SPARQL1.1: new features and friends (OWL2, RIF)

@AxelPolleres
Digital Enterprise Research Institute, National University of Ireland, Galway
Chair-hat off: While trying to convey best I can the current status of the SPARQL Working Group, in this Tutorial I don’t claim to speak officially for the group and will freely blend in my personal opinions… ;-)

What is SPARQL?

- **Query Language for RDF**
  - SQL “look-and-feel” for the Semantic Web
  - Means to query the Web of Data
  - Means to map between vocabularies
  - Means to access RDF stores

- **SPARQL1.0 (standard since 2008):**
  - Query Language
  - Protocol
  - Result Format

- **SPARQL1.1 (in progress):**
  - SPARQL 1.1 query language (additional features: aggregate functions, subqueries, negation, project expressions, property paths, basic federated queries)
  - SPARQL 1.1 Entailment regimes
  - SPARQL 1.1 Update: A full data manipulation language
  - SPARQL 1.1 Uniform HTTP Protocol for Managing RDF Graphs
  - SPARQL 1.1 Service Descriptions
What you’ll hear

- Run through SPARQL1.0
- SPARQL 1.0 Formal Semantics, and theoretical results
- New features in SPARQL 1.1 Query
- SPARQL 1.1 Entailment Regimes
  - Relation to OWL2
  - Relation to RIF
- Implementations, Status
RDF a plain data format for the Web

Various syntaxes, RDF/XML, Turtle, N3, RDFa,…

Subject U x B

Predicate U

Object U x B x L

URIs, e.g.
http://www.w3.org/2000/01/rdf-schema#label
http://ontology.dumontierlab.com/isLocatedIn
http://dbpedia.org/resource/Brixen
http://dbpedia.org/resource/Italy

Blanknodes:
“existential variables in the data” to express incomplete information, written as _:x or []

Literals, e.g.
"2010"^^xsd:gYear
"Brixen"@de
"Bressanone"@it
"Diego Calvanese"

5
RDF Data on the Web: Linked Open Data

The New York Times

July 2009
(i) directly by the publishers
(ii) by exporters

FOAF/RDF linked from a home page: personal data (foaf:name, foaf:phone, etc.), relationships foaf:knows, rdfs:seeAlso)
RDF Data online: Example 2/3

- (i) directly by the publishers
- (ii) by exporters, e.g. D2R and friends, 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 and even a SPARQL query interface!
RDF Data online: Example 3/3

http://dblp.l3s.de/d2r/resource/authors/T

Tim Berners-Lee

Resource URI: http://dblp.l3s.de/d2r/resource/authors/Tim_Berners-Lee

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>is dc:creator of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/aaai/KagalBCW06">http://dblp.l3s.de/d2r/resource/publications/conf/aaai/KagalBCW06</a></td>
</tr>
<tr>
<td>is dc:creator of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/chi/schraefel/AWTBCJKDMSSW09">http://dblp.l3s.de/d2r/resource/publications/conf/chi/schraefel/AWTBCJKDMSSW09</a></td>
</tr>
<tr>
<td>is dc:creator of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/esws/OmitolaKPYSSBGhsS10">http://dblp.l3s.de/d2r/resource/publications/conf/esws/OmitolaKPYSSBGhsS10</a></td>
</tr>
<tr>
<td>is dc:creator of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/policy/HansonBKW07">http://dblp.l3s.de/d2r/resource/publications/conf/policy/HansonBKW07</a></td>
</tr>
<tr>
<td>is dc:creator of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/policy/KagalBCW06">http://dblp.l3s.de/d2r/resource/publications/conf/policy/KagalBCW06</a></td>
</tr>
<tr>
<td>foaf:homepage</td>
<td><a href="http://www.w3.org/People/Berners-Lee/">http://www.w3.org/People/Berners-Lee/</a></td>
</tr>
<tr>
<td>rdfs:label</td>
<td>Tim Berners-Lee</td>
</tr>
<tr>
<td>is foaf:maker of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/aaai/KagalBCW06">http://dblp.l3s.de/d2r/resource/publications/conf/aaai/KagalBCW06</a></td>
</tr>
<tr>
<td>is foaf:maker of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/chi/schraefel/AWTBCJKDMSSW09">http://dblp.l3s.de/d2r/resource/publications/conf/chi/schraefel/AWTBCJKDMSSW09</a></td>
</tr>
<tr>
<td>is foaf:maker of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/esws/OmitolaKPYSSBGhsS10">http://dblp.l3s.de/d2r/resource/publications/conf/esws/OmitolaKPYSSBGhsS10</a></td>
</tr>
<tr>
<td>is foaf:maker of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/policy/HansonBKW07">http://dblp.l3s.de/d2r/resource/publications/conf/policy/HansonBKW07</a></td>
</tr>
<tr>
<td>is foaf:maker of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/policy/KagalBCW06">http://dblp.l3s.de/d2r/resource/publications/conf/policy/KagalBCW06</a></td>
</tr>
<tr>
<td>foaf:name</td>
<td>Tim Berners-Lee</td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
<td><a href="http://dblp.l3s.de/Authors/Authors/TimBerners-Lee">http://dblp.l3s.de/Authors/Authors/TimBerners-Lee</a></td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
<td><a href="http://www.bibonomy.org/uri/author/Tim-Berners-Lee">http://www.bibonomy.org/uri/author/Tim-Berners-Lee</a></td>
</tr>
<tr>
<td>rdf:type</td>
<td>foaf:Agent</td>
</tr>
</tbody>
</table>
DBLP Data in RDF: **Triples Turtle Syntax**:

```turtle
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix swrc: <http://swrc.ontoware.org/ontology#> .

<http://dblp.l3s/journals/tplp/Berners-LeeCKSH08> dcterms:issued "2008"^^xsd:gYear .
...
...
<http://dblp.l3s/Tim_Berners-Lee> foaf:homepage <http://www.w3.org/People/Berners-Lee/> .
```
DBLP Data in RDF: **Triples Turtle Syntax**:

```turtle
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix swrc: <http://swrc.ontoware.org/ontology#> .

<http://dblp.l3s.../journals/tplp/Berners-LeeCKSH08> rdf:type swrc:Article ;
  dcterms:issued "2008"^^xsd:gYear ;
  foaf:maker <http://dblp.l3s.../Tim_Berners-Lee> ,
  <http://dblp.l3s.../Dan_Connolly> ,
  <http://dblp.l3s.../Jim_Hendler> ,
  <http://dblp.l3s.../Lalana_Kagal> ,
  <http://dblp.l3s.../Yosi_Scharf> .

... 

<http://dblp.l3s.../conf/aaai/KagalBCW06> rdf:type swrc:inProceedings ;

... 

<http://dblp.l3s.../Tim_Berners-Lee> foaf:homepage <http://www.w3.org/People/Berners-Lee/> ;
  foaf:name "Tim Berners-Lee" .
```
Linked Data: What’s the point?

- Loads of **structured data** out there

- You want to do **structured queries** on top of it …

- SPARQL1.0  W3C Rec 15 January 2008… Now you can!

- Without exaggeration, SPARQL is probably a not too small a part of the LOD success story! … at least an important building block
How can I query that data? SPARQL

Basic graph pattern matching ~ Conjunctive queries

Example:

“Give me all documents by Tim Berners-Lee”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?D
FROM <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>
WHERE { ?D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee> }

FROM clause/Dataset can be implicit, e.g. when querying DBLP's SPARQL endpoint
How can I query that data? SPARQL

Basic graph pattern matching ~ Conjunctive queries

Example:
“Give me all names of co-authors of Tim Berners-Lee”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE { _:D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
[ foaf:name ?N ] } . }

- Blank nodes in Queries play a similar role as (non-distinguished) variables.
- Turtle style shortcuts are allowed (a bit extreme here, admittedly)
More complex patterns in SPARQL 1.0

- UNION
- OPTIONAL
- FILTER
- Querying named GRAPHs
- Solution Modifiers (ordering, slicing/dicing results)
- ... plus some non-trivial combinations of these
UNIONs of conjunctive queries...

Unions of conjunctive queries

Example:
“Give me all names of co-authors or friends of Tim Berners-Lee”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE {

?N
"Lalana Kagal"
"Tim Berners-Lee"
"Dan Connolly"
"Jim Hendler"
...

} \n
UNIONs of conjunctive queries…

Note: Duplicates possible, bag/multiset semantics!

?N
"Lalana Kagal"
"Tim Berners-Lee"
"Dan Connolly"
"Jim Hendler"
...

?N
"Michael Hausenblas"
"Jim Hendler"
"Charles McCathieNevile"
...

U =

?N
"Michael Hausenblas"
"Jim Hendler"
"Charles McCathieNevile"
...
Avoid Duplicates: keyword `DISTINCT`

Example:

"Give me all names of co-authors or friends of Tim Berners-Lee"

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT `DISTINCT` ?N
WHERE {
  { [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
      [ foaf:name ?N ] ] . } 
  UNION 
  { <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F . 
    ?F foaf:name ?N } 
}

<table>
<thead>
<tr>
<th>?N</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lalana Kagal&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Tim Berners-Lee&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Dan Connolly&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Jim Hendler&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Michael Hausenblas&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Jim Hendler&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Charles McCathieNevile&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Michael Hausenblas&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Charles McCathieNevile&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Unions of conjunctive queries

Example:
“Give me all names of co-authors or friends of Tim Berners-Lee”

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?CoAuthN ?FrN
WHERE {
  { [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee> [ foaf:name ?CoAuthN ] ] . }
  UNION
  { <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F . ?F foaf:name ?FrN }
}
```

<table>
<thead>
<tr>
<th>?CoAuthN</th>
<th>?FrN</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lalana Kagal&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Tim Berners-Lee&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Dan Connolly&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Jim Hendler&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Michael Hausenblas&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Jim Hendler&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Charles McCathieNevile&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Optional parts in queries (Left Outer Join)

Example:

“Give me all names of co-authors of Tim Berners-Lee and optionally their homepage”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N ?H
WHERE {
  ?D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>.
  OPTIONAL { ?CoAuth foaf:homepage ?H }
}

Another example where variables can be unbound in results!

<table>
<thead>
<tr>
<th>N</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lalana Kagal&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Tim Berners-Lee&quot;</td>
<td><a href="http://www.w3.org/People/Berners-Lee/">http://www.w3.org/People/Berners-Lee/</a></td>
</tr>
<tr>
<td>&quot;Dan Connolly&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Daniel J. Weitzner&quot;</td>
<td><a href="http://www.w3.org/People/Weitzner.htm">http://www.w3.org/People/Weitzner.htm</a></td>
</tr>
<tr>
<td>&quot;m. c. schraefel&quot;</td>
<td><a href="http://www.ecs.soton.ac.uk/~mo/">http://www.ecs.soton.ac.uk/~mo/</a></td>
</tr>
<tr>
<td>&quot;Paul André&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Ryen White&quot;</td>
<td><a href="http://www.dcs.gla.ac.uk/~whiter/">http://www.dcs.gla.ac.uk/~whiter/</a></td>
</tr>
<tr>
<td>&quot;Desney S. Tan&quot;</td>
<td><a href="http://research.microsoft.com/~%7Edesney/">http://research.microsoft.com/~%7Edesney/</a></td>
</tr>
<tr>
<td>&quot;Tim Berners-Lee&quot;</td>
<td><a href="http://www.w3.org/People/Berners-Lee/">http://www.w3.org/People/Berners-Lee/</a></td>
</tr>
<tr>
<td>&quot;Sunny Consolvo&quot;</td>
<td>-</td>
</tr>
</tbody>
</table>
FILTERING out query results

**FILTERs** allow to specify FILTER conditions on patterns

Example:

"Give me all names of co-authors of Tim Berners-Lee and whose homepage starts with http://www.w3 different from Tim B.-L. himself"

```
  ?D foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>. 
  FILTER( regex( str(?H) , "^http://www.w3" ) && 
  ?CoAuth != <http://dblp.l3s.de/.../authors/Tim_Berners-Lee> ) }
```

"Daniel J. Weitzner" <http://www.w3.org/People/Weitzner.html>
FILTERING out query results

**FILTERs** allow to specify FILTER conditions on pattern

- Can use an extensible library of built-in functions
  - **checking**: bound(), isIRI(), isBlank(), regex() …
  - **Conversion/extraction**: str(), datatype(), lang() …
- Can be complex: &&, ||, !
- **ATTENTION**: Evaluated in a 3-valued logic: true, false, error

**Example:**

``` reasoning
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N ?H
WHERE {
  ?D foaf:maker <http://dblp.l3s.de/…/authors/Tim_Berners-Lee>.
  OPTIONAL { ?CoAuth foaf:homepage ?H . }
  FILTER(!regex(str(?H), "^http://www.w3") &&
  ?CoAuth != <http://dblp.l3s.de/…/authors/Tim_Berners-Lee> )
}
```

Will result in E for unbound ?H

> Whole FILTER expr always E for unbound ?H
ATTENTION: FILTERs can NOT assign/create new values…

PREFIX ex: <http://example.org/>
SELECT ?Item ?NewP

Obviously, common query languages like SQL can do this…

SELECT Item, Price+10 AS NewPrice FROM Table

… FILTER in SPARQL is like WHERE in SQL, but SPARQL1.0 doesn’t have AS
Find me people who have been involved with at least three ISWC or ESWC conference events.

(from SPARQL endpoint at data.semanticweb.org)

```
SELECT ?person WHERE {
  GRAPH ?g1 { ?person a foaf:Person }
  GRAPH ?g2 { ?person a foaf:Person }
  GRAPH ?g3 { ?person a foaf:Person }
  FILTER(?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3) . }
```

- The GRAPH ?g construct allows a pattern to match against one of the named graphs in the RDF dataset. The URI of the matching graph is bound to ?g (or whatever variable was actually used).
- The FILTER assures that we're finding a person who occurs in three distinct graphs.
Slicing and Dicing results

- Solution Modifiers
  - DISTINCT
  - ORDER BY
  - LIMIT/OFFSET

- Example:

```sql
SELECT DISTINCT ?person WHERE {
    GRAPH ?g1 { ?person a foaf:Person }
    GRAPH ?g2 { ?person a foaf:Person }
    GRAPH ?g3 { ?person a foaf:Person }
    FILTER(?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3) .
}
ORDER BY ?person
LIMIT 10
```
ASC, DESC, ORDER BY Expressions
More complex query examples 1/2

- **“IF-THEN-ELSE”**
  - “Give me the names of persons, if it exists, otherwise the nicknames, if it exists, otherwise the labels”
    
    ```
    SELECT ?X ?N
    WHERE{ ?X rdf:type foaf:Person
      OPTIONAL { ?X foaf:name ?N }
      OPTIONAL { ?X foaf:nickname ?N }
      OPTIONAL { ?X rdfs:label ?N }
    }
    ```

- **“Conditional OPTIONAL”**
  - “Give me the names and - only of those whose name starts with ‘D’ - the homepage”
    
    ```
    SELECT ?N ?H
    WHERE{ ?X foaf:name ?N
      OPTIONAL { ?X foaf:homepage ?H
        FILTER ( regex( str(?N), "^D" ) ) }
    }
    ```

*Non-compositionality raised some eyebrows... [Angles&Gutierrez, 2008] showed that compositional semantics can be achieved by rewriting.*
Negation

“Give me all Persons without a homepage”

Option 1: by combination of OPTIONAL and FILTER(!bound(… ) )

```
SELECT ?X
WHERE{ ?X rdf:type foaf:Person
    OPTIONAL { ?X foaf:homepage ?H }
    FILTER( !bound( ?H ) ) }
```

Option 2: by even weirder combination of OPTIONAL with GRAPH queries...

```
SELECT ?X
WHERE{ ?X rdf:type foaf:Person
    OPTIONAL { ?X foaf:homepage ?H }
    GRAPH boundcheck.ttl {?H :is :unbound }
}
```

where the aux. graph boundcheck.ttl contains the single triple [] :is :unbound.
Constructing Graphs

Construct new graphs:

- “everybody knows their co-authors”

CONSTRUCT { ?X foaf:knows ?Y }
  FILTER( ?X != ?Y ) }
Map between ontologies:
- E.g. for expressing complex ontology mappings between **FOAF** and **SiOC**
- “an sioc:name of a sioc:User is a foaf:nick”

Actually, expressible in new OWL2 (but not in OWL1):

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

- **Limitations**
  - Again, no assignment, creation of values
    - How to concatenate first name and last name?
  
  - No aggregation (e.g. COUNT, SUM, ...):
    - How to create a graph that has publication count per person for DBLP?
    
    - No RDFS/OWL inference (so combining mappings in RDFS/OWL with queries in SPARQL not possible)
SPARQL1.0 Formal Semantics

- **Graph patterns:**
  - BGPs
  - \( P1 \ P2 \)
  - \( P \ \text{FILTER} \ R \)
  - \( P1 \ \text{UNION} \ P2 \)
  - \( P1 \ \text{OPTIONAL} \ P2 \)

- **Semantics**
  - \( \text{eval}(D(G), \text{graph pattern}) \) ...
  - \( D \) is a dataset,
  - \( G \) is the “active graph”
  - recursively defined for all graph patterns in Section 12.5 of
  - [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)
  - Spec. semantics is a bit hard to read ...
Easier to explain... let’s steal from that here and explain the diffs:

**Definition 1:**

The evaluation of the BGP P over a graph G, denoted by eval(P,G), is the set of all mappings \( \mu \) such that:

- \( \text{dom}(\mu) \) is exactly the set of variables occurring in P
- \( \mu(P) \subseteq G \)

**Example Graph:**

```turtle
:tim foaf:knows :jim.
:jim foaf:knows :tim.
```

**Example Pattern:**

\[
P = \{ ?X \text{ foaf:knows } ?Y \}.
\]

\[
eval(P,G) = \{ \mu_1 = \{ ?x \rightarrow \text{tim}, ?y \rightarrow \text{jim} \}, \mu_2 = \{ ?x \rightarrow \text{jim}, ?y \rightarrow \text{tim} \}, \mu_3 = \{ ?x \rightarrow \text{tim}, ?y \rightarrow \text{lalana} \} \}
\]
Algebra á la [Perez et al. 2006]

Definition 2:
mappings $\mu_1$, $\mu_2$ are compatible iff they agree in their shared variables.

Let $M_1$, $M_2$ be sets of mappings

Definition 3:

Join:
$M_1 \bowtie M_2 = \{ \mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2, \text{ and } \mu_1, \mu_2 \text{ are compatible} \}$

Union:
$M_1 \cup M_2 = \{ \mu \mid \mu \in M_1 \text{ or } \mu \in M_2 \}$

Diff:
$M_1 \setminus M_2 = \{ \mu \in M_1 \mid \text{forall } \mu' \in M_2, \mu \text{ and } \mu' \text{ are not compatible} \}$

LeftJoin:
$M_1 \bowtie M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2)$

Filter:
$M|_R = \{ \mu \mid \mu \in M \text{ and } \mu(R) = \text{true} \}$
Semantics full as per [Perez et al. 2006]

\[
\begin{align*}
\text{eval}(BGP, G) & \quad \quad \text{... see Definition 1} \\
\text{eval}(P_1 P_2, G) & = \text{eval}(P_1, G) \otimes \text{eval}(P_2, G) \\
\text{eval}(P_1 \text{ UNION } P_2, G) & = \text{eval}(P_1, G) \cup \text{eval}(P_2, G) \\
\text{eval}(P_1 \text{ OPTIONAL } P_2, G) & = \text{eval}(P_1, G) \oslash \text{eval}(P_2, G) \\
\text{eval}(P \text{ FILTER } R, G) & = \text{eval}(P, G) |_R
\end{align*}
\]
Unions of conjunctive queries

Example:

“Give me all names of co-authors or friends of Tim Berners-Lee”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE {
  { foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
    foaf:name ?N } .
  UNION
  { <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F .
    ?F foaf:name ?N }

Note: Duplicates possible, bag/multiset semantics!
ISSUE 2) Recall from before: FILTERS can make OPTIONAL non-compositional!

- "Conditional OPTIONAL"
  - "Give me the names and only of those whose name starts with ‘D’ the homepage"

```sparql
SELECT ?N ?H
WHERE { ?X foaf:name ?N
  OPTIONAL { ?X foaf:homepage ?H
    FILTER ( regex( str(?N), "^D" ) ) }
}
```

OPTIONAL is NOT modular/compositional! (?N is not considered unsafe in this FILTER)*

<table>
<thead>
<tr>
<th>N</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lalana Kagal&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Tim Berners-Lee&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Dan Connolly&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Daniel J. Weitzner&quot;<a href="http://www.w3.org/People/Weitzner.html">http://www.w3.org/People/Weitzner.html</a></td>
<td>-</td>
</tr>
<tr>
<td>&quot;m. c. schraefel&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Paul André&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Ryen White&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Desney S. Tan&quot;<a href="http://research.microsoft.com/%7Edesney/">http://research.microsoft.com/%7Edesney/</a></td>
<td>-</td>
</tr>
<tr>
<td>&quot;Tim Berners-Lee&quot;</td>
<td>-</td>
</tr>
<tr>
<td>&quot;Sunny Consolvo&quot;</td>
<td>-</td>
</tr>
</tbody>
</table>
Adapting [Perez et al. 2006]

**ISSUE1:** Algebra operations need to be adapted to multiset/bag semantics:

**ISSUE2:** non-compositionality of FILTERs in OPTIONAL

**Definition 3:**

**Join:**
\[ M_1 \bowtie M_2 = \{ \mu_1 \cup \mu_2 | \mu_1 \in M_1, \mu_2 \in M_2, \text{ and } \mu_1, \mu_2 \text{ are compatible} \} \]

**Union:**
\[ M_1 \cup M_2 = \{ \mu | \mu \in M_1 \text{ or } \mu \in M_2 \} \]

**Diff:**
\[ M_1 \setminus M_2 = \{ \mu \in M_1 | \forall \mu' \in M_2, \mu \text{ and } \mu' \text{ are not compatible} \} \]

**LeftJoin:**
\[ M_1 \bowtie M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2) \]

**Filter:**
\[ \text{M}|_R = \{ \mu | \mu \in M \text{ and } \mu(R) = \text{true} \} \]
Semantics as per SPARQL1.0 spec:

\[
\text{eval}(BGP, G) \quad \text{... see Definition 1}
\]

\[
\text{eval}(P_1 \ P_2, G) = \text{eval}(P_1, G) \ltimes \text{eval}(P_2, G)
\]

\[
\text{eval}(P_1 \text{ UNION } P_2, G) = \text{eval}(P_1, G) \cup \text{eval}(P_2, G)
\]

\[
\text{eval}(P \text{ FILTER } R, G) = \text{eval}(P, G)\mid_R
\]

\[
\text{eval}(P_1 \text{ OPTIONAL } \{P_2 \text{ FILTER } R\}, G) \text{ consists of all } \mu \text{ such that:}
\]

1. \(\mu = \mu_1 \cup \mu_2\), such that
   \(\mu_1 \subseteq \text{eval}(P_1, G)\) and \(\mu_2 \subseteq \text{eval}(P_2, G)\) are compatible and \(\mu (R) = \text{true}\), or
2. \(\mu \subseteq \text{eval}(P_1, G)\) and
   there is no compatible \(\mu_2 \subseteq \text{eval}(P_2, G)\) for \(\mu\), or
3. \(\mu \subseteq \text{eval}(P_1, G)\) and
   for any compatible \(\mu_2 \subseteq \text{eval}(P_2, G)\), \(\mu \cup \mu_2\) does not satisfy \(R\).
Academic works around SPARQL

- **SPARQL semantics**
  - [Perez et al. 2006] (pre-dates the spec) [Perez et al. 2009]

- **SPARQL equivalences**
  - also in [Perez et al. 2006], [Perez et al. 2009]
  - More in [Schmidt et al. 2010]

- **SPARQL expressivity**
  - Reducible to datalog with negation [Polleres 2007]
  - Other way around also works [Angles & Gutierrez 2008]

- **Proposed Extensions**
  - Aggregates [Polleres et al. 2007]
  - Property Paths [Alkhateeb et al. 2009], [Perez et al. 2008]
SPARQL1.1

WG might still change some of the syntax/semantics definitions presented here based on community input
This is where SPARQL1.1 starts

- Missing common feature requirements in existing implementations or requested urgently by the community:
  - Assignment/Project Expressions
  - Aggregate functions (SUM, AVG, MIN, MAX, COUNT, …)
  - Subqueries
  - Property paths
    - complaint: SPARQL1.0 isn’t quite a “graph” query language

- Ease of use:
  - Why is Negation “hidden” in SPARQL1.0?

- Interplay with other SW standards:
  - SPARQL1.0 only defined for simple RDF entailment
  - Other Entailment regimes missing:
    - RDF(S), OWL
    - OWL2
    - RIF
Goals of SPARQL1.1

- **Per charter** (http://www.w3.org/2009/05/sparql-phase-ll-charter.html)
  - “The scope of this charter is to extend SPARQL technology to include some of the features that the community has identified as both desirable and important for interoperability based on experience with the initial version of the standard.”

  ➔ No inclusion of new features that still require research
  ➔ Upwards compatible with SPARQL1.0
  ➔ The name SPARQL1.1 shall indicate an incremental change rather than any fundamental changes.
Goals of SPARQL 1.1

List of agreed features:

- **Additions to the Query Language:**
  - Project Expressions
  - Aggregate functions
  - Subqueries
  - Negation
  - Property Paths *(time permitting)*
  - Extend the function library *(time permitting)*
  - Basic federated Queries *(time permitting)*

- **Entailment *(time permitting)*

- **SPARQL Update**
  - Full Update language
  - plus simple RESTful update methods for RDF graphs (HTTP methods)

- **Service Description**
  - Method for discovering a SPARQL endpoint’s capabilities
  - Summary of its data

We will focus on these in today’s Tutorial.
Part 1: new query features

- Project Expressions
- Aggregate functions
- Subqueries
- Negation
- Property Paths
Assignments, Creating new values...

PREFIX ex: <http://example.org/>
SELECT ?Item (?Pr * 1.1 AS ?NewP )
WHERE { ?Item ex:price ?Pr }

Data:
@prefix ex: <http://example.org/> .
ex:lemonade1 ex:price 3 .
ex:beer1 ex:price 3. 
ex:winel ex:price 3.50 .
ex:liqueur1 ex:price "n/a".

Results:

<table>
<thead>
<tr>
<th>?Item</th>
<th>?NewP</th>
</tr>
</thead>
<tbody>
<tr>
<td>lemonade</td>
<td>3.3</td>
</tr>
<tr>
<td>beer</td>
<td>3.3</td>
</tr>
<tr>
<td>wine</td>
<td>3.85</td>
</tr>
<tr>
<td>liqueur1</td>
<td>Leave unbound!</td>
</tr>
</tbody>
</table>

Ignore entire row in result?
Assignments, Creating new values...

PREFIX ex: <http://example.org/>
SELECT ?Item (?Pr * 1.1 AS ?Pr )
WHERE { ?Item ex:price ?Pr }

Semantics:
extend(μ, var, expr) = μ if var not in dom(μ) and eval(expr) is an error
extend(μ, var, expr) = μ U { var → value | var not in dom(μ) and value = eval(expr) is defined}
extend(μ, var, expr) undefined if var in dom(μ)

For sets of solutions:
extend(M, var, term) = { extend(μ, var, term) | μ in M }
Aggregates
Aggregates

“Count items”

PREFIX ex: <http://example.org/>
SELECT (Count(?Item) AS ?C) WHERE { ?Item ex:price ?Pr }

Data:

@prefix ex: <http://example.org/> .
ex:lemonade1 ex:price 3 ;
   rdf:type ex:Softdrink.
ex:beer1 ex:price 3 ;
   rdf:type ex:Beer.
ex:wine1 ex:price 3.50 ;
   rdf:type ex:Wine.
ex:wine2 ex:price 4 .
   rdf:type ex:Wine.
ex:wine3 ex:price "n/a";
   rdf:type ex:Wine.

Results:

?C 5
Aggregates

■ “Count categories”

PREFIX ex: <http://example.org/>
SELECT (Count(DISTINCT ?T) AS ?C)
WHERE { ?Item rdf:type ?T }

Data:

@prefix ex: <http://example.org/> .
ex:lemonade1 ex:price 3 ;
    rdf:type ex:Softdrink.
ex:beer1 ex:price 3 ;
    rdf:type ex:Beer.
ex:wine1 ex:price 3.5 ;
    rdf:type ex:Wine.
ex:wine2 ex:price 4 .
    rdf:type ex:Wine.
ex:wine3 ex:price "n/a" ;
    rdf:type ex:Wine.

Results:

?C
3
Aggregates - Grouping

“Count items per categories”

PREFIX ex: <http://example.org/>
SELECT ?T (Count(?Item) AS ?C)
WHERE { ?Item rdf:type ?T }
GROUP BY ?T

Data:

@prefix ex: <http://example.org/> .
ex:lemonade1 ex:price 3 ;
ex:beer1 ex:price 3 ;
ex:wine1 ex:price 3.50 ;
ex:wine2 ex:price 4 .
ex:wine3 ex:price "n/a" ;

Results:

<table>
<thead>
<tr>
<th>?T</th>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softdrink</td>
<td>1</td>
</tr>
<tr>
<td>Beer</td>
<td>1</td>
</tr>
<tr>
<td>Wine</td>
<td>3</td>
</tr>
</tbody>
</table>

Enabling networked knowledge.
“Count items per categories, for those categories having more than one item”

PREFIX ex: <http://example.org/>
SELECT ?T (Count(?Item) AS ?C)
WHERE { ?Item rdf:type ?T }
GROUP BY ?T
HAVING Count(?Item) > 1

Data:

@prefix ex: <http://example.org/> .

<table>
<thead>
<tr>
<th>?T</th>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wine</td>
<td>3</td>
</tr>
</tbody>
</table>

ex:lemonade1 ex:price 3 ;
ex:beer1    ex:price 3 ;
ex:wine1    ex:price 3.50 ;
ex:wine2    ex:price 4 .
ex:wine3    ex:price "n/a" ;
          rdf:type ex:Softdrink.
          rdf:type ex:Beer.
          rdf:type ex:Wine.
          rdf:type ex:Wine.

Other Aggregates

- SUM  ... as usual
- AVG  ... as usual
- MIN  ... as usual
- MAX  ... as usual
- SAMPLE  ... “pick” one non-deterministically
- GROUP_CONCAT  ... concatenate values with a designated separator string

...this list is extensible ... new built-ins will need to define error-behaviour, extra-parameters (like SEPARATOR in GROUP_CONCAT)
Example SUM

“Sum Prices per categories”

PREFIX ex: <http://example.org/>
SELECT ?T (Sum(IF(isNumeric(?Pr),?Pr,0) AS ?P)
WHERE { ?Item rdf:type ?T; ex:price ?Pr }
GROUP BY ?T

Data:

@prefix ex: <http://example.org/> .
ex:lemonade1 ex:price 3 ;
rdf:type ex:Softdrink.
ex:beer1 ex:price 3;
rdf:type ex:Beer.
ex:wine1 ex:price 3.50 ;
rdf:type ex:Wine.
ex:wine2 ex:price 4 .
rdf:type ex:Wine.
ex:wine3 ex:price "n/a";
rdf:type ex:Wine.

Results:

<table>
<thead>
<tr>
<th>?T</th>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Softdrink</td>
<td>3</td>
</tr>
<tr>
<td>Beer</td>
<td>3</td>
</tr>
<tr>
<td>Wine</td>
<td>7.5</td>
</tr>
</tbody>
</table>
Example GROUP_CONCAT, SAMPLE

“pick one sample name per person, plus a concatenated list of nicknames”

```sql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>;
SELECT ( SAMPLE(?N) as ?Name) 
(GROUP_CONCAT(?M; SEPARATOR = ",\" ") AS ?Nicknames ) 
WHERE { ?P a foaf:Person ; 
foaf:name ?N ;
foaf:nick ?M . } 
GROUP BY ?P
```

@prefix ex: <http://example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
ex:alice a foaf:Person; foaf:name "Alice Wonderland"; 
foaf:nick "Alice", "The real Alice".
ex:bob a foaf:Person;
 foaf:name "Robert Doe", "Robert Charles Doe", "Robert C. Doe";
foaf:nick "Bob", "Bobby", "RobC", "BobDoe".
ex:charles a foaf:Person;
 foaf:name "Charles Charles";
foaf:nick "Charlie" .

<table>
<thead>
<tr>
<th>Name</th>
<th>Nicknames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice Wonderland</td>
<td>The real Alice, Alice</td>
</tr>
<tr>
<td>Charles Charles</td>
<td>Charlie</td>
</tr>
<tr>
<td>Robert C. Doe</td>
<td>Bob, BobDoe, RobC, Bobby</td>
</tr>
</tbody>
</table>
Evaluate a list of (GROUP BY) expressions:

ListEval(ExprList, μ) returns a list E, where E[i] = μ(ExprList[i])

Use these to partition a solution sequence:

\[
\begin{align*}
\text{Group}(\emptyset, \Omega) &= \{ 1 \to \Omega \} \\
\text{Group}(\text{ExprList}, \Omega) &= \{ \text{ListEval}(\text{ExprList}, \mu) \to \\
&\quad \{ \mu' \mid \mu' \in \Omega, \text{ListEval}(\text{ExprList}, \mu) = \text{ListEval}(\text{ExprList}, \mu') \} \mid \mu \in \Omega \}
\end{align*}
\]

produces a grouped solution sequence

\[
\text{GROUP BY } ?x
\]

Assume solution sequence \( S = (\{?x \to 2, ?y \to 3\}, \{?x \to 2, ?y \to 5\}, \{?x \to 6, ?y \to 7\}) \),

\[
\text{Group}((?x), S) = \{ (2) \to (\{?x \to 2, ?y \to 3\}, \{?x \to 2, ?y \to 5\}), \\
(6) \to (\{?x \to 6, ?y \to 7\}) \}
\]
Aggregates - Semantics

**Definition: Aggregation (simplified)**

Aggregation applies set function “func” (e.g. sum, min, max, …) to a **multiset of lists of expressions** and a **grouped solution sequence**, $G$ as produced by the Group function. It produces a single value for each key and partition for that key (key, X).

\[
\text{Aggregation}(\text{ExprList}, \text{func}, G) = \{ \text{dom}(g) \to F \mid g \in G \}
\]

where

\[
M = \text{ListEvalE}(\text{ExprList}, \text{range}(g))
\]

\[
F = \text{func}(M), \text{ for non-DISTINCT}
\]

\[
F = \text{func}(\text{Distinct}(M)), \text{ for DISTINCT}
\]

$G = \{ (2) \to (\{x\to2, y\to3\}, \{x\to2, y\to3\}),
(6) \to (\{x\to6, y\to7\}) \}$

Aggregation( ?y, Sum, G ) = \{ (2) \to \text{Sum( (3), (3) )}, (6) \to \text{Sum( (7) )} \}

= \{ (2) \to 6, (6) \to 7 \}
Aggregates - Semantics

Definition: Aggregation (simplified)

Aggregation applies set function “func” (e.g. sum, min, max, …) to a multiset of lists of expressions and a grouped solution sequence, G as produced by the Group function. It produces a single value for each key and partition for that key (key, X).

Aggregation(ExprList, func, G) = { dom(g) → F | g in G }
where
M = ListEvalE(ExprList, range(g))
F = func(M), for non-DISTINCT
F = func(Distinct(M)), for DISTINCT

G =
{ (2) → ( {?x→2, ?y→3}, {?x→2, ?y→5} ),
   (6) → ( {?x→6, ?y→7} ) } 

Aggregation( ?y, Sum, G ) = { (2) → Sum( (3), (5) ), (6) → Sum( (7) ) }
   = { (2) → 8, (6) → 7 }

Aggregations subsequently mapped back via Extend(...) to solution multisets

SELECT Sum(?y) AS ?Sy
WHERE { :s :p ?x; :q ?y } { { ?x → 2 , ?Sy → 8 }, {?x→6, ?Sy→7} }
GROUP BY ?x

Omitted details on error handling and scalar Parameters like “SEPERATOR” in GROUP_CPONCAT
Subqueries
Subqueries to realise complex mappings

- How to concatenate first name and last name?
- Now possible without problems per subqueries!

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>

CONSTRUCT { ?P foaf:name ?FullName }
WHERE {
  WHERE { ?P foaf:firstName ?F ; foaf:lastName ?L. }
}

59
Subqueries “Limit per resource”

- Give me all titles of papers of 10 persons who co-authored with Tim Berners-Lee

```sql
SELECT ?T
WHERE {

  { SELECT DISTINCT ?P
       FILTER ( ?P != <http://dblp.l3s.de/.../authors/Tim_Berners-Lee> )
    }
  LIMIT 10
}
}
```

- Returns titles for 10 persons, instead of just 10 rows
Subqueries - Semantics

- **Note:** Before Solution Modifiers are applied, SPARQL semantics converts solution multisets to solution sequences.

```
SELECT ?T
WHERE {
}
```

```
SELECT DISTINCT ?P
WHERE {
  FILTER ( ?P != <http://dblp.../Tim_Berners-Lee> )
}
ORDER BY ?P
LIMIT 10
```

Subqueries require one additional algebra operator, **toMultiSet**, which takes Sequences and returns Multisets.
MINUS and NOT EXISTS
MINUS and NOT EXISTS

- Negation as failure in SPARQL1.0 is “ugly”:

  SELECT ?X
  WHERE { ?X rdf:type foaf:Person
    MINUS { ?X foaf:homepage ?H } } }

- SPARQL1.1 has two alternatives to do the same
  - NOT EXISTS in FILTERs
    - detect non-existence
  - (P1 MINUS P2) as a new binary operator
    - Remove rows with matching bindings
    - only effective when P1 and P2 share variables

- Semantics
May have different results, e.g.:

```
PREFIX ex: <http://example.org/>

SELECT *
WHERE{ ?S ?P ?O
    MINUS { ex:a ex:b ex:c } }
```

@prefix ex: <http://example.org/> .

ex:a ex:b ex:c
Property Path expressions

- Concatenate property paths, Arbitrary Length paths, etc.
- E.g. names of people Tim Berners-Lee transitively co-authored papers with...

```
SELECT DISTINCT ?N
WHERE {<http://dblp.../Tim_Berners-Lee>,
       (^foaf:maker/foaf:maker)+/foaf:name ?N
}
```
# Path expressions full list of operators

## **elt** … Path Element

<table>
<thead>
<tr>
<th>Syntax Form</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>uri</code></td>
<td>A URI or a prefixed name. A path of length one.</td>
</tr>
<tr>
<td><code>~elt</code></td>
<td>Inverse path (object to subject).</td>
</tr>
<tr>
<td>`/uri OR !((uri₁</td>
<td>...</td>
</tr>
<tr>
<td>`/uri and ((uri₁</td>
<td>...</td>
</tr>
<tr>
<td><code>(elt)</code></td>
<td>A group path <code>elt</code>, brackets control precedence.</td>
</tr>
<tr>
<td><code>elt₁ / elt₂</code></td>
<td>A sequence path of <code>elt₁</code>, followed by <code>elt₂</code></td>
</tr>
<tr>
<td>`elt₁</td>
<td>elt₂`</td>
</tr>
<tr>
<td><code>elt*</code></td>
<td>A path of zero or more occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt+</code></td>
<td>A path of one or more occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt?</code></td>
<td>A path of zero or one <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt{n,m}</code></td>
<td>A path between n and m occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt{n}</code></td>
<td>Exactly n occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt{n,}</code></td>
<td>n or more occurrences of <code>elt</code>.</td>
</tr>
<tr>
<td><code>elt{,n}</code></td>
<td>Between 0 and n occurrences of <code>elt</code>.</td>
</tr>
</tbody>
</table>

## Semantics: by translation to native SPARQL with two core property paths

Operators:
- ArbitraryPath(X, path, Y)
- ZeroLengthPath(X, path, Y)
Path expressions

- Can be used for some ontological inference (well known since [Perez et al. 2008])
- E.g. Find all Beers in the Beer ontology

```
PREFIX beer: <http://www.purl.org/net/ontology/beer#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?beer
FROM <http://www.purl.org/net/ontology/beer>
WHERE {
}
```
Implementations of SPARQL 1.1 Query:

Some current (partial) SPARQL1.1 implementations:

- ARQ
  - http://sourceforge.net/projects/jena/
  - http://sparql.org/sparql.html
- OpenAnzo
  - http://www.openanzo.org/
- Perl RDF
  - http://github.com/kasei/perlrdf/
- Corese
- etc.

Others probably forthcoming...

- Loads of SPARQL1.0 endpoints around
  - Dbpedia: http://dbpedia.org/snorql/
  - DBLP: http://dblp.l3s.de/d2r/snorql/
  - Etc.
SPARQL 1.1 querying over OWL2 ontologies and RIF rulesets?
SPARQL1.1 Entailment Regimes

- SPARQL1.1 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/)
  - RDF Entailment Regime
  - RDFS Entailment Regime
  - D-Entailment Regime
  - OWL 2 RDF-Based Semantics Entailment Regime
  - OWL 2 Direct Semantics Entailment Regime
  - RIF Core Entailment

  - Won’t go into details of those, but sketch the main ideas!
General Idea: Answer Queries with implicit answers

E.g. example from before:

```
PREFIX beer: <http://www.purl.org/net/ontology/beer#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?beer
FROM <http://www.purl.org/net/ontology/beer>
WHERE {
}
```
General Idea: Answer Queries with implicit answers

E.g. Graph/Ontology:

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

:jeff a Person .
```

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

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

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

- SPARQL 1.1 other features: e.g. Aggregates, etc.
  - *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:
:rr2010Proceedings :hasEditors [ a rdf:Seq;
    rdf:_1 :pascal_hitzler.
    rdf:_2 :thomas_lukasiewicz.
]

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 .
...
Possibly Infinite answers RDF(S): Container Membership

Graph:
:rr2010Proceedings :hasEditors [ a rdf:Seq;
    rdf:_1 :pascal_hitzler.
    rdf:_2 :thomas_lukasiewicz.
]

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 in SPARQL1.1 are:
{ ?CM/rdfs:_1, ?CM/rdfs:_2, }
More on SPARQL 1.1 + RDFS

SPARQL 1.1 restricts answers to `rdf:/rdfs:/owl:vocabulary` minus `rdf:_1` ...

`rdf:_n` plus terms occurring in the data graph

- **ATTENTION:** this also means no “surrogate blank nodes”!
- **Graph**

```
:alice :name "Alice" .
```

- **Note:** the informative RDFS inference rules at
  - [http://www.w3.org/TR/rdf-mt/#rules](http://www.w3.org/TR/rdf-mt/#rules)
  contains the following rules:

```
```

  where `_:L` is a blank node allocated to the literal bound to `?L`

- **BUT:** the following query

```
SELECT ?L WHERE { ?L rdf:type rdfs:Literal }
```

has no answers by above restriction!
Possibly Infinite answers OWL: datatype reasoning

Stupid way to say Peter is 50:

```sparql
ex:Peter a [ a owl:Restriction ;
  owl:onProperty ex:age ;
  owl:allValuesFrom [ rdf:type rdfs:Datatype .
```

Stupid query asking What is NOT Peters age:

```sparql
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**

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

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

```sparql
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...
SPARQL1.1 Entailment & complex graph patterns

- Once again: SPARQL entailment defined only at the level of BGP matching
  ➔ SPARQL1.1 Algebra is layered “on top”, no interaction

```sparql

SELECT ?X { {?X rdf:type :male } UNION {?X rdf:type :female } }

➔ No result!
```
Similar as before... aggregates are evaluated within algebra 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 .
foaf:knows rdfs:range foaf:Person.
```

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

Under RDFS/OWL entailment returns: `{?X/jeff, ?X/aidan}`
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 .
foaf:knows rdfs:range foaf:Person .
:jeff :hasFather [a Person].
:jeff owl:sameAs :aidan .

SELECT (Count(?X) AS ?Y) { ?X a foaf:Person }
```

Under RDFS/OWL entailment returns : {?Y/3}
SPARQL1.1 + RIF

- **RIF ... Rule Interchange format, Rec. since 2010**
  - **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)
  - Combinations of RIF with RDF, RDFS, and OWL defined in: http://www.w3.org/TR/rif-rdf-owl/
RIF Dialects

<table>
<thead>
<tr>
<th>Core</th>
<th>BLD</th>
<th>PRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>horn rules,</td>
<td>equality, class membership</td>
<td>non-monotonic</td>
</tr>
<tr>
<td>monotonic</td>
<td>in conclusions</td>
<td>actions in conclusions</td>
</tr>
<tr>
<td>datatypes &amp;</td>
<td>frame subclasses</td>
<td>negation</td>
</tr>
<tr>
<td>built-ins</td>
<td>open lists</td>
<td>subclasses</td>
</tr>
<tr>
<td>external</td>
<td></td>
<td>membership in conclusion</td>
</tr>
<tr>
<td>functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frames, class</td>
<td></td>
<td></td>
</tr>
<tr>
<td>memberships</td>
<td></td>
<td></td>
</tr>
<tr>
<td>equality (in</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conditions)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ground lists</td>
<td></td>
<td></td>
</tr>
<tr>
<td>existential</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quantification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(in conditions)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SPARQL1.1 so far only defines Entailment for RIF Core... room for improvement (cf. e.g. Demo Obermeier et al. RR20110)
RIF Core allows to encode RDFS, e.g.:

\[\text{rdfs1:} \{ ?S \text{ rdf:type } ?C \} :- \{ ?S \ ?P \ ?O . \ ?P \text{ rdfs:domain } ?C . \} \]

RIF Core allows to encode OWL2 RL, e.g.:

\[\text{owl1:} \{ ?S1 \text{ owl:SameAs } ?S2 \} :- \]
\[\quad \{ ?S1 \ ?P \ ?O . \ ?S2 \ ?P \ ?O . \ ?P \text{ rdf:type owl:InverseFunctionalProperty} \} \]
\[\text{owl2:} \{ ?X \ ?P \ ?O \} :- \{ ?X \text{ owl:SameAs } ?Y . \ ?X \ ?P \ ?O \} \]
\[\text{owl3:} \{ ?S \ ?Y \ ?O \} :- \{ ?X \text{ owl:SameAs } ?Y . \ ?S \ ?X \ ?O \} \]
\[\text{owl4:} \{ ?S \ ?P \ ?Y \} :- \{ ?X \text{ owl:SameAs } ?Y . \ ?S \ ?P \ ?X \} \]

Plus more (custom rules, including Built-ins):

\[\{ ?X \text{ foaf:name } ?\text{FullN} \} :- \{ ?X \text{ foaf:firstName } ?F . \ ?X \text{ foaf:lastName } ?L \} \]
\[\text{AND } ?\text{FullN} = \text{fn:concat(?F, " ", ?L)} \]

<http://ruleset1.rif>
How to reference to a RIF Rule set from SPARQL?

- In OWL Entailment Regime, OWL is assumed to be part of the RDF Graph (OWL/RDF)

- RIF’s so far only a normative syntax is RIF/XML
  - RIF encoding in RDF (RIF/RDF) underway:
    http://www.w3.org/2005/rules/wiki/RIF_In_RDF
  - Will also provide a new RDF property rif:usedWithProfile to import RIF rulesets (in RIF/XML or RIF/RDF). e.g.

```xml
<http://ruleset1.rif> rif:usedWithProfile
  <http://www.w3.org/ns/entailment/Simple> .
<http://dblp.l3s.org/Tim_Berners-Lee>
  foaf:homepage <http://www.w3.org/People/Berners-Lee/> ;
  foaf:name "Tim Berners-Lee" .
<http://www.w3.org/People/Berners-Lee/card#i>
  foaf:homepage <http://www.w3.org/People/Berners-Lee/> ;
  foaf:firstName "Timothy";
  foaf:lastName "Berners-Lee" .
```

- In current draft called rif:imports

```sparql
```

<table>
<thead>
<tr>
<th>?P</th>
<th>?N</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;dblp/Tim&gt;</td>
<td>Tim Berners-Lee</td>
</tr>
<tr>
<td>&lt;w3/B-Lee/card#i&gt;</td>
<td>Tim Berners-Lee</td>
</tr>
<tr>
<td>&lt;dblp/Tim&gt;</td>
<td>Timothy Berners-Lee</td>
</tr>
<tr>
<td>&lt;w3/B-Lee/card#i&gt;</td>
<td>Timothy Berners-Lee</td>
</tr>
</tbody>
</table>
Semantics:
- Relatively Straightforward: **BGPs matching** defined as being RIF-RDF-entailed by RDF data graph in combination with the referenced ruleset.

Infinite answers possible
- (even though RIF Core has no function symbols):

```
:a :b 1 .
```

SELECT ?O { :a :b ?O .}

So far (as opposed to SPARQL/OWL SPARQL/RDFS) **no restrictions on finiteness in SPARQL1.1/RIF**
- Finite answers up to the user, or
- Restrict to strongly safe RIF Core (inspired by [Eiter et al. 2006]) or
- System streams out answers (e.g. a la Prolog)
Wrapping up

- **SPARQL 1.0**
  - UNIONS of Conjunctive Queries, FILTERs, GRAPH queries, OPTIONAL, (hidden) negation
  - contributed largely to the current Linked Data boom
  - Inspired interesting academic work

- **SPARQL 1.1**
  - A reasonable next step
    - Incorporating highly demanded features
    - Closing the gaps to neighbour standards (OWL2, RIF)
  - Not all of it is trivial → SPARQL1.1 takes a very pragmatic path

- **Hopefully inspiring for more research, more data, and more applications!**
What I didn’t talk about…

List of agreed features:

- **Additions to the Query Language:**
  - Project Expressions
  - Aggregate functions
  - Subqueries
  - Negation
  - Property Paths (*time permitting*)
  - Extend the function library (*time permitting*)
  - Basic federated Queries (*time permitting*)

- **Entailment** (*time permitting*)

- **SPARQL Update**
  - Full Update language
  - plus simple RESTful update methods for RDF graphs (HTTP methods)

- **Service Description**
  - Method for discovering a SPARQL endpoint’s capabilities
  - Summary of its data
Extended Function Library

- Functions Library in SPARQL1.0 is insufficient:
  - Bound( . )
  - isLiteral( . )
  - isBlank( . )
  - isIRI( . )
  - Str( . )
  - Regex( . , . )
  - +,-,*, <, >, =

- New functions to be included in standard library:
  - COALESCE, IF
  - Functions from the Xpath/Xquery function library
    - String manipulation, more math, etc. ... e.g. fn:concat

**Essentially: rubber-stamp common functions present in current implementations**
Basic federated Queries *(time permitting)*

- [http://www.w3.org/TR/sparql11-federated-query/](http://www.w3.org/TR/sparql11-federated-query/)
  - Will be integrated in Query spec
- **Essentially new pattern** `SERVICE`
  - Similar to `GRAPH`
  - allows delegate query parts to a specific (remote) endpoint

Recall: *We were cheating in this query before!!*

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
WHERE {
  { <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F .
    ?F foaf:name ?N }
  UNION
  { [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
  }
}
```
Basic federated Queries (*time permitting*)

- [http://www.w3.org/TR/sparql11-federated-query/](http://www.w3.org/TR/sparql11-federated-query/)
  - Will be integrated in Query spec
- **Essentially new pattern** SERVICE
  - Similar to GRAPH
  - allows delegate query parts to a specific (remote) endpoint

```sql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?N
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  { <http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?F .
    ?F foaf:name ?N } 
  UNION 
  { SERVICE <http://dblp.l3s.de/d2r/sparql> 
    { [ foaf:maker <http://dblp.l3s.de/.../authors/Tim_Berners-Lee>,
     [ foaf:name ?N ] ] . } } 
}
```
Like SQL ... SPARQL/RDF Stores need a standard Data Manipulation Language
http://www.w3.org/TR/sparql11-update/

SPARQL 1.1 Update Language

- Graph Update
  - INSERT DATA
  - DELETE DATA
  - DELETE/INSERT
  - DELETE
  - INSERT
  - DELETE WHERE
  - LOAD
  - CLEAR

- Graph Management
  - CREATE
  - DROP

Issue: Graph-aware stores vs. Quad Stores
Service Description

Base vocabulary to describe
• features of SPARQL endpoints
• datasets (via vocabularies external to the Spec, e.g. VOID)

- http://www.w3.org/TR/sparql11-service-description/

3.2 Classes
  3.2.1 sd:Service
  3.2.2 sd:Language
  3.2.3 sd:Function
  3.2.4 sd:Aggregate
  3.2.5 sd:EntailmentRegime
  3.2.6 sd:EntailmentProfile
  3.2.7 sd:GraphCollection
  3.2.8 sd:Dataset
  3.2.9 sd:Graph
  3.2.10 sd:NamedGraph

3.3 Instances
  3.3.1 sd:SPARQL10Query
  3.3.2 sd:SPARQL11Query
  3.3.3 sd:SPARQL11Update
  3.3.4 sd:DereferencesURIs
  3.3.5 sd:UnionDefaultGraph
  3.3.6 sd:RequiresDataset
  3.3.7 sd:EmptyGraphs

3.4 Properties
  3.4.1 sd:url
  3.4.2 sd:feature
  3.4.3 sd:defaultEntailmentRegime
  3.4.4 sd:supportedEntailmentProfile
  3.4.5 sd:entailmentRegime
  3.4.6 sd:extensionFunction
  3.4.7 sd:extensionAggregate
  3.4.8 sd:languageExtension
  3.4.9 sd:supportedLanguage
  3.4.10 sd:propertyFeature
  3.4.11 sd:defaultDatasetDescription
  3.4.12 sd:availableGraphDescriptions
  3.4.13 sd:resultFormat
  3.4.14 sd:defaultGraph
  3.4.15 sd:namedGraph
  3.4.16 sd:name
  3.4.17 sd:graph
Chair-hat on: Ppleaseease
(1) Read the specs!
(2) Send us comments public-rdf-dawg-comments@w3.org
References


Relevant W3C Specs

- SPARQL Query Language for RDF [http://www.w3.org/TR/rdf-sparql-query/]
- SPARQL1.1 Query Language for RDF (working draft) [http://www.w3.org/TR/sparql11-query/]
- SPARQL1.1 Entailment Regimes (working draft) [http://www.w3.org/TR/sparql11-entailment/]

RDF(S) Entailment/D-Entailment:
- RDF Semantics [http://www.w3.org/TR/rdf-mt/]

OWL Entailment:
- OWL2 Web Ontology Language Primer [http://www.w3.org/TR/owl2-primer/]
- OWL2 Web Ontology Language Profiles [http://www.w3.org/TR/owl2-profiles/]

RIF Entailment:
- RIF Core Dialect [http://www.w3.org/TR/rif-core/]
- RIF Basic Logic Dialect [http://www.w3.org/TR/rif-bld/]
- RIF RDF and OWL compatibility [http://www.w3.org/TR/rif-rdf-owl/]
Acknowledgements

- The members of the W3C SPARQL WG, particularly Lee Feigenbaum, who I stole some examples from
- The members of the W3C RIF WG

- The RR2010 chairs for inviting me 😊
GiaBATA

Implementing SPARQL, OWL2RL, RIF on top of DLV
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, 2007]
- Feed resulting program into a rules engine (DLV-DB) that runs over a Rel DB storing RDF graphs.

Check Details at [Ianni et al. 2009]:
How to implement this?

- **GiaBATA system [Ianni 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 2007]

```
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,0,default) :- &rdf[ "alice.org" ](S,P,0).
(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:


%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/1999/02/22-rdf-syntax-ns#type">",_),
resource_literal(Property,"<http://www.w3.org/1999/02/22-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/1999/02/22-rdf-syntax-ns#type">",G),
resource_literal(Property,"<http://www.w3.org/1999/02/22-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/1999/02/22-rdf-syntax-ns#type">",G),
resource_literal(Property,"<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property">",G),
resource_literal(SubPropertyOf,"<http://www.w3.org/2000/01/rdf-schema#subPropertyOf">",G).
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 based on statistics/selectivity e.g. a la [Vidal et al. 2009], not yet implemented, but we did some manual reordering to optimize the query plan in the experiments.
Extending GiaBATA to SPARQL1.1 – Future Work

- OWL2RL and RIF Entailment, done, but doesn’t yet consume RIF directly:
  - Integration with RIF-plugin [Obermeier et al. RR2010] planned

SPARQL1.1 features:
- Subqueries, doable (however: modulo solution modifiers)
- NOT EXISTS, MINUS, doable
- Property Path Expressions, doable
- Aggregates, probably doable, cf. [Polleres et al. 2007], [Faber et al. 2004]
Additional References


Additional Acknowledgements:

Giovambattista Ianni, Alessandra Martello, Thomas Krennwallner, Philipp Obermeier