Logic and Reasoning in the Semantic Web

Fulvio Corno, Laura Farinetti
Politecnico di Torino
Dipartimento di Automatica e Informatica
e-Lite Research Group – http://elite.polito.it
Outline

- Reasoning in Semantic Web Knowledge Bases
- RDF/RDFS Semantics and Entailments
- OWL Semantics
- Pellet
Reasoning

- Reasoning is required when a program must determine some information or some action that has not been explicitly told about.
- It must figure out what it needs to know from what it already knows.
Example

- Facts:
  - Robins are birds
  - All birds have wings

- Questions:
  - Are Robins birds? → Yes
  - Do all birds have wings? → Yes
  - Do robins have wings? → ?????
Semantic Web peculiarities

- Traditional KBS used to be developed top down
- The Semantic Web is developed bottom-up
- Problems:
  - **Scalability**: Everybody having a web page is a potential Knowledge Engineer
  - **Distribution**: Pieces of knowledge can be all over the place, finding them can be hard
  - **Heterogeneity**: People use different ways of describing the same information
  - **Quality**: Knowledge will be incomplete and even inconsistent across different sources of mapping approaches
Reasoning in the Semantic Web

- New methods are needed that
  - Scale to large amounts of data and knowledge
  - Are tolerant to errors, incompleteness and inconsistency between and within sources
  - Work on distributed representations and ideally are distributed themselves
  - ...and are compatible with Semantic web standards
Outline

- Reasoning in Semantic Web Knowledge Bases
- RDF/RDFS Semantics and Entailments
- OWL Semantics
- Pellet
Recall: RDF Schema

Some RDFS inference rules

- \((X \ R \ Y), (R \ \text{subPropertyOf} \ Q) \Rightarrow (X \ Q \ Y)\)
- \((X \ R \ Y), (R \ \text{domain} \ C) \Rightarrow (X \ \text{type} \ C)\)
- \((X \ \text{type} \ C), (C \ \text{subClassOf} \ D) \Rightarrow (X \ \text{type} \ D)\)
- ...

RDF Semantics

Semantics is not about the “meaning” of assertions

- The 'meaning' of an assertion in RDF or RDFS may depend on many factors, including social conventions, comments in natural language or links to other content-bearing documents, … (non machine-processable information)

- Semantics restricts itself to a formal notion of meaning which could be characterized as the part that is common to all other accounts of meaning, and can be captured in mechanical inference rules
Model Semantics

- The RDF Semantics W3C Recommendation uses model theory for specifying the semantics of the RDF language.
- Model theory assumes that the language refers to a 'world', and describes the minimal conditions that a world must satisfy in order to assign an appropriate meaning for every expression in the language.
- A particular world is called an interpretation.
- The idea is to provide an abstract, mathematical account of the properties that any such interpretation must have, making as few assumptions as possible about its actual nature or intrinsic structure, thereby retaining as much generality as possible.
Goals of the inference process

- To provide a technical way to determine when inference processes are valid, i.e., when they preserve truth.
- Starting from a set of assertions that are regarded as true in an RDF model, derive whether a new RDF model contains provably true assertions.
- We never known about the “real” truth of any assertion in the “real” world.
Formalization

- **Interpretations (Normative)**
  - Mapping of RDF assertions into an abstract model, based on set-theory
  - With an “interpretation operator” \( I() \), maps a RDF graph into a highly abstract set of high-cardinality sets
  - Highly theoretical model, useful to prove mathematical properties

- **Entailments (Informative)**
  - Transformation rules to derive new assertions from existing ones
  - May be proven complete and consistent with the formal interpretation
Interpretation: minimum notions

- I : interpretation operator
- E : fragment of RDF syntax
- If E is a graph:
  - I(E) = false if I(E') = false for some triple E' in E, otherwise I(E) =true
- If E is a triple <s, p, o>:
  - I(E) = true if s, p and o are in V, I(p) is in IP and <I(s),I(o)> is in IEXT(I(p))
  - otherwise I(E)= false
- IP : set of properties, IR : set of resources
- IEXT(I(p)) : mapping from IP into the powerset of IR x IR i.e. the set of sets of pairs <x,y> with x and y in IR
## Entailment

- I *satisfies* E if I(E)=true
- A set S of RDF graphs **entails** a graph E if every interpretation which satisfies every member of S also satisfies E
- In human words:
  - assertion = a claim that the world is an interpretation which assigns the value true to the assertion
  - If A entails B, then
    - any interpretation that makes A true also makes B true
    - an assertion of A already contains the same "meaning" as an assertion of B
    - the meaning of B is somehow contained in, or subsumed by, that of A
RDF Entailment rules (legend)

- aaa, bbb: any URI reference
- uuu, vvv: any URI reference or blank node identifier
- xxx, yyy: any URI reference, blank node identifier or literal
- lll: any literal
- _:nnn: blank node identifiers
## Simple Entailment rules

<table>
<thead>
<tr>
<th>Rule name</th>
<th>If E contains</th>
<th>then add</th>
</tr>
</thead>
<tbody>
<tr>
<td>se1</td>
<td>uuu aaa xxx .</td>
<td>uuu aaa _:nnn . where _:nnn identifies a blank node allocated to xxx by rule se1 or se2.</td>
</tr>
<tr>
<td>se2</td>
<td>uuu aaa xxx .</td>
<td>_:nnn aaa xxx . where _:nnn identifies a blank node allocated to uuu by rule se1 or se2.</td>
</tr>
<tr>
<td>lg</td>
<td>uuu aaa lll .</td>
<td>uuu aaa _:nnn . where _:nnn identifies a blank node allocated to the literal lll by this rule.</td>
</tr>
<tr>
<td>gl</td>
<td>uuu aaa _:nnn . where _:nnn identifies a blank node allocated to the literal lll by rule lg.</td>
<td>uuu aaa lll .</td>
</tr>
</tbody>
</table>
Role of Entailment rules (1/3)

- Simple entailment satisfies the “Interpolation Lemma”:
  - S entails a graph E if and only if a subgraph of S is an instance of E
  - It completely characterizes simple RDF entailment in syntactic terms

- RDF entailment satisfies the “RDF entailment lemma”:
  - S rdf-entails E if and only if there is a graph which can be derived from S plus the RDF axiomatic triples by the application of rule Ig and the RDF entailment rules and which simply entails E
  - This means that the entailment rules are “complete”
## RDF Entailment rules

<table>
<thead>
<tr>
<th>Rule name</th>
<th>If E contains</th>
<th>then add</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf1</td>
<td>uuu aaa yyy .</td>
<td>aaa rdf:type rdf:Property .</td>
</tr>
</tbody>
</table>
| rdf2      | uuu aaa lll .  
where lll is a well-typed XML literal . | _:nnn rdf:type  
rdf:XMLLiteral .  
where _:nnn identifies a blank node allocated to lll by rule lg. |
RDF Axiomatic triples

<table>
<thead>
<tr>
<th>rdf:type</th>
<th>rdf:type</th>
<th>rdf:Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:subject</td>
<td>rdf:type</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdf:predicate</td>
<td>rdf:type</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdf:object</td>
<td>rdf:type</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdf:first</td>
<td>rdf:type</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdf:rest</td>
<td>rdf:type</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdf:value</td>
<td>rdf:type</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdf:_1</td>
<td>rdf:type</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdf:_2</td>
<td>rdf:type</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rdf:nil</td>
<td>rdf:type</td>
<td>rdf:List</td>
</tr>
</tbody>
</table>
Role of Entailment rules (2/3)

- RDF entailment satisfies the “RDF entailment lemma”:
  - $S$ rdf-entails $E$ if and only if there is a graph which can be derived from $S$ plus the RDF axiomatic triples by the application of rule lg and the RDF entailment rules and which simply entails $E$
  - This means that the entailment rules are “complete”
## RDFS Entailment rules (1/2)

<table>
<thead>
<tr>
<th>Rule name</th>
<th>If E contains</th>
<th>then add</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs1</td>
<td>uuu aaa lll. where lll is a plain literal (with or without a language tag)</td>
<td>_:nnn rdf:type rdfs:Literal . where _:nnn identifies a blank node allocated to lll by rule rule lg.</td>
</tr>
<tr>
<td>rdfs2</td>
<td>aaa rdfs:domain xxx . uuu aaa yyy .</td>
<td>uuu rdf:type xxx .</td>
</tr>
<tr>
<td>rdfs3</td>
<td>aaa rdfs:range xxx . uuu aaa vvv .</td>
<td>vvv rdf:type xxx .</td>
</tr>
<tr>
<td>rdfs4a</td>
<td>uuu aaa xxx .</td>
<td>uuu rdf:type rdfs:Resource .</td>
</tr>
<tr>
<td>rdfs4b</td>
<td>uuu aaa vvv.</td>
<td>vvv rdf:type rdfs:Resource .</td>
</tr>
<tr>
<td>rdfs5</td>
<td>uuu rdfs:subPropertyOf vvv . vvv rdfs:subPropertyOf xxx .</td>
<td>uuu rdfs:subPropertyOf xxx .</td>
</tr>
</tbody>
</table>
## RDFS Entailment rules (2/2)

<table>
<thead>
<tr>
<th>Rule name</th>
<th>If E contains</th>
<th>then add</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs6</td>
<td>uuu rdf:type</td>
<td>uuu rdfs:subPropertyOf uuu .</td>
</tr>
<tr>
<td></td>
<td>rdf:Property</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rdfs7</td>
<td>aaa rdfs:subPropertyOf bbb uuu aaa yyy .</td>
<td>uuu bbb yyy .</td>
</tr>
<tr>
<td>rdfs8</td>
<td>uuu rdf:type rdfs:Class .</td>
<td>uuu rdfs:subClassOf rdfs:Resource .</td>
</tr>
<tr>
<td>rdfs9</td>
<td>uuu rdfs:subClassOf xxx . vvv rdf:type uuu .</td>
<td>vvv rdf:type xxx .</td>
</tr>
<tr>
<td>rdfs10</td>
<td>uuu rdf:type rdfs:Class .</td>
<td>uuu rdfs:subClassOf uuu .</td>
</tr>
<tr>
<td>rdfs11</td>
<td>uuu rdfs:subClassOf vvv . vvv rdfs:subClassOf xxx .</td>
<td>uuu rdfs:subClassOf xxx .</td>
</tr>
<tr>
<td>rdfs12</td>
<td>uuu rdf:type rdfs:ContainerMembershipProperty .</td>
<td>uuu rdfs:subPropertyOf rdfs:member .</td>
</tr>
<tr>
<td>rdfs13</td>
<td>uuu rdf:type rdfs:Datatype</td>
<td>uuu rdfs:subClassOf rdfs:Literal .</td>
</tr>
</tbody>
</table>
### RDFS Axiomatic triples (1/3)

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:type</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>rdfs:domain</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdfs:range</td>
<td>rdfs:domain</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>rdfs:domain</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
<td>rdfs:domain</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdf:subject</td>
<td>rdfs:domain</td>
<td>rdf:Statement</td>
</tr>
<tr>
<td>rdf:predicate</td>
<td>rdfs:domain</td>
<td>rdf:Statement</td>
</tr>
<tr>
<td>rdf:object</td>
<td>rdfs:domain</td>
<td>rdf:Statement</td>
</tr>
<tr>
<td>rdfs:member</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:first</td>
<td>rdfs:domain</td>
<td>rdf:List</td>
</tr>
<tr>
<td>rdf:rest</td>
<td>rdfs:domain</td>
<td>rdf:List</td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:isDefinedBy</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:comment</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:value</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
</tbody>
</table>
### RDFS Axiomatic triples (2/3)

<table>
<thead>
<tr>
<th>RDF Property</th>
<th>RDFS Range</th>
<th>RDFS Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:type</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:range</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>rdfs:range</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
<td>rdfs:range</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdf:subject</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:predicate</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:object</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:member</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:first</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:rest</td>
<td>rdfs:range</td>
<td>rdf:List</td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:isDefinedBy</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:comment</td>
<td>rdfs:range</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>rdfs:range</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>rdf:value</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
</tbody>
</table>
RDFS Axiomatic triples (3/3)

<table>
<thead>
<tr>
<th>RDF Term</th>
<th>Subclass Of</th>
<th>RDFS:Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:Alt</td>
<td>rdfs:subClassOf</td>
<td>rdfs:Container</td>
</tr>
<tr>
<td>rdf:Bag</td>
<td>rdfs:subClassOf</td>
<td>rdfs:Container</td>
</tr>
<tr>
<td>rdf:Seq</td>
<td>rdfs:subClassOf</td>
<td>rdfs:Container</td>
</tr>
<tr>
<td>rdfs:ContainerMembershipProperty</td>
<td>rdfs:subClassOf</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdfs:isDefinedBy</td>
<td>rdfs:subPropertyOf</td>
<td>rdfs:seeAlso</td>
</tr>
<tr>
<td>rdf:XMLLiteral</td>
<td>rdf:type</td>
<td>rdfs:Datatype</td>
</tr>
<tr>
<td>rdf:XMLLiteral</td>
<td>rdfs:subClassOf</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>rdfs:Datatype</td>
<td>rdfs:subClassOf</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdf:_1</td>
<td>rdf:type</td>
<td>rdfs:ContainerMembershipProperty</td>
</tr>
<tr>
<td>rdf:_1</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:_1</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:_2</td>
<td>rdf:type</td>
<td>rdfs:ContainerMembershipProperty</td>
</tr>
<tr>
<td>rdf:_2</td>
<td>rdfs:domain</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdf:_2</td>
<td>rdfs:range</td>
<td>rdfs:Resource</td>
</tr>
</tbody>
</table>

...
Role of Entailment rules (3/3)

- RDFS entailment satisfies the “RDFS entailment lemma”:
  - $S \text{ rdfs-entails } E$ if and only if there is a graph which can be derived from $S$ plus the RDF and RDFS axiomatic triples by the application of rule $lg$, rule $gl$ and the RDF and RDFS entailment rules and which either simply entails $E$ or contains an XML clash
Effect of Entailment rules (1/3)

- The outputs of these rules will often trigger others.

<table>
<thead>
<tr>
<th>rule:</th>
<th>triggers rule:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4a</th>
<th>4b</th>
<th>5a</th>
<th>5b</th>
<th>6</th>
<th>7a</th>
<th>7b</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4a</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4b</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5a</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Dependencies between RDFS entailment rules

Effect of Entailment rules (2/3)

- Rules propagate all rdf:type assertions in the graph up the subproperty and subclass hierarchies
- rdf1 generates type assertions for all the property names used in the graph
  - uuu aaa yyy → aaa rdf:type rdf:Property
- rdfs3 together with the last RDFS axiomatic triple adds all type assertions for all the class names used
  - → rdf:_2 rdfs:range rdfs:Resource
  - aaa rdfs:range xxx ∧ uuu aaa vvv → vvv
    rdf:type xxx
Effect of Entailment rules (3/3)

- Any subproperty or subclass assertion generates type assertions for its subject and object via rdfs2 and rdfs3 and the domain and range assertions in the RDFS axiomatic triple set
  - $\text{aaa rdfs:domain xxx} \land \text{uuu aaa yyy} \rightarrow \text{uuu rdf:type xxx}$
  - $\text{aaa rdfs:range xxx} \land \text{uuu aaa vvv} \rightarrow \text{vvv rdf:type xxx}$
- For every $\text{uuu}$ in $V$, the rules generate all assertions
  - $\text{uuu rdf:type rdfs:Resource}$
- For every class name $\text{uuu}$, the rules generate
  - $\text{uuu rdfs:subClassOf rdfs:Resource}$
Closure and Reduction

- The **closure** of a graph is the graph defined by all triples that are **inferred** by the deduction system
  - Using the closure of a graph for storing minimizes query processing time

- The **reduction** of a graph is the **minimal** subset of triples needed to compute its closure
  - Using the reduction of a graph for storing minimizes the storage space needed.
Example

rdfs:domain

ex:paints

rdfs:subPropertyOf

ex:creates

rdfs:subClassOf

ex:Human

rdf:type

ex:Artist

rdf:type

ex:Painter

ex:r1

ex:r2

ex:paints
Example: closure

- ex:creates
- ex:paints
- ex:Human
- ex:Artist
- ex:Painter
- rdfs:Property
- rdfs:Class
- rdf:type
- rdfs:subPropertyOf
- rdfs:subClassOf
- rdfs:domain
- ex:r1
- ex:r2
Example: reduction
Using closure for reasoning

- What kinds of statements are actually added to the closure?
- Which inference rule(s) contribute to the closure?
- Look at behavior on realistic data
  - CIA World Fact Book (mostly instances, simple schema)
  - TAP KB (some hierarchies with many instances)
  - SUMO (rich schema structures, very few instances)
  - WordNet (huge hierarchy, no instances)
Results for CIA World Fact Book

(resource types)

- Rule 4b: 13%
- Rule 6: 3%
- Rule 9: 2%
- Rule 3: 13%
- Rule 2: 69%

(Range restriction) (Domain restriction)

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>other</td>
<td>25,676</td>
<td>25,676</td>
</tr>
<tr>
<td>range</td>
<td>111</td>
<td>127</td>
</tr>
<tr>
<td>domain</td>
<td>112</td>
<td>128</td>
</tr>
<tr>
<td>subPropertyOf</td>
<td>0</td>
<td>146</td>
</tr>
<tr>
<td>subClassOf</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>type</td>
<td>376</td>
<td>4,150</td>
</tr>
</tbody>
</table>
Results for TAP KB

(Domain restriction)

Rule 9 55%
Rule 4b 1%
Rule 3 1%

(inheritance)

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>other</td>
<td>71,044</td>
<td>71,044</td>
</tr>
<tr>
<td>range</td>
<td>49</td>
<td>65</td>
</tr>
<tr>
<td>domain</td>
<td>55</td>
<td>71</td>
</tr>
<tr>
<td>subPropertyOf</td>
<td>0</td>
<td>73</td>
</tr>
<tr>
<td>subClassOf</td>
<td>283</td>
<td>1,491</td>
</tr>
<tr>
<td>type</td>
<td>36,988</td>
<td>191,972</td>
</tr>
</tbody>
</table>
Results for SUMO

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>other</td>
<td>7,126</td>
<td>8,192</td>
</tr>
<tr>
<td>range</td>
<td>634</td>
<td>650</td>
</tr>
<tr>
<td>domain</td>
<td>646</td>
<td>662</td>
</tr>
<tr>
<td>subPropertyOf</td>
<td>231</td>
<td>1,080</td>
</tr>
<tr>
<td>subClassOf</td>
<td>3,797</td>
<td>18,439</td>
</tr>
<tr>
<td>type</td>
<td>6,709</td>
<td>18,198</td>
</tr>
</tbody>
</table>
Results for WordNet

- **Subclassing**: Rule 11 (48%)
- **Subproperty**
  - Rule 2 (20%)
  - Rule 3 (2%)
  - Rule 7 (10%)
- **Subclass Reflexivity**
  - Rule 10 (10%)
  - Rule 8 (10%)

<table>
<thead>
<tr>
<th>Category</th>
<th>Original</th>
<th>Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>subPropertyOf</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>subClassOf</td>
<td>78,446</td>
<td>606,418</td>
</tr>
<tr>
<td>type</td>
<td>99,653</td>
<td>277,267</td>
</tr>
<tr>
<td>other</td>
<td>295,528</td>
<td>373,973</td>
</tr>
<tr>
<td>range</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>domain</td>
<td>0</td>
<td>16</td>
</tr>
</tbody>
</table>
Conclusions

- The greatest potential in saving closure space is in type and subclass statements
- Type relations show a significant increase in all models
- Once the transitive closure of the hierarchies are known, type statements can easily be computed online
- We can reduce the size of the closure by leaving out type statements that follow from the model and computing them online.
Discussion on RDF

- RDF is a simple, but useful language to make relations between resources explicit.
- It allows to define a simple form of schema which mostly consists of class and property hierarchies as well as relations with domain and range restriction.
- There is a simple rule based reasoning mechanism for RDF schema that in most cases can be used offline because the increase is only factor 1.5 – this can further be reduced by only completing the schema offline.
- Unlike most traditional KR languages RDF schema allows to freely combine modeling primitives which affects reasoning.
Problems with RDF

- Not always meaningful representation
Need logical axioms!

Teacher ↔ ∃teaches.Courses

Teacher → Person

Person → Thing ∧ ¬Town
We can detect inconsistencies

Teacher \wedge Person \wedge Thing \wedge \neg Town \wedge Town

Teacher \wedge Person \wedge Thing \wedge \neg Town

Teacher \wedge Person

Teacher

http://www.harmelen.nl

teaches

Town

type

http://bb.vu.nl/wbkr
What properties do we need?

- Subclass relations
- Sub-properties
- Set operations

- Intersection (logical conjunction)
  - Painter ∩ Writer = \{x | x ∈ Painter AND x ∈ Writer\}
- Union (logical disjunction)
  - Painter ∪ Writer = \{x | x ∈ Painter OR x ∈ Writer\}
- Complement
  - complementOf(III, Healthy)
- Disjoint
  - disjointWith(Lung, Liver, Kidney)
What properties do we need? /2

- Cardinality restriction
- Existential quantifier
- Universal quantifier
What properties do we need? /3

- Inverse properties
- Symmetric properties
- Transitivity
Outline

- Reasoning in Semantic Web Knowledge Bases
- RDF/RDFS Semantics and Entailments
- OWL Semantics
- Pellet
OWL supports…

- Superset of RDF/RDFS
  - Fact stating facilities from RDF
  - Class property structuring from RDF Schema

- New logical operators
  - Boolean operators
  - Property hierarchies
  - Properties can be defined transitive, functional, inverse...
  - Individuals can be defined instances
  - Equivalence and disjointness statements on classes
  - Equivalence statements on properties
  - Equality and inequality can be asserted between individuals
OWL keywords

**OWL Light**
- (sub)classes, individuals
- (sub)properties, domain, range
- conjunction
- (in)equality
- cardinality 0/1
- datatypes
- inverse, transitive, symmetric
- hasValue
- someValuesFrom
- allValuesFrom

**RDF Schema**

**OWL DL**
- Negation
- Disjunction
- Full Cardinality
- Enumerated types

**OWL Full**
- Allow meta-classes etc
RDFS vs OWL

- In RDFS you can:
  - declare classes like Artist, Museum or Paintings
  - state that Painter is a subclass of Artist
  - state that rembrandt is an instances of class Painter
  - state that hasPainted is a property, with domain Painter and range Painting.
  - state that rembrand is an instance of Dutchman with deathdate value 1669.
RDFS vs OWL

In OWL you can also:

- state that Country and Person are disjoint classes
- state that the nl and england are distinct individuals of the class Country
- declare hasPainted as inverse property of paintedBy
- state that the class stateless is defined as those members of the class Person that have no values for the property nationality
- state that the class Canadian is defined as those members of the class Person that have canada as a value of the property nationality
- state that age is a functional property.
And now?

- OWL comes with formal reasoning systems
  - OWL-Lite with 1st-order logic
  - OWL-DL with Description Logic
- We can reason by combining
  - Available ontologies
  - Available facts
The ingredients

- Ontology (e.g., in OWL-DL)
- Facts (e.g., objects and instances)
- A reasoning algorithm (for DL)
- A reasoning engine
- A way to express queries
- A way to present results
Description Logics: Syntax

- **Concepts** corresponds to classes
- **Roles** correspond to class properties
- **Constructors** mix of set notation and FO quantification

Booleans: $C \cap D$, $C \cup D$, $\neg C$
Qualified quantification: $\forall R.C$, $\exists R.C$

- Variable free notation for concepts (classes)
  - $\text{artist}(x) = \text{person}(x) \land \exists y \text{ created}(x, y) \land \text{Artwork}(y)$
    is written as $\text{Artist}\sqsubseteq\text{Person} \cap \exists \text{created}.\text{Artwork}$
Description Logic: Semantics

- Interpretations are pairs \((\Delta, \cdot^\mathcal{I})\), with a universe \(\Delta\) and a mapping \(\mathcal{I}\) from
  - concept names to subsets of \(\Delta\)
  - role names to binary relations

- Booleans: 
  \[C \sqcap D, \quad (C \sqcap D)^\mathcal{I} = C^\mathcal{I} \sqcap D^\mathcal{I}\]
  \[C \sqcup D, \quad (C \sqcup D)^\mathcal{I} = C^\mathcal{I} \sqcup D^\mathcal{I}\]
  \[\neg C, \quad (\neg C)^\mathcal{I} = \Delta \setminus C^\mathcal{I}\]

Qualified quantification:

\[\forall R. C \quad \forall R. C^\mathcal{I} = \{x \in \Delta \mid \forall y \in \Delta : R^\mathcal{I}(x, y) \rightarrow y \in C^\mathcal{I}\}\]

\[\exists R. C \quad \exists R. C^\mathcal{I} = \{x \in \Delta \mid \exists y \in \Delta : R^\mathcal{I}(x, y) \& y \in C^\mathcal{I}\}\]
Concept Reasoning

Based on these semantics there are two basic reasoning services:

- **Concept satisfiability**, \( \models C \neq \bot \).
  - Check whether for some interpretation \( \mathcal{I} \) we have \( C^\mathcal{I} \neq \emptyset \).
  - \( \models \forall \text{creates}. \text{Sculpture} \sqcap \exists \text{creates}. (\text{Artwork} \sqcap \neg \text{Sculpture}) = \bot \).

- **Concept subsumption**, \( \models C_1 \subseteq C_2 \).
  - Check whether for all interpretations \( \mathcal{I} \) we have \( C_1^\mathcal{I} \subseteq C_2^\mathcal{I} \).
  - \( \forall \text{creates}. \text{Painting} \sqcap \exists \text{creates}. \top \sqsubseteq \exists \text{creates}. \text{Painting} \).
## Modular Definition of Description Logics

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept name</td>
<td>$C$</td>
<td>$C^I$</td>
</tr>
<tr>
<td>conjunction</td>
<td>$C_1 \sqcap C_2$</td>
<td>$C_1^I \sqcap C_2^I$</td>
</tr>
<tr>
<td>univ. quant.</td>
<td>$\forall R.C$</td>
<td>$\forall d_1 \in \Delta^I. (R^I d_1 d_2 \rightarrow d_2 \in C^I)$</td>
</tr>
<tr>
<td>top</td>
<td>$\top$</td>
<td>$\Delta^I$</td>
</tr>
<tr>
<td>negation ($\neg$)</td>
<td>$\neg C$</td>
<td>$\Delta^I \setminus C^I$</td>
</tr>
<tr>
<td>disjunction ($\lor$)</td>
<td>$C_1 \sqcup C_2$</td>
<td>$C_1^I \sqcup C_2^I$</td>
</tr>
<tr>
<td>exist. quant. ($\exists$)</td>
<td>$\exists R.C$</td>
<td>$\forall d_1 \in \Delta^I. (R^I d_1 d_2 \land d_2 \in C^I)$</td>
</tr>
<tr>
<td>number restr. ($\geq n$)</td>
<td>$\geq n R$</td>
<td>${d_1 \mid {d_2 \mid R^I d_1 d_2} \geq n}$</td>
</tr>
<tr>
<td></td>
<td>$\leq n R$</td>
<td>${d_1 \mid {d_2 \mid R^I d_1 d_2} \leq n}$</td>
</tr>
<tr>
<td>one-of ($\mathcal{O}$)</td>
<td>${a_1, \ldots, a_n}$</td>
<td>${d \mid d = a_i^I \text{ for some } a_i}$</td>
</tr>
<tr>
<td>filler ($\mathcal{B}$)</td>
<td>$\exists R.{a}$</td>
<td>${d \mid d = R^I d a^I}$</td>
</tr>
<tr>
<td>role name</td>
<td>$R$</td>
<td>$R^I$</td>
</tr>
<tr>
<td>role conj. ($\mathcal{R}$)</td>
<td>$R_1 \sqcap R_2$</td>
<td>$R_1^I \sqcap R_2^I$</td>
</tr>
<tr>
<td>inverse roles ($\mathcal{I}$)</td>
<td>$R^{-1}$</td>
<td>${(d_1, d_2) \mid R^I (d_2, d_1)}$</td>
</tr>
</tbody>
</table>
Terminological Reasoning

\[ T = \{ \text{Painting} \sqsubset \text{Artwork} \sqcap \neg \text{Sculpture}, \]
\[ \text{Painter} \sqsubset \exists \text{creates}. \text{Paintings}, \]
\[ \text{Sculpturer} \sqsubset \exists \text{creates}. \text{Artwork} \sqcap \forall \text{creates}. \text{Sculpture} \} \]

- **Concept satisfiability**, \( \Sigma \models C \neq \bot. \)
  - Check whether there is an interpretation \( \mathcal{I} \) such that \( \mathcal{I} \models \Sigma \) and \( C_1^\mathcal{I} \subseteq C_2^\mathcal{I} \).
  - **Concept unsatisfiability**: \( \Sigma \models \text{Painter} \sqcap \text{Sculpturer} = \bot. \)

- **Subsumption**, \( \Sigma \models C_1 \sqsubseteq C_2. \)
  - Check whether for all interpretations \( \mathcal{I} \) such that \( \mathcal{I} \models \Sigma \) we have \( C_1^\mathcal{I} \subseteq C_2^\mathcal{I}. \)
  - **Subsumption**: \( \Sigma \models \text{Painter} \sqsubseteq \neg \text{Sculpurer} \)
Assertional reasoning

\[ \mathcal{A} = \{ \text{rembrandt:Artist, nightwatch:Painting}, \] 
\[ (\text{rembrandt,nightwatch}):\text{created} \} \]

and \( \Sigma = \langle \mathcal{T}, \mathcal{A} \rangle \)

- **Consistency**, \( \Sigma \notmodels \bot \).
  - Check whether there exists \( \mathcal{I} \) such that \( \mathcal{I} \models \Sigma \).
  - \( \Sigma \models \mathcal{A} \neq \bot \) but \( \Sigma \models \mathcal{A} \cup \{ \text{rembrandt:Sculpturor} \} = \bot \)

- **Instance Checking**, \( \Sigma \models a : C \).
  - Check whether \( a^\mathcal{I} \in C^\mathcal{I} \) for all interpretations \( \mathcal{I} \models \Sigma \).
  - \( \text{rembrandt} \in \Sigma \text{ Painter} \).

- **Defined reasoning tasks:**
  - **Retrieval**: \( \text{retrieve(Artists)} = \{ \text{rembrandt} \} \).
  - **Realization**: find most specific concepts in \( \mathcal{T} \) for instances in \( \mathcal{A} \)
    \[ \text{realize(rembrandt)} = \text{Painter} \]
OWL-DL Semantics

- The semantics of OWL-DL constructs is derived by the corresponding Description Logic operators.
- The formal definition is extremely concise
  - http://www.w3.org/TR/owl-semantics/semantics-all.html
  - ...but not so straightforward to understand!
Is this applicable?

- Logic = Perfect reasoning under perfect conditions
- therefore…
  - unlimited time
  - homogeneous knowledge
  - correct and consistent knowledge
Is Logic good?

- **Pro**
  - Strong theoretical basis
  - Well known properties
  - Well known implementation techniques

- **Con**
  - Strict (no “good enough” answers)
  - Abrupt (no intermediate answers)
  - Inefficient (no time/quality trade-off)
Practical limitations

- Terminologies will be sloppy
  - Made by non-experts
  - Made by machines
  - Inference rules break!
- No standard vocabulary
  - Communication problems
  - Need approximate equivalence
- Computational explosion
Example

Shared ontology:

Private ontology:
Possible reconciliations

- **Upper bounds (lub):**
  - Pet → (Domestic-Animal)
  - Farm-Animal → (Domestic-Animal & Production-Animal)
  - Zoo-Animal → (Foreign-Animal)

- **Lower bounds (glb):**
  - Pet → (Cat v Dog)
  - Farm Animal → (Cow v Pig)
  - Zoo Animal → (Camel v Elephant v Tiger v Lion)
Outline

- Reasoning in Semantic Web Knowledge Bases
- RDF/RDFS Semantics and Entailments
- OWL Semantics
- Pellet
Pellet: an OWL-DL Reasoner

- Pellet is a complete and feature-rich reasoner for OWL-DL
- Available open-source at http://clarkparsia.com/pellet
- Available standalone, or integrated with the major ontology development environments (including Protégé)
What is an OWL-DL reasoner

- The official normative definition:
  - An OWL consistency checker takes a document as input, and returns one word being Consistent, Inconsistent, or Unknown. [J. J. Carroll, J. D. Roo, OWL Web Ontology Language Test Cases, W3C Recommendation http://www.w3.org/TR/owl-test/ (2004).]
  - Rather restrictive... and not very useful for ontology development, debug and querying
Practical Description Logics

- Most theoretical works on Description Logics are concerned with the “upper” part of the ontology (classes, relationships)
- Object Instances are equally important, if not more, in the Semantic Web
- Reasoning over instances is “easier”, but their number may be far larger that the number of classes
# DL Jargon

<table>
<thead>
<tr>
<th>Abbr.</th>
<th>Stands for</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABox</td>
<td>Assertional Box</td>
<td>Component that contains assertions about individuals, i.e. OWL facts such as type, property-value, equality or inequality assertions.</td>
</tr>
<tr>
<td>TBox</td>
<td>Terminological Box</td>
<td>Component that contains axioms about classes, i.e. OWL axioms such as subclass, equivalent class or disjointness axioms.</td>
</tr>
<tr>
<td>KB</td>
<td>Knowledge Base</td>
<td>A combination of an ABox and a TBox, i.e. a complete OWL ontology.</td>
</tr>
</tbody>
</table>
Classical Types of Logic Inference

- **Consistency checking**, which ensures that an ontology does not contain any contradictory facts.
- The OWL Abstract Syntax & Semantics document [S&AS] provides a formal definition of ontology consistency that Pellet uses.
- In DL terminology, this is the operation to check the consistency of an ABox with respect to a Tbox.
- Equivalent to OWL Consistency Checking
Classical Types of Logic Inference

- **Concept satisfiability**, which checks if it is possible for a class to have any instances. If class is unsatisfiable, then defining an instance of the class will cause the whole ontology to be inconsistent.
Classical Types of Logic Inference

- **Classification**, which computes the subclass relations between every named class to create the complete class hierarchy. The class hierarchy can be used to answer queries such as getting all or only the direct subclasses of a class.
Classical Types of Logic Inference

- **Realization**, which finds the most specific classes that an individual belongs to; or in other words, computes the direct types for each of the individuals. Realization can only be performed after classification since direct types are defined with
Pellet architecture
Launching Pellet
Using Pellet in Protégé
Using Pellet in Protégé

![Protégé 3.4 interface with Pellet reasoner configuration]
Activating Inference steps
Consistency checking

![Reasoner log](image)
Consistency checking – usually
Consistency checking – usually

Computing inconsistent concepts: Querying reasoner for inconsistent concepts...

Reasoner log
- Synchronize reasoner
  - Time to clear knowledgebase = 0.0030 seconds
  - Time for DIG conversion = 0.011 seconds
  - Time to update reasoner = 0.056 seconds
  - Time to synchronize = 0.079 seconds
- Check concept consistency
  - Time to build query = 0.0010 seconds

Reasoner Error: General Ask Error [ID: ]
References

- RDF Semantics - W3C Recommendation 10 February 2004
  - [http://www.w3.org/TR/rdf-mt/](http://www.w3.org/TR/rdf-mt/)

- Course material for “Practical Reasoning for the Semantic Web” course at the 17th European Summer School in Logic, Language and Information (ESSLLI)
  - [http://www.few.vu.nl/~schlobac/](http://www.few.vu.nl/~schlobac/)

References

  - http://www.w3.org/TR/owl-semantics/
References

- http://media.cwi.nl/survey/
- http://dose.sourceforge.net/
- http://www.mkbergman.com
License

- This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License.

- To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.