Knowledge Representation (Prolog) Provide a detailed explanation of the syntax and semantics of Prolog, a logic programming language. Describe the structure of Prolog programs, including terms (atoms, variables, numbers, and compound terms), and explain how they are used. Discuss the declarative nature of Prolog and how it differs from imperative programming. Explain the role of unification, backtracking, and resolution in Prolog's execution model. Provide examples to illustrate key concepts, such as how Prolog processes queries and matches facts and rules.

ChatGPT said:

Here's a detailed explanation of **Prolog (Programming in Logic)** with a focus on its **syntax, semantics, structure**, and the key features that differentiate it from imperative languages:

What Is Prolog?

Prolog is a **logic programming language** used for symbolic, non-numeric computation, often applied in AI, natural language processing, and knowledge representation. It is **declarative** in nature—programs define *what* should be true rather than *how* to compute results.

Syntax and Semantics of Prolog

1. Terms in Prolog

Terms are the fundamental building blocks of Prolog programs. They come in several types:

Term Type	Description	Example
Atom	Constant symbol used to represent fixed values (like strings or identifiers)	apple, john, 'Hello World'
Number	Integer or float values	42, 3.14
Variable	Placeholder for any term; starts with a capital letter or underscore _	X, Name, _Temp
Compound Term	Term with a *functor* and *arguments* (like a structured object)	likes(john, pizza)

---

2. Structure of a Prolog Program

A Prolog program consists of:

-

Facts – Statements known to be true.

-
-

Rules – Conditional statements (like “if-then”).

-
-

Queries – Questions asked to the Prolog system.

-

➤ Fact Example

```
prolog
CopyEdit
parent(john, mary).
```

This means: “John is a parent of Mary.”

➤ Rule Example

```
prolog
CopyEdit
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

This reads as: “X is a grandparent of Z if X is a parent of Y and Y is a parent of Z.”

➤ Query Example

```
prolog
CopyEdit
?- grandparent(john, Z).
```

This asks: “Who is the grandchild of John?”

Declarative vs Imperative Programming

Feature	Declarative (Prolog)	Imperative (e.g., Python, C)
Focus	*What* is true	*How* to perform computations
Control Flow	Implicit via unification and backtracking	Explicit using loops, conditionals
State Changes	No direct state mutation	Frequent use of variable assignments
Execution Model	Based on logic inference (resolution)	Based on sequential commands

---

Key Execution Concepts

1. Unification

Unification is the process of making two terms identical by finding a suitable substitution for variables.

```
prolog
CopyEdit
?- X = john.
X = john.
prolog
CopyEdit
?- likes(john, X) = likes(john, pizza).
X = pizza.
```

2. Backtracking

Prolog tries to satisfy goals by trying facts/rules in order. If one fails, it *backtracks* and tries alternatives.

Example:

```
prolog
CopyEdit
likes(john, pizza).
likes(john, pasta).
```

Query:

```
prolog
CopyEdit
```

```
?- likes(john, X).
```

Prolog will:

1.

First return `X = pizza`.

2.
3.

On user prompt (`;`), backtrack and return `X = pasta`.

4.

3. Resolution

Resolution is the core inference mechanism. It tries to prove the truth of a query by recursively applying facts and rules.

Full Example: Family Tree

➤ Program:

```
prolog
CopyEdit
% Facts
parent(john, mary).
parent(mary, alice).
parent(mary, bob).

% Rule
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

➤ Query:

```
prolog
CopyEdit
?- grandparent(john, Z).
```

➤ Execution:

1.

Find `Y` such that `parent(john, Y)` → `Y = mary`

2.
3.

Find `Z` such that `parent(mary, Z)` → `Z = alice` and `Z = bob`

4.
5.

Return:

6.

```
prolog
CopyEdit
Z = alice ;
Z = bob.
```

Summary

Concept	Description
Atom	Named constant
Variable	Placeholder for unknown
Compound Term	Functor + arguments
Fact	Basic truth
Rule	Conditional truth
Query	Question to evaluate
Unification	Matching with variable substitution
Backtracking	Searching for alternative matches
Resolution	Logical inference engine

Would you like a visual diagram of how backtracking or unification works in Prolog?

4o

You said:

Explain how to write facts, rules, and queries in Prolog with a focus on their syntax and practical applications. Define what facts and rules are, and describe how they are used to represent knowledge in a Prolog program. Provide a step-by-step guide on constructing facts

(e.g., representing relationships like 'parent(john, mary)') and rules (e.g., defining 'grandparent(X, Z) :- parent(X, Y), parent(Y, Z)'). Explain how to write and execute queries to retrieve information or test relationships. Include multiple examples, such as a family tree or a simple expert system, and demonstrate how Prolog evaluates queries using backtracking and unification

ChatGPT said:

Here's a step-by-step, practical guide to writing **facts**, **rules**, and **queries** in **Prolog**, focusing on syntax, semantics, and real-world applications like **family trees** and **expert systems**.

---

1. Understanding the Basics

✅ What is a Fact in Prolog?

-

A **fact** is a statement that represents a known truth.

-
-

Syntax: `predicate(arg1, arg2, ..., argN).`

-
-

Example: `parent(john, mary).` — "John is a parent of Mary."

-

✅ What is a Rule in Prolog?

-

A **rule** defines a relationship based on other facts or rules.

-
-

Syntax: `head :- body.`

-
-

The **head** is true if the **body** is true (logical implication).

Example:

```
prolog
```

```
CopyEdit
```

```
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

"X is a grandparent of Z if X is a parent of Y and Y is a parent of Z."

✅ What is a Query in Prolog?

A **query** asks a question of the Prolog knowledge base.

Syntax: `?- goal.`

Example:

```
prolog
```

CopyEdit

?- parent(john, mary).

---

2. Writing Facts and Rules: Step-by-Step

Example: Family Tree

Step 1: Define Basic Facts

prolog
CopyEdit

```
% Parent relationships
parent(john, mary).
parent(mary, alice).
parent(mary, bob).
parent(jane, mary).
parent(john, mark).
```

Step 2: Add Additional Relationships (e.g., gender)

prolog
CopyEdit

```
male(john).
male(mark).
female(mary).
female(alice).
female(jane).
female(bob).  % Assume bob is a nickname for Roberta
```

Step 3: Define Rules Using Facts

prolog
CopyEdit

```
% Grandparent rule
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
```

```
% Sibling rule
sibling(X, Y) :- parent(Z, X), parent(Z, Y), X \= Y.

% Father and mother rules
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
```

---

3. Writing Queries

Query 1: Who are Mary’s children?

```
prolog
CopyEdit
?- parent(mary, X).
```

➡ **Prolog Answer**: `X = alice ; X = bob.`

Query 2: Who are John’s grandchildren?

```
prolog
CopyEdit
?- grandparent(john, X).
```

➡ **Answer**: `X = alice ; X = bob.`

Query 3: Are Alice and Bob siblings?

```
prolog
CopyEdit
?- sibling(alice, bob).
```

➡ **Answer**: `true.`

Query 4: Who is Alice’s mother?

```
prolog
CopyEdit
?- mother(X, alice).
```

➡ **Answer**: `X = mary.`

---

4. Prolog Evaluation Model

Backtracking

Prolog tries to satisfy a goal by trying clauses from top to bottom. If it hits a failure, it **backtracks** to try the next option.

Example:

```
prolog
CopyEdit
likes(john, pizza).
likes(john, pasta).
?- likes(john, X).
```

➡ Returns: `X = pizza ; X = pasta ; false.`

Unification

Prolog matches terms with pattern matching + variable binding.

```
prolog
CopyEdit
likes(john, pizza).
?- likes(john, X).
```

➡ `X` unifies with `pizza`.

---

5. A Simple Expert System Example

Problem: Diagnose a plant disease

Facts

```
prolog
CopyEdit
has_symptom(plant1, yellow_leaves).
has_symptom(plant1, wilting).
has_symptom(plant2, spots_on_leaves).
```

Rules

```
prolog
```

```
CopyEdit
disease(plant1, root_rot) :-
    has_symptom(plant1, yellow_leaves),
    has_symptom(plant1, wilting).

disease(plant2, leaf_spot) :-
    has_symptom(plant2, spots_on_leaves).
```

Queries

```
prolog
CopyEdit
?- disease(plant1, X).
```

➡ `X = root_rot.`

---

6. Best Practices

- Always end facts and rules with a **period (`.`)**.
- Use meaningful predicate names (e.g., `has_symptom`, `grandparent`).
- Start variables with a **capital letter** (e.g., `X`, `Person`).
- For anonymous variables, use `_` (e.g., `parent(_, mary)`).

Summary Table

Component	Syntax	Example
Fact	`predicate(arg1, arg2).`	`parent(john, mary).`

Component	Syntax	Example
Rule	`head :- body.`	`grandparent(X,Z):-...`
Query	`?- goal.`	`?- parent(X, mary).`
Backtracking	Implicit search strategy	`?- likes(john, X).` → all options
Unification	Pattern matching	`?- likes(john, pizza).` → true

Would you like a downloadable `.pl` Prolog file for one of these examples to run in SWI-Prolog or a similar interpreter?