European Semantic Web Conference 2007
Tutorial

SPARQL – Where are we?
Current state, theory and practice

Marcelo Arenas (Pontificia Universidad Católica de Chile)
Claudio Gutierrez (Universidad de Chile)
Bijan Parsia (University of Manchester)
Jorge Pérez (Pontificia Universidad Católica de Chile)
Axel Polleres (DERI, National University of Ireland, Galway)
Andy Seaborne (Hewlett-Packard Laboratories)

Supported by FONDECYT grants 1070732, 1070348, CWR project P04-067-F,
EU FP6 project inContext IST-034718, MEC/URJC project SWOS URJC-CM-2006-CET-0300.

SPARQL – Where are we?
Current state, theory and practice

Marcelo Arenas[1], Claudio Gutierrez[2], Bijan Parsia[3],
Jorge Pérez[1], Axel Polleres[4], and Andy Seaborne[5]

[1] Pontificia Universidad Católica de Chile
[2] Universidad de Chile
[3] University of Manchester
[4] DERI, National University of Ireland, Galway
[5] Hewlett-Packard Laboratories

Abstract. After the data and ontology layers of the Semantic Web stack have achieved considerable stability, the query layer, realized by SPARQL, is the next item on W3C's agenda. Short before its completion, we will take the opportunity to reflect on the current state of the language, its applications, recent results on theoretical foundations, and also future challenges. This tutorial will teach SPARQL along two complementary streams: On the one hand, we will provide a practical introduction for newcomers, giving examples from various application domains, providing formal underpinnings and guiding attendees through the jungle of existing implementations, including those which reach beyond the current specification to query more expressive semantic web languages. On the other hand, we will go further into the theoretical foundations of SPARQL, presenting recent results of SPARQL's complexity, formalization in terms of database theory, as well as its exact semantic relation to the other building blocks in the SW stack, namely, RDF Schema, OWL and the rules layer.

1 Motivation and Objectives

After the data and ontology layers of the Semantic Web stack have achieved considerable stability through standard recommendations such as RDF and OWL, the query layer is the next item to be completed on W3C's agenda. This layer is realized by the SPARQL Protocol and RDF Query Language (SPARQL) currently under development by W3C's Data Acceess working group (DAWG). Although the SPARQL specification is not yet 100% stable, people are taking up this specification at tremendous pace, driven by the strong need for a long awaited standard in querying the Semantic Web and being able of making use of the advantages of RDF together with common metadata-vocabularies at large scale.

This is just the right moment to reflect on the current state of the language and its applications. The contributions of this tutorial will be along two complementary main streams: On the one hand we will provide a practical introduction to SPARQL for newcomers, giving examples from various application domains, providing formal underpinnings and guiding attendees through the jungle of existing implementations, including those which reach beyond the current specification to query more expressive semantic web languages. Thus, participants will get a clear sense of the language as it is specified and as it exists in implementations. On the other hand, we will go further going into the theoretical foundations of SPARQL, presenting recent results of SPARQL's complexity, and its exact semantics relation to the other building blocks in the SW stack, namely, RDF Schema, OWL and the upcoming rules layer. Finally, we will bring these two streams together, identifying the current limitations and challenges around SPARQL, pointing to possible extensions and emerging application fields.

After the tutorial, attendees new to SPARQL should be able to formulate queries, understand the differences and overlaps of SPARQL with traditional Database query languages and have sufficient insight to understand issues in existing SPARQL engines that might affect their applications. The theoretical background given in the afternoon session will provide deeper understanding of SPARQL's underlying semantics and complexity. Moreover, we will provide a detailed picture of SPARQL's position in the space of related Semantic Web standards. Finally, we will give an outlook to emerging research challenges and possible future directions.

2 Outline

The tutorial will be divided in two main parts: The morning part covering primarily the practical side of SPARQL, and an afternoon part going more into depth towards the foundational aspects of SPARQL and discussing Semantic Web data access in the bigger context of related standards. More specifically, the tutorial will be organized in six units, where Units 1-3 mark the morning part and Units 4-6 the afternoon part, as follows.

Unit 1 – SPARQL Basics (90min) The first Unit is tailored to give a gentle introduction to all the features of SPARQL, starting from simple queries towards more complex less used, but interesting features of the language. In this session we aim to guide new users, but also users already roughly familiar with SPARQL, through all major features of the language. Moreover, we will provide interesting insights in design rationale and requirements which guided the inclusion of these features in the process of the working group.

Unit 2 – SPARQL Semantics (45min) In this Unit we will present the algebra underlying and defining a formal semantics for SPARQL. The formal semantics presented here will be exemplified by several examples from the first unit and practical users of the language will get sufficient insight to understand the formal underpinnings of SPARQL. Here, we will restrict ourselves to only the level of detail necessary to understand implementation aspects covered in Unit 3. More in-depth considerations on theoretical foundations will be covered in Unit 4.

Unit 3 – SPARQL Implementations and Applications (45min) First, presenting some basic implementation strategies for SPARQL, we will present several actual implementations and their actual deployment in use cases. The focus in this Unit is to give practitioners hints on tools and available implementations, APIs which they can use off-the-shelf or optimize in order to develop Semantic Web applications on top of SPARQL. This will include examining implementations that go beyond the current specification to evaluate SPARQL queries against RDFS and OWL datasets.

Unit 4 – SPARQL Foundations (90min). In this session of the tutorial, we address the database foundations of SPARQL covering: I) Formal aspects of querying RDF data. II) Formal Semantics of SPARQL: algebraic syntax, compositional semantics for a core fragment (continuation of Unit 2). III) Complexity of SPARQL: covering the computational complexity of evaluating a query for several fragments of the language, identifying the main sources of complexity. IV) Ad-hoc optimization procedures: well-designed queries, reordering and normal forms, and optimization based on normal forms.

Unit 5 – SPARQL and its neighbour components in the Semantic Web stack (90min). The definitions in the current specification of SPARQL focus mainly on RDF simple entailment. In this unit, we show how they can be extended towards coverage of the ontology layer in terms of RDFS and OWL entailment. As it turns out, this extension is not straightforward and a complete coverage of SPARQL imposes new challenges on OWL Reasoners. Next, we will study the emerging Semantic Web rules layer and its relation to SPARQL. On the one hand, will see that a large part of SPARQL can be mapped to extended Datalog, a deductive rule based query. On the other hand, we will discuss the use of SPARQL itself as a declarative rules language on top of RDF and OWL. As it turns out, the challenges arising in such a combination are closely related to those combining deductive non-monotonic rules languages with Ontologies.

Unit 6 – SPARQL Extensions and Outlook (45min). In the last unit of this tutorial, we will discuss further practical extensions of the current standard from simple extensions which simply did not find their way yet in the first version of the specification, to other extensions which seemingly easy will require significant more investigation and raise new research problems. One aim of this unit is to spark further ideas on solving open issues by providing a down-to earth analysis of current limitations of the language.

European Semantic Web Conference 2007
Tutorial

SPARQL – Where are we?
Current state, theory and practice

Unit 1: SPARQL Basics

SPARQL

Andy Seaborne

SPARQL

1. Query Language
2. Protocol
   - HTTP binding
   - SOAP binding
3. XML Results Format
   - Easy to transform (XSLT, XQuery)

- Status: Later stages of standardisation
  - Design finished, getting implementation feedback

SPARQL

- Basic Graph Pattern Matching
  - Building block for data access and extensibility
- Algebra: combining graph patterns
  - Building block for data access and extensibility
  - Filters for restricting values
- Solution Modifiers
  - ORDER BY, LIMIT/OFFSET, DISTINCT, REDUCED
- Result forms
  - SELECT, CONSTRUCT, DESCRIBE, ASK

It’s Turtles all the way down

- Turtle: An RDF serialization
  - The RDF part of N3
  - Commonly used in examples (and tutorials and papers)
  - SPARQL uses Turtle+variables as triple pattern syntax

```
@prefix person: <http://example/person/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .

person:A foaf:name "Alice" .
person:A foaf:mbox <mailto:alice@example.net> .

person:B foaf:name "Bob" .
```

SPARQL : Triple Pattern

```
@prefix person: <http://example/person/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .

person:A foaf:name "Alice" .
person:A foaf:mbox <mailto:alice@example.net> .
person:B foaf:name "Bob" .
```

```
PREFIX person: <http://example/person/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE
  { ?x foaf:name ?name }
```

```
-----------
| name    |
===========
| "Bob"   |
| "Alice" |
-----------
```

SPARQL : Basic Graph Pattern

```
@prefix person: <http://example/person/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .

person:A foaf:name "Alice" .
person:A foaf:mbox <mailto:alice@example.net> .
person:B foaf:name "Bob" .
```

```
PREFIX person: <http://example/person/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE
  { ?person foaf:mbox <mailto:alice@example.net> .
    ?person foaf:name ?name . }
```

```
-----------
| name    |
===========
| "Alice" |
-----------
```

SPARQL : FILTER

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix stock: <http://example.org/stock#> .
@prefix inv: <http://example.org/inventory#> .

stock:book1 dc:title "SPARQL Query Language Tutorial" .
stock:book1 inv:price 10 .
stock:book1 inv:quantity 3 .

stock:book2 dc:title "SPARQL Query Language (2nd ed)" .
stock:book2 inv:price 20 ; inv:quantity 5 .

stock:book3 dc:title "Moving from SQL to SPARQL" .
stock:book3 inv:price 5 ; inv:quantity 0 .

stock:book4 dc:title "Applying XQuery" .
stock:book4 inv:price 20 ; inv:quantity 8 .
```

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX stock: <http://example.org/stock#>
PREFIX inv: <http://example.org/inventory#>

SELECT ?book ?title
WHERE {
 ?book dc:title ?title .
 ?book inv:price ?price . FILTER ( ?price < 15 )
 ?book inv:quantity ?num . FILTER ( ?num > 0 ) }
```

```
 --------------------------------------------------
 | book        | title                            |
 ==================================================
 | stock:book1 | "SPARQL Query Language Tutorial" |
--------------------------------------------------
```

SPARQL : OPTIONAL

```
@prefix person: <http://example/person/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

person :a foaf:name "Alice" .
person :a foaf:nick "A-online" .

person:b foaf:name "Bob" .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?nick
{ ?x foaf:name ?name .
   OPTIONAL {?x foaf:nick ?nick }
}
```

```
------------------------
| name    | nick       |
========================
| "Alice" | "A-online" |
| "Bob"   |            |
------------------------
```

SPARQL : UNION

```
@prefix book: <http://example/book/> .
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .

book:a   dc10:title "SPARQL Query Language Tutorial" .
book:b   dc11:title "SPARQL Query Language (2nd ed)" .
book:c   dc10:title "SPARQL" .
book:c   dc11:title "SPARQL" .
```

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>

SELECT DISTINCT ?title
{
   { ?book dc10:title ?title } UNION { ?book dc11:title ?title }
}
```

```
------------------------------------
| title                            |
====================================
| "SPARQL Query Language Tutorial" |
| "SPARQL"                         |
| "SPARQL Query Language (2nd ed)" |
------------------------------------
```

Solution Modifiers

- After matching, the set of solutions is turned into a sequence then:
  - ORDER BY
  - Project
  - DISTINCT, REDUCED
  - OFFSET
  - LIMIT

Result Sets

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">

  <head>
    <variable name="name"/>
    <variable name="mbox"/>
  </head>

  <results ordered="false" distinct="false">
    <result>
      <binding name="name"><literal>Johnny Lee Outlaw</literal></binding>
      <binding name="mbox"><uri>mailto:jlow@example.com</uri></binding>
    </result>

    <result>
      <binding name="mbox"><uri>mailto:peter@example.org</uri></binding>
    </result>
  </results>

</sparql>
```

```
----------------------------------------------------
| name                | mbox                       |
====================================================
| "Johnny Lee Outlaw" | <mailto:jlow@example.com>  |
|                     | <mailto:peter@example.org> |
----------------------------------------------------
```

Inference

- An RDF graph may be backed by inference
  - OWL, RDFS, application, rules

```
:x rdf:type :C .
:C rdfs:subClassOf :D .
```

```
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?type
WHERE
{
    ?x rdf:type ?type .
}
```

```
--------
| type |
========
| :C   |
| :D   |
--------
```

CONSTRUCT

```
@prefix person: <http://example/person/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .

person:a foaf:name "Alice" .
person:a foaf:mbox <mailto:alice@example.net> .
person:b foaf:name "Bob" .
```

```
PREFIX person: <http://example/person/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?person vcard:FN ?name }
WHERE
  {?person foaf:name ?name . }
```

```
@prefix person: <http://example/person/> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

person:a vcard:FN "Alice" .
person:b vcard:FN "Bob" .
```

SPARQL : RDF Dataset

- RDF Dataset – collection of graphs
  - One, unnamed default graph ;
  - Zero or more named graphs
- Access with the GRAPH keyword

```
SELECT . . .
FROM <contact.ttl>
FROM NAMED <aliceFoaf.ttl>
FROM NAMED <bobFoaf.ttl>
WHERE { . . . }
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?graph ?name
WHERE
{ ?alice foaf:name "Alice" .
  ?alice foaf:mbox ?mbox .
  GRAPH ?graph
  { ?x foaf:mbox ?mbox .
    ?x foaf:knows ?person .
    ?person foaf:name ?name .
  }
}
```

European Semantic Web Conference 2007
Tutorial

SPARQL – Where are we?
Current state, theory and practice

Unit 2: SPARQL Semantics

SPARQL Formalization

Marcelo Arenas, Claudio Gutierrez, Jorge Pérez

Department of Computer Science
Pontificia Universidad Católica de Chile
Universidad de Chile

Center for Web Research
http://www.cwr.cl

SPARQL: A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

- The *semantics* of simple SPARQL queries is easy to understand, at least intuitively.

> *"Give me the name and email of the resources in the datasource"*

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- .....

```
{ { P1
    P2 }
    OPTIONAL { P5 }  }

  { P3
    P4 }
    OPTIONAL { P7 }  }
      OPTIONAL { P8 }  }  }
}
UNION
{ P9 }
  FILTER ( R ) }
```

A formal semantics for SPARQL is needed.

A formal approach would be beneficial

- Clarifying corner cases
- Helping in the implementation process
- Providing sound foundations

We will see:

- A formal compositional semantics based on **[PAG06: Semantics and Complexity of SPARQL]**
- This formalization is the starting point of the official semantics of the SPARQL language by the W3C.

Outline

Motivation

Basic Syntax

Semantics

Datasets

Query result forms

Dealing with bnodes

Dealing with duplicates

First of all, a simplified algebraic syntax

- Triple patterns: RDF triple + variables (no bnodes for now)

$$(?X,\ \text{name},\ ?\mathit{Name})$$

- The base case for the algebra is a set of triple patterns

$$\{t_1, t_2, \ldots, t_k\}.$$

This is called basic graph pattern (BGP).

Example

$$\{\ (?X,\ \text{name},\ ?\mathit{Name}),\ \ (?X,\ \text{email},\ ?\mathit{Email})\ \}$$

First of all, a simplified algebraic syntax (cont.)

- We consider initially three basic operators:

  AND, UNION, OPT.
- We will use them to construct graph pattern expressions from basic graph patterns.
- A SPARQL graph pattern:

  $$(((\{t_1, t_2\} \text{ AND } t_3) \text{ OPT } \{t_4, t_5\}) \text{ AND } (t_6 \text{ UNION } \{t_7, t_8\}))$$

  it is a full parenthesized expression
- Full parenthesized expressions give us explicit precedence/association.

Mappings: building block for the semantics

Definition

A mapping is a partial function from variables to RDF terms.

Given a mapping μ and a basic graph pattern P:

- dom(μ): the domain of μ.
- $\mu(P)$: the set obtained from P replacing the variables according to μ

Example

$$\mu = \{?X \rightarrow R_1, ?Y \rightarrow R_2, ?Name \rightarrow \text{john}, ?Email \rightarrow \text{J@ed.ex}\}$$

$$P = \{(?X, \text{name}, ?Name), (?X, \text{email}, ?Email)\}$$

$$\mu(P) = \{(R_1, \text{name}, \text{john}), (R_1, \text{email}, \text{J@ed.ex})\}$$

The semantics of basic graph pattern

Definition

The evaluation of the BGP P over a graph G, denoted by $[\![P]\!]_G$, is the set of all mappings μ such that:

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

Example

G

$(R_1, \text{name}, \text{john})$

$(R_1, \text{email}, \text{J@ed.ex})$

$(R_2, \text{name}, \text{paul})$

$$[\![\{(?X, \text{name}, ?Y)\}]\!]_G$$

$$\left\{ \begin{array}{l} \mu_1 = \{?X \to R_1, ?Y \to \text{john}\} \\ \mu_2 = \{?X \to R_2, ?Y \to \text{paul}\} \end{array} \right\}$$

	$?X$	$?Y$
μ_1	R_1	john
μ_2	R_2	paul

$$[\![\{(?X, \text{name}, ?Y), (?X, \text{email}, ?Z)\}]\!]_G$$

$$\{\ \mu = \{?X \to R_1, ?Y \to \text{john}, ?Z \to \text{J@ed.ex}\}\ \}$$

	$?X$	$?Y$	$?Z$
μ	R_1	john	J@ed.ex

Example

G

$(R_1$, name, john$)$

$(R_1$, email, J@ed.ex$)$

$(R_2$, name, paul$)$

$$\llbracket\{(R_1, \text{webPage}, ?W)\}\rrbracket_G$$

$$\{\quad\}$$

$$\llbracket\{(R_3, \text{name}, \text{ringo})\}\rrbracket_G$$

$$\{\quad\}$$

$$\llbracket\{(R_2, \text{name}, \text{paul})\}\rrbracket_G$$

$$\{\ \mu_\emptyset = \{\ \}\ \}$$

$$\llbracket\{\ \}\rrbracket_G$$

$$\{\ \mu_\emptyset = \{\ \}\ \}$$

Compatible mappings: mappings that can be merged.

Definition

The mappings μ_1, μ_2 are compatibles iff they agree in their shared variables:

- $\mu_1(?X) = \mu_2(?X)$ for every $?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)$.

$\mu_1 \cup \mu_2$ is also a mapping.

Example

	$?X$	$?Y$	$?U$	$?V$
μ_1	R_1	john		
μ_2	R_1		J@edu.ex	
μ_3			P@edu.ex	R_2
$\mu_1 \cup \mu_2$	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$	R_1	john	P@edu.ex	R_2

$\mu_\emptyset = \{\ \}$ is compatible with every mapping.

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- $\{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2, \text{ and } \mu_1, \mu_2 \text{ are compatibles}\}$
- extending mappings in M_1 with compatible mappings in M_2

will be used to define AND

Definition

Union: $M_1 \cup M_2$

- $\{\mu \mid \mu \in M_1 \text{ or } \mu \in M_2\}$
- mappings in M_1 plus mappings in M_2 (the usual set union)

will be used to define UNION

Sets of mappings and operations

Definition

Difference: $M_1 \smallsetminus M_2$

- $\{\mu \in M_1 \mid \text{ for all } \mu' \in M_2,\ \mu \text{ and } \mu' \text{ are not compatibles}\}$
- mappings in M_1 that cannot be extended with mappings in M_2

Definition

Left outer join: $M_1 \mathbin{\text{⟕}} M_2 = (M_1 \bowtie M_2) \cup (M_1 \smallsetminus M_2)$

- extension of mappings in M_1 with compatible mappings in M_2
- plus the mappings in M_1 that cannot be extended.

will be used to define OPT

Semantics of general graph patterns

Definition

Given a graph G the evaluation of a pattern is recursively defined

- $[\![(P_1 \text{ AND } P_2)]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$
- $[\![(P_1 \text{ UNION } P_2)]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$
- $[\![(P_1 \text{ OPT } P_2)]\!]_G = [\![P_1]\!]_G \mathbin{⟕} [\![P_2]\!]_G$

the base case is the evaluation of a BGP.

Example (AND)

G :

$(R_1, \text{name}, \text{john})$ $(R_2, \text{name}, \text{paul})$ $(R_3, \text{name}, \text{ringo})$
$(R_1, \text{email}, \text{J@ed.ex})$ $(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{webPage}, \text{www.ringo.com})$

$$[\![\{(?X, \text{name}, ?N)\} \text{ AND } \{(?X, \text{email}, ?E)\}]\!]_G$$

$$[\![\{(?X, \text{name}, ?N)\}]\!]_G \bowtie [\![\{(?Y, \text{email}, ?E)\}]\!]_G$$

	?X	?N
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

$\bowtie$

	?X	?E
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

	?X	?N	?E
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex

Example (OPT)

G : $(R_1, \text{name}, \text{john})$ $(R_2, \text{name}, \text{paul})$ $(R_3, \text{name}, \text{ringo})$
$(R_1, \text{email}, \text{J@ed.ex})$ $(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{webPage}, \text{www.ringo.com})$

$$[\![\{(?X, \text{name}, ?N)\} \text{ OPT } \{(?X, \text{email}, ?E)\}]\!]_G$$

$$[\![\{(?X, \text{name}, ?N)\}]\!]_G ⟕ [\![\{(?X, \text{email}, ?E)\}]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

⟕

	$?X$	$?E$
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

	$?X$	$?N$	$?E$
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex
μ_2	R_2	paul	

Example (UNION)

G : $(R_1, \text{name}, \text{john})$ $(R_2, \text{name}, \text{paul})$ $(R_3, \text{name}, \text{ringo})$
$(R_1, \text{email}, \text{J@ed.ex})$ $(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{webPage}, \text{www.ringo.com})$

$$[\![\{(?X, \text{email}, ?Info)\} \text{ UNION } \{(?X, \text{webPage}, ?Info)\}]\!]_G$$

$$[\![\{(?X, \text{email}, ?Info)\}]\!]_G \cup [\![\{(?X, \text{webPage}, ?Info)\}]\!]_G$$

	$?X$	$?Info$
μ_1	R_1	J@ed.ex
μ_2	R_3	R@ed.ex

$\cup$

	$?X$	$?Info$
μ_3	R_3	www.ringo.com

	$?X$	$?Info$
μ_1	R_1	J@ed.ex
μ_2	R_3	R@ed.ex
μ_3	R_3	www.ringo.com

Boolean filter expressions (value constraints)

In filter expressions we consider

- the equality $=$ among variables and RDF terms
- a unary predicate bound
- boolean combinations ($\wedge$, $\vee$, $\neg$)

A mapping μ satisfies

- $?X = c$ if $\mu(?X) = c$
- $?X = ?Y$ if $\mu(?X) = \mu(?Y)$
- bound$(?X)$ if μ is defined in $?X$, i.e. $?X \in \text{dom}(\mu)$

Satisfaction of value constraints

- If P is a graph pattern and R is a value constraint then (P FILTER R) is also a graph pattern.

Definition

Given a graph G

- $[\![(P \text{ FILTER } R)]\!]_G = \{\mu \in [\![P]\!]_G \mid \mu \text{ satisfies } R\}$
  i.e. mappings in the evaluation of P that satisfy R.

Example (FILTER)

G : $(R_1, \text{name}, \text{john})$ $(R_1, \text{email}, \text{J@ed.ex})$ $(R_2, \text{name}, \text{paul})$ $(R_3, \text{name}, \text{ringo})$ $(R_3, \text{email}, \text{R@ed.ex})$ $(R_3, \text{webPage}, \text{www.ringo.com})$

$$[\![(\{(?X, \text{name}, ?N)\}\ \text{FILTER}\ (?N = \text{ringo} \vee ?N = \text{paul}))]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

$?N = \text{ringo} \vee ?N = \text{paul}$

	$?X$	$?N$
μ_2	R_2	paul
μ_3	R_3	ringo

Example (FILTER)

G : $(R_1, \text{name}, \text{john})$ $(R_1, \text{email}, \text{J@ed.ex})$ $(R_2, \text{name}, \text{paul})$ $(R_3, \text{name}, \text{ringo})$ $(R_3, \text{email}, \text{R@ed.ex})$ $(R_3, \text{webPage}, \text{www.ringo.com})$

$$[\![((\{(?X, \text{name}, ?N)\}\ \text{OPT}\ \{(?X, \text{email}, ?E)\})\ \text{FILTER}\ \neg\,\text{bound}(?E))]\!]_G$$

	$?X$	$?N$	$?E$
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex
μ_2	R_2	paul	

$\neg\,\text{bound}(?E)$

	$?X$	$?N$
μ_2	R_2	paul

FILTER: differences with the official specification

- We restrict to the case in which all variables in R are mentioned in P.
- This restriction is not imposed in the official specification by W3C.
- The semantics without the restriction does not modify the expressive power of the language.

SPARQL Datasets

- One of the interesting features of SPARQL is that a query may retrieve data from different sources.

Definition

A SPARQL dataset is a set

$$\mathcal{D} = \{G_0, \langle u_1, G_1\rangle, \langle u_2, G_2\rangle, \ldots, \langle u_n, G_n\rangle\}$$

- G_0 is the default graph, $\langle u_i, G_i\rangle$ are named graphs
- $\text{name}(\mathcal{D}) = \{u_1, u_2, \ldots, u_n\}$
- $d_{\mathcal{D}}$ is a function such $d_{\mathcal{D}}(u_i) = G_i$.

The GRAPH operator

if u is an IRI, $?X$ is a variable and P is a graph pattern, then

- $(u \text{ GRAPH } P)$ is a graph pattern
- $(?X \text{ GRAPH } P)$ is a graph pattern

GRAPH will permit us to dynamically change the graph against which our pattern is evaluated.

Semantics of GRAPH

Definition

Given a dataset $\mathcal{D}$ and a graph pattern P

$$[\![(u \text{ GRAPH } P)]\!]_G = [\![P]\!]_{d_{\mathcal{D}}(u)}$$

$$[\![(?X \text{ GRAPH } P)]\!]_G = \bigcup_{u \in \text{name}(\mathcal{D})} \left([\![P]\!]_{d_{\mathcal{D}}(u)} \bowtie \{\{?X \to u\}\} \right)$$

Definition

The evaluation of a general pattern P against a dataset $\mathcal{D}$, denoted by $[\![P]\!]_{\mathcal{D}}$, is the set $[\![P]\!]_{G_0}$ where G_0 is the default graph in $\mathcal{D}$.

Example (GRAPH)

$\mathcal{D}$

G_0:

⟨ tb, G_1: (R_1, name, john) (R_2, name, paul) ⟩
(R_1, email, J@ed.ex)

⟨ trs, G_2: (R_4, name, mick) (R_5, name, keith) ⟩
(R_4, email, M@ed.ex) (R_5, email, K@ed.ex)

$$[\![(\ \text{trs}\ \ \text{GRAPH}\ \{(?X,\ \text{name},\ ?N)\})]\!]_{\mathcal{D}}$$

$$[\![(\ \text{trs}\ \ \text{GRAPH}\ \{(?X,\ \text{name},\ ?N)\})]\!]_{G_0}$$

$$[\![\{(?X,\ \text{name},\ ?N)\}]\!]_{G_2}$$

	$?X$	$?N$
μ_1	R_4	mick
μ_2	R_5	keith

Example (GRAPH)

$\mathcal{D}$

G_0:

⟨ tb, G_1: (R_1, name, john) (R_2, name, paul) ⟩
(R_1, email, J@ed.ex)

⟨ trs, G_2: (R_4, name, mick) (R_5, name, keith) ⟩
(R_4, email, M@ed.ex) (R_5, email, K@ed.ex)

$$[\![(?G\ \text{GRAPH}\ \{(?X,\ \text{name},\ ?N)\})]\!]_{\mathcal{D}}$$

$$[\![\{(?X,\ \text{name},\ ?N)\}]\!]_{G_1} \bowtie \{\{?G \to \text{tb}\}\} \cup$$
$$[\![\{(?X,\ \text{name},\ ?N)\}]\!]_{G_2} \bowtie \{\{?G \to \text{trs}\}\}$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul

$\bowtie \{\{?G \to \text{tb}\}\} \cup$

	$?X$	$?N$
μ_3	R_4	mick
μ_4	R_5	keith

$\bowtie \{\{?G \to \text{trs}\}\}$

$?G$	$?X$	$?N$
tb	R_1	john
tb	R_2	paul
trs	R_4	mick
trs	R_5	keith

SELECT

- Up to this point we have concentrated in the body of a SPARQL query, i.e. in the graph pattern matching expression.
- A query can also process the values of the variables. The most simple processing operation is the selection of some variables appearing in the query.

Definition

- A SELECT query is a tuple (W, P) where P is a graph pattern and W is a set of variable.
- The answer of a SELECT query against a dataset $\mathcal{D}$ is

$$\{\mu_{|W} \mid \mu \in [\![P]\!]_{\mathcal{D}}\}$$

where $\mu_{|W}$ is the restriction of μ to domain W.

CONSTRUCT

- A query can also output an RDF graph.
- The construction of the output graph is based on a template.
- A template is a set of triple patterns possibly with bnodes.

Example

$$T_1 = \{(?X, \text{name}, ?Y), (?X, \text{info}, ?I), (?X, \text{addr}, B)\}$$

with B a bnode

Definition

- A CONSTRUCT query is a tuple (T, P) where P is a graph pattern and T is a template.

CONSTRUCT: Semantics

Definition

The answer of a CONSTRUCT query (T, P) against a dataset $\mathcal{D}$ is obtained by

- for every $\mu \in [\![P]\!]_{\mathcal{D}}$ create a template T_μ with fresh bnodes
- take the union of $\mu(T_\mu)$ for every $\mu \in [\![P]\!]_{\mathcal{D}}$
- discard the not valid RDF triples
  - some variables have not been instantiated.
  - bnodes in predicate positions

Blank nodes in graph patterns

- We allow now bnodes in triple patterns.
- Bnodes act as existentials scoped to the basic graph pattern.

Definition

The evaluation of the BGP P with bnodes over the graph G denoted $[\![P]\!]_G$, is the set of all mappings μ such that:

- $\text{dom}(\mu)$ is exactly the set of variables occurring in P,
- there exists a function θ from bnodes of P to G such that

$$\mu(\theta(P)) \subseteq G.$$

- A natural extension of BGPs without bnodes.
- The algebra remains the same.

Bag/Multiset semantics

- In a bag, a mapping can have cardinality greater than one.
- Every mapping μ in a bag M is annotated with an integer $c_M(\mu)$ that represents its cardinality ($c_M(\mu) = 0$ if $\mu \notin M$).
- Operations between sets of mappings can be extended to bags maintaining duplicates:

Definition

$$\begin{aligned}
\mu \in M = M_1 \bowtie M_2, \quad & c_M(\mu) = \sum_{\mu=\mu_1\cup\mu_2} c_{M_1}(\mu_1)\cdot c_{M_2}(\mu_2),\\
\mu \in M = M_1 \cup M_2, \quad & c_M(\mu) = c_{M_1}(\mu) + c_{M_2}(\mu),\\
\mu \in M = M_1 \setminus M_2, \quad & c_M(\mu) = c_{M_1}(\mu).
\end{aligned}$$

- Intuition: we simply do not discard duplicates.

References

- R. Cyganiak, *A Relational Algebra for SPARQL*. Tech Report HP Laboratories, HPL-2005-170.
- E. Prud'hommeaux, A. Seaborne, *SPARQL Query Language for RDF*. W3C Working Draft, 2007.
- J. Pérez, M. Arenas, C. Gutierrez, *Semantics and Complexity of SPARQL*. In *Int. Semantic Web Conference 2006*.
- J. Pérez, M. Arenas, C. Gutierrez, *Semantics of SPARQL*. Tech Report Universidad de Chile 2006, TR/DCC-2006-17.

SPARQL Algebra

Andy Seaborne

SPARQL Algebra

- SPARQL Syntax => SPARQL Algebra
- Algebra works on
  - MultiSets (= bags) for pattern matching
  - sequences for solution modifiers
- Spec defines the algorithm for translating syntax to algebra
- Spec defines the correct results for evaluation of an algebra expression
  - Implementations are free to choose any way that gives the same results

SPARQL Pattern Semantics

- Bottom-up evaluation
  - Meaning for all queries
  - All syntactically correct queries have defined semantics
- Optional/LeftJoin is conditional
  - Makes queries more natural to write
- Account of pattern matching and solution modifiers

Basic Graph Patterns

- Set of triple patterns
- Building block in SPARQL
- Extension point for other levels of entailment
  - Blank nodes are extensional variables

Compatible Solutions

- Solutions
  - set of variable/value pairs
  - each variable name occurs at most once.
- Two solutions are *compatible* if variables of the same name are associated with the same RDF term
  - “same” means term-equals

```
S0: { }
S1: { ?x/”foo” , ?y/<http://example/y> }
S2: { ?y/<http://example/y> }
S3: { ?x/”bar” }

S0, S1 and S2 are compatible
S0 and S3 are compatible
S2 and S3 are compatible
S1 and S3 are not compatible
```

SPARQL Algebra

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT *
WHERE
  { ?x foaf:name ?name }
```

```
(tolist
  (BGP [triple ?x foaf:name ?name]))
```

- BGP of one triple pattern
- Group of one element
- Removed: Join({}, P) ➔ P

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  *
WHERE
  { ?x  foaf:mbox  ?mbox ;
        foaf:name  ?name .
  }
```

```
(tolist
  (BGP
    [triple ?x foaf:mbox ?mbox]
    [triple ?x foaf:name ?name]
  ))
```

- BGP of two triple patterns
- Group of one element
- BGP triples are not joined together : BGP is the fundamental unit

---

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  ?name ?mbox
WHERE
  { ?x  foaf:mbox  ?mbox ;
        foaf:name  ?name .
  }
```

```
(project (?mbox ?name)
  (tolist
    (BGP
      [triple ?x foaf:mbox ?mbox]
      [triple ?x foaf:name ?name]
    )))
```

- Project modifier

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  ?name
WHERE
  { ?x  foaf:name  ?name .
    FILTER regex(?name, "^Smith")
  }
```

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  ?name
WHERE
  { FILTER regex(?name, "^Smith")
    ?x  foaf:name  ?name .
  }
```

```
(project (?name)
  (tolist
    (filter (regex ?name "^Smith")
      (BGP [triple ?x foaf:name ?name]))))
```

- FILTER
- Filter over the whole group

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  *
WHERE
  { ?x  foaf:name  ?name .
    ?x  foaf:mbox  <mailto:xyz> .
    OPTIONAL
      { ?x  foaf:nick  ?nick }
  }
```

```
(tolist
  (leftjoin
    (BGP
      [triple ?x foaf:name ?name]
      [triple ?x foaf:mbox <mailto:xyz>]
    )
    (BGP [triple ?x foaf:nick ?nick])
    true
  ))
```

- OPTIONAL => LeftJoin
- No explicit null – variables left unbound

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT *
WHERE
  { ?x  foaf:name  ?name .
    ?x  foaf:mbox  <mailto:xyz> .
    OPTIONAL
      { ?x  foaf:nick  ?nick
        FILTER regex(?name, "^Smith")
      }
  }
```

```
(tolist
  (leftjoin
    (BGP
      [triple ?x foaf:name ?name]
      [triple ?x foaf:mbox <mailto:xyz>]
    )
    (BGP [triple ?x foaf:nick ?nick])
    (regex ?name "^Smith")))
```

- Conditional LeftJoin
- filter scope includes OPTIONAL left-hand side (fixed part)
  - Here, ?name is available to the regex()

SPARQL Algebra

- BasicGraphPattern (BGP)
- Filter
- Join
- LeftJoin
- Union
- ToList
- OrderBy
- Distinct
- Reduced
- Project
- Slice

SPARQL Algebra

```
Step 0 : Expand abbreviations for IRIs and triple patterns.

Step 1 : BasicGraphPatterns

Replace all BasicGraphPattern elements by BGP(list of triple patterns)

Step 2 : GroupOrUnionGraphPattern

Replace any GroupOrUnionGraphPattern elements:

    * If the element consists of a single GroupGraphPattern,
          replace with the GroupGraphPattern.
    * If the element consists of multiple GroupGraphPatterns,
          connected with 'UNION' terminals, then
          replace with a sequence of nested union operators:
      e.g. Union(Union(GroupGraphPattern, GroupGraphPattern), GroupGraphPattern).

Step 3 : GraphGraphPattern

Map GRAPH IRI GroupGraphPattern to Graph(IRI, GroupGraphPattern)

Map GRAPH Var GroupGraphPattern to Graph(var, GroupGraphPattern)

Step 4 : . . .
```

SPARQL Algebra

```
Step 4 : GroupGraphPattern

Map all sub-patterns contained in this group
Let SP := List of algebra expressions for sub-patterns
Let F := all filters in the group (not in sub-patterns)
Let G := the empty pattern, {}

for i := 0 ; i < length(SP); i++
   If SP[i] is an OPTIONAL,
      If SP[i] is of the form OPTIONAL(Filter(F, A))
           G := LeftJoin(G, A, F)
      else
           G := LeftJoin(G , A, true)
   Otherwise for expression SP[i], G := Join(G, SP[i])

If F is not empty:
  If G = empty pattern then G := Filter(F, empty pattern)
  If G = LeftJoin(A1, A2, true) then G := LeftJoin(A1, A2, F)
  If G = Join(A1, A2) then G := Filter(F, Join(A1, A2)
  If G = Union(A1, A2) then G := Filter(F, Union(A1, A2))
  If G = Graph(x, A) then G := Filter(F, Graph(x, A))
         where x is a variable or IRI.
The result is G

Step 5 : ...
```

SPARQL Algebra

```
Step 5 : Simplification

Groups of one graph pattern (not a filter) become join({}, A) and can be replaced by A

Replace join({}, A) by A
Replace join(A, {}) by A

Solution Modifiers

Step 1 : ToList
    Let M := ToList(Pattern)
Step 2 : Order By
    M := OrderBy(M, list of order comparators)
Step 3 : Projection
    M := Project(M, vars)
Step 4 : Distinct
    M := Distinct(M)
Step 5 : Reduced
    M := Reduced(M)
Step 5 : OFFSET and LIMIT
    M := Slice(M, start, length)
Result is M
```

European Semantic Web Conference 2007
Tutorial

SPARQL – Where are we?
Current state, theory and practice

Unit 3: SPARQL Implementations and Applications

SPARQL Implementations

Andy Seaborne

SPARQL Implementations

1. ARQ – Complete, general purpose query system
   - SPARQL Parser and serializer
   - SPARQL Algebra
   - SPARQL Execution
   - Results handling
2. SDB – Specialised ARQ extension
   - SPARQL to SQL rewriter

ARQ

http://jena.sf.net/ARQ

Execution Issues

- Transformations:
  - Query String => Algebra expression
  - Algebra => Execution Plan
  - Execution plan => solutions
  - Solutions => query results form
- Stream-based
  - Transformation possible for majority of queries
  - Multi-sets as iterators : ToList is a no-op

Linearization

- Where possible execution is by one stage extending/removing results of previous stage
- Indexing = Substitution
- Exceptions:
  - Nested optionals with locally free variables
    - Non “well-designed” patterns
  - Join/LeftJoin : Out of scope variables in filters
  - These are done bottom-up for correctness

```
PREFIX :    <http://example/>
SELECT *
{ :x1 :p ?v .
  OPTIONAL {
      :x3 :q ?w .
      OPTIONAL { :x2 :p ?v } } }
```

```
PREFIX :    <http://example/>
SELECT *
{ :x1 :p ?v .
  { :x2 :q ?w . FILTER( ?v + ?w < 5 ) }
}
```

SDB

http://jena.svn.sf.net/viewvc/jena/SDB/

SDB Table layouts

- Layout 1
  - Single triple table, RDF terms encoded into the entries
  - c.f. Jena's RDB layout
  - Optimal for fine-grain API use
- Layout 2
  - Triples table ; Quads table ; Nodes table
  - Better for plain SPARQL queries
  - Id and hash forms
- Layout 2+
  - Cached partial queries
  - Inference support
  - (values)

Layout2 / hash variant / single graph

```
CREATE TABLE Triples (
    s BIGINT NOT NULL,
    p BIGINT NOT NULL,
    o BIGINT NOT NULL,
    PRIMARY KEY (s, p, o)
    )

CREATE INDEX PredObj ON Triples (p, o)
CREATE INDEX ObjSubj ON Triples (o, s)


CREATE TABLE Node (
   hash BIGINT NOT NULL,
   lex TEXT NOT NULL,
   lang varchar NOT NULL default '',
   datatype varchar(200) NOT NULL default '',
   type integer NOT NULL default '0',
   PRIMARY KEY (hash)
   )

CREATE UNIQUE INDEX Hash ON Nodes (hash)
```

Example 1

```
PREFIX person: <http://example/person/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?nick
WHERE
{ ?x foaf:name "Fred" .
  ?x foaf:nick ?nick . }
```

```
SELECT                                    -- V_1=?nick
  R_1.lex AS V_1_lex, R_1.datatype AS V_1_datatype,
  R_1.lang AS V_1_lang, R_1.type AS V_1_type
FROM
    Triples AS T_1                        -- ?x foaf:name "Fred"
  INNER JOIN
    Triples AS T_2                        -- ?x foaf:nick ?nick
  ON ( T_1.p = -2290624521842110797       -- Const: <http://xmlns.com/foaf/0.1/name>
   AND T_1.o = 6622531991636827042        -- Const: "Fred"
   AND T_2.p = 5173304175992580252        -- Const: <http://xmlns.com/foaf/0.1/nick>
   AND T_1.s = T_2.s                      -- Join var: ?x
   )
  LEFT OUTER JOIN
    Nodes AS R_1                          -- Var: ?nick
  ON ( T_2.o = R_1.hash )
```

Example 2

```
PREFIX :    <http://example/>
SELECT *
{ :x1 :p ?v .
  OPTIONAL {
      :x3 :q ?w .
      OPTIONAL { :x2 :p ?v } } }
```

```
(leftjoin
  (BGP [triple :x1 :p ?v])
  (leftjoin
    (BGP [triple :x3 :q ?w])
    (BGP [triple :x2 :p ?v])))
```

Example 2

```
SELECT                                   -- V_1=?v V_2=?w
  R_1.lex AS V_1_lex, R_1.datatype AS V_1_datatype, R_1.lang AS V_1_lang, R_1.type AS V_1_type,
  R_2.lex AS V_2_lex, R_2.datatype AS V_2_datatype, R_2.lang AS V_2_lang, R_2.type AS V_2_type
FROM
    ( SELECT *
      FROM Triples AS T_1                -- :x1 :p ?v
      WHERE ( T_1.s = -7272111352983262523 -- Const: <http://example/x1>
         AND T_1.p = 2004134117598721274 -- Const: <http://example/p>
         )
    ) AS T_1
  LEFT OUTER JOIN
    (   ( SELECT *
          FROM Triples AS T_2            -- :x3 :q ?w
          WHERE ( T_2.s = 4693521611208290624 -- Const: <http://example/x3>
             AND T_2.p = -4884978200120352820 -- Const: <http://example/q>
             )
        ) AS T_2
      LEFT OUTER JOIN
        Triples AS T_3                   -- :x2 :p ?v
      ON ( T_3.s = -6898947185675171362  -- Const: <http://example/x2>
       AND T_3.p = 2004134117598721274   -- Const: <http://example/p>
       )
    )
  ON ( ( ( T_3.o IS NULL ) OR ( T_1.o = T_3.o ) ) -- Join var: ?v
   )
  LEFT OUTER JOIN
    Nodes AS R_1                         -- Var: ?v
  ON ( T_1.o = R_1.hash )
  LEFT OUTER JOIN
    Nodes AS R_2                         -- Var: ?w
  ON ( T_2.o = R_2.hash )
```

Example 3

```
PREFIX :    <http://example/>

SELECT *
{
    ?x :p ?v .
    OPTIONAL { ?x :p ?a }
    OPTIONAL { ?x :q ?a }
}
```

```
(leftjoin
  (leftjoin
    (BGP [triple ?x :p ?v])
    (BGP [triple ?x :p ?a])
  )
  (BGP [triple ?x :q ?a])
)
```

Example 3

```
SELECT                                   -- V_1=?x V_2=?v V_3=?a
  R_1.lex AS V_1_lex, R_1.datatype AS V_1_datatype, R_1.lang AS V_1_lang, R_1.type AS V_1_type,
  R_2.lex AS V_2_lex, R_2.datatype AS V_2_datatype, R_2.lang AS V_2_lang, R_2.type AS V_2_type,
  R_3.lex AS V_3_lex, R_3.datatype AS V_3_datatype, R_3.lang AS V_3_lang, R_3.type AS V_3_type
FROM
    ( SELECT COALESCE(T_2.o, T_3.o) AS VC_1, T_1.o AS VC_2, T_1.s AS VC_3
        FROM
          ( SELECT *
            FROM Triples AS T_1          -- ?x :p ?v
            WHERE ( T_1.p = 2004134117598721274 -- Const: <http://example/p>
               )
          ) AS T_1
        LEFT OUTER JOIN
          Triples AS T_2                 -- ?x :p ?a
        ON ( T_2.p = 2004134117598721274 -- Const: <http://example/p>
         AND T_1.s = T_2.s               -- Join var: ?x
         )
        LEFT OUTER JOIN
          Triples AS T_3                 -- ?x :q ?a
        ON ( T_3.p = -4884978200120352820 -- Const: <http://example/q>
         AND T_1.s = T_3.s               -- Join var: ?x
         AND ( ( T_2.o IS NULL ) OR ( T_2.o = T_3.o ) ) -- Join var: ?a
         )
      ) AS M_1                           -- ?a as M_1.VC_1, ?v as M_1.VC_2, ?x as M_1.VC_3
    LEFT OUTER JOIN
      Nodes AS R_1                       -- Var: ?x
    ON ( M_1.VC_3 = R_1.hash )
    LEFT OUTER JOIN
      Nodes AS R_2                       -- Var: ?v
    ON ( M_1.VC_2 = R_2.hash )
    LEFT OUTER JOIN
      Nodes AS R_3                       -- Var: ?a
    ON ( M_1.VC_1 = R_3.hash )
```

Engines, and Endpoints, and Apps!

(oh my)

- The ESW Wiki is a good source:
  - http://esw.w3.org/topic/SparqlImplementations
  - http://esw.w3.org/topic/DawgShows
- Far too much to explore now!
  - Brief mention of notable engines
  - Tour of several SPARQL based apps
- Excellent web client
  - http://demo.openlinksw.com/sparql_demo/#

(some) Notable RDF engines

- Oracle (SPARQL syntax coming)
- AllegroGraph
- OpenLink Virtuoso (Open Source as well)
- ARQ and Joseki from HP
- IBM's Boca (ARQ and native interface)
- Rasqal for Redland
- SWI-Prolog
- Sesame
- D2R Server

(Some)notable OWL engines

(With conj. Query support)

- Pellet
- KAON2
- Racer (Not SPARQL syntax yet)
- QuOnto (DL Lite, online demo, not SPARQL syntax yet)

Garlik.com

- UK Based tech startup
  - "give people real power over their online data"
  - $18.5m in venture capital
  - Incorporates members from the 3Store team
- DataPatrol
  - Reports on personal information online
  - Uses SPARQL to build these reports
  - Currently 57,000 users!
  - See the demo:
    - http://www.garlik.com/index3.php?page=demo
- Key developer, Steve Harris, member of DAWG

Garlik: Tech details

- Reports
  - 500-2000 SPARQL queries to build a report
    - Often recursive, i.e., using prior results to find next ones
  - 8 knowledge bases of 2 billion triples each
  - Reports take **1-2 seconds** to generate
- Query characteristics
  - Highly heterogenous
  - Lots of GRAPH and OPTIONAL
  - Some FILTER and ORDER BY
- Results
  - XML Format but not the protocol (for performance)

JSpace

- An extended mSpace clone
  - http://clarkparsia.com/jspace
  - mSpace developed at U. of Southampton
  - “Google meets iTunes”
  - http://www.mspace.fm/
- Selections drive query building
  - Each column selection instantiates a variable and adds some conjuncts
  - One can browse intermediate results

Built by ClarkParsia, LLC.

POPS (a JSpace app)

- Expertise location service for NASA
  - NASA has lots of idiosyncratic problems/systems
  - Roladex culture
  - Serendipity is key
- Federates 4 diverse data sources
  - 4.5M triples
  - Most queries are built by browsing
  - Fixed queries for info pane and socnet
- Pilot for Office of the Chief Engineer
  - Production will see 10,000 users

Built by Clark & Parsia, LLC.

BIANCA

- Network Asset Management Service
  - Integrated view of applications, servers, networks, and changes, and their relations
  - Supports interruption analysis
  - Sensitive data, so few users (~50) but high impact
  - One of the first deployed SemWeb Apps at NASA
- Tech details
  - 100,000 triples
  - 6-8 sorts of queries
    - Classification tree, instance retrieval, graph building

Built by Clark & Parsia, LLC.

HCLS demos

- Health Care and Life Sciences Interest Group
  - Organized by W3C; about 60 members
  - “chartered to develop and support the use of Semantic Web technologies and practices to improve **collaboration**, **research and development**, and **innovation adoption** in the [of HCLS] domains”
- Demo for WWW
  - Google Maps based interface for Allen Brain Atlas
  - 20,000 genes, 400000 images
  - Scraped 80,000 web pages to RDF

Allen Brain Atlas

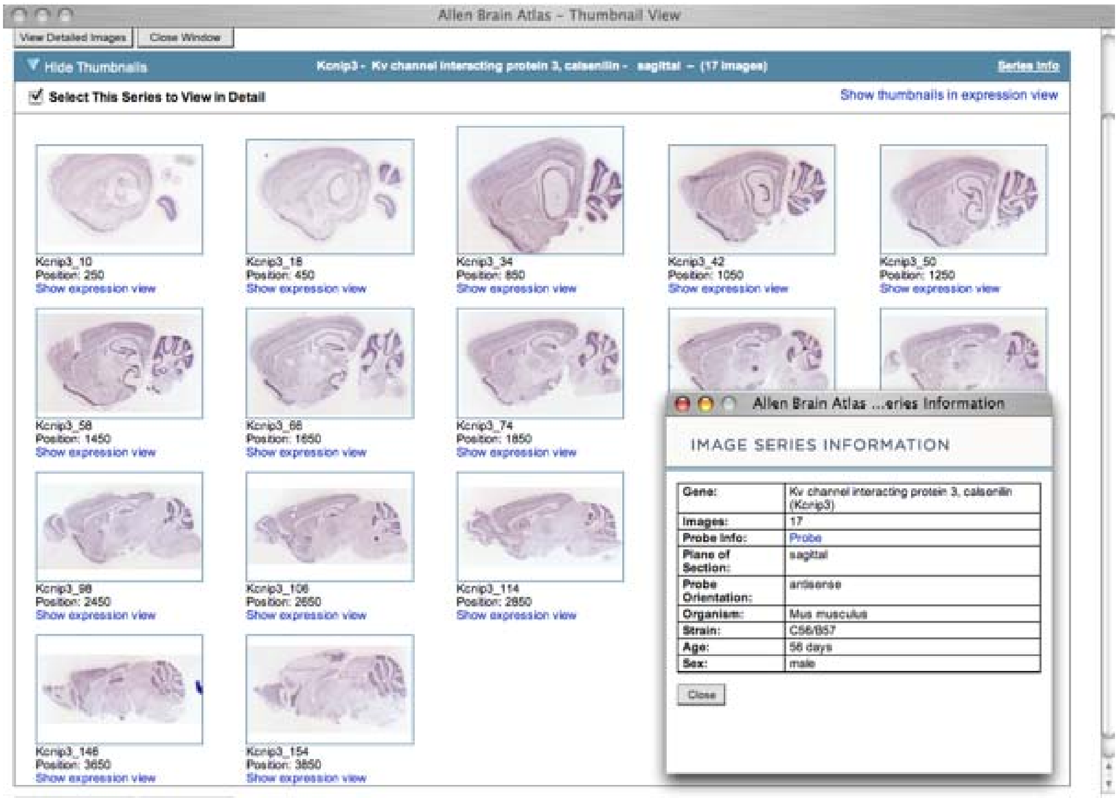
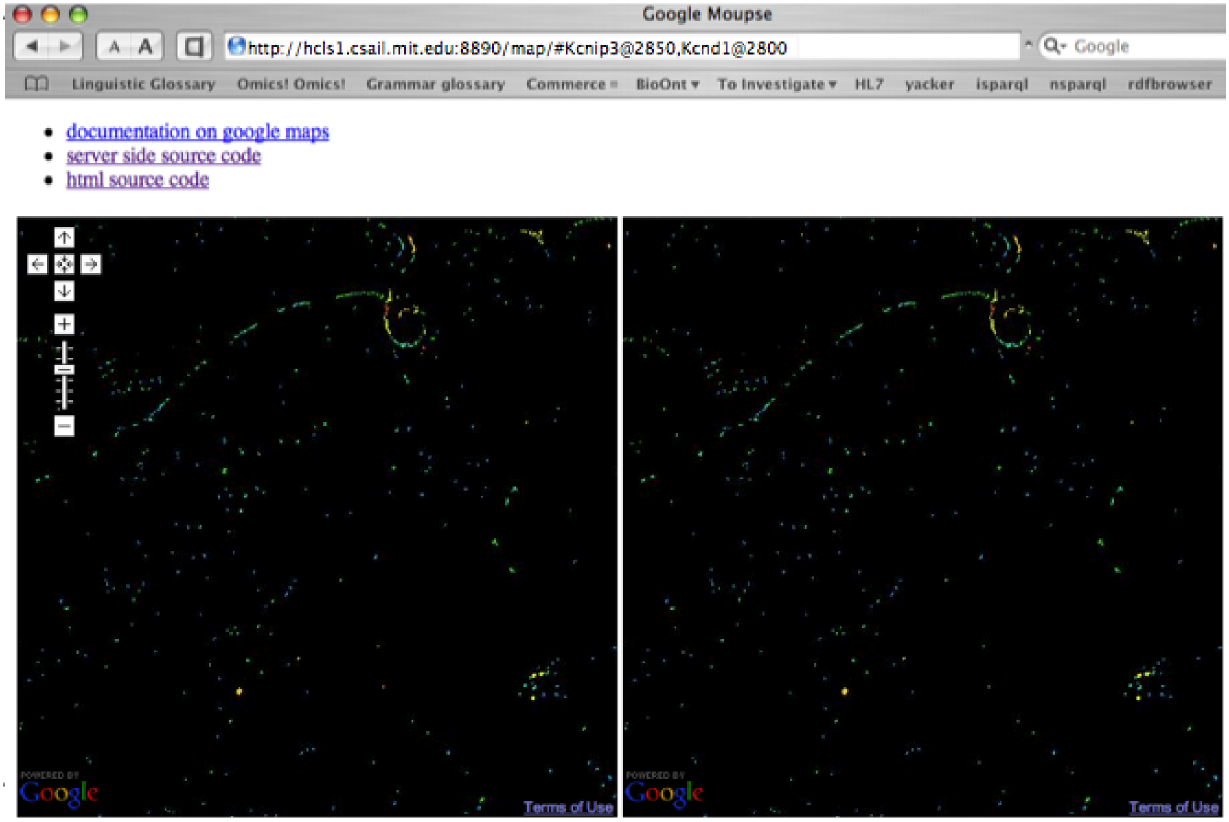


IMAGE SERIES INFORMATION

Gene:	Kv channel interacting protein 3, calsenilin (Kcnip3)
Images:	17
Probe Info:	Probe
Plane of Section:	sagittal
Probe Orientation:	antisense
Organism:	Mus musculus
Strain:	C56/B57
Age:	56 days
Sex:	male

Slide from Alan Ruttenberg
http://tinyurl.com/ysqm3z

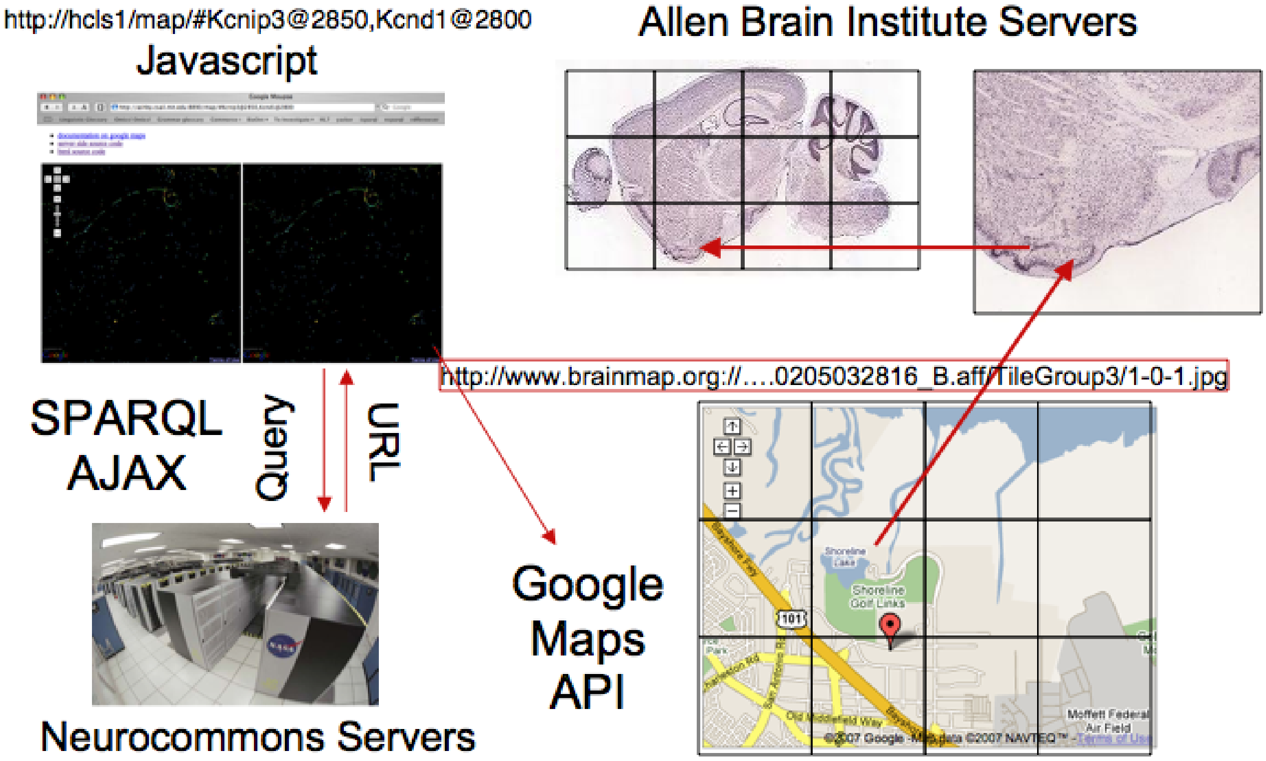
Google Maps/SPARQL/Allen Brain Atlas



- documentation on google maps
- server side source code
- html source code

Slide from Alan Ruttenberg
http://tinyurl.com/ysqm3z

Architecture



http://hcls1/map/#Kcnip3@2850,Kcnd1@2800

Javascript

Allen Brain Institute Servers

http://www.brainmap.org://….0205032816_B.aff/TileGroup3/1-0-1.jpg

SPARQL
AJAX

Query

URL

Google
Maps
API

Neurocommons Servers

Slide from Alan Ruttenberg
http://tinyurl.com/ysqm3z

Thanks

- To Steve Harris for Garlik.com info
- To Kendall Clark and Andy Schain for POPS/BIANCA details
  - See Kendall's seminal article: SPARQL: Web 2.0 Meet the Semantic Web
- To Mike Grove and Mike Smith for JSpace demo set up
- To Alan Ruttenberg for HCLS slides

European Semantic Web Conference 2007
Tutorial

SPARQL – Where are we?
Current state, theory and practice

Unit 4: SPARQL Foundations

RDF and SPARQL: Database Foundations

Marcelo Arenas, Claudio Gutierrez, Jorge Pérez

Department of Computer Science
Pontificia Universidad Católica de Chile
Universidad de Chile

Center for Web Research
http://www.cwr.cl/

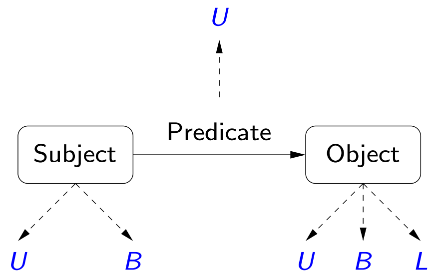
Outline

- Part I: Querying RDF Data
    - The RDF data model
    - Querying: The simple and the ideal
    - Querying: Semantics and Complexity
- Part II: Querying Data with SPARQL
    - Decisions taken
    - Decisions to be taken
- Conclusions

RDF in a nutshell

- RDF is the W3C proposal framework for representing information in the Web.
- Abstract syntax based on directed labeled graph.
- Schema definition language (RDFS): Define new vocabulary (typing, inheritance of classes and properties).
- Extensible URI-based vocabulary.
- Support use of XML schema datatypes.
- Formal semantics.

RDF formal model



U = set of Uris

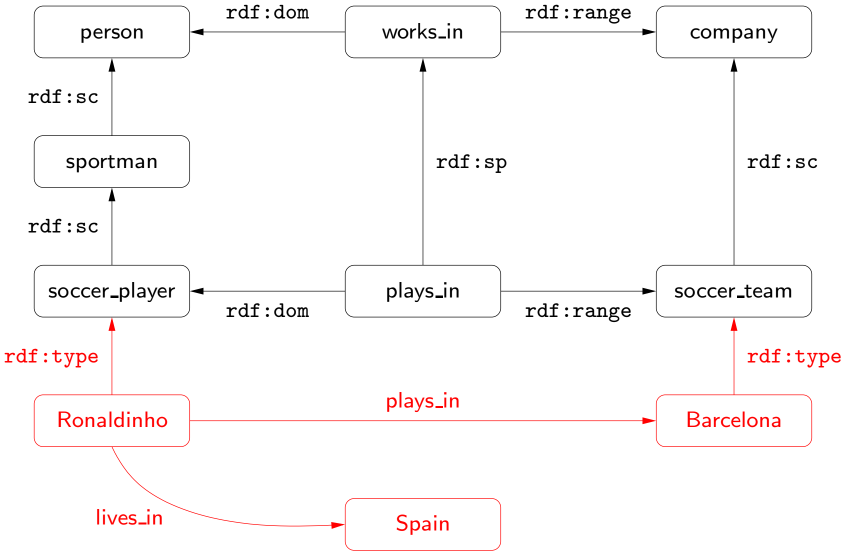
B = set of Blank nodes

L = set of Literals

$(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple

A set of RDF triples is called an RDF graph

RDFS: An example



RDF model

Some difficulties:

- Existential variables as datavalues
- Built-in vocabulary with fixed semantics (RDFS)
- Graph model where nodes may also be edge labels

RDF data processing can take advantage of database techniques:

- Query processing
- Storing
- Indexing

Entailment of RDF graphs

Entailment of RDF graphs:

- Can be defined in terms of classical notions such model, interpretation, etc
  - As for the case of first order logic
- Has a graph characterization via homomorphisms.

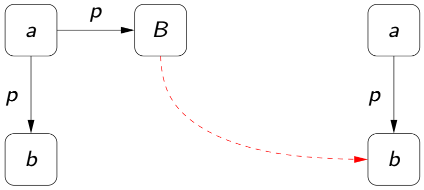
Homomorphism

A function $h : U \cup B \cup L \to U \cup B \cup L$ is a homomorphism h from G_1 to G_2 if:

- $h(c) = c$ for every $c \in U \cup L$;
- for every $(a, b, c) \in G_1$, $(h(a), h(b), h(c)) \in G_2$

Notation: $G_1 \to G_2$

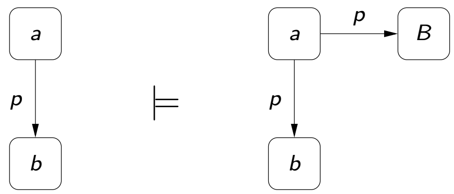
Example: $h = \{B \mapsto b\}$



Entailment

Theorem (CM77)

$G_1 \models G_2$ if and only if there is a homomorphism $G_2 \rightarrow G_1$.

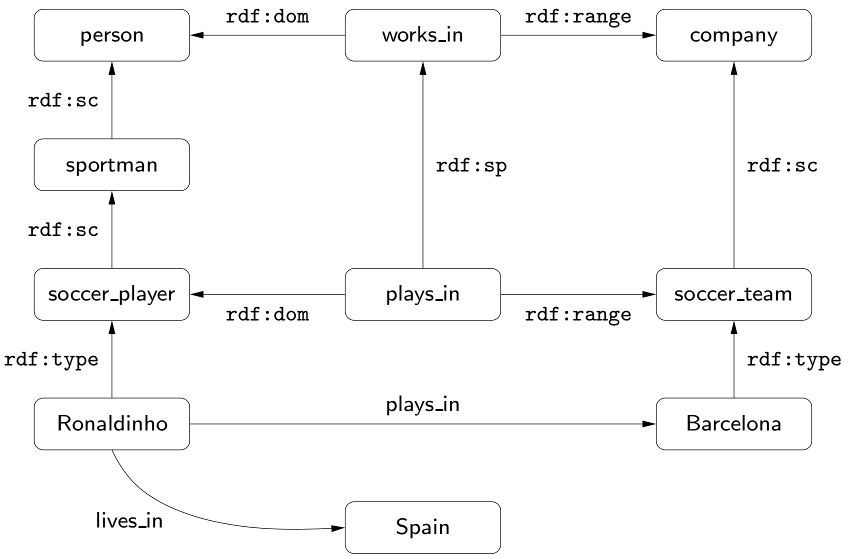


Complexity

Entailment for RDF is NP-complete

Graphs with RDFS vocabulary

Previous characterization of entailment is not enough to deal with RDFS vocabulary: $(\texttt{Ronaldinho}, \texttt{rdf}:\texttt{type}, \texttt{person})$



Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

`rdf:sc`: transitive

`rdf:sp`: transitive

More complicated interactions: $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

RDFS-entailment can be characterized by a set of rules

- An Existential rule
- Subproperty rules
- Subclass rules
- Typing rules
- Implicit typing

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

Existential rule : $\dfrac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\dfrac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules : $\dfrac{(a, \texttt{rdf:sc}, b) \quad (b, \texttt{rdf:sc}, c)}{(a, \texttt{rdf:sc}, c)}$

Typing rules : $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing : $\dfrac{(q, \texttt{rdf:dom}, a) \quad (p, \texttt{rdf:sp}, q) \quad (b, p, c)}{(b, \texttt{rdf:type}, a)}$

RDFS Entailment

Theorem (H04,GHM04,MPG07)

$G_1 \models G_2$ iff there is a proof of G_2 from G_1 using the system of 14 inference rules.

Complexity

RDFS-entailment is NP-complete.

Proof idea

Membership in NP: If $G_1 \models G_2$, then there exists a polynomial-size proof of this fact.

Closure of an RDF Graph

Notation:

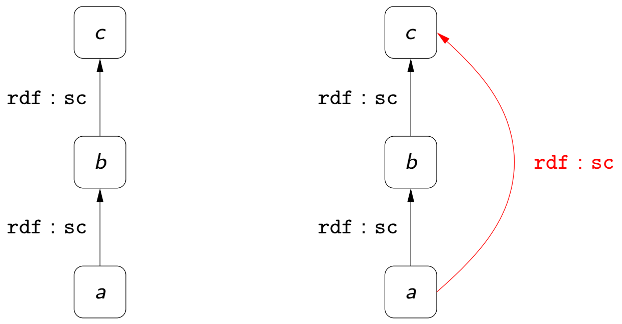
ground(G) : Graph obtained by replacing every blank B in G by a constant c_B.

ground^{-1}(G) : Graph obtained by replacing every constant c_B in G by B.

Closure of an RDF graph G (denoted by closure(G)):

$$G \cup \{t \in (U \cup B) \times U \times (U \cup B \cup L) \mid \text{there exists a ground tuple } t' \text{ such that } \text{ground}(G) \models t' \text{ and } t = \text{ground}^{-1}(t')\}$$

Closure of an RDF Graph: Example



Closure of an RDF graph: complexity

Proposition (H04,GHM04,MPG07)

$G_1 \models G_2$ *iff* $G_2 \rightarrow closure(G_1)$

Complexity

The closure of G can be computed in time $O(|G|^4 \cdot \log |G|)$.

Can the closure be used in practice?

- Can we use an alternative materialization?
- Can we materialize a small part of the closure?

Core of an RDF Graph

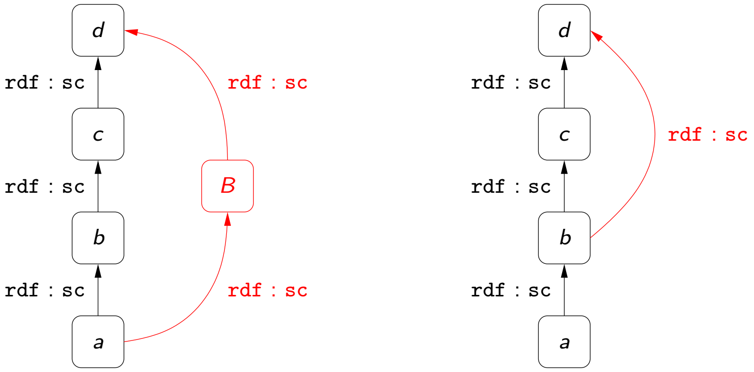
An RDF Graph G is a *core* if there is no homomorphism from G to a proper subgraph of it.

Theorem (HN92,FKP03,GHM04)

- *Each RDF graph G has a unique core (denoted by core(G)).*
- *Deciding if G is a core is coNP-complete.*
- *Deciding if $G = core(G')$ is DP-complete.*

Core and RDFS

For RDF graphs with RDFS vocabulary, the core of G may contain redundant information:



A normal form for RDF graphs

To reduce the size of the materialization, we can combine both core and closure.

- $\text{nf}(G) = \text{core}(\text{closure}(G))$

Theorem (GHM04)

- *G_1 is equivalent to G_2 iff $nf(G_1) \cong nf(G_2)$.*
- *$G_1 \models G_2$ iff $G_2 \rightarrow nf(G_1)$*

Complexity

The problem of deciding if $G_1 = nf(G_2)$ is DP-complete.

Querying RDF data: Desiderata

Let D be a database, Q a query, and $Q(D)$ the answer.

- Outputs should belong to the same family of objects as inputs
- If $D \equiv D'$, then $Q(D) = Q(D')$
  (Weaker) If $D \equiv D'$, then $Q(D) \cong Q(D')$
- $Q(D)$ should have no (or minimal) redundancies
- The framework should be extensible to RDFS
  (Should the framework be extensible to OWL?)
- Incorporate to the framework the notion of entailment

Querying RDF data: Desiderata

Outputs should belong to the same family of objects as inputs

- Allows compositionality of queries
- Allows defining views
- Allows rewriting

In RDF, the natural objects of input/output are RDF graphs.

Querying RDF data: Desiderata

If $D \equiv D'$, then $Q(D) = Q(D')$
(Weaker) If $D \equiv D'$, then $Q(D) \cong Q(D')$

- Outputs are syntactic or semantic objects?
- Need a notion of "equivalent" databases ($\equiv$)
  (In RDF, there is a standard notion of logical equivalence)
- One could just ask logical equivalence in the output
- In RDF there is an intermediate notion: graph isomorphism

Querying RDF data: Desiderata

$Q(D)$ should have no (or minimal) redundancies

- Desirable to avoid inconsistencies
- Desirable to improve processing time and space
- Standard requirement for exchange information

Querying RDF data: Desiderata

The framework should be extensible to RDFS
(Should the framework be extensible to OWL?)

- A basic requirement of the Semantic Web Architecture
- Extension to OWL are not trivial because of the known mismatch
- Not necessarily related to the type of semantics given (logical framework, graph matching, etc.)

Querying RDF data: Desiderata

Incorporate to the framework the notion of entailment

- RDF graphs are not purely syntactic objects
- Would like to incorporate KB framework
- Beware of the complexity issues! RDF navigates on the Web
- Find the good compromise

Querying RDF data: Definitions

A conjunctive query Q is a pair of RDF graphs H, B where some resources have been replaced by variables $\bar{X}, \bar{Y}$ in V.

$$Q : \quad H(\bar{X}) \leftarrow B(\bar{X}, \bar{Y})$$

Issues:

- Free variables in B (projection)
- Treatment of blank nodes in B
- Treatment of blank nodes in H

Querying RDF data: Definitions (cont.)

A valuation is a function $v : V \to U \cup B \cup L$

A matching of a graph B in the database D is a valuation v such that $v(B) \subseteq D$.

A pre-answer to Q over D is the set

$$preans(Q, D) = \{v(H) : \ v \text{ is a matching of } B \text{ in } D\ \}$$

A single answer is an element of $preans(Q, D)$

Querying RDF data: Two semantics

Union: answer $Q(D)$ is the union of all single answers

$$ans_U(Q, D) = \bigcup preans(Q, D)$$

Merge: answer $Q(D)$ is the merge of all single answers

$$ans_M(Q, D) = \biguplus preans(Q, D)$$

Proposition

1. *For both semantics, if* $D \models D'$ *then* $ans(Q, D') \models ans(Q, D)$
2. *For all* D, $ans_U(Q, D) \models ans_M(Q, D)$
3. *With merge semantics, we cannot represent the identity query*

Querying RDF data: refined semantics

Problem

Two non-isomorphic datasets D, D' give different answers to the same query.

A slightly refined semantics:

1. Normalize D before querying
2. Then query as usual over $nf(D)$

Good News: if $D \equiv D'$ then $Q(D) \cong Q(D')$
Bad News: computing $nf(D)$ is hard

Querying RDF data: refined semantics (cont.)

The news as formal results:

Theorem (MPG07)

Do not need to compute the normal form.

Theorem (FG06)

If a query language has the following two properties:

1. *for all Q, if $D \equiv D'$ then $Q(D) = Q(D')$,*
2. *can represent the identity query,*

then the complexity of evaluation is NP-hard (in data complexity).

Querying RDF data: Containment

A query Q contains a query Q, denoted $Q \sqsubseteq Q'$ iff $ans(Q, D)$ comprises all the information of $ans(Q', D)$.

In classical DB: $ans(Q, D) \subseteq ans(Q', D)$

In our setting we have two versions:

- $ans(Q', D) \subseteq ans(Q, D) \quad (Q \sqsubseteq_p Q')$
- $preans(Q, D) \subseteq preans(Q', D)$ (modulo iso) $(Q \sqsubseteq_m Q')$

For ground RDF both notions coincide.

Querying RDF data: Complexity

Query complexity version: The evaluation problem is NP-complete

Data complexity version: The evaluation problem is polynomial

Querying with SPARQL

- SPARQL is the W3C candidate recommendation query language for RDF.
- SPARQL is a graph-matching query language.
- A SPARQL query consists of three parts:
  - Pattern matching: optional, union, nesting, filtering.
  - Solution modifiers: projection, distinct, order, limit, offset.
  - Output part: construction of new triples, ...

Recall the formalization from Unit-2

Syntax:

- Triple patterns: RDF triple + variables (no bnodes)
- Operators between triple patterns: AND, UNION, OPT.
- Filtering of solutions: FILTER.
- A full parenthesized algebra.

Recall the formalization from Unit-2

Semantics:

- Based on mappings, partial functions from variables to terms.
- A mapping μ is a solution of triple pattern t in G iff
  - $\mu(t) \in G$
  - $\text{dom}(\mu) = \text{var}(t)$.
- $[\![t]\!]_G$ is the evaluation of t in G, the set of solutions.

Example

G

$(R_1$, name, john$)$
$(R_1$, email, J@ed.ex$)$
$(R_2$, name, paul$)$

t

$(?X$, name, $?Y)$

$[\![t]\!]_G$

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

Compatible mappings

Definition

Two mappings are compatible if they agree in their shared variables.

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

- μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \smallsetminus M_2$

- mappings in M_1 that cannot be extended with mappings in M_2

Union: $M_1 \cup M_2$

- mappings in M_1 plus mappings in M_2 (set theoretical union)

Definition

Left Outer Join: $M_1 \mathbin{⟕} M_2 = (M_1 \bowtie M_2) \cup (M_1 \smallsetminus M_2)$

Semantics of general graph patterns

Definition

Given a graph G the evaluation of a pattern is recursively defined

- $[\![(P_1 \text{ AND } P_2)]\!]_G = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$
- $[\![(P_1 \text{ UNION } P_2)]\!]_G = [\![P_1]\!]_G \cup [\![P_2]\!]_G$
- $[\![(P_1 \text{ OPT } P_2)]\!]_G = [\![P_1]\!]_G \mathbin{⟕} [\![P_2]\!]_G$
- $[\![(P \text{ FILTER } R)]\!]_G = \{\mu \in [\![P]\!]_G \mid \mu \text{ satisfies } R\}$

Differences with Relational Algebra / SQL

- Not a fixed output schema
  - mappings instead of tables
  - schema is implicit in the domain of mappings
- Too many NULLs
  - mappings with disjoint domains can be joined
  - mappings with distinct domains in output solutions
- SPARQL-to-SQL translations experience this issues
  - need of `IS NULL`/`IS NOT NULL` in join/outerjoin conditions
  - need of `COALESCE` in constructing output schema

SPARQL complexity: the evaluation problem

Input:

A mapping μ, a graph pattern P, and an RDF graph G.

Question:

Is the mapping in the evaluation of the pattern against the graph?

$$\mu \in [\![P]\!]_G?$$

Evaluation of AND-FILTER patterns is polynomial.

Theorem (PAG06)

For patterns using only AND and FILTER operators, the evaluation problem is polynomial:

$$O(|P| \times |G|).$$

Proof idea

- *Check that the mapping makes every triple to match.*
- *Then check that the mapping satisfies the* FILTER*s.*

Evaluation including UNION is NP-complete.

Theorem (PAG06)

For patterns using AND, FILTER and UNION operators, the evaluation problem is NP-complete.

Proof idea

- *Reduction from 3SAT.*
- *A pattern encodes the propositional formula.*
- $\neg$ bound *is used to encode negation.*

Evaluation including OPT is PSPACE-complete.

Theorem (PAG06)

For patterns using AND, FILTER and OPT operators, the evaluation problem is PSPACE-complete.

Proof idea

- *Reduction from QBF*
- *A pattern encodes a quantified propositional formula:*

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \psi.$$

- *nested OPTs are used to encode quantifier alternation.*

  (This time, we do not need $\neg$ bound.*)*

PSPACE-hardness: A closer look

Assume $\varphi = \forall x_1 \exists y_1\, \psi$, where $\psi = (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1)$.

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

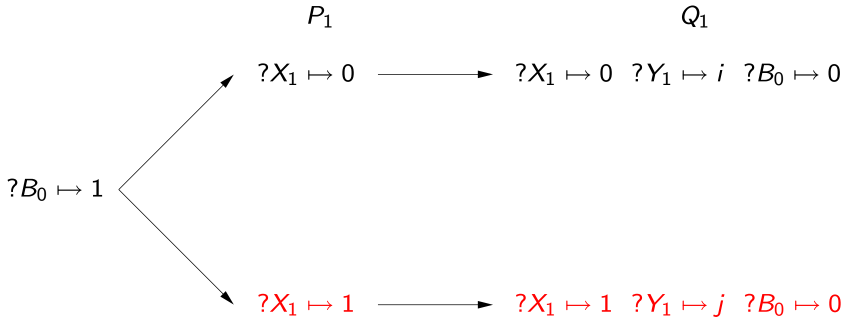
$$G \;:\; \{(a, \texttt{tv}, 0),\ (a, \texttt{tv}, 1),\ (a, \texttt{false}, 0),\ (a, \texttt{true}, 1)\}$$

$$\begin{aligned} P_\psi \;&:\; ((a, \texttt{tv}, ?X_1)\ \text{AND}\ (a, \texttt{tv}, ?Y_1))\ \text{FILTER} \\ &\;\; ((?X_1 = 1 \vee ?Y_1 = 0) \wedge (?X_1 = 0 \vee ?Y_1 = 1)) \\ P_\varphi \;&:\; (a, \texttt{true}, ?B_0)\ \text{OPT}\ (P_1\ \text{OPT}\ (Q_1\ \text{AND}\ P_\psi)) \end{aligned}$$

$$\mu_0 \;:\; \{?B_0 \mapsto 1\}$$

PSPACE-hardness: A closer look

$$\begin{array}{lcl} P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\ P_1 & : & (a, \texttt{tv}, ?X_1) \\ Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0) \end{array}$$



P_1 Q_1

$?X_1 \mapsto 0$ → $?X_1 \mapsto 0 \;\; ?Y_1 \mapsto i \;\; ?B_0 \mapsto 0$

$?B_0 \mapsto 1$

$?X_1 \mapsto 1$ → $?X_1 \mapsto 1 \;\; ?Y_1 \mapsto j \;\; ?B_0 \mapsto 0$

Data–complexity is polynomial

Theorem (PAG06)

When patterns are consider to be fixed (data complexity), the evaluation problem is in LOGSPACE.

Proof idea

From data–complexity of first–order logic.

SPARQL reordering/optimization: a simple normal from

- AND and UNION are commutative and associative.
- AND, OPT, and FILTER distribute over UNION.

Theorem (UNION Normal Form)

Every graph pattern is equivalent to one of the form

$$P_1 \;\text{UNION}\; P_2 \;\text{UNION}\; \cdots \;\text{UNION}\; P_n$$

where each P_i is UNION–free.

We concentrate in UNION-free patterns.

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

$$(\underset{\uparrow}{\cdots\cdots\cdots\cdots} \quad (\quad \underset{\uparrow}{A} \quad \text{OPT} \quad \underset{\uparrow}{B} \quad) \quad \underset{\uparrow}{\cdots\cdots\cdots\cdots})$$

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example

$$\Big(\big((\underset{\times}{?Y}, \text{name}, \text{paul}) \;\text{OPT}\; (\underset{\uparrow}{?X}, \text{email}, ?Z) \big) \;\;\text{AND}\;\; (\underset{\uparrow}{?X}, \text{name}, \text{john}) \Big)$$

Well–designed patterns and PSPACE-hardness

In the PSPACE-hardness reduction we use this formula:

$$\begin{array}{lcl} P_\varphi & : & (a, \texttt{true}, ?B_0)\ \text{OPT}\ (P_1\ \text{OPT}\ (Q_1\ \text{AND}\ P_\psi)) \\ P_1 & : & (a, \texttt{tv}, ?X_1) \\ Q_1 & : & (a, \texttt{tv}, ?X_1)\ \text{AND}\ (a, \texttt{tv}, ?Y_1)\ \text{AND}\ (a, \texttt{false}, ?B_0) \end{array}$$

It is not well-designed: B_0

Well–designed patterns: reordering/optimization

For well-designed patterns

- P_1 AND $(P_2$ OPT $P_3) \equiv (P_1$ AND $P_2)$ OPT P_3
- $(P_1$ OPT $P_2)$ OPT $P_3 \equiv (P_1$ OPT $P_3)$ OPT P_2

Theorem (OPT Normal Form)

Every well–designed pattern is equivalent to one of the form

$$(\cdots(\, t_1\ \text{AND}\ \cdots\ \text{AND}\ t_k\,)\ \text{OPT}\ O_1\,)\cdots)\ \text{OPT}\ O_n\,)$$

where each t_i is a triple pattern, and each O_j is a pattern of the same form.

Final remarks

- RDFS can be considered a new data model.
    - It is the W3C's recommendation for describing Web metadata.
- RDFS can definitely benefit from database technology.
    - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).
    - SPARQL: Formal semantics, complexity of query evaluation, query optimization.
    - Updating
    - . . .

References

- A. Chandra, P. Merlin, *Optimal Implementation of Conjunctive Queries in Relational Databases*. In *STOC* 1977.
- R. Fagin, P. Kolaitis, L. Popa, *Data Exchange: Getting to the Core*. In *PODS* 2003.
- C. Gutierrez, C. Hurtado, A. O. Mendelzon, *Foundations of Semantic Web Databases*. In *PODS* 2004.
- P. Hayes, *RDF Semantics*. W3C Recommendation 2004.
- P. Hell, J. Nesetril, *The Core of a Graph*. *Discrete Mathematics* 1992.
- S. Muñoz, J. Pérez, C. Gutierrez, *Minimal Deductive Systems for RDF*. In *ESWC* 2007.
- J. Pérez, M. Arenas, C. Gutierrez, *Semantics and Complexity of SPARQL*. In *ISWC* 2006.

European Semantic Web Conference 2007
Tutorial

**SPARQL – Where are we?
Current state, theory and practice**

Unit 5: SPARQL and its neighbour components in the Semantic Web stack

SPARQL and the Rules Layer

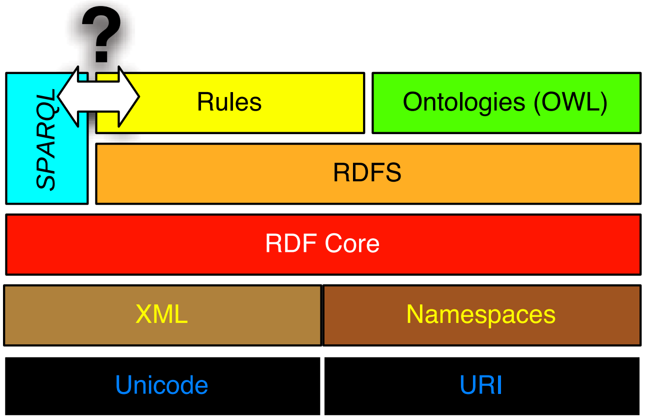
Axel Polleres[1]

[1]DERI Galway, National University of Ireland, Galway
axel.polleres@deri.org

European Semantic Web Conference 2007

Outline

Back to the layer cake...



How does SPARQL relate to Rules?

Rules for/on the Web: Where are we?

- Several proposals for systems and rules languages on the Web usable on top of RDF/RDFS:
  - TRIPLE [Decker et al., 2005]
  - N3 [Berners-Lee et al., 2005]
  - dlvhex [Eiter et al., 2005]
  - SWI-Prolog's semweb library [Wielemaker,]
  - SWRL [Horrocks et al., 2004]
  - SWSL Rules [Battle et al., 2005]
  - WRL, WSML [Angele et al., 2005, de Bruijn et al., 2005]
- RIF working group chartered in Dec 2005 to provide common interchange format (sic! Not a rule language) for the Web:
  - Is currently producing first concrete results and first draft format, in the future likely a common format for the approaches above
  - apart from deductive rules also concerned with other "rules": business rules, ECA rules, (integrity) constraints

Outline

SPARQL and LP 1/2

- Starting point: SQL can (to a large extent) be encoded in LP with *negation as failure* (=Datalog$^{\text{not}}$)

  Example: Two tables containing adressbooks
  `myAddr(Name, Street, City, Telephone)`
  `yourAddr(Name, Address)`

```
SELECT name FROM myAddr WHERW City = "Innsbruck"
  UNION
SELECT name FROM yourAddresses
```

```
answer1(Name) :- myAddr(Name, Street, "Innsbruck", Tel).
answer1(Name) :- yourAddr(Name, Address).
?- answer1(Name).
```

- That was easy... Now what about SPARQL?
- OPTIONAL and UNION probably cause some trouble, see Unit 4!

SPARQL and LP 2/2

We take as an example the language of dlvhex (`http://con.fusion.at/dlvhex/`):

- Prolog-like syntax
- We assume availability of built-in predicate `rdf[URL](S,P,O)` to import RDF data.

- dlvhex is implemented on top of the DLV engine (`http://www.dlvsystem.com/`)
- supports so-called answer set semantics (extension of the stable model semantics) for a language extending Datalog [Eiter et al., 2006].
- plugin-mechanism for easy integration of external function calls (built-in predicates).
- `rdf[URL](S,P,O)` is one such built-in to import RDF data.

The example translations in this Unit work similarly using e.g. SWI-Prolog's rdf_db module (see, `http://www.swi-prolog.org/packages/semweb.html`).

SPARQL and LP: Basic Graph Patterns

- We import all triples in a predicate `triple(Subj,Pred,Object,Graph)` which carries an additional argument for the dataset.
- For the import, we use the `rdf[URL](S,P,O)` built-in.

"select persons and their names"

```
SELECT ?X ?Y
FROM <http://alice.org>
FROM <http://ex.org/bob>
WHERE { ?X a foaf:Person . ?X foaf:name ?Y . }
```

```
triple(S,P,O,def) :- rdf["http://ex.org/bob"](S,P,O).
triple(S,P,O,def) :- rdf["http://alice.org"](S,P,O).
answer1(X,Y,def)  :- triple(X,"rdf:type","foaf:Person",def),
                     triple(X,"foaf:name",Y,def).

?- answer1(X,Y,def).
```

SPARQL and LP: GRAPH Patterns and NAMED graphs

"select creators of graphs and the persons they know"

```
SELECT ?X ?Y
FROM <alice.org>
FROM NAMED <alice.org>
FROM NAMED <ex.org/bob>
WHERE { ?G foaf:maker ?X .
         GRAPH ?G { ?X foaf:knows ?Y. } }
```

```
triple(S,P,O,def) :- rdf["alice.org"](S,P,O).
triple(S,P,O,"alice.org") :- rdf["alice.org"](S,P,O).
triple(S,P,O,"ex.org/bob") :- rdf["ex.org/bob"](S,P,O).
answer1(X,Y,def) :- triple(G,"foaf:maker",X,def),
                    triple(X,"foaf:knows",Y,G).
```

For legibility we left out the `http://` prefix

SPARQL and LP: UNION Patterns 1/2

UNIONs are split of into several rules:

"select Persons and their names **or** *nicknames"*

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

```
triple(S,P,O,def) :- ...
answer1(X,Y,def) :- triple(X,"foaf:name",Y,def).
answer1(X,Y,def) :- triple(X,"foaf:nick",Y,def).
```

SPARQL and LP: UNION Patterns 2/2

What if variables of the of constituent patterns don't coincide?
Slightly different than in SQL!
We emulate this by special `null` values!

```
SELECT ?X ?Y ?Z
FROM ...
WHERE { { ?X foaf:name ?Y . }
         UNION { ?X foaf:nick ?Z .} }
```

Data:
`<alice.org#me> foaf:name "Alice".`
`<ex.org/bob#me> foaf:name "Bob"; foaf:nick "Bobby".`
Result:

?X	?Y	?Z
<alice.org#me>	"Alice"	null
<ex.org/bob#me>	"Bob"	null
<ex.org/bob#me>	null	"Bobby"

SPARQL and LP: UNION Patterns 2/2

What if variables of the of constituent patterns don't coincide?
Slightly different than in SQL!
We emulate this by special `null` values!

```
SELECT ?X ?Y ?Z
FROM ...
WHERE { { ?X foaf:name ?Y . }
         UNION { ?X foaf:nick ?Z .} }
```

```
triple(S,P,O,def) :- ...
answer1(X,Y,null,def) :- triple(X,"foaf:name",Y,def).
answer1(X,null,Z,def) :- triple(X,"foaf:nick",Z,def).
```

SPARQL and LP: *OPTIONAL* Patterns 1/2

“select all persons and optionally their names”

```
SELECT *
WHERE
{
  ?X a foaf:Person .
  OPTIONAL {?X foaf:name ?N }
}
```

OPTIONAL is similar to an OUTER JOIN in SQL, actually it is a combination of a **join** and **set difference**:

$$\{P_1\ \texttt{OPTIONAL}\ \{P_2\}\}: \quad M_1 ⟕ M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2)$$

where M_1 and M_2 are variable binding for P_1 and P_2, resp.

SPARQL’s OPTIONAL has “negation as failure”, hidden:

- Observation: SPARQL allows to express set difference / negation as failure by combining OPT and !bound

“select all persons *without* an email address”

```
SELECT ?Name ?Email
WHERE
{
  ?X a ?Person
  OPTIONAL {?X :email ?Email }
  FILTER ( !bound( ?Email ) )
}
```

- Same effect as “NOT EXISTS” in SQL, set difference!.
- We’ve seen before that OPTIONAL, has set difference inherent, with the “!bound” we get it back again “purely”.

SPARQL and LP: OPT Patterns – First Try

```
SELECT *
WHERE
{
  ?X a foaf:Person .
  OPTIONAL {?X foaf:name ?N }
}
```

Recall: $(P_1 \text{ OPT } P_2)$: $M_1 \mathbin{⟕} M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2)$

```
triple(S,P,O,def) :- ...
answer1(X,N,def) :- triple(X,"rdf:type","foaf:Person",def),
                        triple(X,"foaf:name",N,def).
answer1(X,null,def) :- triple(X,"rdf:type","foaf:Person",def),
                          not answer2(X).
answer2(X) :- triple(X,"foaf:name",N,def).
```

We use `null` and negation as failure `not` to “emulate” set difference.

SPARQL and LP: OPT Patterns – Example

```
# Graph: ex.org/bob
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix bob: <ex.org/bob#> .

  <ex.org/bob> foaf:maker _:a.
  _:a a foaf:Person ; foaf:name "Bob";
           foaf:knows _:b.

  _:b a foaf:Person ; foaf:nick "Alice".
  <alice.org/> foaf:maker _:b
```

```
# Graph: alice.org

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix alice: <alice.org#> .

  alice:me a foaf:Person ; foaf:name "Alice" ;
           foaf:knows _:c.

  _:c  a foaf:Person ; foaf:name "Bob" ;
            foaf:nick "Bobby".
```

```
SELECT *
FROM <http://alice.org>
FROM <http://ex.org/bob>
WHERE { ?X a foaf:Person . OPTIONAL { ?X foaf:name ?N } }
```

Result:

?X	?N
_:a	"Bob"
_:b	null
_:c	"Bob"
alice.org#me	"Alice"

```
{ answer1("_:a","Bob",def), answer1("_:b",null, def),
  answer1("_:c","Bob",def), answer1("alice.org#me","Alice", def) }
```

SPARQL and LP: OPT Patterns – Nasty Example

Ask for pairs of persons ?X1, ?X2 who share the same name and nickname where both, name and nickname are optional:

```
SELECT *
FROM ...
WHERE { { ?X1 a foaf:Person . OPTIONAL { ?X1 foaf:name ?N } }
        { ?X2 a foaf:Person . OPTIONAL { ?X2 foaf:nick ?N } } }
```

?X1	?N
_:a	"Bob"
_:b	
_:c	"Bob"
alice.org#me	"Alice"

⋈

?X2	?N
_:a	
_:b	"Alice"
_:c	"Bobby"
alice.org#me	

Now this is strange, as we join over unbound variables.

Remark: this pattern is not well-designed, following Unit 4!

SPARQL and LP: OPT Patterns – With our translation?:

?X1	?N
_:a	"Bob"
_:b	null
_:c	"Bob"
alice.org#me	"Alice"

⋈

?X2	?N
_:a	null
_:b	"Alice"
_:c	"Bobby"
alice.org#me	null

=

?X1	?N	X2
_:b	null	_:a
_:b	null	alice.org#me
alice.org#me	"Alice"	_:b

What's wrong here? Join over `null` , as if it was a normal constant. Compared with SPARQL's notion of compatibility of mappings, this is too cautious!

SPARQL and LP: OPT Patterns – Correct Result:

?X1	?N
_:a	"Bob"
_:b	
_:c	"Bob"
alice.org#me	"Alice"

⋈

?X2	?N
_:a	
_:b	"Alice"
_:c	"Bobby"
alice.org#me	

=

?X1	?N	X2
_:a	"Bob"	_:a
_:a	"Bob"	alice.org#me
_:b		_:a
_:b	"Alice"	_:b
_:b	"Bobby"	_:c
_:b		alice.org#me
_:c	"Bob"	_:a
_:c	"Bob"	alice.org#me
alice.org#me	"Alice"	_:a
alice.org#me	"Alice"	_:b
alice.org#me	"Alice"	alice.org#me

SPARQL defines a very brave way of joins: unbound, i.e. `null` should join with anything!

Semantic variations of SPARQL

We could call these alternatives of treatment of possibly `null`-joining values alternative semantics for SPARQL:

- c-joining: cautiously joining semantics
- b-joining: bravely joining semantics (normative)

Which is the most intuitive? DAWG basically decided for b-join.

Now let's see to how to fix our translation to logic programs...

```
SELECT *
FROM ...
WHERE { { ?X1 a foaf:Person . OPTIONAL { ?X1 foaf:name ?N } }
        { ?X2 a foaf:Person . OPTIONAL { ?X2 foaf:nick ?N } } }
```

```
triple(S,P,O,def) :- rdf["ex.org/bob"](S,P,O).
triple(S,P,O,def) :- rdf["alice.org"](S,P,O).

answer1(N,X1,X2,def) :- answer2(N,X1,def), answer4(N,X2,def).

answer2(N,   X1,def) :- triple(X1,"a","Person",def),
                        triple(X1,"name",N,def).
answer2(null,X1,def) :- triple(X1,"a","Person",def),
                        not answer3(X1,def).
answer3(X1,def)      :- triple(X1,"name",N,def).

answer4(N,   X2,def) :- triple(X2,"a","Person",def),
                        triple(X2,"nick",N,def).
answer4(null,X2,def) :- triple(X2,"a","Person",def),
                        not answer5(X2,def).
answer5(X2,def)     :- triple(X2,"nick",N,def).
```

Here is the problem! Join over a *possibly* `null`-*joining variable*

SPARQL and LP: OPT Patterns – Improved!

How do I emulate b-joining Semantics? **Solution**:
We need to take care for variables which are joined and possibly unbound, due to the special notion of compatibility in SPARQL

```
triple(S,P,O,def) :- rdf["ex.org/bob"](S,P,O).
triple(S,P,O,def) :- rdf["alice.org"](S,P,O).

answer1(N,X1,X2,def) :- answer2(N,X1,def), answer4(N,X2,def).
answer1(N,X1,X2,def) :- answer2(N,X1,def), answer4(null,X2,def).
answer1(N,X1,X2,def) :- answer2(null,X1,def), answer4(N,X2,def).

answer2(N,   X1,def) :- triple(X1,"a","Person",def),
                        triple(X1,"name",N,def).
answer2(null,X1,def) :- triple(X1,"a","Person",def),
                        not answer3(X1,def).
answer3(X1,def)     :- triple(X1,"name",N,def).

answer4(N,   X2,def) :- triple(X2,"a","Person",def),
                        triple(X2,"nick",N,def).
answer4(null,X2,def) :- triple(X2,"a","Person",def),
                        not answer5(X2,def).
answer5(X2,def)     :- triple(X2,"nick",N,def).
```

SPARQL and LP: OPT Patterns

Attention:

- The "fix" we used to emulate b-joining semantics is potentially exponential in the number of possibly-null-joining variables.
- This is not surprising, since the complexity of OPTIONAL/UNION corner cases is PSPACE, see [Pérez et al., 2006].
- But: A slight modification of the translation (in the tech. report version of [Polleres, 2007]) shows that this translation is optimal: Non-recursive Datalog with negation as failure is also PSPACE complete!

From SPARQL to Rules . . . Summary!

- With these ingredients any SPARQL query Q can be translated recursively to a Datalog program P_q with a dedicated predicate $\texttt{answer1}_Q$ which contains exactly the answer substitutions for Q.
- The target language is non-recursive Datalog with neg. as failure
- Non-well-designed combinations of OPTIONAL and UNION are nasty and need special care: **Special treatment for the case where possibly `null` values are joined.**
- Prototype engine implemented and available at `http://con.fusion.at/dlvhex/`
- Full details of the translation in [Polleres, 2007].
- FILTERS not treated in detail, basically an implementation issue, needs a rules engine with support for external built-ins.
- In order to properly deal with the **multiset**-semantics of SPARQL, UNIONS and projections need special care!

Short DEMO:

`http://con.fusion.at/dlvhex/sparql-query-evaluation.php`

Outline

The SW Rules layer in a nutshell
- Rules for the Semantic Web

Translating SPARQL to LP style rules languages
- Basic Graph Patterns
- GRAPH Patterns
- UNION Patterns
- OPTIONAL and Negation as failure
- OPTIONAL and Negation as failure

Other Rules languages and formats
- SWI Prolog, TRIPLE, N3
- SPARQL and RIF

Other LP style languages

Similar considerations apply to other rule systems that allow to process RDF data, each of which has some syntactic peculiarities. We exemplify here:

- dlvhex
  - Done! SPARQL-plugin available.
- SWI-Prolog
  - similar... rdf_db module supports rdf/3, rdf/4 predicates, analogous to dlvhex rdf built-in.
- TRIPLE
- N3

TRIPLE

- RDF rules processor on top of XSB Prolog, developed by Michael Sintek, Stefan Decker.
- F-Logic style syntax, i.e. triple `S P O.` viewed as *F-Logic* molecule `S[P->O]`
- Special features: module mechanism.

Basic pattern SPARQL query "emulated" in TRIPLE:

```
@PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
SELECT ?X ?Y
FROM <http://alice.org>
FROM <http://ex.org/bob>
WHERE { ?X foaf:name ?Y .
         ?X a foaf:Person . }
```

```
foaf:= 'http://xmlns.com/foaf/0.1/'.
rdf:= 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'.
FORALL S,P,O S[P->O] <- S[P->O]@'http://alice.org' OR
                        S[P->O]@'http://ex.org/bob'.
```

N3

- RDF rules processor, CWM, implemented in python, developed by Dan Conolly, et al.
- N3 logic syntax, an extension of Turtle syntax.
- Special features: has negation as failure (log:notIncludes).
- Semantics... ? Probably perfect model semantics (i.e. only deals with stratified negation as failure)

Basic pattern SPARQL query “emulated” in N3:

```
@PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
SELECT ?X ?Y
FROM <http://alice.org>
FROM <http://ex.org/bob>
WHERE { ?X foaf:name ?Y .
         ?X a foaf:Person . }
```

```
{ <http://alice.org> log:semantics ?A.
  <http://ex.org/bob> log:semantics ?B.
  (?A ?B) log:conjunction ?C.
```

SPARQL and RIF

- RIF charter requires rules to deal with RDF data
- It is also written in the RIF charter that RIF should compatible to deal with SPARQL queries to access (external) datasets
- Both not yet addressed in WD1, first step:
  - Simple “webbish” Horn-style rules language (RIF Core)
  - Trouble: Has to address incompatibilities at lower levels... e.g.
    - URIs: Qnames in XML vs. RDF treatment of namespaces
    - compatibility with RDFS, OWL (not fully tackled in SPARQL even)
- Last but not least: SPARQL itself may be viewed as a rules language e.g. take the RDFS entailment rule (rdfs3) from [Hayes, 2004]

  If an RDF graph contains triples (`P rdfs:range C`) and (`S P O`) then the triple `O rdf:type C` is entailed.

```
CONSTRUCT {?O a ?C . }
WHERE { ?P rdfs:range ?C . ?S ?P ?O . }
```

References I

Angele, J. et al. (2005).
Web rule language (WRL).
W3C Member Submission, available from `http://www.w3.org/Submission/WRL/`.

Battle, S. et al. (2005).
Semantic web services framework (SWSF).
W3C Member Submission, available from `http://www.w3.org/Submission/SWSF/`.

Berners-Lee, T., Connolly, D., Prud'homeaux, E., and Scharf, Y. (2005).
Experience with n3 rules.
In *W3C Workshop on Rule Languages for Interoperability*, Washington, D.C., USA.

de Bruijn, J., Fensel, D., Keller, U., Lausen, M. K. H., Krummenacher, R., Polleres, A., and Predoiu, L. (2005).
Web Service Modeling Language (WSML).
W3C.
Member Submission. Available from `http://www.w3.org/Submission/WSML/`.

Decker, S. et al. (2005).
TRIPLE - an RDF rule language with context and use cases.
In *W3C Workshop on Rule Languages for Interoperability*, Washington, D.C., USA.

Eiter, T., Ianni, G., Polleres, A., and Schindlauer, R. (2006).
Answer set programming for the semantic web.
Tutorial at the European Semantic Web Conference (ESWC), see `http://asptut.gibbi.com/`.

Eiter, T., Ianni, G., Schindlauer, R., and Tompits, H. (2005).
A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming.
In *International Joint Conference on Artificial Intelligence (IJCAI) 2005*, pages 90–96, Edinburgh, UK.

References II

Hayes, P. (2004).
RDF semantics.
Technical report, W3C.
W3C Recommendation, `http://www.w3.org/TR/rdf-mt/`.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004).
SWRL: A semantic web rule language combining OWL and RuleML.
W3C Member Submission.

Pérez, J., Arenas, M., and Gutierrez, C. (2006).
Semantics and complexity of sparql.
Technical Report DB/0605124, arXiv:cs.

Polleres, A. (2007).
From SPARQL to rules (and back).
In *Proceedings of the 16th World Wide Web Conference (WWW2007)*, Banff, Canada.
Extended technical report version available at
`http://www.polleres.net/publications/GIA-TR-2006-11-28.pdf`.

Wielemaker, J.
SWI-Prolog Semantic Web Library.
available at `http://www.swi-prolog.org/packages/semweb.html`.

European Semantic Web Conference 2007
Tutorial

SPARQL – Where are we?
Current state, theory and practice

Unit 6: SPARQL Extensions and Outlook

SPARQL Extensions and Outlook

Axel Polleres[1]

[1]DERI Galway, National University of Ireland, Galway
axel.polleres@deri.org

European Semantic Web Conference 2007

Outline

- Translation to LP, a bit more formal
- Next steps? Some possible Examples
- Lessons to be learned from SQL?
  - Nested queries – Nesting ASK
  - Aggregates
- Lessons to be learned from Datalog, Rules Languages, etc. ?
  - Use SPARQL as rules
  - Mixing data and rules – Recursion?

Translation to LP, a bit more formal

Given a query $q = (V, P, DS)$, $DS = (G, G_N)$

SELECT V
FROM G
FROM NAMED G_N
WHERE P

we denote by Π_q the logic program obtained by the translation sketched in the previous Unit, where we give the auxiliary predicates non-ambiguous names, i.e. answeri_q.

Then, the extension of the predicate answer1_q contains all answer substitutions for q.

Example: $q_1 = ($ $\{?E, ?N\},$
$(((?X \ \ :name\ ?N)$ OPT $(?X \ \ :email\ ?E))),$
$(\{http://alice.org\}, \emptyset)$ $)$

```
SELECT ?N ?E
FROM <http://alice.org>
WHERE { ?X :name ?N
        OPTIONAL {?X :email ?E } }
```

$\Pi_{q_1} =$

triple(S,P,O,default$_{q_1}$) :- rdf["alice.org"](S,P,O).
answer1$_{q_1}$(E,N,default$_{q_1}$) :- triple(X,":name",N,default$_{q_1}$),
triple(X,":email",E,default$_{q_1}$).
answer1$_{q_1}$(null,N,default$_{q_1}$) :- triple(X,":name",N,default$_{q_1}$),
not answer2$_{q_1}$(X).
answer2$_{q_1}$(X) :- triple(X,":email",E,default$_{q_1}$).

More complex queries are decomposed recursively introducing more auxiliary predicates for nested sub-patterns: answer2$_q$, answer3$_q$, answer4$_{q_1}$, answer5$_{q_1}$, ...

Next steps?

Disclaimer: What follows in this unit is a speculative outlook and does not necessarily reflect the SPARQL working group's agenda. We discuss in this unit two starting points for such extensions:

- Lessons to be learned from SQL
- Lessons to be learned from Datalog

Both these partially overlap, and we will discuss how they integrate with the current SPARQL spec by using the translation from the previous unit.

Lessons to be learned from SQL: Nested ASK queries (1/2)

Nested queries are very common in SQL e.g.

```
SELECT ...FROM WHERE EXISTS ( SELECT ...
```

a simple and very useful extension for SPARQL could be nesting of boolean queries (ASK) in FILTERS:

SELECT ...FROM WHERE { P FILTER (ASK P_{ASK}) }

So, how could we implement e.g.

```
SELECT ?N
FROM <http://alice.org>
WHERE { ?X :name ?N
         FILTER (!(ASK {?X :email ?E }) ) }
```

Note that this give a more elegant solution for "set difference" queries avoiding the OPTIONAL/!bound combination!

Lessons to be learned from SQL: Nested ASK queries (2/2)

Given query $q = (V, P, DS)$, with sub-pattern
$(P_1$ FILTER (ASK $q_{\text{ASK}}))$ and $q_{\text{ASK}} = (\emptyset, P_{\text{ASK}}, DS_{\text{ASK}})$:

- modularly translate such sub-queries by extending Π_q with $\Pi_{q'}$ where $q' = (vars(P_1) \cap vars(P_{\text{ASK}}), P_{\text{ASK}}, DS_{\text{ASK}}))$
- let DS_{ASK} default to DS if not specified otherwise.

Example:

```
SELECT ?N
FROM <http://alice.org>
WHERE { ?X :name ?N
         FILTER ( !(ASK {?X :email ?E }) ) }
```

$vars(P_1) \cap vars(P_{\text{ASK}}) \;=\; \{X\}$
$q' = (\ \{?X\}, (?X : email?E), (\{http : //alice.org\}, \emptyset)\ )$

Π_q:
answer1$_{q'}$(X) :- triple(X,":email",E, default).
answer1$_q$(N) :- triple(X,":name",N, default), not answer1$_{q'}$(X).

Lessons to be learned from SQL: Aggregates (1/4)

Example Count:

```
SELECT ?X
FROM <http://example.org/lotsOfFOAFData.rdf>
WHERE { ?X a person .

         FILTER (
                  COUNT{ ?Y : ?X foaf:knows ?Y} > 3
                ) }
```

```
SELECT ?X
FROM <http://example.org/lotsOfFOAFData.rdf>
WHERE { ?X a person .
         ?X foaf:knows ?Y1 , ?Y2, ?Y3 .
         FILTER ( !( ?Y1 = ?Y2 ) AND
                  !( ?Y1 = ?Y3 ) AND
                  !( ?Y2 = ?Y3 ) ) }
```

Lessons to be learned from SQL: Aggregates (2/4)

Aggregates: A mockup syntax proposal:

- **Symbolic Set:** Expression

  $$\{Vars : Pattern\}$$

  of a list *Vars* of variables and a pattern *P* (e.g. `{ ?K : ?X foaf:knows ?K }`).
- **Aggregate Function:** Expression

  $$f\,\{Vars : Pattern\}$$

  where
  - $f \in \{COUNT, MIN, MAX, SUM, TIMES\}$, and
  - $\{Vars : Pattern\}$ is a symbolic set (e.g. `COUNT{ ?K : ?X foaf:knows ?K }`)

Lessons to be learned from SQL: Aggregates (3/4)

- **Aggregate Atom:** Expression

  $$\begin{array}{rll} Agg\_Atom ::= & val \ \circledcirc \ f\,\{Vars : Pattern\} \\ | & f\,\{Vars : Conj\} \ \circledcirc \ val \\ | & val_l \ \circledcirc_l \ f\,\{Vars : Pattern\} \ \circledcirc_r \ val_u \end{array}$$

  where
  - val, val_l, val_u are constants or variables,
  - $\circledcirc \in \{<, >, \leq, \geq, =\}$,
  - $\circledcirc_l, \circledcirc_r \in \{<, \leq\}$, and
  - $f\,\{Vars : Pattern\}$ is an aggregate function (e.g. `COUNT{ ?K : ?X foaf:knows ?K } }< 3`)

Lessons to be learned from SQL: Aggregates (4/4)

Examples of usage:

- Aggregate atoms in FILTERs:

```
SELECT ?X
WHERE { ?X a foaf:Person .
        FILTER ( COUNT{ ?K : ?X foaf:knows ?K } }< 3 )
```

- Aggregate atoms in result forms:

```
SELECT ?X COUNT{ ?K : ?X foaf:knows ?K } }
WHERE { ?X a foaf:Person .  )
```

Implementation:

- The aggregate syntax chosen here is a straight-forward extension of the aggregate syntax of DLV → implementation possible by a slight extension of the LP translation with DLV's aggregates.

Semantics:

- Semantics of Aggregates in LP, even possibly involving recursive rules agreed [Faber et al., 2004]

CONSTRUCT 1/3

CONSTRUCTs themselves may be viewed as rules over RDF.
How to handle CONSTRUCT in the outlined translation to LP?

```
CONSTRUCT { ?X foaf:name ?Y . ?X a foaf:Person .  }
WHERE { ?X vCard:FN ?Y }.
```

For blanknode-free CONSTRUCTs our translation can be simply extended:

```
triple(X,foaf:name,Y,constructed) :-
            triple(X,rdf:type,foaf:Person,default).
```

and export the RDF triples from predicate

`triple(S,P,O,constructed)`

in post-processing to get the constructed RDF graph

CONSTRUCT 2/3

More interesting: With this translation, we get for free a way to process mixed RDF and SPARQL CONSTRUCTs in ONE file.

Mock-up syntax, mixing TURTLE and SPARQL to describe implicit data or mappings within RDF[1]:

`foafWithImplicitdData.rdf`

```
:me a foaf:Person.
:me foaf:name "Axel Polleres".
CONSTRUCT{ :me foaf:knows ?X }
FROM <http://www.deri.ie/about/team>
WHERE { ?X a foaf:Person.  }
:me foaf:knows [foaf:name "Marcelo Arenas"],
               [foaf:name "Claudio Gutierrez"],
               [foaf:name "Bijan Parsia"],
               [foaf:name "Jorge Perez"],
               [foaf:name "Andy Seaborne"].
```

---

[1] see e.g. RIF use case 2.10, `http://www.w3.org/TR/rif-ucr/`

CONSTRUCT 3/3

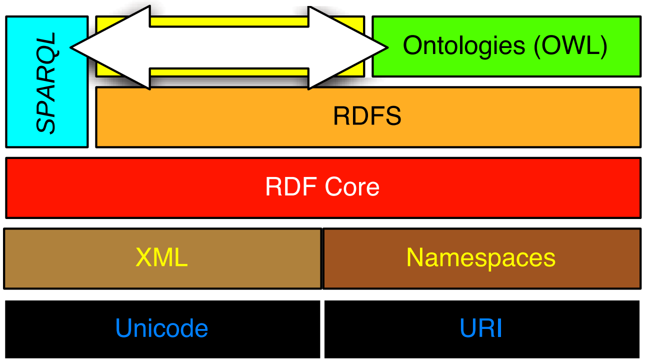
Attention! If you apply the translation to LP and two RDF+CONSTRUCT files refer mutually to each other, you might get a **recursive** program!

- even non-stratified negation as failure!
- two basic semantics for such "networked RDF graphs" possible:
  - well-founded [Schenk and Staab, 2007]
  - stable [Polleres, 2007]

etc., etc.

These were just some ideas for useful extensions!
More to come! Up to you!
Opens up interesting research directions!

Now let's get back to the next logical step. . .
. . . how to combine with OWL and RDFS?



As it turns out, not so simple! Bijan, the stage is yours!

References

Faber, W., Leone, N., and Pfeifer, G. (2004).
Recursive aggregates in disjunctive logic programs: Semantics and complexity.
In Alferes, J. J. and Leite, J., editors, *Proceedings of the 9th European Conference on Artificial Intelligence (JELIA 2004)*, number 3229 in Lecture Notes in AI (LNAI), pages 200–212. Springer Verlag.

Polleres, A. (2007).
From SPARQL to rules (and back).
In *Proceedings of the 16th World Wide Web Conference (WWW2007)*, Banff, Canada.
Extended technical report version available at
`http://www.polleres.net/publications/GIA-TR-2006-11-28.pdf`.

Schenk, S. and Staab, S. (2007).
Networked rdf graph networked rdf graphs.
Technical Report 3/2007, Universsity of Koblenz.
available at `http://www.uni-koblenz.de/~sschenk/publications/2006/ngtr.pdf`.

SPARQL Extensibility

- Arbitrary functions in FILTERs
  - Identified by URI
  - Can extend operators as well
- New semantics for Basic Graph Patterns
  - BGPs *extract* mappings from data sets
    - The algebra is independent of the extraction
      - We hope!
  - One document/graph has many semantics
    - Simple, RDF, RDFS, OWL....
  - Queries (should be) sensitive to the semantics
    - See prior unit for some examples (RDF, etc.)

Trickiness

- Controlling answers
  - Too many:
    - Simple entailment can yield infinite answers
  - Too few:
    - Finding all proofs of an answer difficult
- New sorts of issue
  - E.g., inconsistent data or equality
- Performance
  - Bare consistency of OWL DL is NEXPTIME
  - Query languages very expressive!
  - Performance model unclear

Standardizing these things is hard

- Not a lot of experience
  - Conjunctive query for DLs is (fairly) new
    - Theoretically and implementationally
  - Concept language expressive
    - Can express many common queries
  - Lots of decisions
- Database experience not always reliable
- LP experience not always reliable

Inconsistency

- Some logics can express **inconsistent** data
  - RDF (with certain datatypes), RDFS, and OWL
  - Inconsistencies entail **everything**
    - So, every mapping is a "**correct**" answer!
  - Inconsistencies often signal error
    - But may indicate mere disagreement!
- What should a query engine return?
  - Nothing
  - No answers, but explanations
  - Implementation dependent "best" answers
  - Answers from a weaker logic

“Strange” queries

- In RDF(S)
  - Thin distinction between schema and data
    - Schema language very inexpressive
  - So easy to treat the schema and data uniformly
- In OWL-DL
  - Strong distinction between schema and dat
    - TBox vs. Abox
    - Concept language very expressive
  - OWL Full tries to do the RDF thing
    - High cost: Undecidability, no implementations, hard to understand semantics

---

Types of Query Variables

- 2 key axis of a variable with 4 combinations

A)Distinguished
B)"Semi-"distinguished
C)"Projected away"
D)Non-distinguished

	In head of query	
Binds to names only	A. Yes/Yes	C. Yes/No
	B. No/Yes	D. No/No

- In a databases, only A and C are possible
  - C and D collapse (no non-named entities)
- In DLs, A and D are standard
  - D make query answering harder!
- In SPARQL/RDF, all variables are B

Query Variable Position

- "Conjunctive ABox queries"
  - Standard in DL systems: KAON2, Racer, Pellet
  - No variables in property or class positions
- "Higher order" queries
  - ?x rdf:type ?C. ?C rdfs:subClassOf ?D.
  - Careful restrictions make this feasible
- "Syntax reflective" qureies
  - ?s ?p ?o, where ?p can bind to rdf:type
  - ?x rdf:type [a Restriction; someValuesFrom ?C]
    - Only bind to asserted axioms? (essentially SPARQL/RDF)
- Latter two coming (see OWLED)

Counting

- What is a redundant answer?
  - BNodes in answers can be tricky
    - Might want to **lean** the answers
    - Distinguish between redundancy due to algebra and stated redundancy and inferred redundancy
  - Equality can be tricky
    - If two answers differ only by the value of one binding, and those values are inferred to be sameAs, how many answers?
    - No UNA in RDF-OWL
- We could count **answers** (instead of entities)
  - But then answers proliferate, often pointlessly

Information about the speakers

Marcelo Arenas, Department of Computer Science, Pontificia Universidad Católica de Chile.

Home Page: `http://www.ing.puc.cl/~marenas/`.

Short Bio: Prof. Marcelo Arenas received B.Sc. degrees in Mathematics (1997) and Computer Engineering (1998) and a M.Sc. degree in Computer Science (1998) from the Pontificia Universidad Católica de Chile, and a Ph.D. degree in Computer Science (2005) from the University of Toronto, Canada. In 2005, he joined the Computer Science Department at the Pontificia Universidad Católica de Chile as an Assistant Professor. His research interests are in different aspects of database theory, such as expressive power of query languages, database semantics, integrity constraints, inconsistency handling, database design, XML databases, data exchange and database aspects of the semantic web. Marcelo has received an IBM Ph.D. Fellowship (2004), three best paper awards (PODS 2003 in San Diego, California, PODS 2005 in Baltimore, Maryland and ISWC 2006 in Athens, Georgia) and an Honorable Mention Award in 2006 from the ACM Special Interest Group on Management of Data (SIGMOD) for his Ph.D dissertation, "Design Principles for XML Data."

Marcelo Arenas is supported by FONDECYT 1070732 and by Millennium Nucleus Center for Web Research, P04-067-F, Mideplan, Chile.

Claudio Gutierrez, Department of Computer Science, Universidad de Chile.

Home page: `http://www.dcc.uchile.cl/cgutierr/`.

Short Bio: Claudio Gutierrez received degrees in mathematics and mathematical logic from Universidad de Chile and Pontificia Universidad Católica de Chile, and a Ph.D. degree in computer science from Wesleyan University, U.S.A. Currently, he is associated professor in the Computer Science Department at the Universidad de Chile, and associated researcher at the Center for Web Research. His research interest lie in the intersection of databases and the Semantic Web. He has received Best Research Paper Awards at the European Semantic Web Conference in 2005, and at the International Semantic Web Conference in 2006.

Claudio Gutierrez is supported by FONDECYT 1070348 *RDF Databases* and by Millennium Nucleus Center for Web Research, P04-067-F, Mideplan, Chile.

Bijan Parsia, Information Management Group, School of Computer Science, University of Manchester, UK.

Home page: `http://homepages.manchester.ac.uk/~bparsia/`.

Short Bio: Bijan Parsia is a lecturer (since 2006) in the School of Computer Science at the University of Manchester, UK. He has published over 50 papers in such areas as description logic reasoning, explanation, trust, ontology editing, planning, web service composition, ontology partitioning, and ontology visualization. He has been a member of the WSDL, WS-Architecture, Data Access, and WS-Policy working groups.

Jorge Pérez, Department of Computer Science, Pontificia Universidad Católica de Chile.

Home Page: `http://www.ing.puc.cl/~jperez/`.

Short Bio: Jorge Pérez received B.Sc. degree in Computer Engineering and a M.Sc. degree in Computer Science from the Pontificia Universidad Católica de Chile. He is currently a Ph.D. student under the supervision of Prof. Marcelo Arenas. His research interests are primarily in database theory and the application of database technologies to the Web. Jorge has received the Best Research Paper Award at the 5th International Semantic Web Conference for work on SPARQL formalization from a database perspective.

The work of Jorge Pérez is supported by Dirección de Investigación – Universidad de Talca, and by Millennium Nucleus Center for Web Research, P04-067-F, Mideplan, Chile.

Axel Polleres, DERI, National University of Ireland, Galway.

Home page: `http://www.polleres.net`.

Short Bio: Axel Polleres obtained his PhD in Computer Science at the Vienna University of Technology in 2003. From 2003 to 2006 he worked at DERI at the Leopold-Franzens Universität Innsbruck in the areas of Semantic Web Services, Ontologies, Rules & Query Languages and Logic Programming. While there, he was involved in managing several EU projects. He worked for one year at Univ. Rey Juan Carlos, Madrid, under a "Juan de la Cierva" research award and joined DERI Galway in April 2007. Dr. Polleres has published more than 30 articles in journals, books, and conference and workshop contributions and has recently co-authored a book on Semantic Web Services. He has organised several international workshops in the areas of logic programming, Semantic Web, Semantic Web services and also ExpertFinding. He actively contributes to working groups such as WSMO, WSML, and the W3C Rule Interchange Format (RIF) WG.

The work of Axel Polleres was partly supported by the EU FP6 project inContext (IST-034718)[1] as well as by the Spanish MEC and Universidad Rey Juan Carlos under the project SWOS (URJC-CM-2006-CET-0300).

Andy Seaborne, Hewlett-Packard Laboratories.

Home page: `http://www.hpl.hp.com/people/afs/`.

Short Bio: Dr Andy Seaborne is a member of the Semantic Web Research Group in Hewlett-Packard Laboratories and he is based in Bristol, UK. He has been involved in RDF query languages since 2001, firstly with the development of RDQL for the Jena framework and latterly with the development of SPARQL. He is co-editor of the SPARQL query language specification. In addition, he has built two implementations of SPARQL, one, a reference implementation of SPARQL and one is a query engine that that is based on SQL.

---

[1] http://www.in-context.eu/