SWI-Prolog

Intelligent Systems II

Mário Antunes

2026

Universidade de Aveiro

Getting Started with SWI-Prolog

Installing SWI-Prolog

- Debian/Ubuntu

```
sudo apt-get update
sudo apt-get install swi-prolog
```

- Fedora/RHEL

```
sudo dnf install swi-prolog
```

This installs the SWI-Prolog interpreter (`swipl`*). You can now run* `swipl` *from your terminal.*

Creating a Prolog File

Save your code to a file, e.g., `family.pl`:

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

Loading and Querying in SWI-Prolog

1. Start Prolog:

```
swipl
```

2. Load your file:

```
?- consult('family.pl').
% or
?- [family].
```

3. Run queries:

```
?- grandparent(X, Y).
```

Output: `X = john, Y = alice.`

Prolog Syntax and Fundamentals

Introduction to SWI-Prolog

- SWI-Prolog is the most popular environment for Logic Programming.
- Implements the **Warren Abstract Machine (WAM)**.
- Key concept: Query a database of truths.

Basic Syntax: Atoms and Constants

- **Atoms:** Literal constants. Must start with a lowercase letter. E.g. `apple`
- Use quotes: `'John Smith'` for spaces or uppercase.

Variables in Prolog

- **Variables:** Uppercase letter or underscore. E.g. `X`, `Result`, `_hidden`.
- **Anonymous Variable (_)** when value doesn't matter.

Facts and Rules

- **Facts:** e.g., `at(agent, hall).`, `temperature(hall, 22).`
- **Rules:** e.g., `should_cool(Room) :- temperature(Room, T), T > 25.`

Queries and Inference

3. Queries and Inference

The Query Operator: `?-`

- Used in the interactive shell or as a directive in source files to ask a logical question of the knowledge base.
- Example (interactive shell):

```
?- parent(john, X).
X = mary.
```

- You can use `;` to request more answers (backtracking):

```
?- parent(X, mary).
X = john ;
false.
```

- **Directives:** In `.pl` files, `?- ...` runs at load time.

Querying with Unification: =

- The = operator checks if two terms can be unified. Not assignment, but a logical match.
- Examples:

```
?- X = mary.
X = mary.

?- foo(A, B) = foo(1, 2).
A = 1,
B = 2.
```

Arithmetic and Comparison in Queries i

- `is` - Evaluates right-hand side and unifies result with left.

```
?- X is 2 + 2.
X = 4.
```

- `=:=` - True if both sides evaluate to the same. Used for equality tests.

```
?- 2 + 2 =:= 4.
true.
```

Arithmetic and Comparison in Queries ii

- > and < - Arithmetic comparisons:

```
?- 7 > 3.
true.
```

- `=\=` - True if values are not equal after evaluation.

```
?- 7 =\= 5.
true.
```

Arithmetic Operators: Division

- Standard division `/` in Prolog returns a floating-point number (e.g., `5 / 2` is `2.5`).
- **Integer Division (`//`)** truncates the decimal and returns a whole number.
- Crucial for array indexing or binary search (like our agent's midpoint calculation).

```
?- X is 5 // 2.
X = 2.
```

The Neck Operator: `:-`

- Separates **head** from **body** of rule (Head `:-` Body).
- Logical implication (if Body true, then Head is true).

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

Using Queries in Source Files (Directives)

- Starting a line with `?-` in a `.pl` file executes that goal when the file is loaded.

Practical Query Examples

- Find all grandparents:

```
?- grandparent(X,Y).
```

- Test arithmetic equality:

```
?- X is 5*2, X =:= 10.
X = 10.
```

Facts vs. Rules

- A clause with a neck operator is a **rule**, while a clause without one is a **fact**
- **Rule:** `parent(X, Y) :- mother(X, Y)`
- **Fact:** `parent(anna, bob)` is a rule with body `true`.

Debugging and Understanding Execution

The `trace` Mechanism

- Use `trace.` to visualize search and unification in resolution tree.
- **Ports:** Call, Exit, Redo, Fail
- **Example of Trace:** Tracing `solve(10, 8)` shows stepwise recursive search (calls to perception, action, recursive loop, and base case).

Why Tracing is Essential

- **Explainability, Optimization (with cut `!`), Unification**

Unification and Assignment Differences

- Unification (=) is not assignment, but an equality constraint: binds variables only if possible.
- E.g., `?- parent(john, X) = parent(john, mary).` gives `X = mary`.
- Python assignment stores value directly.

Lists, Recursion

List Processing

- Lists: `[a, b, c]` is shorthand for `.(a, .(b, .(c, []))).`
- `[H|T]` unifies head/tail, e.g. `[1,2,3] = [H|T]` $ o$ `H=1, T=[2,3].`

Recursion

```
count([], 0).
count([_|T], N) :- count(T, N1), N is N1 + 1.
```

Negation as Failure (\+)

- `\+ goal` is true if goal cannot be proved.

The Cut Operator (!)

- The `!` operator tells Prolog to **stop backtracking** and commit to the choices made so far.
- Think of it as a one-way door: once Prolog crosses the `!`, it cannot go back and try alternative rules for that specific goal.
- **Why use it?** Efficiency and logic control.

```
% If it is hot, stop checking! Do not evaluate 'cold' or 'found'.
perceive_hint(Secret, Guess, hot) :- Guess > Secret, !.
perceive_hint(Secret, Guess, cold) :- Guess < Secret, !.

# Agent Memory and State

## Dynamic Predicates: Modifying the Knowledge Base

* Standard Prolog facts and rules are static (loaded from a file and c
* **Dynamic predicates** allow an agent to "learn" or change its state
* You must declare them at the top of your `.pl` file:
  ```prolog
```
```

Assert and Retract

- `asserta(Fact).` - Adds a fact to the *beginning* of the knowledge base.
- `assertz(Fact).` - Adds a fact to the *end* of the knowledge base.
- `retract(Fact).` - Removes the first matching fact from the knowledge base.
- `retractall(Fact).` - Removes all matching facts.

Example: Updating Agent State

```prolog
location(agent, hall). % Initial dynamic fact

move(NewRoom) :-
    location(agent, CurrentRoom),
    retract(location(agent, CurrentRoom)),  % Forget old location
    assertz(location(agent, NewRoom)),      % Remember new location
    write('Moved from '), write(CurrentRoom),
    write(' to '), write(NewRoom).
```

Agent Example: Hot or Cold Search

Logic PoC: Hot/Cold

- **Environment:**
  - Hot: Guess > Secret
  - Cold: Guess < Secret
- **Goal:** Iterate guesses until found.

Perception and Actions in Prolog

```
% World feedback logic (Unchanged)
perceive_hint(Secret, Guess, hot) :- Guess > Secret, !.  % "hot" means
perceive_hint(Secret, Guess, cold) :- Guess < Secret, !. % "cold" mean
perceive_hint(Secret, Guess, found) :- Guess =:= Secret.

% Actions with limits (Min and Max)
% act(Hint, CurrentGuess, CurrentMin, CurrentMax, NextGuess, NewMin, N
act(hot, Guess, Min, _, NextGuess, Min, NewMax) :-
    NewMax is Guess - 1,                   % The secret must be lower
    NextGuess is (Min + NewMax) // 2.      % Calculate new midpoint

act(cold, Guess, _, Max, NextGuess, NewMin, Max) :-
    NewMin is Guess + 1,                   % The secret must be higher
    NextGuess is (NewMin + Max) // 2.      % Calculate new midpoint
```

Agent Recursive Loop

```
% Starter function to define initial boundaries
smart_solve(Secret, MinLimit, MaxLimit) :-
    InitialGuess is (MinLimit + MaxLimit) // 2,
    solve(Secret, InitialGuess, MinLimit, MaxLimit).

% Base Case
solve(Secret, Guess, _, _) :-
    perceive_hint(Secret, Guess, found),
    write('Goal Reached: '), write(Guess), !.

% Recursive Step
solve(Secret, Guess, Min, Max) :-
    perceive_hint(Secret, Guess, Hint),
    write('Perception: '), write(Hint), write(' | Guessed: '), write(Gu
    act(Hint, Guess, Min, Max, NextGuess, NewMin, NewMax),
    solve(Secret, NextGuess, NewMin, NewMax).
```

Execution Example

```
?- smart_solve(10, 1, 100).
Perception: cold | Guessed: 50
Perception: cold | Guessed: 25
Perception: hot | Guessed: 12
Perception: hot | Guessed: 18
Perception: hot | Guessed: 15
Perception: cold | Guessed: 8
Goal Reached: 10
true.
```

Agent Example 2: Stateful Memory

Dynamic Knowledge Base Setup

- To let our agent remember its search limits without passing them as arguments, we define dynamic predicates.

```
:- dynamic current_min/1.
:- dynamic current_max/1.

% Initialize the agent's memory
init_agent(Min, Max) :-
    retractall(current_min(_)), % Clear old memory
    retractall(current_max(_)),
    assertz(current_min(Min)),  % Store new limits
    assertz(current_max(Max)).
```

Stateful Actions

- The agent reads its state from the knowledge base, calculates the new bounds, and updates its memory.

```
% act(Hint, Guess, NextGuess)
act(hot, Guess, NextGuess) :-
    current_min(Min),
    NewMax is Guess - 1,
    retractall(current_max(_)),    % Forget old max
    assertz(current_max(NewMax)),  % Remember new max
    NextGuess is (Min + NewMax) // 2.

act(cold, Guess, NextGuess) :-
    current_max(Max),
    NewMin is Guess + 1,
    retractall(current_min(_)),    % Forget old min
    assertz(current_min(NewMin)),  % Remember new min
    NextGuess is (NewMin + Max) // 2.
```

Stateful Recursive Loop

```
% Starter function
smart_solve_state(Secret, MinLimit, MaxLimit) :-
    init_agent(MinLimit, MaxLimit),
    InitialGuess is (MinLimit + MaxLimit) // 2,
    solve_state(Secret, InitialGuess).

% Base Case
solve_state(Secret, Guess) :-
    perceive_hint(Secret, Guess, found),
    write('Goal Reached: '), write(Guess), !.

% Recursive Step
solve_state(Secret, Guess) :-
    perceive_hint(Secret, Guess, Hint),
    write('Perception: '), write(Hint), write(' | Guessed: '), write(Gu
    act(Hint, Guess, NextGuess),
    solve_state(Secret, NextGuess).
```

Advanced Prolog Constructs

DCG (Definite Clause Grammars)

```prolog
% command(Action) --> list of words
command(find(N)) --> [search, for], number(N).
number(N) --> [N], {number(N)}.
% Usage:
% phrase(command(A), [search, for, 50]). % A = find(50)
```

Probabilistic Logic

- Standard Prolog is yes/no; agents handling uncertain info use probabilistic logic programming (e.g., ProbLog).

Prolog and Python Integration

Using Prolog from Python with PySwip

- Install: `pip install pyswip` (requires swipl in PATH)

Example

Suppose you have `family.pl`:

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

Python:

```
from pyswip import Prolog
prolog = Prolog()
prolog.consult('family.pl')
gps = list(prolog.query("grandparent(X, Y)"))
for solution in gps:
    print(f"{solution['X']} is grandparent of {solution['Y']}")
```