Semantic Data

Chapter 3 : The semantic web resource description framework

Jean-Louis Binot
Sources and recommended readings

- There are no additional references required for this chapter for the theory.
  - The students are expected to consult the W3C sources for practice, and to understand XML (as a basis for RDF; XML itself is not part of the required material).

- Sources and useful additional readings:
  - The RDF section is inspired from the W3C RDF Primer 1.1 (2014) and the RDF Primer (2004).
  - The RDFS section is inspired from the same sources and RDF Semantics, section 7.
  - Books: Semantic Web Primer (Antoniou and van Harmelen 2004); Web Data Management (Abiteboul et al. 2011).

- University courses having partially inspired ideas and examples for this chapter:
  - Web sémantique, O. Corby, Inria.
  - Apprentissage symbolique et web sémantique, B. Amman, UPMC.
  - Semantic Web Technologies, H. Paulheim, Universität Manheim.
The semantic web stack

Why not HTML or XML?

Resource description framework

RDF Schema
The semantic web standards (W3C consortium)

The stack of semantic web standards is made of multiple layers building on each other:

- **RDF** (Resource Description Framework):
  - Semantic information expressed as triples.

- **RDFS** (Resource Description Framework Schema):
  - Extension of RDF to support simple ontologies.

- **OWL** (Web Ontology Language):
  - Representation of ontologies mapped on description logics.

- **SPARQL** (RDF query language).

- **RIF** (Rule Interchange Format) and **SWRL** (Semantic Web Rule Language).

- The initial syntax is based on **XML** (other syntaxes have been added).
Agenda

1. The semantic web stack
2. Why not HTML or XML?
3. Resource description framework
4. RDF Schema
Why not just HTML?

- HTML is oriented toward visual presentation and keyword search, targeting human users.
- However, the semantic web is intended to be a web of linked data accessible by machines.
- We need a way to describe data content in a wide variety of fields, from e-commerce to bioinformatics, and many others.
- HTML is not suited for that task.

(Example from Antoniou and van Harmelen, 2004)
Might XML be a better solution?

Reminders:

- **XML** (eXtensible Markup Language) is a standard for data exchange on the Web:
  - Flexible, generic, platform independent;
  - Coming with a wide variety of tools (parsers, programming interfaces, manipulation languages);
  - Now an industry standard for data integration.

- An XML element consists of an opening tag, its content, and a closing tag:
  
  ```xml
  <lecturer>Jean-Louis Binot</lecturer>
  ```
  - Tag names can be chosen almost freely.

- Typing an XML document can be done with a Document Type Definition, or an XML Schema, richer and newer.
  - Typing means precision which elements and attributes can be used.
Why not just XML: an example

```xml
<course name="Semantic Data">
  <lecturer>Jean-Louis Binot</lecturer>
</course>

<lecturer name="Jean-Louis Binot">
  <teaches>Semantic Data</teaches>
</lecturer>

<teachingOffering>
  <lecturer>Jean-Louis Binot</lecturer>
  <course>Semantic Data</course>
</teachingOffering>

(Adapted from Antoniou and van Harmelen 2004)
```

- The examples on the left correspond to the same relation (below) but show different ways to represent it in XML.
- If we want to query who is the lecturer of the course, it is impossible without knowing the specific tag structure used.
- An XML tag structure does not have any standard meaning:
  - The meaning is only assigned by the application using it.
  - XML has a syntax but no way to describe the semantics of the data!
Agenda

1. The semantic web stack
2. Why not HTML or XML?
3. Resource description framework
4. RDF Schema
Basic ideas of RDF

RDF aims at describing web resources. It relies on 3 fundamental concepts:

- Resources.
- Properties.
- Statements.
Resources

- **Resources** can represent anything we want to talk about or to refer to:
  - Web resources (documents, images, services, groups of resources...);
  - Individual entities or objects of the real world (humans, corporations, books...);
  - Abstract concepts (classes, relations, properties...).

- Each resource is identified by a **Universal resource identifier** or URI ([RFC 3986](https://tools.ietf.org/html/rfc3986)):
  - URLs (uniform resource locators) are a type of URI, providing explicit location of the resource:
    - URI = scheme://authority[?query][#fragment]
    - http://dbpedia.org/resource/Leonardo_da_Vinci
    - http://www.example.org/bob#me
  - An URI can also be a universal resource name (URN) for a resource without location information.
Properties

- **Properties**: special kind of resources, which describe relations between resources.
  - Examples: *written by*, *age*, *title*…

- Properties are also identified by URIs.

- Using URIs both for resources and properties has important advantages:
  - URIs are global identifiers: they provide a worldwide unique naming scheme.
  - Hence they reduce the homonym problems arising with distributed data representation.
Statements

- Statements about resources take the form of triples $<$Subject Predicate Object$>^{(*)}$. A statement can be interpreted as a logical (FOL) formula: $Predicate(Subject, Object)$.

- Triples can also express properties of resources: $<$Resource Property Value$>$. Can be interpreted in FOL as $Property(Resource, Value)$.

- Abstract example: the course Semantic Data has a creator who is JL Binot.

Triple: $<$Semantic Data$> <$creator$> <$JL Binot$>^{(*)}$.

*: We use $<$ to denote informally triple elements.
The RDF data model is based on directed labelled graphs

- Abstract example:

  `<Bob> <is a> <person> .
  `<Bob> <is a friend of> <Alice> .
  `<Bob> <is born on> <14 July 1990> .
  `<Bob> <is interested in> <The Mona Lisa> .
  `<the Mona Lisa > <was created by> <Leonardo da Vinci> .
  `<the video “la Joconde in Washington”> <is about> < The Mona Lisa > .

- These triples form a labelled directed graph.

- It is a semantic network.

(Source: RDF primer 1.1)
Triples with URIs

- Let us assume the following sources:
  - http://www.example.org describes an organisation teaching courses;
  - http://www.example.org/courses contains the resource descriptions of the courses;
  - http://www.example.org/staffid contains the resource descriptions of people;

- Using explicit URIs, our abstract example would become:
RDF serialization - Turtle

- An RDF graph must be serialized to be stored into files.
  - Several syntaxes exist (cf. later in this chapter).

When not said otherwise, we will use the Turtle syntax (Terse RDF Triple language).

- Syntax of a basic RDF statement in Turtle:
  
  Subject Predicate Object .  
  
  (do not forget the dot !)

- Example from previous slide in Turtle:


  Still not so terse ! There are ways to make the syntax much more readable.
Namespaces and reference vocabularies

- RDF is agnostic about what URIs represent. URIs are given meaning by vocabularies.

- Vocabularies are organized by using namespaces:
  - A namespace is a set of unique symbols used to organize the names of objects.
  - The structure and meaning of the names can be fixed within the namespace.
  - Namespaces can be used for disambiguation: within a namespace, a name is always unique.
  - The namespace used is indicated by a prefix abbreviation, which shortens URI notation.

- Ontologies defining vocabularies for specific domains can be published as standard namespaces.

Example using the Dublin Core Metadata, a reference vocabulary for digital documents.

```plaintext
PREFIX dc: <http://purl.org/dc/terms/>
...  
dc:creator
```
Dublin Core Metadata Initiative

- ISO Standard 15836; 15 vocabulary terms; a small but worldwide used taxonomy:
  - contributor
  - format
  - rights
  - coverage
  - identifier
  - source
  - creator
  - language
  - subject
  - date
  - publisher
  - title
  - description
  - relation
  - type

- Can be embedded into HTML; provides among others metadata for all ebooks.

```xml
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:identifier id="BookId" opf:scheme="ISBN">2-290-33529-0</dc:identifier>
  <dc:title>Hamlet</dc:title>
  <dc:language>fr</dc:language>
  <dc:creator>William Shakespeare</dc:creator>
  <dc:date>2004</dc:date>
</metadata>
```
Some usual vocabularies

- Some well-know vocabularies used in ontology contexts:
  - **PREFIX dcterms**: `<http://purl.org/dc/terms/>`
    - Dublin Core Metadata: digital documents ontology.
  - **PREFIX foaf**: `<http://xmlns.com/foaf/0.1/>`
    - Friend Of A Friend, ontology describing people and their relations.
  - **PREFIX dbpedia**: `<http://dbpedia.org/resource/>`
    - DBPedia: ontology of structured data extracted from Wikipedia.
  - **PREFIX skos**: `<http://www.w3.org/2004/02/skos/core/>`
    - Simple Knowledge Organisation System: a simplified ontology.

- Abbreviations for W3C standards:
  - **PREFIX rdf**: `<http://www.w3.org/1999/02/22-rdf-syntax-ns/>`
    - Namespace for RDF.
  - **PREFIX rdfs**: `<http://www.w3.org/2000/01/rdf-schema/>`
    - Namespace for RDFS.
  - **PREFIX owl**: `<http://www.w3.org/2002/07/owl>`
    - Namespace for OWL.
Namespaces ./.

- http://www.example.org is a “second-level” domain name reserved for documentation purposes. It can be used to make examples in textbooks.

- We will use ex: as the corresponding namespace abbreviation:
  
  PREFIX ex: <http://www.example.org>

- Other variations on the “example” prefix can also be used, for instance:

  PREFIX externs: <http://www.example.org/terms/> (for terms used in an example organisation)
  PREFIX exstaff: <http://www.example.org/staffid/> (for staff within the organisation)

- A default name space can be specified (for any namespace):

  PREFIX: <http://example.org/>
Types of triple elements

- Three types of elements can be used in a triple:
  - **URIs** or Universal Resource Identifiers;
  - **Literals**: basic values which are not URIs;
  - **Blanks nodes**: can stand for an object without identifying it (like a variable).

- Their use is subject to the following limitations:
  - **URIs** can be used in any position of a triple;
  - **Literals** can only be used in object position;
  - **Blank nodes** only in subject or object position.

<table>
<thead>
<tr>
<th></th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Literal</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Blank Node</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Example with URIs and literals


2. The URI for an INA video about the Mona Lisa in Europeana.

3. “Mona Lisa” is a literal of type string (in Turtle, specifying the type string is not mandatory).

4. “1990-07-04”^^xsd:date is a literal of type date.

The notation ^^xsd: introduces a type following the conventions of XML Schema datatypes.

Example 1 (source: RDF primer 1.1)
Literals and datatypes

- Literals are used for values such as strings, numbers, and dates.

- A literal in an RDF graph consists of two or three elements:
  - A lexical form, being a Unicode string;
  - A datatype URI, which determines how the lexical form maps to a literal value, and
  - Iff the datatype is langString (a language string), a non-empty language tag.

Concrete syntaxes may omit the string datatype for literals of type string. Similarly, most concrete syntaxes represent language-tagged strings without the langstring datatype.

- Examples:
  - "1990-07-04"^^xsd:date
  - "That Seventies Show"@en
Blank nodes

- The third type of element which can appear in a triple is a blank node.
  - They represent unknown resources, which are not literals and not identified by URIs.
  - In Turtle the blank node is represented by a variable prefixed by _: , such as _:x, or as an anonymous blank node :[].

- Example (Turtle syntax)

  Alice knows someone named Leonardo.

  :Alice foaf:knows _:x .
  _:x foaf:name "Leonardo" .

  :Alice foaf:knows [foaf:name "Leonardo"] .
Use of blank nodes for structured data

- Structured data are conveniently represented by blank nodes:

Example: a structured address.

```
exstaff:85740 exterms:address _:johnaddress .
_:johnaddress exterms:street "1501 Grant Avenue" .
_:johnaddress exterms:city "Bedford" .
_:johnaddress exterms:state "Massachusetts" .
_:johnaddress exterms:postalCode "01730" .
```

(Source: RDF primer 2004)
Instances and typing

- **rdf:type** models the **instance** relationship, or **types** a resource:
  
  \[
  \text{ex:Bob} \ \text{rdf:type} \ \text{foaf:Person}.
  \]

  The resource **Bob** is an **instance** of the **class Person**.

- Resources may have several classes or types:
  
  \[
  \text{ex:SemanticData} \ \text{rdf:type} \ \text{ex:course}.
  \text{ex:SemanticData} \ \text{rdf:type} \ \text{ex:document}.
  \]

- A class may also be an instance:
  
  \[
  \text{ex:tweety} \ \text{rdf:type} \ \text{ex:robin}.
  \text{ex:robin} \ \text{rdf:type} \ \text{ex:species}.
  \]

  This is the first (insufficient) step to take web data towards ontologies.
Syntactic sugar

- Turtle introduces the abbreviation `a` to stand for `rdf:type`.

  ```
  ex:Bob a foaf:Person .
  ```
  close to the intuitive understanding `Bob (is) a Person`.

- A symbol `;` separates triples with the same `subject`.

- A symbol `,` separates triples with the same `predicate`.

```
PREFIX ex: <http://www.example.org>
PREFIX dc: <http://purl.org/dc/terms/>

ex:SemanticData dc:subject "Donnée sémantiques"@fr ,
      "Semantic data"@en ;
    dc:creator "Jean-Louis Binot".
```
The Mona Lisa example in Turtle

```turtle
PREFIX : <http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX schema: <http://schema.org/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX wd: <http://www.wikidata.org/entity/>

:bob
    a foaf:Person ;
    foaf:knows :alice ;
    schema:birthDate "1990-07-04"^^xsd:date ;
    foaf:topic_interest wd:Q12418 .

wd:Q12418
    dcterms:title "Mona Lisa" ;

<http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619>
    dcterms:subject wd:Q12418 .
```
Reification

- RDF provides a mechanism for reification.

- It allows to make statements about another statement seen as a resource:
  - The class `rdf:Statement` is used to type a resource as a statement.
  - The properties `rdf:subject`, `rdf:predicate` and `rdf:object` identify the subject, predicate and object of a statement.

  \( S \text{ rdf:subject } R \) states that \( S \) is an instance of `rdf:Statement` and that the subject of \( S \) is \( R \).

- **Example**: resource item 10245 weights 2.4 Kgs. This information was created by staff member 3405.

\[
\text{exproducts:triple12345} \quad \text{rdf:type} \quad \text{rdf:Statement} .
\]
\[
\text{exproducts:triple12345} \quad \text{rdf:subject} \quad \text{exproducts:item10245} .
\]
\[
\text{exproducts:triple12345} \quad \text{rdf:predicate} \quad \text{exterms:weight} .
\]
\[
\text{exproducts:triple12345} \quad \text{rdf:object} \quad "2.4"^\text{xsd:decimal} .
\]
\[
\text{exstaff:id3405} \quad \text{dc:creator} \quad \text{exproducts:triple12345} .
\]
Syntaxes for expressing RDF graphs

RDF graphs can be serialized using different syntaxes:

- The **Turtle family** of RDF languages (**N-Triples**, **Turtle**, **TriG** and **N-Quads**).
- **RDF/XML** (XML syntax for RDF).
  - Initially it was the only available syntax; some people still call this syntax "RDF".
- **JSON-LD** (JSON for **Linked Data**) is a JSON-based RDF syntax.
  - Used to transform JSON documents to RDF with minimal changes, or to serialize RDF datasets in JSON.
- **RDFa** (RDF in HTML attributes): used for embedding RDF in HTML and XML.

For details see the W3C manuals.
The Mona Lisa example in RDF/XML

```xml
<rdf:RDF
  xmlns:dcterms="http://purl.org/dc/terms/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns="http://example.org/"
  xmlns:wd="http://www.wikidata.org/entity/"
  xmlns:schema="http://schema.org/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <rdf:Description rdf:about="http://www.wikidata.org/entity/Q12418">
    <dcterms:title>Mona Lisa</dcterms:title>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/bob">
    <foaf:topic_interest rdf:resource="http://www.wikidata.org/entity/Q12418"/>
    <schema:birthDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">1990-07-04</schema:birthDate>
    <foaf:knows rdf:resource="http://example.org/alice"/>
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619">
    <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
  </rdf:Description>
</rdf:RDF>
```

- Attributes of the `rdf:RDF` start tag contain namespace declarations with their abbreviations.

- The `rdf:Description` element defines sets of triples that have as subject the URI specified by `rdf:about`, following the structure:

```xml
<description subject>
  <predicate object >
    <predicate object > ...
  </predicate object>
</description>
```

Very verbose!
**RDFa (RDF in HTML attributes)**

- RDFa is a serialization of RDF embedded in HTML, XML or XHTML to enrich web pages with structured semantic data.

- Why is that important?
  - Most of web data is in HTML; new content is generated everyday.
  - Authors of web sites traditionally do not like to generate separate RDF files.
  - Solution: add structured data to the HTML pages; let processors extract those and turn them into RDF.
  - Search engines can use these annotations to enrich search results (e.g. schema.org and Rich Snippets).

- RDFa evolved into a huge source of RDF triples on the web.

- It is a bridge between the web of documents and the web of data.
<div vocab="http://xmlns.com/foaf/0.1/" about="#me">

My name is <span property="name">John Doe</span> and my blog is called <a rel="homepage" href="http://example.org/blog/">Understanding Semantics</a>. </div>

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

<#me> foaf:name "John Doe" ;

    foaf:homepage <http://example.org/blog/> .
A review snippet from Google

The review itself is embedded as an anonymous filler of the property review.

(source https://developers.google.com/search/docs/data-types/review-snippet)
Triples generated from the review snipet

[] a ns1:Product;
ns1:image <catcher-in-the-rye-book-cover.jpg>;
ns1:name "The Catcher in the Rye";
ns1:review [ a ns1:Review;
    ns1:author [ a ns1:Person;
        ns1:name "John Doe" ];
    ns1:datePublished "2006-05-04";
    ns1:name "A masterpiece of literature";
    ns1:publisher [ a ns1:Organization;
        ns1:name "Washington Times" ];
    ns1:reviewBody "I really enjoyed this book. It captures the essential challenge people face as they try make sense of their lives and grow to adulthood.";
    ns1:reviewRating [ a ns1:Rating;
        ns1:ratingValue "5" ] ].
Formal semantics of RDF

Why does RDF need semantics?

- Semantic networks by themselves do not have clear semantics (cf. chapter 1).
- RDF is a basis for more advanced ontology languages such as RDFS and OWL.
  - These languages aim to model knowledge and support inferences; they need formal semantics.
- Semantic web query engines using SPARQL operate on RDF documents.
  - Tools were incompatible: same RDF documents, same SPARQL queries, different results!
- A formal semantics was defined for RDF (W3C document [RDF11-MT]).
  - The approach of reference is to define directly a model-theoretic semantics for RDF.
  - For the sake of simplicity, we will introduce the main points by referring to first order logic.
Semantics of RDF graphs

- Sentences of the language:
  - Any Subject Predicate Object statement or triple is a basic sentence of the RDF language.
  - Complex sentences of the RDF language are finite sets of triples, forming an RDF graph.

- Basic semantic rules
  - An RDF statement can be interpreted as a FOL formula: $\text{Predicate(Subject, Object)}$.
  - A set of triples is interpreted as a conjunction. An RDF graph is True exactly when all the triples in it are True.
  - URIs used in subject, predicate, and object are global in scope, naming the same thing each time they are used.
  - A blank node corresponds to an existentially quantified variable.

\[
\exists p \ (\text{knows(ex:Alice, p)} \land \text{name(p, "Leonardo")})
\]
(prefixes omitted for the sake of readability)
Semantics of RDF graphs ./.

- Using our usual logic terminology, we will say that:
  - An interpretation $I$ satisfies a statement or triple of the form $s \ p \ o$ if $p^I(s^I, o^I)$ is True.
  - An interpretation $I$ satisfies a graph $G$ if every triple included in $G$ is True under $I$.
  - A graph $G$ is satisfiable if at least one interpretation satisfies it.
  - A graph $G$ entails a graph $E$, noted $G \models E$, when every interpretation which satisfies $G$ also satisfies $E$.
  - If two graphs $E$ and $F$ entail each other, they are logically equivalent.

Note: To be complete, the definitions of this section concern the concepts of simple interpretation and simple entailment. RDF is intended to be used as a basis for other languages (RDFS, OWL) and more complex entailment regimes will occur for them.
The interpolation lemma

The interpolation lemma allows to check RDF graph entailment on a syntactic basis.

- **Subgraph**: a subgraph of an RDF graph is a subset of the triples in the graph.
  - A graph entails all its subgraphs.

- **Graph instance**:
  - if $M$ is a functional mapping from a set of blank nodes to another set of literals, blank nodes and URIs,
    - Any graph obtained from a graph $G$ by replacing a subset $N$ of the blank nodes in $G$ by $M(N)$ is an instance of $G$.
    - A graph is entailed by any of its instances.

- **Interpolation lemma**:
  - If $G_1$ and $G_2$ are two RDF graphs, $G_1 \vdash G_2$ iff there is a subgraph of $G_1$ which is an instance of $G_2$.
  - It follows directly from the two previous points.
Interpolation lemma: example

- There exists a subgraph of $G_1$ which is an instance of $G_2$. Hence $G_1 \models G_2$
  - Using the mapping $M(_{b1})=\text{ex:car1}$.

- There is no subgraph of $G_2$ being an instance of $G_1$. Hence $G_2 \not\models G_1$

- Simple graph entailment is \textit{decidable and NP-complete} in general.
  - Complexity reduces to polynomial order when $G_2$ contains no blank node.

(example from \textit{Apprentissage symbolique et web sémantique}, UPMC)
Agenda

1. The semantic web stack
2. Why not HTML or XML?
3. Resource description framework
4. RDF Schema
Motivation for RDFS

- RDF is a universal language annotating web resources with specific properties.
- It does not say anything more about the meaning of the resources or annotations.
  - And, unlike e.g. Dublin Core, RDFS does not redefine any application-specific terminology.
- RDF as such does not allow us to build ontologies.
- RDF Schema (RDFS) is a first step in that direction (W3C’s *one step at a time* strategy):
  - RDFS allows to express **simple** ontologies by adding a few constructs to RDF.
Example of a simple RDFS ontology

RDFS: terminological knowledge
RDF: specific properties about resources
RDFS basic ideas

- RDFS provides syntactic constructs allowing to:
  - Specify concepts (classes), class hierarchies and instances.
  - Specify properties (roles) and property hierarchies.
  - Specify which classes and properties should be used together.
  - Support inheritance.

- RDFS acts as a typing system for RDF resources.

- RDFS statements are RDF triples. RDFS ontologies are valid RDF graphs.
  - RDF Schema constructs are provided as an RDF vocabulary; its namespaces and usual prefix are:
    rdfs: http://www.w3.org/2000/01/rdf-schema#
## RDFS main constructs

<table>
<thead>
<tr>
<th>Construct (construct type)</th>
<th>Syntactic form</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class (a class)</td>
<td>C rdf:type rdfs:Class</td>
<td>C (a resource) is an RDF class</td>
</tr>
<tr>
<td>Property (a class)</td>
<td>P rdf:type rdf:Property</td>
<td>P (a resource) is an RDF property</td>
</tr>
<tr>
<td>type (a property)</td>
<td>I rdf:type C</td>
<td>I (a resource) is an instance of C (a class)</td>
</tr>
<tr>
<td>subClassOf (a property)</td>
<td>C1 rdfs:subClassOf C2</td>
<td>C1 (a class) is a subclass of C2 (a class)</td>
</tr>
<tr>
<td>subPropertyOf (a property)</td>
<td>P1 rdfs:subPropertyOf P2</td>
<td>P1 (a property) is a sub-property of P2 (a property)</td>
</tr>
<tr>
<td>domain (a property)</td>
<td>P rdfs:domain C</td>
<td>domain of P (a property) is C (a class)</td>
</tr>
<tr>
<td>range (a property)</td>
<td>P rdfs:range C</td>
<td>range of P (a property) is C (a class)</td>
</tr>
</tbody>
</table>
Defining classes and instances

Example: an organization wants to provide information about motor vehicles.

- The following RDF triples define classes:

  \[
  \text{ex:MotorVehicle} \quad \text{rdf:type} \quad \text{rdfs:Class}.
  \]

  \[
  \text{ex:Van} \quad \text{rdf:type} \quad \text{rdfs:Class}.
  \]

  \[
  \text{ex:Truck} \quad \text{rdf:type} \quad \text{rdfs:Class}.
  \]

- The following RDF triple defines companyCar as an instance of MotorVehicle:

  \[
  \text{ex:things:companyCar} \quad \text{rdf:type} \quad \text{ex:MotorVehicle}.
  \]

Notes:
1) we use again Turtle syntax.
2) by convention class names have an initial uppercase letter; property and instance names have an initial lowercase.
Defining classes and instances ./.

- A **class** is any resource having an `rdf:type` whose value is the resource `rdfs:Class`.
  The statement `C rdf:type rdfs:Class .` declares `C` as a class.

- An **instance** is also declared by using the `rdf:type` property.
  The statement `I rdf:type C .` declares that `I` (a resource) is an instance of `C` (a class).
  - The same statement is also interpreted as an implicit declaration that `C` is an instance of `rdfs:Class`.
    Class declarations are not mandatory; but they are useful for ontology documentation.
  - The declaration `C rdf:type rdfs:Class .` is also saying that `C` is an instance of `rdfs:Class`.
    Any class is an instance of `rdfs:Class`.

⚠️ The resource `rdfs:Class` itself is an instance of `rdfs:Class`:
  => a class may be a member of its own extension (an instance of itself).
More on RDFS classes

- Classes are identified by URIs.
- RDFS distinguishes between the class itself and its extension (the set of its instances). Two classes may have the same extension but be different.
  
  Example: *USPresident* and *WhiteHouseMainResident*

- All things described by RDFS are resources, and are instances of the class *rdfs:Resource*.
- A resource may be an instance of more than one class.
Subsumption

- Subsumption is expressed by the predefined property `rdfs:subClassOf`.
  
The statement `C1 rdfs:subClassOf C2` declares that class `C2` subsumes `C1`.

- `rdfs:subClassOf` is transitive: a class is a subclass of all its super classes. 
  This is captured by an RDFS semantic inference rule, or entailment pattern:

  \[ C \ rdfs:subClassOf \ D \ \land \ D \ rdfs:subClassOf \ E \ \Rightarrow \ C \ rdfs:subClassOf \ E \]

- Example: from:
  
  \[ ex:Van \ rdfs:subClassOf \ ex:MotorVehicle . \]
  \[ ex:MiniVan \ rdfs:subClassOf \ ex:Van . \]

  we can infer:

  \[ ex:MiniVan \ rdfs:subClassOf \ ex:MotorVehicle . \]
Class hierarchy and instances

- An instance of a class is also an instance of its super classes. Semantic inference rule:

\[
C \text{ rdfs:subClassOf } D \land I \text{ rdf:type } C \text{ entails } I \text{ rdf:type } D.
\]

- Example: from the following statements:

  \[\text{ex:Van rdfs:subClassOf ex:MotorVehicle} .\]
  \[\text{exthings:myCar rdf:type ex:Van} .\]

  RDFS allows us to infer:
  \[\text{exthings:myCar rdf:type ex:MotorVehicle} .\]
Example of hierarchy with corresponding triples

(Source: RDF primer 2004)

Strict multiple inheritance is allowed!
Defining Properties

- All properties in RDF are described as instances of class `rdf:Property`.
  The statement `P rdf:type rdf:Property` declares that P is a property:

Examples:

```
ex:weightInKg   rdf:type   rdf:Property.
ex:hasAuthor    rdf:type   rdf:Property.
```

- Properties can be used in any RDF triple:

```
ex:NameOfTheRose ex:hasAuthor ex:UmbertoEco.
```

- Declaring a property is optional. Semantic inference rule:

```
subject property object . entails property rdf:type rdf:Property .
```
Range and domain of properties

- The range of a property (the set of values it can take), is expressed by rdfs:range.
  
The statement $P \text{ rdfs:range } C$. indicates that the values of property $P$ are instances of class $C$.
  
  Example:
  
  ```
  ex:Person rdfs:type rdfs:Class.
  ex:hasAuthor rdfs:type rdfs:Property.
  ex:hasAuthor rdfs:range ex:Person.
  ```

- The domain of a property (the set of values to which it can be applied), is expressed by rdfs:domain.
  
The statement $P \text{ rdfs:domain } C$. indicates that property $P$ applies to values from class $C$.
  
  Example (added to the above triples):
  
  ```
  ex:hasAuthor rdfs:domain ex:Book.
  ```
Range and domain inference rules

- If a property $P$ has a range property, any value of $P$ must be a member of the range class:

\[
P \text{ rdfs:range } C . \land X P Y . \text{ entails } Y \text{ rdfs:type } C .
\]

- If a property $P$ has a domain property, any resource to which $P$ is applied must be a member of the domain class:

\[
P \text{ rdfs:domain } C . \land X P Y . \text{ entails } X \text{ rdfs:type } C .
\]

- Example: from the following statements:

\[
\begin{align*}
\text{ex:Book} & \text{ rdf:type } \text{ rdfs:Class .} \\
\text{ex:Person} & \text{ rdf:type } \text{ rdfs:Class .} \\
\text{ex:hasAuthor} & \text{ rdf:type } \text{ rdf:Property .} \\
\text{ex:hasAuthor} & \text{ rdfs:domain } \text{ ex:Book .} \\
\text{ex:hasAuthor} & \text{ rdfs:range } \text{ ex:Person .} \\
\text{ex:MobyDick} & \text{ ex:hasAuthor ex:Stevenson .}
\end{align*}
\]

we can deduce:

\[
\begin{align*}
\text{ex:Stevenson} & \text{ rdf:type ex:Person .} \\
\text{ex:MobyDick} & \text{ rdf:type ex:Book .}
\end{align*}
\]
Specialization of properties

- RDF Schema allows to specialize properties, using `rdfs:subPropertyOf`.

  **Example:** `ex:primaryDriver` is a specialization of `ex:driver`:

  \[
  \begin{align*}
  & \text{ex:driver} & \text{rdf:type} & \text{rdf:Property}. \\
  & \text{ex:primaryDriver} & \text{rdf:type} & \text{rdf:Property}. \\
  & \text{ex:primaryDriver} & \text{rdfs:subPropertyOf} & \text{ex:driver}. \\
  \end{align*}
  \]

- Semantic inference rule exploiting the property hierarchy:

  \[
  \text{P rdfs:subPropertyOf Q, } \land X \land P \land Y \text{ entails } X \land Q \land Y. 
  \]

  **Example:**

  - From `ex:staff:fred ex:primaryDriver ex:companyVan`.
  - We can deduce `ex:staff:fred ex:driver ex:companyVan`. 

---

55
Complex cases for range and domain of properties

⚠️ The `rdfs:range` property can be applied to itself.

\[
\text{rdfs:range } \text{rdfs:range } \text{rdfs:Class}.
\]

Any resource which is the value of `rdfs:range` property is an instance of `rdfs:Class`.

⚠️ The `rdfs:domain` property can be applied to itself.

\[
\text{rdfs:domain } \text{rdfs:domain } \text{rdf:Property}.
\]

Any resource with an `rdfs:domain` property is an instance of `rdf:Property`.

Such possibilities are expensive: they have an important impact on the complexity of the language, as we will see later.
Multiple range or domain statements

- Two range statements may apply to the same property:

  \[ P \text{ rdfs:range } C1. \]
  \[ P \text{ rdfs:range } C2. \]

  The values of property \( P \) are instances of both \( C1 \) and \( C2 \).

- Example: from

  \[ ex:hasMother \text{ rdfs:range } ex:Person. \]
  \[ ex:hasMother \text{ rdfs:range } ex:Female. \]
  \[ ex:stafffrank \text{ ex:hasMother ex:stafffrances}. \]

  we can infer that \( ex:stafffrances \) is an instance of both \( ex:Female \) and \( ex:Person \).

- A similar conclusion applies for two domain statements.
Containers

- RDF containers are resources used to represent collections (lists, sets...).

- Three classes are available with the same semantics, but a different intended use:
  - `rdf:Seq` : used to indicate that the ordering of the elements in the container is important.
  - `rdf:Alt` : used to indicate that a typical processing will only select one element of the container.

All three are subclasses of `rdfs:Container`.

- Access to specific items in a container:
  - Properties `rdf:_1`, `rdf:_2`, `rdf:_3`... are used to state that a resource is a member of a container.
  - A statement of the form `C rdf_nnn O` states that `O` is member `nnn` of container `C`. 
RDFS Semantics

The formal semantics of RDF and RDFS is defined in the W3C document [RDF11-MT]. We will again summarize the key points by referring to first order logic.

- **RDFS statements** can be interpreted by FOL formulas as indicated below:

<table>
<thead>
<tr>
<th>RDF/RDFS Statement</th>
<th>FOL Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;s P o&gt;</code></td>
<td><code>P(s, o)</code></td>
</tr>
<tr>
<td><code>i rdf:type C</code></td>
<td><code>C(i)</code></td>
</tr>
<tr>
<td>C1 rdfs:subclass C2</td>
<td><code>∀x (C1(x) → C2(x))</code></td>
</tr>
<tr>
<td>P1 rdfs:subPropertyof P2</td>
<td><code>∀x ∀y (P1(x, y) → P2(x, y))</code></td>
</tr>
<tr>
<td>P rdfs:domain C</td>
<td><code>∀x ∀y (P(x, y) → C(x))</code></td>
</tr>
<tr>
<td>P rdfs:range D</td>
<td><code>∀x ∀y (P(x, y) → D(y))</code></td>
</tr>
</tbody>
</table>
RDFS Semantics

- Entailment patterns:
  - Given a set of RDF(S) triples accepted as \textit{True}, one can deduce that other RDF(S) triples must be \textit{True}.
  - These \textit{entailment patterns}, or inference rules, can be used by a reasoner.
  - They can also be used operationally to generate new triples until it is no longer possible to do so. At this stage the RDFS graph will be called \textit{saturated}.

  The saturated graph does not change entailment and is semantically equivalent to the original one. It allows to decouple the execution of queries against the graph from the computation of entailment.

- RDF and RDFS entailment patterns have been illustrated in previous slides. A complete list is given in annex and be found in the following places:
  - \url{https://www.w3.org/TR/rdf11-mt/#patterns-of-rdf-entailment-informative}
  - \url{https://www.w3.org/TR/rdf11-mt/#patterns-of-rdfs-entailment-informative}
Revisiting the example from chapter 1

Madrid is a city.

+ :capitalOf rdfs:domain :city.
\[\Rightarrow :Madrid a :city.\]

Spain is a country.

+ :capitalOf rdfs:range :country.
\[\Rightarrow :Spain a :country.\]

Madrid is located in Spain.

+ :capitalOf rdfs:subPropertyOf :locatedIn.
\[\Rightarrow :Madrid :locatedIn :Spain.\]
Revisiting the example from chapter 1

✓ Madrid is a city.
✓ Spain is a country.
✓ Madrid is located in Spain.

✗ Barcelona is not the capital of Spain.
   Every country has exactly one capital (cardinality restrictions).

✗ Madrid is not the capital of France.
   A city is only the capital of one country (functional properties).

✗ Madrid is not a country.
   A city cannot be a country at the same time (disjoint classes).

☐ We need a more expressive language!

(example after Paulheim, Semantic Web Technologies)
More on the limitations of RDFS

- **Local scope of properties**: we cannot declare range restrictions that apply to some classes only. For example, we cannot say that *cows eat only plants*, while *other animals may eat meat*, too.

- **Disjoint classes**: some classes, for example, *male* and *female* are disjoint. In RDFS we cannot say that.

- **Boolean combinations of classes**: we may wish to build new classes using union, intersection and complement. For example, the class *person* is the disjoint union of *male* and *female*. RDFS does not allow such definitions.

- **Cardinality restrictions**: as stated before, cardinality restrictions are impossible to express in RDFS.

- **Special characteristics of properties**: Stating that a property is transitive (like *greater than*), unique (like *is mother of*), or the inverse of another property (like *eats* and *is eaten by*) is not possible in RDFS.

*(after Antoniou and Van Harmelen 2004)*
Summary

- RDF allows to represent web resources, identified by URIs. Its data structure is based on triples. A set of triples builds the equivalent of a semantic network.

- RDF has a clear model-theoretic semantics definition. It however lacks any specific constructs to represent ontologies.

- RDFS extends RDF with basic constructs such as classes, subclasses, and properties, allowing to start building simple ontologies.

- RDFS has clear semantics and a number of entailment patterns which can be used to make inferences by producing new triples.

- RDFS however also has limitations in terms of knowledge representation. We need a more expressive language to build complex ontologies.
### Annex: RDF/RDFS entailment patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>If S contains:</th>
<th>then S RDF(S)_entails:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rdfD1</strong></td>
<td>a p &quot;v&quot;^^d .</td>
<td>a p _b . _b rdf:type d .</td>
</tr>
<tr>
<td><strong>rdfD2</strong></td>
<td>a p b .</td>
<td>p rdf:type rdf:Property .</td>
</tr>
<tr>
<td><strong>rdfs1</strong></td>
<td>any IRI a in D</td>
<td>a rdf:type rdfs:Datatype .</td>
</tr>
<tr>
<td><strong>rdfs2</strong></td>
<td>p rdfs:domain x . a p b .</td>
<td>a rdf:type x .</td>
</tr>
<tr>
<td><strong>rdfs3</strong></td>
<td>p rdfs:range x . a p b .</td>
<td>b rdf:type x .</td>
</tr>
<tr>
<td><strong>rdfs4a</strong></td>
<td>a p b .</td>
<td>a rdf:type rdf:Resource .</td>
</tr>
<tr>
<td><strong>rdfs4b</strong></td>
<td>a p b .</td>
<td>b rdf:type rdf:Resource .</td>
</tr>
<tr>
<td><strong>rdfs5</strong></td>
<td>p rdfs:subPropertyOf q . q rdfs:subPropertyOf r .</td>
<td>p rdfs:subPropertyOf r .</td>
</tr>
<tr>
<td><strong>rdfs6</strong></td>
<td>a rdf:type rdf:Property .</td>
<td>a rdfs:subPropertyOf a .</td>
</tr>
<tr>
<td><strong>rdfs7</strong></td>
<td>rdfs:subPropertyOf q . a p b .</td>
<td>a q b .</td>
</tr>
<tr>
<td><strong>rdfs8</strong></td>
<td>c rdf:type rdfs:Class .</td>
<td>c rdfs:subClassOf rdf:Resource .</td>
</tr>
<tr>
<td><strong>rdfs9</strong></td>
<td>c rdfs:subClassOf d . a rdf:type c .</td>
<td>a rdf:type d .</td>
</tr>
<tr>
<td><strong>rdfs10</strong></td>
<td>c rdf:type rdfs:Class .</td>
<td>c rdfs:subClassOf c .</td>
</tr>
<tr>
<td><strong>rdfs11</strong></td>
<td>c rdfs:subClassOf d . d rdfs:subClassOf e .</td>
<td>c rdfs:subClassOf e .</td>
</tr>
<tr>
<td><strong>rdfs12</strong></td>
<td>rdf:type rdfs:ContainerMembershipProperty .</td>
<td>p rdfs:subPropertyOf rdfs:member .</td>
</tr>
<tr>
<td><strong>rdfs13</strong></td>
<td>d rdf:type rdfs:Datatype .</td>
<td>d rdfs:subClassOf rdfs:Literal .</td>
</tr>
</tbody>
</table>
References


THANK YOU