Building an Intelligent Expert System in Prolog — A Complete Guide

Introduction

Prolog—that's short for PROgramming in LOGic—is a cool programming language. People use it a lot for AI stuff, like getting computers to understand language, storing information, and building smart systems. It's different from languages like C or Java. With Prolog, you tell it what you want, not exactly how to get it.

In this comprehensive tutorial, we will:

- Build a real-world expert system from scratch in Prolog.

- Cover facts, rules, queries, recursive relationships, and dynamic knowledge bases.

- Discuss performance optimizations and real-world usage.

- Demonstrate how Prolog can model complex logical reasoning tasks.

By the end, you'll see why Prolog remains relevant in today's AI-driven world — and why it’s valued by tech giants who seek engineers and technical writers fluent in logical reasoning.

Part 1: Fundamentals of Prolog

1.1. Facts

Facts are the basic building blocks of a Prolog knowledge base.

```
% Syntax: predicate(arguments).

likes(alice, pizza).
```

```
likes(bob, sushi).
likes(charlie, pasta).
```

1.2. Rules

Rules infer new knowledge based on existing facts.

```
% Syntax: head :- body.

friends(X, Y) :- likes(X, Food), likes(Y, Food).
```

1.3. Queries

Queries ask Prolog to infer facts based on the rules.

```
?- friends(alice, bob).
false.

?- friends(alice, charlie).
false.
```

Part 2: Building a Real Expert System

2.1. Knowledge Base

```
% Symptoms
symptom(john, fever).
symptom(john, cough).
symptom(john, sore_throat).

symptom(mary, rash).
symptom(mary, fever).

symptom(david, fatigue).
symptom(david, weight_loss).

% Diseases and Symptoms
disease(flu, [fever, cough, sore_throat]).
```

```
disease(measles, [rash, fever, cough]).
disease(diabetes, [fatigue, weight_loss, excessive_thirst]).
```

2.2. Diagnosis Rules

```
% Check if a patient has all symptoms for a disease
has_all_symptoms(_, []).
has_all_symptoms(Patient, [Symptom|Symptoms]) :-
    symptom(Patient, Symptom),
    has_all_symptoms(Patient, Symptoms).

% Diagnose patient
diagnose(Patient, Disease) :-
    disease(Disease, Symptoms),
    has_all_symptoms(Patient, Symptoms).
```

2.3. Querying the Expert System

```
?- diagnose(john, Disease).
Disease = flu.

?- diagnose(mary, Disease).
Disease = measles.

?- diagnose(david, Disease).
false.
```

Part 3: Advanced Features

3.1. Dynamic Knowledge Base

```
:- dynamic symptom/2.

add_symptom(Patient, Symptom) :-
    assertz(symptom(Patient, Symptom)).
```

3.2. Interactive Diagnosis

```
ask(Patient, Symptom) :-
   format('Does ~w have ~w? (yes/no): ', [Patient, Symptom]),
   read(Reply),
   (Reply == yes -> assertz(symptom(Patient, Symptom)) ; true).

collect_symptoms(Patient, []) :- true.
collect_symptoms(Patient, [Symptom|Symptoms]) :-
   ask(Patient, Symptom),
   collect_symptoms(Patient, Symptoms).

start_diagnosis(Patient) :-
   findall(S, (disease(_, L), member(S, L)), SymptomList),
   list_to_set(SymptomList, UniqueSymptoms),
   collect_symptoms(Patient, UniqueSymptoms),
   (diagnose(Patient, Disease) ->
      format('~w is diagnosed with ~w.~n', [Patient, Disease])
   ;
      writeln('No diagnosis could be made.')
   ).
```

Part 4: Deep Dive — Logical Reasoning in Prolog

```
father(john, mary).
father(john, david).
father(mike, sarah).

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



?- parent(john, Child).
Child = mary ;
Child = david.
```

Part 5: Performance Considerations

```
happy(X) :- rich(X), !, healthy(X).
happy(X) :- healthy(X).
```

Real-World Applications

Prolog powers:
- IBM Watson's question-answering system.
- NASA's fault diagnosis systems.
- Medical expert systems.
- AI planning algorithms.
- Natural language understanding (Chatbots, Semantic Parsers).

Conclusion

Prolog is a hidden powerhouse for applications needing reasoning, pattern matching, and logical inference.
This full guide demonstrated:
- Basic syntax (facts, rules, queries).
- Building a real-world expert system.
- Advanced topics like dynamic updates, user interaction, performance tuning.
- Prolog's profound relevance in modern AI and logic-driven applications.

Bonus: Full Source Code (Paste-and-Run)

```
:- dynamic symptom/2.

symptom(john, fever).
symptom(john, cough).
symptom(john, sore_throat).

symptom(mary, rash).
symptom(mary, fever).

symptom(david, fatigue).
symptom(david, weight_loss).

disease(flu, [fever, cough, sore_throat]).
```

```
disease(measles, [rash, fever, cough]).
disease(diabetes, [fatigue, weight_loss, excessive_thirst]).

has_all_symptoms(_, []).
has_all_symptoms(Patient, [Symptom|Symptoms]) :-
    symptom(Patient, Symptom),
    has_all_symptoms(Patient, Symptoms).

diagnose(Patient, Disease) :-
    disease(Disease, Symptoms),
    has_all_symptoms(Patient, Symptoms).

add_symptom(Patient, Symptom) :-
    assertz(symptom(Patient, Symptom)).

ask(Patient, Symptom) :-
    format('Does ~w have ~w? (yes/no): ', [Patient, Symptom]),
    read(Reply),
    (Reply == yes -> assertz(symptom(Patient, Symptom)) ; true).

collect_symptoms(_, []) :- true.
collect_symptoms(Patient, [Symptom|Symptoms]) :-
    ask(Patient, Symptom),
    collect_symptoms(Patient, Symptoms).

start_diagnosis(Patient) :-
    findall(S, (disease(_, L), member(S, L)), SymptomList),
    list_to_set(SymptomList, UniqueSymptoms),
    collect_symptoms(Patient, UniqueSymptoms),
    (diagnose(Patient, Disease) ->
        format('~w is diagnosed with ~w.~n', [Patient, Disease])
    ;
        writeln('No diagnosis could be made.')
    ).
```