
# COMP318

## Ontologies and Semantic Web

# RDF - Part 14


**Dr Valentina Tamma**

`V.Tamma@liverpool.ac.uk`

# Where were we

- RDF entailment rules
- RDFS entailment rules

# Bijection between literals and surrogate bonds

- **(gl inverse of lg)** If G contains the triple `:u :a _:n.` we can infer
  - `:u :a :l.` However:
  - this rule can be applied only when `_:n` identifies a bnode that was introduced earlier by weakening the literal `:l` via the rule lg
  - This inference rule is necessary to bring back a literal that has been substituted by a blank node using rule lg, then some other inference rule produced a triple with this blank node (`_:n`) in the object position, and rule gl can now be used to bring this literal back!
    - E.g. `:Murray atp:name “Andy Murray".`
      `:atp:name rdfs:range atp:PlayerName.`
    - Would entail `"Andy Murray" a atp:PlayerName.` which is problematic. Why?

# Bijection between literals and surrogate bonds

- The latter triple is not a valid RDF triple
  - a literal should not appear in the subject position!

- thus it will not be inferred (the domain of the ?v variable in the rdfs3 rule would prevent the inference). And so, to achieve the valid inference:
  - `_:AndyMurray a atp:PlayerName .`

    Requires the surrogate blank node (`_:AndyMurray`) to be used through the rule lg.
    - The inverse rule gl then allows surrogates to “travel” back as literals into the object position, though examples of such behaviour are not necessarily intuitive.

# Reasoning engines

- Systems that perform inference are often called reasoning engines or reasoners.
  - ***Reasoner engine***: a system that infers new information based on the contents of a knowledge base.
    - This can be accomplished using rules and a rule engine, triggers on a database or RDF store, decision trees, tableau algorithms, or even programmatically using hard-coded business logic
  - A reasoner must be compliant to the semantics of the ontology language it supports
  - Hence, an ontology language must state its semantics in a formal way

- The RDFS reasoner uses ***entailment rules*** that are supposed to capture the intended semantics

# Soundness and Completeness

- Theorem. A graph G1 RDFS-entails a graph G2 if there is a graph G1’ which has been derived from G1 via the rules lg, gl, rdfax, rdf1, rdf2, rdfsax and rdfs1 … rdfs13 such that:
  - G1’ simply entails G2 or
  - G1’ contains an XML clash.
- The inference rules for RDFS-entailment we presented previously are sound but not complete (*ter Horst, 2005*).

# Example

- The following graph:

```
ex:isHappilyMarriedTo rdfs:subPropertyOf _:bnode.
_:bnode rdfs:domain ex:Person.
ex:john ex:isHappilyMarriedTo ex:mary .
```

- The triple `ex:john rdf:type ex:Person .` is a semantic consequence of the graph above, but this cannot be derived from the inference rules

# Decidability and complexity

- RDFS entailment is decidable, even though one has to deal with the infinite number of axiomatic triples:
  - due to the fact that the RDF vocabulary for encoding lists includes property names `rdf:_i` for all i $\geq$ 1, with several RDFS axiomatic triples for each `rdf:_i`

- The problem of deciding whether a graph G1 RDFS-entails another graph G2 is NP-complete. The problem becomes polynomial if G2 contains no blank nodes

# RDFS entailment in state of the art systems

- Existing RDF stores (Jena, Sesame, Virtuoso, Oracle, etc) offer implementations of RDFS entailment together with ways of querying the stored graphs through SPARQL
- Implementations may be based on applying the rules in a backward chaining or a forward chaining fashion

# RDFS entailment cheatsheet

**RDFS entailment patterns.**

| | If S contains: | then S RDFS entails recognizing D: |
|---|---|---|
| ***rdfs1*** | any IRI aaa in D | aaa `rdf:type rdfs:Datatype .` |
| ***rdfs2*** | aaa `rdfs:domain` xxx .<br>yyy aaa zzz . | yyy `rdf:type` xxx . |
| ***rdfs3*** | aaa `rdfs:range` xxx .<br>yyy aaa zzz . | zzz `rdf:type` xxx . |
| ***rdfs4a*** | xxx aaa yyy . | xxx `rdf:type rdfs:Resource .` |
| ***rdfs4b*** | xxx aaa yyy. | yyy `rdf:type rdfs:Resource .` |
| ***rdfs5*** | xxx `rdfs:subPropertyOf` yyy .<br>yyy `rdfs:subPropertyOf` zzz . | xxx `rdfs:subPropertyOf` zzz . |
| ***rdfs6*** | xxx `rdf:type rdf:Property .` | xxx `rdfs:subPropertyOf` xxx . |
| ***rdfs7*** | aaa `rdfs:subPropertyOf` bbb .<br>xxx aaa yyy . | xxx bbb yyy . |
| ***rdfs8*** | xxx `rdf:type rdfs:Class .` | xxx `rdfs:subClassOf rdfs:Resource .` |
| ***rdfs9*** | xxx `rdfs:subClassOf` yyy .<br>zzz `rdf:type` xxx . | zzz `rdf:type` yyy . |
| ***rdfs10*** | xxx `rdf:type rdfs:Class .` | xxx `rdfs:subClassOf` xxx . |
| ***rdfs11*** | xxx `rdfs:subClassOf` yyy .<br>yyy `rdfs:subClassOf` zzz . | xxx `rdfs:subClassOf` zzz . |
| ***rdfs12*** | xxx `rdf:type rdfs:ContainerMembershipProperty .` | xxx `rdfs:subPropertyOf rdfs:member .` |
| ***rdfs13*** | xxx `rdf:type rdfs:Datatype .` | xxx `rdfs:subClassOf rdfs:Literal .` |

# RDFS entailment

- Given the graph G below,

```
⟨d:Poe, o:wrote, d:TheGoldBug .⟩
⟨d:TheGoldBug, rdf:type, o:Novel .⟩
⟨d:Baudelaire, o:translated, d:TheGoldBug .⟩

⟨d:Poe, o:wrote, d:TheRaven .⟩
⟨d:TheRaven,rdf:type,o:Poem .⟩
⟨d:Mallarme´, o:translated, d:TheRaven .⟩

⟨d:Mallarme´,o:wrote,_:b .⟩
⟨_:b, rdf:type, o:Poem .⟩
<o:Poem rdfs:subClassOf ex:Literature .>
<o:Novel rdfs:subClassOf ex:Literature .>
```

- And the following graph S, determine if G entails (using simple and RDFS entailment) S, and explain why.

```
S= <d:Poe wrote _:c .> <_:c rdf:type ex:Literature .>
```

# RDFS entailment

```
1. ⟨d:Poe, o:wrote, d:TheGoldBug .⟩
2. ⟨d:TheGoldBug, rdf:type, o:Novel .⟩
3. ⟨d:Baudelaire, o:translated,
d:TheGoldBug .⟩

4. ⟨d:Poe, o:wrote, d:TheRaven .⟩
5. ⟨d:TheRaven,rdf:type,o:Poem .⟩
6. ⟨d:Mallarme´, o:translated,
d:TheRaven .⟩

7. ⟨d:Mallarme´,o:wrote,_:b .⟩
⟨_:b, rdf:type, o:Poem .⟩

8. <o:Poem rdfs:subClassOf ex:Literature .>

9. <o:Novel rdfs:subClassOf
ex:Literature .>

S= <d:Poe wrote _:c .>
<_:c rdf:type ex:Literature .>
```

```
From RDFS 9 applied to 9 and 2

10. ⟨d:TheGoldBug, rdf:type,
o:Literature .⟩

Apply SE1 to 1 and we obtain
<d:Poe o:wrote _:c>
with -:c -> d;TheGoldBug

Apply SE2 to 10, and we obtain
11.⟨_:c, rdf:type, o:Literature .⟩
```

**So, the graph S is RDFS entailed**


# COMP318

## Ontologies and Semantic Web

# End of RDF - Part 14

**Dr Valentina Tamma**
V.Tamma@liverpool.ac.uk