The Role of Logics and Logic Programming in Semantic Web Standards (OWL2, RIF, SPARQL1.1)

Axel Polleres
Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway
Outline

- Quick intro of RDF/OWL/Linked Open Data
- OWL2
  - Overview: from OWL 1 to OWL 2
  - Reasoning services in OWL 2
  - OWL2 Treactable Fragments (OWL2RL, OWL2EL, OWL2QL)
- OWL2 and RIF
- OWL2 and SPARQL1.1

- Time allowed: Implementing SPARQL, OWL2RL, RIF on top of DLV – The GiaBATA system
Example: Finding experts/reviewers?


- Who are the right reviewers? Who has the right expertise?
- Which reviewers are in conflict?
- Observation: Most of the necessary data already on the Web, as RDF!

More and more of it follows the Linked Data principles, i.e.:

1. Use URIs as names for things
2. Use HTTP dereferenceable URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information.
4. Include links to other URIs so that they can discover more things.
RDF on the Web

- (i) directly by the publishers
- (ii) by e.g. transformations, D2R, RDFa exporters, etc.

FOAF/RDF linked from a home page: personal data (foaf:name, foaf:phone, etc.), relationships foaf:knows, rdfs:seeAlso)
RDF on the Web

- (i) directly by the publishers
- (ii) by e.g. transformations, D2R, RDFa exporters, etc.

E.g. L3S’ RDF export of the DBLP citation index, using FUB’s D2R (http://dblp.l3s.de/d2r/)

Gives unique URIs to authors, documents, etc. on DBLP! E.g.,
http://dblp.l3s.de/d2r/resource/authors/Tim_Berners-Lee,
http://dblp.l3s.de/d2r/resource/publications/journals/tplp/Berners-LeeCKSH08

Provides RDF version of all DBLP data + query interface!
Excellent tutorial here: http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/
Data in RDF: Triples

DBLP:

<http://dblp.l3s.de/.../journals/tplp/Berners-LeeCKSH08> rdf:type swrc:Article.
<http://dblp.l3s.de/.../journals/tplp/Berners-LeeCKSH08> dc:creator

   <http://dblp.l3s.de/d2r/.../Tim_Berners-Lee> .

...

<http://dblp.l3s.de/d2r/.../Tim_Berners-Lee> foaf:homepage

   <http://www.w3.org/People/Berners-Lee/> .

...

<http://dblp.l3s.de/d2r/.../Dan_Brickley> foaf:name “Dan Brickley”^^xsd:string.

Tim Berners-Lee’s FOAF file:

<http://www.w3.org/People/Berners-Lee/card#i> foaf:knows

   <http://dblp.l3s.de/d2r/.../Dan_Brickley> .

<http://www.w3.org/People/Berners-Lee/card#i> rdf:type foaf:Person .
<http://www.w3.org/People/Berners-Lee/card#i> foaf:homepage

   <http://www.w3.org/People/Berners-Lee/> .
How can I query such data? SPARQL

- **SPARQL** – W3C approved standardized query language for RDF:
  - look-and-feel of “SQL for the Web”
  - allows to ask queries like
    - “All documents by Tim Berners-Lee”
    ...

**Example:**

```sql
SELECT ?D
FROM <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>
WHERE {?D dc:creator <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>}
```
SPARQL more complex patterns: e.g. CQs

- “Names of all persons who co-authored with authors of http://dblp.l3s.de/d2r/.../Berners-LeeCKSH08”

```sparql
SELECT ?Name WHERE {
  <http://dblp.l3s.de/d2r/resource/publication/journals/tplp/Berners-LeeCKSH08>
  dc:creator ?Author.
  ?CoAuthor foaf:name ?Name
}
```
ESWC2010

SPARQL more complex patterns: e.g. UCQs

- “Names of all persons who co-authored with authors of http://dblp.l3s.de/d2r/…/Berners-LeeCKSH08 or known by co-authors”

```
SELECT ?Name WHERE {
  {  ?CoAuthor foaf:name ?Name . }
  UNION
  { ?CoAuthor foaf:knows ?Person.
    ?Person foaf:name ?Name }}
```

- Doesn’t work… no foaf:knows relations in DBLP 😞

- Needs Linked Data! E.g. TimBL’s FOAF file!
Back to the Data:

- **DBLP:**
  
  <http://dblp.l3s.de/.../journals/tplp/Berners-LeeCKSH08> rdf:type swrc:Article.
  <http://dblp.l3s.de/.../journals/tplp/Berners-LeeCKSH08> dc:creator
  
  <http://dblp.l3s.de/d2r/.../Tim_Berners-Lee> .
  
  ...

  <http://dblp.l3s.de/d2r/.../Tim_Berners-Lee> foaf:homepage
  
  <http://www.w3.org/People/Berners-Lee/> .

- **Tim Berners-Lee’s FOAF file:**

  <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows
  <http://dblp.l3s.de/d2r/.../Dan_Brickley> .

  <http://www.w3.org/People/Berners-Lee/card#i> foaf:homepage
  <http://www.w3.org/People/Berners-Lee/> .

- **Even if I have the FOAF data, I cannot answer the query:**
  - Different identifiers used for Tim Berners-Lee
  - Who tells me that Dan Brickley is a foaf:Person?

- **Linked Data needs **Reasoning**!
Reasoning on Semantic Web Data

ESWC2010

Vocabularies (i.e. collections of classes and properties that belong together, e.g. foaf):

- Properties: foaf:name, foaf:homepage, foaf:knows
- Classes: foaf:Person, foaf:Document

Typically should have formal descriptions of their structure:

- RDF Schema, and OWL
- These formal descriptions often "called" ontologies.

Ontologies add "semantics" to the data.

Ontologies are themselves written in RDF, using special vocabularies (rdf:, rdfs:, owl:) with special semantics

→ Ontologies are themselves part of the Linked Data Web!
foaf:knows rdfs:domain foaf:Person
\[ \exists \text{knows}. \top \subseteq \text{Person} \]

foaf:knows rdfs:range foaf:Person
\[ \exists \text{knows}^{-}. \top \subseteq \text{Person} \]

foaf:Person rdfs:subclassOf foaf:Agent
\[ \text{Person} \subseteq \text{Agent} \]

foaf:homepage rdf:type owl:inverseFunctionalProperty .
\[ \top \sqsubseteq 1\text{homepage}^{-} \]

...
Semantics of RDFS can be partially expressed as (Datalog like) rules:

- **rdfs1**: `triple(S, rdf:type, C) :- triple(S, P, O), triple(P, rdfs:domain, C)`
- **rdfs2**: `triple(O, rdf:type, C) :- triple(S, P, O), triple(P, rdfs:range, C)`

- **rdfs3**: `triple(S, rdf:type, C2) :- triple(S, rdf:type, C1), triple(C1, rdfs:subclassOf, C2)`

cf. informative Entailment rules in [RDF-Semantics, W3C, 2004], [Muñoz et al. 2007]
Semantics of RDFS can be partially expressed as (Datalog like) rules:

\[
\begin{align*}
\end{align*}
\]

cf. informative Entailment rules in [RDF-Semantics, W3C, 2004], [Muñoz et al. 2007]
OWL Reasoning e.g. inverseFunctionalProperty can also (partially) be expressed by Rules:

owl1: { ?S1 owl:SameAs ?S2 } :-


cf. pD* fragment of OWL, [ter Horst, 2005], SAOR [Hogan et al. 2009] or, more recent: OWL2 RL
RDFS+OWL inference by rules: Example:

- By rules of the previous slides we can infer additional information needed, e.g.

  **TimBL’s FOAF:** 
  `<.../Berners-Lee/card#i> foaf:knows <.../Dan_Brickley> .
  **FOAF Ontology:** 
  `foaf:knows rdfs:range foaf:Person`

  by rdfs2  →
  `<.../Dan_Brickley> rdf:type foaf:Person.``

  **TimBL’s FOAF:** 
  `<.../Berners-Lee/card#i> foaf:homepage
  <http://www.w3.org/People/Berners-Lee/> .
  **FOAF Ontology:** 
  `foaf:homepage rdfs:type owl:InverseFunctionalProperty.`

  by owl1  →
  `<.../Berners-Lee/card#i> owl:sameAs <.../Tim_Berners-Lee>.

- Who tells me that Dan Brickley is a foaf:Person? → solved!
- Different identifiers used for Tim Berners-Lee → solved!
Note: Not all of OWL Reasoning can be expressed in Datalog straightforwardly, e.g.:

foaf:Person owl:disjointWith foaf:Organisation

Can be written/and reasoned about with FOL/DL reasoners:

**FOL Syntax:** \( \forall X. Person(X) \sqsupset \neg Organisation(X) \)

**DL Syntax:** \( Person \sqcap Organisation \subseteq \bot \)

Problem: **Inconsistencies!** **Complete** FOL/DL reasoning is often not suitable per se for Web data... [Hogan et al.2009,]

But can be “approximated” by Rules (without explosion):

```
```
The more “common” view on OWL...

Expressing property characteristics:

<table>
<thead>
<tr>
<th>OWL property axioms as RDF triples</th>
<th>DL syntax</th>
<th>FOL short representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$ rdfs:domain $C$ .</td>
<td>$T \subseteq \forall P^- . C$</td>
<td>$\forall x, y. P(x, y) \supseteq C(x)$</td>
</tr>
<tr>
<td>$P$ rdfs:range $C$ .</td>
<td>$T \subseteq \forall P . C$</td>
<td>$\forall x, y. P(x, y) \supseteq C(y)$</td>
</tr>
<tr>
<td>$P$ owl:inverseOf $P_0$ .</td>
<td>$P \equiv P_0^-$</td>
<td>$\forall x, y. P(x, y) \equiv P_0(y, x)$</td>
</tr>
<tr>
<td>$P$ rdf:type owl:SymmetricProperty.</td>
<td>$P \equiv P^-$</td>
<td>$\forall x, y. P(x, y) \equiv P(y, x)$</td>
</tr>
<tr>
<td><strong>$P$ rdf:type owl:FunctionalProperty.</strong></td>
<td>$T \subseteq \leq 1 P$</td>
<td>$\forall x, y, z. P(x, y) \land P(x, z) \supseteq y = z$</td>
</tr>
<tr>
<td>$P$ rdf:type owl:InverseFunctionalProperty.</td>
<td>$T \subseteq \leq 1 P^-$</td>
<td>$\forall x, y, z. P(x, y) \land P(z, y) \supseteq x = z$</td>
</tr>
<tr>
<td>$P$ rdf:type owl:TransitiveProperty.</td>
<td>$P^+ \subseteq P$</td>
<td>$\forall x, y, z. P(x, y) \land P(y, z) \supseteq P(x, z)$</td>
</tr>
</tbody>
</table>

Expressing complex class descriptions:

<table>
<thead>
<tr>
<th>OWL complex class descriptions*</th>
<th>DL syntax</th>
<th>FOL short representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>owl:Thing</td>
<td>$T$</td>
<td>$x = x$</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>$\bot$</td>
<td>$\neg x = x$</td>
</tr>
<tr>
<td>owl:intersectionOf ($C_1 \ldots C_n$)</td>
<td>$C_1 \sqcap \ldots \sqcap C_n$</td>
<td>$C_1(x) \land \ldots \land C_n(x)$</td>
</tr>
<tr>
<td>owl:unionOf ($C_1 \ldots C_n$)</td>
<td>$C_1 \sqcup \ldots \sqcup C_n$</td>
<td>$C_1(x) \lor \ldots \lor C_n(x)$</td>
</tr>
<tr>
<td>owl:complementOf ($C$)</td>
<td>$\neg C$</td>
<td>$\neg C(x)$</td>
</tr>
<tr>
<td>owl:oneOf ($o_1 \ldots o_n$)</td>
<td>${o_1, \ldots, o_n}$</td>
<td>$x = o_1 \lor \ldots \lor x = o_n$</td>
</tr>
<tr>
<td>owl:restriction ($P$ owl:someValuesFrom ($C$))</td>
<td>$\exists P. C$</td>
<td>$\exists y. P(x, y) \land C(y)$</td>
</tr>
<tr>
<td>owl:restriction ($P$ owl:allValuesFrom ($C$))</td>
<td>$\forall P. C$</td>
<td>$\forall y. P(x, y) \supseteq C(y)$</td>
</tr>
<tr>
<td>owl:restriction ($P$ owl:value ($o$))</td>
<td>$\exists P. {o}$</td>
<td>$P(x, o)$</td>
</tr>
<tr>
<td>owl:restriction ($P$ owl:minCardinality ($n$))</td>
<td>$\geq np$</td>
<td>$\exists y_1 \ldots y_n. \bigwedge_{k=1}^{n} P(x, y_k) \land \bigwedge_{i&lt;j} y_j \neq y_i$</td>
</tr>
<tr>
<td>owl:restriction ($P$ owl:maxCardinality ($n$))</td>
<td>$\leq np$</td>
<td>$\forall y_1 \ldots y_{n+1}. \bigwedge_{k=1}^{n+1} P(x, y_k) \supseteq \bigvee_{i&lt;j} y_i = y_j$</td>
</tr>
</tbody>
</table>

*For reasons of legibility, we use a variant of the OWL abstract syntax [Patel-Schneider et al., 2004] in this table.
Why OWL1 is Not Enough

- Too expensive to reason with
  - High complexity: NEXPTIME-complete
  - The most lightweight sublanguage OWL-Lite is **NOT** lightweight
  - Some ontologies only use some limited expressive power; e.g. The SNOMED (Systematised Nomenclature of Medicine) ontology

- Not expressive enough; e.g.
  - No user defined datatypes [Pan 2004; Pan and Horrocks 2005; Motik and Horrocks 2008]
  - No metamodelling support [Pan 2004; Pan, Horrocks, Schreiber, 2005; Motik 2007]
  - Limited property support [Horrocks et al., 2006]
OWL2: A new version of OWL

Main goals:
1. To define “profiles” of OWL that are:
   - smaller, easier to implement and deploy
   - cover important application areas and are easily understandable to non-expert users

2. To add a few extensions to current OWL that are useful, and is known to be implementable
   - many things happened in research since 2004
Common ontologies on the Web don’t use it a lot as of yet…

... but adds interesting functionality, potentially useful for Web ontologies, e.g.

- PropertyChains
  - E.g. could be useful to tie sioc:name and foaf:nick via foaf:holdsAccount:

```
foaf:nick owl:propertyChainAxiom (foaf:holdsAccount sioc:name)
```

![Diagram of foaf:Agent and foaf:OnlineAccount relationships]
Common ontologies on the Web don’t use it a lot as of yet…

… but adds interesting functionality, potentially useful for Web ontologies, e.g.

- **hasKey:**
  - *Multi-attribute Keys now possible in OWL, e.g.*
    - foaf:OnlineAccount/sioc:User members are uniquely identified by a combination of foaf:accountName and foaf:accountServiceHomepage:

      foaf:OnlineAccount owl:hasKey

      (foaf:accountName foaf:accountServiceHomepage) .
New Expressiveness in OWL 2

- **New expressive power**
  - user defined datatypes, e.g.:
    
    "Ages are integers between 0 and 150"
    
    ```
    personAge owl:equivalentClass _:x
    _:x rdf:type rdfs:Datatype
    _:x owl:onDatatype xsd:integer
    _:x owl:withRestrictions ((_:y1 _:y2)
    _:y1 xsd:minInclusive "0"^^xsd:integer
    _:y2 xsd:maxInclusive "150"^^xsd:integer
    ```

  - punning (metamodelling), e.g.:
    
    John rdf:type Father
    Father rdf:type SocialRole
New Expressiveness in OWL 2

- New expressive power on properties
  - qualified cardinality restrictions, e.g.:
    
```
_:x rdf:type owl:Restriction
_:x owl:onProperty foaf:knows
_:x owl:minQualifiedCardinality "10"^^xsd:nonNegativeInteger
_:x owl:onClass Irish
```
  
- property chain inclusion axioms, e.g.:
  
```
foaf:nick owl:propertyChainAxiom (foaf:holdsAccount sioc:name)
```
  
- local reflexivity restrictions, e.g.:
  
```
_:x rdf:type owl:Restriction
_:x owl:onProperty like
_:x owl:hasSelf "true"^^xsd:boolean  [for narcissists]
```
  
- reflexive, irreflexive, symmetric, and antisymmetric properties, e.g.:
  
```
foaf:know rdf:type owl:ReflexiveProperty
rel:childOf rdf:type owl:IrreflexiveProperty
```
  
- disjoint properties, e.g.:
  
```
rel:childOf owl:propertyDisjointWith rel:parentOf
```
  
- keys, e.g.:
    
```
foaf:OnlineAccount owl:hasKey
  (foaf:accountName foaf:accountServiceHomepage)
```
**Syntactic sugar** *(make things easier to say)*

- Disjoint unions, e.g.:
  
  ```plaintext
  child owl:disjointUnionOf (boy girl)
  ```

- Disjoint classes, e.g.:
  
  ```plaintext
  _:x rdf:type owl:AllDisjointClasses
  _:x owl:members (boy girl)
  ```

- Negative assertions, e.g.:
  
  ```plaintext
  _:x rdf:type owl:NegativePropertyAssertion
  _:x owl:sourceIndividual John
  _:x owl:assertionProperty foaf:know
  _:x owl:targetIndividual Mary
  ```
OWL 2 DL

- $\mathcal{R}$ often used for $\mathcal{ALC}$ extended with property chain inclusion axioms
  - following the notion introduced in $\mathcal{RTQ}$ [Horrocks and Sattler, 2003]
  - including transitive property axioms
- **Additional letters** indicate other extensions, e.g.:
  - $S$ for property characteristics (e.g., reflexive and symmetric)
  - $\mathcal{O}$ for **nominals**/singleton classes
  - $I$ for inverse roles
  - $Q$ for qualified number restrictions
- property characteristics ($S$) + $\mathcal{R}$ + nominals ($\mathcal{O}$) + inverse ($I$) + qualified number restrictions ($Q$) = $\mathcal{SROIQ}$
- $\mathcal{SROIQ}$ [Horrocks et al., 2006] is the basis for OWL 2 DL
- **Available Reasoners**: Hermit (Oxford), Pellet (Clark&Parsia)
OWL 2 Profiles and Reasoning Services

- **Rationale:**
  - Tractable
  - Tailored to specific reasoning services

- **Popular reasoning services**
  - ABox reasoning: OWL 2 RL
  - TBox reasoning: OWL 2 EL
  - Query answering: OWL 2 QL

The family tree

OWL 2 Full

OWL 2 DL

OWL 2 QL

OWL 1 DL

SHOIN

OWLS RL

SROIQ

Undecidable

2NExpTime-
Complete

NExpTime-
Complete

PTime-
Complete

In AC^0

DL-Lite

EL++
Maximal fragment of OWL expressible in Horn Rules

- Rules for subclassing, subproperties, propChains, (inverse) functionalProperties, hasValue...
- No support for arbitrary card restrictions, existentials in rule heads, etc.
- See before... more later...

See also discussion of the rule set in [Hogan&Decker, 2009]
A (near maximal) fragment of OWL 2 such that
- Satisfiability checking is in PTime (PTime-Complete)
- Data complexity of query answering also PTime-Complete

Based on EL family of description logics [Baader et al. 2005]

Can exploit saturation based reasoning techniques
- Computes complete classification in “one pass”
- Computationally optimal (PTime for EL)
- Can be extended to Horn fragment of OWL DL [Kazakov 2009]
Saturation-based Technique (basics)

- Normalise ontology axioms to standard form:
  \[
  A \subseteq B \quad A \cap B \subseteq C \quad A \subseteq \exists R.B \quad \exists R.B \subseteq C
  \]

- Saturate using inference rules:
  \[
  \begin{align*}
  A \subseteq B & \quad B \subseteq C \quad A \subseteq C \\
  A \subseteq B & \quad A \subseteq C \quad B \cap C \subseteq D \quad A \subseteq D
  \end{align*}
  \]

- Extension to Horn fragment requires (many) more rules
Saturation-based Technique (basics)

Example:

\[
\begin{align*}
A &\subseteq B \\
B &\subseteq C \\
\hline
A &\subseteq C
\end{align*}
\]

\[
\begin{align*}
A &\subseteq B \\
A &\subseteq C \\
B\cap C &\subseteq D \\
\hline
A &\subseteq D
\end{align*}
\]

\[
\begin{align*}
A &\subseteq \exists R.B \\
B &\subseteq C \\
\exists R.C &\subseteq D \\
\hline
A &\subseteq D
\end{align*}
\]

foaf:Person rdfs:subClassOf foaf:Agent.
foaf:Agent rdfs:subClassOf owl:Thing.

foaf:Person rdfs:subClassOf
    [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person. ].

:hasFather domain :Child.

Person \subseteq \top

Person \subseteq Child
OWL 2 QL

- A (near maximal) fragment of OWL 2 such that
  - Data complexity of conjunctive query answering in $AC^0$
- Based on DL-Lite family of description logics [Calvanese et al. 2005; 2006; 2008]
- Can exploit query rewriting based reasoning technique
  - Computationally optimal
  - Data storage and query evaluation can be delegated to standard RDBMS
  - Novel technique to prevent exponential blowup produced by rewritings [Kontchakov et al. 2010, Rosati and Almatelli 2010]
  - Can be extended to more expressive languages (beyond $AC^0$) by delegating query answering to a Datalog engine [Perez-Urbina et al. 2009]
Query Rewriting Technique (basics)

- Given ontology $\mathcal{O}$ and query $\mathcal{Q}$, use $\mathcal{O}$ to rewrite $\mathcal{Q}$ as $\mathcal{Q}'$ s.t., for any set of ground facts $\mathcal{A}$:
  - $\text{ans}(\mathcal{Q}, \mathcal{O}, \mathcal{A}) = \text{ans}(\mathcal{Q}', \emptyset, \mathcal{A})$
- Use (GAV) mapping $\mathcal{M}$ to map $\mathcal{Q}'$ to SQL query

![Diagram showing the process of query rewriting and mapping to SQL]

$\mathcal{Q}$ \rightarrow \text{Rewrite} \rightarrow \mathcal{Q}' \rightarrow \text{Map} \rightarrow \text{SQL} \rightarrow \mathcal{A} \rightarrow \text{Ans}$
**Query Rewriting Technique (basics)**

- Given ontology $\mathcal{O}$ and query $\mathcal{Q}$, use $\mathcal{O}$ to rewrite $\mathcal{Q}$ as $\mathcal{Q}'$ s.t., for any set of ground facts $\mathcal{A}$:
  - $\text{ans}(\mathcal{Q}, \mathcal{O}, \mathcal{A}) = \text{ans}(\mathcal{Q}', \emptyset, \mathcal{A})$
- Use (GAV) mapping $\mathcal{M}$ to map $\mathcal{Q}'$ to SQL query
- Resolution based query rewriting
  - **Clausify** ontology axioms
  - **Saturate** (clausified) ontology and query using resolution
  - **Prune** redundant query clauses
Example:

\[
\text{Doctor} \sqsubseteq \exists \text{treats.Patient}
\]
\[
\text{Consultant} \sqsubseteq \text{Doctor}
\]

\[Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)\]
Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats} \cdot \text{Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor}
\end{align*}
\]

\[
\begin{align*}
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x))
\end{align*}
\]
Query Rewriting Technique (basics)

Example:

Doctor ⊑ ∃treats.Patient
Consultant ⊑ Doctor

treats(x, f(x)) ← Doctor(x)
Patient(f(x)) ← Doctor(x)
Doctor(x) ← Consultant(x)

Q(x) ← treats(x, y) ∧ Patient(y)
Q(x) ← Doctor(x) ∧ Patient(f(x))
Q(x) ← treats(x, f(x)) ∧ Doctor(x)
Example:

\[
\text{Doctor} \sqsubseteq \exists \text{treats.} \text{Patient} \\
\text{Consultant} \sqsubseteq \text{Doctor}
\]

\[
\text{treats}(x, f(x)) \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) \leftarrow \text{Consultant}(x)
\]

\[
Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
Q(x) \leftarrow \text{Doctor}(x)
\]
Query Rewriting Technique (basics)

Example:

\[
\begin{align*}
\text{Doctor} & \sqsubseteq \exists \text{treats. Patient} \\
\text{Consultant} & \sqsubseteq \text{Doctor} \\
\text{treats}(x, f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Patient}(f(x)) & \leftarrow \text{Doctor}(x) \\
\text{Doctor}(x) & \leftarrow \text{Consultant}(x) \\
\end{align*}
\]

\[
\begin{align*}
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
Q(x) & \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \\
Q(x) & \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Doctor}(x) \\
Q(x) & \leftarrow \text{Consultant}(x) \\
\end{align*}
\]
Query Rewriting Technique (basics)

Example:

Doctor ⊑ ∃treats. Patient
Consultant ⊑ Doctor

treats(x, f(x)) ← Doctor(x)
Patient(f(x)) ← Doctor(x)
Doctor(x) ← Consultant(x)

\[ Q(x) ← \text{treats}(x, y) \land \text{Patient}(y) \]
\[ Q(x) ← \text{Doctor}(x) \land \text{Patient}(f(x)) \]
\[ Q(x) ← \text{treats}(x, f(x)) \land \text{Doctor}(x) \]
\[ Q(x) ← \text{Doctor}(x) \]
\[ Q(x) ← \text{Consultant}(x) \]

For DL-Lite, result is a union of conjunctive queries

\[ Q(x) ← (\text{treats}(x, y) \land \text{Patient}(y)) \lor \text{Doctor}(x) \lor \text{Consultant}(x) \]
Query Rewriting Technique (basics)

- UCQ translated into SQL query → OWL2QL can be "delegated" to RDBMS:

\[ Q(x) \leftarrow (\text{treats}(x, y) \land \text{Patient}(y)) \lor \text{Doctor}(x) \lor \text{Consultant}(x) \]  

\[ \downarrow \]

SELECT Name FROM Doctor UNION SELECT DName FROM Treats, Patient WHERE PName=Name
Interplay OWL2 ↔ RIF ↔ SPARQL1.1

- **OWL2 and RIF**
  - RIF fly-over
  - OWL2RL in RIF
  - RIF/OWL joint interpretations and what you need to know about them

- **OWL2 and SPARQL1.1**
  - SPARQL Entailment Regimes
  - Challenges+Pitfalls
  - What’s in the current SPARQL 1.1 Draft?

- **GiaBATA**
  - A prototype implementation of SPARQL with dynamic Entailment regimes (e.g. RDFS, OWL2RL).
RIF fly-over
OWL and RIF

- RIF: Rule Interchange Format (rather than Rule language)
  - Framework for Rule Languages
  - Support RDF import: interesting for rule languages on top of RDF
  - Built-Ins support (close to XPath/XQuery functions and operators)
  - RIF Dialects:
    - RIF BLD: basic logic dialect = Horn rules with Built-ins, Equality
    - RIF Core: Datalog fragment (no logical function symbols, no head-equality)
    - RIF PRD: Production rules dialect
  - Normative XML syntax

- Commonalities with OWL:
  - RIF can model OWL2 RL
  - Share same Datatypes (XSD Datatypes, most OWL2 Datatypes)

- Differences
  - Different target audience: E.g. production rules (RIF PRD dialect)
  - Not necessarily focused on decidability, BLD = generic HORN rules with built-ins and function symbols (Turing-complete language)
### RIF Dialects

**Core**
- horn rules, monotonic
- datatypes & built-ins
- external functions
- Frames, class memberships
- equality (in conditions)
- ground lists
- existential quantification (in conditions)

**BLD**
- equality, class membership in conclusions
- frame subclasses
- open lists

**PRD**
- non-monotonic
- actions in conclusions
- negation
- subclasses
- membership in conclusion
Example – RIF Core

- Full name in FOAF from givenName, familyName

if ({ ?X a foaf:Person ; foaf:givenName ?F ; foaf:familyName ?S } AND 
then { ?F foaf:name ?N }

□ Not expressible in OWL2, neither in SPARQL1.0 CONSTRUCT

CONSTRUCT { ?X foaf:name ?N }
WHERE {?X a foaf:Person; foaf:givenName ?F ; foaf:familyName ?S
  FILTER (?N = fn:concat(?F, " ", ?S)) }
Example – RIF Core

- Full name in FOAF from givenName, familyName

{ ?F foaf:name ?N } :-
{ ?X foaf:givenName ?F ; foaf:familyName ?S } AND

- Can be read like Logic Programming rule
- Presentation syntax not normative, we use a Mix of N3 and non-normative Presentation syntax in the spec here.
Example – RIF Core

- Full name in FOAF from givenName, familyName

```prolog
?f[->foaf:name ?N] :-
  ?x[foaf:givenName ?F], ?x[foaf:familyName->?S],
```

- Can be read like Logic Programming rule
- Presentation syntax not normative, we use a Mix of N3 and non-normative Presentation syntax in the spec here.
  - RIF has F-Logic style Frames (e.g. FLORA-2)… same semantics as RDF-Triples
  - Further in rif # corresponds to class membership, ## to subclassing
  - in combination with RDF, # is the same as rdf:type )
ATTENTION: Class membership # in conclusions is not in RIF Core.

{ ?x rdf:type ?y } :- ?x # ?y
?x # ?y :- { ?x rdf:type ?y }
Translating OWL2RL into RIF
OWL 2RL can be rewritten to RIF

- [http://www.w3.org/TR/rif-owl-rl/](http://www.w3.org/TR/rif-owl-rl/)

- Translates OWL2RL profile into RIF, relatively straightforward translation of abstract rules from [http://www.w3.org/TR/owl2-profiles/#Reasoning_in/owl2-RL_and_RDF_Graphs_using_Rules](http://www.w3.org/TR/owl2-profiles/#Reasoning_in/owl2-RL_and_RDF_Graphs_using_Rules)
  - Appendix 7: Static ruleset
  - Appendix 8: Dynamically instantiating a RIF Core rule set for a given OWL 2 RL, similar in spirit to the embedding in [http://www.w3.org/TR/rif-rdf-owl/](http://www.w3.org/TR/rif-rdf-owl/) Section 9.2
Static ruleset

- Some rules straightforward, e.g.
  
  ```
  prp-ifp: { ?S1 owl:SameAs ?S2 } :-
  ```

  ```
  ```

- Others need auxiliary predicates for the static version:

  ```
  ```

  We’d need that rule for all \( n \), i.e. different property chain lengths appearing in the ontology at hand.
prp-spo2: **Can be handled with auxiliary predicates:**

```
   AND _checkChain(?S ?pc ?O_{n+1})

_checkChain(?start ?pc ?last) :-  

_checkChain(?start ?pc ?last) :- And (  
   _checkChain(?next ?tl ?last) )
```
Other rules, e.g. subclassOf, inverseOf

cax-sco:


prp-inv1:


prp-inv2:


Similarly for other rules:

- all of OWL2RL can be translated to RIF Core rules, fed into your favorite rules engine, used for
  - Query answering,
  - Consistency checking
Idea:
- Translate each ontology axiom by axiom dynamically
- E.g. ontology in RDF

```
foaf:Person rdfs:subClassOf foaf:Agent

foaf:topic owl:inverseOf foaf:page
```
Dynamic ruleset (Template-Rules)

- **Idea:**
  - Translate each ontology axiom by axiom dynamically
  - E.g. ontology in RDF

- **Matching Template Rules in Appendix 8.2:**

  ```
  { ?x rdf:type foaf:Agent} :- { ?x rdf:type foaf:Person }
  
  ```

*Plus some fixed ruleset (Appendix 8.1 FixedRules in http://www.w3.org/TR/rif-owl-rl/*
Embedding OWL2RL Ontologies into RIF for combinations with arbitrary RIF rulesets
RIF + RDF and OWL in combination

- **RIF/OWL joint interpretations**
  - [http://www.w3.org/TR/rif-rdf-owl/](http://www.w3.org/TR/rif-rdf-owl/) defines semantic correspondence between RIF and RDF/RDFS/OWL interpretations,
  - i.e., semantics for combinations of RDF graphs, OWL ontologies and RIF rulesets
- **Defines:**
  - **RIF-OWL-Direct Entailment:** Based on OWL direct semantics
    - RIF-OWL-DL combination disallows certain RIF documents (only constants for classes in #, ##, only constants for predicates in frames) ...
  - **RIF-OWL RDF-Based Entailment:**
    - Based on OWL RDF-Based Semantics.
(Informative) Embedding in http://www.w3.org/TR/rif-rdf-owl/ give rise for implementation of combination of OWL2RL and RIF:

1. Embedding RIF DL-document formulas into RIF BLD, Section 9.2.1
2. Embedding OWL 2 RL axioms into RIF BLD, Section 9.2.2

We focus on the latter part…
Section 9.2.2 defines recursive translation from OWL axioms to RIF rules… tr()

Very similar to Dynamic Rules we saw before

E.g. OWL2RL ontology in RDF

```xml
foaf:Person rdfs:subClassOf foaf:Agent

foaf:topic owl:inverseOf foaf:page
foaf:topic type owl:ObjectProperty
foaf:page type owl:ObjectProperty
```
Section 9.2.2 defines recursive translation from OWL axioms to RIF rules… \( \text{tr()} \)

Very similar to Dynamic Rules we saw before

Translated to OWL abstract syntax axioms:

\[
\text{SubClassOf}(\text{foaf:Person} \ \text{foaf:Agent})
\]

\[
\text{InverseObjectProperties}(\text{foaf:topic} \ \text{foaf:page})
\]
Section 9.2.2 defines recursive translation from OWL axioms to RIF rules… tr()

Very similar to Dynamic Rules we saw before

Translated to RIF by translation tr() in Section 9.2.2 of http://www.w3.org/TR/rif-rdf-owl/:

```
{ ?x rdf:type foaf:Agent} :- { ?x rdf:type foaf:Person }

```

*Plus some static ruleset (ROWL-Direct(V,R))*
Subtle differences to direct OWL2RL translation from before:

- Most fundamentally **equality**: owl:sameAs is directly translated to RIF’s =, rather than axiomatised as in slide 54:

  - **OWL RDF**:

    ```
    ```

    is embedded as:

    ```
    <http://a> = <http://b>
    ```

- **E.g. in Combination with RIF ruleset**:

  ```
  _q(<http://a>).
  _p(?x) :- iri-to-string(?y, ?x) and _q(?y)
  ```

  entails:

  ```
  _p("http://a").
  _p("<http://b>").
  ```

- Not so if I take the axiomatisation of sameAs from above

- **Bottomline**: To straightforwardly implement the embedding for combinations, You need a rule system that supports equality.
SPARQL 1.1 querying over OWL2 ontologies?
OWL2 and SPARQL1.1

- SPARQL1.1 working group will define SPARQL query answering over OWL2 ontologies and RIF rule sets:
  - [http://www.w3.org/TR/sparql11-entailment/](http://www.w3.org/TR/sparql11-entailment/)

- Latest Working Draft just released...
  - Contains Draft Semantics for
    - SPARQL1.1 on top of RDFS
    - SPARQL1.1 on top of OWL2
    - SPARQL1.1 on top of RIF
OWL2 and SPARQL1.1

- General Idea: Answer Queries with implicit answers
- E.g. Graph

```
foaf:Person rdfs:subClassOf foaf:Agent.
foaf:Person rdfs:subclassOf
    [ a owl:Restriction;
      owl:onProperty :hasFather;
      owl:someValuesFrom foaf:Person. ]

:jeff a Person
:jeff foaf:knows :aidan
foaf:knows rdfs:range foaf:Person.
```

```
SELECT ?X { ?X a foaf:Person }
```

Pure SPARQL 1.0 returns only :Jeff, should also return :aidan
SPARQL+RDFS/OWL: Challenges+Pitfalls

- **Challenges+Pitfalls:**
  - Possibly Infinite answers (by RDFS ContainerMembership properties, OWL datatype reasoning, etc.)
  - Conjunctive Queries: non-distinguished variables
  - SPARQL 1.1 features: Aggregates
SPARQL+RDFS/OWL: Challenges+Pitfalls

Current Solution:

- Possibly Infinite answers (by RDFS ContainerMembership properties, OWL datatype reasoning, etc.)
  - Restrict answers to `rdf:/rdfs:/owl:vocabulary` minus `rdf:_1` ... `rdf:_n` plus terms occurring in the data graph

- Non-distinguished variables
  - No non-distinguished variables, answers must result from BGP matching, projection a post-processing step not part of entailment.

- SPARQL 1.1 other features: Aggregates
  - Again not affected, answers must result from BGP matching, projection a post-processing step not part of entailment.

- Simple, BUT: maybe not always entirely intuitive, so
  - Good to know ;-)
**Possibly Infinite answers RDF(S): Container Membership**

- **Graph:**

  ```
  :me :hasFavouritePresenter [ a rdf:Seq;
    rdf:_1 :jeff.
    rdf:_2 :aidan.
    rdf:_3 :axel. ]
  ```

- **Query with RDFS Entailment in mind:**

  ```
  SELECT ?CM { ?CM a rdfs:ContainerMembershipProperty}
  ```

- **Entailed by RDFS (axiomatic Triples):**

  ```
  rdfs:_1 a rdfs:ContainerMembershipProperty .
  rdfs:_2 a rdfs:ContainerMembershipProperty .
  rdfs:_3 a rdfs:ContainerMembershipProperty .
  rdfs:_4 a rdfs:ContainerMembershipProperty .
  ```
Graph:

:me :hasFavouritePresenter [ a rdf:Seq;
    rdf:_1 :jeff.
    rdf:_2 :aidan.
    rdf:_3 :axel. ]

Query with RDFS Entailment in mind:

SELECT ?CM { ?CM a rdfs:ContainerMembershipProperty}

SPARQL 1.1 restricts answers to rdf:/rdfs:/owl:vocabulary minus rdf:_1 ... rdf:_n plus terms occurring in the data graph

So, the only answers are:

{ ?CM/rdfs:_1, ?CM/rdfs:_2, ?CM/rdfs:_3 }
Possibly Infinite answers OWL: datatype reasoning

Stupid way to say Peter is 50:

ex:Peter a [ a owl:Restriction ;
    owl:onProperty ex:age ;
    owl:allValuesFrom [ rdf:type rdfs:Datatype .
    owl:oneOf ("50"^^xsd:integer) ] ]

Stupid query asking What is NOT Peters age:
SELECT ?x WHERE {
    ex:Peter a [ a owl:Restriction ; owl:onProperty ex:age ;
    owl:allValuesFrom [ a rdfs:Datatype ;
    owl:datatypeComplementOf [ a
    rdfs:Datatype ; owl:oneOf (?x) ] ] ]
}

Theoretical answer: all literal different from 50
No danger in SPARQL 1.1 restricts answers to rdf:/rdfs:/owl:vocabulary minus rdf:_1 ... rdf:_n plus terms occurring in the data graph
Non-distinguished variables:

- **E.g. Graph**

  
  foaf:Person rdfs:subClassOf foaf:Agent .
  
  foaf:Person rdfs:subclassOf
  
  [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person. ]
  
  :jeff a Person
  
  :jeff foaf:knows :aidan
  
  foaf:knows rdfs:range foaf:Person.

```
SELECT ?X ?Y { ?X :hasFather ?Y }
```

No answer, because no known value for ?Y in the data graph.
Non-distinguished variables:

- E.g. Graph

  foaf:Person rdfs:subClassOf foaf:Agent .
  foaf:Person rdfs:subclassOf
  [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person. ]

  :jeff a Person
  :jeff foaf:knows :aidan
  foaf:knows rdfs:range foaf:Person.

  SELECT ?X { ?X :hasFather ?Y }

  But what about this one? ?Y looks like a “non-distinguished” variable
  Solution: In SPARQL 1.1 answers must result from BGP matching,
    projection a post-processing step not part of entailment ➔ so, still no
    answer.
Non-distinguished variables:

- Simple Solution may seem not always intuitive, but:
  - OWL Entailment in SPARQL based on BGP matching, i.e.
    - always only returns results with named individuals
    - Doesn’t affect SELECT: takes place before projection
    - That is: non-distinguished variables can't occur “by design”

- In fact, conjunctive queries with non-distinguished variable still
  an open research problem for OWL:
  - Decidable for SHIQ, [B. Glimm et al. 2008]
  - Decidable for OWL1 DL without transitive properties OWL1 Lite
    without nominals [B. Glimm, KR-10]
  - Decidability for SHOIN, SROIQ though still unknown...
Similar as before... aggregates are evaluated as post-processing after BGP matching, so, no effect:

```sparql
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person rdfs:subClassOf
    [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person. ]
:jeff a Person
:jeff foaf:knows :aidan
foaf:knows rdfs:range foaf:Person.
```

SELECT ?X { ?X a foaf:Person }

Under RDFS/OWL entailment returns : {?X/jeff, ?X/aidan}
**SPARQL 1.1 other features: Aggregates**

- Similar as before... aggregates are evaluated as post-processing after BGP matching, so, no effect:

  ```sparql
  foaf:Person rdfs:subClassOf foaf:Agent .
  foaf:Person rdfs:subclassOf
  [ a owl:Restriction ;
    owl:onProperty :hasFather ;
    owl:someValuesFrom foaf:Person. ]

  :jeff a Person
  :jeff foaf:knows :aidan
  foaf:knows rdfs:range foaf:Person.

  SELECT ?Y AS Count(?X) { ?X a foaf:Person }
  
  Under RDFS/OWL entailment returns: {?Y/2}
  ```
GiaBATA

Implementing SPARQL, OWL2RL, RIF on top of DLV
Time allowed, we will show a system which implements dynamic SPARQL querying, under different entailment regimes and how it can be implemented.

Based on LP engine DLV
  - Datalog with built-ins (covers roughly RIF Core),
  - persistent Database backend (DLV-DB)
  - Optimisations (rewriting to push join processing into SQL as far as possible, magic sets,...)
  - plus a lot more features (nonmonotonicity, aggregates, ...)

Overall idea for SPARQL+RDFS/OWL2RL over RDF graphs:
  - Translate OWL2RL to Datalog rules a la RIF, see above.
  - Translate SPARQL query to Datalog [Polleres, WWW2007]
  - Feed resulting program into a rules engine (DLV-DB) that runs over a Rel DB storing RDF graphs.

Check Details at:
How to implement this?

- **GiaBATA system [lanni et al., 2009]:**
  - SPARQL $\rightarrow$ dlvhex (logic program) $\rightarrow$ SQL
  - Ruleset $\rightarrow$ dlvhex (logic program)

- **Deductive Database techniques:**
  - Datalog engine (dlvhex)
  - Postgres SQL Database underneath (dlv-db)
  - RDF storable in different schemas in RDB
  - Magic sets, storage
Based on [Polleres, WWW2007]

SPARQL $\rightarrow$ dlvhex (logic program)

```
select * from <http://alice.org/>
where { ?X a foaf:Person. ?X foaf:name ?N.
  filter ( ?N != "Alice") optional { ?X foaf:mbox ?M } }
```

(r1) "triple"(S,P,O,default) :- &rdf[ "alice.org" ](S,P,O).
(r2) answer1(X_N,X_X,default) :- "triple"(X_X,"rdf:type","foaf:Person",default),
  "triple"(X_X,"foaf:name",X_N,default),
  &eval[ "?N != 'Alice' ","N", X_N ](true).
(r3) answer2(X_M,X_X,default) :- "triple"(X_X,"foaf:mbox",X_M,default).
(r4) answer_b_join_1(X_M,X_N,X_X,default) :- answer1(X_N,X_X,default),
  answer2(X_M,X_X,default).
(r5) answer_b_join_1(null,X_N,X_X,default) :- answer1(X_N,X_X,default),
  not answer2_prime(X_X,default).
(r6) answer2_prime(X_X,default) :- answer1(X_N,X_X,default),
  answer2(X_M,X_X,default).
(r7) answer(X_M,X_N,X_X) :- answer_b_join1(X_M,X_N,X_X,default).
```
OWL2RL Static Ruleset → dlvhex (logic program)

- Straightforward, just translates rules in a way “compatible” with the SPARQL translation:

\[
\{\text{?s ?q ?o } \}\ \leq\ \{\text{?s ?p ?o } . \ ?p \text{ rdfs:subPropertyOf ?q}\}
\]

%FROM CLAUSES
triple(P,SubPropertyOf,P,G) :- triple(P,Type,Property,G),graph(G,D),data(D),defaultGraph(D),
resource_literal(Type,"http://www.w3.org/1999/02-rdf-syntax-ns#type","_"),
resource_literal(Property,"http://www.w3.org/1999/02-rdf-syntax-ns#Property","_"),
resource_literal(SubPropertyOf,"http://www.w3.org/2000/01/rdf-schema#subPropertyOf","_").

%FROM NAMED CLAUSES
triple(P,SubPropertyOf,P,G) :- triple(P,Type,Property,G),graph(G,D),data(D),namedGraph(D),
resource_literal(Type,"http://www.w3.org/1999/02-rdf-syntax-ns#type","G"),
resource_literal(Property,"http://www.w3.org/1999/02-rdf-syntax-ns#Property","G"),
resource_literal(SubPropertyOf,"http://www.w3.org/2000/01/rdf-schema#subPropertyOf","G").

%USING ONTOLOGIES
triple(P,SubPropertyOf,P,G) :- triple(P,Type,Property,G),graph(G,D),data(D),ontology(D),
resource_literal(Type,"http://www.w3.org/1999/02-rdf-syntax-ns#type","G"),
resource_literal(Property,"http://www.w3.org/1999/02-rdf-syntax-ns#Property","G"),
resource_literal(SubPropertyOf,"http://www.w3.org/2000/01/rdf-schema#subPropertyOf","G").

SPARQL+Rules $\rightarrow$ SQL

- Done by dlv-DB, cf. [Terracina, et al., 2008]
  - All non-recursive parts are pushed to the Database
  - All recursive parts handled by semi-naïve evaluation
    (more efficient than WITH RECURSIVE views in SQL, where necessary, intermediate results temporarily materialized into the DB)

- Some necessary optimisations to make this reasonably performant:
  - FILTER expression evaluation is pushed to SQL (3-valued semantics of SPARQL Filters is handled natively in SQL)
  - No miracles… but magic: Magic set optimisations for focused fwd-chaining evaluation.
  - Join-reordering, not yet implemented, but we did some manual reordering to optimize the query plan in the experiments.
References OWL2, Scalable/Tractable
OWL Reasoning:

References OWL2, Scalable/Tractable OWL Reasoning:


References on RDF and SPARQL Querying


Relevant W3C Standard Specs

- **RDF Semantics** [http://www.w3.org/TR/rdf-mt/](http://www.w3.org/TR/rdf-mt/)
- **OWL2 Web Ontology Language Primer** [http://www.w3.org/TR/owl2-primer/](http://www.w3.org/TR/owl2-primer/)
- **OWL2 Web Ontology Language Profiles** [http://www.w3.org/TR/owl2-profiles/](http://www.w3.org/TR/owl2-profiles/)
- **SPARQL Query Language for RDF** [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)
- **SPARQL1.1 Query Language for RDF** (working draft) [http://www.w3.org/TR/sparql11-query/](http://www.w3.org/TR/sparql11-query/)
- **SPARQL1.1 Entailment Regimes** (working draft) [http://www.w3.org/TR/sparql11-entailment/](http://www.w3.org/TR/sparql11-entailment/)
- **RIF Core Dialect** [http://www.w3.org/TR/rif-core/](http://www.w3.org/TR/rif-core/)
- **RIF Basic Logic Dialect** [http://www.w3.org/TR/rif-bld/](http://www.w3.org/TR/rif-bld/)
- **RIF RDF and OWL compatibility** [http://www.w3.org/TR/rif-rdf-owl/](http://www.w3.org/TR/rif-rdf-owl/)
Other Tutorials

- Some more basic lectures&Tutorials on my Website: [http://www.polleres.net/] e.g.
  - Semantic Web and ASP Tutorial ESWC2006
  - SPARQL Tutorial ESWC2007
  - Scalable OWL Reasoning Tutorial ESWC2010

- Also recommended:
  - Reasoning Web Summer Schools (since 2005), many good tutorials/slides:
    - [http://reasoningweb.org/2005/]
    - [http://reasoningweb.org/2006/]
    - [http://reasoningweb.org/2007/]
    - [http://reasoningweb.org/2008/]
    - [http://reasoningweb.org/2009/]
    - [http://reasoningweb.org/2010/]

- Linked Data Tutorial:
  - [http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/]
Acknowledgements

- Jeff Pan
- Aidan Hogan
- Giovambattista Ianni
- Alessandra Martello
- Thomas Krennwallner
- The members of the W3C OWL2 WG
- The members of the W3C RIF WG
- The members of the W3C SPARQL WG