CITY
UNIVERSITY OF LONDON
EST 1894

RDFS Semantics

Ernesto Jiménez-Ruiz

Lecturer in Artificial Intelligence

Extra session and Forums

- Extra Q&A hours:
  - During during drop-in session (Wednesdays 1-2pm).
  - Thursdays 1-2pm
- Reading week (Week 6) extra session for Q&A.
- Please do not hesitate to use the forums.

Recap

Recap: RDF Example

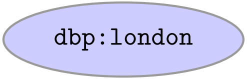
London is a city in England called Londres in Spanish

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

Recap: RDF Example

London is a city in England called Londres in Spanish

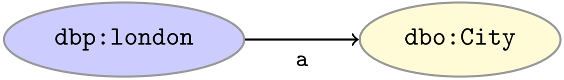
```
dbp:london a dbo:City .
dbp:london dbo:locationCountry dbp:england .
dbp:london rdfs:label "Londres"@es .
```



Recap: RDF Example

London is a city in England called Londres in Spanish

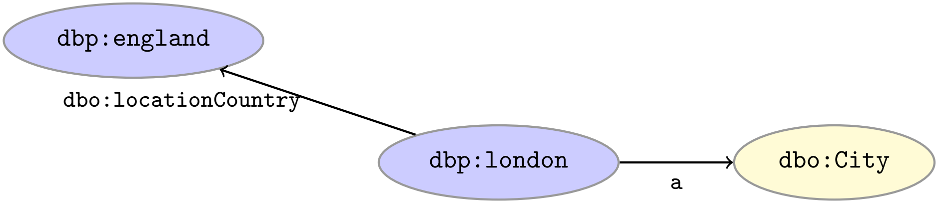
```
dbp:london a dbo:City .
dbp:london dbo:locationCountry dbp:england .
dbp:london rdfs:label "Londres"@es .
```



Recap: RDF Example

London is a city in England called Londres in Spanish

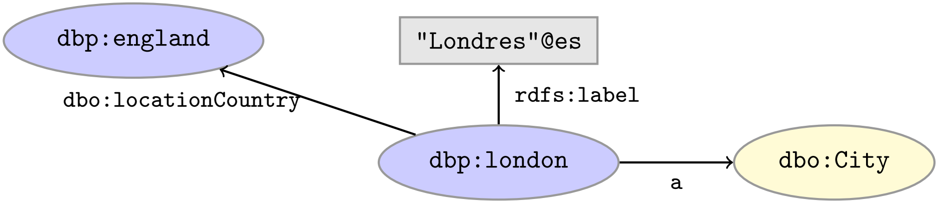
```
dbp:london a dbo:City .
dbp:london dbo:locationCountry dbp:england .
dbp:london rdfs:label "Londres"@es .
```



Recap: RDF Example

London is a city in England called Londres in Spanish

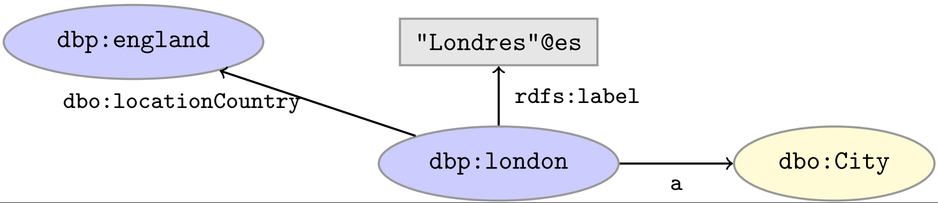
```
dbp:london a dbo:City .
dbp:london dbo:locationCountry dbp:england .
dbp:london rdfs:label "Londres"@es .
```



Recap: SPARQL Example (i)

Return all Cities:

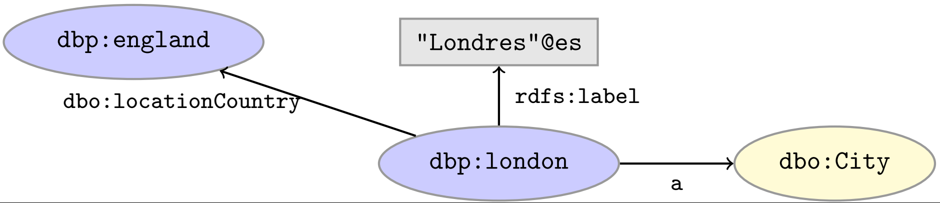
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?city WHERE {
    ?city rdf:type dbo:City .
}
```



Recap: SPARQL Example (i)

Return all Cities: Query Result= {dbp:london}

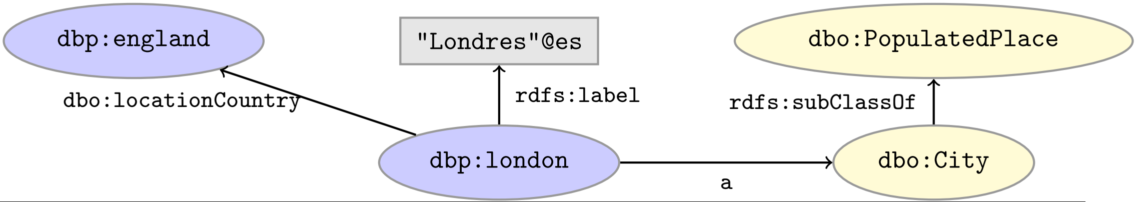
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?city WHERE {
    ?city rdf:type dbo:City .
}
```



Recap: SPARQL Example (ii)

Return all Populated Places:

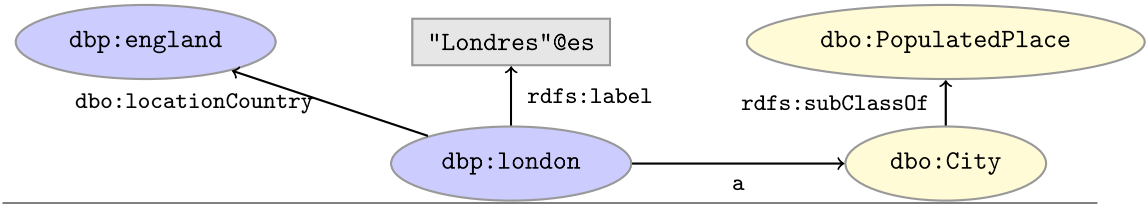
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?place WHERE {
    ?place rdf:type dbo:PopulatedPlace .
}
```



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: 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:

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

Recap: 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:

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

– RDF Schema (RDFS) extends the grammar for the “**expected**” triples, extends the vocabulary, and include a set of inference rules.

Recap: 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:

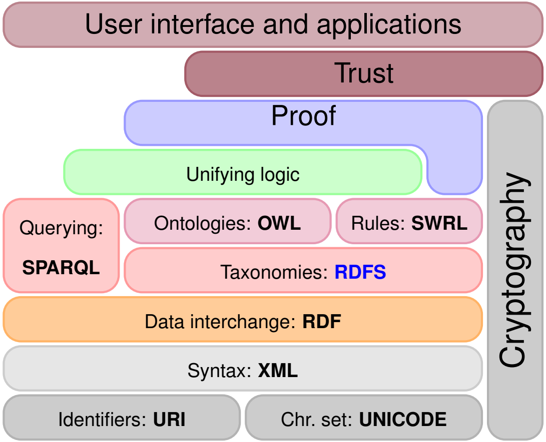
```
dbp:london rdf:type "some string"^^xsd:string .
```

– RDF Schema (RDFS) extends the grammar for the “**expected**” triples, extends the vocabulary, and include a set of inference rules.

– We will need to wait until OWL to have a proper validation mechanism.

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 though of 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

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
  - `dbp:london rdf:type "some string"^^xsd:string .`
  - *RDFS*: Not expected triple, but not prohibited (by specification).
  - *OWL semantic extension*: prohibited triple will lead to an error.

RDFS Vocabulary

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

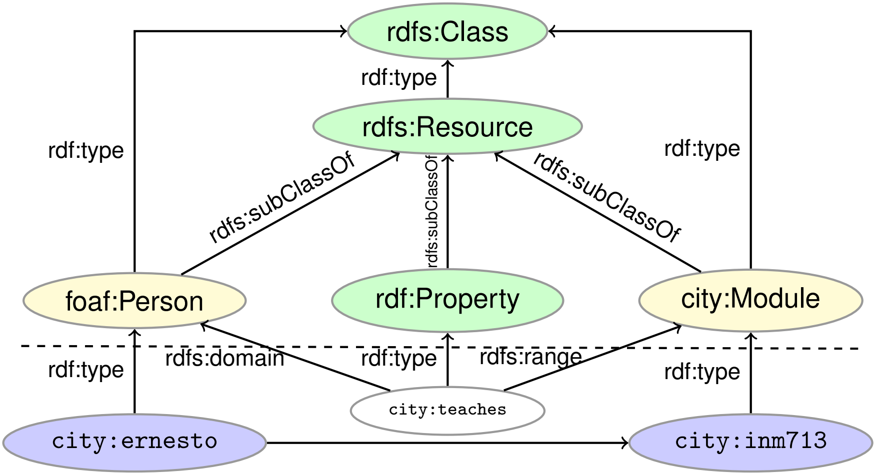
RDFS Vocabulary

- 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.
  - `rdfs:Literal`: The class of all literal values.
  - `rdfs:Datatype`: The class of all datatypes.

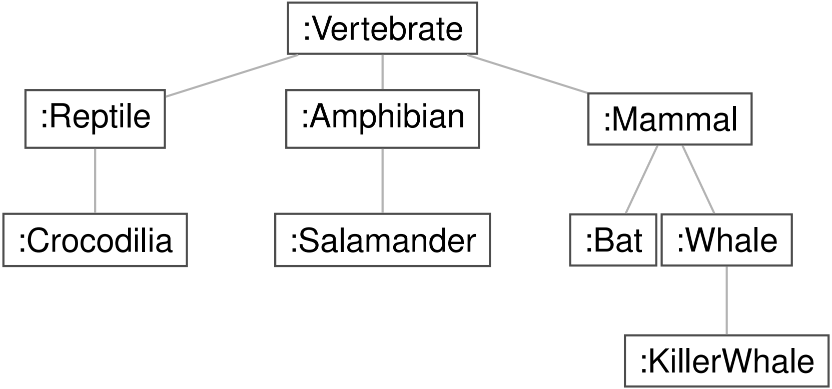
RDFS Vocabulary

- 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.
  - `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.

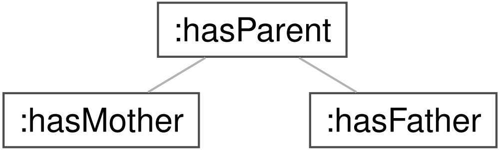
Example RDF graph and RDF Schema



Class Taxonomy (`via rdfs:subClassOf`)



Property Taxonomy (`via rdfs:subPropertyOf`)



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 .`

(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)

Classes as Sets

Sets

- A set is a mathematical object:

$$\{\text{'a'}, \mathtt{1}, \triangle\}$$

$$\{\cdots\}$$

  - Contains 'a', $\mathtt{1}$, and $\triangle$, and nothing else.

Sets

– A set is a mathematical object:

$$\{\text{‘a’}, 1, \triangle\}$$

$$\{\cdots\}$$

– Contains ‘a’, 1, and $\triangle$, and nothing else.

– There is no order between elements

$$\{1, \triangle\} = \{\triangle, 1\}$$

Sets

- A set is a mathematical object:

$$\{\text{‘a’}, 1, \triangle\}$$

$$\{\cdots\}$$

  - Contains ‘a’, 1, and $\triangle$, and nothing else.
- There is no order between elements

$$\{1, \triangle\} = \{\triangle, 1\}$$

- Nothing can be in a set several times

$$\{1, \triangle, \triangle\} = \{1, \triangle\}$$

Sets

- A set is a mathematical object:

$$\{\text{‘a’}, 1, \triangle\}$$

$$\{\cdots\}$$

  - Contains ‘a’, 1, and $\triangle$, and nothing else.
- There is no order between elements

$$\{1, \triangle\} = \{\triangle, 1\}$$

- Nothing can be in a set several times

$$\{1, \triangle, \triangle\} = \{1, \triangle\}$$

- Sets with different elements are different:

$$\{1, 2\} \neq \{2, 3\}$$

Sets: Element of-relation

– $\in$ indicates that something is element of a set:

$$1 \in \{\text{‘a’}, 1, \triangle\}$$

$$\text{‘b’} \notin \{\text{‘a’}, 1, \triangle\}$$

$\in$

Sets: Element of-relation

- $\in$ indicates that something is element of a set:

$$1 \in \{\text{'a'}, 1, \triangle\}$$

$$\text{'b'} \notin \{\text{'a'}, 1, \triangle\}$$

$\in$

- $\{3, 7, 12\}$: a set of numbers
  - $3 \in \{3, 7, 12\}$, $0 \notin \{3, 7, 12\}$
- $\{\text{'a'}, \text{'b'}, \ldots, \text{'z'}\}$: a set of letters
  - $\text{'y'} \in \{\text{'a'}, \text{'b'}, \ldots, \text{'z'}\}$, $\text{'æ'} \notin \{\text{'a'}, \text{'b'}, \ldots, \text{'z'}\}$,
- $\mathbb{N} = \{1, 2, 3, \ldots\}$: the set of all natural numbers
  - $713 \in \mathbb{N}$, $\pi \notin \mathbb{N}$.

Sets: Element of-relation

- $\in$ indicates that something is element of a set:

$$1 \in \{\text{‘a’}, 1, \triangle\}$$

$$\text{‘b’} \notin \{\text{‘a’}, 1, \triangle\}$$

$\in$

- $\{3, 7, 12\}$: a set of numbers
  - $3 \in \{3, 7, 12\}$, $0 \notin \{3, 7, 12\}$
- $\{\text{‘a’}, \text{‘b’}, \ldots, \text{‘z’}\}$: a set of letters
  - $\text{‘y’} \in \{\text{‘a’}, \text{‘b’}, \ldots, \text{‘z’}\}$, $\text{‘æ’} \notin \{\text{‘a’}, \text{‘b’}, \ldots, \text{‘z’}\}$,
- $\mathbb{N} = \{1, 2, 3, \ldots\}$: the set of all natural numbers
  - $713 \in \mathbb{N}$, $\pi \notin \mathbb{N}$.
- The set P_{inm713} of people in the zoom meeting right now
  - $\texttt{city:ernesto} \in P_{inm713}$, $\texttt{dbp:Johnny\_Depp} \notin P_{inm713}$.

The Empty Set

– A set that has no elements.

– This is called the *empty set*

– Notation: $\emptyset$ or $\{\}$

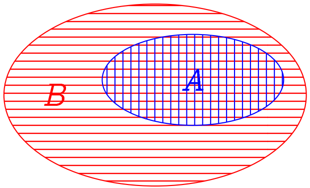
– $x \notin \emptyset$, for any x

$$\emptyset$$

Subsets

- Let A and B be sets
- *if* every element of A is also in B
- *then* A is called a *subset* of B

$$A \subseteq B$$

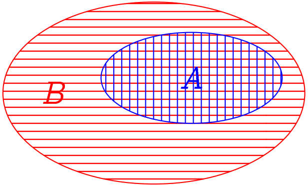


$$\subseteq$$

Subsets

- Let A and B be sets
- *if* every element of A is also in B
- *then* A is called a *subset* of B

$$A \subseteq B$$



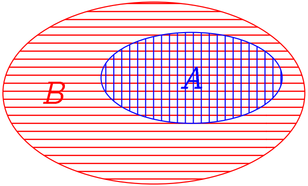
$\subseteq$

- Examples
  - $\{\texttt{city:ernesto},\ \texttt{city:dave}\} \subseteq P_{inm713}$
  - $\{1,3\} \not\subseteq \{1,2\}$
  - $\{1,3\} \subseteq \mathbb{N}$
  - $\emptyset \subseteq A$ for any set A

Subsets

- Let A and B be sets
- *if* every element of A is also in B
- *then* A is called a *subset* of B

$$A \subseteq B$$



$$\subseteq$$

- Examples
  - $\{\texttt{city:ernesto},\ \texttt{city:dave}\} \subseteq P_{inm713}$
  - $\{1,3\} \not\subseteq \{1,2\}$
  - $\{1,3\} \subseteq \mathbb{N}$
  - $\emptyset \subseteq A$ for any set A
- $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$

Intuition: Classes as Sets of Resources

- We can think of an `rdfs:Class` as denoting a set of Resources.
- (Not exactly, but OK for intuition).

Intuition: Classes as Sets of Resources

- We can think of an `rdfs:Class` as denoting a set of Resources.
- (Not exactly, but OK for intuition).

RDFS	Set Theory
A `rdf:type rdfs:Class`	A is a set of resources
x `rdf:type` A	$x \in A$
A `rdfs:subClassOf` B	$A \subseteq B$
`:Person rdf:type rdfs:Class`	`:Person` is a set of resources
`:ernesto rdf:type :Person`	`:ernesto` $\in$ `:Person`
`:Person rdfs:subClassOf :Animal`	`:Person` $\subseteq$ `:Animal`

Properties as Relations

Pairs

– A pair is an *ordered* collection of two objects

$$\langle x, y \rangle \qquad\qquad \Big\langle \cdot \ \cdot \ \cdot \Big\rangle$$

– Equal if components are equal:

$$\langle a, b \rangle = \langle x, y \rangle \quad \text{if and only if} \quad a = x \quad \text{and} \quad b = y$$

Pairs

– A pair is an *ordered* collection of two objects

$$\langle x, y \rangle \qquad\qquad \Big\langle \cdot \cdot \cdot \Big\rangle$$

– Equal if components are equal:

$$\langle a, b \rangle = \langle x, y \rangle \quad \text{if and only if} \quad a = x \quad \text{and} \quad b = y$$

– Order matters:

$$\langle 1, \text{‘a’} \rangle \neq \langle \text{‘a’}, 1 \rangle$$

– An object can be twice in a pair:

$$\langle 1, 1 \rangle$$

– $\langle x, y \rangle$ is a pair, no matter if $x = y$ or not.

The Cross Product

– Let A and B be sets.

– Construct the set of all pairs $\langle a, b \rangle$ with $a \in A$ and $b \in B$.

– This is called the *cross product* of A and B, written

$$A \times B \qquad \times$$

The Cross Product

- Let A and B be sets.
- Construct the set of all pairs $\langle a, b \rangle$ with $a \in A$ and $b \in B$.
- This is called the *cross product* of A and B, written

$$A \times B \qquad\qquad \times$$

- Example:
  - $A = \{1, 2, 3\}$, $B = \{\text{‘a’}, \text{‘b’}\}$.
  - $A \times B = \{\ \langle 1, \text{‘a’} \rangle,\ \langle 2, \text{‘a’} \rangle,\ \langle 3, \text{‘a’} \rangle,$
    $\langle 1, \text{‘b’} \rangle,\ \langle 2, \text{‘b’} \rangle,\ \langle 3, \text{‘b’} \rangle\ \}$

Relations

- A *relation* R between two sets A and B is...
- ...a set of pairs $\langle a, b \rangle \in A \times B$

$$R \subseteq A \times B$$

- We often write $a \; R \; b$ to say that $\langle a, b \rangle \in R$
- A relation R *on* some set A is a relation between A and A:

$$R \subseteq A \times A = A^2$$

Example: Family Relations

– Consider the set $A = \{\text{Homer}, \text{Marge}, \text{Bart}, \text{Lisa}, \text{Maggie}\}$.

– Consider a relation P on A such that

$$x \; P \; y \quad \text{iff} \quad x \text{ is parent of } y$$

– As a set of pairs:

$$P = \{ \begin{array}{lll} \langle \text{Homer}, \text{Bart} \rangle, & \langle \text{Homer}, \text{Lisa} \rangle, & \langle \text{Homer}, \text{Maggie} \rangle, \\ \langle \text{Marge}, \text{Bart} \rangle, & \langle \text{Marge}, \text{Lisa} \rangle, & \langle \text{Marge}, \text{Maggie} \rangle \end{array} \} \subseteq A^2$$

– For instance:

$$\langle \text{Homer}, \text{Bart} \rangle \in P \qquad \langle \text{Marge}, \text{Maggie} \rangle \in P$$

Set operations on relations

- Since relations are just sets of pairs, we can use set operations and relations on them.
- We say that R_1 is a subrelation of R if $R_1 \subseteq R$.

Set operations on relations

– Since relations are just sets of pairs, we can use set operations and relations on them.

– We say that R_1 is a subrelation of R if $R_1 \subseteq R$.

– E.g.: if F is the father-of-relation,

$$F = \{\langle \text{Homer}, \text{Bart} \rangle, \langle \text{Homer}, \text{Lisa} \rangle, \langle \text{Homer}, \text{Maggie} \rangle\}$$

then $F \subseteq P$ (P=parent-of relation).

– If M is the mother-of-relation,

$$M = \{\langle \text{Marge}, \text{Bart} \rangle, \langle \text{Marge}, \text{Lisa} \rangle, \langle \text{Marge}, \text{Maggie} \rangle\}$$

then $M \subseteq P$ (P=parent-of relation).

Domain and Range of Relations

– Given a relation R from A to B ($R \subseteq A \times B$)
– The *domain* of R is the set of all x with $x\ R \cdots$:

$$\mathrm{dom}\, R = \{x \in A \mid xRy \text{ for some } y \in B\}$$

Domain and Range of Relations

- Given a relation R from A to B ($R \subseteq A \times B$)
- The *domain* of R is the set of all x with $x\ R \cdots$:

$$\operatorname{dom} R = \{x \in A \mid xRy \text{ for some } y \in B\}$$

- The *range* of R is the set of all y with $\cdots R\ y$:

$$\operatorname{rg} R = \{y \in B \mid xRy \text{ for some } x \in A\}$$

Domain and Range of Relations

- Given a relation R from A to B ($R \subseteq A \times B$)
- The *domain* of R is the set of all x with $x\ R \cdots$:

$$\operatorname{dom} R = \{x \in A \mid xRy \text{ for some } y \in B\}$$

- The *range* of R is the set of all y with $\cdots R\ y$:

$$\operatorname{rg} R = \{y \in B \mid xRy \text{ for some } x \in A\}$$

- Example:
  - $R = \{\langle 1, \triangle \rangle, \langle 1, \square \rangle, \langle 2, \Diamond \rangle\}$
  - $\text{dom}_R = \{1, 2\}$
  - $\text{rg}_R = \{\triangle, \square, \Diamond\}$

Intuition: Properties as Relations

- An `rdf:Property` is like a relation on resources.
- (not exactly, but OK as intuition).

RDFS	Set Theory
R `rdf:type rdf:Property`	R is a relation on resources
$x\ R\ y$	$\langle x, y \rangle \in R$
R `rdfs:subPropertyOf` S	$R \subseteq S$
R `rdfs:domain` A	$\text{dom}_R \subseteq A$
R `rdfs:range` B	$\text{rg}_R \subseteq B$

(*) Without domain and range R is a relation from `rdf:Class` to `rdf:Class` (*i.e.*, $R \subseteq$ $\texttt{rdf:Class} \times \texttt{rdf:Class} = \texttt{rdf:Class}^2$)

Intuition: Properties as Relations

– An `rdf:Property` is like a relation on resources.
– (not exactly, but OK as intuition).

RDFS	Set Theory
`:teaches rdf:type rdf:Property`	`:teaches` is a relation on resources
`:ernesto :teaches :inm713`	$\langle$`:ernesto`,`:inm713`$\rangle \in$ `:teaches`
`:teaches rdfs:subPropertyOf :manages`	`:teaches` $\subseteq$ `:manages`
`:teaches rdfs:domain :Person`	dom `:teaches` $\subseteq$ `:Person`
`:teaches rdfs:range :Module`	rg `:teaches` $\subseteq$ `:Module`

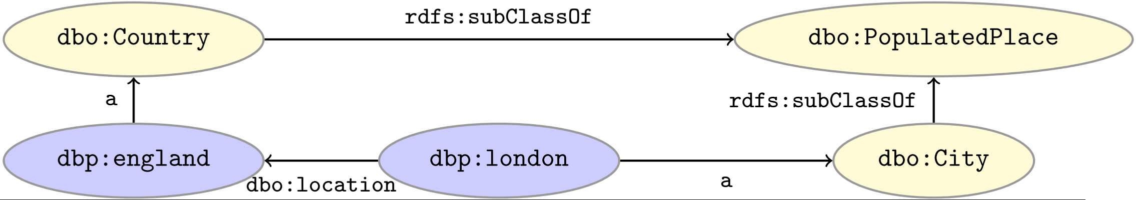
(*) With domain and range `:teaches` is a relation from `:Person` to `:Module` (*i.e.*, `:teaches` $\subseteq$ `:Person` $\times$ `:Module`)

Entailment via Model-Theoretic Semantics

SPARQL Example

Return all Populated Places:

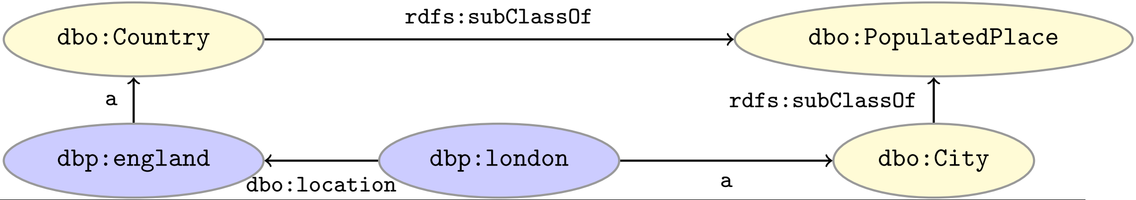
```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?place WHERE {
    ?place rdf:type dbo:PopulatedPlace .
}
```



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 .
}
```

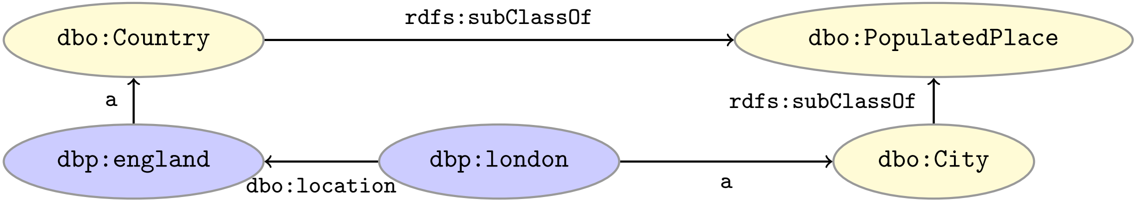


Entailment in RDFS

– 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: `dbp:london rdf:type dbo:PopulatedPlace` and add it to the graph below?

– Similarly for `dbp: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 $\mathcal{G}$:
  - $\texttt{dbo:City}^{\mathcal{I}} = \{\texttt{dbp:london}\}$
  - $\texttt{dbo:Country}^{\mathcal{I}} = \{\texttt{dbp:england}\}$
  - $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbp:london}, \texttt{dbp:england}\}$
  - $\texttt{dbo:location}^{\mathcal{I}} = \{\langle\texttt{dbp:london, dbp:england}\rangle\}$

Model-Theoretic Semantics (ii)

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

Model-Theoretic Semantics (iii)

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

Model-Theoretic Semantics (iii)

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

Model-Theoretic Semantics (iii)

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

Model-Theoretic Semantics (iii)

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

Model-Theoretic Semantics (iii)

- t = `dbp:london rdf:type dbo:PopulatedPlace`
- Does $\mathcal{I} \models t$?
  - Yes: $\texttt{dbo:PopulatedPlace}^{\mathcal{I}} = \{\texttt{dbp:london}, \texttt{dbp:england}\}$
- 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=`dbp:london rdf:type dbo:Country`)?
  - No: $\mathcal{I}$ is a counter example. $\mathcal{I} \models \mathcal{G}$ but $\mathcal{I} \not\models t_2$

Model-Theoretic Semantics (iv)

- 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**
  - and P is the **conclusion.**
- 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:

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

Inference rules (iii)

- The whole set of inference rules given for a logic is called **deduction calculus**.
- $\vdash$ is the i**nference relation**, while $\models$ was the entailment relation using model theoretic semantics.
  - We write $\Gamma \vdash P$ if we can deduce P from the premises Γ.
- In our setting
  - the **premises** Γ are a **set of triples** (*i.e.*, a (sub)graph $\mathcal{G}$),
  - the **conclusion** is a **new triple** t

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. . . ”

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.

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.

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.

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:**

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

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 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:**

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

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:**

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

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:**

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

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 $\mathcal{G} \cup \{t_1\} \vdash t$)

Foundations of Semantic Web Technologies. Chapter 3.

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 $\mathcal{G} \models t$

Foundations of Semantic Web Technologies. Chapter 3.

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 $\mathcal{G} \vdash t$

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 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** includes additional vocabulary and includes consistency-checks (next week!).

Laboratory: RDFS Semantics

Tasks

- Manually checking inferences.
- Extracting inferences programmatically and checking via SPARQL.
- **Python**: We are using the OWL-RL library (new) `owlrl.DeductiveClosure(owlrl.RDFS_Semantics).expand(g)`
- **Java**: Jena API `InfModel inf_model = ModelFactory.createRDFSModel(model);`