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

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

A set of RDF triples is called an **RDF graph**

\[U = \text{set of } U\text{ris}\]
\[B = \text{set of } \text{Blank nodes}\]
\[L = \text{set of } \text{Literals}\]
RDFS: An example

person rdf:dom works_in rdf:range company

sportman rdf:sc

soccer_player rdf:dom plays_in rdf:range soccer_team

Ronaldinho rdf:type plays_in Barcelona rdf:type

lives_in Spain
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:

- Can be defined in terms of classical notions such as model, interpretation, etc.
  - As for the case of first order logic
- Has a graph characterization via homomorphisms.
A function $h : U \cup B \cup L \rightarrow U \cup B \cup L$ 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: $G_1 \rightarrow G_2$

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

Theorem (CM77)

\[ G_1 \models G_2 \text{ if and only if there is a homomorphism } G_2 \rightarrow G_1. \]

Complexity

Entailment for RDF is NP-complete
Previous characterization of entailment is not enough to deal with RDFS vocabulary: \( (\text{Ronaldinho}, \text{rdf:type}, \text{person}) \)
Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:

- `rdf:sc`: transitive
- `rdf:sp`: transitive

More complicated interactions:

\[
(p, \text{rdf:dom}, c) \quad (a, p, b) \\
(a, \text{rdf:type}, c)
\]

RDFS entailment can be characterized by a set of rules:

- An Existential rule
- Subproperty rules
- Subclass rules
- Typing rules
- Implicit typing
Inference system in [MPG07] has 14 rules:

**Existential rule**
\[
\frac{G_1}{G_2} \text{ if } G_2 \rightarrow G_1
\]

**Subproperty rules**
\[
\frac{(p, \text{rdf:sp}, q)}{(a, p, b)} \frac{(a, p, b)}{(a, q, b)}
\]

**Subclass rules**
\[
\frac{(a, \text{rdf:sc}, b)}{(b, \text{rdf:sc}, c)} \frac{(b, \text{rdf:sc}, c)}{(a, \text{rdf:sc}, c)}
\]

**Typing rules**
\[
\frac{(p, \text{rdf:dom}, c)}{(a, \text{rdf:type}, c)} \frac{(a, \text{rdf:type}, c)}{(a, p, b)}
\]

**Implicit typing**
\[
\frac{(q, \text{rdf:dom}, a)}{(b, \text{rdf:sp}, q)} \frac{(b, \text{rdf:sp}, q)}{(b, p, c)} \frac{(b, p, c)}{(b, \text{rdf:type}, a)}
\]
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:

\[
\text{ground}(G) : \text{Graph obtained by replacing every blank } B \text{ in } G \text{ by a constant } c_B.
\]

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

Closure of an RDF graph \( G \) (denoted by \( \text{closure}(G) \)):

\[
G \cup \{ t \in (U \cup B) \times U \times (U \cup B \cup L) \mid \text{there exists a ground tuple } t' \text{ such that } \text{ground}(G) \models t' \text{ and } t = \text{ground}^{-1}(t') \}
\]
Closure of an RDF Graph: Example
Closure of an RDF graph: complexity

Proposition (H04, GHM04, MPG07)

\[ G_1 \models G_2 \iff G_2 \rightarrow \text{closure}(G_1) \]

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?
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 $\text{core}(G)$).
- Deciding if $G$ is a core is coNP-complete.
- Deciding if $G = \text{core}(G')$ is DP-complete.
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 \( \text{nf}(G_1) \cong \text{nf}(G_2) \).
- \( G_1 \models G_2 \) iff \( G_2 \rightarrow \text{nf}(G_1) \)

**Complexity**

*The problem of deciding if \( G_1 = \text{nf}(G_2) \) is DP-complete.*
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 $Q(D) \simeq Q(D')$
- $Q(D)$ 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.
If $D \equiv D'$, then $Q(D) = Q(D')$
(Weaker) If $D \equiv D'$, then $Q(D) \simeq Q(D')$

- 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
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.)
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
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 $V$.

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

Issues:

- Free variables in $B$ (projection)
- Treatment of blank nodes in $B$
- Treatment of blank nodes in $H$
A valuation is a function \( v : V \rightarrow U \cup B \cup L \)

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

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

A single answer is an element of \( \text{preans}(Q, D) \)
Querying RDF data: Two semantics

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

$$\text{ans}_U(Q, D) = \bigcup \text{preans}(Q, D)$$

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

$$\text{ans}_M(Q, D) = \biguplus \text{preans}(Q, D)$$

**Proposition**

1. For both semantics, if $D \models D'$ then $\text{ans}(Q, D') \models \text{ans}(Q, D)$
2. For all $D$, $\text{ans}_U(Q, D) \models \text{ans}_M(Q, D)$
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
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).*
A query $Q$ contains a query $Q'$, denoted $Q \subseteq Q'$ iff $\text{ans}(Q, D)$ comprises all the information of $\text{ans}(Q', D)$.

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

In our setting we have two versions:

$\begin{align*}
\text{ans}(Q', D) & \subseteq \text{ans}(Q, D) & (Q \sqsubseteq_p Q') \\
\text{preans}(Q, D) & \subseteq \text{preans}(Q', D) & (\text{modulo iso}) \quad (Q \sqsubseteq_m Q')
\end{align*}$

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 $\mu$ 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**

<table>
<thead>
<tr>
<th>$G$</th>
<th>$t$</th>
<th>$[[t]]_G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R_1, \text{name, john})$</td>
<td>$(?X, \text{name, ?Y})$</td>
<td>$\mu_1$:</td>
</tr>
<tr>
<td>$(R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>})$</td>
<td></td>
<td>$R_1$ \quad john</td>
</tr>
<tr>
<td>$(R_2, \text{name, paul})$</td>
<td></td>
<td>$R_2$ \quad paul</td>
</tr>
</tbody>
</table>

$\mu_2$:
Compatible mappings

**Definition**

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

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ :</td>
<td>$R_1$</td>
<td>john</td>
<td>$\text{<a href="mailto:J@edu.ex">J@edu.ex</a>}$</td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td>$R_1$</td>
<td>john</td>
<td>$\text{<a href="mailto:P@edu.ex">P@edu.ex</a>}$</td>
<td>$R_2$</td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_2$ :</td>
<td>$R_1$</td>
<td>john</td>
<td>$\text{<a href="mailto:J@edu.ex">J@edu.ex</a>}$</td>
<td></td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_3$ :</td>
<td>$R_1$</td>
<td>john</td>
<td>$\text{<a href="mailto:P@edu.ex">P@edu.ex</a>}$</td>
<td>$R_2$</td>
</tr>
</tbody>
</table>

$\mu_2$ and $\mu_3$ are not compatible
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:

**Definition**

**Join:** $M_1 \Join M_2$
- extending mappings in $M_1$ with compatible mappings in $M_2$

**Difference:** $M_1 \setminus M_2$
- 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:** $M_1 \Join_{lo} M_2 = (M_1 \Join M_2) \cup (M_1 \setminus M_2)$
Semantics of general graph patterns

Definition

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

- $\llbracket (P_1 \text{ AND } P_2) \rrbracket_G \ = \ \llbracket P_1 \rrbracket_G \land \llbracket P_2 \rrbracket_G$
- $\llbracket (P_1 \text{ UNION } P_2) \rrbracket_G \ = \ \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G$
- $\llbracket (P_1 \text{ OPT } P_2) \rrbracket_G \ = \ \llbracket P_1 \rrbracket_G \lor \llbracket P_2 \rrbracket_G$
- $\llbracket (P \text{ FILTER } R) \rrbracket_G \ = \ \{ \mu \in \llbracket P \rrbracket_G | \mu \text{ satisfies } R \}$
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 $\mu$, 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 FILTERs.
Evaluation including \textbf{UNION} is NP-complete.

\textbf{Theorem (PAG06)}

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

\textbf{Proof idea}

\begin{itemize}
\item Reduction from \textit{3SAT}.  
\item A pattern encodes the propositional formula.  
\item $\neg$\textit{bound} is used to encode negation.
\end{itemize}
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.)*
Assume $\varphi = \forall x_1 \exists y_1 \psi$, where $\psi = (x_1 \lor \neg y_1) \land (\neg x_1 \lor y_1)$.

We generate $G$, $P_\varphi$ and $\mu_0$ such that $\mu_0$ belongs to the answer of $P_\varphi$ over $G$ iff $\varphi$ is valid:

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

$P_\psi : ((a, tv, ?X_1) \text{ AND } (a, tv, ?Y_1)) \text{ FILTER } ((?X_1 = 1 \lor ?Y_1 = 0) \land (?X_1 = 0 \lor ?Y_1 = 1))$

$P_\varphi : (a, true, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi))$

$\mu_0 : \{?B_0 \mapsto 1\}$
PSPACE-hardness: A closer look

\[ P_\varphi : (a,\text{true},?B_0) \ \text{OPT} \ (P_1 \ \text{OPT} \ (Q_1 \ \text{AND} \ P_\psi)) \]

\[ P_1 : (a,\text{tv},?X_1) \]

\[ Q_1 : (a,\text{tv},?X_1) \ \text{AND} \ (a,\text{tv},?Y_1) \ \text{AND} \ (a,\text{false},?B_0) \]
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.*
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

\[
( \cdots \cdots \cdots \ ( A \ OPT \ B ) \ \cdots \cdots \cdots )
\]

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

Example

\[
( ( (?Y, name, paul) \ OPT \ (?X, email, ?Z) ) \ \text{AND} \ (?X, name, john) )
\]
In the PSPACE-hardness reduction we use this formula:

\[
P_\varphi : (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi))
\]

\[
P_1 : (a, tv, ?X_1)
\]

\[
Q_1 : (a, tv, ?X_1) \text{ AND } (a, tv, ?Y_1) \text{ AND } (a, false, ?B_0)
\]

It is not well-designed: \( B_0 \)
Well–designed patterns: reordering/optimization

For well-designed patterns

\[ P_1 \text{ AND } (P_2 \text{ OPT } P_3) \equiv (P_1 \text{ AND } P_2) \text{ OPT } P_3 \]

\[ (P_1 \text{ OPT } P_2) \text{ OPT } P_3 \equiv (P_1 \text{ OPT } P_3) \text{ OPT } P_2 \]

**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