

Semantic Web Technologies

Advanced RDF and RDF Schema

Jos de Bruijn

`jos.debruijn@deri.org`

Digital Enterprise Research Institute (DERI)
University of Innsbruck, Austria

May 9, 2006

Outline

RDF Schema

- RDFS Vocabulary

- RDFS Metadata

- Literals and Datatypes in RDFS

Semantics of RDF and RDF Schema

- Semantic notions

- RDF Model Theoretic Semantics

- RDF(S) Entailment Rules

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

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⟩

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`

RDFS Vocabulary

RDFS **Extends** the RDF Vocabulary:

RDFS vocabulary is defined in the namespace:

`http://www.w3.org/2000/01/rdf-schema#`

▶ RDFS Classes

- ▶ `rdfs:Resource`
- ▶ `rdfs:Class`
- ▶ `rdfs:Literal`
- ▶ `rdfs:Datatype`
- ▶ `rdfs:Container`
- ▶ `rdfs:ContainerMembershipProperty`

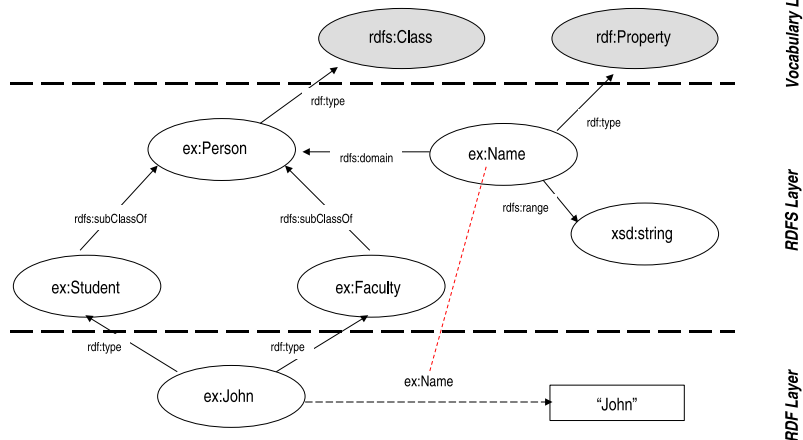
▶ RDFS Properties

- ▶ `rdfs:domain`
- ▶ `rdfs:range`
- ▶ `rdfs:subPropertyOf`
- ▶ `rdfs:subClassOf`
- ▶ `rdfs:member`
- ▶ `rdfs:seeAlso`
- ▶ `rdfs:isDefinedBy`
- ▶ `rdfs:comment`
- ▶ `rdfs:label`

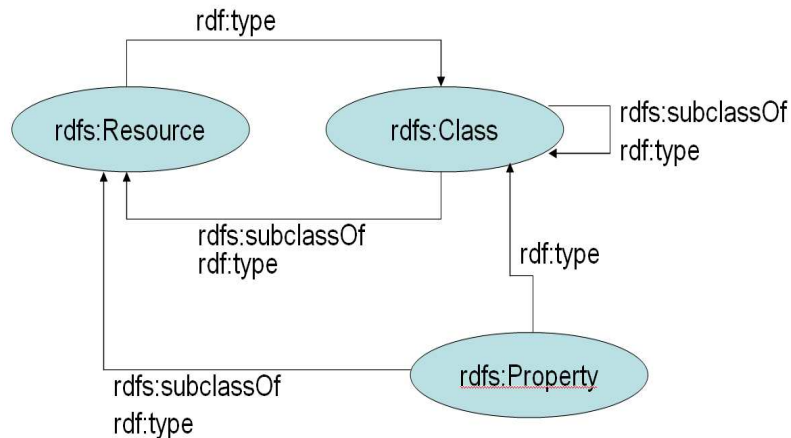
RDFS Principles

- ▶ **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`

RDFS Example



RDFS Vocabulary Example



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/Human>>`

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

Literals in RDFS

- ▶ Each literal is an `rdfs:Literal`
- ▶ Say, we have: $\langle \#john, \#hasName, "John" \rangle$
- ▶ Does this mean: $\langle "John", rdf:type, rdfs:Literal \rangle$
No! Literals may not occur as subject
- ▶ Add: $\langle \#john, \#hasName, _ :X \rangle$
 $\langle _ :X, rdf:type, rdfs:Literal \rangle$

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

Subgraph

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

$\langle \langle \#john \rangle, \langle \#hasName \rangle, _ :johnsname \rangle$
 $\langle _ :johnsname, \langle \#firstName \rangle, "John" \wedge \wedge xsd:string \rangle$
 $\langle _ :johnsname, \langle \#lastName \rangle, "Smith" \wedge \wedge xsd:string \rangle$

Subgraphs:

$\langle \langle \#john \rangle, \langle \#hasName \rangle, _ :johnsname \rangle$
 $\langle _ :johnsname, \langle \#firstName \rangle, "John" \wedge \wedge xsd:string \rangle$
 $\langle _ :johnsname, \langle \#firstName \rangle, "John" \wedge \wedge xsd:string \rangle$
 $\langle _ :johnsname, \langle \#lastName \rangle, "Smith" \wedge \wedge xsd:string \rangle$
 $\langle \langle \#john \rangle, \langle \#hasName \rangle, _ :johnsname \rangle$

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!

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 : $S'' \models S$

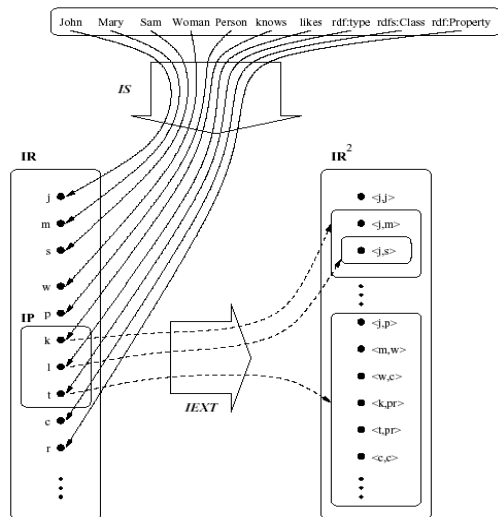
Side-step: model-theoretic semantics

- ▶ Semantics of a language is defined using a model theory
- ▶ **Interpretation**:
 - ▶ Domain D
 - ▶ Interpretation function \cdot^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

Basic RDF Interpretation

- ▶ Interpretation domain for resources IR
- ▶ Interpretation domain for properties $IP \subseteq IR$
- ▶ Interpretation function IS which maps URIs and blank nodes to elements of IR
- ▶ Extension function $IEXT$ which maps between IP and $IR \times IR$

Basic RDF Interpretation (cont'd)



RDF Satisfiability

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

Do the following RDF graphs have a model?

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

<code><http://.../#john></code>	<code><http://.../#marriedTo></code>	<code><http://.../#john></code>
---------------------------------------	--	---------------------------------------

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

RDF Entailment

- ▶ “ S **rdf-entails** E if every rdf-interpretation which satisfies every member of S also satisfies E .”
- ▶ Any subgraph S' of an RDF graph S **is entailed by** S
- ▶ Any instance E of an RDF graph S **entails** S
- ▶ Entailment is **transitive**

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

instance

entails

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

entails

<code><http://.../#mary></code>	<code><http://.../#hasName></code>	<code>"Mary Jane"</code>
---------------------------------------	--	--------------------------

subgraph

Adding Literals

- ▶ 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

RDF Semantic Conditions

Every RDF interpretation must satisfy these conditions:

1. x is in IP if and only if $\langle x, I(rdf : Property) \rangle$ is in $IEXT(I(rdf : type))$
 Informally: $\langle x, rdf:type, rdf:Property \rangle$
2. If $"xxx" \hat{\hat{}} rdf:XMLLiteral$ is in the vocabulary and xxx is a **well-typed** XML literal string, then
 $IL("xxx" \hat{\hat{}} rdf : XMLLiteral)$ is the XML value of xxx ;
 $IL("xxx" \hat{\hat{}} rdf : XMLLiteral)$ is in LV ;
 $IEXT(I(rdf : type))$ contains $\langle IL("xxx" \hat{\hat{}} rdf : XMLLiteral), I(rdf : XMLLiteral) \rangle$
 Informally: $\langle "xxx" \hat{\hat{}} rdf:XMLLiteral, rdf:type, rdf:XMLLiteral \rangle$
3. If $"xxx" \hat{\hat{}} rdf:XMLLiteral$ is in the vocabulary and xxx is an **ill-typed** XML literal string, then
 $IL("xxx" \hat{\hat{}} rdf : XMLLiteral)$ is not in LV ;
 $IEXT(I(rdf : type))$ does not contain $\langle IL("xxx" \hat{\hat{}} rdf : XMLLiteral), I(rdf : XMLLiteral) \rangle$
 Informally: $\langle \cancel{"xxx" \hat{\hat{}} rdf:XMLLiteral}, rdf:type, rdf:XMLLiteral \rangle$

RDFS entailment

```
<http://example.org/#john> rdf:type <http://example.org/#Student>  
<http://example.org/#Student> rdfs:subClassOf <http://example.org/#Person>
```

entails

```
<http://example.org/#john> rdf:type <http://example.org/#Person>
```

```
<http://example.org/#hasName> rdfs:domain <http://example.org/#Student>  
<http://example.org/#mary> <http://example.org/#hasName> "Mary"
```

entails

```
<http://example.org/#mary> rdf:type <http://example.org/#Student>
```

```
<http://example.org/#john> <http://example.org/#hasMother> <http://example.org/#mary>  
<http://example.org/#hasMother> rdfs:subPropertyOf <http://example.org/#hasParent>
```

entails

```
<http://example.org/#john> <http://example.org/#hasParent> <http://example.org/#mary>
```

Entailment rules

- ▶ Semantics defined through **entailment rules**
- ▶ IF S contains \langle **triple pattern** \rangle then add \langle **triple pattern** \rangle
- ▶ Executing all entailment rules yields **realization** of S
- ▶ S entails E if E is a subgraph of the realization of S (modulo blank node renaming)
- ▶ Axiomatic triple are always added

RDF Axiomatic triples

⟨rdf:type, rdf:type, rdf:Property⟩
⟨rdf:subject, rdf:type, rdf:Property⟩
⟨rdf:predicate, rdf:type, rdf:Property⟩
⟨rdf:object, rdf:type, rdf:Property⟩
⟨rdf:first, rdf:type, rdf:Property⟩
⟨rdf:rest, rdf:type, rdf:Property⟩
⟨rdf:value, rdf:type, rdf:Property⟩
⟨rdf:_1, rdf:type, rdf:Property⟩
⟨rdf:_2, rdf:type, rdf:Property⟩
...
⟨rdf:nil, rdf:type, rdf:List⟩

RDFS Axiomatic triples I

- ⟨rdf:type, rdfs:domain, rdfs:Resource⟩
- ⟨rdfs:domain, rdfs:domain, rdf:Property⟩
- ⟨rdfs:range, rdfs:domain, rdf:Property⟩
- ⟨rdfs:subPropertyOf, rdfs:domain, rdf:Property⟩
- ⟨rdfs:subClassOf, rdfs:domain, rdfs:Class⟩
- ⟨rdf:subject, rdfs:domain, rdf:Statement⟩
- ⟨rdf:predicate, rdfs:domain, rdf:Statement⟩
- ⟨rdf:object, rdfs:domain, rdf:Statement⟩
- ⟨rdfs:member, rdfs:domain, rdfs:Resource⟩
- ⟨rdf:first, rdfs:domain, rdf:List⟩
- ⟨rdf:rest, rdfs:domain, rdf:List⟩
- ⟨rdfs:seeAlso, rdfs:domain, rdfs:Resource⟩
- ⟨rdfs:isDefinedBy, rdfs:domain, rdfs:Resource⟩
- ⟨rdfs:comment, rdfs:domain, rdfs:Resource⟩
- ⟨rdfs:label, rdfs:domain, rdfs:Resource⟩

RDFS Axiomatic triples II

- ⟨rdf:value, rdfs:domain, rdfs:Resource⟩
- ⟨rdf:type, rdfs:range, rdfs:Class⟩
- ⟨rdfs:domain, rdfs:range, rdfs:Class⟩
- ⟨rdfs:range, rdfs:range, rdfs:Class⟩
- ⟨rdfs:subPropertyOf, rdfs:range, rdf:Property⟩
- ⟨rdfs:subClassOf, rdfs:range, rdfs:Class⟩
- ⟨rdf:subject, rdfs:range, rdfs:Resource⟩
- ⟨rdf:predicate, rdfs:range, rdfs:Resource⟩
- ⟨rdf:object, rdfs:range, rdfs:Resource⟩
- ⟨rdfs:member, rdfs:range, rdfs:Resource⟩
- ⟨rdf:first, rdfs:range, rdfs:Resource⟩
- ⟨rdf:rest, rdfs:range, rdf:List⟩
- ⟨rdfs:seeAlso, rdfs:range, rdfs:Resource⟩
- ⟨rdfs:isDefinedBy, rdfs:range, rdfs:Resource⟩
- ⟨rdfs:comment, rdfs:range, rdfs:Literal⟩

RDFS Axiomatic triples III

⟨rdfs:label, rdfs:range, rdfs:Literal⟩

⟨rdf:value, rdfs:range, rdfs:Resource⟩

⟨rdf:Alt, rdfs:subClassOf, rdfs:Container⟩

⟨rdf:Bag, rdfs:subClassOf, rdfs:Container⟩

⟨rdf:Seq, rdfs:subClassOf, rdfs:Container⟩

⟨rdfs:ContainerMembershipProperty, rdfs:subClassOf, rdf:Property⟩

⟨rdfs:isDefinedBy, rdfs:subPropertyOf, rdfs:seeAlso⟩

⟨rdf:XMLLiteral, rdf:type, rdfs:Datatype⟩

⟨rdf:XMLLiteral, rdfs:subClassOf, rdfs:Literal⟩

⟨rdfs:Datatype, rdfs:subClassOf, rdfs:Class⟩

⟨rdf:_1, rdf:type, rdfs:ContainerMembershipProperty⟩

⟨rdf:_1, rdfs:domain, rdfs:Resource⟩

⟨rdf:_1, rdfs:range, rdfs:Resource⟩

RDFS Axiomatic triples IV

$\langle \text{rdf:}_2, \text{rdf:type}, \text{rdfs:ContainerMembershipProperty} \rangle$

$\langle \text{rdf:}_2, \text{rdfs:domain}, \text{rdfs:Resource} \rangle$

$\langle \text{rdf:}_2, \text{rdfs:range}, \text{rdfs:Resource} \rangle$

...

RDF Entailment

if E contains $\langle A, B, C \rangle$ then add $\langle B, \text{rdf:type}, \text{rdf:Property} \rangle$

if E contains $\langle A, B, I \rangle$ (I is a valid XML literal) then add $\langle _ : X, \text{rdf:type}, \text{rdf:XMLLiteral} \rangle$

RDFS Entailment I

everything in the subject is a resource:

if E contains $\langle A, B, C \rangle$ then add $\langle A, \text{rdf:type}, \text{rdfs:Resource} \rangle$

every non-literal in the object is a resource:

if E contains $\langle A, B, C \rangle$ (C is not a literal) then add $\langle C, \text{rdf:type}, \text{rdfs:Resource} \rangle$

every class is subclass of `rdfs:Resource`:

if E contains $\langle A, \text{rdf:type}, \text{rdfs:Class} \rangle$ then add $\langle A, \text{rdfs:subClassOf}, \text{rdfs:Resource} \rangle$

inheritance: **if E contains $\langle A, \text{rdf:type}, B \rangle, \langle B, \text{rdfs:subClassOf}, C \rangle$ then add $\langle A, \text{rdf:type}, C \rangle$**

`rdfs:subClassOf` is transitive:

if E contains $\langle A, \text{rdfs:subClassOf}, B \rangle, \langle B, \text{rdfs:subClassOf}, C \rangle$ then add $\langle A, \text{rdfs:subClassOf}, C \rangle$

RDFS Entailment II

rdfs:subClassOf is reflexive:

if E **contains** $\langle A, \text{rdf:type}, \text{rdfs:Class} \rangle$ **then add** $\langle A, \text{rdfs:subClassOf}, A \rangle$

rdfs:subPropertyOf is transitive:

if E **contains** $\langle A, \text{rdfs:subPropertyOf}, B \rangle$, $\langle B, \text{rdfs:subPropertyOf}, C \rangle$ **then add** $\langle A, \text{rdfs:subPropertyOf}, C \rangle$

rdfs:subPropertyOf is reflexive:

if E **contains** $\langle P, \text{rdf:type}, \text{rdf:Property} \rangle$ **then add** $\langle P, \text{rdfs:subPropertyOf}, P \rangle$

domain of properties:

if E **contains** $\langle P, \text{rdfs:domain}, C \rangle$, $\langle A, P, B \rangle$ **then add** $\langle A, \text{rdf:type}, C \rangle$

range of properties:

if E **contains** $\langle P, \text{rdfs:range}, C \rangle$, $\langle A, P, B \rangle$ **then add** $\langle B, \text{rdf:type}, C \rangle$

RDFS Entailment III

every literal is a member of rdfs:Literal:

if E **contains** $\langle A, B, I \rangle$ (I is a plain literal) **then add**
 $\langle _ : X, \text{rdf:type}, \text{rdfs:Literal} \rangle$

every datatype is subclass of rdfs:Literal:

if E **contains** $\langle A, \text{rdf:type}, \text{rdfs:Datatype} \rangle$ **then add**
 $\langle A, \text{rdfs:subClassOf}, \text{rdfs:Literal} \rangle$

More on literals

Recall:

if E contains $\langle A, B, I \rangle$ (I is a valid XML literal) then add $\langle _ : X, \text{rdf:type}, \text{rdf:XMLLiteral} \rangle$

every literal is a member of `rdfs:Literal`:

if E contains $\langle A, B, I \rangle$ (I is a plain literal) then add $\langle _ : X, \text{rdf:type}, \text{rdfs:Literal} \rangle$

allocating blank nodes to literals:

if E contains $\langle A, B, I \rangle$ (I is a literal) then add $\langle A, B, _ : n \rangle$
 $_ : n$ is **allocated to** I

“dereferencing” blank nodes:

if E contains $\langle A, B, _ : n \rangle$ ($_ : n$ is allocated to a literal I) then add $\langle A, B, I \rangle$

Summary

RDF Schema

- RDFS Vocabulary

- RDFS Metadata

- Literals and Datatypes in RDFS

Semantics of RDF and RDF Schema

- Semantic notions

- RDF Model Theoretic Semantics

- RDF(S) Entailment Rules

Required reading

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

Further reading

- ▶ 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.
- ▶ RDF Vocabulary Description Language 1.0: RDF Schema: <http://www.w3.org/TR/rdf-schema/>
- ▶ RDF Semantics: <http://www.w3.org/TR/rdf-mt/>
- ▶ (Semantic Web Primer, Sections 3.4–3.6)