Semantic Web Systems

RDF Schema

Jacques Fleuriot

School of Informatics
In the previous lecture

- Turtle syntax

Triple in abbreviated form

@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://inf.ed.ac.uk/ont#> .
:arost dc:title “Artisan Roast” .

More Turtle abbreviations

:arost dc:title “Artisan Roast” ;
  db:locatedIn :eastEnd ;
  :rating “5”^^xsd:decimal ;
  :lovedBy :stu , :rod .
In the previous lecture

- Blank nodes

  - Artisan Roast is run by a manager whose telephone number is 0131 229 0001.

```
Blank Node Version
:aroast db:runBy _:a .
_:a pim:telno "0131 229 0001" .
_:a a db:Manager .
```

```
Alternative Blank Node Notation
:aroast db:runBy [ pim:telno "0131 229 0001" ;
 a db:Manager ] .
```
In the previous lecture

- DB Records as Triples

<table>
<thead>
<tr>
<th>ID0039</th>
<th>&quot;The Red Vineyard&quot;</th>
<th>V. Van Gogh</th>
<th>Arles</th>
</tr>
</thead>
</table>

DB2RDF Triples

`:Paintings_0039 :name "The Red Vineyard";
 :artist db:Vincent_Van_Gogh;
 :place db:Arles .`
In this lecture

- Serialising RDF in XML
- RDF Schemas (RDFS)
  - Most common constructs
  - Inference
  - Other constructs
Serialising RDF in XML
Document tree

We can represent XML documents as trees.

Example XML Document

```xml
<film>
  <title lang="en">Seven Samurai</title>
  <title lang="ja">Shichinin no samurai</title>
  <date>1954</date>
</film>
```
Document trees

- Elements are represented as nodes.
- So are Attributes and Text items.
RDF triples as XML fields

RDF Triples are encoded as rdf:Description elements.

RDF Triples with literal Object
edstaff:9888 foaf:name ‘Ewan Klein’.
RDF triples as XML trees

RDF Triples with resource Object

Element: rdf:Description
- Attr: rdf:about
  Text: "http://.../masws"

Element: dc:creator
- Attr: rdf:resource
  Text: "http://...#9888"
RDF triples as XML trees

- RDF Triples are serialised as `rdf:Description` elements.
- The Subject is the value of the `rdf:about` attribute on `rdf:Description`.
- The Predicate becomes a child element of `rdf:Description`.
- Objects:
  - Literal Objects are text content of the ‘Predicate’ element.
  - Resource Objects are values of the `rdf:resource` attribute of the ‘Predicate’ element.
  - URIs have to be written out in full (no Qnames) when they are attribute values.
RDF triples as XML trees

RDF Triples with literal Object
edstaff:9888 foaf:name ‘Ewan Klein’.

Element: rdf:Description
  Attr: rdf:about
  Element: foaf:name
    Text: "Ewan Klein"
    Text: "http://...#9888"

Linear version

<rdf:Description rdf:about="http://...#9888">
  <foaf:name>Ewan Klein</foaf:name>
</rdf:Description>
RDF triples as XML trees

RDF Triples with resource Object

```
```

**Element:** rdf:Description

**Attr:** rdf:about

**Text:** "http://.../masws"

**Element:** dc:creator

**Attr:** rdf:resource

**Text:** "http://...#9888"

**Linear version**

```
<rdf:Description rdf:about="http://...masws">
  <dc:creator rdf:resource="http://...#9888"/>
</rdf:Description>
```
Abbreviating multiple properties

RDF Triples with shared Subject

```xml
<rdf:Description rdf:about="http://...#9888">
  <foaf:name>Ewan Klein</foaf:name>
</rdf:Description>
<rdf:Description rdf:about="http://... #9888">
  <foaf:homepage rdf:resource="http://.../~ewan"/>
</rdf:Description>
```

Linear version – no abbreviation

```
edstaff:9888 foaf:name ‘Ewan Klein’.
edstaff:9888 foaf:homepage http://.../~ewan.
```
Abbreviating multiple properties

RDF Triples with shared Subject

edstaff:9888 foaf:name ‘Ewan Klein’.
edstaff:9888 foaf:homepage http://.../~ewan.

Linear version - abbreviated

<rdf:Description rdf:about="http://...#9888">
  <foaf:name>Ewan Klein</foaf:name>
  <foaf:homepage rdf:resource="http://.../~ewan"/>
</rdf:Description>
Blank nodes

XML version of blank node

```xml
<rdf:Description rdf:about="http://.../~kim">
  <dc:creator rdf:nodeID="abc"/>
</rdf:Description>
<rdf:Description rdf:nodeID="abc">
  <foaf:mailbox rdf:resource="mailto:kim@wanna.be"/>
  <foaf:name>Kim</foaf:name>
</rdf:Description>
```
Summary: RDF and XML

- XML is just a way of serialising RDF
- But think of RDF models in terms of graphs (cf. thinking of XML in terms of trees).
- Hard to avoid RDF/XML
- But knowledge of RDF/XML will not be tested in the exam. We’ll stick to Turtle syntax.
RDF recap
RDF in a nutshell

- RDF allows us to make **factual statements** (about individuals) in the form of triples: **subject-predicate-object**

- Suppose you want to talk about the 2005 record "In the Heart of the Moon" by Ali Farka Touré. You want to say that:
  - "In the Heart of the Moon" was created by Ali Farka Touré.
  - "In the Heart of the Moon" was released in 2005.

- But ambiguity is present:
  - Which Ali Farka Touré?
    - We use **URIs** to identify our resources.
  - What do you mean that it was "created by" Ali Farka Touré?
    - We use terms from **vocabularies** that define their meanings. We use URIs to identify those terms.
## RDF in a nutshell

### RDF Triples (N-triples)

```

```

### RDF Triples (Turtle)

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .

dbpedia:In_The_Heart_of_the_Moon dc:date "2005" .
```
RDFS
Ground assertions

- RDF allows us to make factual statements (assertions).
- These statements are always about individual objects.
- We can say things like Kim is a man (using `rdf:type`).
- But we can’t say things like:
  - *Giraffes are mammals.*
  - *If you are a friend of someone then you know that person.*
Missing syntactic constraints

- By itself, RDF places no restrictions on how predicates combine with subjects and objects.
- Indeed, RDF has no way of telling which URIs can semantically act as predicates.

Anomalous statements

```
meals:lunch06 terms:homepage dc:title.
mailto:kim@wanna.be edstaff:9888 'chicken'.
```
RDF has been extended with mechanisms to allow new vocabularies to be defined.

Resulting language known as RDF Schema (RDFS):
- http://www.w3.org/TR/rdf-schema/

Basic idea is to allow statements like the following:

Example RDFS Constraints

The subject of ‘birthday’ must be an Agent.
The object of ‘homepage’ must be a Document.
Every instance of Person is an instance of an Agent.
RDFS as a schema language

- RDFS also known as a *schema language*
- It helps provide meaning to RDF data.
- *meaning* ⇒ inference; i.e. you get out more than is directly asserted.
- RDFS is expressed in RDF syntax (i.e. as a set of triples).
- Cf. XML Schema vs. XML DTDs
RDF / RDFS classes (most common)

- **RDF:**
  - rdf:Resource – the class resource, everything
  - rdf:Property – the class of properties
  - rdf:Statement – the class of RDF statements

- **RDFS:**
  - rdfs:Literal – the class of literal values, e.g. strings and integers
  - rdfs:Class – the class of classes
  - rdfs:Datatype – the class of RDF datatypes
RDF / RDFS classes (most common)

- **RDF:**
  - `rdf:type` – an instance of `rdf:Property` used to state that a resource is an instance of a class
  - `rdf:value` – idiomatic property used for structured values

- **RDFS:**
  - `rdfs:subClassOf` – the subject is a subclass of a class
  - `rdfs:subPropertyOf` – the subject is a subproperty of a property
  - `rdfs:domain` – a domain of the subject property
  - `rdfs:range` – a range of the subject property
  - `rdfs:label` – a human-readable name for the subject
  - `rdfs:comment` – a description of the subject resource
  - `rdfs:seeAlso` – further information about the subject resource
Frame-based KR

- ISA and IO links from frame-based knowledge representation:
Classes and instances

Declaring Classes

```
terms:Giraffe rdf:type rdfs:Class .
terms:Herbivore rdf:type rdfs:Class .
```

Giraffe and Herbivore are classes.

Instances

```
myzoo:jerome rdf:type terms:Giraffe .
myzoo:jerome a terms:Giraffe .
```

jerome is an instance of (IO) Giraffe.

Subclasses

```
terms:Giraffe rdfs:subClassOf terms:Herbivore .
```

Giraffe is a subclass of (ISA) Herbivore.
Properties

- **Properties**
  
  terms:eats rdf:type rdf:Property.
  terms:eats a rdf:Property.

  eats is a property.

- **Domain**
  

  The subjects of eats are instances of Animals.

- **Range**
  
  terms:eats rdfs:range terms:Plant.

  The values of eats are instances of Plant.

  \[ \text{eats} : \text{Animal} \rightarrow \text{Plant} \]
Domain and range

Domain
Animal

eats

Range
Plant
Type propagation in RDFS

- $\forall x. (P(x) \land P \subseteq Q) \rightarrow Q(x)$
- Jerome is a Giraffe and Giraffes are Mammals. Therefore Jerome is a Mammal.

**Type Propagation Rule**

|----|--------------------------|
Type propagation in RDFS

Schema Statements

:Cafe rdf:type rdfs:Class .
:Restaurant rdf:type rdfs:Class .
:EatingPlace rdf:type rdfs:Class .

:Cafe rdfs:subClassOf :EatingPlace .
:Restaurant rdfs:subClassOf :EatingPlace .
# Type propagation in RDFS

### Asserted type statements

- `:ebagel rdf:type :Cafe`.
- `:aroast rdf:type :Cafe`.
- `:pyard rdf:type :Cafe`.
- `:hacraft rdf:type :Cafe`.
- `:vittoria rdf:type :Restaurant`.

### Inferred type statements

- `:ebagel rdf:type :EatingPlace`.
- `:aroast rdf:type :EatingPlace`.
- `:pyard rdf:type :EatingPlace`.
- `:hacraft rdf:type :EatingPlace`.
- `:vittoria rdf:type :EatingPlace`.
Inference support in Jena

- Jena provides support for RDFS Inference.
- But not via command line.
- See http://jena.apache.org/
Disjointedness?
Disjointedness?

Diagram:
- **EatingPlace**
  - isa: **Cafe**
  - isa: **Restaurant**
Disjointedness?

Semantic Web Systems: RDF Schema

 asserted type statements
: vittoria rdf:type :Cafe.
: vittoria rdf:type :Restaurant.
Disjointedness?

- RDFS **cannot** express the statement that two sets are disjoint.
- We need OWL for this.
- We also **cannot** infer from previous example that :Cafe and :Restaurant have a non-null intersection.
Relationship Propagation in RDFS

- $\forall R, S, x, y. (R(x, y) \land R \subseteq S) \rightarrow S(x, y)$
- Ann is a sister of Bea, and ‘sister’ is a subproperty of ‘sibling’. Therefore, Ann is sibling of Bea.
- NB subproperty can be expressed set theoretically if we regard a relation as a set of pairs:
  $$\forall \langle x,y \rangle. (\langle x,y \rangle \in R \land R \subseteq S) \rightarrow \langle x,y \rangle \in S$$

Relationship Propagation Rule

IF
\(?R \text{ rdfs:subPropertyOf } ?S\).
AND
\(?x \ ?R \ ?y\).
THEN
\(?x \ ?S \ ?y\).
Relationship Propagation in RDFS

Schema Statement

```
dbp:likes rdfs:subPropertyOf foaf:knows .
```

Asserted triple

```
```

Inferred triple

```
:bea foaf:knows :stu .
```
Domain and range

Domain: Animal

Range: Plant

Relation: eats
Domain and range typing in RDFS

Domain Typing Rule

IF
AND
THEN

Range Typing Rule

IF
AND
THEN
Relationship Propagation in RDFS

Schema statements

foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.

Asserted triple

:bea foaf:knows :stu.

Inferred triples

:bea rdf:type foaf:Person.
:stu rdf:type foaf:Person.
Relationship Propagation in RDFS

Schema statements
:hasCuisine rdfs:domain :Restaurant .
:hasCuisine rdfs:range :Cuisine .

Asserted triple
:vittoria :hasCuisine :italian .

Inferred triples
:vittoria rdf:type :Restaurant .
:italian rdf:type :Cuisine .
### Relationship Propagation in RDFS

<table>
<thead>
<tr>
<th>Schema statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>:hasCuisine rdfs:domain :Restaurant .</td>
</tr>
<tr>
<td>:hasCuisine rdfs:domain :Cafe .</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Asserted triple</th>
</tr>
</thead>
<tbody>
<tr>
<td>:witchery :hasCuisine :scottish .</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inferred triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>:witchery rdf:type :Restaurant .</td>
</tr>
<tr>
<td>:witchery rdf:type :Cafe .</td>
</tr>
</tbody>
</table>

- Multiple statements of form \( x \) rdf:type \( A \) are interpreted **conjunctively**.
- Unwanted consequence: \( :\text{witchery} \) is both a \( :\text{Restaurant} \) and a \( :\text{Cafe} \).
Relationship Propagation in RDFS

A simple mistake:

Schema statements
:hasCuisine rdfs:domain :Cuisine .
:hasCuisine rdfs:range :Restaurant .

Asserted triples
:vittoria :hasCuisine :italian .
:vittoria rdf:type :Restaurant .

RDFS doesn’t complain!

Inferred triples
:vittoria rdf:type :Cuisine .
:italian rdf:type :Restaurant .
RDFS vs Type checking

- In typed languages, domain and range are used to control syntactic well-formedness.

- If $\tau(R) = (A \rightarrow B) \rightarrow \text{Sent}$ and $\tau(x) = A$, $\tau(y) = B$ then $\tau(R(x, y)) = \text{Sent}$.

- No such constraints in RDF(S) – domain and range used for inference rather than syntactic correctness.
Combined inference in RDFS

Schema statements
:hasCuisine rdfs:domain :Restaurant.
:Restaurant rdfs:subClassOf :EatingPlace.

Asserted triple
:vittoria :hasCuisine :italian.

Inferred triple
:vittoria rdf:type :EatingPlace.

NB Contrast with inheritance in OOP languages, where e.g. Restaurant could extend behaviour of EatingPlace.
Set intersection

Schema statements
:Winebar rdfs:subClassOf :EatingPlace .
:Winebar rdfs:subClassOf :LicencedPremises .

This is equivalent to statement of the form $C \subseteq A \cap B$

Asserted triple
:maxies rdf:type :WineBar .

Inferred triples
:maxies rdf:type :EatingPlace .
:maxies rdf:type :LicencedPremises .
rdfs:label

- URIs often not readable or not informative
- rdfs:label provides a printable name for any resource
- can be used by presentation engines where available

Uninformative URI
https://projects.inf.ed.ac.uk/msc/project?number=P090

URI with Label
<hps://projects.inf.ed.ac.uk/msc/project?number=P090>
rdfs:label
“Detecting web spam using machine learning”.

Use existing triples
<hps://projects.inf.ed.ac.uk/msc/project?number=P090>
:title
“Detecting web spam using machine learning”.
:title rdfs:subPropertyOf rdfs:label.
rdfs:seeAlso

- Generally recommended that Semantic Web URIs should be de-referencable.
- But URI might just resolve to e.g. a bunch of RDF statements.
- Can use rdfs:seeAlso to point to additional (human-readable) documentation about a resource.
- Also crucial for providing links in FOAF files.

rdfs:seeAlso for FOAF

:bea foaf:knows
  [rdf:type foaf:Person;
   foaf:mbox <mailto:stu@gmail.com>;
   rdfs:seeAlso <http://example.com/~stu/foaf.rdf> ]


Summary: RDFS entailment

- All schema information is expressed as RDF triples.
- Meaning of RDFS constructs is stated in terms of inferences that can be drawn.
- Inference in RDFS based on notion of set inclusion.
- `rdfs:subClassOf` and `rdfs:subPropertyOf` can be used for data integration (cf. SWWO Ch6)
Summary: RDFS

- RDF Schema (RDFS) provides mechanisms for describing (simple) ontologies.
- RDFS build on top of RDF, using rdf:type.
- Provides
  - Class, subClassOf
  - Property, subPropertyOf
  - domain, range
- Deliberately not ‘object-oriented’:
  - Properties are defined independently, not relative to classes.
- Classes are primitives, not defined in terms of necessary and sufficient properties.
Reading

- SWWO Ch 6, 7
Task

- Consider your RDF from last task. Use RDFS to add domains and ranges. Add a few more assertions and use them to infer new facts.