the system, inputting data and receiving results.

**Types of Inferences**

1. **Forward Chaining**:
   * **Description**: Starts with a set of known facts and applies inference rules to extract more data (from the knowledge base) until a goal is reached.
   * **Process**: It operates by checking if the conditions of rules are met (based on known facts) and then executing the rules to infer new facts.
   * **Use Case**: Diagnosis systems where initial symptoms are known and the system deduces the possible diseases.
2. **Backward Chaining**:
   * **Description**: Starts with potential hypotheses or goals and works backward to see if there are facts that support these hypotheses.
   * **Process**: It starts with the goal and works backward through inference rules to determine what facts must be true to achieve the goal.
   * **Use Case**: Expert systems in troubleshooting where the goal is to find the cause of a problem by checking necessary conditions.

**Inference Methods**

1. **Rule-Based Inference**:
   * Uses "if-then" rules stored in the knowledge base.
   * **Example**: If a patient has a fever and a sore throat, then the system infers that the patient might have a throat infection.
2. **Frame-Based Inference**:
   * Uses structures called frames, which are data structures for dividing knowledge into substructures by representing "stereotyped situations."
   * **Example**: A frame for a car might include slots for the make, model, year, and owner.
3. **Fuzzy Logic**:
   * Allows for reasoning with uncertainty by dealing with approximate rather than fixed and exact reasoning.
   * **Example**: Diagnosing medical conditions where symptoms do not exactly match any known condition.
4. **Bayesian Inference**:
   * Uses probability theory to predict outcomes based on prior knowledge and evidence.
   * **Example**: Predicting the likelihood of a disease given the presence of various symptoms and their known probabilities.

**Applications**

* **Medical Diagnosis**: Systems like MYCIN, which helped diagnose bacterial infections and recommended antibiotics.
* **Financial Services**: Systems that help in credit scoring, fraud detection, and investment analysis.
* **Customer Support**: Automated systems that provide technical support based on common issues and solutions.
* **Industrial Process Control**: Systems that manage and control industrial processes, ensuring efficiency and safety.

**Conclusion**

Expert systems utilize a structured approach to mimic human reasoning by applying rules and logic to a well-defined knowledge base. By using forward chaining, backward chaining, and various inference methods, these systems can solve complex problems, offering valuable support in fields requiring specialized expertise.

% Define user preferences

prefers(furry).

prefers(four\_legs).

prefers(small).

% Define pet characteristics

pet(cat) :-

prefers(furry),

prefers(four\_legs).

pet(dog) :-

prefers(furry),

prefers(four\_legs).

pet(hamster) :-

prefers(furry),

prefers(small).

pet(fish) :-

\+ prefers(furry),

prefers(small).

% Recommendation rule

recommend\_pet(Pet) :-

pet(Pet).

% Start the recommendation

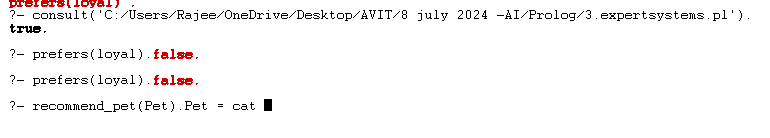
start :-

( recommend\_pet(Pet)

-> format('You should get a ~w.~n', [Pet])

; write('No suitable pet found.')

).



**3. Write a program for finding inferential relationship from family tree using Prolog**

% Facts: parent(X, Y) means X is a parent of Y

parent(john, mary).

parent(john, luke).

parent(sarah, mary).

parent(sarah, luke).

parent(mary, anne).

parent(mary, peter).

parent(luke, emily).

parent(luke, david).

% Additional facts (optional): gender(X, G) specifies the gender of person X

gender(john, male).

gender(sarah, female).

gender(mary, female).

gender(luke, male).

gender(anne, female).

gender(peter, male).

gender(emily, female).

gender(david, male).

% Rules: sibling(X, Y) means X and Y are siblings if they have the same parent

sibling(X, Y) :-

parent(Z, X),

parent(Z, Y),

X \= Y.

% Rules: grandparent(X, Y) means X is a grandparent of Y

grandparent(X, Y) :-

parent(X, Z),

parent(Z, Y).

% Rules: ancestor(X, Y) means X is an ancestor of Y (transitive closure)

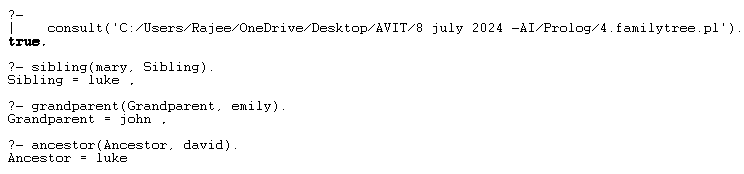
ancestor(X, Y) :-

parent(X, Y).

ancestor(X, Y) :-

parent(X, Z),

ancestor(Z, Y).



**­­­­­­­­­­­­­­­**

**4. Write program for simple reflex response generation in Prolog**

% Define the reflex responses

response(see\_cat, run\_away).

response(see\_dog, stay\_still).

response(see\_food, approach).

response(hear\_noise, alert).

response(feel\_threatened, fight\_back).

% Rule to generate a response based on input stimulus

generate\_response(Stimulus, Response) :-

response(Stimulus, Response).

% Interactive loop to get stimuli and provide responses

start :-

write('Enter a stimulus (type "exit" to quit): '), nl,

read(Stimulus),

(Stimulus == exit ->

write('Goodbye!'), nl;

(generate\_response(Stimulus, Response) ->

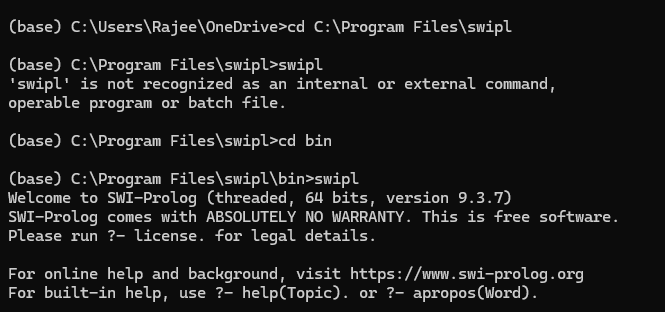
format('Response: ~w~n', [Response]);

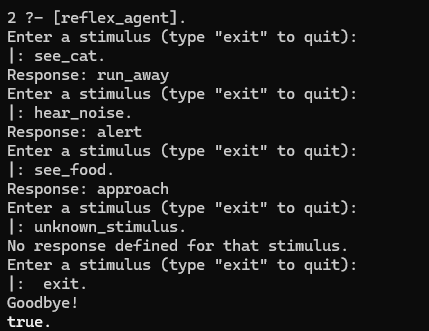
write('No response defined for that stimulus.'), nl),

start).

% Running the program

:- initialization(start).





**5. Write program to conduct game search using Minimax Algorithm approach in IDP**

% Graph representation

edge(a, b).

edge(a, c).

edge(b, d).

edge(b, e).

edge(c, f).

edge(c, g).

edge(d, h).

edge(d, i).

edge(e, j).

edge(f, k).

edge(g, l).

% Iterative Deepening Depth-First Search (IDDFS) implementation

% iddfs/3: Predicate to perform IDDFS

iddfs(Start, Goal, Path) :-

% Start with depth limit 0 and increment until goal is found

between(0, inf, Depth), % `between` generates integers from 0 to infinity

dfs\_limit(Start, Goal, Depth, [], Path).

% dfs\_limit/4: Perform DFS up to given Depth limit

dfs\_limit(Node, Node, \_, \_, [Node]). % Goal reached

dfs\_limit(Node, Goal, DepthLimit, Visited, [Node | Path]) :-

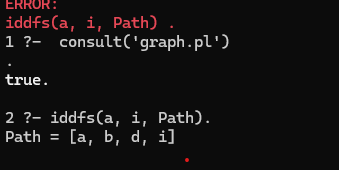
DepthLimit > 0,

DepthLimit1 is DepthLimit - 1,

edge(Node, Neighbor),

\+ member(Neighbor, Visited), % Ensure Neighbor hasn't been visited

dfs\_limit(Neighbor, Goal, DepthLimit1, [Node | Visited], Path).



**6. Write a program to infer from the Bayesian network**

% Node probabilities

probability(burglary, true, 0.001).

probability(earthquake, true, 0.002).

% Conditional probabilities

probability(alarm, true, burglary, earthquake, 0.95).

probability(alarm, false, burglary, earthquake, 0.05).

probability(john\_calls, true, alarm, 0.90).

probability(john\_calls, false, alarm, 0.10).

probability(mary\_calls, true, alarm, 0.70).

probability(mary\_calls, false, alarm, 0.30).

% Query for conditional probability P(Query | Evidence)

% conditional\_probability(Query, Evidence, Probability)

conditional\_probability(Query, Evidence, Probability) :-

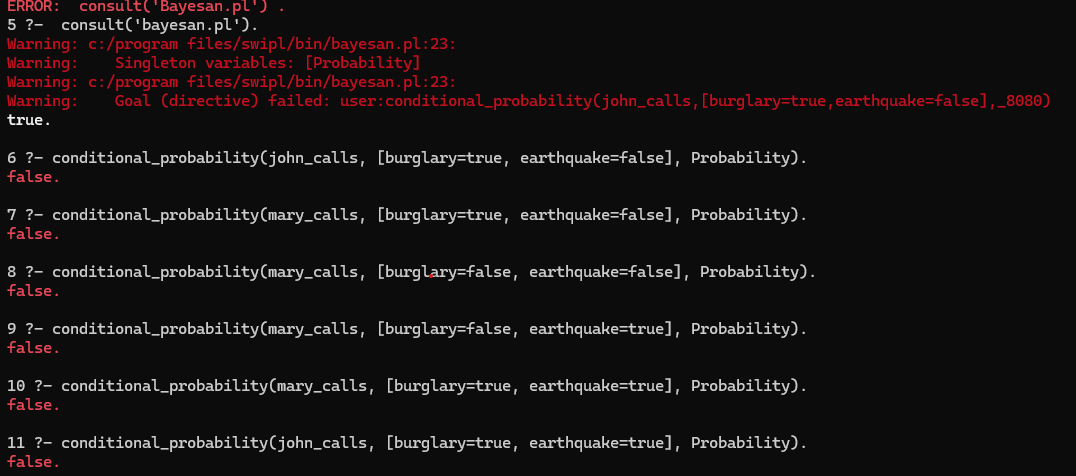
probability(Query, true, Evidence, Ptrue),

probability(Query, false, Evidence, Pfalse),

Probability is Ptrue / (Ptrue + Pfalse).

% Query example: P(john\_calls | burglary=true, earthquake=false)

?- conditional\_probability(john\_calls, [burglary=true, earthquake=false], Probability).



**7. Write a program for character recognition using IDP**

% Grammar for arithmetic expressions using DCGs

expr(E) --> number(E).

expr(E) --> expr(A), "+", expr(B), { E is A + B }.

expr(E) --> expr(A), "\*", expr(B), { E is A \* B }.

number(N) --> integer(N).

integer(N) --> digit(D), { number\_codes(N, [D]) }.

integer(N) --> digit(D), integer(R), { number\_codes(Digit, [D]), N is Digit \* 10 + R }.

digit(D) --> [D], { code\_type(D, digit) }.

% Predicate to recognize arithmetic expressions

recognize(Input, Value) :-

phrase(expr(Value), Input).

**8. Write program for Image classification using IDP libraries**

% Example Prolog knowledge base

% Rules for reasoning based on image classification results

suggested\_action([(\_, \_, 'dog') | \_], 'dog'):-

write('Action: Play with the dog.').

suggested\_action([(\_, \_, 'cat') | \_], 'cat'):-

write('Action: Feed the cat.').

suggested\_action(\_, 'unknown'):-

write('Unknown object. No action suggested.').

% Prolog predicate to interface with Python script

classify\_image(ImagePath, Suggestions) :-

python\_exec('python\_script.py', [ImagePath], Suggestions).

% Predicate to execute Python script and retrieve suggestions

python\_exec(Script, Args, Suggestions) :-

atomic\_list\_concat(['python3', Script | Args], ' ', Command),

shell(Command, 0, Process),

process\_create(Process, stdout(pipe(Out)), []),

read\_lines(Out, Suggestions).

% Helper predicate to read output from Python script

read\_lines(Out, Suggestions) :-

read\_line\_to\_codes(Out, Line),

( Line \== end\_of\_file ->

atom\_codes(Atom, Line),

atom\_json\_dict(Atom, Suggestions, []),

read\_lines(Out, Suggestions)

; close(Out)

).

% Example usage in Prolog

?- classify\_image('path/to/your/image.jpg', Suggestions), suggested\_action(Suggestions, Action).

**9. Write program for Image classification using IDP libraries**

% Predicate for human player move

human\_move(Board, NewBoard) :-

repeat,

write('Enter your move (row, column): '),

read(Row-Col),

( valid\_move(Board, Row, Col) ->

make\_move(Board, Row, Col, 'X', NewBoard), !

; write('Invalid move. Try again.'), nl,

fail

).

% Predicate for AI player move (simple random move)

ai\_move(Board, NewBoard) :-

findall(Row-Col, (between(1, 3, Row), between(1, 3, Col), valid\_move(Board, Row, Col)), ValidMoves),

random\_member(Row-Col, ValidMoves),

make\_move(Board, Row, Col, 'O', NewBoard).

% Predicate to play the game

play :-

initial(Board),

repeat,

display\_board(Board),

human\_move(Board, NewBoard),

( win(NewBoard, 'X') ->

display\_board(NewBoard),

write('You win!'), nl, !

; full(NewBoard) ->

display\_board(NewBoard),

write('It\'s a draw!'), nl, !

; ai\_move(NewBoard, AIBoard),

( win(AIBoard, 'O') ->

display\_board(AIBoard),

write('AI wins!'), nl, !

; full(AIBoard) ->

display\_board(AIBoard),

write('It\'s a draw!'), nl, !

; play

)

).

% Start the game

:- initialization(play).

1. **What does the cut operator (!) do in Prolog?**
   * A. Deletes a variable
   * B. Forces backtracking
   * C. Prevents backtracking
   * D. Starts a new clause
   * **Answer: C**
2. **What is the purpose of the "is" operator in Prolog?**
   * A. To check for equality
   * B. To assign a value after evaluation
   * C. To declare a fact
   * D. To declare a rule
   * **Answer: B**
3. **In Prolog, what does the query "?- append([1,2], [3,4], X)." return?**
   * A. X = [1,2,3,4]
   * B. X = [3,4,1,2]
   * C. X = [[1,2],[3,4]]
   * D. X = [1,2]
   * **Answer: A**
4. **Which of the following is true about Prolog?**
   * A. Prolog is case-insensitive
   * B. Prolog is a procedural language
   * C. Prolog uses unification instead of assignment
   * D. Prolog does not support recursion
   * **Answer: C**
5. **What does the "trace" command do in Prolog?**
   * A. Tracks variable values
   * B. Outputs debugging information
   * C. Traces memory usage
   * D. Generates a stack trace
   * **Answer: B**
6. **In Prolog, which symbol is used for logical negation?**
   * A. !
   * B. +
   * C. not
   * D. ~
   * **Answer: B**
7. **In Prolog, what does the predicate "fail" do?**
   * A. Succeeds the goal
   * B. Ends the program
   * C. Causes the goal to fail
   * D. Throws an error
   * **Answer: C**
8. **How is disjunction represented in Prolog?**
   * A. ,
   * B. ;
   * C. &
   * D. |
   * **Answer: B**
9. **What does the Prolog predicate "assert" do?**
   * A. Removes a fact
   * B. Adds a fact
   * C. Updates a fact
   * D. Checks a fact
   * **Answer: B**
10. **Which of the following is a Prolog term?**
    * A. Atom
    * B. Number
    * C. Variable
    * D. All of the above
    * **Answer: D**
11. **What does the predicate "reverse(List, ReversedList)" do?**
    * A. Sorts List in descending order
    * B. Removes duplicates from List
    * C. Reverses the order of elements in List
    * D. Concatenates List with ReversedList
    * **Answer: C**
12. **Which of the following is an atom in Prolog?**
    * A. john
    * B. John
    * C. \_john
    * D. 123
    * **Answer: A**
13. **What is the purpose of the "cut" operator in Prolog?**
    * A. Deletes a fact
    * B. Prevents further backtracking
    * C. Ends the program
    * D. Forces backtracking
    * **Answer: B**
14. **Which of the following is a built-in predicate in Prolog?**
    * A. print
    * B. output
    * C. show
    * D. display
    * **Answer: A**
15. **How can you ensure a predicate is evaluated only once?**
    * A. Using a cut (!)
    * B. Using repeat
    * C. Using fail
    * D. Using assert
    * **Answer: A**
16. **What is the effect of the "retract" predicate in Prolog?**
    * A. Adds a fact
    * B. Removes a fact
    * C. Updates a fact
    * D. Checks a fact
    * **Answer: B**
17. **What does the query "?- X =.. [functor, arg1, arg2]." do?**
    * A. Decomposes a term into its functor and arguments
    * B. Creates a list from a term
    * C. Reverses the arguments
    * D. Combines functor and arguments into a term
    * **Answer: D**
18. **Which of the following is true about Prolog lists?**
    * A. Lists are mutable
    * B. Lists have fixed sizes
    * C. Lists are dynamic
    * D. Lists cannot contain other lists
    * **Answer: C**
19. **In Prolog, what does the "repeat" predicate do?**
    * A. Repeats a goal infinitely
    * B. Ensures a goal is executed once
    * C. Repeats a goal until failure
    * D. Repeats a goal until success
    * **Answer: A**
20. **In Prolog, how do you define a list with a head and tail?**
    * A. [Head | Tail]
    * B. {Head | Tail}
    * C. (Head, Tail)
    * D. [Head, Tail]
    * **Answer: A**