(Optional) # What is Ontology? - Example Part 2

Code location: `ontology/`

Files in the Example

```
ontology/
├── computers.ttl            # Ontology with OWL definitions
├── query.sparql             # Semantic query
├── query_manual.sparql      # Manual filter (comparison)
├── run.py                   # Python script with reasoning
├── run.sh                   # Quick start script
├── demo_reasoning.py        # Step-by-step demo
└── README.md                # Complete documentation
```

Quick Start Commands

```
# Navigate to the directory
cd topics/ontology/code/ontology/

# Install required libraries
pip install rdflib owlrl

# Run the main demo
python3 run.py

# Or use the shell script
chmod +x run.sh
./run.sh

# For interactive step-by-step demo
python3 demo_reasoning.py
```

Understanding OWL Restrictions

AffordableComputer Definition

```
:AffordableComputer a owl:Class ;
    owl:equivalentClass [
        owl:intersectionOf (
            :Computer                    # Must be a Computer
            [
                owl:onProperty :hasPrice  # Check price property
                owl:someValuesFrom [       # Has some value that...
                    owl:onDatatype xsd:integer
                    owl:withRestrictions (
                        [ xsd:maxExclusive 1000 ]  # < 1000
                    )
                ]
            ]
        )
    ] .
```


Translation: "AffordableComputer IS any Computer that has a price less than 1000"

Understanding OWL Restrictions (cont.)

GamingComputer Definition

```
:GamingComputer a owl:Class ;
    owl:equivalentClass [
        owl:intersectionOf (          # ALL of these must be true
            :Computer                  # 1. Must be a Computer
            [
                owl:onProperty :hasRAM
                owl:someValuesFrom [
                    owl:withRestrictions (
                        [ xsd:minInclusive 16 ]  # RAM ≥ 16
                    )
                ]
            ]
            [
                owl:onProperty :hasGPU
                owl:someValuesFrom :DedicatedGPU  # 2. Has DedicatedGPU
            ]
        )
```

The Reasoning Process Step-by-Step

Step 1: Load Data (89 triples)

```
Computer1 is a Laptop.
Computer1 hasPrice 899.
Computer1 hasRAM 16.
Computer1 hasGPU RTX4060.
RTX4060 is a DedicatedGPU.
...
```

Step 2: Apply Reasoner

```
from owlrl import DeductiveClosure, OWLRL_Semantics
DeductiveClosure(OWLRL_Semantics).expand(g)
```

Step 3: Reasoner Checks Each Computer

For Computer1 (ASUS TUF Gaming):

```
Check AffordableComputer:
  ✓ Is it a Computer? YES (Laptop → Computer)
  ✓ Has price < 1000? YES (899 < 1000)
  → INFER: Computer1 is AffordableComputer

Check GamingComputer:
  ✓ Is it a Computer? YES
  ✓ Has RAM ≥ 16? YES (16 ≥ 16)
  ✓ Has DedicatedGPU? YES (RTX4060)
  → INFER: Computer1 is GamingComputer
```

Step 4: After Reasoning (147 triples)

New inferred triples:

```
Computer1 is AffordableComputer.  # INFERRED!
Computer1 is GamingComputer.       # INFERRED!
Computer2 is AffordableComputer.  # INFERRED!
Computer2 is GamingComputer.       # INFERRED!
Computer3 is GamingComputer.       # INFERRED! (not affordable)
Computer4 is AffordableComputer.  # INFERRED! (not gaming)
Computer5 is AffordableComputer.  # INFERRED! (not gaming)
```

58 new triples automatically added!

Python Implementation

```python
from rdflib import Graph
from owlrl import DeductiveClosure, OWLRL_Semantics

# 1. Load ontology
g = Graph()
g.parse("computers.ttl", format="turtle")
print(f"Loaded {len(g)} triples")  # 89 triples

# 2. Apply reasoning
DeductiveClosure(OWLRL_Semantics).expand(g)
print(f"After reasoning: {len(g)} triples")  # 147 triples

# 3. Run semantic query
with open("query.sparql", 'r') as f:
    query = f.read()
results = g.query(query)

# 4. Display results
for row in results:
    print(f"{row.brand} {row.model} - ${row.price}")
```

Two Query Approaches

Method 1: Semantic (query.sparql)

```
SELECT ?brand ?model ?price ?ram WHERE {
    ?computer rdf:type :AffordableComputer .
    ?computer rdf:type :GamingComputer .
    ?computer :hasBrand ?brand .
    ?computer :hasModel ?model .
    ?computer :hasPrice ?price .
    ?computer :hasRAM ?ram .
}
```

Uses **inferred classifications** from reasoning!

Method 2: Manual (query_manual.sparql)

```
SELECT ?brand ?model ?price ?ram WHERE {
    ?computer rdf:type :Computer .
    ?computer :hasBrand ?brand .
    ?computer :hasModel ?model .
    ?computer :hasPrice ?price .
    ?computer :hasRAM ?ram .
    ?computer :hasGPU ?gpu .
    ?gpu rdf:type :DedicatedGPU .

    FILTER(?price < 1000 && ?ram >= 16)  # Manual rules!
}
```

Must **hard-code the rules** in every query!

Expected Output Comparison

Both methods return the same results:

```
METHOD 1: Semantic Query
================================================================================
  1. ASUS TUF Gaming A15
     Price: $899, RAM: 16GB

  2. CyberPowerPC Gamer Xtreme
     Price: $949, RAM: 16GB

Total: 2 computer(s) found

METHOD 2: Manual Filter
================================================================================
  1. ASUS TUF Gaming A15
     Price: $899, RAM: 16GB

  2. CyberPowerPC Gamer Xtreme
     Price: $949, RAM: 16GB
```

But Method 1 is Better!

Maintainability Example

Scenario: Change "affordable" from <$1000 to <$1200

Method 1 (Semantic): Change 1 line

```
# In computers.ttl
owl:withRestrictions ( [ xsd:maxExclusive 1200 ] )  # was 1000
```

All queries using `:AffordableComputer` automatically updated!

Method 2 (Manual): Change N lines

```
# In every query file (maybe 100 files!)
FILTER(?price < 1200 && ...)  # was 1000
```

Interactive Demo

`demo_reasoning.py` shows the process step-by-step:

```
Step 1: Load the ontology
  Loaded 89 triples

Step 2: Check computers BEFORE reasoning
  AffordableComputers found: 0
  GamingComputers found: 0
  ❌ No computers classified yet!

Step 3: Apply OWL Reasoning
  [Press Enter to continue...]
  ✓ Reasoning complete! Graph now has 147 triples

Step 4: Check computers AFTER reasoning
  AffordableComputers found: 4
  GamingComputers found: 3

Step 5: Query for Affordable Gaming Computers
```

Exercise 1: Add a New Computer

Add this to `computers.ttl`:

```
:Computer6 a :Laptop ;
    :hasBrand "Lenovo" ;
    :hasModel "Legion 5" ;
    :hasPrice 1099 ;
    :hasRAM 16 ;
    :hasGPU :RTX4060 .
```

Questions:

- Will it be classified as AffordableComputer?
- Will it be classified as GamingComputer?
- Will it appear in our query results?

Exercise 1: Answer

```
Computer6 (Lenovo Legion 5):
  Price: 1099
  RAM: 16
  GPU: RTX4060 (DedicatedGPU)

AffordableComputer check:
  ✓ Is Computer? YES
  ✗ Price < 1000? NO (1099 ≥ 1000)
  → NOT AffordableComputer

GamingComputer check:
  ✓ Is Computer? YES
  ✓ RAM ≥ 16? YES (16 ≥ 16)
  ✓ Has DedicatedGPU? YES
  → IS GamingComputer

Result: Will NOT appear in query (not affordable!)
```

Exercise 2: Define BudgetComputer

Create a new semantic class: **BudgetComputer** = Computer with price < $500

```
:BudgetComputer a owl:Class ;
    owl:equivalentClass [
        owl:intersectionOf (
            :Computer
            [
                owl:onProperty :hasPrice ;
                owl:someValuesFrom [
                    owl:onDatatype xsd:integer ;
                    owl:withRestrictions (
                        [ xsd:maxExclusive 500 ]
                    )
                ]
            ]
        )
    ] .
```


Which computers will be BudgetComputer?

Exercise 2: Answer

Check each computer:

Computer	Price	Budget?
ASUS TUF Gaming A15	$899	✗
CyberPowerPC Gamer Xtreme	$949	✗
Alienware M15 R7	$1499	✗
HP Pavilion 15	$599	✗
Dell Inspiron 15	$449	✓

Only **Dell Inspiron 15** ($449) will be BudgetComputer!

Exercise 3: Define and Query WorkLaptop

Define: WorkLaptop = Laptop with RAM ≥ 16GB (any GPU)

```
:WorkLaptop a owl:Class ;
    owl:equivalentClass [
        owl:intersectionOf (
            :Laptop
            [
                owl:onProperty :hasRAM ;
                owl:someValuesFrom [
                    owl:onDatatype xsd:integer ;
                    owl:withRestrictions (
                        [ xsd:minInclusive 16 ]
                    )
                ]
            ]
        )
    ] .
```

Exercise 3: Query

```
PREFIX : <http://example.org/computers#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?brand ?model ?price ?ram ?gpu
WHERE {
    ?computer rdf:type :WorkLaptop .
    ?computer :hasBrand ?brand .
    ?computer :hasModel ?model .
    ?computer :hasPrice ?price .
    ?computer :hasRAM ?ram .
    ?computer :hasGPU ?gpu .
}
ORDER BY ?price
```

Exercise 3: Answer

WorkLaptops (Laptop with RAM ≥ 16):

Computer	Price	RAM	GPU
HP Pavilion 15	$599	16GB	Intel UHD
ASUS TUF Gaming A15	$899	16GB	RTX4060
CyberPowerPC	$949	16GB	GTX1660
Alienware M15 R7	$1499	32GB	RTX4060

4 computers! (All except Dell Inspiron with 8GB RAM)

Common Pitfalls

1. Forgetting Reasoning

```
# ❌ Wrong - query before reasoning
g.parse("computers.ttl")
results = g.query(query)  # Returns 0 results!

# ✓ Correct - apply reasoning first
g.parse("computers.ttl")
DeductiveClosure(OWLRL_Semantics).expand(g)
results = g.query(query)  # Returns 2 results!
```

2. Using Wrong Restriction Type

```
# ❌ Wrong - checks for exact value
owl:hasValue 899

# ✓ Correct - checks constraint
owl:someValuesFrom [ xsd:maxExclusive 1000 ]
```

Summary of Key Concepts

Concept	Traditional DB	Ontology
Data Structure	Tables, rows	Classes, instances, triples
Queries	SQL with filters	SPARQL with concepts
Rules	In application code	In ontology (OWL)
Reasoning	None	Automatic inference
Maintenance	Change N queries	Change once
Semantics	Implicit	Explicit

Benefits Recap

✓ **Semantic Clarity**

- "AffordableComputer" vs "price < 1000"

✓ **Maintainability**

- Change once, all queries benefit

✓ **Reasoning**

- Automatic classification based on rules

✓ **Interoperability**

- Share domain knowledge across systems

✓ **Extensibility**

- Easy to add new defined classes

Real-World Applications Revisited

E-commerce

```
:PremiumProduct ≡ Product with price > 1000 AND rating > 4.5
:BudgetProduct ≡ Product with price < 100
:PopularProduct ≡ Product with reviews > 100
```

Query: "Find premium products on sale"

```
SELECT ?product WHERE {
  ?product rdf:type :PremiumProduct .
  ?product :onSale true .
}
```

Healthcare

```
:HighRiskPatient ≡ Patient with (age > 65 OR hasCondition Diabetes)
:UrgentCase ≡ Patient with triageLevel = "Critical"
```

Query: "Find high-risk patients needing urgent care"

```
SELECT ?patient WHERE {
  ?patient rdf:type :HighRiskPatient .
  ?patient rdf:type :UrgentCase .
}
```

Definitions in ONE place, used everywhere!

Final Thoughts

Question: "Find affordable gaming computers"

Traditional Approach:

```
WHERE price < 1000 AND ram >= 16 AND gpu = 'dedicated'
```

Hard-coded rules everywhere!

Ontology Approach:

```
WHERE ?computer rdf:type :AffordableComputer .
      ?computer rdf:type :GamingComputer .
```

Define once, query semantically!

This is the power of the Semantic Web!