

Unit 3 – Beyond Simple Entailment: Semantics of the `rdf:` & `rdfs:` vocabularies, Datatypes, and the `owl:` vocabulary

Axel Polleres

Siemens AG Österreich

VU 184.729 Semantic Web Technologies

Unit Outline

1. Assignment 1 – Some words on common mistakes
2. Some words on the usage of blank nodes on the Web
3. RDF/S Entailment: semantics of the `rdf:` and `rdfs:` vocabularies
4. D-Entailment: Giving Semantics to Datatypes
5. OWL Entailment: Giving Semantics to the `owl:` Vocabulary

Assignment 1 – Some words on common mistakes 1/3

ATTENTION: If you have NOT received a confirmation email from me, please re-send the assignment!

For future assignments:

- please only use this email address:
axel.polleres@siemens.com
- please use the following email-subject:
Semantic Web Technologies: Assignment 2

Assignment 1 – Some words on common mistakes 2/3

FOAF:

- **Use URIs when you have them...**

```
:me foaf:knows _:b1 .
```

```
_:b1 a foaf:Person;
```

```
    rdfs:seeAlso <http://www.polleres.net/foaf.rdf> .
```

Assignment 1 – Some words on common mistakes 2/3

FOAF:

- Use URIs when you have them...

```
:me foaf:knows _:b1 .  
_:b1 a foaf:Person;  
      rdfs:seeAlso <http://www.polleres.net/foaf.rdf> .
```

Better:

```
:me foaf:knows <http://www.polleres.net/foaf.rdf#me> .
```

Assignment 1 – Some words on common mistakes 2/3

FOAF:

- Use URIs when you have them...

```
:me foaf:knows _:b1 .  
_:b1 a foaf:Person;  
      rdfs:seeAlso <http://www.polleres.net/foaf.rdf> .
```

Better:

```
:me foaf:knows <http://www.polleres.net/foaf.rdf#me> .
```

- ... but don't use URIs of **Webpages** for persons:

```
:me foaf:knows <http://www.pascal-hitzler.de> .
```

Assignment 1 – Some words on common mistakes 2/3

FOAF:

- Use URIs when you have them...

```
:me foaf:knows _:b1 .
_:b1 a foaf:Person;
      rdfs:seeAlso <http://www.polleres.net/foaf.rdf> .
```

Better:

```
:me foaf:knows <http://www.polleres.net/foaf.rdf#me> .
```

- ... but don't use URIs of **Webpages** for persons:

```
:me foaf:knows <http://www.pascal-hitzler.de> .
```

Better:

```
:me foaf:knows _:b1 .
_:b1 foaf:homepage <http://www.pascal-hitzler.de> .
```

Assignment 1 – Some words on common mistakes 3/3

FOAF, other problems

- the URI of your foaf file is not derefernceable (except in a browser): e.g. docs.google.com (you could use sites.google.com instead.
- HTTP 403 Forbidden responses, probably a Web server problem.

SPARQL queries were generally quite ok (you can still send updates for assignment 1 to me).

Generally: please consider my feedback, where possible.

Some words on the usage of blank nodes

We did some analysis on the usage of blank nodes among Semantic Web experts [Mallea et al., 2011] here the surprising result:

Question 1: When would you publish the triple ":John :telephone _:b ." alone?			
OPTION		RESPONSES	
		ALL (88)	LOD (10)
1a	<i>John has a tel. number whose value is unknown.</i>	46.4%	20.0%
1b	<i>John has a tel. number but its value is hidden, e.g., for privacy.</i>	23.9%	0.0%
1c	<i>John has no tel. number.</i>	0.0%	0.0%
1d	<i>John may or may not have a tel. number.</i>	2.3%	0.0%
1e	<i>John's number should not be externally referenced.</i>	18.2%	0.0%
1f	<i>I do not want to mint a URI for the tel. number.</i>	37.5%	30.0%
1g	<i>I would not publish such a triple.</i>	41.0%	70.0%

Question 2: When would you publish the triples ":John :telephone _:b1, _:b2 ." alone?			
OPTION		RESPONSES	
		ALL (88)	LOD (10)
2a	<i>John does not have a tel. number.</i>	0.0%	0.0%
2b	<i>John may not have a tel. number.</i>	0.0%	0.0%
2c	<i>John has at least one tel. number.</i>	23.9%	0.0%
2d	<i>John has two different tel. numbers.</i>	23.9%	10.0%
2e	<i>John has at least two different tel. numbers.</i>	35.2%	40.0%
2f	<i>I would not publish such triples.</i>	50.0%	70.0%

Table 4: Details and results of the survey of two mailing lists (public-lod@w3.org and semantic-web@w3.org) (non-)leanness study on BTC data... ongoing

Unit Outline

1. Assignment 1 – Some words on common mistakes
2. Some words on the usage of blank nodes on the Web
3. RDF/S Entailment: semantics of the `rdf:` and `rdfs:` vocabularies
4. D-Entailment: Giving Semantics to Datatypes
5. OWL Entailment: Giving Semantics to the `owl:` Vocabulary

In the following, we use the following namespace prefixes:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
@prefix owl: <http://www.w3.org/2002/07/owl#> .
```

The `rdf:` vocabulary

The **RDF vocabulary**, `rdfV`, is a set of URI references in the `rdf:` namespace:

- `rdf:type` ... expresses class membership
- `rdf:Property` ... the class of properties
- `rdf:XMLLiteral` ... the datatype of XML Literals.
- `rdf:first`, `rdf:rest`, `rdf:nil`, `rdf:List` ... vocabulary to write lists (called “collections”) in RDF
- `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf:_1`, `rdf:_2` ... vocabulary to describe (ordered, unordered, alternate) containers of values.
- `rdf:Statement`, `rdf:subject`, `rdf:predicate`, `rdf:object` ... vocabulary to make statements about statement (aka “reification”)
- `rdf:value` ... to describe structured values

The `rdf:` vocabulary

The **RDF vocabulary**, `rdfV`, is a set of URI references in the `rdf:` namespace:

- `rdf:type` ... expresses class membership
- `rdf:Property` ... the class of properties
- `rdf:XMLLiteral` ... the datatype of XML Literals.
- `rdf:first`, `rdf:rest`, `rdf:nil`, `rdf:List` ... vocabulary to write lists (called “collections”) in RDF
- `rdf:Seq`, `rdf:Bag`, `rdf:Alt`, `rdf:_1`, `rdf:_2` ... vocabulary to describe (ordered, unordered, alternate) containers of values.
- `rdf:Statement`, `rdf:subject`, `rdf:predicate`, `rdf:object` ... vocabulary to make statements about statement (aka “reification”)
- `rdf:value` ... to describe structured values

Note: The first three are the most important for the formal semantics. The semantics doesn't make any strong restrictions on the grayed out vocabulary.

Collection vocabulary – Example

Collection vocabulary intended to be used to write lists, e.g.

```
ex:article ex:hasAuthors ( ex:johnny ex:jim ex:jack )
```

Collection vocabulary – Example

Collection vocabulary intended to be used to write lists, e.g.

```
ex:article ex:hasAuthors _:b1.  
_:b1 rdf:first ex:johnny ; rdf:rest _:b2.  
_:b2 rdf:first ex:jim ; rdf:rest _:b3.  
_:b3 rdf:first ex:jack ; rdf:rest rdf:nil.
```

Comes usually from RDF/XML's `parseType = "Collection"`.

Collection vocabulary – Example

Collection vocabulary intended to be used to write lists, e.g.

```
ex:article ex:hasAuthors _:b1.  
_:b1 rdf:first ex:johnny ; rdf:rest _:b2.  
_:b2 rdf:first ex:jim ; rdf:rest _:b3.  
_:b3 rdf:first ex:jack ; rdf:rest rdf:nil.
```

Comes usually from RDF/XML's `parseType = "Collection"`.

However, the semantics does not impose strong formal restrictions/conditions on the use of this vocabulary, e.g. a graph

```
ex:a rdf:first ex:a.
```

is fine (no restrictions on cyclic, incomplete, etc. lists.)

Container vocabulary – Example

Container vocabulary intended to be used to write containers of values,
e.g.

```
ex:semWebCourse ex:hasParticipants [ a rdf:Bag;  
                                     rdf:_1 ex:johnny,  
                                     rdf:_2 ex:jim;  
                                     rdf:_3 ex:jack ] .
```

```
ex:semWebCourse ex:hasLecturer [ a rdf:Alt;  
                                  rdf:_1 ex:prof1  
                                  rdf:_2 ex:prof2 ] .
```

Comes usually from RDF/XML's `...`.

Container vocabulary – Example

Container vocabulary intended to be used to write containers of values, e.g.

```
ex:semWebCourse ex:hasParticipants [ a rdf:Bag;  
                                     rdf:_1 ex:johny,  
                                     rdf:_2 ex:jim;  
                                     rdf:_3 ex:jack ] .
```

```
ex:semWebCourse ex:hasLecturer [ a rdf:Alt;  
                                  rdf:_1 ex:prof1  
                                  rdf:_2 ex:prof2 ] .
```

Comes usually from RDF/XML's `...`.

However, the semantics does not impose strong formal restrictions/conditions on the use of this vocabulary, e.g. a graph.

```
ex:a rdf:_4 rdf:Seq;  
     rdf:_3 rdf:_3 .
```

is fine (e.g. no restrictions on multiple occurrences of a numbered membership property, etc.).

Reification vocabulary & rdf:value – Example

Reification vocab intended to be used to write statements about statements, e.g.

```
ex:axel ex:thinks [ rdf:subject dbpedia:Web_Ontology_Language;  
                   rdf:predicate owl:differentFrom  
                   rdf:object dbpedia:Description_Logics] .
```

Again, no semantics restrictions on the use of this vocabulary.

Reification vocabulary & rdf:value – Example

Reification vocab intended to be used to write statements about statements, e.g.

```
ex:axel ex:thinks [ rdf:subject dbpedia:Web_Ontology_Language;  
                    rdf:predicate owl:differentFrom  
                    rdf:object dbpedia:Description_Logics] .
```

Again, no semantics restrictions on the use of this vocabulary.

rdf:value intended to be used to denote “structured values”

```
ex:telephone1 ex:hasPrice [ rdf:value 20; ex:currency ‘EUR’]
```

Again, no semantics restrictions on the use of this vocabulary.

Reification vocabulary & rdf:value – Example

Reification vocab intended to be used to write statements about statements, e.g.

```
ex:axel ex:thinks [ rdf:subject dbpedia:Web_Ontology_Language;  
                  rdf:predicate owl:differentFrom  
                  rdf:object dbpedia:Description_Logics] .
```

Again, no semantics restrictions on the use of this vocabulary.

rdf:value intended to be used to denote “structured values”

```
ex:telephone1 ex:hasPrice [ rdf:value 20; ex:currency ‘EUR’]
```

Again, no semantics restrictions on the use of this vocabulary.

Note on RDF1.1

Intuitively, all these parts of the vocabulary have been left without fixed semantics for the moment... could be changed/replaced by later standards, e.g. “Reification-Entailment”, but doesn’t seem to be a priority cf. also <http://www.w3.org/2011/rdf-wg/track/issues/25>

... now to the formal part.

Recall from last time (lecture 2, slide 20):

Simple Interpretation

A **simple interpretation** I over vocabulary V is a 6-tuple
 $I = \langle IR, IP, IEXT, IS, IL, LV \rangle$

rdf-interpretation

An **rdf-interpretation** is a simple interpretation

$I = \langle IR, IP, IEXT, IS, IL, LV \rangle$ which fulfills the following conditions and all the following set of **axiomatic triples** A_{rdf} :

RDF semantic conditions.

x is in IP if and only if $\langle x, I(\text{rdf:Property}) \rangle$ is in $IEXT(I(\text{rdf:type}))$

If $"xxx" \wedge \text{rdf:XMLLiteral}$ is in V and xxx is a well-typed XML literal string, then

$IL("xxx" \wedge \text{rdf:XMLLiteral})$ is the XML value of xxx ;
 $IL("xxx" \wedge \text{rdf:XMLLiteral})$ is in LV ;
 $IEXT(I(\text{rdf:type}))$ contains $\langle IL("xxx" \wedge \text{rdf:XMLLiteral}), I(\text{rdf:XMLLiteral}) \rangle$

If $"xxx" \wedge \text{rdf:XMLLiteral}$ is in V and xxx is an ill-typed XML literal string, then

$IL("xxx" \wedge \text{rdf:XMLLiteral})$ is not in LV ;
 $IEXT(I(\text{rdf:type}))$ does not contain $\langle IL("xxx" \wedge \text{rdf:XMLLiteral}), I(\text{rdf:XMLLiteral}) \rangle$

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 .
  
```

rdf-interpretation

An **rdf-interpretation** is a simple interpretation

$I = \langle IR, IP, IEXT, IS, IL, LV \rangle$ which fulfills the following conditions and all the following set of **axiomatic triples** A_{rdf} :

RDF semantic conditions.

x is in IP if and only if $\langle x, I(\text{rdf:Property}) \rangle$ is in $IEXT(I(\text{rdf:type}))$

If " xxx "^{^^rdf:XMLLiteral} is in V and xxx is a well-typed XML literal string, then

$IL("xxx"$ ^{^^rdf:XMLLiteral}) is the XML value of xxx ;
 $IL("xxx"$ ^{^^rdf:XMLLiteral}) is in LV ;
 $IEXT(I(\text{rdf:type}))$ contains $\langle IL("xxx"$ ^{^^rdf:XMLLiteral}), $I(\text{rdf:XMLLiteral}) \rangle$

If " xxx "^{^^rdf:XMLLiteral} is in V and xxx is an ill-typed XML literal string, then

$IL("xxx"$ ^{^^rdf:XMLLiteral}) is not in LV ;
 $IEXT(I(\text{rdf:type}))$ does not contain $\langle IL("xxx"$ ^{^^rdf:XMLLiteral}), $I(\text{rdf:XMLLiteral}) \rangle$

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 .
  
```

rdf-satisfaction, entailment

written \models_{rdf} defined analogously to \models , but with respect to rdf-interpretations.

rdf-interpretation

Some consequences:

Proposition 1

If $s \ p \ o. \in G$ then $G \models_{rdf} \{p \text{ a } rdf:Property\}$.

rdf-interpretation

Some consequences:

Proposition 1

If $s \ p \ o. \in G$ then $G \models_{rdf} \{p \text{ a } rdf:Property\}$.

Particularly, note that the first axiomatic triple is redundant!

rdf-interpretation

Some consequences:

Proposition 1

If $s \text{ p } o. \in G$ then $G \models_{rdf} \{p \text{ a } rdf:Property\}$.

Particularly, note that the first axiomatic triple is redundant!

Let L_{XML} denote the set of well-formed XML literals.

Proposition 2

If $s \text{ p } "xyz" \wedge rdf:XMLLiteral. \in G$, such that $xyz \in L_{XML}$ then $G \models_{rdf} \{[] \text{ a } rdf:XMLLiteral\}$.

rdf-interpretation

Some consequences:

Proposition 1

If $s \text{ p } o. \in G$ then $G \models_{rdf} \{p \text{ a } rdf:Property\}$.

Particularly, note that the first axiomatic triple is redundant!

Let L_{XML} denote the set of well-formed XML literals.

Proposition 2

If $s \text{ p } "xyz" \wedge rdf:XMLLiteral. \in G$, such that $xyz \in L_{XML}$ then $G \models_{rdf} \{[] \text{ a } rdf:XMLLiteral\}$.

Remark: ill-formed XML literals are hardly (impossibly?) writeable in RDF/XML, but in other syntaxes.

Computing entailment for \models_{rdf}

Recall main result for simple entailment (\models):

Interpolation Lemma

$S \models E$ if and only if a subgraph of S is an instance of E .

Computing entailment for \models_{rdf}

Recall main result for simple entailment (\models):

Interpolation Lemma

$S \models E$ if and only if a subgraph of S is an instance of E .

Does no longer hold for \models_{rdf} :-)

BUT! Can be enabled by rules!

rdf-closure ($Cl_{rdf}(G)$)

The **rdf-closure** $Cl_{rdf}(G)$ of graph G is defined by $G \uplus A_{rdf}$ plus exhaustive application of the following rules to G :

- If $s \text{ p } o. \in Cl_{rdf}(G)$ then add **p a rdf:Property**.
- If $s \text{ p } "xyz" \wedge \text{rdf:XMLElement} \in Cl_{rdf}(G)$ and $xyz \in L_{XML}$ then add **__ :xyz a rdf:XMLElement**. where **__ :xyz** is a distinct bnode for that literal.

Slight abstraction of derivation rules *rdf1* and *rdf2* from the spec.

Computing entailment for \models_{rdf}

Recall main result for simple entailment (\models):

Interpolation Lemma

$S \models E$ if and only if a subgraph of S is an instance of E .

Does no longer hold for \models_{rdf} :- (

BUT! Can be enabled by rules!

rdf-closure ($Cl_{rdf}(G)$)

The **rdf-closure** $Cl_{rdf}(G)$ of graph G is defined by $G \uplus A_{rdf}$ plus exhaustive application of the following rules to G :

- p a `rdf:Property` . $\leftarrow s p o$.
- `_:xyz` a `rdf:XMLLiteral` . $\leftarrow s p \text{"xyz"} \wedge \text{rdf:XMLLiteral}$.

Let's write them in more "rule" syntax.

Computing entailment for \models_{rdf}

Recall main result for simple entailment (\models):

Interpolation Lemma

$S \models E$ if and only if a subgraph of S is an instance of E .

Does no longer hold for \models_{rdf} :- (

BUT! Can be enabled by rules!

rdf-closure ($Cl_{rdf}(G)$)

The **rdf-closure** $Cl_{rdf}(G)$ of graph G is defined by $G \uplus A_{rdf}$ plus exhaustive application of the following rules to G :

- $\forall s, p, o (triple(p, rdf:type, rdf:Property) \leftarrow triple(s, p, o))$
- $\forall s, p \exists b (triple(b, rdf:type, rdf:XMLLiteral) \leftarrow triple(s, p, "xyz" \wedge rdf:XMLLiteral))$

Or as first-order sentences.

Computing entailment for \models_{rdf}

Recall main result for simple entailment (\models):

Interpolation Lemma

$S \models E$ if and only if a subgraph of S is an instance of E .

Does no longer hold for \models_{rdf} :- (

BUT! Can be enabled by rules!

rdf-closure ($Cl_{rdf}(G)$)

The **rdf-closure** $Cl_{rdf}(G)$ of graph G is defined by $G \uplus A_{rdf}$ plus exhaustive application of the following rules to G :

- `triple(P,rdf:type,rdf:Property) :- triple(S,P,0).`
- `triple(bxyz,rdf:type,rdf:XMLLiteral) :- triple(S,P,"xyz"^^rdf:XMLLiteral)`

Or as Datalog rules.

rdf-entailment: Properties

RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S) \models E$

rdf-entailment: Properties

RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S) \models E$

Wait, slight problem for actual computation... $Cl_{rdf}(G)$ is infinite because of A_{rdf} .

rdf-entailment: Properties

RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S) \models E$

Wait, slight problem for actual computation... $Cl_{rdf}(G)$ is infinite because of A_{rdf} .

But: A finite subset of $Cl_{rdf}(G)$ is sufficient! Let the **reduced closure of G_1 w.r.t. G_2** , written $Cl_{rdf}(G_1, G_2)$, be defined as $Cl_{rdf}(G_1)$ with only those axiomatic triples $rdf:_n \quad rdf:type \quad rdf:Property$. where $rdf:_n \in V(G_2)$

rdf-entailment: Properties

RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S) \models E$

Wait, slight problem for actual computation... $Cl_{rdf}(G)$ is infinite because of A_{rdf} .

But: A finite subset of $Cl_{rdf}(G)$ is sufficient! Let the **reduced closure of G_1 w.r.t. G_2** , written $Cl_{rdf}(G_1, G_2)$, be defined as $Cl_{rdf}(G_1)$ with only those axiomatic triples $rdf:_n \quad rdf:type \quad rdf:Property$. where $rdf:_n \in V(G_2)$

\Rightarrow For a finite graph G_2 , $Cl_{rdf}(G_1, G_2)$ is always finite.

rdf-entailment: Properties

RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S) \models E$

Wait, slight problem for actual computation... $Cl_{rdf}(G)$ is infinite because of A_{rdf} .

But: A finite subset of $Cl_{rdf}(G)$ is sufficient! Let the **reduced closure of G_1 w.r.t. G_2** , written $Cl_{rdf}(G_1, G_2)$, be defined as $Cl_{rdf}(G_1)$ with only those axiomatic triples $rdf:_n \quad rdf:type \quad rdf:Property$. where $rdf:_n \in V(G_2)$

\Rightarrow For a finite graph G_2 , $Cl_{rdf}(G_1, G_2)$ is always finite.

Modified RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S, E) \models E$

rdf-entailment: Properties

RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S) \models E$

Wait, slight problem for actual computation... $Cl_{rdf}(G)$ is infinite because of A_{rdf} .

But: A finite subset of $Cl_{rdf}(G)$ is sufficient! Let the **reduced closure of G_1 w.r.t. G_2** , written $Cl_{rdf}(G_1, G_2)$, be defined as $Cl_{rdf}(G_1)$ with only those axiomatic triples $rdf:_n \quad rdf:type \quad rdf:Property$. where $rdf:_n \in V(G_2)$

\Rightarrow For a finite graph G_2 , $Cl_{rdf}(G_1, G_2)$ is always finite.

Modified RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S, E) \models E$

Means that we can compute RDF entailment finitely! :-)

So what?

Now that was still fairly uninteresting. . .

So what?

Now that was still fairly uninteresting...

... Didn't buy us too much, except that:

- entailment became more complicated
- we have a way to express that a predicate is an `rdf:Property`
- we have a way to express that XML Literals are `rdf:XMLLiterals`

So what?

Now that was still fairly uninteresting...

... Didn't buy us too much, except that:

- entailment became more complicated
- we have a way to express that a predicate is an `rdf:Property`
- we have a way to express that XML Literals are `rdf:XMLLiterals`

Still, we cannot **describe other vocabularies** \Rightarrow **`rdfs:`**

The rdfs: vocabulary

The **RDFS vocabulary**, `rdfsV`, extends `rdfV`:

- `rdfs:domain`, `rdfs:range`
- `rdfs:subClassOf`, `rdfs:subPropertyOf`
- `rdfs:Resource` ... all resources.
- `rdfs:Class` ... all resources which denote classes.
- `rdfs:Literal` ... the “class of literal values”.
- `rdfs:Datatype` ... the “class of datatypes”.
- `rdfs:Container` ... a joint superclass of `rdf:Alt`, `rdf:Seq`, `rdf:Bag`.
- `rdfs:ContainerMembershipProperty` ... the “class of container membership predicates” (`rdf:_1`, `rdf:_2`, ...).
- `rdfs:member` a joint superproperty of `rdf:_1`, `rdf:_2`, ...
- `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`, `rdfs:label` some more conventional vocabulary without strong “formal” semantics¹

¹for illustrative examples, let's check the FOAF ontology

RDF/XML version: <http://xmlns.com/foaf/spec/20071002.rdf>

Extracted Turtle version:

http://www.polleres.net/teaching/SemWebTech_2009/testdata/foaf0ntology.ttl

rdfs-interpretation

rdfs-interpretation

An **rdfs-interpretation** is an rdf-interpretation I defined by additional

- semantic conditions
- axiomatic triples A_{rdfs}

rdfs-interpretation

rdfs-interpretation

An **rdfs-interpretation** is an rdf-interpretation I defined by additional

- semantic conditions
- axiomatic triples A_{rdfs}

Additionally to IR and IP , the semantic conditions for RDF also talk about the **set of classes**, denoted by $IC \subseteq IR$:

rdfs-interpretation

rdfs-interpretation

An **rdfs-interpretation** is an rdf-interpretation I defined by additional

- semantic conditions
- axiomatic triples A_{rdfs}

Additionally to IR and IP , the semantic conditions for RDF also talk about the **set of classes**, denoted by $IC \subseteq IR$:

IC

IC is defined as the set of resources which appear in the range of the extension of `rdf:type`, i.e. all the y such that $\exists x. \langle x, y \rangle \in IEXT(I(\text{rdf:type}))$.

rdfs-interpretation

rdfs-interpretation

An **rdfs-interpretation** is an rdf-interpretation I defined by additional

- semantic conditions
- axiomatic triples A_{rdfs}

Additionally to IR and IP , the semantic conditions for RDF also talk about the **set of classes**, denoted by $IC \subseteq IR$:

IC

IC is defined as the set of resources which appear in the range of the extension of `rdf:type`, i.e. all the y such that $\exists x. \langle x, y \rangle \in IEXT(I(\text{rdf:type}))$.

For brevity, we also define:

$ICEXT$

$ICEXT : IC \rightarrow 2^{IR}$ defined as $x \in ICEXT(y) \Leftrightarrow \langle x, y \rangle \in IEXT(I(\text{rdf:type}))$

rdfs-interpretation

rdfs-interpretation

An **rdfs-interpretation** is an rdf-interpretation I defined by additional

- semantic conditions
- axiomatic triples A_{rdfs}

Additionally to IR and IP , the semantic conditions for RDF also talk about the **set of classes**, denoted by $IC \subseteq IR$:

IC

IC is defined as the set of resources which appear in the range of the extension of rdf:type , i.e. all the y such that $\exists x. \langle x, y \rangle \in IEXT(I(\text{rdf:type}))$.

For brevity, we also define:

$ICEXT$

$ICEXT : IC \rightarrow 2^{IR}$ defined as $x \in ICEXT(y) \Leftrightarrow \langle x, y \rangle \in IEXT(I(\text{rdf:type}))$

Let's look into the **semantic conditions** and **axiomatic triples** for rdfs-interpretations now...

rdfs-interpretations: Semantic conditions

x is in $\text{ICEXT}(y)$ if and only if $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:type}))$

$\text{IC} = \text{ICEXT}(\text{I}(\text{rdfs:Class}))$

$\text{IR} = \text{ICEXT}(\text{I}(\text{rdfs:Resource}))$

$\text{LV} = \text{ICEXT}(\text{I}(\text{rdfs:Literal}))$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:domain}))$ and $\langle u,v \rangle$ is in $\text{IEXT}(x)$ then u is in $\text{ICEXT}(y)$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:range}))$ and $\langle u,v \rangle$ is in $\text{IEXT}(x)$ then v is in $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ is transitive and reflexive on IP

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ then x and y are in IP and $\text{IEXT}(x)$ is a subset of $\text{IEXT}(y)$

If x is in IC then $\langle x, \text{I}(\text{rdfs:Resource}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ then x and y are in IC and $\text{ICEXT}(x)$ is a subset of $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ is transitive and reflexive on IC

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:ContainerMembershipProperty}))$ then:
 $\langle x, \text{I}(\text{rdfs:member}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:Datatype}))$ then $\langle x, \text{I}(\text{rdfs:Literal}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

rdfs-interpretations: Semantic conditions

x is in $\text{ICEXT}(y)$ if and only if $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:type}))$

$\text{IC} = \text{ICEXT}(\text{I}(\text{rdfs:Class}))$

$\text{IR} = \text{ICEXT}(\text{I}(\text{rdfs:Resource}))$

$\text{LV} = \text{ICEXT}(\text{I}(\text{rdfs:Literal}))$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:domain}))$ and $\langle u,v \rangle$ is in $\text{IEXT}(x)$ then u is in $\text{ICEXT}(y)$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:range}))$ and $\langle u,v \rangle$ is in $\text{IEXT}(x)$ then v is in $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ is transitive and reflexive on IP

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ then x and y are in IP and $\text{IEXT}(x)$ is a subset of $\text{IEXT}(y)$

If x is in IC then $\langle x, \text{I}(\text{rdfs:Resource}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ then x and y are in IC and $\text{ICEXT}(x)$ is a subset of $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ is transitive and reflexive on IC

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:ContainerMembershipProperty}))$ then:
 $\langle x, \text{I}(\text{rdfs:member}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:Datatype}))$ then $\langle x, \text{I}(\text{rdfs:Literal}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

o a Class . $\Leftarrow s$ a o .

$[]$ a s . $\Leftarrow s$ a Class .

s a Resource . $\Leftarrow s$ p o .

p a Resource . $\Leftarrow s$ p o .

o a Resource . $\Leftarrow s$ p o .

$_ : 1$ a Literal . $\Leftarrow s$ p l .

s p $_ : 1$. $\Leftarrow s$ p l .

$o \notin L$

$l \in L$

$l \in L$

rdfs-interpretations: Semantic conditions

x is in $\text{ICEXT}(y)$ if and only if $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:type}))$

$\text{IC} = \text{ICEXT}(\text{I}(\text{rdfs:Class}))$

$\text{IR} = \text{ICEXT}(\text{I}(\text{rdfs:Resource}))$

$\text{LV} = \text{ICEXT}(\text{I}(\text{rdfs:Literal}))$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:domain}))$ and $\langle u,v \rangle$ is in $\text{IEXT}(x)$ then u is in $\text{ICEXT}(y)$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:range}))$ and $\langle u,v \rangle$ is in $\text{IEXT}(x)$ then v is in $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ is transitive and reflexive on IP

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ then x and y are in IP and $\text{IEXT}(x)$ is a subset of $\text{IEXT}(y)$

If x is in IC then $\langle x, \text{I}(\text{rdfs:Resource}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

If $\langle x,y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ then x and y are in IC and $\text{ICEXT}(x)$ is a subset of $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ is transitive and reflexive on IC

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:ContainerMembershipProperty}))$ then:
 $\langle x, \text{I}(\text{rdfs:member}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:Datatype}))$ then $\langle x, \text{I}(\text{rdfs:Literal}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

o a Class . $\Leftarrow s$ a o .

$[]$ a s . $\Leftarrow s$ a Class .

s a Resource . $\Leftarrow s$ p o .

p a Resource . $\Leftarrow s$ p o .

o a Resource . $\Leftarrow s$ p o .

$_ : 1$ a Literal . $\Leftarrow s$ p l .

s p $_ : 1$. $\Leftarrow s$ p l .

$o \notin L$
 $l \in L$
 $l \in L$

s a c . $\Leftarrow p$ domain c . s p o .

rdfs-interpretations: Semantic conditions

x is in $\text{ICEXT}(y)$ if and only if $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:type}))$

$\text{IC} = \text{ICEXT}(\text{I}(\text{rdfs:Class}))$

$\text{IR} = \text{ICEXT}(\text{I}(\text{rdfs:Resource}))$

$\text{LV} = \text{ICEXT}(\text{I}(\text{rdfs:Literal}))$

o a Class . $\Leftarrow s$ a o .

$[]$ a s . $\Leftarrow s$ a Class .

s a Resource . $\Leftarrow s$ p o .

p a Resource . $\Leftarrow s$ p o .

o a Resource . $\Leftarrow s$ p o .

$_ : 1$ a Literal . $\Leftarrow s$ p l .

s p $_ : 1$. $\Leftarrow s$ p l .

$o \notin L$

$l \in L$

$l \in L$

If $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:domain}))$ and $\langle u, v \rangle$ is in $\text{IEXT}(x)$ then u is in $\text{ICEXT}(y)$

s a c . $\Leftarrow p$ domain c . s p o .

If $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:range}))$ and $\langle u, v \rangle$ is in $\text{IEXT}(x)$ then v is in $\text{ICEXT}(y)$

o a c . $\Leftarrow p$ range c . s p o . $o \notin L$

$\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ is transitive and reflexive on IP

If $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ then x and y are in IP and $\text{IEXT}(x)$ is a subset of $\text{IEXT}(y)$

If x is in IC then $\langle x, \text{I}(\text{rdfs:Resource}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

If $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ then x and y are in IC and $\text{ICEXT}(x)$ is a subset of $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ is transitive and reflexive on IC

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:ContainerMembershipProperty}))$ then:
 $\langle x, \text{I}(\text{rdfs:member}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:Datatype}))$ then $\langle x, \text{I}(\text{rdfs:Literal}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

rdfs-interpretations: Semantic conditions

x is in $\text{ICEXT}(y)$ if and only if $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:type}))$

$\text{IC} = \text{ICEXT}(\text{I}(\text{rdfs:Class}))$

$\text{IR} = \text{ICEXT}(\text{I}(\text{rdfs:Resource}))$

$\text{LV} = \text{ICEXT}(\text{I}(\text{rdfs:Literal}))$

If $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:domain}))$ and $\langle u, v \rangle$ is in $\text{IEXT}(x)$ then u is in $\text{ICEXT}(y)$

If $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:range}))$ and $\langle u, v \rangle$ is in $\text{IEXT}(x)$ then v is in $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ is transitive and reflexive on IP

If $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$ then x and y are in IP and $\text{IEXT}(x)$ is a subset of $\text{IEXT}(y)$

If x is in IC then $\langle x, \text{I}(\text{rdfs:Resource}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

If $\langle x, y \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ then x and y are in IC and $\text{ICEXT}(x)$ is a subset of $\text{ICEXT}(y)$

$\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$ is transitive and reflexive on IC

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:ContainerMembershipProperty}))$ then:
 $\langle x, \text{I}(\text{rdfs:member}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subPropertyOf}))$

If x is in $\text{ICEXT}(\text{I}(\text{rdfs:Datatype}))$ then $\langle x, \text{I}(\text{rdfs:Literal}) \rangle$ is in $\text{IEXT}(\text{I}(\text{rdfs:subClassOf}))$

o a Class . $\Leftarrow s$ a o .

$[\]$ a s . $\Leftarrow s$ a Class .

s a Resource . $\Leftarrow s$ p o .

p a Resource . $\Leftarrow s$ p o .

o a Resource . $\Leftarrow s$ p o .

$_ : l$ a Literal . $\Leftarrow s$ p l .

s p $_ : l$. $\Leftarrow s$ p l .

$o \notin L$

$l \in L$

$l \in L$

s a c . $\Leftarrow p$ domain c . s p o .

o a c . $\Leftarrow p$ range c . s p o . $o \notin L$

s subPropertyOf r . $\Leftarrow s$ subPropertyOf o .
 o subPropertyOf r .

s subPropertyOf s . $\Leftarrow s$ a Property .

s a Property . $\Leftarrow s$ subPropertyOf o .

o a Property . $\Leftarrow s$ subPropertyOf o .

s q $o \Leftarrow p$ subPropertyOf q . s p o . $q \notin BL$

s subClassOf Resource . $\Leftarrow s$ a Class .

s a Class . $\Leftarrow s$ subClassOf o .

o a Class . $\Leftarrow s$ subClassOf o . $o \notin L$

s a c . $\Leftarrow o$ suClassOf c . s a o .

s subClassOf r . $\Leftarrow s$ subClassOf o .

o subClassOf r .

s subClassOf s . $\Leftarrow s$ a Property .

s subPropertyOf member . \Leftarrow

s a ContainerMembershipProperty .

s subClassOf Literal . $\Leftarrow s$ a Datatype .

rdfs-interpretations: Additional Axiomatic Triples

 A_{rdfs}

<code>rdf:type</code>	<code>rdfs:domain rdfs:Resource;</code> <code>rdfs:range rdfs:Class .</code>		
<code>rdfs:domain</code>	<code>rdfs:domain rdf:Property;</code> <code>rdfs:range rdfs:Class .</code>	<code>rdf:subject</code>	<code>rdfs:domain rdf:Statement;</code> <code>rdfs:range rdfs:Resource .</code>
<code>rdfs:range</code>	<code>rdfs:domain rdf:Property;</code> <code>rdfs:range rdfs:Class .</code>	<code>rdf:predicate</code>	<code>rdfs:domain rdf:Statement;</code> <code>rdfs:range rdfs:Resource .</code>
<code>rdfs:subPropertyOf</code>	<code>rdfs:domain rdf:Property;</code> <code>rdfs:range rdf:Property .</code>	<code>rdf:object</code>	<code>rdfs:domain rdf:Statement;</code> <code>rdfs:range rdfs:Resource .</code>
<code>rdfs:subClassOf</code>	<code>rdfs:domain rdfs:Class;</code> <code>rdfs:range rdfs:Class .</code>		<code>rdf:Alt rdfs:subClassOf rdfs:Container .</code> <code>rdf:Bag rdfs:subClassOf rdfs:Container .</code> <code>rdf:Seq rdfs:subClassOf rdfs:Container .</code> <code>rdfs:ContainerMembershipProperty</code> <code> rdfs:subClassOf rdf:Property .</code>
<code>rdfs:member</code>	<code>rdfs:domain rdfs:Resource;</code> <code>rdfs:range rdfs:Resource .</code>		<code>rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .</code>
<code>rdfs:seeAlso</code>	<code>rdfs:domain rdfs:Resource;</code> <code>rdfs:range rdfs:Resource .</code>		<code>rdf:XMLLiteral rdf:type rdfs:Datatype .</code> <code>rdf:XMLLiteral rdfs:subClassOf rdfs:Literal .</code> <code>rdfs:Datatype rdfs:subClassOf rdfs:Class .</code>
<code>rdfs:isDefinedBy</code>	<code>rdfs:domain rdfs:Resource;</code> <code>rdfs:range rdfs:Resource .</code>		<code>rdf:_1 rdf:type rdfs:ContainerMembershipProperty .</code> <code>rdf:_1 rdfs:domain rdfs:Resource .</code> <code>rdf:_1 rdfs:range rdfs:Resource .</code> <code>rdf:_2 rdf:type rdfs:ContainerMembershipProperty .</code> <code>rdf:_2 rdfs:domain rdfs:Resource .</code> <code>rdf:_2 rdfs:range rdfs:Resource .</code> <code>...</code>
<code>rdf:value</code>	<code>rdfs:domain rdfs:Resource;</code> <code>rdfs:range rdfs:Resource .</code>		
<code>rdfs:comment</code>	<code>rdfs:domain rdfs:Resource;</code> <code>rdfs:range rdfs:Literal .</code>		
<code>rdfs:label</code>	<code>rdfs:domain rdfs:Resource;</code> <code>rdfs:range rdfs:Literal .</code>		
<code>rdf:first</code>	<code>rdfs:domain rdf:List;</code> <code>rdfs:range rdfs:Resource .</code>		
<code>rdf:rest</code>	<code>rdfs:domain rdf:List .</code> <code>rdfs:range rdf:List .</code>		

rdfs-interpretations: Axiomatic Triples

A_{rdfs} has some “somewhat redundant triples” (let’s call the “gist” A_{rdfs}^-)

<code>rdfs:type</code>	<code>rdfs:range rdfs:Class .</code>	<code>rdf:subject</code>	<code>rdfs:domain rdf:Statement;</code>
<code>rdfs:domain</code>	<code>rdfs:range rdfs:Class .</code>	<code>rdf:predicate</code>	<code>rdfs:domain rdf:Statement;</code>
<code>rdfs:range</code>	<code>rdfs:range rdfs:Class .</code>	<code>rdf:object</code>	<code>rdfs:domain rdf:Statement;</code>
<code>rdfs:subClassOf</code>	<code>rdfs:domain rdfs:Class;</code> <code>rdfs:range rdfs:Class .</code>	<code>rdf:Alt</code>	<code>rdfs:subClassOf rdfs:Container .</code>
		<code>rdf:Bag</code>	<code>rdfs:subClassOf rdfs:Container .</code>
		<code>rdf:Seq</code>	<code>rdfs:subClassOf rdfs:Container .</code>
		<code>rdfs:ContainerMembershipProperty</code>	<code>rdfs:subClassOf rdf:Property .</code>
<code>rdfs:comment</code>	<code>rdfs:range rdfs:Literal .</code>	<code>rdfs:isDefinedBy</code>	<code>rdfs:subPropertyOf rdfs:seeAlso .</code>
<code>rdfs:label</code>	<code>rdfs:range rdfs:Literal .</code>		
		<code>rdf:XMLLiteral</code>	<code>rdf:type rdfs:Datatype .</code>
<code>rdf:first</code>	<code>rdfs:domain rdf:List .</code>	<code>rdf:XMLLiteral</code>	<code>rdfs:subClassOf rdfs:Literal .</code>
<code>rdf:rest</code>	<code>rdfs:domain rdf:List .</code> <code>rdfs:range rdf:List .</code>	<code>rdfs:Datatype</code>	<code>rdfs:subClassOf rdfs:Class .</code>
		<code>rdf:_1</code>	<code>rdf:type rdfs:ContainerMembershipProperty .</code>
		<code>rdf:_2</code>	<code>rdf:type rdfs:ContainerMembershipProperty .</code>
			<code>...</code>

Actually, one could think: A_{rdfs}^- is enough:

- Axiomatic triples s `domain/range rdfs:Resource .`
- Axiomatic triples p `domain/range rdf:Property .`

rdfs-interpretations: Axiomatic Triples

A_{rdfs} has some “somewhat redundant triples” (let’s call the “gist” A_{rdfs}^-)

		<code>rdf:subject</code>	<code>rdfs:domain rdf:Statement;</code>
		<code>rdf:predicate</code>	<code>rdfs:domain rdf:Statement;</code>
		<code>rdf:object</code>	<code>rdfs:domain rdf:Statement;</code>
<code>rdf:type</code>	<code>rdfs:range rdfs:Class .</code>		
<code>rdfs:domain</code>	<code>rdfs:range rdfs:Class .</code>		
<code>rdfs:range</code>	<code>rdfs:range rdfs:Class .</code>		
<code>rdfs:subClassOf</code>	<code>rdfs:domain rdfs:Class;</code> <code>rdfs:range rdfs:Class .</code>	<code>rdf:Alt rdfs:subClassOf rdfs:Container .</code> <code>rdf:Bag rdfs:subClassOf rdfs:Container .</code> <code>rdf:Seq rdfs:subClassOf rdfs:Container .</code> <code>rdfs:ContainerMembershipProperty</code> <code>rdfs:subClassOf rdf:Property .</code>	
<code>rdfs:comment</code>	<code>rdfs:range rdfs:Literal .</code>		<code>rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .</code>
<code>rdfs:label</code>	<code>rdfs:range rdfs:Literal .</code>		
			<code>rdf:XMLLiteral rdf:type rdfs:Datatype .</code> <code>rdf:XMLLiteral rdfs:subClassOf rdfs:Literal .</code> <code>rdfs:Datatype rdfs:subClassOf rdfs:Class .</code>
<code>rdf:first</code>	<code>rdfs:domain rdf:List .</code>		
<code>rdf:rest</code>	<code>rdfs:domain rdf:List .</code> <code>rdfs:range rdf:List .</code>	<code>rdf:_1 rdf:type rdfs:ContainerMembershipProperty .</code> <code>rdf:_2 rdf:type rdfs:ContainerMembershipProperty .</code> ...	

Actually, one could think: A_{rdfs}^- is enough:

- Axiomatic triples s `domain/range rdfs:Resource .`
- Axiomatic triples p `domain/range rdf:Property .`

More on that later. **bottomline: if we aren't interested in inferring triples**
 x `rdfs:domain rdfs:Resource.` , we can probably do without these.

rdfs-entailment: Properties

Recall:

Modified RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S, E) \models E$

rdfs-entailment: Properties

Recall:

Modified RDF entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdf}(S, E) \models E$

Can we do something similar for rdfs?

rdfs-entailment: Properties

Recall:

Modified RDFS entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdfs}(S, E) \models E$

Can we do something similar for rdfs?

Good news: Yes! Extension from $Cl_{rdf}(G, E)$ to $Cl_{rdfs}(G, E)$ is straightforward, just add the new inference rules.

rdfs-entailment: Properties

Recall:

Modified RDFS entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdfs}(S, E) \models E$

Can we do something similar for rdfs?

Good news: Yes! Extension from $Cl_{rdf}(G, E)$ to $Cl_{rdfs}(G, E)$ is straightforward, just add the new inference rules.

RDF Spec[Hayes, 2004, Section 7] provides an *informative* set of entailment rules to capture RDF- and RDFS entailment. Pretty much our rules in slides 16+21.

rdfs-entailment: Properties

Recall:

Modified RDFS entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdfs}(S, E) \models E$

Can we do something similar for rdfs?

Good news: Yes! Extension from $Cl_{rdf}(G, E)$ to $Cl_{rdfs}(G, E)$ is straightforward, just add the new inference rules.

RDF Spec[Hayes, 2004, Section 7] provides an *informative* set of entailment rules to capture RDF- and RDFS entailment. Pretty much our rules in slides 16+21.

Some errors found later, several refinements in the literature.[ter Horst, 2005][Muñoz *et al.*, 2007][Bruijn and Heymans, 2007][Ianni *et al.*, 2009].

rdfs-entailment: Properties

Recall:

Modified RDFS entailment lemma (slightly adapted from the spec)

$S \models_{rdf} E$ if and only if $Cl_{rdfs}(S, E) \models E$

Can we do something similar for rdfs?

Good news: Yes! Extension from $Cl_{rdf}(G, E)$ to $Cl_{rdfs}(G, E)$ is straightforward, just add the new inference rules.

RDF Spec[Hayes, 2004, Section 7] provides an *informative* set of entailment rules to capture RDF- and RDFS entailment. Pretty much our rules in slides 16+21.

Some errors found later, several refinements in the literature.[ter Horst, 2005][Muñoz *et al.*, 2007][Bruijn and Heymans, 2007][Ianni *et al.*, 2009].

... but, means that we can compute RDFS entailment finitely! :-)

Spec entailment rules are – incomplete!

Some of the rules we had before had restrictions in the antecedent, e.g.

$s \text{ a } c . \Leftarrow p \text{ domain } c . \quad s \text{ p } o .$
 $s \text{ q } o \Leftarrow p \text{ subPropertyOf } q . \quad s \text{ p } o .$

$q \notin BL$

Spec entailment rules are – incomplete!

Some of the rules we had before had restrictions in the antecedent, e.g.

```
s a c .  $\Leftarrow$  p domain c . s p o .
s q o  $\Leftarrow$  p subPropertyOf q . s p o .
```

q \notin BL

But what about that:

```
p rdfs:subPropertyOf _:p
_:p rdfs:domain c.
s p o .
```


Spec entailment rules are – incomplete!

Some of the rules we had before had restrictions in the antecedent, e.g.

```
s a c .  $\Leftarrow$  p domain c . s p o .
s q o  $\Leftarrow$  p subPropertyOf q . s p o .
```

q \notin BL

But what about that:

```
p rdfs:subPropertyOf _:p
_:p rdfs:domain c.
s p o .
```

Problem: If we execute the rules with the restrictions in place, we won't get:

```
s rdf:type c.
```

The invalid “intermediate” triple

```
s _:p c.
```

cannot be inferred “within RDF”.

Spec entailment rules are – incomplete!

Some of the rules we had before had restrictions in the antecedent, e.g.

$$s \text{ a } c . \Leftarrow p \text{ domain } c . \quad s \text{ p o } .$$

$$s \text{ q o } \Leftarrow p \text{ subPropertyOf } q . \quad s \text{ p o } .$$

$q \notin BL$

But what about that:

$$p \text{ rdfs:subPropertyOf } _ : p$$

$$_ : p \text{ rdfs:domain } c .$$

$$s \text{ p o } .$$

Problem: If we execute the rules with the restrictions in place, we won't get:

$$s \text{ rdf:type } c .$$

The invalid “intermediate” triple

$$s _ : p \text{ c} .$$

cannot be inferred “within RDF”.

Solution: easy, (i) allow “generalized” RDF ($UBL \times UBL \times UBL$) for intermediate computation of closure [ter Horst, 2005], or (ii) use modified “implicit typing rule”, cf. [Muñoz *et al.*, 2007]:

$$s \text{ a } c \Leftarrow p \text{ subPropertyOf } q . \quad q \text{ rdfs:domain } c . \quad s \text{ p o} .$$

Inconsistency

Proposition 3

ANY Graph

- containing a triple $s \ p \ "xyz" \wedge \text{rdf:XMLLiteral}$. such that xyz is NOT a well-formed XML literal and
- allowing to infer the triple $p \ \text{rdfs:range} \ \text{rdf:XMLLiteral}$.

is **false in any rdfs-interpretation**.

Inconsistency

Proposition 3

ANY Graph

- containing a triple $s \ p \ "xyz" \wedge \text{rdf:XMLLiteral}$. such that xyz is NOT a well-formed XML literal and
- allowing to infer the triple $p \ \text{rdfs:range} \ \text{rdf:XMLLiteral}$.

is **false in any rdfs-interpretation**.

Why?

- by semantic conditions of rdf- and rdfs-interpretations!

Inconsistency

Proposition 3

ANY Graph

- containing a triple $s \ p \ "xyz" \wedge \text{rdf:XMLLiteral}$. such that xyz is NOT a well-formed XML literal and
- allowing to infer the triple $p \ \text{rdfs:range} \ \text{rdf:XMLLiteral}$.

is **false in any rdfs-interpretation**.

Why?

- by semantic conditions of `rdf-` and `rdfs-interpretations`!

Implication:

- There can be `rdfs-inconsistent` RDF graphs, e.g.

```
ex:a ex:p "<notLegalXML"^^rdf:XMLLiteral .
```

```
ex:p rdfs:range rdf:XMLLiteral .
```

rdfs-entailment: Example 1

G_5 :

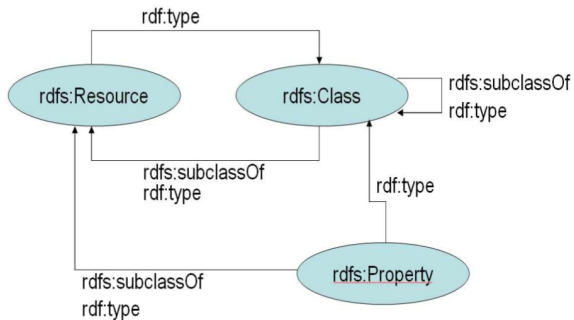
```
rdfs:Resource rdf:type rdfs:Class.  
rdf:Property rdf:type rdfs:Resource.  
rdf:Property rdfs:subClassOf rdfs:Resource.  
rdf:Property rdf:type rdfs:Class.  
rdfs:Class rdf:type rdfs:Resource.  
rdfs:Class rdf:type rdfs:Class.  
rdfs:Class rdfs:subClassOf rdfs:Resource.  
rdfs:Class rdfs:subClassOf rdfs:Class.
```

rdfs-entailment: Example 1

 $\{\} \models_{rdfs} G_5$ $G_5 :$

```
rdfs:Resource rdf:type rdfs:Class.  
rdf:Property rdf:type rdfs:Resource.  
rdf:Property rdfs:subClassOf rdfs:Resource.  
rdf:Property rdf:type rdfs:Class.  
rdfs:Class rdf:type rdfs:Resource.  
rdfs:Class rdf:type rdfs:Class.  
rdfs:Class rdfs:subClassOf rdfs:Resource.  
rdfs:Class rdfs:subClassOf rdfs:Class.
```

rdfs-entailment: Example 1

 $\{\} \models_{rdfs} G_5$ $G_5 :$ 

rdfs-entailment: Example 2

$$G_1 \models_{rdfs} G_2$$

G_1 :

```
ex:alice foaf:knows ex:bob.  
ex:alice foaf:name "Alice".  
foaf:knows rdfs:domain foaf:Person.
```

G_2 :

```
ex:alice rdf:type foaf:Person.
```

rdfs-entailment: Example 3

$$\{ \} \models_{rdfs} \{ xxx \text{ rdf:type rdfs:Resource. } \}$$

for all xxx in the Vocabulary of any interpretation I .

Implication: Since all rdfs-Interpretations are over an infinite vocabulary (rdf:_n), even without the axiomatic triples that would already entail infinite triples.

Unit Outline

1. Assignment 1 – Some words on common mistakes
2. Some words on the usage of blank nodes on the Web
3. RDF/S Entailment: semantics of the `rdf:` and `rdfs:` vocabularies
4. D-Entailment: Giving Semantics to Datatypes
5. OWL Entailment: Giving Semantics to the `owl:` Vocabulary

Datatype

A **datatype** d is an entity characterized by a set of character strings called lexical forms (or *lexical space*) and a mapping from that set to a set of values (called *value space*). Exactly how these sets and mappings (called **lexical-to-value mapping** $L2V(d)$) are defined is a matter external to RDF, e.g. XML Schema [XML Schema Datatypes, 2001].

Datatype

A **datatype** d is an entity characterized by a set of character strings called lexical forms (or *lexical space*) and a mapping from that set to a set of values (called *value space*). Exactly how these sets and mappings (called **lexical-to-value mapping** $L2V(d)$) are defined is a matter external to RDF, e.g. XML Schema [XML Schema Datatypes, 2001].

A **datatype map** is a set of datatypes.

Datatype

A **datatype** d is an entity characterized by a set of character strings called lexical forms (or *lexical space*) and a mapping from that set to a set of values (called *value space*). Exactly how these sets and mappings (called *lexical-to-value mapping* $L2V(d)$) are defined is a matter external to RDF, e.g. XML Schema [XML Schema Datatypes, 2001].

A *datatype map* is a set of datatypes.

Besides `rdf:XMLLiteral`, the RDF spec suggests to support the following XML Schema datatypes:

XML datatypes

`xsd:string`, `xsd:boolean`, `xsd:decimal`, `xsd:float`, `xsd:double`, `xsd:dateTime`, `xsd:time`, `xsd:date`, `xsd:gYearMonth`, `xsd:gYear`, `xsd:gMonthDay`, `xsd:gDay`, `xsd:gMonth`, `xsd:hexBinary`, `xsd:base64Binary`, `xsd:anyURI`, `xsd:normalizedString`, `xsd:token`, `xsd:language`, `xsd:NMTOKEN`, `xsd:Name`, `xsd:NCName`, `xsd:integer`, `xsd:nonPositiveInteger`, `xsd:negativeInteger`, `xsd:long`, `xsd:int`, `xsd:short`, `xsd:byte`, `xsd:nonNegativeInteger`, `xsd:unsignedLong`, `xsd:unsignedInt`, `xsd:unsignedShort`, `xsd:unsignedByte`, `xsd:positiveInteger`

D-interpretations

Let D be a set of pairs $\langle aaa, x \rangle$, s.t. aaa is a URI reference and x a datatype (datatype map).

D-interpretations

Let D be a set of pairs $\langle aaa, x \rangle$, s.t. aaa is a URI reference and x a datatype (datatype map).

D-interpretation

A **D-interpretation** is an rdfs-interpretation I defined by the following additional semantic conditions:

General semantic conditions for datatypes.

if $\langle aaa, x \rangle$ is in D then $I(aaa) = x$

if $\langle aaa, x \rangle$ is in D then $ICEXT(x)$ is the value space of x and is a subset of LV

if $\langle aaa, x \rangle$ is in D then for any typed literal "sss"^^ddd in V with $I(ddd) = x$,
if sss is in the lexical space of x then $IL("sss"^^ddd) = L2V(x)(sss)$, otherwise $IL("sss"^^ddd)$ is not in LV

if $\langle aaa, x \rangle$ is in D then $I(aaa)$ is in $ICEXT(I(rdfs:Datatype))$

D-interpretations

Let D be a set of pairs $\langle aaa, x \rangle$, s.t. aaa is a URI reference and x a datatype (datatype map).

D-interpretation

A **D-interpretation** is an rdfs-interpretation I defined by the following additional semantic conditions:

General semantic conditions for datatypes.

if $\langle aaa, x \rangle$ is in D then $I(aaa) = x$

if $\langle aaa, x \rangle$ is in D then $ICEXT(x)$ is the value space of x and is a subset of LV

if $\langle aaa, x \rangle$ is in D then for any typed literal "sss"^^ddd in V with $I(ddd) = x$,
if sss is in the lexical space of x then $IL("sss"^^ddd) = L2V(x)(sss)$, otherwise $IL("sss"^^ddd)$ is not in LV

if $\langle aaa, x \rangle$ is in D then $I(aaa)$ is in $ICEXT(I(rdfs:Datatype))$

- In principle, just generalization of the Semantic conditions for `rdf:XMLLiteral` to other datatypes.
- Inuitive Reading: fix the interpretation of typed literals according to $L2V(d)$

That's it!

D-entailment (\models_D) – Examples
$$\{ s \text{ p } "2.0"^\wedge \text{xsd:decimal.} \} \models_D \{ s \text{ p } "2"^\wedge \text{xsd:integer.} \}$$
$$\{ s \text{ p } "2"^\wedge \text{xsd:integer.} \} \models_D \{ s \text{ p } "2.0"^\wedge \text{xsd:decimal.} \}$$

D-entailment (\models_D) – Examples

`{ s p "2.0"^^xsd:decimal. } \models_D { s p "2"^^xsd:integer. }`

`{ s p "2"^^xsd:integer. } \models_D { s p "2.0"^^xsd:decimal. }`

D-inconsistent graphs:

`{ ex:a ex:b "25"^^xsd:decimal . ex:b rdfs:range xsd:string .}`
 (because the value spaces of `sd:string` and `xsd:decimal` are disjoint)

`ex:a ex:p "2.5"^^xsd:decimal . ex:p rdfs:range xsd:integer .}`
 (overlapping value spaces, but 2.5 outside `xsd:integer`'s value space)

`{ ex:a ex:b "haha"^^xsd:decimal. }`

(outside lexical space, i.e. "ill-formed" for `xsd:decimal`)

Rule-based computation of D-entailment?

- Basically, needs an oracle to evaluate $L2V$...

e.g. could be done by a built-in that maps all typed literals in a graph to compute $L2V$ of their value:

$$s \text{ p } L2V(l) . \Leftarrow s \text{ p } l \wedge d . \quad \text{for } l \text{ in the lex. space of } d.$$

Rule-based computation of D-entailment?

- Basically, needs an oracle to evaluate $L2V$...

e.g. could be done by a built-in that maps all typed literals in a graph to compute $L2V$ of their value:

$$s \text{ p } L2V(l) \text{ .} \Leftarrow s \text{ p } l \wedge d \text{ .} \quad \text{for } l \text{ in the lex. space of } d.$$

Again needs “generalized intermediate triples”

$(UBL \cup V_s \times UBL \cup V_s \times UBL \cup V_s)$ for intermediate computation of closure, where V_s is the union of the value spaces of all supported datatypes.

Rule-based computation of D-entailment?

- Basically, needs an oracle to evaluate $L2V$...

e.g. could be done by a built-in that maps all typed literals in a graph to compute $L2V$ of their value:

$$s \text{ p } L2V(l) \text{ .} \Leftarrow s \text{ p } l^{\wedge} d \text{ .} \quad \text{for } l \text{ in the lex. space of } d.$$

Again needs “generalized intermediate triples”
 $(UBL \cup V_s \times UBL \cup V_s \times UBL \cup V_s)$ for intermediate computation of closure, where V_s is the union of the value spaces of all supported datatypes.

Implementation often use “*canonicalization*”, i.e. internally only handle canonical representations of typed literals.

Rule-based computation of D-entailment?

- Basically, needs an oracle to evaluate $L2V$...

e.g. could be done by a built-in that maps all typed literals in a graph to compute $L2V$ of their value:

$$s \text{ p } L2V(l) \text{ .} \Leftarrow s \text{ p } l^{\wedge} d \text{ .} \quad \text{for } l \text{ in the lex. space of } d.$$

Again needs “generalized intermediate triples”
 $(UBL \cup V_s \times UBL \cup V_s \times UBL \cup V_s)$ for intermediate computation of closure, where V_s is the union of the value spaces of all supported datatypes.

Implementation often use “*canonicalization*”, i.e. internally only handle canonical representations of typed literals.

Note again: D-entailment is on top of rdfs-Entailment!

Unit Outline

1. Assignment 1 – Some words on common mistakes
2. Some words on the usage of blank nodes on the Web
3. RDF/S Entailment: semantics of the `rdf:` and `rdfs:` vocabularies
4. D-Entailment: Giving Semantics to Datatypes
5. OWL Entailment: Giving Semantics to the `owl:` Vocabulary

Semantics of the owl:Vocabulary

Not part of the RDF Spec, separate document, for the moment, we will restrict ourselves to OWL Version 1 [Patel-Schneider *et al.*, 2004].

- OWL adds the possibility to add some Description Logics (DL) axioms (TBox, ABox) to RDF ($SHOIN(\mathcal{D})$).
- OWL defines a mapping how to write such DL axioms in RDF.
- Defines two semantics:
 - **Direct Model-Theoretic Semantics:** DL-style model-theoretic semantics, for the syntactic subset of OWL which corresponds to DL
 - **RDF-Compatible semantics:** refining D-interpretations by further semantic conditions

OWL DL in two slides: 1/2

Expressing property characteristics:

OWL property axioms as RDF triples	DL syntax	FOL short representation
$P \text{ rdfs:domain } C .$	$\top \sqsubseteq \forall P^- . C$	$\forall x, y. P(x, y) \supset C(x)$
$P \text{ rdfs:range } C .$	$\top \sqsubseteq \forall P. C$	$\forall x, y. P(x, y) \supset C(y)$
$P \text{ owl:inverseOf } P_0 .$	$P \equiv P_0^-$	$\forall x, y. P(x, y) \equiv P_0(y, x)$
$P \text{ rdf:type owl:SymmetricProperty.}$	$P \equiv P^-$	$\forall x, y. P(x, y) \equiv P(y, x)$
$P \text{ rdf:type owl:FunctionalProperty.}$	$\top \sqsubseteq \leq 1P$	$\forall x, y, z. P(x, y) \wedge P(x, z) \supset y = z$
$P \text{ rdf:type owl:InverseFunctionalProperty.}$	$\top \sqsubseteq \leq 1P^-$	$\forall x, y, z. P(x, y) \wedge P(z, y) \supset x = z$
$P \text{ rdf:type owl:TransitiveProperty.}$	$P^+ \sqsubseteq P$	$\forall x, y, z. P(x, y) \wedge P(y, z) \supset P(x, z)$

OWL DL in two slides: 1/2

Expressing property characteristics:

OWL property axioms as RDF triples	DL syntax	FOL short representation
$P \text{ rdfs:domain } C .$	$\top \sqsubseteq \forall P^- . C$	$\forall x, y. P(x, y) \supset C(x)$
$P \text{ rdfs:range } C .$	$\top \sqsubseteq \forall P . C$	$\forall x, y. P(x, y) \supset C(y)$
$P \text{ owl:inverseOf } P_0 .$	$P \equiv P_0^-$	$\forall x, y. P(x, y) \equiv P_0(y, x)$
$P \text{ rdf:type owl:SymmetricProperty.}$	$P \equiv P^-$	$\forall x, y. P(x, y) \equiv P(y, x)$
$P \text{ rdf:type owl:FunctionalProperty.}$	$\top \sqsubseteq \leq 1P$	$\forall x, y, z. P(x, y) \wedge P(x, z) \supset y = z$
$P \text{ rdf:type owl:InverseFunctionalProperty.}$	$\top \sqsubseteq \leq 1P^-$	$\forall x, y, z. P(x, y) \wedge P(z, y) \supset x = z$
$P \text{ rdf:type owl:TransitiveProperty.}$	$P^+ \sqsubseteq P$	$\forall x, y, z. P(x, y) \wedge P(y, z) \supset P(x, z)$

Expressing complex class descriptions:

OWL complex class descriptions*	DL syntax	FOL short representation
owl:Thing	\top	$x = x$
owl:Nothing	\perp	$\neg x = x$
owl:intersectionOf ($C_1 \dots C_n$)	$C_1 \sqcap \dots \sqcap C_n$	$C_1(x) \wedge \dots \wedge C_n(x)$
owl:unionOf ($C_1 \dots C_n$)	$C_1 \sqcup \dots \sqcup C_n$	$C_1(x) \vee \dots \vee C_n(x)$
owl:complementOf (C)	$\neg C$	$\neg C(x)$
owl:oneOf ($o_1 \dots o_n$)	$\{o_1, \dots, o_n\}$	$x = o_1 \vee \dots \vee x = o_n$
owl:restriction ($P \text{ owl:someValuesFrom } (C)$)	$\exists P . C$	$\exists y. P(x, y) \wedge C(y)$
owl:restriction ($P \text{ owl:allValuesFrom } (C)$)	$\forall P . C$	$\forall y. P(x, y) \supset C(y)$
owl:restriction ($P \text{ owl:value } (o)$)	$\exists P . \{o\}$	$P(x, o)$
owl:restriction ($P \text{ owl:minCardinality } (n)$)	$\geq nP$	$\exists y_1 \dots y_n . \bigwedge_{k=1}^n P(x, y_k) \wedge \bigwedge_{i < j} y_i \neq y_j$
owl:restriction ($P \text{ owl:maxCardinality } (n)$)	$\leq nP$	$\forall y_1 \dots y_{n+1} . \bigwedge_{k=1}^{n+1} P(x, y_k) \supset \bigvee_{i < j} y_i = y_j$

*For reasons of legibility, we use a variant of the OWL abstract syntax [Patel-Schneider et al., 2004] in this table.

OWL DL in two slides: 2/2

Relating Class descriptions:

 $C_1 \text{ rdfs:subClassOf } C_2$
 $C_1 \sqsubseteq C_2$
 $C_1 \text{ owl:equivalentClass } C_2$
 $C_1 \equiv C_2$
 $C_1 \text{ owl:disjointWith } C_2$
 $C_1 \sqcap C_2 \sqsubseteq \perp$

Relating individuals:

 $o_1 \text{ owl:sameAs } o_1$
 $o_1 = o_2$
 $o_1 \text{ owl:differentFrom } o_2$
 $o_1 \neq o_2$

OWL DL in two slides: 2/2

Relating Class descriptions:

 $C_1 \text{ rdfs:subClassOf } C_2$
 $C_1 \text{ owl:equivalentClass } C_2$
 $C_1 \text{ owl:disjointWith } C_2$
 $C_1 \sqsubseteq C_2$
 $C_1 \equiv C_2$
 $C_1 \sqcap C_2 \sqsubseteq \perp$

Relating individuals:

 $o_1 \text{ owl:sameAs } o_2$
 $o_1 \text{ owl:differentFrom } o_2$
 $o_1 = o_2$
 $o_1 \neq o_2$

Let's look into some examples of

- How to write OWL in RDF.
- what it means
- possible problems

How to write OWL in RDF – A simple ontology about reviewers:

- **Properties:** title, isAuthorOf, publishedIn, etc.
- **Classes:** Senior, Paper, Publication, etc.
- **Relations:**
 - *A Publication is a Paper which has been published* (subclass + existential condition on property)
 - *isAuthorOf is the opposite of Dublin Core's dc:creator Property*²
 - *A Senior researcher is a foaf:Person who isAuthorOf 10+ Publications* (subclass + condition on cardinality)
 - *Each item can be publishedIn at most one venue* (functional property)
 -
 -
 -

²reuse of external ontologies!

OWL Example: A simple ontology about reviewers: in DL

$\exists ex:title.\top \sqsubseteq ex:Paper$ (i)

$\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)

$ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)

$ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)

$\top \sqsubseteq \leq 1 \ ex:publishedIn^{\neg}$ (v)

$ex:Senior \equiv foaf:Person \sqcap \geq 10 \ ex:isAuthorOf \sqcap$ (vi)

$\exists ex:isAuthorOf.ex:Publication$

$ex:Club100 \equiv foaf:Person \sqcap \geq 100 \ ex:isAuthorOf$ (vii)

OWL Example: A simple ontology about reviewers: in OWL/RDF

■ $\exists ex:title.\top \sqsubseteq ex:Paper$

(i)

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.T \sqsubseteq ex:Paper$
[a owl:Restriction; owl:onProperty ex:title; owl:someValuesFrom owl:Thing]
rdfs:subclassOf ex:Paper .

(i)

OWL Example: A simple ontology about reviewers: in OWL/RDF

■ $\exists ex:title.\top \sqsubseteq ex:Paper$

(i)

```
ex:title rdfs:domain dc:creator .
```

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`
- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`
- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)
`ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`
- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)
`ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`
- $\top \sqsubseteq \leq 1 ex:publishedIn^{\neg}$ (v)

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`
- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)
`ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`
- $\top \sqsubseteq \leq 1 ex:publishedIn^{\neg}$ (v)
`owl:Thing rdfs:subclassOf [a owl:restriction; on Property ex:publishedIn; owl:maxCardinality 1] .`

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`
- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)
`ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`
- $\top \sqsubseteq \leq 1 ex:publishedIn^{\neg}$ (v)
`ex:publishedIn a owl:FunctionalProperty .`

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`
- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)
`ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`
- $\top \sqsubseteq \leq 1 ex:publishedIn^{\neg}$ (v)
`ex:publishedIn a owl:FunctionalProperty .`
- $ex:Senior \equiv foaf:Person \sqcap \geq 10 ex:isAuthorOf \sqcap \exists ex:isAuthorOf.ex:Publication$ (vi)

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`
- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)
`ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`
- $\top \sqsubseteq \leq 1 ex:publishedIn^{\neg}$ (v)
`ex:publishedIn a owl:FunctionalProperty .`
- $ex:Senior \equiv foaf:Person \sqcap \geq 10 ex:isAuthorOf \sqcap \exists ex:isAuthorOf.ex:Publication$ (vi)
`ex:Senior owl:intersectionOf (foaf:Person [a owl:Restriction; owl:onProperty ex:isAuthorOf ; owl:minCardinality 10] [a owl:Restriction; owl:onProperty ex:isAuthorOf ; owl:someValuesFrom ex:Publication]) .`

OWL Example: A simple ontology about reviewers: in OWL/RDF

- $\exists ex:title.\top \sqsubseteq ex:Paper$ (i)
`ex:title rdfs:domain dc:creator .`
- $\exists ex:title^{\neg}.\top \sqsubseteq xsd:string$ (ii)
`ex:title rdfs:range xs:string .`
- $ex:isAuthorOf^{\neg} \equiv dc:creator$ (iii)
`ex:isAuthorOf owl:inverseOf dc:creator .`
- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn.\top$ (iv)
`ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`
- $\top \sqsubseteq \leq 1 ex:publishedIn^{\neg}$ (v)
`ex:publishedIn a owl:FunctionalProperty .`
- $ex:Senior \equiv foaf:Person \sqcap \geq 10 ex:isAuthorOf \sqcap \exists ex:isAuthorOf.ex:Publication$ (vi)
`ex:Senior owl:intersectionOf (foaf:Person [a owl:Restriction; owl:onProperty ex:isAuthorOf ; owl:minCardinality 10] [a owl:Restriction; owl:onProperty ex:isAuthorOf ; owl:someValuesFrom ex:Publication]) .`
- (vii) $ex:Club100 \equiv foaf:Person \sqcap \geq 100 ex:isAuthorOf$
Left for exercises! Check here [Patel-Schneider *et al.*, 2004, Section 4]

OWL Example: A simple ontology about reviewers: What does it mean?

As we have seen, all these axioms boil down to first-order formulas, e.g.

- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn. \top$ (iv)
- `ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`

Amounts to:

$$\forall x. Publication(x) \equiv Paper(x) \wedge \exists y. publishedIn(x, y)$$

OWL Example: A simple ontology about reviewers: What does it mean?

As we have seen, all these axioms boil down to first-order formulas, e.g.

- $ex:Publication \equiv ex:Paper \sqcap \exists ex:publishedIn. \top$ (iv)
- `ex:Publication owl:intersectionOf (ex:Paper [a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1]) .`

Amounts to:

$$\forall x. Publication(x) \equiv Paper(x) \wedge \exists y. publishedIn(x, y)$$

Recall, so far, when we wrote first-order, we always used a predicate *triple*:

$$\forall x. triple(x, a, ex:Publication) \equiv triple(x, a, ex:Publication) \wedge \exists y. triple(x, ex:publishedIn, y)$$

Possible Problems (differences between OWL DL and OWL Full):

- classes/properties used as instances and vice versa (called “meta-modelling”)
- “faulty” RDF encoding of OWL.

⇒ Strictly speaking, if any of this occurs, I can't use my DL reasoner anymore, or at least it will be incomplete. :-)

Possible Problems – faulty OWL/RDF

```
ex:Publication owl:intersectionOf ( ex:Paper [ a owl:Restriction;  
owl:onProperty ex:publishedIn ; owl:minCardinality 1 ] ) .
```

Possible Problems – faulty OWL/RDF

```
ex:Publication owl:intersectionOf _:b1 .
_:b1 rdf:first ex:Paper ; rdf:rest _:b2 .
_:b2 rdf:first _:b3 ; rdf:rest rdf:nil .
_:b3 a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1 .
```

Possible Problems – faulty OWL/RDF

```
ex:Publication owl:intersectionOf _:b1 .
_:b1 rdf:first ex:Paper ; rdf:rest _:b2 .
_:b2 rdf:first _:b3 ; rdf:rest rdf:nil .
_:b3 a owl:Restriction; owl:onProperty ex:publishedIn ; owl:minCardinality 1 .
_:b1 rdfs:first _:b1 .
```

Has no meaning in the DL reading! One could just ignore such statements.

Possible Problems – “meta-modelling”

```
ex:Publication owl:intersectionOf ( ex:Paper [ a owl:Restriction;  
owl:onProperty ex:publishedIn ; owl:minCardinality 1 ] ) .
```

Possible Problems – “meta-modelling”

```
ex:Publication owl:intersectionOf ( ex:Paper [ a owl:Restriction;  
owl:onProperty ex:publishedIn ; owl:minCardinality 1 ] ) .
```

```
ex:Paper owl:sameAs ex:publishedIn .
```

Possible Problems – “meta-modelling”

```
ex:Publication owl:intersectionOf ( ex:Paper [ a owl:Restriction;  
owl:onProperty ex:publishedIn ; owl:minCardinality 1 ] ) .
```

```
ex:Paper owl:sameAs ex:publishedIn .
```

Unary/Binary first-order translation doesn't work anymore:

$$\forall x. Publication(x) \equiv Paper(x) \wedge \exists y. publishedIn(x, y) \\ \wedge Publication = publishedIn$$

Ouch! This is no longer first-order! And outside OWL DL...

Possible Problems – “meta-modelling”

```
ex:Publication owl:intersectionOf ( ex:Paper [ a owl:Restriction;
owl:onProperty ex:publishedIn ; owl:minCardinality 1 ] ) .
```

```
ex:Paper owl:sameAs ex:publishedIn .
```

Unary/Binary first-order translation doesn't work anymore:

$$\forall x. Publication(x) \equiv Paper(x) \wedge \exists y. publishedIn(x, y) \\ \wedge Publication = publishedIn$$

Ouch! This is no longer first-order! And outside OWL DL...

triple encoding somewhat more stable here:

$$\forall x. triple(x, a, ex:Publication) \equiv \\ triple(x, a, ex:Publication) \wedge \exists y. triple(x, ex:publishedIn, y) \\ \wedge triple(ex:Publication, owl:sameAs, ex:publishedIn)$$

With the latter, I could still use a first-order reasoner for OWL Full, however, becomes undecidable in the general case.

Possible decidable extensions for meta modelling include “punning” in OWL2, or adding meta-reasoning facilities, cf. [Motik, 2007][Glimm *et al.*, 2010]

Can I do OWL rules-based FWD-chaining inference as before?

Can I do OWL rules-based FWD-chaining inference as before?

- with some caveats, yes. . .
- inherently incomplete. . .
- . . . but some interesting subset of sound inferences might be covered

Can I do OWL rules-based FWD-chaining inference as before?

Can I do OWL rules-based FWD-chaining inference as before?

- with some caveats, yes. . .
- inherently incomplete. . .
- . . . but some interesting subset of sound inferences might be covered

OWL 2 RL. . .

Can I do OWL rules-based FWD-chaining inference as before?

Can I do OWL rules-based FWD-chaining inference as before?

- with some caveats, yes. . .
- inherently incomplete. . .
- . . . but some interesting subset of sound inferences might be covered

OWL 2 RL. . .

Can I do OWL use other scalable techniques such as query rewriting?

- Yes! **OWL 2 QL**

→ More on that in Lecture 5 or 6 (OWL2 and OWL2 profiles)!

Recommended Reading

- [ter Horst, 2005], RDF, RDFS and parts of OWL entailment in Rules.
- [Muñoz *et al.*, 2007], minimal sets of complete inference rules for parts of the RDFS vocabulary.

A bit more tough reading (specs), but also recommended:

- [Hayes, 2004, Sections 3 onwards], official RDF semantics specification.

Possible topics for the final presentations

- RDF1.1 WG – status semantics, changes, semantics for named graphs, etc.
- Linked Data Platform WG –
http://www.w3.org/2012/ldp/wiki/Main_Page
- SPARQL1.1 spec parts which we don't cover in detail (e.g. update, federated queries, etc.)
- OWL2 and meta-modeling [Motik, 2007; Glimm *et al.*, 2010]
- SKOS
- RDFa
- `schema.org`
- Berlin SPARQL Benchmark, latest edition, see <http://lists.w3.org/Archives/Public/semantic-web/2013Apr/0237.html>
- ... (to be extended, own topics suggestions welcome!)



Jos de Bruijn and Stijn Heymans.

Logical foundations of (e)RDF(S): Complexity and reasoning.

In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, number 4825 in Lecture Notes in Computer Science, pages 86–99, Busan, Korea, November 2007. Springer.



Birte Glimm, Sebastian Rudolph, and Johanna Völker.

Integrated metamodeling and diagnosis in owl 2.

In *International Semantic Web Conference (ISWC2010)*, pages 257–272, 2010.



P. Hayes.

RDF semantics, 2004.

<http://www.w3.org/TR/rdf-mt/>.



Giovambattista Ianni, Alessandra Martello, Claudio Panetta, and Giorgio Terracina.

Efficiently querying RDF(S) ontologies with Answer Set Programming.

Journal of Logic and Computation (Special issue), 2009.

Forthcoming.



Alejandro Mallea, Marcelo Arenas, Aidan Hogan, and Axel Polleres.

On Blank Nodes.

In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*,

volume 7031 of *Lecture Notes in Computer Science (LNCS)*, Bonn, Germany, October

2011. Springer.



Boris Motik.

On the Properties of Metamodeling in OWL.

Journal of Logic and Computation, 17(4):617–637, 2007.



Sergio Muñoz, Jorge Pérez, and Claudio Gutiérrez.

Minimal deductive systems for rdf.

In *ESWC*, pages 53–67, 2007.



Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks.

OWL Web Ontology Language Semantics and Abstract Syntax, February 2004.

W3C Recommendation.



Herman 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(2), July 2005.



Xml schema part 2: Datatypes, May 2001.

W3C Recommendation, available at <http://www.w3.org/TR/xmlschema-2/>.