Semantic Web Modeling Languages

Lecture 4: Querying

Markus Krötzsch & Sebastian Rudolph

ESSLLI 2009 Bordeaux

Slides for remaining lectures at http://semantic-web-book.org/page/ESSLLI_2009

Query Languages for the Semantic Web?

How to *access* information that was specified in RDF or OWL?

- Querying information in RDF(S):
  Simple/RDF/RDFS entailment
  – "Can a certain RDF graph be derived from the given data?"
- Querying information in OWL:
  Logical entailment
  – "Can a subclass relation be derived from the ontology?"
  – "What are the instances of a given OWL class?"

Are OWL and RDF enough?

Even OWL (2) is too weak for many queries:

- “Who lives together with their parents?” (logical expressivity)
- “Who has married parents?” (logical expressivity)
- “Which properties connect two given individuals?” (schema-level query)
- “Which strings in the ontology are in French language?” (datatype expressivity)

Requirements for Query Languages

- Large query expressivity
- Well-specified semantics

But also:

- Tool support
  (trade-off with large expressivity)
- Result restriction and manipulation
- Result formatting options
- Machine-readable syntax for queries *and* results

Queries for RDF: SPARQL

SPARQL [sparkle]:

SPARQL Protocol And RDF Query Language

- W3C specification since Jan 15 2008
- Query language for data from RDF documents
- Extremely successful in practice
- Update in progress (new SPARQL WG since Feb'09)

Parts of the SPARQL specification:

- Query language: discussed here
- Result format: encode results in XML
- Query protocol: transmitting queries and results

Basic Queries

A simple example query:

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
WHERE
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  ?book ex:title  ?title .
  ?book ex:author ?author . }
```

- Main part is a **query pattern** (`WHERE`)
  - Patterns use RDF Turtle syntax
  - Variables can be used, even in predicate positions (`?variable`)
- **Abbreviations** for URIs (`PREFIX`)
- Query result based on **selected variables** (`SELECT`)

Query Results

A simple example document:

```
@prefix ex: <http://example.org/> .
ex:SemanticWeb
 ex:publishedBy  <http://crc-press.com/uri> ;
 ex:title "Foundations of Semantic Web Technologies" ;
 ex:author ex:Hitzler, ex:Krötzsch, ex:Rudolph .
```

Query results are **tables**, each row is one query result:

title	author
`“Foundations of ...”`	`http://example.org/Hitzler`
`“Foundations of ...”`	`http://example.org/Krötzsch`
`“Foundations of ...”`	`http://example.org/Rudolph`

Blank Nodes in SPARQL

What do bnodes in SPARQL mean?

Blank nodes in query patterns:

- Allowed as subject or object
- Behave like variables that cannot be selected
- bnode ID irrelevant – should not occur more than once per query

Blank nodes in query results:

- Placeholder for unknown elements
- ID arbitrary, but may indicate relations between results:

subject	value
_:a	"for"
_:b	"example"

subject	value
_:y	"for"
_:g	"example"

subject	value
_:z	"for"
_:z	"example"

Grouping Query Patterns

Simple graph patterns are grouped with **{ }**

Example:

```
{ { ?book ex:publishedBy <http://crc-press.com/uri> .
    ?book ex:title  ?title . }
  {}
  ?book ex:author ?author
}
```

→ Useful with additional query features

Optional Patterns

Optional parts can be specified with **`OPTIONAL`**

Example:

```
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  OPTIONAL { ?book ex:title  ?title . }
  OPTIONAL { ?book ex:author ?author . }
}
```

→ Parts of the result can be **unbound:**

book	title	author
`http://example.org/book1`	`“title 1”`	`http://example.org/johndoe`
`http://example.org/book2`	`“title 2”`	
`http://example.org/book3`	`“title 3”`	`_:a`
`http://example.org/book4`		

Alternative Patterns

Alternatives can be specified with **`UNION`**

Example:

```
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  { ?book ex:title  ?title . } UNION
  { ?book ex:author ?author . }
}
```

→ Result = union of results for one of the alternatives
→ Parts of the result can be **unbound**

Note: no interaction between multiple variable occurrences in alternative query parts

Combining `OPTIONAL` and `UNION`

What does the following mean?

```
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  { ?book ex:author ?author . } UNION
  { ?book ex:writer ?author . } OPTIONAL
  { ?autor ex:surname ?name . }
}
```

- Union of two patters, with an optional condition **or**
- Union of two patterns, the second of which includes an optional?

→ First interpretation is correct:

```
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  { { ?book ex:author ?author . } UNION
    { ?book ex:writer ?author . }
  } OPTIONAL { ?autor ex:surname ?name . }
}
```

Combining `OPTIONAL` and `UNION`

General rules:

- `OPTIONAL` refers to exactly one grouped pattern to the right
- `OPTIONAL` and `UNION` refer to all expressions to their left (left associative), and neither has precedence over the other

Example:

```
{ {s1 p1 o1} OPTIONAL {s2 p2 o2} UNION {s3 p3 o3}
             OPTIONAL {s4 p4 o4} OPTIONAL {s5 p5 o5}
}
```

means

```
{ { { { {s1 p1 o1} OPTIONAL {s2 p2 o2}
      } UNION {s3 p3 o3}
    } OPTIONAL {s4 p4 o4}
  } OPTIONAL {s5 p5 o5}
}
```

Filters

Additional "filter conditions" can be specified with **`FILTER`**

Example:

```
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  ?book ex:price  ?price .
  FILTER( (?price < 17) && !isBlank(?book) )
}
```

→ Filter condition: "price a number below 17 and book not a blank node"

→ Results that do not match the filter are removed

SPARQL provides many filter functions:

Comparisons (=, <, >, <=, >=, !=), arithmetics (+, −, *, /), Booleans (`&&`, `||`, `!`), RDF-specific functions (`isLiteral()`, `Lang()`, `BOUND()`, …)

SPARQL: Summary / More Features

- Based on matching simple graph patterns
- Grouping, optionals, and alternatives
- Filters: "extra-logical" result restrictions

Further features:

- Modifiers: postprocess query result set
  E.g.: `ORDER BY ?age LIMIT 10 OFFSET 5`
  (→ order by ?age and return 10 results, starting at result 5)
- Result formats: choose encoding of results
  E.g.: `SELECT ?name, ?age` (→ as in earlier examples)
  `CONSTRUCT {?name ex:hasAge ?age .}`
  (→ construct RDF graph as result)

Yet Another Semantics? Is That Really Necessary?

Informal meaning of queries leaves open questions:

- User: "Which results can I expect?"
- Developer: "How should my software behave?"
- Vendor: "Is my product SPARQL-conformant?"

… and, most importantly:

- Computer Scientist: "What's the worst-case complexity of SPARQL?"

Semantics of Query Languages (I)

Semantics for formal logic:

- Model theoretic semantics: Which interpretations satisfy a knowledge base?
- Proof theoretic semantics: What are valid derivations from a knowledge base?
- …

Semantic of programming languages:

- Axiomatic Semantics: Which logical statements are valid for a given program?
- Operational Semantics: What effects does the execution of a program have?
- Denotational semantics: How can a program be described as an abstract function?
- …

What to do with query languages?

Semantics of Query Languages (2)

- Query entailment
  - Queries as *descriptions* of valid results
  - Database as set of logical statements (*theory*)
  - Result as logical *conclusion*

Examples: OWL and RDF as query languages, deductive databases (datalog)

- Query algebra
  - Query as a *procedure* for calculating results
  - Database as *input*
  - Results as *output*

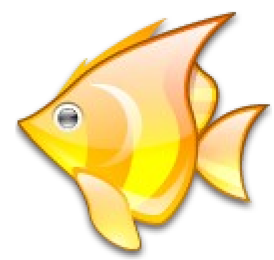
Examples: Relational algebra for SQL, **SPARQL algebra**

Mapping to SPARQL Algebra

```
{ ?book  ex:price  ?price .
  FILTER (?price < 15)
  OPTIONAL
    { ?book  ex:title  ?title . }
  { ?book  ex:author  ex:Shakespeare . } UNION
  { ?book  ex:author  ex:Marlowe . }
}
```

Semantics of a SPARQL query:

1) Translate query to algebraic expression
2) Evaluate algebraic expression



Mapping to SPARQL Algebra: *BGP*

```
{ BGP(?book  <http://eg.org/price>  ?price .)
  FILTER (?price < 15)
  OPTIONAL
    { BGP(?book  <http://eg.org/title>  ?title .) }
  { BGP(?book  <http://eg.org/author>
                    <http://eg.org/Shakespeare> .) }
  UNION
  { BGP(?book  <http://eg.org/author>
                        <http://eg.org/Marlowe> .) }
}
```

First step: **Replacing simple graph patterns**

- Operator *BGP*
- Also resolve URI abbreviations

Mapping to SPARQL Algebra: *Union*

```
{ BGP(?book  <http://eg.org/price>  ?price .)
  FILTER (?price < 15)
  OPTIONAL
    { BGP(?book  <http://eg.org/title>  ?title .) }
  Union({ BGP(?book  <http://eg.org/author>
                     <http://eg.org/Shakespeare> .) },
        { BGP(?book  <http://eg.org/author>
                         <http://eg.org/Marlowe> .) })
}
```

Second step: **Merging alternative graph patterns**

- Operator *Union*
- Refering to patterns next to **`UNION`** (binding stronger than conjunction!)
- Grouping of multiple alternatives as discussed earlier

Mapping to SPARQL Algebra

Join(P1,P2)	results P1 and P2 are combined conjunctively
Filter(F,P)	filter expression F is applied to result P
LeftJoin(P1,P2,F)	results of P1 are conjunctively combined with the results of P2 and the filter condition F is applied; results of P1 that are eliminated by this operation are directly added to the output
Z	Constant for the empty expression

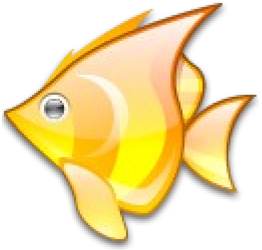
Remaining translation stepwise, from inside to outside:

(1) Select an innermost graph pattern *P*

(2) Remove all filter conditions from *P*
GF := conjunction of all filter conditions

(3) Initialise $G := Z$ and process all subexpressions *SE* of *P*:

- If *SE* = **`OPTIONAL`** *Filter*(*F*, *A*): $G := LeftJoin(G, A, F)$
- Otherwise, if *SE* = **`OPTIONAL`** *A*: $G := LeftJoin(G, A,$ **`true`**$)$
- Otherwise: $G := Join(G, SE)$



(4) If *GF* is not empty: $G := Filter(GF, G)$

Mapping to SPARQL Algebra: Joins

```
{ BGP(?book  <http://eg.org/price>  ?price .)
  FILTER (?price < 15)
  OPTIONAL
    { BGP(?book  <http://eg.org/title>  ?title .) }
  Union({ BGP(?book  <http://eg.org/author>
                         <http://eg.org/Shakespeare> .) },
        { BGP(?book  <http://eg.org/author>
                             <http://eg.org/Marlowe> .) })
}
```

Mapping to SPARQL Algebra: Joins

```
{ BGP(?book  <http://eg.org/price>  ?price .)
  FILTER (?price < 15)
  OPTIONAL
    Join(Z,BGP(?book  <http://eg.org/title>  ?title .) )
  Union( Join(Z,BGP(?book  <http://eg.org/author>
                      <http://eg.org/Shakespeare> .) ),
         Join(Z,BGP(?book  <http://eg.org/author>
                          <http://eg.org/Marlowe> .) ) )
}
```

Mapping to SPARQL Algebra: Joins

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    Join(Z,BGP(?book  <http://eg.org/title>  ?title .) ),
    true
  ),
  Union( Join(Z,BGP(?book  <http://eg.org/author>
                      <http://eg.org/Shakespeare> .) ),
         Join(Z,BGP(?book  <http://eg.org/author>
                          <http://eg.org/Marlowe> .) )
  )
 )
)
```

Defining the SPARQL operators

How are SPARQL algebra operators defined?

Output:

- "Result table"

Input:

- Queried RDF database
- Partial results from subexpressions
- Various parameters for certain operations

How should "results" be represented formally?

SPARQL Results

Intuition: results encode tables with variable assignments

- **Result:**
  List of solutions (solution sequence)
  → each solution corresponds to one table row
- **Solution:**
  Partial mapping (function)
  - Domain: set of selected variables
  - Range: URIs ∪ blank nodes ∪ RDF literals

  → unbound variables are those that are not assigned values in a solution

The Empty Expression Z

Which solution is represented by the empty expression Z?

- Domain: $\emptyset$ (no results selected)
- Solutions: exactly one (there is one and only one function with empty domain)

→ "Table with one row but no columns"

Calculating Basic Graph Patterns

A partial function μ is a **solution of the expression *BGP*(*P*)** (where P is a list of triples) if:

- Domain of μ is the set of variables in P
- By replacing bnodes with URIs, bnodes, or RDF literals, P can be transformed into a pattern P' such that:
  All triples in $\mu(P')$ occur in the input graph

Result of *BGP*(*P*):

List of all such solutions μ (order undefined)

Unions of Solutions

- Two solutions μ and μ' are **compatible** if for all x for which μ and μ' are defined we have $\mu(x) = \mu'(x)$
- **Union** of two compatible solutions μ and μ':
  - $(\mu \cup \mu')(x) = \mu(x)$ if μ is defined for x
  - $(\mu \cup \mu')(x) = \mu'(x)$ if μ' is defined for x
  - $(\mu \cup \mu')(x) = \mathit{undefined}$ otherwise

→ simple intuition: union of compatible table rows

(Attention: not related to `UNION` operator, see below)

Defining SPARQL Operators

Now we can define the essential operations:

- $\mathit{Filter}(F, \Psi) = \{\mu \mid \mu \in \Psi$ and the expression $\mu(F)$ evaluates to `true`$\}$
- $\mathit{Join}(\Psi 1, \Psi 2) = \{\mu 1 \cup \mu 2 \mid \mu 1 \in \Psi 1, \mu 2 \in \Psi 2,$ and $\mu 1$ compatible with $\mu 2\}$
- $\mathit{Union}(\Psi 1, \Psi 2) = \{\mu \mid \mu \in \Psi 1$ or $\mu \in \Psi 2\}$
- $\mathit{LeftJoin}(\Psi 1, \Psi 2, F) =$
  $\{\mu 1 \cup \mu 2 \mid \mu 1 \in \Psi 1, \mu 2 \in \Psi 2,$ and $\mu 1$ is compatible to $\mu 2$, and the expression $(\mu 1 \cup \mu 2)(F)$ evaluates to `true`$\}$ $\cup$
  $\{\mu 1 \mid \mu 1 \in \Psi 1$ and for all $\mu 2 \in \Psi 2$ we find that: either $\mu 1$ is not compatible with $\mu 2$ or $(\mu 1 \cup \mu 2)(F)$ is not `true`$\}$

Symbols: $\Psi, \Psi 1, \Psi 2$ results; $\mu, \mu 1, \mu 2$ solutions; F filter condition

Example

```
@prefix ex: <http://eg.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet         ex:author  ex:Shakespeare ;
                  ex:price   "10.50"^^xsd:decimal .
ex:Macbeth        ex:author  ex:Shakespeare .
ex:Tamburlaine    ex:author  ex:Marlowe ;
                  ex:price   "17"^^xsd:integer .
ex:DoctorFaustus ex:author  ex:Marlowe ;
                  ex:price   "12"^^xsd:integer ;
       ex:title "The Tragical History of Doctor Faustus" .
ex:RomeoJuliet    ex:author  ex:Brooke ;
                  ex:price   "9"^^xsd:integer .
```

```
{ ?book  ex:price  ?price . FILTER (?price < 15)
  OPTIONAL { ?book  ex:title  ?title . }
  { ?book  ex:author  ex:Shakespeare . } UNION
  { ?book  ex:author  ex:Marlowe . }  }
```

Example Calculation (I)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    Join(Z,BGP(?book  <http://eg.org/title>  ?title .) ),
    true ),
  Union( Join(Z,BGP(?book  <http://eg.org/author>
                            <http://eg.org/Shakespeare> .) ),
         Join(Z,BGP(?book  <http://eg.org/author>
                                <http://eg.org/Marlowe> .) ) )
 ) )
```

→ joins with Z do not affect result and can be removed

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

Example Calculation (3)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

book
ex:Tamburlaine
ex:DoctorFaustus

Example Calculation (4)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

book
ex:MacBeth
ex:Hamlet

Example Calculation (5)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

book
ex:Hamlet
ex:Tamburlaine
ex:DoctorFaustus
ex:MacBeth

Example Calculation (6)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

book	price
ex:Hamlet	"10.50"^^xsd:decimal
ex:Tamburlaine	"17"^^xsd:integer
ex:DoctorFaustus	"12"^^xsd:integer
ex:RomeoJuliet	"9"^^xsd:integer

Example Calculation (7)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

book	title
ex:DoctorFaustus	**"The Tragical ..."**

Example Calculation (8)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

book	price	title
ex:Hamlet	**"10.50"^^xsd:decimal**	
ex:Tamburlaine	**"17"^^xsd:integer**	
ex:DoctorFaustus	**"12"^^xsd:integer**	**"The Tragical ..."**
ex:RomeoJuliet	**"9"^^xsd:integer**	

Example Calculation (9)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

book	price	title
ex:Hamlet	"10.50"^^xsd:decimal	
ex:Tamburlaine	"17"^^xsd:integer	
ex:DoctorFaustus	"12"^^xsd:integer	"The Tragical ..."

Example Calculation (10)

```
Filter((?price < 15),
 Join(
  LeftJoin(
    BGP(?book  <http://eg.org/price>  ?price .),
    BGP(?book  <http://eg.org/title>  ?title .),
    true ),
  Union( BGP(?book  <http://eg.org/author>
                        <http://eg.org/Shakespeare> .),
         BGP(?book  <http://eg.org/author>
                            <http://eg.org/Marlowe> .) )
 ) )
```

book	price	title
ex:Hamlet	"10.50"^^xsd:decimal	
ex:DoctorFaustus	"12"^^xsd:integer	"The Tragical ..."

Hardness of SPARQL

Problem: “Given an input graph, a SPARQL query, and a potential solution, find out whether the solution is correct.”

How hard is this, computationally speaking?

- It's certainly as hard as *Sudoku* or *Minesweeper* (NP)*
- It's also as hard as *Reversi* or *Sokoban* (PSpace) (needs Union & Optional, but no Filters)
- But it's not as hard as *Draughts* or *Chess* (ExpTime)

SPARQL is indeed PSpace-complete
(recall: OWL 2 profiles→PTime, RDF(S) entailment→NP, OWL 2 DL→N2ExpTime)

Word of Warning: generalised games make nice memory hooks, but human intuition may still be very wrong about algorithmic hardness.

*) *Homework: find out how to solve a given Sudoku with SPARQL; try doing it without Filters, Unions, or Optional*

Negation in SPARQL

Can we query negative information with SPARQL?

- RDF knows no negation
- SPARQL has no negation operator, but inequality and `!` in filters
- Yet "negation as failure" can be implemented using `OPTIONAL`, Boolean negation, and `BOUND`:

```
PREFIX ex: <http://example.org/>
SELECT ?book
WHERE
{ ?book ex:publishedBy <http://crc-press.com/uri> .
  OPTIONAL { ?book ex:editor ?editor }
  FILTER( !BOUND(?editor) ) }
```

This is the only use for `BOUND`! (better syntax in next SPARQL?!)

SPARQL for OWL?

SPARQL fully defined only for plain RDF → what about RDFS and OWL?

- SPARQL with a model-theoretic twist:
  "Which graph patterns occur in all models?"
- Semantics of basic patters and unions clear (variables as subjects/objects only)
  → Closely related to "conjunctive queries"
- Semantics for optional patterns and filters not so clear
- Returning unnamed individuals as blank nodes could lead to infinite results in OWL – treat input bnodes like constants?
- Even if semantics is clear, it may not be implementable:
  Is SPARQL for OWL DL decidable? (open problem)
- Very high worst-case complexities expected (above 2ExpTime)

→ Current implementations support subsets of SPARQL for OWL DL

The Future of SPARQL

The new SPARQL working group will define a compatible extension of SPARQL.

Considered features:

- Extended query language:
  - Aggregate functions
  - Subqueries
  - Negation (i.e. a better syntax for `OPTIONAL` + `BOUND`)
- Service description (report service's query capabilities)
- Definition of semantics for RDF(S), OWL (hopefully)
- Further features as time permits …

Summary and Outlook

SPARQL: a query language for RDF

- W3C standard, widely used
- Queries based on graph patterns
- Various extensions (filters, modifiers, result formats)
- Semantics via translation to SPARQL algebra
- Extension to OWL possible but not trivial
- Update of SPARQL specification in progress

Further Reading

- P. Hitzler, S. Rudolph, M. Krötzsch: **Foundations of Semantic Web Technologies.** CRC Press, 2009. (Chapter 7 closely related to this lecture)
- E. Prud'hommeaux, A. Seaborne: **SPARQL query language for RDF.** See http://www.w3.org/TR/rdf-sparql-query/ W3C Recommendation, Jan 15 2008. (the official standard)

Selected research articles:

- M. Arenas, J. Perez, C. Gutierrez: **Semantics and Complexity of SPARQL.** Proc. 19th Int. Conf. On Semantic Web Conf. (ISWC 2006), Springer, 2006. (proof of worst-case complexity)
- B. Glimm, I. Horrocks, C. Lutz, U. Sattler: **Conjunctive Query Answering for the Description Logic SHIQ.** Prof. 20th Int. Conf. On Artificial Intelligence (IJCAI 2007), Morgan Kaufmann, 2007. (query answering for a logical fragment of OWL DL)
- B. Glimm, S. Rudolph: **Conjunctive Query Entailment: Decidable in Spite of O, I, and Q.** Proc. 22rd Int Workshop on Description Logics (DL 2009). CEUR, 2009. (latest results related to querying expressive logics underlying OWL DL)
- C. Lutz: **Inverse Roles Make Conjunctive Queries Hard.** Proc. 20th Int Workshop on Description Logics (DL 2007). CEUR, 2007. (impact of property inverses on querying light-weight logics)
- M. Krötzsch, S. Rudolph, and P. Hitzler. **Conjunctive queries for a tractable fragment of OWL 1.1.** Proc. 6th Int. Semantic Web Conf. (ISWC 2007), Springer, 2007. (querying "OWL EL")
- D. Calvanese, T. Eiter, M.M. Ortiz. **Answering regular path queries in expressive description logics: An automata-theoretic approach.** Proc. 22rd AAAI Conference on Artificial Intelligence (AAAI-07), 2007. (answering queries that describe property paths with regular expressions)