Semantic-Web Ontology Languages
Part I: RDF

Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph
Informatik 2009 Lübeck

semantic-web-book.org
Outline

• A Brief Motivation
• RDF
• Simple Semantics for RDF
• RDF Schema
• Semantics for RDF(S)
Why Semantic Web Modelling?

- Initially, the Web was made for humans reading webpages.
- But there's too much information out there to be entirely checked by a human with a specific information need.
- Machines can process large amounts of data.
- Normal Web data (such as HTML) is not suitable for content-sensitive machine processing (ambiguous, relies on background knowledge, etc.)
- Semantic Web is concerned with representing information distributed across the Web in a machine-interpretable way.

- So, why not use XML?
Shortcomings of (Pure) XML

• Task: express “The Book ‘Foundations of Semantic Web Technologies’ is published by CRC Press.”

• Many options:
  
  ```xml
  <published>
    <publisher>CRC Press</publisher>
  </published>

  <publisher name="CRC Press">
    <published book="Foundations of Semantic Web Technologies/>
  </publisher>

  <book name="Foundations of Semantic Web Technologies">
    <published publisher="CRC Press”/>
  </book>

  • ambiguity and tree structure inappropriate for intended purpose
Web-Wide Linked Open Data – The Vision Becoming True

• Solution: representation by directed graphs

• “Resource Description Framework”
• W3C Recommendation (http://www.w3.org/RDF)
• RDF is a data model (not one specific syntax)
  – originally designed for providing metadata for Web resources, later used for more general purposes
  – encodes structured information
  – universal machine-readable exchange format
Building blocks for RDF Graphs

- URIs
- literals
- blank nodes (aka: empty nodes, bnodes)
• URI = Uniform Resource Identifier
• allow for denoting resources in a world-wide unambiguous way
• resources can be any object that possesses a clear identity (within the context of a given application)
• Examples: books, cities, humans, publishers, but also relations between those, abstract concepts, etc.
• already realized in some domains: e.g., ISBN for books
• Builds on concept of URLs but not every URI refers to a Web document (but often the URL of a document is used as its URI)

• URI starts with so-called URI schema separated from the following part by ":" (e.g, http, ftp, mailto)

• mostly hierarchically organized
Self-defined URIs

- necessary if no URI exists (yet) for a resource (or it is not known)
- strategy for avoiding unwanted clashes: use http URIs of webspace you control
- this also allows you to provide some documentation about the URI
- How to distinguish URI of a resource from URI of the associated documents describing it?
- Example: URI for "Othello"
  - don’t use:
    http://de.wikipedia.org/wiki/Othello
  - rather use:
    http://de.wikipedia.org/wiki/Othello#URI
• used for representing data values
• written down as strings
• interpreted via assigned *datatype*
• literals without explicitly associated datatype are treated like strings
• used to state existence of an entity the reference of which is not known
• from a logic perspective: existentially quantified variables
• there are several ways for representing graphs
• in RDF we see graphs as set of vertex-edge-vertex triples
Graphs as Triple Sets

- there are several ways for representing graphs
- in RDF we see graphs as set of vertex-edge-vertex triples

Graphs as Triple Sets

- there are several ways for representing graphs
- in RDF we see graphs as set of vertex-edge-vertex triples
RDF Triples

• constituents of an RDF triple

subject  predicate  object


• terms inspired by linguistics but doesn’t always coincide

• eligible instantiations:
  subject : URI or bnode
  predicate : URI
  objekt : URI or bnode or literal
Turtle notation:
- unabbreviated URIs in <...>
- literals in “...”
- period at the end of each triple
- extra spaces and linebreaks outside of names irrelevant

```
```
Turtle notation:
– unabbreviated URIs in <...> but can be abbreviated by namespaces
– literals in “...”
– period at the end of each triple
– extra spaces and linebreaks outside of names irrelevant

@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .
crc:uri ex:name "CRC Press" .
Turtle notation:
- unabbreviated URIs in `<...>` but can be abbreviated by namespaces
- literals in “…”
- period at the end of each triple
- extra spaces and linebreaks outside of names irrelevant

@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

book:uri  ex:publishedBy  crc:uri ;
    ex:title  "Foundations of Semantic Web Technologies" .

crc:uri  ex:name  "CRC Press" .
Turtle notation:

- unabbreviated URIs in `<...>` but can be abbreviated by namespaces
- literals in “…”
- period at the end of each triple
- extra spaces and linebreaks outside of names irrelevant

```turtle
@prefix ex: <http://example.org/> .
@prefix crc: <http://crcpress.com/> .

repeated subjects may be left out
book:uri  ex:publishedBy crc:uri ;
         ex:title             "Foundations of Semantic Web Technologies“ ;

crc:uri  ex:name         "CRC Press" .
several objects can be assigned to the same subject-predicate pairs
```
there is also an XML syntax for RDF
it’s for machines, so we don’t deal with it here

```xml
<rdf:Description rdf:about="http://semantic-web-book.org/uri">
  <ex:title>Foundations of Semantic Web Technologies</ex:title>
  <ex:publishedBy>
    <rdf:Description rdf:about="http://crcpress.com/uri">
      <ex:name>CRC Press</ex:name>
    </rdf:Description>
  </ex:publishedBy>
</rdf:Description>
```
Datatypes in RDF

• by now: literals were untyped, interpreted as strings (making e.g. "02", "2", "2.0" all different)
• typing literals with datatypes allows for more adequate (semantic = meaning-appropriate) treatment of values
• datatypes denoted by URIs and can be freely chosen
• frequently: xsd datatypes from XML
• syntax of typed literal:
  "datavalue"^^datatype-URI

• rdf:XMLLiteral is the only datatype that is part of the RDF standard
• denotes arbitrary balanced XML “snippets”
Datatypes – the Abstract View

- Example: xsd:decimal

    lexical space  
    datatype mapping  
    value space

"3.14"  
"3.140000"  
"+03.14"  
"-2.5"  
"100.00"

3,14  
-2,5  
100

"3.14" = "+03.14" holds for xsd:decimal but not for xsd:string

Datatypes in RDF – Example

• Graph:

```
http://www.w3.org/TR/rdf-primer

http://example.org/title

http://example.org/publicationDate

"2004-02-10"^^http://www.w3.org/2001/XMLSchema#date
```

• Turtle:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
<http://www.w3.org/TR/rdf-primer>
  <http://example.org/title> "RDF Primer"^^xsd:string ;
  <http://example.org/publicationDate>
  "2004-02-10"^^xsd:date .
```
Language Settings and Datatypes

- language settings only applicable to untyped literals
  
  `<http://www.w3.org/TR/rdf-primer>`
  `<http://example.org/title>`
  "Initiation à RDF"@fr, "RDF Primer"@en .

- distinct types or language settings – distinct literals
  
  `<http://crcpress.com/uri>` `<http://example.org/Name>`
  "CRC Press" ,
  "CRC Press"@en ,
  "CRC Press"^^xsd:string .
Cooking with RDF:
“For the preparation of Chutney, we need the following: 1 lb green mango, 1 tsp. Cayenne pepper, …”

<table>
<thead>
<tr>
<th>dish</th>
<th>ingredient</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>chutney</td>
<td>green mango</td>
<td>1 lb</td>
</tr>
<tr>
<td>chutney</td>
<td>cayenne pepper</td>
<td>1 tsp.</td>
</tr>
</tbody>
</table>

solved by auxiliary nodes (may be blank)
• Turtle version 1:
  @prefix ex: <http://example.org/> .
  ex:Chutney ex:hasIngredient _:id1 .
  _:id1 ex:ingredient ex:greenMango; ex:amount "1lb" .

• Turtle version 2:
  @prefix ex: <http://example.org/> .
  ex:Chutney ex:hasIngredient
  [ ex:ingredient ex:greenMango; ex:amount "1lb" ] .
• open lists (containers)
• closed lists (collections)
• reified triples
Open Lists (Container)

- Graph:
  - by `rdf:type` we assign a list type to the root node
    - `rdf:Seq` – ordered list (sequence)
    - `rdf:Bag` – unordered list
    - `rdf:Alt` – set of alternatives or choices

Closed Lists (Collections)

- **Graph:**

  ```
  ```

- **Abbreviation for Turtle:**

  ```
  ```
Reification

• How to model propositions about propositions such as:
  „The Detective supposes that the butler killed the gardener.‟
• Solution: auxiliary node for nested proposition
• RDF is focused on information exchange and interoperability
• answers of RDF tools to entailment queries should coincide
• therefore, formal semantics needed
• defined in a model-theoretic way, i.e. we start by defining interpretations
• Interpretations in RDF:

<table>
<thead>
<tr>
<th>names</th>
<th>literals</th>
<th>URIs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>untyped</td>
<td>typed</td>
</tr>
</tbody>
</table>

*Interpretations in RDF diagram:*
• when is a triple valid in an interpretation?

• a graph is valid, if all its triples are

• this settles the case for "grounded" graphs

• graph with blank nodes is valid if they can be mapped to elements such that the condition on the right is satisfied
Simple Entailment

- this model theory defines simple entailment
- this is essentially graph matching with bnodes being wildcards (more precisely: graph homomorphism)

Example: the graph

![Graph Diagram]

simply entails the graph
• RDF allows for specification of factual data

• = propositions about single resources (individuals) and their relationships

• desirable: propositions about generic groups of individuals, such as the class of publishers, of organizations, or of persons

• in database terminology: *schema knowledge*

• RDF Schema (RDFS): part of the RDF W3C recommendation
Classes and Instances


– characterizes the specific book as an instance of the (self-defined) class of textbooks

– class-membership not exclusive:


– URIs can be typed as class-identifiers:

  ex:Textbook rdf:type rdfs:Class .
Subclasses

• we want to express that every textbook is a book, e.g., that every instance of the class ex:Textbook is “automatically” an instance of the class ex:Book
• realized by rdfs:subClassOf property:

   ex:Textbook rdfs:subClassOf ex:Book .

• rdfs:subClassOf is defined to be transitive and reflexive
• rule of thumb:

   rdf:type means ∈
   rdfs:subClassOf means ⊆
Properties

- technical term for Relations, Correspondencies
- Property names usually occur in predicate position in factoid RDF triples
- characterize, how two resources are related
- mathematically: set of pairs:
  \[ \text{married\_with} = \{(Adam, Eva), (Brad, Angelina), \ldots\} \]
- URI can be marked as property name by typing it accordingly:

\[ \text{ex:publishedBy} \text{ rdf:type } \text{ rdf:Property} . \]
Subproperties

• in analogy to subclass relationships
• representation in RDFS via `rdfs:subPropertyOf` e.g.:
  ```
  ex:happilyMarriedWith rdf:subPropertyOf rdf:marriedWith .
  ```

• then, given
  ```
  ex:Markus ex:happilyMarriedWith ex:Anja .
  ```

  we can deduce
  ```
  ex:Markus ex:marriedWith ex:Anja .
  ```
• properties may give hints what types the linked resources have, e.g. we know that \texttt{ex:publishedBy} connects publications with publishers.

• i.e., for all URIs \( a, b \) where we know 
  \[ a \texttt{ ex:publishedBy } b . \]

we want to automatically follow:
  \[ a \texttt{ rdf:type } \texttt{ex:Publication} . \]
  \[ b \texttt{ rdf:type } \texttt{ex:Publisher} . \]

• this generic correspondence can be encoded in RDFS:
  \[ \texttt{ex:publishedBy rdfs:domain } \texttt{ex:Publication} . \]
  \[ \texttt{ex:publishedBy rdfs:range } \texttt{ex:Publisher} . \]
• with property restrictions, semantic interdependencies between properties and classes can be specified
• Caution: property restrictions are interpreted globally and conjunctively, e.g.

\[
\text{ex:authorOf rdfs:range ex:Cookbook . ex:authorOf rdfs:range ex:Storybook .}
\]

means: everything which is authored by somebody is both a cookbook and a storybook
• thus: always use most generic classes for domain/range statements
Additional Information

- used to add human-readable information (comments or names)
- for compatibility reasons graph-based representation recommended; set of properties for that purpose:
  - `rdfs:label` assigns an alternative name (encoded as literal) to an arbitrary resource
  - `rdfs:comment` assigns a more comprehensive comment (also literal)
  - `rdfs:seeAlso`, `rdfs:definedBy` refer to resources (URIs!) containing further information about the subject resource
RDFS Entailment

- RDFS interpretations take care of RDF(S)-specific vocabulary by imposing additional conditions on simple interpretations:
  - all URIs and bnodes are of type rdf:Resource
  - triple predicates are of type rdf:Property
  - all well-typed and untyped literals are of type rdf:Literal
  - types of triple subjects/objects correspond to rdfs:domain/rdfs:range statements
  - rdfs:subClassOf and rdfs:subPropertyOf are interpreted reflexive and transitive and “inheriting”
  - well-formed XML-Literals are mapped into LV, ill-formed ones go somewhere else
  - ...and many more
• RDFS entailment can be decided via rule-like deduction calculus (NP-complete)
other option for defining RDF(S) semantics: embedding into first order logic

2 Problems:

- FOL doesn’t provide literals/datatypes
  - can be tackled by “built-in” predicates
- straightforward translation $s \ p \ o \ . \ \Rightarrow \ p(s,o)$ does not work, as $p$ might also occur in subject or object position
  - solved by alternative translation with one ternary predicate: $s \ p \ o \ . \ \Rightarrow \ \text{triple}(s,p,o)$
• RDF graph is translated into FOL theory by introducing statement \texttt{triple(s,p,o)} for every triple \texttt{s p o}.
• for every blank node, one distinct variable is used (whereas URIs and literals are treated as constants)
• the final translation is obtained by conjunctively combining all the obtained statements and then existentially quantifying over all variables
Semantics of RDFS via Translation into FOL

- RDFS semantics can then be implemented by axiomatising the deduction calculus:

\[
\begin{align*}
\text{rdfs2:} & \quad \forall x.\forall y.\forall u.\forall v. \text{triple}(x, \text{rdfs:domain}, y) \land \text{triple}(u, x, v) \\
& \quad \quad \quad \rightarrow \text{triple}(u, \text{rdfs:type}, y) \\
\text{rdfs3:} & \quad \forall x.\forall y.\forall u.\forall v. \text{triple}(x, \text{rdfs:range}, y) \land \text{triple}(u, x, v) \\
& \quad \quad \quad \rightarrow \text{triple}(v, \text{rdfs:type}, y) \\
\text{rdfs4a, rdfs4b:} & \quad \forall x. \text{triple}(x, \text{rdfs:type}, \text{rdfs:Resource}) \\
\text{rdfs5:} & \quad \forall x.\forall y.\forall z. \text{triple}(x, \text{rdfs:subPropertyOf}, y) \\
& \quad \quad \land \text{triple}(y, \text{rdfs:subPropertyOf}, z) \\
& \quad \quad \quad \rightarrow \text{triple}(x, \text{rdfs:subPropertyOf}, z) \\
\text{rdfs6:} & \quad \forall x. \text{triple}(x, \text{rdfs:type}, \text{rdfs:Property}) \\
& \quad \quad \rightarrow \text{triple}(x, \text{rdfs:subPropertyOf}, x) \\
\text{rdfs7:} & \quad \forall x.\forall y.\forall u.\forall v. \text{triple}(x, \text{rdfs:subPropertyOf}, y) \land \text{triple}(u, x, v) \\
& \quad \quad \quad \rightarrow \text{triple}(u, y, v) \\
\text{rdfs8:} & \quad \forall x. \text{triple}(x, \text{rdfs:type}, \text{rdfs:Class}) \\
& \quad \quad \rightarrow \text{triple}(x, \text{rdfs:subClassOf}, \text{rdfs:Resource}) \\
\text{rdfs9:} & \quad \forall x.\forall y.\forall z. \text{triple}(x, \text{rdfs:subClassOf}, y) \land \text{triple}(z, \text{rdfs:type}, x) \\
& \quad \quad \rightarrow \text{triple}(z, \text{rdfs:type}, y) \\
\text{rdfs10:} & \quad \forall x. \text{triple}(x, \text{rdfs:type}, \text{rdfs:Class}) \\
& \quad \quad \rightarrow \text{triple}(x, \text{rdfs:subClassOf}, x) \\
\text{rdfs11:} & \quad \forall x.\forall y.\forall z. \text{triple}(x, \text{rdfs:subClassOf}, y) \\
& \quad \quad \land \text{triple}(y, \text{rdfs:subClassOf}, z) \\
& \quad \quad \rightarrow \text{triple}(x, \text{rdfs:subClassOf}, z) \\
\text{rdfs12:} & \quad \forall x. \text{triple}(x, \text{rdfs:type}, \text{rdfs:ContainerMembershipProperty}) \\
& \quad \quad \rightarrow \text{triple}(x, \text{rdfs:subPropertyOf}, \text{rdfs:member}) \\
\text{rdfs13:} & \quad \forall x. \text{triple}(x, \text{rdfs:type}, \text{rdfs:Datatype}) \\
& \quad \quad \rightarrow \text{triple}(x, \text{rdfs:subClassOf}, \text{rdfs:Literal})
\end{align*}
\]
• today there is a variety of RDF tools
• software libraries for virtually every programming language
• freely available systems for handling large sets of RDF data (so-called RDF stores or triple stores)
• increasingly supported by commercial actors (e.g. Oracle)
• basis for several data formats: RSS 1.0, XMP (Adobe), SVG (vector graphics format)
• RDFS language features allow for modeling certain semantic aspects of a domain of interest
• hence, RDFS can be seen as a lightweight ontology language
RDF(S) as Ontology Language?

Shortcomings of RDF(S):

• “weak” semantics:

\[
\text{ex:speaksWith} \quad \text{rdfs:domain} \quad \text{ex:Homo} . \\
\text{ex:Homo} \quad \text{rdfs:subClassOf} \quad \text{ex:Primates} .
\]

does not entail

\[
\text{ex:speaksWith} \quad \text{rdfs:domain} \quad \text{ex:Primates} .
\]

• expressivity: no negative information can be specified, no cardinality, no disjunction...
References

- W3C Specification: http://www.w3.org/RDF/
