Avoiding Poisonous Food - Ontology/OWL Reasoning Example

Food Safety Ontology - OWL Reasoning Example

Overview

The example in shows how OWL ontologies can help machines reason about domain knowledge, specifically:

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

This is an practical demonstration of the concepts from the lecture "Avoiding Poisonous Food".

Quick Start

```
# First time setup
chmod +x setup.sh run.sh
./setup.sh

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

Files

- **setup.sh** - Initial setup script (creates venv, installs packages)
- **run.sh** - Main runner script (runs the demo)
- **reason.py** - Python script that performs OWL-RL reasoning
- **food_safety.ttl** - Ontology in Turtle format (human-readable)
- **food_safety.rdf** - Ontology in RDF/XML format (standard XML)
- **venv/** - Virtual environment (auto-created by setup.sh)

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)

We define concepts and rules:

Concepts (Classes):

- `Food` - something a person can eat
- `Mushroom` - a type of food
- `Meat` - a type of food
- `UnsafeFood` - food that is NOT OK to eat

Properties:

- `isPoisonous` - indicates if food contains poison (boolean)
- `isSmelly` - indicates if food has bad smell (boolean)

Rules (OWL Restrictions):

```
Rule 1: IF (Food AND isPoisonous=true) THEN UnsafeFood
Rule 2: IF (Food AND isSmelly=true) THEN UnsafeFood
```

3. Data Level (Actual Facts)

We have specific instances:

- `mushroom1` : type=Mushroom, isPoisonous=true
- `meat1` : type=Meat, isSmelly=true
- `apple1` : type=Food, isPoisonous=false, isSmelly=false

4. Reasoning Result

The OWL reasoner automatically infers:

- ✕ `mushroom1` is **UnsafeFood** (because isPoisonous=true)
- ✕ `meat1` is **UnsafeFood** (because isSmelly=true)
- ✓ `apple1` remains **SAFE** (no dangerous properties)

Example Output

```
======================================================================
FOOD SAFETY ONTOLOGY - OWL REASONING DEMONSTRATION
======================================================================

Processing: food_safety.ttl
Format: turtle

Original triples: 45
After reasoning: 78 triples

======================================================================
REASONING RESULTS
======================================================================

1. UNSAFE FOOD INSTANCES (Inferred by Reasoner):
----------------------------------------------------------------------
   ✗ Mushroom 1 is UNSAFE
     Reason: because it is poisonous
   ✗ Meat 1 is UNSAFE
     Reason: because it is smelly

2. SAFE FOOD INSTANCES:
----------------------------------------------------------------------
   ✓ Apple 1 is SAFE

3. INFERENCE EXPLANATION:
----------------------------------------------------------------------
The reasoner 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: UnsafeFood
   • meat1: isSmelly=true → INFERRED: UnsafeFood
   • apple1: isPoisonous=false, isSmelly=false → SAFE
```

Technical Details

There are two widely used formats for OWL ontologies: turtle (.ttl) and RDF/XML (.rdf).

- food_safety.ttl - human-readable Turtle format
- food_safety.rdf - standard RDF/XML format

They represent the same ontology and produce identical reasoning results.

OWL Restrictions

The ontology uses OWL property restrictions to define rules (in Turtle format):

```
# Rule 1: Poisonous food is unsafe
[ rdf:type owl:Class ;
  owl:intersectionOf (
    :Food
    [ rdf:type owl:Restriction ;
      owl:onProperty :isPoisonous ;
      owl:hasValue "true"^^xsd:boolean
    ]
  ) ;
  rdfs:subClassOf :UnsafeFood
] .
```

This means: "Any individual that is both a Food AND has isPoisonous=true is also a member of UnsafeFood class."

- rdf: and owl: are standard RDF and OWL namespaces
- rdfs: is the RDF Schema namespace
- xsd: is the XML Schema datatype namespace
- :Food, :isPoisonous, :UnsafeFood are defined in our ontology namespace

- Namespaces help avoid naming conflicts and clarify term meanings.
- Many concepts are already defined in standard namespaces (rdf:, rdfs:, owl:, xsd:).
- When we need the concept of "Food", we refer to it as ":Food" in our ontology namespace.

We can use the namespaces in different levels of detail (XML, RDF, or OWL) depending on the audience.

Reasoning Engine in Python

This example uses:

- **rdflib**: Python library for working with RDF
- **owlrl**: OWL-RL reasoning engine that implements Description Logic rules

The reasoner performs:

1. Class subsumption
2. Property restrictions evaluation
3. Instance classification

Learning Points

1. **Three Levels of Knowledge Representation**:
    - Domain Level: Natural language knowledge
    - Language Level: Formal definitions and rules
    - Data Level: Concrete facts
2. **OWL Reasoning**:
    - Ontologies capture domain knowledge formally
    - Reasoners automatically derive **new facts from rules**
    - Same ontology can be expressed in different formats

3. **Classes vs Instances**:
    - Classes: `Food`, `Mushroom`, `UnsafeFood` (concepts)
    - Instances: `mushroom1`, `meat1`, `apple1` (specific examples)
4. **Properties and Restrictions**:
    - Properties define relationships
    - Restrictions define rules using property values
    - OWL reasoner applies rules automatically

5. **Practical Application**:
    - Medical diagnosis systems
    - Product recommendation engines
    - Safety and compliance systems
    - Knowledge bases for expert systems

Format Comparison

Turtle Format (.ttl)

```
:mushroom1 rdf:type :Mushroom ;
    rdfs:label "Mushroom 1" ;
    :isPoisonous "true"^^xsd:boolean .
```

Advantages:

- Compact and readable
- Easier for humans to write
- Uses prefixes for namespaces
- Popular in academic contexts

RDF/XML Format (.rdf)

```
<Mushroom rdf:about="http://example.org/food#mushroom1">
    <rdfs:label>Mushroom 1</rdfs:label>
    <isPoisonous rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">
        true
    </isPoisonous>
</Mushroom>
```

Advantages:

- Standard XML format
- Better tool support
- Machine-friendly
- Common in enterprise applications

Both formats represent the same ontology and produce **identical reasoning results**!

Extending the Example

Try modifying the ontology to add:

1. **New food types**:

```
:Vegetable rdf:type owl:Class ;
           rdfs:subClassOf :Food .
```

2. **New properties**:

```
:isExpired rdf:type owl:DatatypeProperty ;
           rdfs:domain :Food ;
           rdfs:range xsd:boolean .
```

3. **New rules**:

Expired food is unsafe food.

```
[ rdf:type owl:Class ;
  owl:intersectionOf (:Food
    [owl:onProperty :isExpired;
     owl:hasValue true]) ;
  rdfs:subClassOf :UnsafeFood ] .
```

- `:Food` → refers to the general class of food items.
- `owl:onProperty :isExpired; owl:hasValue true` → specifies that the `isExpired` property is true for this class.
- `owl:intersectionOf` → combines these conditions (it must be food and expired).
- `rdfs:subClassOf :UnsafeFood` → says that such items belong to, or are a subset of, the class `UnsafeFood`.

4. **More instances**:

`carrot1` is a vegetable. It is not poisonous and not smelly.

```
:carrot1 rdf:type :Vegetable ;
    :isPoisonous false ;
    :isSmelly false .
```

- `:carrot1 rdf:type :Vegetable` → carrot1 belongs to the class of vegetables.
- `:isPoisonous false` → it is not poisonous.
- `:isSmelly false` → it does not have a bad smell.

Requirements

- Python 3.x (tested with 3.8+)
- pip (Python package manager)
- Bash shell (for running scripts)

Python packages (auto-installed by setup.sh):

- rdflib >= 6.0.0
- owlrl >= 6.0.0

Troubleshooting

Virtual Environment Issues

```
# Remove and recreate
rm -rf venv
./setup.sh
```

Package Installation Issues

```
# Manually activate venv and install
source venv/bin/activate
pip install --upgrade pip
pip install rdflib owlrl
```

Permission Issues

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

Understanding the Example

This example is designed for teaching software engineering students about:

- Semantic Web technologies
- Knowledge representation
- Automated reasoning
- Ontology engineering
- Real-world AI applications

The code includes extensive comments and follows the three-level model (Domain → Language → Data) from the lecture materials.

References

- OWL 2 Web Ontology Language Primer
- RDFLib Documentation
- OWL-RL Documentation
- Turtle Syntax
- RDF/XML Syntax