CITY
UNIVERSITY OF LONDON
EST 1894

# RDFS Semantics and OWL 2 Profiles

**Ernesto Jiménez-Ruiz**

Lecturer in Artificial Intelligence

# Before we start...

# Drop-in sessions

– Today from **3pm** to **5pm** (shifted by 1h).

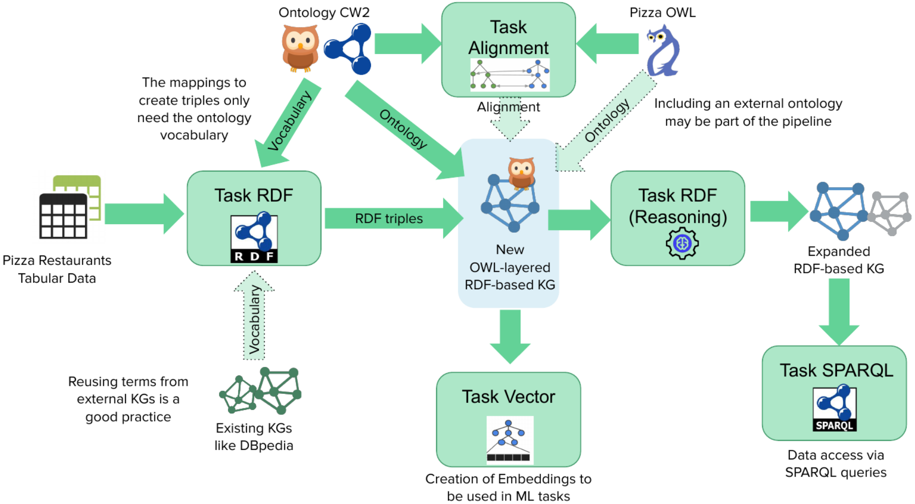
– **Tuesdays, online 10am**.

# Where are we? Module organization.

✓ Introduction: Becoming a knowledge scientist.
✓ RDF-based knowledge graphs.
✓ OWL ontology language. Focus on modelling.
✓ SPARQL 1.0 Query Language.
✓ From tabular data to KG.

6. **RDFS Semantics and OWL 2 profiles.**(Today)
7. SPARQL 1.1, Rules and Graph Database solutions.
8. Ontology Alignment.
9. Ontology (KG) Embeddings and Machine Learning.
10. (Large) Language Models and KGs. (Seminar)

# The global picture



# Coursework part 2

– Sunday, 12 May 2024, 5:00 PM

– Team registration March 17

– Components:
  - ✓ Tabular Data to Knowledge Graph: 40% (Weeks 2 and 5)
  - – SPARQL and Reasoning: 20% (Weeks 4, 7 and 8)
  - – Ontology Alignment: 10% (Week 9)
  - – Ontology Embeddings: 10% (Week 10)

## Learning outcomes for today

- Light understanding of how reasoning works.
- Usefulness of OWL 2 Profiles (specially OWL 2 RL)
- Learning how to do reasoning programmatically (coursework): basically 1 line of code.

# Recap

# Recap: RDF Example

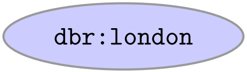
**London is a city in England called Londres in Spanish**

```
dbr:london a dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```

# Recap: RDF Example

**London is a city in England called Londres in Spanish**

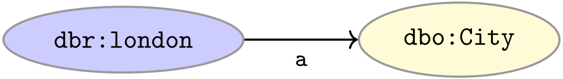
```
dbr:london a dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



# Recap: RDF Example

**London is a city in England called Londres in Spanish**

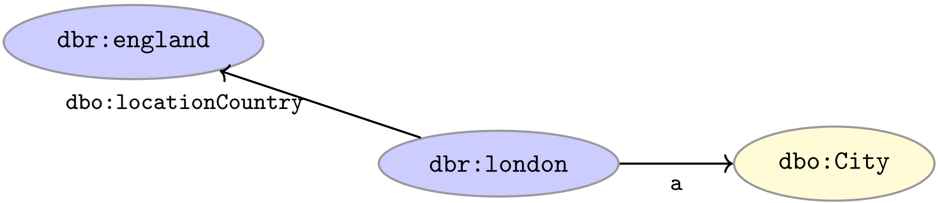
```
dbr:london a dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



# Recap: RDF Example

### London is a city in England called Londres in Spanish

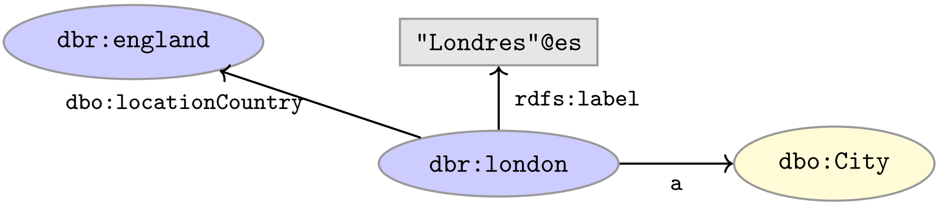
```
dbr:london a dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



# Recap: RDF Example

**London is a city in England called Londres in Spanish**

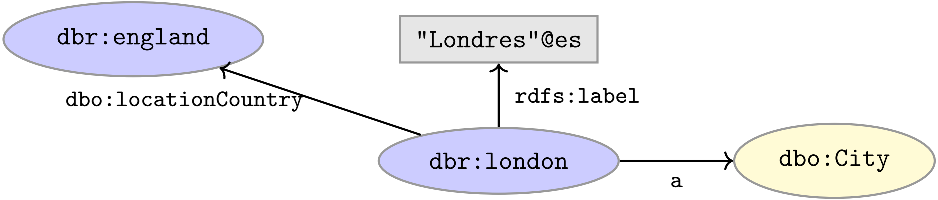
```
dbr:london a dbo:City .
dbr:london dbo:locationCountry dbr:england .
dbr:london rdfs:label "Londres"@es .
```



# Recap: SPARQL Example (i)

## Return all Cities:

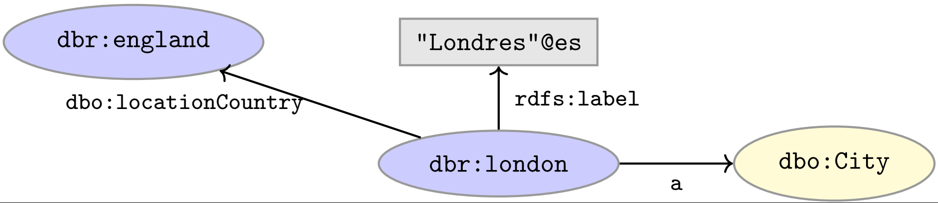
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?city WHERE {
    ?city rdf:type dbo:City .
}
```



# Recap: SPARQL Example (i)

**Return all Cities:** **Query Result= {dbr:london}**

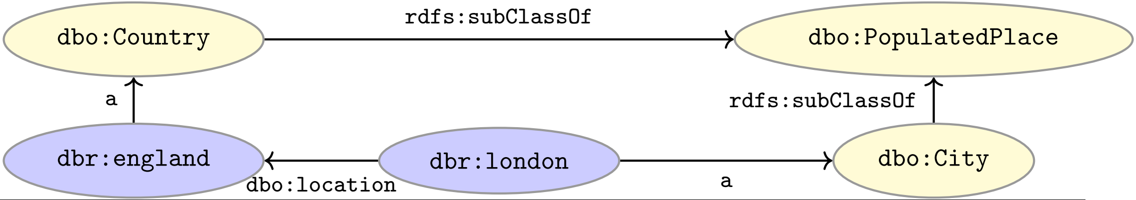
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?city WHERE {
    ?city rdf:type dbo:City .
}
```



# Recap: SPARQL Example (ii)

## Return all Populated Places:

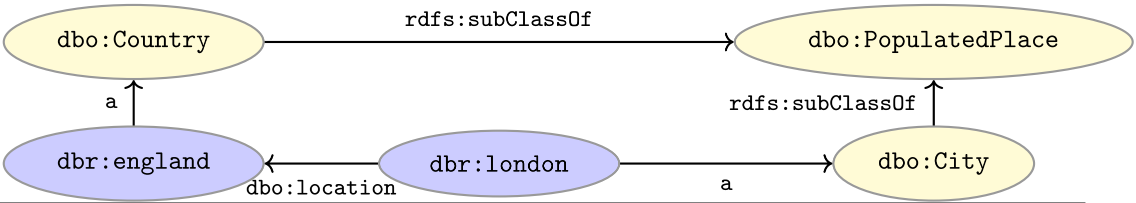
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?place WHERE {
    ?place rdf:type dbo:PopulatedPlace .
}
```



# Recap: SPARQL Example (ii)

**Return all Populated Places: Query Result= {}**

```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?place WHERE {
    ?place rdf:type dbo:PopulatedPlace .
}
```



# Recap: RDF Grammar for triples

– RDF imposes a basic grammar. A triple consists of *subject*, *predicate*, and *object*

| | s | p | o |
|---|---|---|---|
| • URI references may occur in all positions | ✔ | ✔ | ✔ |
| • Literals may only occur in object position | ✘ | ✘ | ✔ |
| • Blank nodes can not occur in predicate position | ✔ | ✘ | ✔ |

– But one could still define:

```
dbr:london rdf:type "some string"^^xsd:string .
```

# Recap: RDF Grammar for triples

– RDF imposes a basic grammar. A triple consists of *subject*, *predicate*, and *object*

| | s | p | o |
|---|---|---|---|
| • URI references may occur in all positions | ✔ | ✔ | ✔ |
| • Literals may only occur in object position | ✘ | ✘ | ✔ |
| • Blank nodes can not occur in predicate position | ✔ | ✘ | ✔ |

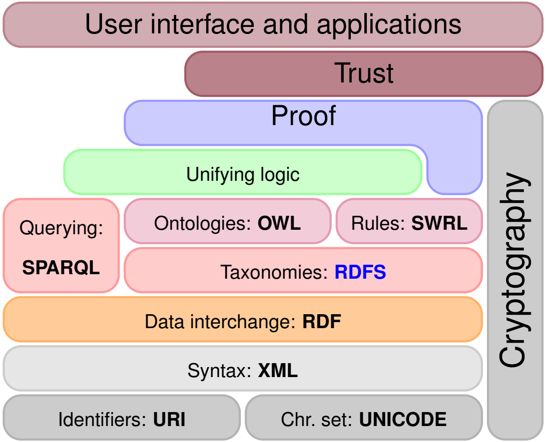
– But one could still define:

```
dbr:london rdf:type "some string"^^xsd:string .
```

– RDF Schema (RDFS) extends the **grammar** for the “**expected**” triples (*e.g.*, `rdf:type rdf:range rdfs:Resource .`), extends the vocabulary, and include a set of **inference rules**.

# RDF Schema (RDFS)

# Semantic Web Technology Stack



# RDF Schema

- RDF Schema (RDFS) is a vocabulary defined by W3C.
  - `https://www.w3.org/TR/rdf-schema/`
  - `https://www.w3.org/TR/rdf11-mt`
- Namespace:
  `rdfs:  http://www.w3.org/2000/01/rdf-schema#`
- Originally designed as a “schema language” like XML Schema.
  - Not strictly – doesn’t describe “valid” RDF graphs.
- A very simple **modeling language** for RDF data $\rightarrow$ Taxonomies
- **(For our purposes)** it can be seen as a language less expressive than OWL, useful to understand how reasoning works.

# RDFS Semantics

- RDFS is a **semantic extension** (adds semantics/meaning) that
  - proposes some **syntactic conditions** on RDF graphs,
  - comes with some (non-ambiguous) **inference rules**, and
  - includes some **(default) triples** as part of the specification.

## RDFS Semantics

- RDFS is a **semantic extension** (adds semantics/meaning) that
  - proposes some **syntactic conditions** on RDF graphs,
  - comes with some (non-ambiguous) **inference rules**, and
  - includes some **(default) triples** as part of the specification.
- For example, RDFS expects as range of `rdf:type` a resource/IRI (*e.g.*, `rdf:type rdf:range rdfs:Resource .`)
  - `dbr:london rdf:type "some string"^^xsd:string .`
  - *RDFS*: Not expected triple, but not prohibited by specification.
  - *OWL*: a prohibited triple will lead to an error (validation).

# RDFS Vocabulary

– RDFS adds the concept of “classes” which are *sets* of resources.

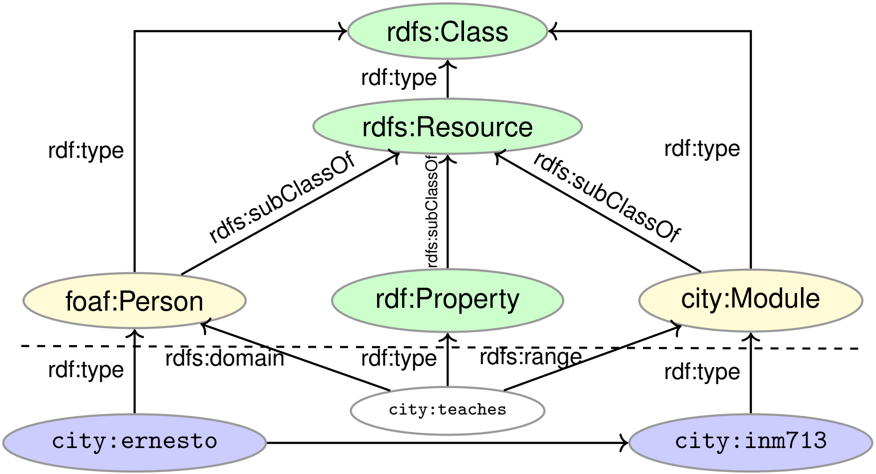
# RDFS Vocabulary

- RDFS adds the concept of “classes” which are *sets* of resources.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything.
  - `rdfs:Class`: The class of classes. (similar to `owl:Class`)
  - `rdfs:Literal`: The class of all literal values.
  - `rdfs:Datatype`: The class of all datatypes.

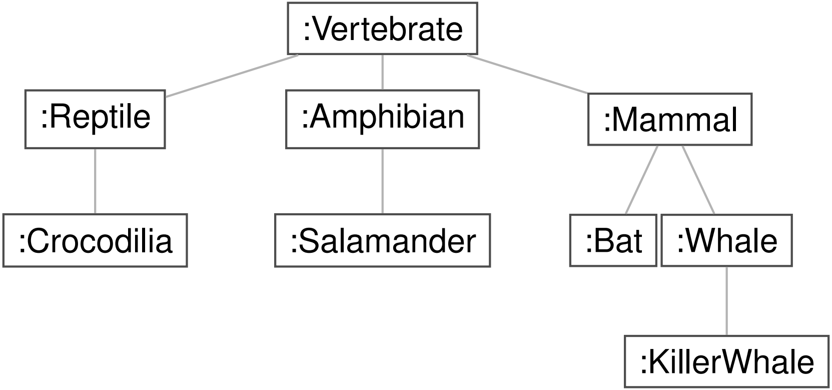
# RDFS Vocabulary

- RDFS adds the concept of “classes” which are *sets* of resources.
- Defined resources:
  - `rdfs:Resource`: The class of resources, everything.
  - `rdfs:Class`: The class of classes. (similar to `owl:Class`)
  - `rdfs:Literal`: The class of all literal values.
  - `rdfs:Datatype`: The class of all datatypes.
- Defined properties:
  - `rdfs:domain`: The domain (sources) of a relation.
  - `rdfs:range`: The range (targets) of a relation.
  - `rdfs:subClassOf`: Class inclusion.
  - `rdfs:subPropertyOf`: Property inclusion.

# Example RDF graph and RDF Schema

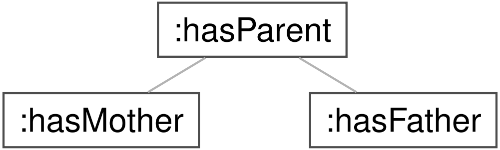


# **Class Taxonomy (`via rdfs:subClassOf`)**




(*) As in OWL.

# Property Taxonomy (`via rdfs:subPropertyOf`)




(*) As in OWL.

# Expected RDF/RDFS resources

Types of resources or elements:

- *Object Properties* like `foaf:knows`
- *Datatype Properties* like `dc:title`, `foaf:name`
- *Classes* like `foaf:Person`
- *Built-ins*, a fixed set including `rdf:type`, `rdfs:domain`, etc.
- *Individuals* (all the rest, "usual" resources) like `city:ernesto`
- *Datatypes* like `xsd:integer`
- *Literals* like `"ernesto"`, `"39"`

(*) Not real split of properties into object and data properties in RDFS. This comes in OWL.

# Expected RDF/RDFS triple grammar

| Triples |
|---|
| `indi o-prop indi .` |
| `indi d-prop "lit" .` |
| `indi rdf:type class .` |
| `class rdfs:subClassOf class .` |
| `o-prop rdfs:subPropertyOf o-prop .` |
| `d-prop rdfs:subPropertyOf d-prop .` |
| `o-prop rdfs:domain class .` |
| `o-prop rdfs:range class .` |
| `d-prop rdfs:domain class .` |
| `d-prop rdfs:range datatype .` |

# (Default) RDFS axiomatic triples (excerpt)

– Indeed RDF and RDFS include a set of default triples to guide the above grammar of expected triples.

– Only resources have types:

```
rdf:type rdfs:domain rdfs:Resource .
```

– Types are classes:

```
rdf:type rdfs:range rdfs:Class .
```

– Ranges apply only to properties:

```
rdfs:range rdfs:domain rdf:Property .
```

RDFS Entailment Rules: `https://www.w3.org/TR/rdf-mt/#rdfs_interp`

# (Default) RDFS axiomatic triples (excerpt)

- Ranges are classes:

  `rdfs:range rdfs:range rdfs:Class .`

- Only properties have subproperties:

  `rdfs:subPropertyOf rdfs:domain rdf:Property .`

- Only classes have subclasses:

  `rdfs:subClassOf rdfs:domain rdfs:Class .`

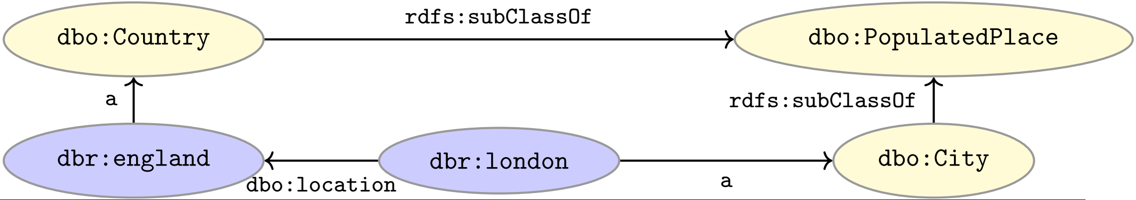
- ... (another 30 or so)

# Entailment via Model-Theoretic Semantics

# SPARQL Example

## Return all Populated Places:

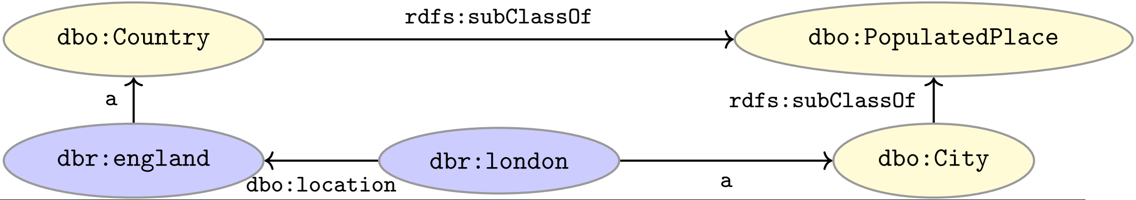
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?place WHERE {
    ?place rdf:type dbo:PopulatedPlace .
}
```



# SPARQL Example

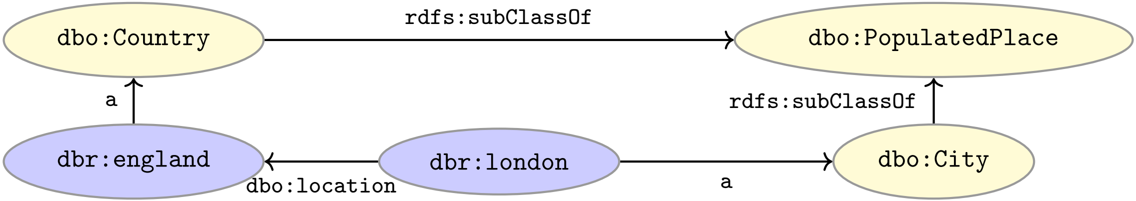
**Return all Populated Places: Query Result= {}**

```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?place WHERE {
    ?place rdf:type dbo:PopulatedPlace .
}
```



# Entailment in RDFS

- Given a set of triples $\mathcal{G}$ (*i.e.*, a Graph) can we entail a triple $t$ ($\mathcal{G} \models t$)?
- Can we entail the triple: `dbr:london rdf:type dbo:PopulatedPlace` and add it to the graph below? *(Graph expansion via reasoning).*
- Similarly for `dbr:england`



# Model-Theoretic Semantics (i)

- **Interpretations** might be conceived as potential "realities" or "worlds".
- Interpretations assign values to elements.
  - *(The **intuitions** behind set-theory are **formally represented**.)*

# Model-Theoretic Semantics (i)

- **Interpretations** might be conceived as potential "realities" or "worlds".
- Interpretations assign values to elements.
  - *(The **intuitions** behind set-theory are **formally represented**.)*
- Given an interpretation $\mathcal{I}$ and a set of triples $\mathcal{G}$
- $\mathcal{G}$ is valid in $\mathcal{I}$ (written $\mathcal{I} \models \mathcal{G}$), iff $\mathcal{I} \models t$ for all $t \in \mathcal{G}$.
- Then $\mathcal{I}$ is also called a **model** of $\mathcal{G}$.

# Model-Theoretic Semantics (ii)

– The following interpretation $\mathcal{I}$ is a model of our example $\mathcal{G}$:
  – $\texttt{dbo:City}^{\mathcal{I}} = \{\texttt{dbr:london}\}$
  – $\texttt{dbo:Country}^{\mathcal{I}} = \{\texttt{dbr:england}\}$
  – $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbr:london}, \texttt{dbr:england}\}$
  – $\texttt{dbo:location}^{\mathcal{I}} = \{\langle\texttt{dbr:london, dbr:england}\rangle\}$

# Model-Theoretic Semantics (ii)

- The following interpretation $\mathcal{I}$ is a model of our example $\mathcal{G}$:
  - $\texttt{dbo:City}^{\mathcal{I}} = \{\texttt{dbr:london}\}$
  - $\texttt{dbo:Country}^{\mathcal{I}} = \{\texttt{dbr:england}\}$
  - $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbr:london}, \texttt{dbr:england}\}$
  - $\texttt{dbo:location}^{\mathcal{I}} = \{\langle\texttt{dbr:london, dbr:england}\rangle\}$
- $\mathcal{I} \models \mathcal{G}$ (is a model of $\mathcal{G}$) as the following holds:
  - $\texttt{dbo:City}^{\mathcal{I}} \subseteq \texttt{dbo:PopulatedPlace}^{\mathcal{I}}$
  - $\texttt{dbo:Country}^{\mathcal{I}} \subseteq \texttt{dbo:PopulatedPlace}^{\mathcal{I}}$
  - $\texttt{dbr:london}^{\mathcal{I}} \in \texttt{dbo:City}^{\mathcal{I}}$

# Model-Theoretic Semantics (iii)

- $t$ = `dbr:london rdf:type dbo:PopulatedPlace`
- Does $\mathcal{I} \models t$ ?

# Model-Theoretic Semantics (iii)

- $t$ = `dbr:london rdf:type dbo:PopulatedPlace`
- Does $\mathcal{I} \models t$ ?
  - Yes: $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbr:london}, \texttt{dbr:england}\}$

# Model-Theoretic Semantics (iii)

- $t$ = `dbr:london rdf:type dbo:PopulatedPlace`
- Does $\mathcal{I} \models t$ ?
  - Yes: $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbr:london}, \texttt{dbr:england}\}$
- Does $\mathcal{G} \models t$ ?

# Model-Theoretic Semantics (iii)

- $t$ = `dbr:london rdf:type dbo:PopulatedPlace`
- Does $\mathcal{I} \models t$ ?
  - Yes: $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbr:london}, \texttt{dbr:england}\}$
- Does $\mathcal{G} \models t$ ?
  - **if and only if**
    - For **any interpretation** $\mathcal{I}$ with $\mathcal{I} \models \mathcal{G}$
    - $\mathcal{I} \models t$.
  - (Yes, in this case too.)

# Model-Theoretic Semantics (iii)

- $t$ = `dbr:london rdf:type dbo:PopulatedPlace`
- Does $\mathcal{I} \models t$ ?
  - Yes: $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbr:london}, \texttt{dbr:england}\}$
- Does $\mathcal{G} \models t$ ?
  - **if and only if**
    - For **any interpretation** $\mathcal{I}$ with $\mathcal{I} \models \mathcal{G}$
    - $\mathcal{I} \models t$.
  - (Yes, in this case too.)
- Does $\mathcal{G} \models t_2$ ($t_2$=`dbr:london rdf:type dbo:Country`)?

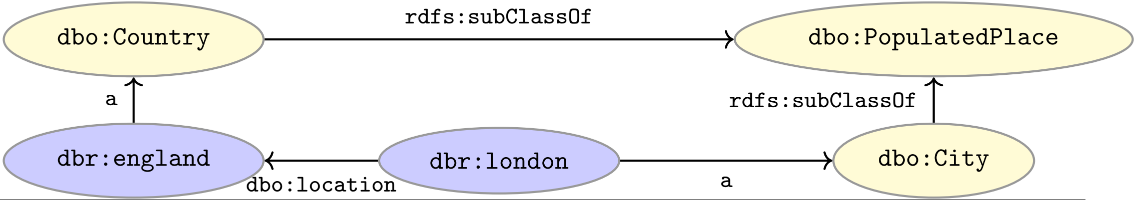
# Model-Theoretic Semantics (iii)

- $t = \texttt{dbr:london rdf:type dbo:PopulatedPlace}$
- Does $\mathcal{I} \models t$ ?
  - Yes: $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbr:london}, \texttt{dbr:england}\}$
- Does $\mathcal{G} \models t$ ?
  - **if and only if**
    - For **any interpretation** $\mathcal{I}$ with $\mathcal{I} \models \mathcal{G}$
    - $\mathcal{I} \models t$.
  - (Yes, in this case too.)
- Does $\mathcal{G} \models t_2$ ($t_2$=$\texttt{dbr:london rdf:type dbo:Country}$)?
  - No: Our $\mathcal{I}$ is a counter example. $\mathcal{I} \models \mathcal{G}$ but $\mathcal{I} \not\models t_2$

# SPARQL Example: with entailment

## Return all Populated places:

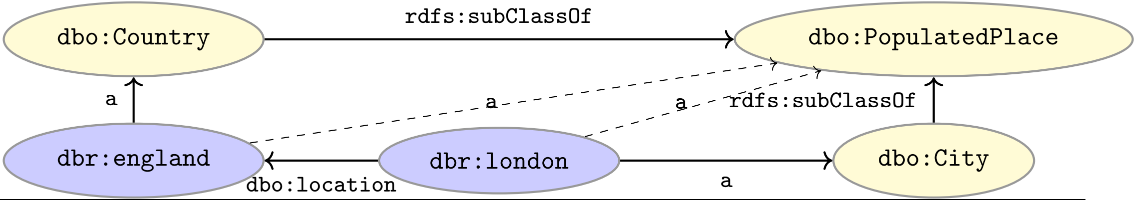
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?place WHERE {
    ?place rdf:type dbo:PopulatedPlace .
}
```



# SPARQL Example: with entailment

**Return all Populated places:** **Query Result= {dbr:england, dbr:london}**

```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?place WHERE {
    ?place rdf:type dbo:PopulatedPlace .
}
```



## Model-Theoretic Semantics in practice

- Model-theoretic semantics yields an unambigous notion of entailment.
- In principle, **all interpretations** need to be considered.
- However there are **infinitely many** such interpretations,
- An **algorithm should terminate** in finite time.

**Foundations of Semantic Web Technologies**. Chapter 3.

# Entailment via Inference Rules

## Syntactic Reasoning

- From the computation point of view, we need means to decide **entailment syntactically**.
- Syntactic methods operate
  - only on the form of a statement, that is on its **concrete grammatical structure** (*i.e.*, triples),
  - without recurring to interpretations.
- Syntactic methods should justify that their so-called **operational semantics** are expected with respect to model-theoretic semantics.

# Inference rules (i)

- Inference rules (also known as deduction rules or derivation rules) is an option to **describe syntactic solutions**.

- The general form of an inference rule is:

$$\frac{P_1, \ldots, P_n}{P}$$

  - the $P_i$ are **premises** (body)
  - and $P$ is the **conclusion** (head).

- An inference rule may have,
  - any number of premises (typically one or two),
  - but only one conclusion.

# Inference rules (ii)

- Recall that syllogisms (*i.e.*, inference) can be traced back to Aristotle
- Example:

$$\frac{\text{All human are mortal} \quad \text{Socrates is a human}}{\text{Therefore, Socrates is mortal}}$$

# Inference rules (iii)

- The whole set of inference rules given for a logic is called **deduction calculus**.

- $\vdash$ is the **inference relation**, while $\models$ was the entailment relation using model theoretic semantics.
  - We write $\Gamma \vdash P$ if we can deduce $P$ from the premises $\Gamma$.

# Inference rules (iii)

- The whole set of inference rules given for a logic is called **deduction calculus**.

- $\vdash$ is the **inference relation**, while $\models$ was the entailment relation using model theoretic semantics.
  - We write $\Gamma \vdash P$ if we can deduce $P$ from the premises $\Gamma$.

- **In our setting:**
  - the **premises** $\Gamma$ are a **set of triples** (*i.e.*, a (sub)graph $\mathcal{G}$),
  - the **conclusion** $P$ is a **new triple** $t$
  - After applying the rules to $\mathcal{G}$ we will get an **expanded graph** $\mathcal{G}'$

# RDFS Inference Rules

RDFS supports several rules. Organized into three groups:

1. **Type propagation:**
   - "London is a City, all Cities are populated places, so..."
2. **Property propagation:**
   - "London is the capital of England, anything that is capital of a country is also located in that country, so..."
3. **Domain and range propagation:**
   - "Everything that has a capital is a country, so England is a..."
   - "Everything that is a capital is a city, so London is a..."

RDFS Entailment Rules: `https://www.w3.org/TR/rdf-mt/#RDFSRules`

# Type propagation

– **Members of superclasses:**

$$\frac{\texttt{A rdfs:subClassOf B .} \qquad \texttt{x rdf:type A .}}{\texttt{x rdf:type B .}}\ \text{rdfs9}$$

(*) rdfs9, rdfs10, rdfs11 are the names of the inference rules in the W3C standard.

A, B are classes; x is an instance.

# Type propagation

– **Members of superclasses:**

$$\frac{\texttt{A rdfs:subClassOf B .} \qquad \texttt{x rdf:type A .}}{\texttt{x rdf:type B .}}\ \text{rdfs9}$$

– **Reflexivity of sub-class relation:**

$$\frac{\texttt{A rdf:type rdfs:Class .}}{\texttt{A rdfs:subClassOf A .}}\ \text{rdfs10}$$

(*) rdfs9, rdfs10, rdfs11 are the names of the inference rules in the W3C standard.

A, B are classes; x is an instance.

# Type propagation

- **Members of superclasses:**

$$\frac{\texttt{A rdfs:subClassOf B .} \qquad \texttt{x rdf:type A .}}{\texttt{x rdf:type B .}}\ \text{rdfs9}$$

- **Reflexivity of sub-class relation:**

$$\frac{\texttt{A rdf:type rdfs:Class .}}{\texttt{A rdfs:subClassOf A .}}\ \text{rdfs10}$$

- **Transitivity of sub-class relation:**

$$\frac{\texttt{A rdfs:subClassOf B .} \qquad \texttt{B rdfs:subClassOf C .}}{\texttt{A rdfs:subClassOf C .}}\ \text{rdfs11}$$

(*) rdfs9, rdfs10, rdfs11 are the names of the inference rules in the W3C standard.

A, B are classes; x is an instance.

# Type propagation: Examples

– **Members of superclasses:**

$$\frac{\texttt{:City rdfs:subClassOf :PopulatedPlace .} \qquad \texttt{:london rdf:type :City .}}{\texttt{:london rdf:type :PopulatedPlace .}}\ \text{rdfs9}$$

– **Reflexivity of sub-class relation:**

$$\frac{\texttt{:City rdf:type rdfs:Class .}}{\texttt{:City rdfs:subClassOf :City .}}\ \text{rdfs10}$$

– **Transitivity of sub-class relation:**

$$\frac{\texttt{:City rdfs:subClassOf :PopulatedPlace .} \qquad \texttt{:PopulatedPlace rdfs:subClassOf :Place .}}{\texttt{:City rdfs:subClassOf :Place .}}\ \text{rdfs11}$$

# Property Propagation

– **Transitivity:**

$$\frac{\texttt{P rdfs:subPropertyOf Q .} \quad \texttt{Q rdfs:subPropertyOf R .}}{\texttt{P rdfs:subPropertyOf R .}}\ \text{rdfs5}$$

(*) P, Q are properties; u, v are instances.

# Property Propagation

- **Transitivity:**

$$\frac{\texttt{P rdfs:subPropertyOf Q .} \qquad \texttt{Q rdfs:subPropertyOf R .}}{\texttt{P rdfs:subPropertyOf R .}}\ \text{rdfs5}$$

- **Reflexivity:**

$$\frac{\texttt{P rdf:type rdf:Property .}}{\texttt{P rdfs:subPropertyOf P .}}\ \text{rdfs6}$$

(*) P, Q are properties; u, v are instances.

# Property Propagation

– **Transitivity:**

$$\frac{\texttt{P rdfs:subPropertyOf Q .} \qquad \texttt{Q rdfs:subPropertyOf R .}}{\texttt{P rdfs:subPropertyOf R .}}\ \text{rdfs5}$$

– **Reflexivity:**

$$\frac{\texttt{P rdf:type rdf:Property .}}{\texttt{P rdfs:subPropertyOf P .}}\ \text{rdfs6}$$

– **Property transfer:**

$$\frac{\texttt{P rdfs:subPropertyOf Q .} \qquad \texttt{u P v .}}{\texttt{u Q v .}}\ \text{rdfs7}$$

(*) P, Q are properties; u, v are instances.

# Property Propagation: Examples

– **Transitivity:**

$$\frac{\texttt{:has\_writer rdfs:subPropertyOf :has\_author .} \qquad \texttt{:has\_author rdfs:subPropertyOf :has\_creator .}}{\texttt{:has\_writer rdfs:subPropertyOf :has\_creator .}}\ \text{rdfs5}$$

– **Reflexivity:**

$$\frac{\texttt{:has\_writer rdf:type rdf:Property .}}{\texttt{:has\_writer rdfs:subPropertyOf :has\_writer .}}\ \text{rdfs6}$$

– **Property transfer:**

$$\frac{\texttt{:has\_author rdfs:subPropertyOf :has\_creator .} \qquad \texttt{:Hamlet :has\_author :Shakespeare .}}{\texttt{:Hamlet :has\_creator :Shakespeare .}}\ \text{rdfs7}$$

# Domain and range propagation

Typing triggered by the use of properties.

– **Domain propagation:**

$$\frac{\texttt{P rdfs:domain A .} \qquad \texttt{x P y .}}{\texttt{x rdf:type A .}}\ \text{rdfs2}$$

(*) P, Q are properties; x, y are instances.

# Domain and range propagation

Typing triggered by the use of properties.

- **Domain propagation:**

$$\frac{\texttt{P rdfs:domain A .} \qquad \texttt{x P y .}}{\texttt{x rdf:type A .}} \text{ rdfs2}$$

- **Range propagation:**

$$\frac{\texttt{P rdfs:range B .} \qquad \texttt{x P y .}}{\texttt{y rdf:type B .}} \text{ rdfs3}$$

(*) P, Q are properties; x, y are instances.

# Domain and Range Propagation: Examples

- **Domain propagation:**

$$\frac{\texttt{:capitalOf rdfs:domain :City .} \qquad \texttt{:london :capitalOf :england .}}{\texttt{:london rdf:type :City .}}\ \text{rdfs2}$$

- **Range propagation:**

$$\frac{\texttt{:capitalOf rdfs:range :Country .} \qquad \texttt{:london :capitalOf :england .}}{\texttt{:england rdf:type :Country .}}\ \text{rdfs3}$$

# Properties of RDFS Semantics

# Entailment and Inference

- Both have the **monotonic** property.
  - If a graph $\mathcal{G} \models t$ (or $\mathcal{G} \vdash t$),
  - then adding more triples (*e.g.*, $t_1$) does not alter the entailment $\mathcal{G} \cup \{t_1\} \models t$ (or derivation $\mathcal{G} \cup \{t_1\} \vdash t$)

# Entailment and Inference

- Both have the **monotonic** property.
  - If a graph $\mathcal{G} \models t$ (or $\mathcal{G} \vdash t$),
  - then adding more triples (*e.g.*, $t_1$) does not alter the entailment $\mathcal{G} \cup \{t_1\} \models t$ (or derivation $\mathcal{G} \cup \{t_1\} \vdash t$)

- The set of RDFS rules we have seen are **sound**.
  - If $\mathcal{G} \vdash t$ then $\mathcal{G} \models t$

# Entailment and Inference

- Both have the **monotonic** property.
  - If a graph $\mathcal{G} \models t$ (or $\mathcal{G} \vdash t$),
  - then adding more triples (*e.g.*, $t_1$) does not alter the entailment $\mathcal{G} \cup \{t_1\} \models t$ (or derivation $\mathcal{G} \cup \{t_1\} \vdash t$)
- The set of RDFS rules we have seen are **sound**.
  - If $\mathcal{G} \vdash t$ then $\mathcal{G} \models t$
- But **not complete**.
  - Not always applies that If $\mathcal{G} \models t$ then $\mathcal{G} \vdash t$
  - Due to how RDFS deals with datatypes.

**Foundations of Semantic Web Technologies**. Chapter 3.

# (Non) Validation in RDFS (i)

- RDFS was conceived of as a schema language for RDF
- However, the statements in an RDFS graph **never trigger inconsistencies**.
- Reasoning will not lead to a “contradiction”, “error”, “non-valid document”
- Inference rules in RDFS add more triples, but **do not detect errors**.

# (Non) Validation in RDFS (ii)

- RDFS has **no notion of negation**
  - For instance, the two triples

    ```
    city:ernesto rdf:type ex:Smoker .
    city:ernesto rdf:type ex:NonSmoker .
    ```

    are not inconsistent.

# (Non) Validation in RDFS (ii)

- RDFS has **no notion of negation**
  - For instance, the two triples

    ```
    city:ernesto rdf:type ex:Smoker .
    city:ernesto rdf:type ex:NonSmoker .
    ```

    are not inconsistent.
- There is also **not clear notion of disjointness** among RDF resources: Object Properties, Datatype Properties, Classes, Built-in properties, Individuals, Datatypes and Literals.

# (Non) Validation in RDFS (ii)

- RDFS has **no notion of negation**
  - For instance, the two triples

    ```
    city:ernesto rdf:type ex:Smoker .
    city:ernesto rdf:type ex:NonSmoker .
    ```

    are not inconsistent.

- There is also **not clear notion of disjointness** among RDF resources: Object Properties, Datatype Properties, Classes, Built-in properties, Individuals, Datatypes and Literals.

- **OWL** (and OWL 2) includes additional vocabulary and also **consistency-checks**.

# OWL 2 Reasoning and Profiles

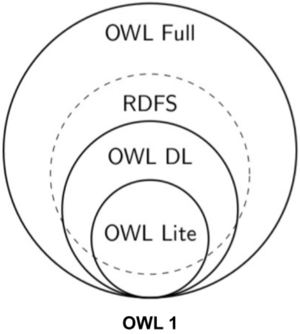
# Recap: OWL (The Web Ontology Language)

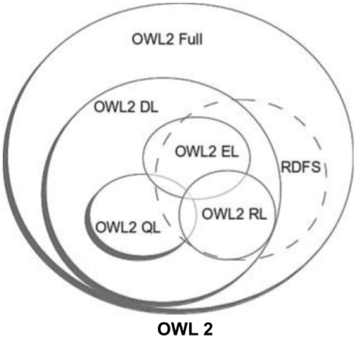
- A **W3C recommendation**:
  - OWL 1 (2004): `http://www.w3.org/TR/owl-ref/`
  - OWL 2 (2009): `https://www.w3.org/TR/owl2-overview/`

# Recap: OWL (The Web Ontology Language)

- A **W3C recommendation**:
  - OWL 1 (2004): `http://www.w3.org/TR/owl-ref/`
  - OWL 2 (2009): `https://www.w3.org/TR/owl2-overview/`
- OWL semantics based on **Description Logics (DL)**.
  - Family of knowledge representation languages
  - Decidable subset of First Order logic (FOL)
  - Original called: **Terminological language** or **concept language**

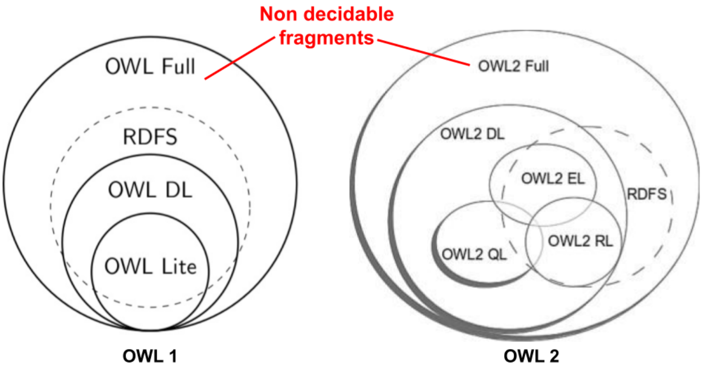
# Recap: OWL 1, OWL 2 (profiles) and RDFS




Olivier Cure and Guillaume Blin. RDF Database Systems (Chapter 3). 2015. Elsevier.

# Recap: OWL 1, OWL 2 (profiles) and RDFS




† **Reasoning in OWL 2** will partially get the same consequences as in the RDFS inference rules and many more.

# Automated Reasoning in OWL

- Formal semantics allows the automatic deduction of **new facts** (*i.e.*, *not explicit but derived from others*).
- Also allows us to perform checks that aim to detect the **correctness** of the designed model (*e.g.*, is `:dolphin` a `:Fish`?). New respect to RDFS.

# Automated Reasoning in OWL

- Formal semantics allows the automatic deduction of **new facts** (*i.e.*, *not explicit but derived from others*).
- Also allows us to perform checks that aim to detect the **correctness** of the designed model (*e.g.*, is `:dolphin` a `:Fish`?). New respect to RDFS.
- In the form of **logical errors**:
  - `:Mammal` and `:Fish` are disjoint classes.
  - `:dolphin` cannot be an individual (or a subclass) of both `:Mammal` and `:Fish`.

# Automated Reasoning in OWL

- Formal semantics allows the automatic deduction of **new facts** (*i.e.*, *not explicit but derived from others*).
- Also allows us to perform checks that aim to detect the **correctness** of the designed model (*e.g.*, is `:dolphin` a `:Fish`?). New respect to RDFS.
- In the form of **logical errors**:
  - `:Mammal` and `:Fish` are disjoint classes.
  - `:dolphin` cannot be an individual (or a subclass) of both `:Mammal` and `:Fish`.
- Extremely valuable for designing **correct** ontologies/KGs, specially when working **collaboratively** and **integrating** various sources.

# OWL 2 Reasoning

# Recap: OWL 2 Axioms into Boxes

- Traditionally OWL 2 axioms are put in boxes.
- The **TBox** (terminological knowledge)
  - Typically independent of any actual instance data.
  - Property axioms are also referred to as **RBox**
- The **ABox** (assertional knowledge)
  - Contains facts about concrete instance.

# OWL 2 TBox Reasoning

**(Standard) Reasoning tasks that use only the TBox** $\mathcal{T}^{\dagger}$

- Concept **unsatisfiability**: Given $C$, does $\mathcal{T} \models C \sqsubseteq \bot$? (*i.e.*, $C^{\mathcal{I}} = \emptyset$)

† **(Model-Theoretic Semantics)** The answer to 'does $\mathcal{T} \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \alpha$ too.

# OWL 2 TBox Reasoning

**(Standard) Reasoning tasks that use only the TBox $\mathcal{T}$†**

- Concept **unsatisfiability**: Given $C$, does $\mathcal{T} \models C \sqsubseteq \bot$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} = \emptyset$)
- Concept **subsumption**: Given $C$ and $D$, does $\mathcal{T} \models C \sqsubseteq D$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} \subseteq \mathcal{D}^{\mathcal{I}}$)

† **(Model-Theoretic Semantics)** The answer to 'does $\mathcal{T} \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \alpha$ too.

# OWL 2 TBox Reasoning

**(Standard) Reasoning tasks that use only the TBox $\mathcal{T}^{\dagger}$**

- Concept **unsatisfiability**: Given $C$, does $\mathcal{T} \models C \sqsubseteq \bot$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} = \emptyset$)
- Concept **subsumption**: Given $C$ and $D$, does $\mathcal{T} \models C \sqsubseteq D$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} \subseteq \mathcal{D}^{\mathcal{I}}$)
- Concept **equivalence**: Given $C$ and $D$, does $\mathcal{T} \models C \equiv D$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} = \mathcal{D}^{\mathcal{I}}$)

† **(Model-Theoretic Semantics)** The answer to 'does $\mathcal{T} \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \alpha$ too.

# OWL 2 TBox Reasoning

**(Standard) Reasoning tasks that use only the TBox $\mathcal{T}^{\dagger}$**

- Concept **unsatisfiability**: Given $C$, does $\mathcal{T} \models C \sqsubseteq \bot$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} = \emptyset$)
- Concept **subsumption**: Given $C$ and $D$, does $\mathcal{T} \models C \sqsubseteq D$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} \subseteq \mathcal{D}^{\mathcal{I}}$)
- Concept **equivalence**: Given $C$ and $D$, does $\mathcal{T} \models C \equiv D$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} = \mathcal{D}^{\mathcal{I}}$)
- Concept **disjointness**: Given $C$ and $D$, does $\mathcal{T} \models C \sqcap D \sqsubseteq \bot$? (*i.e.*, $\mathcal{C}^{\mathcal{I}} \cap \mathcal{D}^{\mathcal{I}} \subseteq \emptyset$)

† **(Model-Theoretic Semantics)** The answer to 'does $\mathcal{T} \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$, $\mathcal{I} \models \alpha$ too.

# OWL 2 ABox Reasoning

**(Standard) Reasoning tasks that involve both the TBox $\mathcal{T}$ and Abox $\mathcal{A}$**

– **Consistency**:
Is there a model for $(\mathcal{T}, \mathcal{A})$? i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?

† **(Model-Theoretic Semantics)** The answer to 'does $(\mathcal{T}, \mathcal{A}) \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, $\mathcal{I} \models \alpha$ too.

# OWL 2 ABox Reasoning

**(Standard) Reasoning tasks that involve both the TBox $\mathcal{T}$ and Abox $\mathcal{A}$**

- **Consistency**:
  Is there a model for $(\mathcal{T}, \mathcal{A})$? i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?
- Concept **membership**: Given $C$ and $a$, does $(\mathcal{T}, \mathcal{A}) \models C(a)$?† (*i.e.*, $a \in C$).

† **(Model-Theoretic Semantics)** The answer to 'does $(\mathcal{T}, \mathcal{A}) \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, $\mathcal{I} \models \alpha$ too.

# OWL 2 ABox Reasoning

**(Standard) Reasoning tasks that involve both the TBox $\mathcal{T}$ and Abox $\mathcal{A}$**

- **Consistency**:
  Is there a model for $(\mathcal{T}, \mathcal{A})$? i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?
- Concept **membership**: Given $C$ and $a$, does $(\mathcal{T}, \mathcal{A}) \models C(a)$?† (*i.e.*, $a \in C$).
- **Role assertion**: Given $R$, $a$ and $b$, does $(\mathcal{T}, \mathcal{A}) \models R(a, b)$?† (*i.e.*, $\langle a, b\rangle \in R$)

† **(Model-Theoretic Semantics)** The answer to 'does $(\mathcal{T}, \mathcal{A}) \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, $\mathcal{I} \models \alpha$ too.

# OWL 2 ABox Reasoning

**(Standard) Reasoning tasks that involve both the TBox $\mathcal{T}$ and Abox $\mathcal{A}$**

- **Consistency**:
  Is there a model for $(\mathcal{T}, \mathcal{A})$? i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?
- Concept **membership**: Given $C$ and $a$, does $(\mathcal{T}, \mathcal{A}) \models C(a)$?† (*i.e.*, $a \in C$).
- **Role assertion**: Given $R$, $a$ and $b$, does $(\mathcal{T}, \mathcal{A}) \models R(a, b)$?† (*i.e.*, $\langle a, b \rangle \in R$)
- **Retrieval**: Given $C$, find all $a$ such that $(\mathcal{T}, \mathcal{A}) \models C(a)$.

† **(Model-Theoretic Semantics)** The answer to 'does $(\mathcal{T}, \mathcal{A}) \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, $\mathcal{I} \models \alpha$ too.

# OWL 2 ABox Reasoning

**(Standard) Reasoning tasks that involve both the TBox $\mathcal{T}$ and Abox $\mathcal{A}$**

- **Consistency**:
  Is there a model for $(\mathcal{T}, \mathcal{A})$? i.e., is there an interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$?
- Concept **membership**: Given $C$ and $a$, does $(\mathcal{T}, \mathcal{A}) \models C(a)$?† (*i.e.*, $a \in C$).
- **Role assertion**: Given $R$, $a$ and $b$, does $(\mathcal{T}, \mathcal{A}) \models R(a, b)$?† (*i.e.*, $\langle a, b\rangle \in R$)
- **Retrieval**: Given $C$, find all $a$ such that $(\mathcal{T}, \mathcal{A}) \models C(a)$.
- Conjunctive Query Answering (**SPARQL**).

† **(Model-Theoretic Semantics)** The answer to 'does $(\mathcal{T}, \mathcal{A}) \models \alpha$?' will be positive if for each interpretation $\mathcal{I}$ such that $\mathcal{I} \models (\mathcal{T}, \mathcal{A})$, $\mathcal{I} \models \alpha$ too.

# OWL 2 Reasoning Algorithms

- Reasoning in OWL 2 is typically based on **(Hyper)Tableau Reasoning Algorithms** (tableau = truth tree)
- Reasoning tasks **reduced to (un)satisfiability**.
- Algorithm tries to construct an **abstraction** of a model.

Chapter 5: Foundations of Semantic Web Technologies. CRC Press 2009
Seminars by Prof. Ian Horrocks: `http://www.cs.ox.ac.uk/people/ian.horrocks/Seminars/seminars.html`

## OWL 2 Reasoning Algorithms

- Reasoning in OWL 2 is typically based on **(Hyper)Tableau Reasoning Algorithms** (tableau = truth tree)
- Reasoning tasks **reduced to (un)satisfiability**.
- Algorithm tries to construct an **abstraction** of a model.
- State-of-the-art algorithms:
  - *e.g.*, **HermiT** (default option in Protégé).
  - Implement a number of (search) **optimisations**.
  - **Effective** with many realistic ontologies

Chapter 5: Foundations of Semantic Web Technologies. CRC Press 2009
Seminars by Prof. Ian Horrocks: `http://www.cs.ox.ac.uk/people/ian.horrocks/Seminars/seminars.html`

# Tractability Problems with OWL 2 Reasoning

- Problems with **very large** and/or **cyclical ontologies**.
  - Ontologies may define hundred of thousands of terms (*e.g.*, SNOMED CT)
  - Large number of tests for classification (each test can lead to the construction of very large models).

Computational properties: `https://www.w3.org/TR/owl2-profiles/#Computational_Properties`
Seminars by Prof. Ian Horrocks: `http://www.cs.ox.ac.uk/people/ian.horrocks/Seminars/seminars.html`

## Tractability Problems with OWL 2 Reasoning

- Problems with **very large** and/or **cyclical ontologies**.
  - Ontologies may define hundred of thousands of terms (*e.g.*, SNOMED CT)
  - Large number of tests for classification (each test can lead to the construction of very large models).
- Problems with **medium/large** data sets (ABoxes)
  - OWL 2 Reasoners typically optimized for TBox reasoning tasks.
  - Data also brings additional complexity.

Computational properties: `https://www.w3.org/TR/owl2-profiles/#Computational_Properties`
Seminars by Prof. Ian Horrocks: `http://www.cs.ox.ac.uk/people/ian.horrocks/Seminars/seminars.html`

# OWL 2 profiles

# OWL 2 profiles

- OWL 2 has various **profiles** that correspond to different DLs.
- These profiles have very interesting **computational properties**.

OWL 2 Validator: `http://visualdataweb.de/validator/`

# OWL 2 profiles

- OWL 2 has various **profiles** that correspond to different DLs.
- These profiles have very interesting **computational properties**.
  - **OWL 2 QL**:
    - Specifically designed for efficient database integration.

OWL 2 Validator: `http://visualdataweb.de/validator/`

# OWL 2 profiles

- OWL 2 has various **profiles** that correspond to different DLs.
- These profiles have very interesting **computational properties**.
  - **OWL 2 QL**:
    - Specifically designed for efficient database integration.
  - **OWL 2 EL**:
    - A lightweight language with polynomial time reasoning.

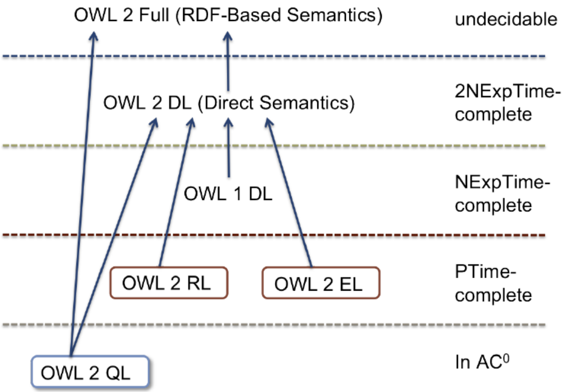
OWL 2 Validator: `http://visualdataweb.de/validator/`

# OWL 2 profiles

- OWL 2 has various **profiles** that correspond to different DLs.
- These profiles have very interesting **computational properties**.
  - **OWL 2 QL**:
    - Specifically designed for efficient database integration.
  - **OWL 2 EL**:
    - A lightweight language with polynomial time reasoning.
  - **OWL 2 RL**:
    - Designed for compatibility with rule-based inference tools.
    - Efficient reasoning with large datasets.

OWL 2 Validator: `http://visualdataweb.de/validator/`

# Data Complexity OWL 2 Profiles




https://www.w3.org/TR/owl2-profiles/

# OWL EL profile (i)

**Based on the DL $\mathcal{EL}^{++}$. Concept descriptions (simplified)**

$$
\begin{array}{rlcl}
C, D \rightarrow & A & \mid & \text{(atomic concept)} \\
& \top & \mid & \text{(universal concept)} \\
& \bot & \mid & \text{(bottom concept)} \\
& \{a\} & \mid & \text{(\textit{singular} enumeration)} \\
& C \sqcap D & \mid & \text{(intersection)} \\
& \exists R.C & \mid & \text{(existential restriction)}
\end{array}
$$

**Axioms**

- $C \sqsubseteq D$ and $C \equiv D$ for concept descriptions $D$ and $C$.
- $P \sqsubseteq Q$ and $P \equiv Q$ for roles $P, Q$. Also Domain and Range.
- $C(a)$ and $R(a, b)$ for concept $C$, role $R$ and individuals $a, b$.

# OWL EL Profile (ii)

- ✓ Standard reasoning tasks in $\mathrm{P}$ time
- ✓ Very good for large ontologies.
- ✓ Used in many biomedical ontologies (*e.g.*, SNOMED CT).

Not supported features, simplified:

- ✗ negation (but $C \sqcap D \sqsubseteq \bot$ possible)
- ✗ disjunction
- ✗ universal quantification and cardinalities
- ✗ inverse roles and some role characteristics
- ✗ reduced list of datatypes

# OWL EL Profile (iii)

- Reasoning can be performed via saturation[†] (*i.e.*, inference rules).
- For example:

$$\frac{A \sqsubseteq B \qquad B \sqsubseteq C}{A \sqsubseteq C}$$

$$\frac{A \sqsubseteq \exists R.B \qquad \exists S.B \sqsubseteq C \qquad S \sqsubseteq R}{A \sqsubseteq C}$$

† Using a saturation-based approach over an OWL 2 ontology is not possible.

**ELK reasoner** (also available as Protégé plugin): `https://github.com/liveontologies/elk-reasoner/wiki`

# OWL QL Profile (i)

**Based on DL-Lite$_R$. Concept descriptions (simplified)**

$$
\begin{array}{rlcl}
C, C' \rightarrow & A & \mid & \text{(atomic concept)} \\
& \exists R.\top & \mid & \text{(existential restriction with } \top \text{ only)} \\
D, D' \rightarrow & A & \mid & \text{(atomic concept)} \\
& \exists R.D & \mid & \text{(existential restriction)} \\
& \neg D & \mid & \text{(negation)} \\
& D \sqcap D' & \mid & \text{(intersection)}
\end{array}
$$

**Axioms**

- $C \sqsubseteq D$ for concept descriptions $D$ and $C$ (and $C \equiv C'$).
- $P \sqsubseteq Q$ and $P \equiv Q$ for roles $P, Q$. Also Domain and Range.
- $C(a)$ and $R(a, b)$ for concept $C$, role $R$ and individuals $a, b$.

# OWL QL Profile (ii)

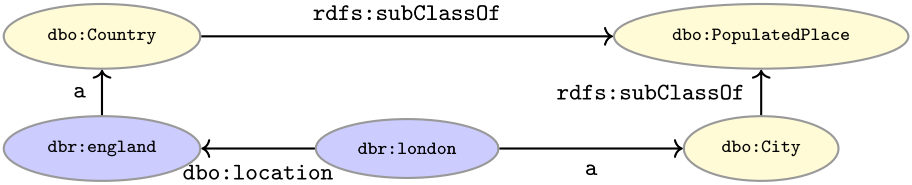
- ✓ Required language so that queries can be rewritten using the TBox.
- ✓ Used in Ontology Based Data Access (OBDA) where SPARQL queries are translated to SQL

Not supported, simplified:

- ✗ disjunction
- ✗ universal quantification, cardinalities, and functional roles
- ✗ $=$ ($\mathtt{SameIndividual}$)
- ✗ enumerations (closed classes)
- ✗ subproperties of chains, transitivity
- ✗ reduced list of datatypes

# OWL QL Profile (iii)

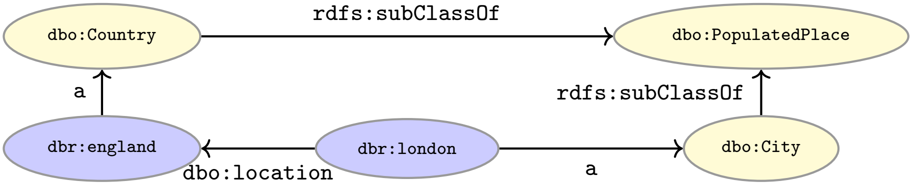
– Reasoning is performed via backward chaining (*e.g.*, rewriting of a given query $Q$ into $Q'$ via the ontology axioms, instead of expanding the graph). For example:



# OWL QL Profile (iii)

– Reasoning is performed via backward chaining (*e.g.*, rewriting of a given query $Q$ into $Q'$ via the ontology axioms, instead of expanding the graph). For example:

$Q$: `SELECT DISTINCT ?place WHERE {?place rdf:type dbo:PopulatedPlace . }`

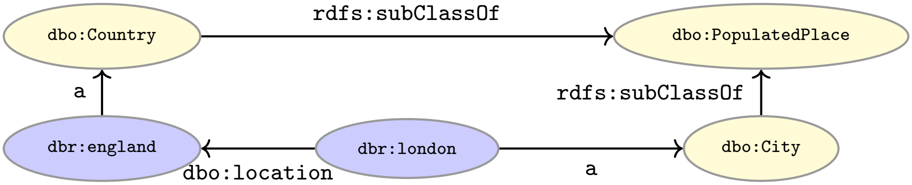


# OWL QL Profile (iii)

– Reasoning is performed via backward chaining (*e.g.*, rewriting of a given query $Q$ into $Q'$ via the ontology axioms, instead of expanding the graph). For example:

$Q$:
```
SELECT DISTINCT ?place WHERE {?place rdf:type dbo:PopulatedPlace .  }
```

$Q'$:
```
SELECT DISTINCT ?place WHERE {
    {?place rdf:type dbo:PopulatedPlace .}
    UNION {?place rdf:type dbo:Country .}
    UNION {?place rdf:type dbo:City .}  }
```



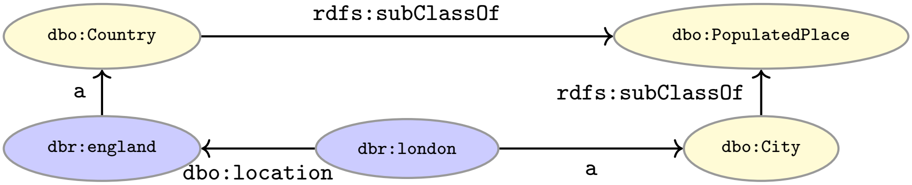
# OWL QL Profile (iii)

– Reasoning is performed via backward chaining (*e.g.*, rewriting of a given query $Q$ into $Q'$ via the ontology axioms, instead of expanding the graph). For example:

$Q$: `SELECT DISTINCT ?place WHERE {?place rdf:type dbo:PopulatedPlace . }`

$Q'$:
```
SELECT DISTINCT ?place WHERE {
    {?place rdf:type dbo:PopulatedPlace .}
    UNION {?place rdf:type dbo:Country .}
    UNION {?place rdf:type dbo:City .}  }
```

$Q'$ Result= {dbr:england, dbr:london}



# OWL 2 RL Profile (i)

**Based on Description Logic Programs (DLP). Concept descriptions:**

$$\begin{array}{rlcl} C, C' \rightarrow & A & \mid & \text{(atomic concept)} \\ & C \sqcap C' & \mid & \text{(intersection)} \\ & C \sqcup C' & \mid & \text{(union)} \\ & \exists R.C & \mid & \text{(existential restriction)} \\ D, D' \rightarrow & A & \mid & \text{(atomic concept)} \\ & D \sqcap D' & \mid & \text{(intersection)} \\ & \forall R.D & \mid & \text{(universal restriction)} \end{array}$$

**Axioms**

– $C \sqsubseteq D$, $C \equiv C'$, $\top \sqsubseteq \forall R.D$, $\top \sqsubseteq \forall R^{-}.D$ $R \sqsubseteq P$, $R \equiv P^{-}$ and $R \equiv P$ for roles $R$, $P$ and concept descriptions $C$, $C'$ and $D$. Also Domain and Range.

– $C(a)$ and $R(a, b)$ for concept $C$, role $R$ and individuals $a$, $b$.

# OWL 2 RL Profile (ii)

- ✗ Puts syntactic constraints in the way in which constructs are used (i.e., syntactic subset of OWL 2).
- ✗ Imposes a reduced list of allowed datatypes
- ✓ OWL 2 RL axioms can be directly translated into datalog rules
- ✓ Enables desirable computational properties using rule-based reasoning engines.

# OWL 2 RL Profile (iii)

– Reasoning via full materialisation of the graph, similarly to RDFS inference rules. *e.g.*,:

$$\frac{\texttt{p1 owl:inverseOf p2 .} \qquad \texttt{?x ?p1 ?y .}}{\texttt{?y ?p2 ?x .}}$$

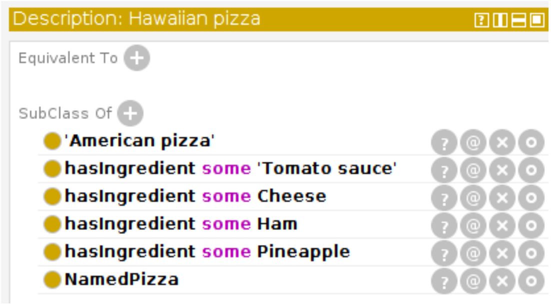
– See W3C specification for further inference rules in OWL 2 RL.

W3C: `https://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules`
GraphDB: `https://graphdb.ontotext.com/documentation/standard/reasoning.html`

# Practical Examples: OWL Reasoning

# Necessary conditions and primitive classes

Hawaiian pizza **implies** having pineapple as ingredient (among others); but not the other way round.

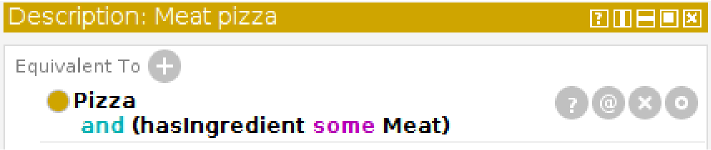


# Sufficient conditions and defined classes

Meat pizza **implies** having meat as ingredient (and being pizza).

A pizza with meat as ingredient **implies** being a meat pizza.

Hawaiian pizza has ham as ingredient and thus is a meat pizza.

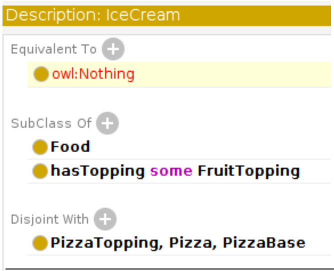


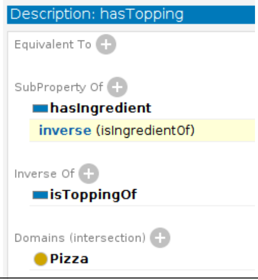
# Detecting modelling errors

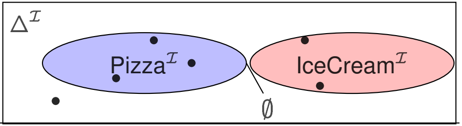
Ice cream **implies** having fruit as topping

Ice cream is **disjoint with** Pizza

The **domain** of has topping is pizza, that is, having any topping **implies** being a pizza (domain is a type of sufficient condition).



# Detecting modelling errors: part-of VS subclass-of

:City rdfs:subClassOf :Country . ?

– **Members of superclasses (rdfs9 rule):**

$$\frac{\texttt{:City rdfs:subClassOf :Country .} \quad \texttt{:london rdf:type :City .}}{}\ \text{rdfs9}$$

# Detecting modelling errors: part-of VS subclass-of

:City rdfs:subClassOf :Country . ? Incorrect

– **Members of superclasses (rdfs9 rule):**

$$\frac{\texttt{:City rdfs:subClassOf :Country .} \quad \texttt{:london rdf:type :City .}}{\texttt{:london rdf:type :Country .}}\ \text{rdfs9}$$

# Detecting modelling errors: part-of VS subclass-of

:City rdfs:subClassOf :Country . ? Incorrect

– **Members of superclasses (rdfs9 rule):**

$$\frac{\texttt{:City rdfs:subClassOf :Country .} \qquad \texttt{:london rdf:type :City .}}{\texttt{:london rdf:type :Country .}}\ \text{rdfs9}$$

:City rdfs:subClassOf :locatedIn some :Country

# Detecting modelling errors: property hierarchy

:isLocatedInCity rdfs:subPropertyOf :isLocatedInCountry . ?

– **Property transfer (rdfs7 rule):**

$$\frac{\texttt{:isLocInCity rdfs:subPropertyOf :isLocInCountry .} \qquad \texttt{:big\_ben :isLocInCity :london .}}{}$$

# Detecting modelling errors: property hierarchy

:isLocatedInCity rdfs:subPropertyOf :isLocatedInCountry . ? Incorrect

– **Property transfer (rdfs7 rule):**

$$\frac{\texttt{:isLocInCity rdfs:subPropertyOf :isLocInCountry .} \qquad \texttt{:big\_ben :isLocInCity :london .}}{\texttt{:big\_ben :isLocInCountry :london .}}$$

# Detecting modelling errors: property hierarchy

:isLocatedInCity rdfs:subPropertyOf :isLocatedInCountry . ? Incorrect

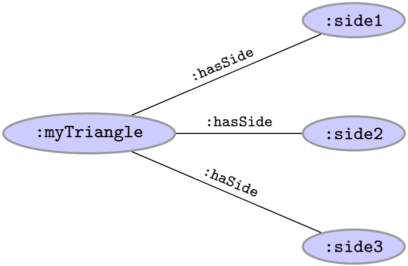
– **Property transfer (rdfs7 rule):**

$$\frac{\texttt{:isLocInCity rdfs:subPropertyOf :isLocInCountry .} \quad \texttt{:big\_ben :isLocInCity :london .}}{\texttt{:big\_ben :isLocInCountry :london .}}$$

:isLocatedInCity rdfs:subPropertyOf :isLocatedIn .
:isLocatedInCountry rdfs:subPropertyOf :isLocatedIn .

# OWL 2 and Open World Assumption

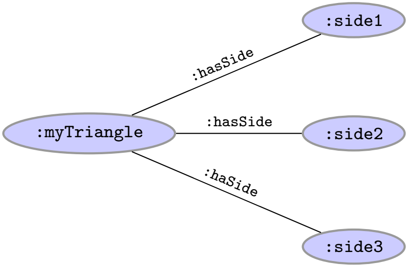
– `:Triangle EquivalentTo :hasSide exactly 3 :Side`




– is `:myTriangle` a `:Triangle`?

# OWL 2 and Open World Assumption

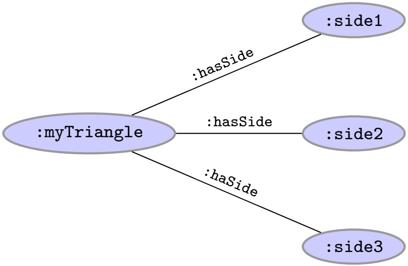
- `:Triangle EquivalentTo :hasSide exactly 3 :Side`




- is `:myTriangle` a `:Triangle`?

# OWL 2 and Open World Assumption

- `:Triangle EquivalentTo :hasSide exactly 3 :Side`




- is `:myTriangle` a `:Triangle`? I don't know because of OWA and NUNA.
- Solution: reasoning in OWL can be complemented with SPARQL queries (in this case with aggregates) $\rightarrow$ SPARQL 1.1

# Laboratory: RDFS Semantics and OWL 2 RL

## Tasks

- Manually checking inferences using RDFS and OWL 2 RL semantics.
- Extracting inferences and expanding the graph/model programmatically and checking via SPARQL.
- **Python**: We are using the OWL-RL library

```
owlrl.DeductiveClosure(owlrl.RDFS_Semantics).expand(g)
owlrl.DeductiveClosure(owlrl.OWLRL_Semantics).expand(g)
```

- **Java**: Jena API

```
Reasoner reasoner = ReasonerRegistry.getRDFSReasoner();
Reasoner reasoner = ReasonerRegistry.getOWLMiniReasoner();
InfModel inf_model = ModelFactory.createInfModel(reasoner,
model);
```

# RDFS inference rules (cheatsheet)

| Rule | If | Then add |
|---|---|---|
| rdf1 | (x p y) | (p rdf:type rdf:Property) |
| rdfs2 | (p rdfs:domain c) (x p y) | (x rdf:type c) |
| rdfs3 | (p rdfs:range c) (x p y) | (y rdf:type c) |
| rdfs4a | (x p y) | (x rdf:type rdfs:Resource) |
| rdfs4b | (x p y) | (y rdf:type rdfs:Resource) |
| rdfs5 | (p rdfs:subPropertyOf q) (q rdfs:subPropertyOf r) | (p rdfs:subPropertyOf r) |
| rdfs6 | (p rdf:type rdf:Property) | (P rdfs:subPropertyOf p) |
| rdfs7 | (p rdfs:subPropertyOf q) (x p y) | (x q y) |
| rdfs8 | (c rdf:type rdfs:Class) | (c rdfs:subClassOf rdfs:Resource) |
| rdfs9 | (c rdfs:subClassOf d) (x rdf:type c) | (x rdf:type d) |
| rdfs10 | (c rdf:type rdfs:Class) | (c rdfs:subClassOf c) |
| rdfs11 | (c rdfs:subClassOf d) (d rdfs:subClassOf e) | (c rdfs:subClassOf e) |
| rdfs12 | (p rdf:type rdfs:ContainerMembershipProperty) | (p rdfs:subPropertyOf rdfs:Member) |
| rdfs13 | (x rdf:type rdfs:Datatype) | (x rdfs:subClassOf rdfs:Literal) |

| | | |
|---|---|---|
| | (p owl:inverseOf q) | (q owl:inverseOf p) |
| | (p owl:inverseOf q) (x p y) | (y q x) |
| | (p rdf:type owl:SymmetricProperty) (x p y) | (y p x) |

schema only

data + schema

not relevant

(*) Figure adapted from “Towards Efficient Schema-Enhanced Pattern Matching over RDF Data Streams”. ISWC’11 slides.