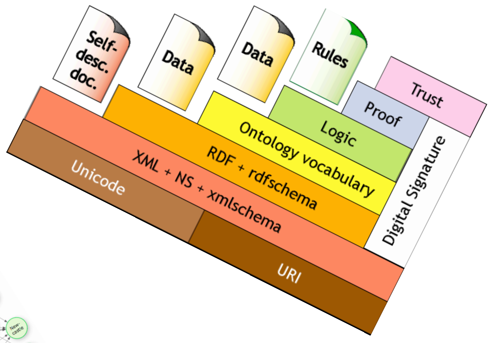
How (well) do Datalog, SPARQL and RIF interplay?

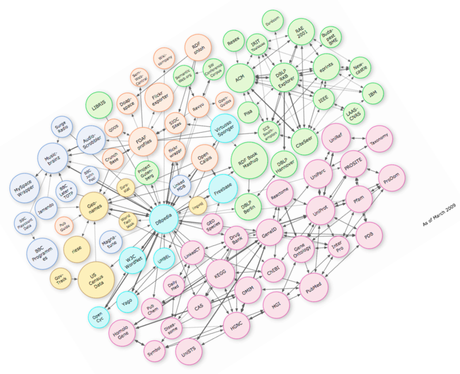
Axel Polleres, Siemens AG Österreich

Introduction / Contents

What have You heard about
"Semantic Web Standards"?

DESCRIPTION
LOGICS





- Many of you have probably heard about mostly **OWL** and **Description Logics**… not today.
- … in fact two other W3C standards are probably much closer to Datalog:
- **SPARQL – RDF Query language**
- **RIF – Rule Interchange Format**
- In this Tutorial:
  - How close are they to Datalog, where do they differ?

Outline

Semantic Web Standards?

- **RDF** and Datalog
- **SPARQL** and Datalog
- **RIF** and Datalog
- **SPARQL1.1** and Datalog - An Outlook

SIEMENS

RDF – The Resource Description Framework [W3C,2004]

```
dbpedia:Vienna dbpedia-ont:country    dbpedia:Austria .
dbpedia:Vienna rdfs:label             ”Wien”@de .

_:x            foaf:name              ”Reinhard Pichler" .
_:x            foaf:based_near        dbpedia:Vienna .
```

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

URIs, e.g.
`http://http://xmlns.com/foaf/0.1/name`
`http://dbpedia.org/resource/Vienna`
`http://dbpedia.org/resource/Austria`

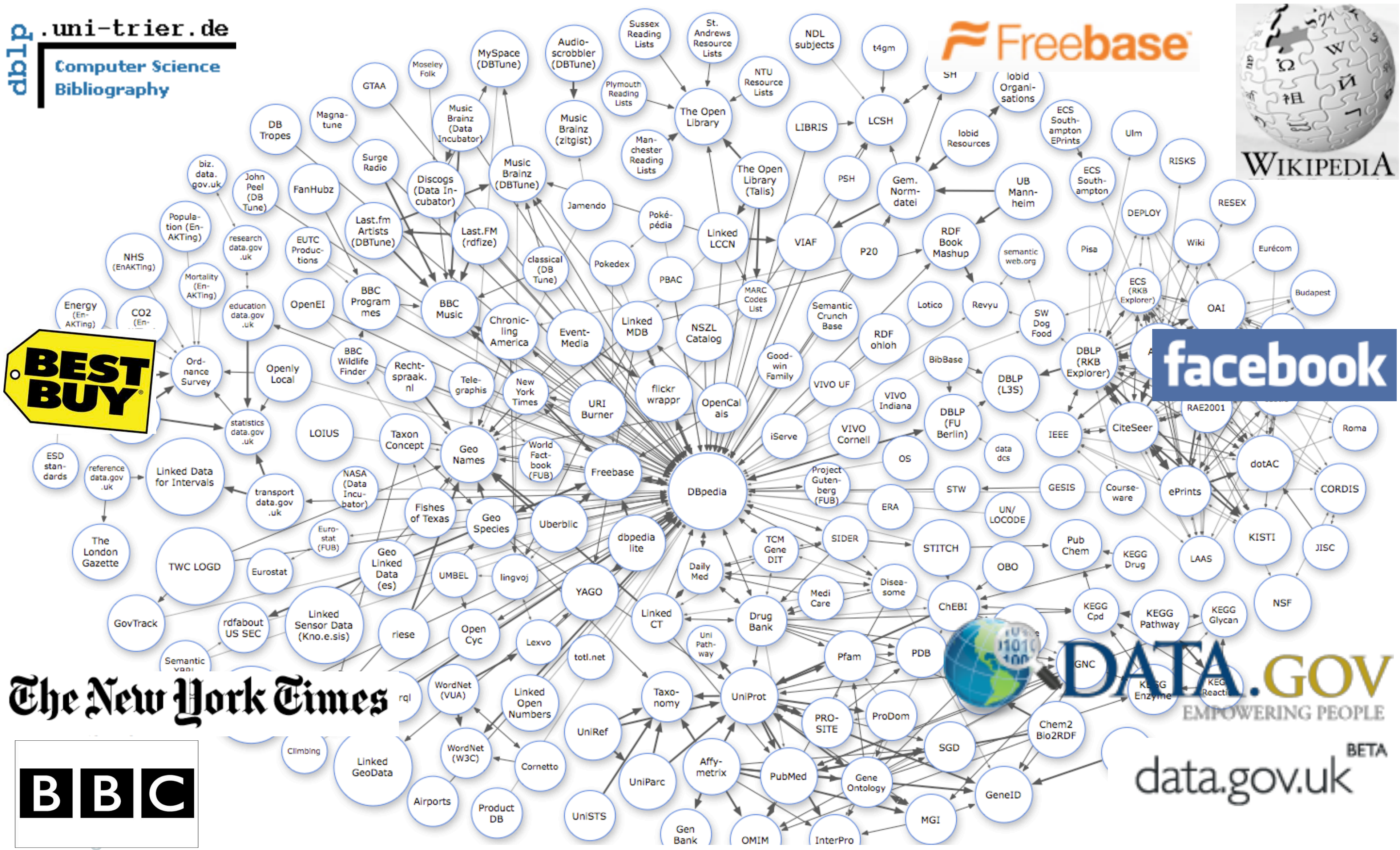
Subject U ∪ B
x
Predicate U
x
Object U ∪ B ∪ L

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

Literals, e.g.
`“2012”^^xsd:gYear`
`“Wien”@de`
`“Vienna”@en`
`“Reinhard Pichler”`

RDF – Adoption

SIEMENS



Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/

SIEMENS

RDF in Datalog? (Almost) No problem

```
dbpediares:Vienna  dbpedia-ont:country  dbpediares:Austria .
dbpediares:Vienna  rdfs:label  "Wien"@de .
_:x  foaf:name   "Reinhard Pichler" .
_:x  foaf:based_near   dbpediares:Vienna .
```

$$\exists X$$

$$triple(vienna, country, austria) \wedge$$

$$triple(vienna, label, "Wien"@de) \wedge$$

$$triple(x, name, "ReinhardPichler) \wedge$$

$$triple(x, based\_near, vienna)$$

What about Blank nodes? …

… let's just use local constants ("Skolemize")

RDF in Datalog? (Almost) No problem

```
dbpediares:Vienna  dbpedia-ont:country  dbpediares:Austria .
dbpediares:Vienna  rdfs:label  "Wien"@de .
_:x  foaf:name   "Reinhard Pichler" .
_:x  foaf:based_near   dbpediares:Vienna .
```

EDB:

triple(vienna, country, austria).
triple(vienna, label, “Wien”@de).
triple(**b1**, name, “Reinhard Pichler”).
triple(**b1**, based_near, vienna).

What about Blank nodes? …

… let’s just use local constants (“Skolemize”)

SIEMENS

RDF Schema 1/2

```
dbpediares:Vienna dbpedia-ont:country dbpediares:Austria .
dbpediares:Vienna rdfs:label "Wien"@de .
_:x foaf:name "Reinhard Pichler" .
_:x foaf:based_near dbpediares:Vienna .

dbpediares:Austria  rdf:type  dbpedia-owl:Country .
_:x rdfs:label "Reinhard Pichler" .
```

```
foaf:name rdfs:subPropertyOf rdfs:label .
dbpedia-ont:country rdfs:range dbpedia-owl:Country .
```

- formal semantics **[W3C, 2004]**
- can be captured by Datalog style rules **[W3C, 2004 §7]**, e.g. …

rdfs3	aaa rdfs:range xxx . uuu aaa vvv .	vvv rdf:type xxx .
rdfs7	aaa rdfs:subPropertyOf bbb . uuu aaa yyy .	uuu bbb yyy .

- … with some caveats **[ter Horst, 2005]**, **[Muñoz+, 2009]**

SIEMENS

RDF Schema 1/2

```
dbpediares:Vienna dbpedia-ont:country dbpediares:Austria .
dbpediares:Vienna rdfs:label "Wien"@de .
_:x foaf:name "Reinhard Pichler" .
_:x foaf:based_near dbpediares:Vienna .

dbpediares:Austria  rdf:type  dbpedia-owl:Country .
_:x rdfs:label "Reinhard Pichler" .
```

```
foaf:name rdfs:subPropertyOf rdfs:label .
dbpedia-ont:country rdfs:range dbpedia-owl:Country .
```

- formal semantics **[W3C, 2004]**
- can be captured by Datalog style rules **[W3C, 2004 §7]**, e.g. …

```
triple( O, rdf:type, C) :- triple( P, rdf:range, C), triple(S,P,O) .
triple( S, Q, O) :- triple( P, rdfs:subPropertyOf, Q), triple(S,P,O) .
```

- … with some caveats **[ter Horst, 2005]**, **[Muñoz+, 2009]**

SIEMENS

RDF Schema 2/2 – RDF(S) Entailment

Core problem described in RDF Semantics document is RDF(S) Entailment **[W3C, 2004]**

$$G1 \models_{RDFS} G2$$

Is there a blank node homomorphism μ from G2 to G1 such that

$$\mu(G2) \subseteq Cl_{RDFS}(G1)$$

RDFS Entailment checking can be easily done in Datalog **[Bruijn&Heymans,2007], [Muñoz+, 2009] , [Ianni+, 2009],** cf. also **[Gutierrez+,2011].**

G1
```
_:x   foaf:name    "Reinhard" .
foaf:name rdfs:subPropertyOf rdfs:label .
```

$\overset{?}{\models}_{RDFS}$

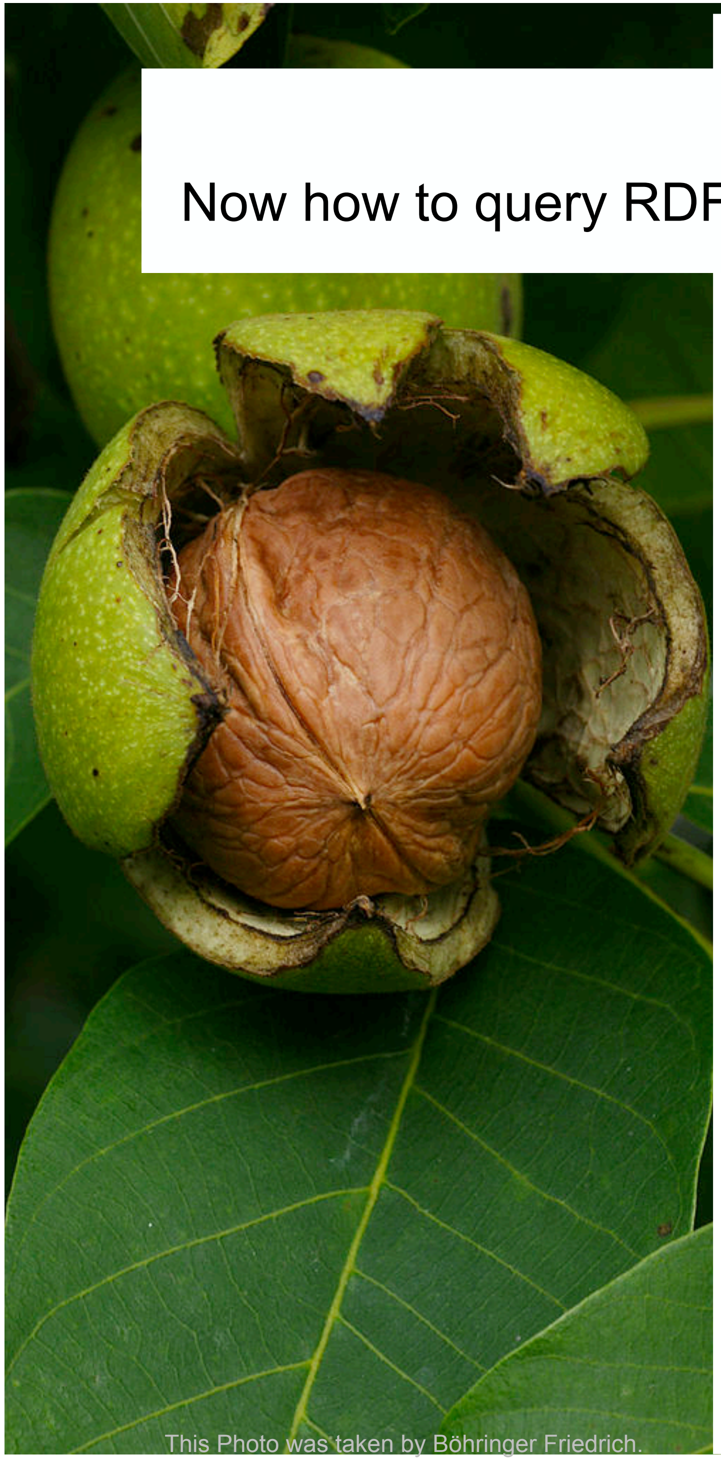
G2
```
_:x   foaf:name    "Reinhard" .
_:y   rdfs:label   "Reinhard" .
```

1) Encode G1 + RDFS Entailment rules in Datalog EDB+IDB
2) Encode G2 as boolean conjunctive query

EDB (G1)
```
triple(x, name, "Reinhard" ) . triple(name rdfs:subPropertyOf, label).
```

IDB (RDFS)
```
triple( S, Q, O) :- triple( P,rdfs:subPropertyOf, Q), triple(S, P, O) .
...
```

Query
```
answer :- triple(X, name,"Reinhard"), triple(Y, label, "Reinhard" )
```



SIEMENS

Now how to query RDF?

SPARQL1.0 [W3C, 2008]
in a Nutshell...

… i.e.,
nonrecursive Datalog^{not}
in a Nutshell...

[Angles, Gutierrez, 2008]

SPARQL + Linked Data give you Semantic search almost "for free"

SIEMENS

Query: Scientists born in Vienna? (Conjunctive Query)
How'd we do it in SQL?

```
SELECT t1.s
FROM triple t1, triple t2
WHERE t1.s = t2.s  AND t1.p = dbpedia:birthPlace AND t1.o = Vienna
                   AND t2.p = rdf:type AND t2.o = dbpedia:Scientist
```

Obiously, we know how to do that in Datalog...

```
answer(X) :-
        triple( X,  birthPlace , Vienna ) ,
        triple( X,  type , Scientist ) .
```

SPARQL + Linked Data give you Semantic search almost “for free”

Query: Scientists born in Vienna? (Conjunctive Query)

*Now how does it look in **SPARQL**?*

```
SELECT ?X
WHERE {
        ?X dbpedia:birthPlace  <dbpedia.org/resource/Vienna> .
        ?X rdf:type dbpedia:Scientist.
      }
```

Obiously, we know how to do that in Datalog...

```
answer(X) :-
        triple( X,  birthPlace , Vienna ) ,
        triple( X,  type , Scientist ) .
```

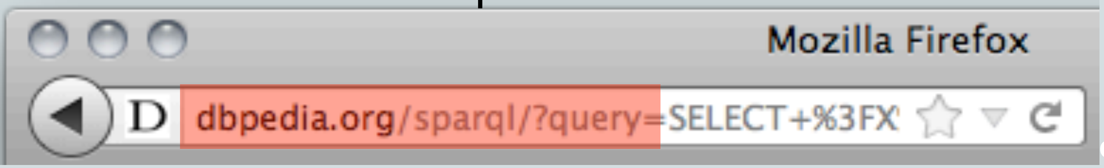
… and SPARQL looks quite similar!

SPARQL – Standard RDF Query Language and Protocol

SPARQL 1.0 (2008):

```
SELECT ?X
WHERE {
        ?X dbpedia:birthPlace  <dbpedia.org/resource/Vienna> .
        ?X rdf:type dbpedia:Scientist.
    }
```

- SQL “Look-and-feel” for the Web
- Essentially “graph matching” by *basic graph patterns (BGPs)*
- Allows conjunction (.) , disjunction (UNION), optional (OPTIONAL) patterns and filters (FILTER)
- Construct new RDF from existing RDF (CONSTRUCT)
- Solution modifiers (DISTINCT, ORDER BY, LIMIT, …)
- A **standardized** HTTP based protocol:



Link

SPARQL Formal Semantics – Basic Graph Patterns [Perez et al. 2006]

SIEMENS

Definition 1:

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

$dom(\mu)$ is exactly the set of variables occurring in P and

$\mu(P) \subseteq G$ (actually, in the official W3C spec it is rather $G \models_{RDF} \mu(P)$)

Example RDF Graph (G):

```
:tim     foaf:knows   :jim .
:jim     foaf:knows   :tim .
:jim     foaf:knows   :juan .
```

Example Pattern (P):

```
SELECT * WHERE { ?X  foaf:knows  ?Y .  ?Y foaf:knows ?Z }.
```

eval(P,G) = { $\mu 1$ = { ?x → :tim , ?y → :jim , ?z → :tim },
$\mu 2$ = { ?x → :tim, ?y → :jim , ?z → :juan } }

SPARQL Algebra as per [Perez et al. 2006]

Definition 2:

mappings $\mu 1, \mu 2$ are compatible iff they agree in their shared variables.

Let M1, M2 be **sets of mappings**

Definition 3:

Join:

$M1 \bowtie M2 = \{ \mu 1 \cup \mu 2 \mid \mu 1 \in M1, \mu 2 \in M2, \text{ and } \mu 1, \mu 2 \text{ are compatible}\}$

Union:

$M1 \cup M2 = \{ \mu \mid \mu \in M1 \text{ or } \mu \in M2\}$

Diff:

$M1 \setminus M2 = \{\mu \in M1 \mid \text{forall } \mu' \in M2, \mu \text{ and } \mu' \text{ are not compatible } \}$

LeftJoin:

$M1 ⟕ M2 = (M1 \bowtie M2) \cup (M1 \setminus M2)$

Filter:

$M|_R = \{ \mu \mid \mu \in M \text{ and } \mu(R) = \text{true}\}$

Semantics full as per [Perez et al.2006]

eval(BGP,G)		… *see* **Definition 1**
eval(P1 . P2,G)	=	$eval(P1, G) \bowtie eval(P2, G)$
eval(P1 `UNION` *P2,G)*	=	$eval(P1, G) \cup eval(P2, G)$
eval(P1 `OPTIONAL` *P2, G)*	=	$eval(P1, G)$ ⟕ $eval(P2, G)$
eval(P `FILTER` *R,G)*	=	$eval(P, G)\|_R$

Example $\bowtie$ **:**

```
P = { ?X  foaf:knows  ?Y . ?Y foaf:knows ?Z }
```

`eval(P1,G)` $\bowtie$ `eval(P2,G)=`

X	Y
tim	jim
jim	tim
jim	juan

$\bowtie$

Y	Z
tim	jim
jim	tim
jim	juan

=

X	Y	Z
tim	jim	tim
tim	jim	juan

SIEMENS

Back to “real” SPARQL examples: UNION

Example RDF Graph:

```
:tim        triple( :tim, knows, :jim ) .
:jim        triple( :jim, knows, :tim ) .
:jim        triple( :jim, worksWith, :juan ) .
```

Example Query:

```
SELECT ?X
  WHERE {
         { :jim foaf:knows ?X }
         UNION
         { :jim foaf:worksWith ?X }
        }
```

X
tim

U

X
juan

=

X
tim
juan

Back to “real” SPARQL examples: UNION

Example RDF Graph in Datalog EDB:

```
triple( :tim, knows, :jim ) .
triple( :jim, knows, :tim ) .
triple( :jim, worksWith, :juan ) .
```

In Datalog:

```
answer(X) :- evalP(X).
evalP(X) :-
          triple( :jim, knows, X ) .
evalP(X) :-
          triple( :jim, worksWith, X ) .
```

X
tim

U

X
juan

=

X
tim
juan

SIEMENS

Back to “real” SPARQL examples: UNION

Example RDF Graph:

```
:tim          foaf:knows     :jim .
:jim          foaf:knows     :tim .
:jim          :worksWith     :juan .
```

Example Query:

```
SELECT ?X ?Y
  WHERE {
        { :jim foaf:knows ?X }
        UNION
        { :jim foaf:worksWith ?Y }
       }
```

X
tim

U

Y
juan

=

X	Y
tim	null
null	juan

Back to “real” SPARQL examples: UNION

Example RDF Graph:

```
triple( :tim, knows, :jim ) .
triple( :jim, knows, :tim ) .
triple( :jim, worksWith, :juan ) .
```

Example Query:

```
answer(X,Y) :- evalP(X,Y).
evalP(X,null) :-
          triple( :jim, knows, X ) .
evalP(null,Y) :-
          triple( :jim, worksWith, Y ) .
```

X
tim

U

Y
juan

=

X	Y
tim	null
null	juan

Back to “real” SPARQL examples: OPTIONAL

Give me people who know somebody and OPTIONALLY their email address:

```
:tim     foaf:knows :jim .   :tim :email <mailto:timbl@w3.org> .
:jim     foaf:knows :tim .
:jim     :worksWith :juan .
```

Example Query:

```
SELECT ?X ?M
  WHERE {
        { ?X foaf:knows ?Y }
        OPTIONAL
        { ?X :email ?M }
       }
```

X	Y
tim	jim
jim	tim

$\bowtie$

X	M
tim	timbl@w3.org

$\cup$

X	Y
tim	jim
jim	tim

$\setminus$

X	M
tim	timbl@w3.org

=

X	Y	M
tim	jim	timbl@w3.org
jim	tim	

$\pi_{X,M}$

X	M
tim	timbl@w3.org
jim	

Back to “real” SPARQL examples: OPTIONAL

SIEMENS

Give me people who know somebody and OPTIONALLY their email address:

```
triple( :tim, knows, :jim ) .    triple(:tim, email, timbl@w3.org ) .
triple( :jim, knows, :tim ) .
triple( :jim, worksWith, :juan ) .
```

Example Query:

```
answer(X,M)  :- evalP(X,Y,M) .

evalP(X,Y,M) :- triple( X, knows, Y ) , triple( X, email, M) .

evalP(X,Y,null) :- triple( X, knows, Y ) , not evalP1(X) .
evalP1(X) :- triple( X, email, M) .
```

X	Y
tim	jim
jim	tim

$\bowtie$

X	M
tim	timbl@w3.org

$\cup$

X	Y
tim	jim
jim	tim

$\setminus$

X	M
tim	timbl@w3.org

$=$

X	Y	M
tim	jim	timbl@w3.org
jim	tim	

$\pi_{X,M}$

X	M
tim	timbl@w3.org
jim	

ATTENTION:

⋈ needs some attention!

Eval(P1,G)

X	Y
a	c
b	null

⋈

Eval(P2,G)

Y	Z
null	e
d	f

```
evalP(X,Y,Z) :- evalP1( X, Y ) , evalP2( Y, Z) .
```

Doesn't work!

Recall (Definition 3):

Join:
M1 M2 = { $\mu 1 \cup \mu 2$ | $\mu 1 \in$ M1, $\mu 2 \in$ M2, and $\mu 1$, $\mu 2$ are **compatible**}

Rather:

```
evalP(X,Y,Z) :- evalP1( X, Y ) , evalP2( Y1, Z), join(Y,Y1) .
join(X,X)    :- HU_G(X).
join(X,null) :- HU_G(X).
join(null(X) :- HU_G(X).
```

… where HU_G(X) is a predicate defining the Herbrand Universe of G.

FILTERs 1/3

Give me people with an email address where the email doesn't contain "w3":

```
:tim    foaf:knows :jim .   :tim :email <mailto:timbl@w3.org> .
:jim    foaf:knows :tim .   :jim :email <mailto:hendler@cs.rpi.edu> .
:jim    :worksWith :juan .
```

Example Query:

```
SELECT ?X ?M
 WHERE { ?X :email ?M .
         FILTER( ! Regex(Str(?M), "w3" )  }
```

Complex FILTER expressions allowed (!, &&, ||)

X	M
jim	hendler@cs.rpi.edu

FILTERs 2/3

*People who know someone & **optionally** their email where the email doesn't contain "w3":*

```
:tim    foaf:knows :jim .  :tim :email <mailto:timbl@w3.org> .
:jim    foaf:knows :tim .  :jim :email <mailto:hendler@cs.rpi.edu> .
:juan   foaf:knows :jim .
```

Example Query:

```
SELECT ?X ?M
 WHERE { ?X foaf:knows ?Y
         OPTIONAL {?X :email ?M . }
         FILTER( ! Regex(Str(?M), "w3" )  }
```

X	M
jim	hendler@cs.rpi.edu

Note: FILTERs are evaluated under a three-values semantics!
(True, False, Error), e.g.

A	!A
T	F
F	T
E	E

FILTERs 3/3

A special FILTER function is bound() – Can be used to “encode” Negation as failure in SPARQL1.0:

*Give me people **without** an email address:*

```
SELECT ?X ?M
 WHERE { ?X foaf:knows ?Y
         OPTIONAL {?X :email ?M . }
         FILTER( ! bound(?M) )  }
```

What about Datalog?

SPARQL FILTERs can in principle be encoded in Datalog,

- need built-ins (or be pre-compiled for HU_G)
- need to encode three-valued semantics for *!, &&, ||*

SPARQL 1.0 = nonrecursive Datalog^{not}

[Polleres, 2007] shows that all of SPARQL 1.0 can be translated to (safe) nonrecursive Datalog^{not}.

In fact, **[Angles&Gutierrez 2008]** vice versa show that (safe) nonrecursive Datalog^{not} likewise be encoded into SPARQL.

PSPACE Program-Complexity for SPARQL 1.0 follows from
[Perez et al. 2006] or alternatively
[Angles&Gutierrez 2008] + [Dantsin et al. 2001].

Some notable peculiarities about SPARQL1.0 ...

Notable about the official SPEC semantics 1/2

SPARQL allows duplicates !

SIEMENS

A slightly modified RDF Graph:

```
triple( :jim, knows, :tim ) .
triple( :jim, worksWith, :tim) .
```

Example Query:

```
answer(X,U) :- evalP(X, U).
evalP(X, u1) :-
          triple( :jim, knows, X ) .
evalP(X, u2) :-
          triple( :jim, worksWith, X ) .
```

!

X	Union1
tim	u1
tim	u2

Notable about the official SPEC semantics 2/2

FILTERS can make OPTIONAL non-compositional!

- *“Conditional OPTIONAL”*
  - *“Give me emails, **and the friends only of those whose email contains ‘W3’**”*

```
SELECT  ?N ?F
WHERE{ ?X :email ?M
            OPTIONAL { ?X foaf:knows ?F
                       FILTER ( regex( str(?M), "w3" ) ) }
     }
```

OPTIONAL with FILTERs is NOT modular/compositional

[Angles&Gutierrez, 2008] showed compositional semantics can be achieved by a rewriting, but non-compositional semantics can be actually be directly encoded in Datalog **[Polleres&Schindlauer, 2007]**

…

SIEMENS

Adapting [Perez et al. 2006] to match the W3C SPARQL1.0 specification

1) Algebra operations need to be adapted to multiset/bag semantics:

2) non-compositionality of FILTERs in OPTIONAL

Let M1, M2 be **multisets** of mappings

Definition 3:

Join:

$M1 \bowtie M2 = \{\{ \mu1 \cup \mu2 \mid \mu1 \in M1, \mu2 \in M2, \text{ and } \mu1, \mu2 \text{ are compatible} \}\}$

Union:

$M1 \cup M2 = \{\{ \mu \mid \mu \in M1 \text{ or } \mu \in M2 \}\}$

Diff:

$M1 \setminus M2 = \{\{ \mu \in M1 \mid \text{forall } \mu' \in M2, \mu \text{ and } \mu' \text{ are not compatible} \}\}$

LeftJoin:

~~$M1 ⟕ M2 = (M1 \bowtie M2) \cup (M1 \setminus M2)$~~

Filter:

$M|_R = \{ \mu \mid \mu \in M \text{ and } \mu(R) = \text{true}\}$

Adapting [Perez et al. 2006] to match the W3C SPARQL1.0 specification

eval(BGP,G) … *see* **Definition 1**

eval(P1 . P2,G) = *eval(P1, G)* $\bowtie$ *eval(P2, G)*

eval(P1 `UNION` *P2,G)* = *eval(P1, G)* ∪ *eval(P2, G)*

eval(P `FILTER` *R,G)* = *eval(P, G)* $|_R$

eval(P1 `OPTIONAL` *{P2* `FILTER` *R}, G) consists of all* μ such that:

1. μ = μ1 ∪ μ2, such that
   μ1 ∈ eval(P1,G) and μ2 ∈ eval(P2,G) are compatible and μ(R) = true, or
2. μ ∈ eval(P1,G) and
   there is no compatible μ2 ∈ eval(P2,G) for μ, or
3. μ ∈ eval(P1,G) and
   for any compatible μ2 ∈ eval(P2,G), μ ∪ μ2 does not satisfy R.

Addresses **2)** non-compositionality of FILTERs in OPTIONAL

What again about Blank nodes?

Related to duplicates: Notably, blank nodes might also be considered surprising in SPARQL:

1) Blank nodes in the **data**:
   Two RDF(S)-**equivalent graphs** can yield **different answers**, in SPARQL!

G1
```
_:x  foaf:name  "Reinhard" .
```

$\equiv_{RDF(S)}$

G2
```
_:x  foaf:name  "Reinhard" .
_:y  foaf:name  "Reinhard" .
```

```
SELECT ?X ?Y
FROM G1
WHERE {
        ?X foaf:name ?Y.
      }
```

X	Y
_:b1	"Reinhard"

More on blank nodes **[Mallea+,2011]**

SIEMENS

What again about Blank nodes?

Related to duplicates: Notably, blank nodes might also be considered surprising in SPARQL:

1) Blank nodes in the **data**:
   Two RDF(S)-**equivalent graphs** can yield **different answers**, in SPARQL!

G1
```
_:x   foaf:name   ”Reinhard” .
```
$\equiv_{RDF(S)}$ G2
```
_:x  foaf:name  ”Reinhard" .
_:y  foaf:name  ”Reinhard" .
```

```
SELECT ?X ?Y
FROM G2
WHERE {
        ?X foaf:name ?Y.
      }
```

X	Y
_:b1	“Reinhard”
_:b2	“Reinhard”

More on blank nodes **[Mallea+,2011]**

What again about Blank nodes?

Related to duplicates: Notably, blank nodes might also be considered surprising in SPARQL:

1) Blank nodes in the **data**:
   Two RDF(S)-**equivalent graphs** can yield **different answers**, in SPARQL!

G1
```
_:x  foaf:name  "Reinhard" .
```

$\equiv_{RDF(S)}$

G2
```
_:x  foaf:name  "Reinhard" .
_:y  foaf:name  "Reinhard" .
```

```
SELECT ?Y
FROM G2
WHERE {
        ?X foaf:name ?Y.
      }
```

Y
"Reinhard"
"Reinhard"

More on blank nodes **[Mallea+,2011]**

What again about Blank nodes?

Related to duplicates: Notably, blank nodes might also be considered surprising in SPARQL:

1) Blank nodes in the **data**:
   Two RDF(S)-**equivalent graphs** can yield **different answers**, in SPARQL!
2) Blank nodes in **query patterns**:
   Blank nodes in queries are behaving just like (distinguished) variables)

G1
```
_:x   foaf:name   "Reinhard" .
```

$\equiv_{RDF(S)}$

G2
```
_:x  foaf:name  "Reinhard" .
_:y  foaf:name  "Reinhard" .
```

```
SELECT ?Y
FROM G2
WHERE {
        _:x foaf:name ?Y.
      }
```

Y
"Reinhard"
"Reinhard"

More on blank nodes **[Mallea+,2011]**

Summary Datalog 2.0: What in SPARQL1.0 can/cannot NOT be done in Datalog?

We can encode SPARQL fairly straightforwardly in in nonrecursive Datalog^{not} . [Angle&Gutierrez, 2008] Polleres&Schindlauer, 2007]

Duplicates a bit tricky, but

- duplicates by UNION can be covered easily
- we may consider procjection (SELECT) as postprocessing

Alternative: How about Datalog with bag semantics?

[Singh, et al. 1993][Green+,2007]

However, bag semantics is problematic, even for conjunctive queries (containment undecidable, cf. **[Jayram+, 2006])**

Other features **not** encodable directly in Datalog:

LIMIT, ORDER BY, OFFSET

??? (Work-Arounds could be thought of, likely not to be very elegant)

The Rule Interchange Format (RIF)

RIF and Datalog

[W3C, 2010]

from http://rossiter-designs.blogspot.co.at/2011/04/reading-is-fun.html

SIEMENS

What is RIF?

- RIF is a Rule **Interchange** Format (XML) to exchange rules
  - different dialects (Core, Basic Logic (RIF-BLD), Production Rules (RIF-PRD)
  - Closest to Datalog: RIF Core

- RIF Core **[W3C,2010a]** is (essentially)
  - Positive Datalog
  - With equality (in facts).
  - With a standard library of Built-in functions and predicates (RIF-DTB),**[W3C, 2010b]**
  - Interplays well with RDF+OWL **[W3C, 2010c]**

Example – Why Rules?

Full name in FOAF from givenName, familyName, assuming Datalog with built-ins:

```
triple(F, foaf:name, N ) :-
  triple(X, rdf:type, foaf:Person),
  triple(X, foaf:givenName, F ),
  triple(X, foaf:familyName S ), N = fn:concat(F, " ", S) .
```

- Not expressible in SPARQL1.0 CONSTRUCT (neither in OWL, btw)

```
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[rdf:type->foaf:Person]
           ?X[foaf:givenName->?F],
           ?X[foaf:familyName->?S],
           ?N = fn:concat(?F, “ “, ?S) .
```

- We use a simplified version of RIF’s presentation syntax here.
- RIF has chosen F-Logic style Frames (e.g. FLORA-2)to represent RDF-Triples, cf. [**W3C 2010c]**
- Can just be viewed as “syntactic sugar” for the triple() predicate we used before

RIF and RDF

1) RDFS entailment rules encodable in RIF Core … obvious.

2) RIF Core Semantics has Datatype reasoning built-in!

RDF Graph:

```
document1 :language  "en"^^xsd:language .
```

RIF Rule:

```
?X[ rdf:type -> :EngDocument ] :-
             ?X[ :language -> "en"^^xsd:string ] .
```

The RDF+RIF combined semantics **[W3C,2010d]** would entail

```
document1 rdf:type :EngDocument .
```

SIEMENS

RIF and SPARQL

Can we Interpret SPARQL CONSTRUCT as a “rules language”?
[Polleres, 2007], [Schenk&Staab,2008], [Knublauch et al. 2011]
Would this rule language be exchangeable in RIF Core?

3 main obstacles:

1) Built-ins:
   → A RIF dialect including SPARQL built-ins would need specific built-ins.
   (e.g. **bound()**, **datatype()** are not in DTB)
   → The error semantics of complex FILTERs in SPARQL would need to be emulated in RIF.

2) Negation as failure or something like OPTIONAL would be needed.

3) Datatype Reasoning is built-in into RIF but not in SPARQL.

```
CONSTRUCT { ?X rdf:type  :EngDocument }
WHERE { ?X[ :language “en”^^xsd:string } .
```

No results on the RDF graph of the previous slide!

***Bottomline**: at seems that SPARQL has both more and less than RIF-Core*
→RIF-SPARQL would need an own RIF-“Dialect”

RIF and Datalog – Summary:

- Positive Datalog is in RIF Core.
- To “cover” RIF Core, you’d need Datalog+Built-ins.
  Termination problems, could be remedied by syntactic restrictions, e.g. “Strong safeness” **[W3C, 2010a, §6,2]**, inspired by **[Eiter+,2006]**
- Common extensions to Datalog would need an own RIF Dialect (e.g. *not*)
- In combination with SPARQL, some obstacles would need to be overcome.

Coming soon!

SIEMENS

SPARQL1.1

Why SPARQL1.1 was needed…

In 2009, a new W3C SPARQL WG was chartered to common feature requests by the community in the query language:

1. **Negation**
2. **Assignment/Project Expressions**
3. **Property paths**
4. **Subqueries**
5. **Aggregate functions (SUM, AVG, MIN, MAX, COUNT, …)**
6. **Simple query federation**
7. **Entailment Regimes**

- ***Goal: SPARQL 1.1 W3C Recommendation by end of this year***

Negation

Negation can now be directly expressed in SPARQL1.1:

Give me people without an email address:

```
SELECT ?X ?M
 WHERE { ?X foaf:knows ?Y
         MINUS {?X :email ?M . }
       }
```

We know how to do that… Negation as failure.

Assignment/Project Expressions

Adds the ability to create new values

```
CONSTRUCT { ?X foaf:name ?N }
    WHERE { ?X a foaf:Person;
            ?X foaf:givenName ?F ; foaf:familyName ?S
           BIND( fn:concat(?F, " ", ?S) AS ?N ) }
```

We spoke about this already, in the context of RIF, need built-ins.

SIEMENS

PropertyPaths in SPARQL1.1

```
SELECT ?X
  WHERE {:tim foaf:knows+ ?X
        }
```

That's transitive closure, we know how to do this!

```
answer(X) :- Path+(tim,knows,X) .


Path+(X,P,Y) :- triple(X, P, Y ) .
Path+(X,P,Z) :- triple(X, P, Y ), Path+(Y,P,Z) .
```

Remark1: Only linear recursion added!

Remark2: No duplicates for *,+ … An earlier WD of the SPARQL1.1 WG had defined a semantics for property paths with duplicates… caused difficulties for implementations and complexity explosion [Arenas et al., 2012], [Losemann&Martens, 2012]

SIEMENS

PropertyPaths in SPARQL1.1

```
SELECT ?X
  WHERE {:tim foaf:knows* ?X
        }
```

That's transitive closure, we know how to do this!

```
answer(X) :- Path*(tim,knows,X) .
Path*(X,P,X).
Path*(X,P,Y) :- Path*(X,P,Y) .
Path+(X,P,Y) :- triple(X, P, Y ) .
Path+(X,P,Z) :- triple(X, P, Y ), Path+(Y,P,Z) .
```

Remark1: Only linear recursion added!

Remark2: No duplicates for *,+ … An earlier WD of the SPARQL1.1 WG had defined a semantics for property paths with duplicates… caused difficulties for implementations and complexity explosion [Arenas et al., 2012], [Losemann&Martens, 2012]

PropertyPaths in SPARQL1.1 + RDFS

```
SELECT ?X ?L
  WHERE {?X rdf:type foaf:Person. ?X rdfs:label ?L
        }
```

Include RDFS inferences by property paths:

```
SELECT ?X ?L
   WHERE { ?X rdf:type/rdfs:subClassOf* foaf:Person.
           ?X ?P ?L . ?P rdfs:subPropertyOf* rdfs:label.
         }
```

Remark3: Essential RDFS reasoning can be “encoded” in property paths.
cf. also PSPARQL [Alkateeb+,2009], nSPARQL [Perez+,2010]

SIEMENS

More on Duplicates in Property Paths in SPARQL1.1

An RDF Graph including RDF lists:

```
:s  :p  _:b1.
        _:b1 rdf:first 1 . _:b1 rdf:rest _:b2 .
        _:b2 rdf:first 1 . _:b1 rdf:rest _:b3 .
        _:b3 rdf:first 2 . _:b1 rdf:rest rdf:nil.
```

Example Query: *Members of the list?*

```
 SELECT ?X
WHERE { :s :p/rdf:rest*/rdf:first ?X}
```

Expected result (by majority in the W3C WG):

X
1
1
2

Boils down to:

```
SELECT ?X
WHERE { :s :p ?P1. ?P1 rdf:rest* ?P2. ?P2 rdf:first ?X}
```

Again! Duplicates (by –implicit – projection)

Subqueries

“Give me a list of scientists (that have been born or died there) for cities in Austria”

```
SELECT ?X
 { ?Y dbpedia:country dbpediares:Austria .
  { SELECT DISTINCT ?Y ?X
    WHERE { { ?X dbpedia:birthPlace  ?Y } UNION { ?X dbpedia:deathPlace ?Y }
              ?X rdf:type dbpedia:Scientist.   }
  }
}
```

Implications:

1) For one: adds “real” projection
2) Can be combined with other features of SPARQL (DISTINCT, LIMIT, ORDER…)

Note that subqueries in SPARQL 1.1 are very simple **[Angles&Gutierrez,2011]**

Why SPARQL1.1 was needed…

In 2009, a new W3C SPARQL WG was chartered to common feature requests by the community in the query language:

1. Negation
2. Assignment/Project Expressions
3. Property paths
4. Subqueries
5. **Aggregate functions (SUM, AVG, MIN, MAX, COUNT, …)**
   related to aggregates in Datalog, e.g. **[Faber+,2011]**?
6. **Simple query federation**
   cf. Jorge Perez' ReasoningWeb Tutorial **[Arenas&Perez,2012]**
7. **Entailment Regimes** (extensions of BGP matching)
   RDFS essentially doable with Entailment Rules,
   OWL …

… Reading W3C specifications is fun! Enjoy! ☺

References 1/5

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

[Bruijn&Heymans,2007] Jos de Bruijn, Stijn Heymans: Logical Foundations of (e)RDF(S): Complexity and Reasoning. ISWC/ASWC 2007: 86-99

[Muñoz+, 2009] Sergio Muñoz, Jorge Pérez, Claudio Gutierrez: Simple and Efficient Minimal RDFS. J. Web Sem. 7(3): 220-234 (2009)

[Ianni+, 2009] Giovambattista Ianni, Thomas Krennwallner, Alessandra Martello, Axel Polleres: Dynamic Querying of Mass-Storage RDF Data with Rule-Based Entailment Regimes. International Semantic Web Conference 2009: 310-327

[Gutierrez+,2011] Claudio Gutierrez, Carlos A. Hurtado, Alberto O. Mendelzon, Jorge Pérez: Foundations of Semantic Web databases. J. Comput. Syst. Sci. 77(3): 520-541 (2011)

[W3C, 2008a] SPARQL Query Language for RDF. Eric Prud'hommeaux, Andy Seaborne (Eds.) W3C Recommendation 15 January 2008 http://www.w3.org/TR/rdf-sparql-query/

[W3C, 2008b] SPARQL Protocol for RDF. Kendall Grant Clark, Lee Feigenbaum, Elias Torres (Eds.) W3C Recommendation 15 January 2008 http://www.w3.org/TR/rdf-sparql-protocol/

SIEMENS

References 2/5

[ter Horst, 2005] Herman J. ter Horst: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. J. Web Sem. 3(2-3): 79-115 (2005)

[Perez+, 2006] Jorge Pérez, Marcelo Arenas, Claudio Gutierrez: Semantics and Complexity of SPARQL. International Semantic Web Conference 2006: 30-43

[Perez+, 2009] Jorge Pérez, Marcelo Arenas, Claudio Gutierrez: Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34(3): (2009)

[Angles&Gutierrez, 2008] Renzo Angles, Claudio Gutierrez: The Expressive Power of SPARQL. International Semantic Web Conference 2008: 114-129

[Dantsin+, 2001] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, Andrei Voronkov: Complexity and expressive power of logic programming. ACM Comput. Surv. 33(3): 374-425 (2001)

[Mallea+,2011] Alejandro Mallea, Marcelo Arenas, Aidan Hogan, Axel Polleres: On Blank Nodes. International Semantic Web Conference (1) 2011: 421-437

[Singh+, 1993] Inderpal Singh Mumick and Oded Shmueli. Finiteness properties of database queries. In 4th Australian Database Conference, 1993.

References 3/5

[Green+,2007] Todd J. Green, Gregory Karvounarakis, Val Tannen: Provenance semirings. PODS 2007: 31-40

[Jayram+, 2006] T. S. Jayram, Phokion G. Kolaitis, Erik Vee: The containment problem for REAL conjunctive queries with inequalities. PODS 2006: 80-89

[Arenas+,2012] Marcelo Arenas, Sebastián Conca, Jorge Pérez: Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. WWW 2012: 629-638

[Losemann&Mrtens,2012] Katja Losemann, Wim Martens: The complexity of evaluating path expressions in SPARQL. PODS 2012: 101-112

[Alkhateeb+, 2009] Faisal Alkhateeb, Jean-François Baget, Jérôme Euzenat: Extending SPARQL with regular expression patterns (for querying RDF). J. Web Sem. 7(2): 57-73 (2009)

[Perez+,2010] Jorge Pérez, Marcelo Arenas, Claudio Gutierrez: nSPARQL: A navigational language for RDF. J. Web Sem. 8(4): 255-270 (2010)

[W3C, 2010] RIF Overview. Michael Kifer, Harold Boley (Eds.) W3C Working Group Note 22 June 2010 http://www.w3.org/TR/rif-overview/

[W3C 2010a] RIF Core Dialect. Harold Boley et al. (Eds.) W3C Recommendation 22 June 2010. http://www.w3.org/TR/rif-core/

References 4/5

[W3C 2010b] RIF Datatypes and Built-Ins 1.0. Axel Poleres et al. (Eds.) W3C Recommendation 22 June 2010 http://www.w3.org/TR/rif-dtb/

[W3C 2010c] RIF RDF and OWL Compatibility. Jos de Bruijn (Ed.) W3C Recommendation 22 June 2010 http://www.w3.org/TR/rif-rdf-owl/

[Knublauch+,2011] SPIN - Overview and Motivation. Holger Knublauch, James A. Hendler, Kingsley Idehen. W3C Member Submission 22 February 2011 http://www.w3.org/Submission/spin-sparql/

[Schenk&Staab, 2008] Simon Schenk, Steffen Staab: Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. WWW 2008: 585-594

[Polleres, 2007] Axel Polleres: From SPARQL to rules (and back). WWW 2007: 787-796

[Polleres&Schindlauer,2007] Axel Polleres, Roman Schindlauer: DLVHEX-SPARQL: A SPARQL Compliant Query Engine Based on DLVHEX. ALPSWS 2007

[Eiter+,2006] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, Hans Tompits: Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning. ESWC 2006: 273-287

[Angles&Gutierrez,2011] Renzo Angles, Claudio Gutierrez: Subqueries in SPARQL. AMW 2011

References 5/5

[Faber+,2011] Wolfgang Faber, Gerald Pfeifer, Nicola Leone: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. 175(1): 278-298 (2011)

[Arenas&Perez,2012] Federation and Navigation in SPARQL 1.1, In Reasoning Web 2012 (Springer)

The latest drafts of the SPARQL1.1 working group are available at:
http://www.w3.org/TR/sparql11-overview/ … **SPARQL 1.1 Overview**
http://www.w3.org/TR/sparql11-query/ … **SPARQL 1.1 Query Language**
http://www.w3.org/TR/sparql11-entailment/ … **SPARQL 1.1 Entailment Regimes**
http://www.w3.org/TR/sparql11-federated-query/ … **SPARQL 1.1 Federated Query**
http://www.w3.org/TR/sparql11-update/ …**SPARQL 1.1 Update**
http://www.w3.org/TR/sparql11-protocol/ … **SPARQL 1.1 Protocol**
http://www.w3.org/TR/sparql11-http-rdf-update/ … **SPARQL 1.1 Graph Store HTTP Protocol**