

## Semantic Web Technologies

### RDF Schema and RDF Simple Entailment

Jos de Bruijn  
debruijn@inf.unibz.it

KRDB Research Group  
Free University of Bolzano, Italy

10 October 2007

1/42

## Outline

### RDF Schema

RDFS Vocabulary  
RDFS Metadata  
Literals and Datatypes in RDFS

### Semantics of RDF: Simple Entailment

Semantic notions  
Model Theoretic Semantics: Simple Entailment  
Evaluating Simple Entailment

2/42

## RDF Vocabulary Description Language

- ▶ Types in RDF:
  - ⟨#john,rdf:type,#Student⟩
- ▶ What is a "#Student"?
- ▶ A language for defining RDF types:
  - ▶ Define classes:
    - "#Student is a class"
  - ▶ Relationships between classes:
    - "#Student is a sub-class of #Person"
  - ▶ Properties of classes:
    - "#Person has a property hasName"
- ▶ RDF Schema is such a language

4/42

## RDF Vocabulary Description Language (cont'd)

- ▶ Classes:
  - ⟨#Student,rdf:type,#rdfs:Class⟩
- ▶ Class hierarchies:
  - ⟨#Student,rdfs:subClassOf,#Person⟩
- ▶ Properties:
  - ⟨#hasName,rdf:type,rdf:Property⟩
- ▶ Property hierarchies:
  - ⟨#hasMother,rdfs:subPropertyOf,#hasParent⟩
- ▶ Associating properties with classes (a):
  - "The property #hasName only applies to # Person:"
  - ⟨#hasName,rdfs:domain,#Person⟩
- ▶ Associating properties with classes (a):
  - "The type of the property #hasName is # xsd:string:"
  - ⟨#hasName,rdfs:range,xsd:string⟩

5/42

## Recap: RDF Vocabulary

- ▶ RDF defines a number of resources and properties
- ▶ We have already seen: rdf:XMLLiteral, rdf:type, ...
- ▶ RDF vocabulary is defined in the namespace:
  - <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- ▶ Classes:
  - rdf:Property rdf:Statement rdf:XMLLiteral rdf:Seq rdf:Bag
  - rdf:Alt rdf:List
- ▶ Properties:
  - rdf:type rdf:subject rdf:predicate rdf:object rdf:first rdf:rest
  - rdf:\_n rdf:value
- ▶ Resources:
  - rdf:nil

7/42

## RDFS Vocabulary

RDFS **Extends** the RDF Vocabulary:

RDFS vocabulary is defined in the namespace:  
<http://www.w3.org/2000/01/rdf-schema#>

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>▶ RDFS Classes           <ul style="list-style-type: none"> <li>▶ rdfs:Resource</li> <li>▶ rdfs:Class</li> <li>▶ rdfs:Literal</li> <li>▶ rdfs:Datatype</li> <li>▶ rdfs:Container</li> <li>▶ rdfs:ContainerMembershipProperty</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>▶ RDFS Properties           <ul style="list-style-type: none"> <li>▶ rdfs:domain</li> <li>▶ rdfs:range</li> <li>▶ rdfs:subPropertyOf</li> <li>▶ rdfs:subClassOf</li> <li>▶ rdfs:member</li> <li>▶ rdfs:seeAlso</li> <li>▶ rdfs:isDefinedBy</li> <li>▶ rdfs:comment</li> <li>▶ rdfs:label</li> </ul> </li> </ul> |
|--|--|

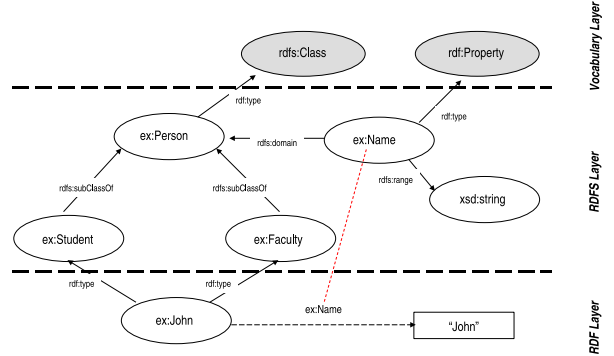
8/42

## RDFS Principles

- ▶ **Resource:** All resources are implicitly instances of `rdfs:Resource`.
- ▶ **Class:** describe sets of resources; classes are resources themselves - e.g. Webpages, people, document types
  - ▶ Class hierarchy can be defined through `rdfs:subClassOf`
  - ▶ Every class is a member of `rdfs:Class`
- ▶ **Property:** subset of RDFS Resources that are properties
  - ▶ **domain:** class associated with property, `rdfs:domain`
  - ▶ **range:** type of the property values, `rdfs:range`
  - ▶ Property hierarchy defined through `rdfs:subPropertyOf`

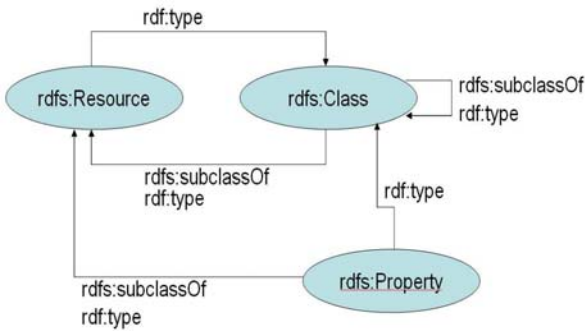
9/42

## RDFS Example



10/42

## RDFS Vocabulary Example



11/42

## RDFS Metadata Properties

Metadata is "data about data": Any meta-data can be attached to a resource, using:

- ▶ `rdfs:comment`
  - ▶ Human-readable description of the resource, e.g. `<ex:Person>, rdfs:comment, "A person is any human being"`
- ▶ `rdfs:label`
  - ▶ Human-readable version of the resource name, e.g. `<ex:Person>, rdfs:label, "Human being"`
- ▶ `rdfs:seeAlso`
  - ▶ Indicate additional information about the resource, e.g. `<ex:Person>, rdfs:seeAlso, <http://xmlns.com/wordnet/1.6/Human`
- ▶ `rdfs:isDefinedBy`
  - ▶ A special kind of `rdfs:seeAlso`, e.g. `<ex:Person>, rdfs:isDefinedBy, <http://xmlns.com/wordnet/1.6/Hu`

13/42

## Recap: Literals

- ▶ Plain literals
  - ▶ E.g. "blabla"
  - ▶ Optional language tag, e.g. "Hello, how are you?"@en-GB
- ▶ Typed literals
  - ▶ E.g. "hello"^^xsd:string, "1"^^xsd:integer
  - ▶ Recommended datatypes: XML Schema datatypes
  - ▶ Datatype mechanism extensible
  - ▶ Type checking not in RDF
- ▶ Literals may only occur as objects of a triple
- ▶ Each literal is an `rdfs:Literal`
- ▶ Each datatype is an `rdfs:Datatype`

15/42

## Literals in RDFS

- ▶ Each literal is an `rdfs:Literal`
- ▶ Say, we have: `<#john,#hasName,"John">`
- ▶ Does this mean: `<"John",rdf:type,rdfs:Literal>`  
**No! Literals may not occur as subject**
- ▶ Add: `<#john,#hasName,..:X>`  
`<.:X,rdf:type,rdfs:Literal>`

16/42

## Semantics

- ▶ RDF(S) vocabulary has built-in “meaning”
- ▶ RDF(S) Semantics
  - ▶ Makes meaning explicit
  - ▶ Defines what follows from an RDF graph
- ▶ Semantic notions
  - ▶ Subgraph
  - ▶ Instance
  - ▶ Entailment
- ▶ Recall: a graph is a **set** of triples
- ▶ Tricky part: blank nodes; are essentially existentially quantified variables

18/42

## Subgraph

$E$  is a subgraph of  $S$  if and only if it is a subset

```
<<#john>,<#hasName>,...johnsname>
<.:johnsname,<#firstName>,"John"^^xsd:string>
<.:johnsname,<#lastName>,"Smith"^^xsd:string>
```

Subgraphs:

```
<<#john>,<#hasName>,...johnsname>
<.:johnsname,<#firstName>,"John"^^xsd:string>
```

```
<.:johnsname,<#firstName>,"John"^^xsd:string>
<.:johnsname,<#lastName>,"Smith"^^xsd:string>
```

```
<<#john>,<#hasName>,...johnsname>
```

20/42

## Instance

$S'$  is an instance of  $S$  if and only if some blank nodes in  $S$  are replaced with blank nodes, literals or URIs

```
<<#john>,<#hasName>,...johnsname>
<.:johnsname,<#firstName>,"John"^^xsd:string>
<.:johnsname,<#lastName>,"Smith"^^xsd:string>
```

Instances:

```
<<#john>,<#hasName>,<#abc>>
<<#abc>,<#firstName>,"John"^^xsd:string>
<<#abc>,<#lastName>,"Smith"^^xsd:string>
```

```
<<#john>,<#hasName>,...X>
<.:X,<#firstName>,"John"^^xsd:string>
<.:X,<#lastName>,"Smith"^^xsd:string>
```

```
<<#john>,<#hasName>,...johnsname>
<.:johnsname,<#firstName>,"John"^^xsd:string>
<.:johnsname,<#lastName>,"Smith"^^xsd:string>
```

Every graph is an instance of itself!

21/42

## Entailment

- ▶  $S$  entails  $E$  if  $E$  logically follows from  $S$ 
  - ▶ Written:  $S \models E$
- ▶ A graph entails all its subgraphs
  - ▶ If  $S'$  is a subgraph of  $S$ :  $S \models S'$
- ▶ All instances of a graph  $S$  entail  $S$ 
  - ▶ If  $S''$  is an instance of  $S$ :  $S'' \models S$

22/42

## Entailment Regimes

- ▶ An entailment regime defines entailments between RDF graphs
- ▶ Formally, an entailment regime  $e$  is a binary relation between sets of RDF graphs
- ▶ In other words, given two RDF graphs  $S, E$ ,  $e$  specifies whether  $S \models_e E$  holds
- ▶ RDF defines 4 entailment regimes:
  - ▶ Simple entailment – does not interpret any RDF or RDFS vocabulary
  - ▶ RDF entailment – interprets RDF vocabulary
  - ▶ RDFS entailment – interprets RDF and RDFS vocabularies
  - ▶ D entailment – additional support for datatypes
- ▶ In this course we focus on **Simple** and **RDFS** entailment
- ▶ Today, we focus on **Simple** entailment

24/42

## Simple entailment

- ▶ Simple entailment “ignores” RDF(S) vocabulary
- ▶ Simple entailment **does** take into account blank node semantics
- ▶  $S \models_s E$  iff some instance of  $E$  is a subset of  $S$ .

Example

```
 $S = \{ \langle \#john, \text{hasName}, \text{"John Smith"} \rangle, \langle \#john, \text{marriedTo}, \dots u \rangle, \langle \dots y, \text{marriedTo}, \#john \rangle \}$ 
```

```
 $E = \{ \langle \#john, \text{hasName}, \text{"John Smith"} \rangle, \langle \dots y, \text{marriedTo}, \#john \rangle \}$ 
```

```
 $S \models_s E$ 
```

25/42

## Simple entailment (continued)

### Example

$S = \{ \langle \#john, \text{hasName}, "John\ Smith" \rangle, \langle \#john, \text{marriedTo}, \_ : u \rangle, \langle \_ : y, \text{marriedTo}, \#john \rangle \}$   
 $E = \{ \langle \#john, \text{hasName}, \_ : w \rangle, \langle \_ : x, \text{marriedTo}, \_ : z \rangle, \langle \_ : y, \text{marriedTo}, \_ : x \rangle \}$   
 $S \models_s E$

- ▶ Observe:  $S$  is an instance of  $E$

26/42

## Simple entailment (continued)

### Example

$S = \{ \langle \#john, \text{hasName}, \_ : w \rangle, \langle \_ : x, \text{marriedTo}, \_ : z \rangle, \langle \_ : y, \text{marriedTo}, \_ : x \rangle \}$   
 $E = \{ \langle \#john, \text{hasName}, "John\ Smith" \rangle, \langle \#john, \text{marriedTo}, \_ : u \rangle, \langle \_ : y, \text{marriedTo}, \#john \rangle \}$   
 $S \not\models_s E$

27/42

## Side-step: model-theoretic semantics

- ▶ Semantics of a language is defined using a model theory
- ▶ **Interpretation:**
  - ▶ Domain  $D$
  - ▶ Interpretation function  $I$ 
    - ▶ Maps symbols in the language to values in  $D$
    - ▶ Maps statements in the language to truth values in  $(D \times \dots \times D)$
- ▶ The model theory describes conditions under which:
  - ▶ The interpretation is model for a theory
- ▶ A model theory provides a **declarative** semantics for a language
  - ▶ Two important semantic notions:
    - ▶ **Entailment:** a theory  $S$  entails  $S'$  iff every model of  $S$  is a model of  $S'$
    - ▶ **Satisfiability:** a theory  $S$  is satisfiable iff  $S$  has a model

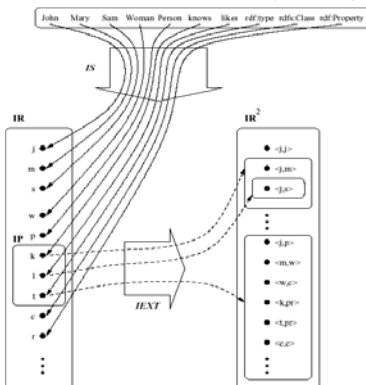
28/42

## Basic RDF Interpretations

- ▶  $I = \langle IR, IP, IS, IEXT \rangle$
- ▶ Interpretation domain for resources  $IR$ , which is a nonempty set
- ▶ Interpretation domain for properties  $IP$ , which is a set (which might be a subset of  $IR$ )
- ▶ Interpretation function  $IS$  which maps URIs to elements of  $IR$
- ▶ Extension function  $IEXT$  which maps between  $IP$  and  $IR \times IR$
- ▶ Given a URI  $u$ , we define  $I(u) = IS(u)$ , i.e. URIs are interpreted using  $IS$

29/42

## Basic RDF Interpretation (cont'd)



30/42

## Blank Node Assignments and Interpretation of Blank Nodes

- ▶ Set of blank nodes
  - ▶ in a triple  $(s,p,o)$  is denoted  $bl((s,p,o))$
  - ▶ in a graph  $S$  is denoted  $bl(S)$
- e.g.  $bl(\langle \_ : x, \text{rdf:type}, \text{ex:Student} \rangle) = \{ \_ : x \}$ ,  $bl(\langle \text{ex:john}, \text{rdf:type}, \text{ex:Student} \rangle) = \{ \_ : \}$
- ▶ A **blank node assignment**  $A$  is a mapping from blank nodes to resources:  $A : bl(S) \rightarrow IR$
- ▶ Given an interpretation  $I$ , a blank node assignment  $A$ , and a blank node  $\_ : x$ , we define  $I(\_ : x) = A(\_ : x)$ , i.e. blank nodes are interpreted using the blank node assignment  $A$

31/42

## Satisfaction of Triples and Graphs

- ▶ An interpretation  $I = \langle IR, IP, IS, IEXT \rangle$  **satisfies** a triple  $\langle s, p, o \rangle$  **relative to a blank node assignment**  $A$  if
  - ▶  $I(p) \in IP$ , i.e.  $p$  denotes a property, and
  - ▶  $\langle I(s), I(o) \rangle \in IEXT(I(p))$ , i.e.  $s, o$  denote the value of the property denoted by  $p$
 which is denoted  $I, A \models \langle s, p, o \rangle$
- ▶ An interpretation  $I = \langle IR, IP, IS, IEXT \rangle$  **satisfies** a graph  $S$  **relative to a blank node assignment**  $A$  if
  - ▶  $I, A \models \langle s, p, o \rangle$  for every triple  $\langle s, p, o \rangle$  in  $S$
 which is denoted  $I, A \models S$
- ▶ If  $I, A$  do not satisfy  $\langle s, p, o \rangle; S$ , we write  $I, A \not\models \langle s, p, o \rangle; I, A \not\models S$
- ▶  $I$  is a model of  $S$ , denoted  $I \models S$ , if there exists **some** blank node assignment  $A : b(S) \rightarrow IR$  such that  $I, A \models S$

32/42

## Adding Literals I

- ▶ Recall: there are 2 kinds of literals:
  - ▶ Plain literals, which are strings with an optional language tag
    - ▶ e.g. "John Smith", "1", "Vino Bianco"@it
  - ▶ Typed literals, which are pairs of strings and datatype URIs
    - ▶ e.g. ("John Smith", xsd:string), ("1", xsd:integer), ("2007-10-10", xsd:date)
    - ▶ We also write these as: "John Smith"8sd:string, "1"8sd:integer, "2007-10-10"8sd:date

33/42

## Adding Literals II

- ▶ Interpretation domain for resources  $IR$
- ▶ Interpretation domain for properties  $IP \subseteq IR$
- ▶ Interpretation function  $IS$  which maps URIs to elements of  $IR$
- ▶ Extension function  $IEXT$  which maps between  $IP$  and  $IR \times IR$
- ▶ Interpretation domain for plain literals  $LV \subseteq IR$
- ▶ Interpretation function  $IL$  which maps typed literals to elements of  $IR$
- ▶ Given a plain literal  $l$ , we define  $I(l) = l$ , i.e. plain literals are interpreted as themselves
- ▶ Given a typed literal  $(l, u)$ , we define  $I((l, u)) = LV((l, u))$ , i.e. typed literals are interpreted using  $LV$

34/42

## Satisfiability

- ▶ An RDF graph  $S$  is satisfiable iff  $S$  has a model

Do the following RDF graphs have a model?

```
<http://.../#john> <http://.../#hasName> "John Smith"
<http://.../#mary> <http://.../#hasName> "Mary Jane"
```

```
<http://.../#john> <http://.../#marriedTo> <http://.../#john>
```

In fact, **every RDF graph is satisfiable!**

35/42

## Simple Entailment

- ▶ " $S$  **simple-entails**  $E$ , denoted  $S \models_s E$ , if every interpretation which is a model of  $S$  is also a model of  $E$ ."
- ▶ Any subgraph  $S'$  of an RDF graph  $S$  is **entailed by**  $S$ , i.e.  $S \models_s S'$
- ▶ Any instance  $E$  of an RDF graph  $S$  **entails**  $S$ , i.e.  $E \models_s S$
- ▶ Entailment is **transitive**

```
<http://.../#john> <http://.../#hasName> "John Smith"
<http://.../#mary> <http://.../#hasName> "Mary Jane"      instance
```

```
<http://.../#john> <http://.../#hasName> ..:x
<http://.../#mary> <http://.../#hasName> "Mary Jane"      entails
```

```
<http://.../#mary> <http://.../#hasName> "Mary Jane"      subgraph
```

36/42

## Simple Entailment II

- ▶ Simple entailment seems weak, because RDF and RDFS vocabularies are not interpreted
- ▶ However, checking simple entailment is "simple"
- ▶ When viewing RDF as pure data, the RDF(S) vocabulary is arguably not "useful"
- ▶ SPARQL, the query language for RDF, uses simple entailment

38/42

## Blank nodes substitution

### Blank node substitution

A substitution is a mapping of the form

$\theta = \{b_1 \mapsto t_1, \dots, b_n \mapsto t_n\}$ , with  $b_1, \dots, b_n$  blank nodes (essentially, variables), and  $t_1, \dots, t_n$  blank nodes, URIs, or literals (essentially, terms).

Example:

$\theta = \{_:x \mapsto \#john, \_:z \mapsto \_:u, \_:y \mapsto \_:y, \_:w \mapsto \text{" John Smith"}\}$

### Substitution Application

$S = \{\langle \#john, \text{hasName}, \_:w \rangle, \langle \_:x, \text{marriedTo}, \_:z \rangle, \langle \_:y, \text{marriedTo}, \_:x \rangle\}$

$\theta(S) = \{\langle \#john, \text{hasName}, \text{" John Smith"} \rangle, \langle \#john, \text{marriedTo}, \_:u \rangle, \langle \_:y, \text{marriedTo}, \#john \rangle\}$

39/42

## Alternative Definition of Simple Entailment

- ▶  $S \models_s E$  iff  $\theta(E) \subseteq S$  for some blank node substitution  $\theta$ .
  - ▶  $\theta$  maps blank nodes in  $E$  to symbols in  $S$
- ▶ In other words,  $S \models_s E$  iff some instance of  $E$  is a subset of  $S$ .

### Example

$S = \{\langle \#john, \text{hasName}, \text{" John Smith"} \rangle, \langle \#john, \text{marriedTo}, \_:u \rangle, \langle \_:y, \text{marriedTo}, \#john \rangle\}$

$E = \{\langle \#john, \text{hasName}, \text{" John Smith"} \rangle, \langle \_:y, \text{marriedTo}, \#john \rangle\}$

$S \models_s E$

40/42

## Summary

### RDF Schema

RDFS Vocabulary

RDFS Metadata

Literals and Datatypes in RDFS

### Semantics of RDF: Simple Entailment

Semantic notions

Model Theoretic Semantics: Simple Entailment

Evaluating Simple Entailment

41/42

## Required reading

- ▶ RDF Primer: <http://www.w3.org/TR/rdf-primer/>, Chapters 5–7
- ▶ RDF Semantics, Chapters 1–2: <http://www.w3.org/TR/rdf-nt/>

## Further reading

- ▶ J. de Bruijn and S. Heymans. Logical foundations of (e)RDF(S): Complexity and reasoning. In *Proceedings of the 6th International Semantic Web Conference (ISWC2007)*.
- ▶ J. de Bruijn, E. Franconi, and S. Tessaris. Logical reconstruction of normative RDF. In *OWL: Experiences and Directions Workshop (OWLED-2005)*, Galway, Ireland, November 2005.
- ▶ H. J. ter Horst. Completeness, decidability and complexity of entailment for rdf schema and a semantic extension involving the owl vocabulary. *Journal of Web Semantics*, 3(23):79115, 2005.
- ▶ RDF Vocabulary Description Language 1.0: RDF Schema: <http://www.w3.org/TR/rdf-schema/>
- ▶ (Semantic Web Primer, Sections 3.4–3.6)

42/42