Unit 3 – Higher Entailment regimes: Semantics of RDF(S), Datatypes, and OWL

Axel Polleres

DERI, National University of Ireland, Galway

VU 184.268 Technologien für das Semantische Web

Unit Outline

1. RDF(S) Entailment: Giving semantics to the rdf: and rdfs: Vocabulary
2. D-Entailment: Giving Semantics to Datatypes
3. OWL Entailment: Giving Semantics to the owl: Vocabulary

So far, we can:

- publish machine readable data (RDF)
- query machine readable data (SPARQL)

So far, we can:

- publish machine readable data (RDF)
- query machine readable data (SPARQL)

What's missing?

- The ability to formalize the structure of this data, or rather...
- ...The ability to infer new data from this structural information
- ...That is: a semantics for the **rdf:**, **rdfs:** and **owl:** vocabularies

So far, we can:

- publish machine readable data (RDF)
- query machine readable data (SPARQL)

What's missing?

- The ability to formalize the structure of this data, or rather...
- ...The ability to infer new data from this structural information
- ...That is: a semantics for the **rdf:**, **rdfs:** and **owl:** vocabularies

Finally, how will we query inferred data?

Unit Outline

1. RDF(S) Entailment: Giving semantics to the rdf: and rdfs: Vocabulary
2. D-Entailment: Giving Semantics to Datatypes
3. 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 semntics. The semantics doesn't make any strong restrictions on the grayed out vocabulary.

Collection vocabulary – Example

Container vocab intended to be used to write lists, e.g.

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

Collection vocabulary – Example

Container vocab 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

Container vocab 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 vocab 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:hasLecture [  a rdf:Alt;
                                 rdf:_1 rdf:axel
                                 rdf:_2 ex:thomas ] .
```

Comes usually from RDF/XML's `<li>...</li>`.

Container vocabulary – Example

Container vocab 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:hasLecture [  a rdf:Alt;
                                 rdf:_1 rdf:axel
                                 rdf:_2 ex:thomas ] .
```

Comes usually from RDF/XML's `<li>...</li>`.

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.

Purely speculative statement by the lecturer

Intuitively, all these parts of the vocabulary have been left without fixed semantics for the moment... may be changed by later standards? e.g. "Reification-Entailment", etc.

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 folllowing set of axiomatic triples A_{rdf}:

RDF semantic conditions.

x is in IP if and only if <x, I(`rdf:Property`)> is in IEXT(I(`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(`rdf:type`)) contains <IL("xxx"`^^rdf:XMLLiteral`), I(`rdf:XMLLiteral`)>
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(`rdf:type`)) does not contain <IL("xxx"`^^rdf:XMLLiteral`), I(`rdf:XMLLiteral`)>

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 folllowing set of axiomatic triples A_{rdf}:

RDF semantic conditions.

x is in IP if and only if <x, I(rdf:Property)> is in IEXT(I(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(rdf:type)) contains <IL("xxx"^^rdf:XMLLiteral), I(rdf:XMLLiteral)>
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(rdf:type)) does not contain <IL("xxx"^^rdf:XMLLiteral), I(rdf:XMLLiteral)>

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-satisaction, 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 a rdf:Property`}.

Particularly, note that the first axiomatic triple is redundant!

rdf-interpretation

Some consequences:

Proposition 1

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

Particularly, note that the first axiomatic triple is redundant!

rdf-interpretation

Some consequences:

Proposition 1

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

rdf-interpretation

Some consequences:

Proposition 1

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

rdf-interpretation

Some consequences:

Proposition 1

If `s p o.` $\in G$ then $G \models_{rdf}$ {`p 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 p "`xyz`"^^rdf:XMLLiteral.` $\in G$, such that $xyz \in L_{XML}$ then $G \models_{rdf}$ {`[] 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 p o.` $\in Cl_{rdf}(G)$ then add `p a rdf:Property.`
- If `s p "`xyz`"^^rdf:XMLLiteral.` $\in Cl_{rdf}(G)$ and $xyz \in L_{XML}$ then add `_:xyz a rdf:XMLLiteral.` 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 .` $\Leftarrow s\ p$ "xyz"^^`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:

- $\forall s, p, o(triple(p, \texttt{rdf:type}, \texttt{rdf:Property}) \leftarrow triple(s, p, o))$
- $\forall s, p \exists b(triple(b, \texttt{rdf:type}, \texttt{rdf:XMLLiteral}) \leftarrow triple(s, p, \texttt{"}xyz\texttt{"}\verb|^^|\texttt{rdf:XMLLiteral}))$

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,O).`
- `triple(`$\mathtt{b}_{xyz}$`,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 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 `rdf:type 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 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 `rdf:type 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 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 `rdf:type 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 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 `rdf:type 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 fairly uninteresting...

So what?

Now that was 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:Propery`
- we have a way to express that XML Literals are `rdf:XMLLiterals`

So what?

Now that was 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:Propery`
- we have a way to express that XML Literals are `rdf:XMLLiterals`

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

The rdf: 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 conainer 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” semantics[1]

[1] for 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/foafOntology.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 $\langle x, y\rangle \in IEXT(I(\texttt{rdf:type}))$.

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 $\langle x, y\rangle \in IEXT(I(\texttt{rdf:type}))$.

For brevity, we also define:

$ICEXT$

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

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 $\langle x, y \rangle \in IEXT(I(\texttt{rdf:type}))$.

For brevity, we also define:

$ICEXT$

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

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

rdfs-interpretations: Semantic conditions

x is in ICEXT(y) if and only if <x,y> is in IEXT(I(`rdf:type`))

IC = ICEXT(I(`rdfs:Class`))

IR = ICEXT(I(`rdfs:Resource`))

LV = ICEXT(I(`rdfs:Literal`))

If <x,y> is in IEXT(I(`rdfs:domain`)) and <u,v> is in IEXT(x) then u is in ICEXT(y)

If <x,y> is in IEXT(I(`rdfs:range`)) and <u,v> is in IEXT(x) then v is in ICEXT(y)

IEXT(I(`rdfs:subPropertyOf`)) is transitive and reflexive on IP

If <x,y> is in IEXT(I(`rdfs:subPropertyOf`)) then x and y are in IP and IEXT(x) is a subset of IEXT(y)

If x is in IC then <x, I(`rdfs:Resource`)> is in IEXT(I(`rdfs:subClassOf`))

If <x,y> is in IEXT(I(`rdfs:subClassOf`)) then x and y are in IC and ICEXT(x) is a subset of ICEXT(y)

IEXT(I(`rdfs:subClassOf`)) is transitive and reflexive on IC

If x is in ICEXT(I(`rdfs:ContainerMembershipProperty`)) then:
< x, I(`rdfs:member`)> is in IEXT(I(`rdfs:subPropertyOf`))

If x is in ICEXT(I(`rdfs:Datatype`)) then <x, I(`rdfs:Literal`)> is in IEXT(I(`rdfs:subClassOf`))

rdfs-interpretations: Semantic conditions

x is in ICEXT(y) if and only if <x,y> is in IEXT(I(`rdf:type`))

IC = ICEXT(I(`rdfs:Class`))

IR = ICEXT(I(`rdfs:Resource`))

LV = ICEXT(I(`rdfs:Literal`))

o `a rdf:Class .` $\Leftarrow$ s `a` o `.`
`[] a` s `.` $\Leftarrow$ s `a rdf:Class .`
s `a rdf:Resource .` $\Leftarrow$ $s\ p\ o$ `.`
p `a rdf:Resource .` $\Leftarrow$ $s\ p\ o$ `.`
o `a rdf:Resource .` $\Leftarrow$ $s\ p\ o$ `.` $o \notin L$
`_:l a rdf:Literal .` $\Leftarrow$ $s\ p\ l$. $l \in L$
$s\ p$ `_:l .` $\Leftarrow$ $s\ p\ l$ `.` $l \in L$

If <x,y> is in IEXT(I(`rdfs:domain`)) and <u,v> is in IEXT(x) then u is in ICEXT(y)

If <x,y> is in IEXT(I(`rdfs:range`)) and <u,v> is in IEXT(x) then v is in ICEXT(y)

IEXT(I(`rdfs:subPropertyOf`)) is transitive and reflexive on IP

If <x,y> is in IEXT(I(`rdfs:subPropertyOf`)) then x and y are in IP and IEXT(x) is a subset of IEXT(y)

If x is in IC then <x, I(`rdfs:Resource`)> is in IEXT(I(`rdfs:subClassOf`))

If <x,y> is in IEXT(I(`rdfs:subClassOf`)) then x and y are in IC and ICEXT(x) is a subset of ICEXT(y)

IEXT(I(`rdfs:subClassOf`)) is transitive and reflexive on IC

If x is in ICEXT(I(`rdfs:ContainerMembershipProperty`)) then:
< x, I(`rdfs:member`)> is in IEXT(I(`rdfs:subPropertyOf`))

If x is in ICEXT(I(`rdfs:Datatype`)) then <x, I(`rdfs:Literal`)> is in IEXT(I(`rdfs:subClassOf`))

rdfs-interpretations: Semantic conditions

x is in ICEXT(y) if and only if <x,y> is in IEXT(I(`rdf:type`))

IC = ICEXT(I(`rdfs:Class`))

IR = ICEXT(I(`rdfs:Resource`))

LV = ICEXT(I(`rdfs:Literal`))

o a rdf:Class . $\Leftarrow$ s a o .
[] a s . $\Leftarrow$ s a rdf:Class .
s a rdf:Resource . $\Leftarrow$ s p o .
p a rdf:Resource . $\Leftarrow$ s p o .
o a rdf:Resource . $\Leftarrow$ s p o . $o \notin L$
_:l a rdf:Literal . $\Leftarrow$ s p l. $l \in L$
s p _:l . $\Leftarrow$ s p l . $l \in L$

If <x,y> is in IEXT(I(`rdfs:domain`)) and <u,v> is in IEXT(x) then u is in ICEXT(y)

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

If <x,y> is in IEXT(I(`rdfs:range`)) and <u,v> is in IEXT(x) then v is in ICEXT(y)

IEXT(I(`rdfs:subPropertyOf`)) is transitive and reflexive on IP

If <x,y> is in IEXT(I(`rdfs:subPropertyOf`)) then x and y are in IP and IEXT(x) is a subset of IEXT(y)

If x is in IC then <x, I(`rdfs:Resource`)> is in IEXT(I(`rdfs:subClassOf`))

If <x,y> is in IEXT(I(`rdfs:subClassOf`)) then x and y are in IC and ICEXT(x) is a subset of ICEXT(y)

IEXT(I(`rdfs:subClassOf`)) is transitive and reflexive on IC

If x is in ICEXT(I(`rdfs:ContainerMembershipProperty`)) then:
< x, I(`rdfs:member`)> is in IEXT(I(`rdfs:subPropertyOf`))

If x is in ICEXT(I(`rdfs:Datatype`)) then <x, I(`rdfs:Literal`)> is in IEXT(I(`rdfs:subClassOf`))

rdfs-interpretations: Semantic conditions

x is in ICEXT(y) if and only if <x,y> is in IEXT(I(`rdf:type`))

IC = ICEXT(I(`rdfs:Class`))

IR = ICEXT(I(`rdfs:Resource`))

LV = ICEXT(I(`rdfs:Literal`))

o `a rdf:Class .` $\Leftarrow$ s `a` o `.`
`[] a` s `.` $\Leftarrow$ s `a rdf:Class .`
s `a rdf:Resource .` $\Leftarrow$ $s\ p\ o$ `.`
p `a rdf:Resource .` $\Leftarrow$ $s\ p\ o$ `.`
o `a rdf:Resource .` $\Leftarrow$ $s\ p\ o$ `.` $o \notin L$
`_:l a rdf:Literal .` $\Leftarrow$ $s\ p\ l$. $l \in L$
$s\ p$ `_:l .` $\Leftarrow$ $s\ p\ l$ `.` $l \in L$

If <x,y> is in IEXT(I(`rdfs:domain`)) and <u,v> is in IEXT(x) then u is in ICEXT(y)

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

If <x,y> is in IEXT(I(`rdfs:range`)) and <u,v> is in IEXT(x) then v is in ICEXT(y)

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

IEXT(I(`rdfs:subPropertyOf`)) is transitive and reflexive on IP

If <x,y> is in IEXT(I(`rdfs:subPropertyOf`)) then x and y are in IP and IEXT(x) is a subset of IEXT(y)

If x is in IC then <x, I(`rdfs:Resource`)> is in IEXT(I(`rdfs:subClassOf`))

If <x,y> is in IEXT(I(`rdfs:subClassOf`)) then x and y are in IC and ICEXT(x) is a subset of ICEXT(y)

IEXT(I(`rdfs:subClassOf`)) is transitive and reflexive on IC

If x is in ICEXT(I(`rdfs:ContainerMembershipProperty`)) then:
< x, I(`rdfs:member`)> is in IEXT(I(`rdfs:subPropertyOf`))

If x is in ICEXT(I(`rdfs:Datatype`)) then <x, I(`rdfs:Literal`)> is in IEXT(I(`rdfs:subClassOf`))

rdfs-interpretations: Semantic conditions

x is in ICEXT(y) if and only if <x,y> is in IEXT(I(`rdf:type`))

IC = ICEXT(I(`rdfs:Class`))

IR = ICEXT(I(`rdfs:Resource`))

LV = ICEXT(I(`rdfs:Literal`))

- o `a rdf:Class .` $\Leftarrow s$ `a` o `.`
- `[] a` s `.` $\Leftarrow s$ `a rdf:Class .`
- s `a rdf:Resource .` $\Leftarrow s\ p\ o$ `.`
- p `a rdf:Resource .` $\Leftarrow s\ p\ o$ `.`
- o `a rdf:Resource .` $\Leftarrow s\ p\ o$ `.` $\quad o \notin L$
- `_:l a rdf:Literal .` $\Leftarrow s\ p\ l$. $\quad l \in L$
- $s\ p$ `_:l .` $\Leftarrow s\ p\ l$ `.` $\quad l \in L$

If <x,y> is in IEXT(I(`rdfs:domain`)) and <u,v> is in IEXT(x) then u is in ICEXT(y)

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

If <x,y> is in IEXT(I(`rdfs:range`)) and <u,v> is in IEXT(x) then v is in ICEXT(y)

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

IEXT(I(`rdfs:subPropertyOf`)) is transitive and reflexive on IP

If <x,y> is in IEXT(I(`rdfs:subPropertyOf`)) then x and y are in IP and IEXT(x) is a s

If x is in IC then <x, I(`rdfs:Resource`)> is in IEXT(I(`rdfs:subClassOf`))

If <x,y> is in IEXT(I(`rdfs:subClassOf`)) then x and y are in IC and ICEXT(x) is a sul

IEXT(I(`rdfs:subClassOf`)) is transitive and reflexive on IC

If x is in ICEXT(I(`rdfs:ContainerMembershipProperty`)) then:
<x, I(`rdfs:member`)> is in IEXT(I(`rdfs:subPropertyOf`))

If x is in ICEXT(I(`rdfs:Datatype`)) then <x, I(`rdfs:Literal`)> is in IEXT(I(`rdfs:subCl`

- 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. $s\ p\ o$. $\quad q \notin BL$
- s `subClassOf Resource .` $\Leftarrow s$ `a Class .`
- s `a Class .` $\Leftarrow s$ `subClassOf` o `.`
- o `a Class .` $\Leftarrow s$ `subClassOf` o `.` $\quad o \notin L$
- 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}

```
rdf:type            rdfs:domain rdfs:Resource;
                    rdfs:range rdfs:Class .
rdfs:domain         rdfs:domain rdf:Property;
                    rdfs:range rdfs:Class .
rdfs:range          rdfs:domain rdf:Property;
                    rdfs:range rdfs:Class .
rdfs:subPropertyOf  rdfs:domain rdf:Property;
                    rdfs:range  rdf:Property .
rdfs:subClassOf     rdfs:domain rdfs:Class;
                    rdfs:range  rdfs:Class .

rdfs:member       rdfs:domain rdfs:Resource;
                  rdfs:range rdfs:Resource .
rdfs:seeAlso      rdfs:domain rdfs:Resource;
                  rdfs:range rdfs:Resource .
rdfs:isDefinedBy  rdfs:domain rdfs:Resource;
                  rdfs:range rdfs:Resource .
rdf:value         rdfs:domain rdfs:Resource;
                  rdfs:range rdfs:Resource .

rdfs:comment      rdfs:domain rdfs:Resource;
                  rdfs:range rdfs:Literal .
rdfs:label        rdfs:domain rdfs:Resource;
                  rdfs:range rdfs:Literal .

rdf:first         rdfs:domain rdf:List;
                  rdfs:range rdfs:Resource .
rdf:rest          rdfs:domain rdf:List .
                  rdfs:range rdf:List .
```

```
rdf:subject      rdfs:domain rdf:Statement;
                 rdfs:range rdfs:Resource .
rdf:predicate    rdfs:domain rdf:Statement;
                 rdfs:range rdfs:Resource .
rdf:object       rdfs:domain rdf:Statement;
                 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 .
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
rdf:_2 rdfs:domain rdfs:Resource .
rdf:_2 rdfs:range rdfs:Resource .
...
```

rdfs-interpretations: Axiomatic Triples

A_{rdfs} – "somewhat redundant triples" (let's call that A^{-}_{rdfs})

```
rdf:type            rdfs:range rdfs:Class .
rdfs:domain         rdfs:range rdfs:Class .
rdfs:range          rdfs:range rdfs:Class .

rdfs:subClassOf     rdfs:domain rdfs:Class;
                    rdfs:range  rdfs:Class .

rdfs:comment     rdfs:range rdfs:Literal .
rdfs:label       rdfs:range rdfs:Literal .

rdf:first        rdfs:domain rdf:List .
rdf:rest         rdfs:domain rdf:List .
                 rdfs:range rdf:List .
```

```
rdf:subject      rdfs:domain rdf:Statement;
rdf:predicate    rdfs:domain rdf:Statement;
rdf:object       rdfs:domain rdf:Statement;

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:_2 rdf:type rdfs:ContainerMembershipProperty .
...
```

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} – "somewhat redundant triples" (let's call that A^{-}_{rdfs})

```
rdf:type          rdfs:range rdfs:Class .
rdfs:domain       rdfs:range rdfs:Class .
rdfs:range        rdfs:range rdfs:Class .

rdfs:subClassOf   rdfs:domain rdfs:Class;
                  rdfs:range  rdfs:Class .

rdfs:comment    rdfs:range rdfs:Literal .
rdfs:label      rdfs:range rdfs:Literal .

rdf:first       rdfs:domain rdf:List .
rdf:rest        rdfs:domain rdf:List .
                rdfs:range rdf:List .
```

```
rdf:subject      rdfs:domain rdf:Statement;
rdf:predicate    rdfs:domain rdf:Statement;
rdf:object       rdfs:domain rdf:Statement;

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:_2 rdf:type rdfs:ContainerMembershipProperty .
...
```

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 the last slide.

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 the last slide.

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 the last slide.

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

Spec entailment rules are – incomplete!

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

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

$s\ q\ o \Leftarrow p\ \mathtt{subPropertyOf}\ q.\quad s\ p\ o.$ $\qquad q \notin BL$

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.

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.

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

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

Inconsistency

Proposition 3

ANY Graph

- containing a triple s p `"`xyz`"^^rdf:XMLLiteral.` such that xyz is NOT a well-formed XML literal and
- allowing to infer the triple p `rdf:range rdf:XMLLiteral .`

is false in any rdfs-interpretation.

Inconsistency

Proposition 3

ANY Graph

- containing a triple s p "xyz"^^`rdf:XMLLiteral`. such that xyz is NOT a well-formed XML literal and
- allowing to infer the triple p `rdf:range rdf:XMLLiteral .`

is false in any rdfs-interpretation.

Why?

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

Inconsistency

Proposition 3

ANY Graph

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

is false in any rdfs-interpretation.

Why?

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

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

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

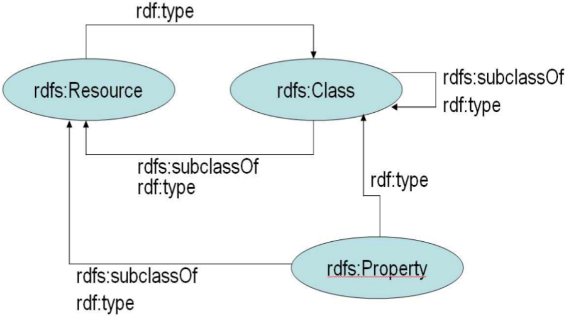
G_5 :

```
rdfs:Resource rdf:type rdfs:Class.
rdfs:Property rdf:type rdfs:Resource.
rdfs:Property rdf:subclassOf rdfs:Resource.
rdfs:Property rdf:type rdfs:Class.
rdfs:Class rdf:type rdfs:Resource.
rdfs:Class rdf:type rdfs:Class.
rdfs:Class rdf:subclassOf rdfs:Resource.
rdfs:Class rdf: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 `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. RDF(S) Entailment: Giving semantics to the rdf: and rdfs: Vocabulary
2. D-Entailment: Giving Semantics to Datatypes
3. 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:

XSD 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

D-interpretation

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

General semantic conditions for datatypes.

if <aaa,x> is in D then I(aaa) = x
if <aaa,x> is in D then ICEXT(x) is the value space of x and is a subset of LV
if <aaa,x> 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 <aaa,x> is in D then I(aaa) is in ICEXT(I(`rdfs:Datatype`))

D-interpretations

D-interpretation

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

General semantic conditions for datatypes.

if <aaa,x> is in D then I(aaa) = x
if <aaa,x> is in D then ICEXT(x) is the value space of x and is a subset of LV
if <aaa,x> 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 <aaa,x> 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 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-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 `xsds: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:

$s \ \ p \ \ L2V(l) \ \ . \Leftarrow s \ \ p \ \ l\text{^^}d \ \ .$ for l 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:

$s\ \ p\ \ L2V(l)\ \ .\ \Leftarrow s\ \ p\ \ l\hat{}\hat{}d\ \ .$ for l 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:

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

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

Unit Outline

1. RDF(S) Entailment: Giving semantics to the rdf: and rdfs: Vocabulary
2. D-Entailment: Giving Semantics to Datatypes
3. OWL Entailment: Giving Semantics to the owl: Vocabulary

Semantics of the owl:Vocabulary

Not part of the RDF Spec, separate document [Patel-Schneider *et al.*, 2004].

- OWL adds the possiblity to add some DL axioms (TBox, ABox) to RDF ($\mathcal{SHOIN}(\mathcal{D})$).
- OWL defines a rewriting how to write such DL axioms in RDF.
- OWL adds the possiblity to add some DL statements to an ontology.
- Defines two semantics:
    - DL-style model-theoretic semantics, for the syntactic subset of OWL which corresponds to DL
    - RDF compatible semantics

OWL DL in two slides: 1/2

Expressing property characteristics:

OWL property axioms as RDF triples	DL syntax	FOL short representation
P `rdfs:domain` C .	$\top \sqsubseteq \forall P^{-}.C$	$\forall x, y.P(x, y) \supset C(x)$
P `rdfs:range` C .	$\top \sqsubseteq \forall P.C$	$\forall x, y.P(x, y) \supset C(y)$
P `owl:inverseOf` P_0 .	$P \equiv P_0^{-}$	$\forall x, y.P(x, y) \equiv P_0(y, x)$
P `rdf:type owl:SymmetricProperty`.	$P \equiv P^{-}$	$\forall x, y.P(x, y) \equiv P(y, x)$
P `rdf:type owl:FunctionalProperty`.	$\top \sqsubseteq \leqslant 1P$	$\forall x, y, z.P(x, y) \wedge P(x, z) \supset y = z$
P `rdf:type owl:InverseFunctionalProperty`.	$\top \sqsubseteq \leqslant 1P^{-}$	$\forall x, y, z.P(x, y) \wedge P(z, y) \supset x = z$
P `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 `rdfs:domain` C .	$\top \sqsubseteq \forall P^{-}.C$	$\forall x, y.P(x, y) \supset C(x)$
P `rdfs:range` C .	$\top \sqsubseteq \forall P.C$	$\forall x, y.P(x, y) \supset C(y)$
P `owl:inverseOf` P_0 .	$P \equiv P_0^{-}$	$\forall x, y.P(x, y) \equiv P_0(y, x)$
P `rdf:type owl:SymmetricProperty`.	$P \equiv P^{-}$	$\forall x, y.P(x, y) \equiv P(y, x)$
P `rdf:type owl:FunctionalProperty`.	$\top \sqsubseteq \leqslant 1P$	$\forall x, y, z.P(x, y) \wedge P(x, z) \supset y = z$
P `rdf:type owl:InverseFunctionalProperty`.	$\top \sqsubseteq \leqslant 1P^{-}$	$\forall x, y, z.P(x, y) \wedge P(z, y) \supset x = z$
P `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`	$\bot$	$\neg x = x$
`owl:intersectionOf` $(C_1 \ldots C_n)$	$C_1 \sqcap \cdots \sqcap C_n$	$C_1(x) \wedge \cdots \wedge C_n(x)$
`owl:unionOf` $(C_1 \ldots C_n)$	$C_1 \sqcup \cdots \sqcup C_n$	$C_1(x) \vee \cdots \vee C_n(x)$
`owl:complementOf` (C)	$\neg C$	$\neg C(x)$
`owl:oneOf` $(o_1 \ldots o_n)$	$\{o_1, \ldots, o_n\}$	$x = o_1 \vee \cdots \vee x = o_n$
`owl:restriction` $(P$ `owl:someValuesFrom` $(C))$	$\exists P.C$	$\exists y.P(x, y) \wedge C(y)$
`owl:restriction` $(P$ `owl:allValuesFrom` $(C))$	$\forall P.C$	$\forall y.P(x, y) \supset C(y)$
`owl:restriction` $(P$ `owl:value` $(o))$	$\exists P.\{o\}$	$P(x, o)$
`owl:restriction` $(P$ `owl:minCardinality` $(n))$	$\geqslant nP$	$\exists y_1 \ldots y_n. \bigwedge_{k=1}^{n} P(x, y_k) \wedge \bigwedge_{i<j} y_j \neq y_j$
`owl:restriction` $(P$ `owl:maxCardinality` $(n))$	$\leqslant nP$	$\forall y_1 \ldots y_{n+1}. \bigwedge_{k=1}^{n+1} P(x, y_k) \supset \bigvee_{i<j} y_i = y_j$

*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 `rdfs:subClassOf` C_1	$C_1 \sqsubseteq C_2$
C_1 `owl:equivalentClass` C_2	$C_1 \equiv C_2$
C_1 `owl:disjointWith` C_2	$C_1 \sqcap C_2 \sqsubseteq \bot$

Relating individuals:

o_1 `owl:sameAs` o_1	$o_1 = o_2$
o_1 `owl:differentFrom` o_2	$o_1 \neq o_2$

OWL DL in two slides: 2/2

Relating Class descriptions:

C_1 `rdfs:subClassOf` C_1	$C_1 \sqsubseteq C_2$
C_1 `owl:equivalentClass` C_2	$C_1 \equiv C_2$
C_1 `owl:disjointWith` C_2	$C_1 \sqcap C_2 \sqsubseteq \bot$

Relating individuals:

o_1 `owl:sameAs` o_1	$o_1 = o_2$
o_1 `owl:differentFrom` 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)

$\vdots$

[2] reuse of external ontologies!

OWL Example: A simple ontology about reviewers: in DL

$$\exists ex{:}title.\top \sqsubseteq ex{:}Paper \tag{i}$$

$$\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string \tag{ii}$$

$$ex{:}isAuthorOf^{-} \equiv dc{:}creator \tag{iii}$$

$$ex{:}Publication \equiv ex{:}Paper \sqcap \exists ex{:}publishedIn.\top \tag{iv}$$

$$\top \sqsubseteq \leqslant 1 \; ex{:}publishedIn^{-} \tag{v}$$

$$\begin{aligned} ex{:}Senior \equiv foaf{:}Person \sqcap \geqslant 10 \; ex{:}isAuthorOf \sqcap \\ \exists ex{:}isAuthorOf.ex{:}Publication \end{aligned} \tag{vi}$$

$$ex{:}Club100 \equiv foaf{:}Person \sqcap \geqslant 100 \; ex{:}isAuthorOf \tag{vii}$$

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

- $\exists ex{:}title.\top \sqsubseteq ex{:}Paper$ (i)
  ```
  [a owl:Restriction; owl:onProperty ex:title; owl:someValuesFrom owl:Thing]
  rdfs:subclassOf ex:Paper .
  ```

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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (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 .`

- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 ] ) .`
- $\top \sqsubseteq \leqslant 1\ ex{:}publishedIn^{-}$ (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)

  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 ] ) .`
- $\top \sqsubseteq \leqslant 1\ ex{:}publishedIn^{-}$ (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 ] ) .`
- $\top \sqsubseteq \leqslant 1\ ex{:}publishedIn^{-}$ (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 ] ) .`
- $\top \sqsubseteq \leqslant 1\ ex{:}publishedIn^{-}$ (v)
  `ex:publishedIn a owl:FunctionalProperty .`
- *A `Senior` researcher is a `foaf:Person` who `isAuthorOf` 10+ `Publications`* (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 ] ) .`
- $\top \sqsubseteq \leqslant 1\ ex{:}publishedIn^{-}$ (v)
  `ex:publishedIn a owl:FunctionalProperty .`
- *A* `Senior` *researcher is a* `foaf:Person` *who* `isAuthorOf` *10+* `Publications` (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 .`
- $\exists ex{:}title^{-}.\top \sqsubseteq xsd{:}string$ (ii)
  `ex:title rdfs:range xs:string .`
- $ex{:}isAuthorOf^{-} \equiv dc{:}creator$ (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 ] ) .`
- $\top \sqsubseteq \leqslant 1 \; ex{:}publishedIn^{-}$ (v)
  `ex:publishedIn a owl:FunctionalProperty .`
- *A `Senior` researcher is a `foaf:Person` who `isAuthorOf` 10+ `Publications`* (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) **Left for exercises!**

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:

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

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 $triple$:

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

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

- classes/properties used as instances and vice versa (called "meta-modelling")
- "faulty" OWL/RDF

$\Rightarrow$ 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 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 ] ) .
```

exPaper owl:sameAs ex:publishedIn .

Possible Problems – "meta-modelling"

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

exPaper owl:sameAs ex:publishedIn .

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

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

$$\land Publication = publishedIn$$

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 ] ) .
```

exPaper 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:

$$\begin{aligned}&\forall x.triples(x, \mathtt{a}, \mathtt{ex:Publication}) \equiv \\ &\quad triples(x, \mathtt{a}, \mathtt{ex:Publication}) \wedge \exists y triple(x, \mathtt{ex:publishedIn}, y) \\ &\quad \wedge triple(\mathtt{ex:Publication}, \mathtt{owl:sameAs}, \mathtt{ex:publishedIn}\end{aligned}$$

With the latter, I could still use a first-order reasoner for OWL Full.

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

[ter Horst, 2005],[Hogan *et al.*, 2008], see also: `http://www.polleres.net/presentations/20081029saor_ISWC_btriples_challenge.pdf`

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.

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.

P. Hayes.
RDF semantics, 2004.
`http://www.w3.org/TR/rdf-mt/`.

Aidan Hogan, Andreas Harth, and Axel Polleres.
SAOR: Authoritative Reasoning for the Web.
In John Domingue and Chutiporn Anutariya, editors, *Proceedings of the 3rd Asian Semantic Web Conference (ASWC 2008)*, volume 5367 of *Lecture Notes in Computer Science* (*LNCS*), pages 76–90, Bankok, Thailand, December 2008.

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.

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/`.