(Optional) Avoiding Poisonous Food - Prolog Example

This directory contains a Prolog implementation of the food safety reasoning system, demonstrating the same Domain → Language → Data approach as the OWL ontology example.

- Prolog is a logic programming language well-suited for knowledge representation and reasoning.
- We can get the similar results using Prolog facts and rules.

Overview

This example shows how Prolog logic programming can reason about food safety:

- **Inference 1**: Eating mushroom is UNSAFE (because mushrooms are poisonous)
- **Inference 2**: Eating smelly meat is UNSAFE (because smelly meat is dangerous)

Quick Start

```
# First time setup (installs SWI-Prolog)
chmod +x setup.sh run.sh
./setup.sh

# Run the demonstration
./run.sh
```

Files

- **setup.sh** - Installation script for SWI-Prolog (Ubuntu & macOS)
- **run.sh** - Main runner script
- **food_safety.pl** - Prolog knowledge base

How It Works

1. Domain Level (Human Knowledge)

From experience, we know:

- "Poisonous mushrooms are dangerous to eat"
- "Meat with a weird smell is dangerous"

2. Language Level (Formal Meaning)

In Prolog, we define concepts and rules:

Concepts (Facts and Rules):

```
% Define what things are food
is_food(X) :- mushroom(X).
is_food(X) :- meat(X).
is_food(X) :- fruit(X).
```

Rules (Logic):

```
% Rule 1: If a food is poisonous, then it is unsafe
unsafe(Food) :-
    is_food(Food),
    is_poisonous(Food).

% Rule 2: If a food is smelly, then it is unsafe
unsafe(Food) :-
    is_food(Food),
    is_smelly(Food).

% If not unsafe, then it's safe
safe(Food) :-
    is_food(Food),
    \+ unsafe(Food).
```

Prolog Grammar

1. Anything that starts with a capital letter is a variable (e.g., `Food`, `X`).
2. Lowercase words are constants or predicates (e.g., `mushroom`, `is_poisonous`).
3. `:-` means "if" (implication).
4. `,` means "and".
5. `\+` means "not" (negation).

3. Data Level (Actual Facts)

We have specific instances:

```
% Instance 1: A specific mushroom
mushroom(mushroom1).
is_poisonous(mushroom1).

% Instance 2: A specific meat
meat(meat1).
is_smelly(meat1).

% Instance 3: A safe apple
fruit(apple1).
```

4. Reasoning Result

Prolog automatically infers:

- ✗ `mushroom1` is **UNSAFE** (because is_poisonous)
- ✗ `meat1` is **UNSAFE** (because is_smelly)
- ✓ `apple1` is **SAFE** (no dangerous properties)

Example Output

```
============================================================
FOOD SAFETY KNOWLEDGE BASE - PROLOG REASONING DEMONSTRATION
============================================================

This demonstrates how Prolog can reason about food safety
using the same Domain -> Language -> Data approach as OWL.

------------------------------------------------------------
1. UNSAFE FOOD INSTANCES (Inferred by Prolog)
------------------------------------------------------------
   ✗ mushroom1 is UNSAFE
     Reason: because it is poisonous
   ✗ meat1 is UNSAFE
     Reason: because it is smelly

------------------------------------------------------------
2. SAFE FOOD INSTANCES
------------------------------------------------------------
   ✓ mushroom2 is SAFE
   ✓ meat2 is SAFE
   ✓ apple1 is SAFE
   ✓ banana1 is SAFE
```

```
------------------------------------------------------------
3. DETAILED ANALYSIS
------------------------------------------------------------

Analyzing: mushroom1
  Type: mushroom
  Properties: [poisonous]
  Status: UNSAFE
  Reason: it is poisonous

Analyzing: meat1
  Type: meat
  Properties: [smelly]
  Status: UNSAFE
  Reason: it is smelly

Analyzing: apple1
  Type: fruit
  Properties: none
  Status: SAFE
```

```
------------------------------------------------------------
4. INFERENCE EXPLANATION
------------------------------------------------------------
Prolog applied these rules:
   Rule 1: IF (Food is poisonous) THEN (Food is unsafe)
   Rule 2: IF (Food is smelly) THEN (Food is unsafe)

Applied to our data:
   • mushroom1: isPoisonous=true → INFERRED: UNSAFE
   • meat1: isSmelly=true → INFERRED: UNSAFE
   • apple1: no dangerous properties → SAFE
```

Interactive Mode

You can also run Prolog interactively to explore:

```
swipl -s food_safety.pl
```

Then try these queries:

```
?- demonstrate.                % Run full demonstration
?- unsafe(mushroom1).          % Check if mushroom1 is unsafe (true)
?- safe(apple1).               % Check if apple1 is safe (true)
?- find_all_unsafe(X).         % Find all unsafe foods
?- find_all_safe(X).           % Find all safe foods
?- explain_unsafe(mushroom1). % Get detailed explanation
?- is_food(X).                 % List all food items
?- food_type(mushroom1, Type). % Get food type
```

Prolog vs OWL Comparison

Similarities:

- Both use the Domain → Language → Data approach
- Both perform automatic reasoning
- Both infer new facts from rules
- Both handle the same use case

Differences:

Aspect	Prolog	OWL
Paradigm	Logic Programming	Description Logic
Syntax	Horn clauses	RDF/XML or Turtle
Reasoning	Backward chaining	Forward/Backward
Queries	Interactive predicates	SPARQL queries
Use Case	AI, Expert Systems	Semantic Web

When to Use Prolog:

- Building expert systems
- Natural language processing
- AI applications
- Academic research
- Quick prototyping

When to Use OWL:

- Semantic web applications
- Enterprise knowledge bases
- Data integration
- Interoperability between systems
- Standards compliance

Learning Points

1. **Logic Programming Paradigm**:
    - Facts: ground truths (mushroom(mushroom1))
    - Rules: logical implications (unsafe(X) :- ...)
    - Queries: questions to prove (?- unsafe(mushroom1))
2. **Declarative vs Procedural**:
    - Prolog: "What is true" (declarative)
    - Python/Java: "How to compute" (procedural)

3. **Automatic Reasoning**:
    - No explicit algorithms needed
    - Prolog engine handles inference
    - Rules automatically applied
4. **Pattern Matching**:
    - Variables (X, Food) match patterns
    - Unification finds solutions
    - Backtracking explores alternatives

5. **Practical Applications**:
    - Medical diagnosis systems
    - Rule-based decision making
    - Natural language understanding
    - Game AI and planning

Installation Notes

Ubuntu/Debian

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

macOS (with Homebrew)

```
brew install swi-prolog
```

Windows

Download from: https://www.swi-prolog.org/Download.html

Requirements

- SWI-Prolog 8.0+ (tested with 8.4+)
- Bash shell (for running scripts)
- Unix-like environment (Linux, macOS, WSL on Windows)

Extending the Example

Try adding:

1. **New food types**:

```
vegetable(carrot1).
is_food(X) :- vegetable(X).
```

2. **New properties**:

```
is_expired(milk1).
```

3. **New rules**:

```
% Expired food is unsafe
unsafe(Food) :-
    is_food(Food),
    is_expired(Food).
```

4. **More complex reasoning**:

```
% Food is edible only if safe and fresh
edible(Food) :-
    safe(Food),
    \+ is_expired(Food).
```

Troubleshooting

SWI-Prolog not found

```
# Check if installed
which swipl

# If not installed, run setup
./setup.sh
```

Permission denied

```
chmod +x setup.sh run.sh
```

Syntax errors

Run interactively to see detailed error messages:

```
swipl -s food_safety.pl
```

Key Differences from OWL Example

1. **Syntax**: Prolog uses logic clauses instead of RDF triples
2. **Execution**: Interactive queries vs batch reasoning
3. **Paradigm**: Logic programming vs knowledge graphs
4. **Tools**: SWI-Prolog vs Python/rdflib

Both achieve the same reasoning results, showing two different approaches to knowledge representation and reasoning!

References

- SWI-Prolog Documentation
- Learn Prolog Now!
- Logic Programming Tutorial
- Prolog Tutorial (SWI)