RDF Data Management: Reasoning on Web Data

Damian Bursztyn\textsuperscript{1,2}, François Goasdoué\textsuperscript{3,1}, Ioana Manolescu\textsuperscript{1,2}, Alexandra Roatiş\textsuperscript{1,2}

\textsuperscript{1}INRIA \hspace{2cm} \textsuperscript{2}Université Paris Sud France \hspace{2cm} \textsuperscript{3}Université Rennes 1 France

ioana.manolescu@inria.fr, http://pages.saclay.inria.fr/Ioana.Manolescu

https://team.inria.fr/oak

ICDE 2015, Korea
Part I

Motivation and outline
Querying Web data

1. Trees, e.g., XML documents, JSON
2. Graphs, e.g., RDF (W3C’s Resource Description Framework)
   - Famous application: the Linked Open Data cloud
Querying Web data

1. Trees, e.g., XML documents, JSON
   - Famous application: the Linked Open Data cloud (2007)
2. Graphs, e.g., RDF (W3C’s Resource Description Framework)
Motivation and context

Querying Web data

1. Trees, e.g., XML documents, JSON
2. Graphs, e.g., RDF (W3C’s Resource Description Framework)
   - Famous application: the Linked Open Data cloud (2008)
Motivation and context

Querying Web data

1. Trees, e.g., XML documents, JSON
2. Graphs, e.g., RDF (W3C’s Resource Description Framework)
   - Famous application: the Linked Open Data cloud (2009)
Querying Web data

1. Trees, e.g., XML documents, JSON
2. Graphs, e.g., RDF (W3C's Resource Description Framework)
   - Famous application: the Linked Open Data cloud (2010)
Querying Web data

1. Trees, e.g., XML documents, JSON
2. Graphs, e.g., RDF (W3C’s Resource Description Framework)
   - Famous application: the Linked Open Data cloud (2014)
Motivation and context

Linked Open Data cloud, 2014

900,129 documents describing 8,038,396 resources
(Schmachtenberg, Bizer, Paulheim, ISWC 2014)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Datasets</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Government</td>
<td>183</td>
<td>18.05</td>
</tr>
<tr>
<td>Publications</td>
<td>96</td>
<td>9.47</td>
</tr>
<tr>
<td>Life sciences</td>
<td>83</td>
<td>8.19</td>
</tr>
<tr>
<td>User-generated</td>
<td>48</td>
<td>4.73</td>
</tr>
<tr>
<td>Cross-domain</td>
<td>41</td>
<td>4.04</td>
</tr>
<tr>
<td>Media</td>
<td>22</td>
<td>2.17</td>
</tr>
<tr>
<td>Geographic</td>
<td>21</td>
<td>2.07</td>
</tr>
<tr>
<td>Social web</td>
<td>520</td>
<td>51.28</td>
</tr>
<tr>
<td>Total</td>
<td>1014</td>
<td>100</td>
</tr>
</tbody>
</table>

There is more (Billion Triple Challenge etc.)
Querying RDF data

1. Three-attribute relation (subject, property, object)
   - The subject is the resource being described
   - The resource has the property property whose value is object
   - Resource type is a property, specified just like any other

Queries are larger (more atoms) than relational counterpart
Motivation and context

Querying RDF data

1. Three-attribute relation (subject, property, object)
   - The subject is the resource being described
   - The resource has the property property whose value is object
   - Resource type is a property, specified just like any other

Queries are larger (more atoms) than relational counterpart
Many RDF DM platforms
**Querying RDF data**

1. **Three-attribute relation** (subject, property, object)
   - The subject is the **resource** being **described**
   - The resource has the property **property** whose value is **object**
   - Resource **type** is a property, specified just like any other

Queries are **larger** (more atoms) than relational counterpart

Many RDF DM platforms

2. **RDFS semantics** providing information about the properties and classes (types) of resources:
   - Any **undergraduateStudent** is a **Student**
   - Anyone having a **graduationDate** is a **Student** (but may also be of other types) ...
Querying RDF data

1. Three-attribute relation (subject, property, object)
   - The subject is the resource being described
   - The resource has the property property whose value is object
   - Resource type is a property, specified just like any other

Queries are larger (more atoms) than relational counterpart
Many RDF DM platforms

2. RDFS semantics providing information about the properties and classes (types) of resources:
   - Any undergraduateStudent is a Student
   - Anyone having a graduationDate is a Student (but may also be of other types) ...

RDF semantics leads to implicit data
Querying RDF data

1. Three-attribute relation (subject, property, object)
   Many RDF data management platforms

2. RDFS semantics providing information about the properties and classes (types) of resources

RDF semantics leads to implicit data
Querying RDF data

1. Three-attribute relation (subject, property, object)
   Many RDF data management platforms
2. RDFS semantics providing information about the properties and classes (types) of resources

RDF semantics leads to implicit data
Thus, RDF queries are (also) over data that isn’t there (yet must be taken into account)
Motivation and context

Querying RDF data

1. Three-attribute relation (subject, property, object)
   Many RDF DM platforms

2. RDFS semantics providing information about the properties and classes (types) of resources

RDF semantics leads to implicit data

Thus, RDF queries are (also) over data that isn’t there (yet must be taken into account)

This tutorial

Reflecting implicit data into query answering = reasoning
Querying RDF data

1. Three-attribute relation (subject, property, object)
   Many RDF DM platforms
2. RDFS semantics providing information about the properties and classes (types) of resources

This tutorial
Reflecting implicit data into query answering = reasoning

3. Richer semantics:
   - No Student is also a Company
   - Every Student has a Birthdate
   - OWL, Description Logic Dialects, Datalog± ...
Do we really need the semantics?

Yes. All the time.

Application knowledge / constraints:
- Every Senator is an ElectedOfficial which is a Person
- (On Wikipedia) being BornInAPlace means being a Person
- The source and destination of a tripFromTo are either a streetAddress, or a cityAddress or a countryAddress
Do we really need the semantics?

Yes. All the time.

Application knowledge / constraints:
- Every Senator is an ElectedOfficial which is a Person
- (On Wikipedia) being BornInAPlace means being a Person
- The source and destination of a tripFromTo are either a streetAddress, or a cityAddress or a countryAddress

1. Without the semantics, we may miss query answers
2. Semantic contraints are a compact way of encoding information ("every ElectedOfficial is a Person" stated only once)
Outline

1. Motivation
2. RDF data model and query language
3. Query answering techniques
4. Performance
5. Open issues
Part II

RDF data model and query language
The Resource Description Framework (RDF)

**RDF graph** – set of **triples**

<table>
<thead>
<tr>
<th>Assertion</th>
<th>Triple</th>
<th>Relational notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>s rdf:type o</td>
<td>o(s)</td>
</tr>
<tr>
<td>Property</td>
<td>s p o</td>
<td>p(s, o)</td>
</tr>
</tbody>
</table>

- **resource (URI)**
- **blank node**
- **literal (string)**
- property

**Example RDF graph**:

```

doi1

  hasTitle
    “El Aleph”

  hasName
    “J. L. Borges”

  publishedIn
    “1949”

  rdf:type

  writtenBy
    _:b1
```

- **doi**
- **:b1**
RDF Schema (RDFS)

Declare **deductive constraints** between classes and properties

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Triple</th>
<th>OWA interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subclass</td>
<td>s rdfs:subClassOf o</td>
<td>s ⊆ o</td>
</tr>
<tr>
<td>Subproperty</td>
<td>s rdfs:subPropertyOf o</td>
<td>s ⊆ o</td>
</tr>
<tr>
<td>Domain typing</td>
<td>s rdfs:domain o</td>
<td>( \Pi_{\text{domain}}(s) \subseteq o )</td>
</tr>
<tr>
<td>Range typing</td>
<td>s rdfs:range o</td>
<td>( \Pi_{\text{range}}(s) \subseteq o )</td>
</tr>
</tbody>
</table>

\( s \subseteq o \)
RDF data model – based on the open-world assumption.

→ deductive constraints – implicitly propagate triples

Implicit triples: part of the graph – not explicitly present

Entailment – reasoning mechanism
Open-world assumption and RDF entailment

**RDF data model** – based on the **open-world assumption**.

→ deductive constraints – implicitly propagate triples

**Implicit triples**: part of the graph – not explicitly present

**Entailment** – reasoning mechanism

```
set of explicit triples + \rightarrow \text{derive implicit triples}
```

some **entailment rules**
Open-world assumption and RDF entailment

RDF data model – based on the open-world assumption. → deductive constraints – implicitly propagate triples

**Implicit triples:** part of the graph – not explicitly present

Entailment – reasoning mechanism

| set of explicit triples + some entailment rules | → derive implicit triples |

Exhaustive application of entailment → **saturation** (closure)
The semantics of an RDF graph $G$ is its saturation $G^\infty$.

**Sample RDFS entailment rules**

<table>
<thead>
<tr>
<th>RDFS entailment rules</th>
<th>Instance entailment from combining schema and instance triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs9</td>
<td>$c_1 \text{ rdfs:subClassOf } c_2 \land s \text{ rdf:type } c_1 \vdash_{\text{RDF}} s \text{ rdf:type } c_2$</td>
</tr>
<tr>
<td>rdfs7</td>
<td>$p_1 \text{ rdfs:subPropertyOf } p_2 \land s p_1 o \vdash_{\text{RDF}} s p_2 o$</td>
</tr>
<tr>
<td>rdfs2</td>
<td>$p \text{ rdfs:domain } c \land s p o \vdash_{\text{RDF}} s \text{ rdf:type } c$</td>
</tr>
<tr>
<td>rdfs3</td>
<td>$p \text{ rdfs:range } c \land s p o \vdash_{\text{RDF}} o \text{ rdf:type } c$</td>
</tr>
</tbody>
</table>
SPARQL query language and SPARQL conjunctive queries

SPARQL is the W3C query language for RDF.

\[
q(a, t) \ :- \ (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, ”1949”})
\]
Query semantics / Answer set

query **evaluation** ≠ query **answering**

- The evaluation of a query only uses the graph’s **explicit triples**
- For the **(complete) answer set**, evaluate \( q \) against the graph’s **saturation**
Query answering example

Given the query $q(x, y) :\equiv \ x \text{ rdf:type } y$:
Query answering example

Given the query $q(x, y) :- x \text{ rdf:type } y$:

- $q(G) = \{(\text{doi}_1, \text{Book})\}$
Given the query $q(x, y) : - x$ rdf:type $y$: 

- $q(G) = \{(\text{doi}_1, \text{Book})\}$
- $q(G^\infty) = \{(\text{doi}_1, \text{Book}), (\text{doi}_1, \text{Publication}), (\_:b_1, \text{Person})\}$
Part III

Query answering techniques
The need for reasoning

Query answering needs explicit and implicit data!

- **Saturation**-based query answering
- **Reformulation**-based query answering
- Hybrids of the above
  - J. Urbani, F. van Harmelen, S. Schlobach, and H. Bal, “QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases”, ISWC 2011
Saturation-based query answering

**RDF Entailment Rules**

$G^\infty$

$G$

query $q$

answer

Widely studied, $q(G^\infty)$ can be computed using an RDBMS.

$G^\infty$ needs time to be computed and space to be stored.

Not suitable for high update rate (data and/or schema triples).
Saturation-based query answering

- Widely studied
- $q(G^\infty)$ can be computed using an RDBMS
- $G^\infty$ needs time to be computed and space to be stored
- Not suitable for high update rate (data and/or schema triples)
Saturation maintenance

Compute $\Delta$ for an update of an RDF graph $G$ s.t.
- $(\mu(G))^\infty = G^\infty \cup \Delta$ when $\mu$ an insertion
- $(\mu(G))^\infty = G^\infty \setminus \Delta$ when $\mu$ a deletion
Saturation maintenance by storing entailment proofs

Adding \((\text{doi}_2, \text{rdf}:\text{type}, \text{Book})\) to \(G\) adds \((\text{doi}_2, \text{rdf}:\text{type}, \text{Publication})\) to \(G^\infty\):

\[
\text{Proofs}((\text{doi}_2, \text{rdf}:\text{type}, \text{Publication})) = \{(\text{doi}_2, \text{rdf}:\text{type}, \text{Book}), (\text{Book}, \text{rdfs:subClassOf}, \text{Publication})\}\}

Deleting \((\text{doi}_1, \text{rdf}:\text{type}, \text{Book})\) from \(G\) does not affect \(G^\infty\):

\[
\text{Proofs}((\text{doi}_1, \text{rdf}:\text{type}, \text{Book})) = \{(\text{doi}_1, \text{writtenBy}, \text{b}_1), (\text{writtenBy}, \text{rdfs:domain}, \text{Book})\}\}

Saturation maintenance by checking entailment proofs


Adding \((\text{doi}_2, \text{rdf:}\text{type}, \text{Book})\) to \(G\) adds \((\text{doi}_2, \text{rdf:}\text{type}, \text{Publication})\) to \(G^\infty\) because of the proof
\((\text{doi}_2, \text{rdf:}\text{type}, \text{Book}), (\text{Book}, \text{rdfs:}\text{subClassOf}, \text{Publication}) \vdash_{RDF} (\text{doi}_2, \text{rdf:}\text{type}, \text{Publication})\) found at runtime

Deleting \((\text{doi}_1, \text{rdf:}\text{type}, \text{Book})\) from \(G\) does not affect \(G^\infty\) because there exists another proof \((\text{doi}_1, \text{writtenBy}, \_b_1), (\text{writtenBy}, \text{rdfs:}\text{domain}, \text{Book}) \vdash_{RDF} (\text{doi}_1, \text{rdf:}\text{type}, \text{Book})\) found at runtime
Saturation maintenance by counting entailment proofs

Adding \((\text{doi}_2, \text{rdf}:\text{type}, \text{Book})\) to \(G\) add \((\text{doi}_2, \text{rdf}:\text{type}, \text{Publication})\) to \(G^\infty\) with derivation count 1 because of the proof

\[(\text{doi}_2, \text{rdf}:\text{type}, \text{Book}), (\text{Book}, \text{rdfs}:\text{subClassOf}, \text{Publication}) \vdash_{\text{RDF}} (\text{doi}_2, \text{rdf}:\text{type}, \text{Publication})\] found at insert time
Saturation maintenance by counting entailment proofs

Deleting \((\text{doi}_1, \text{rdf:type}, \text{Book})\) from \(G\) does not affect \(G^\infty\) but decreases its derivation count to 1 due to:

\((\text{doi}_1, \text{writtenBy}, b_1), (\text{writtenBy}, \text{rdfs:domain}, \text{Book}) \vdash_{\text{RDF}} (\text{doi}_1, \text{rdf:type}, \text{Book})\);

it also decreases to 1 the derivation count of \((\text{doi}_1, \text{rdf:type}, \text{Publication})\)

▷ F. Goasdoué, I. Manolescu, A. Roatiș, ”Efficient query answering against dynamic RDF databases”, EDBT, 2013
Reformulation-based query answering

RDF Inference Rules

G

query $q$

query $q^{ref}$

answer

Robust to updates
Reformulated queries are complex, thus costly to evaluate
Reformulation-based query answering

- $q^{ref}(G)$ can be evaluated using an RDBMS
- Robust to updates
- Reformulated queries are complex, thus costly to evaluate

*RDF Inference Rules*
Reformulation-based query answering

Target reformulation languages for conjunctive queries (CQs):

- **mainly unions of CQs (UCQs)**
  - F. Goasdoué, I. Manolescu, A. Roatiş: “Efficient query answering against dynamic RDF databases”, EDBT 2013

- **joins of single-atom CQs (SCQs)**

- **joins of UCQs (JUCQs)**
Reformulation-based query answering

Target reformulation languages for conjunctive queries (CQs):

- mainly **unions of CQs (UCQs)**
  - F. Goasdoué, I. Manolescu, A. Roatiş: “Efficient query answering against dynamic RDF databases”, EDBT 2013

**Wait: is this about SQL syntax?!…**

- **joins of UCQs (JUCQs)**
Reformulation-based query answering

Target reformulation languages for conjunctive queries (CQs):

Wait: is this about SQL syntax?!

Yes. And it makes a big difference.

From failing to feasible, or 4 orders of magnitude speed-up on the 8 M triples DBLP dataset.
CQ-to-UCQ query reformulation example

\[ q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, t}), (b, \text{hasAuthor, a}), (b, \text{publishedIn, ”1949”}) \] leads to \( q^{ref} \):

\[ (0) \quad q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, t}), (b, \text{hasAuthor, a}), (b, \text{publishedIn, ”1949”}) \]
CQ-to-UCQ query reformulation example

\[
q(a, t) \leftarrow (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \text{ leads to } q^{ref}:
\]

\[
(0) \quad q(a, t) \leftarrow (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949")
\]

\[
(1) \quad q(a, t) \leftarrow (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949")
\]
CQ-to-UCQ query reformulation example

$$q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, "1949"})$$ leads to  $$q^{ref} :$$

(0) $$q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, "1949"})$$

(1) $$q(a, t) :- (b, \text{writtenBy, } x), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, "1949"})$$

(2) $$q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{writtenBy, } a), (b, \text{publishedIn, "1949"})$$
CQ-to-UCQ query reformulation example

$q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, ”1949”})$ leads to $q^{ref} :$

1. $q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, ”1949”})$
2. $q(a, t) :- (b, \text{writtenBy, x}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, ”1949”})$
3. $q(a, t) :- (b, \text{writtenBy, x}), (b, \text{hasTitle, } t), (b, \text{writtenBy, a}), (b, \text{publishedIn, ”1949”})$
CQ-to-UCQ query reformulation example

$q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, ”1949”})$ leads to $q^{ref}:

(0) \quad q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, ”1949”}) \cup

(1) \quad q(a, t) :- (b, \text{writtenBy, } x), (b, \text{hasTitle, } t), (b, \text{hasAuthor, } a), (b, \text{publishedIn, ”1949”}) \cup

(2) \quad q(a, t) :- (b, \text{rdf:type, Book}), (b, \text{hasTitle, } t), (b, \text{writtenBy, } a), (b, \text{publishedIn, ”1949”}) \cup

(3) \quad q(a, t) :- (b, \text{writtenBy, } x), (b, \text{hasTitle, } t), (b, \text{writtenBy, } a), (b, \text{publishedIn, ”1949”})$
CQ-to-UCQ query reformulation example

\[
q(a, t) : - (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), \\
(b, \text{publishedIn}, "1949") \text{ leads to } q^{\text{ref}}:
\]

(0) \[ q(a, t) : - (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), \\
(b, \text{publishedIn}, "1949") \]

(1) \[ q(a, t) : - (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), \\
(b, \text{publishedIn}, "1949") \]

(2) \[ q(a, t) : - (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a), \\
(b, \text{publishedIn}, "1949") \]

(3) \[ q(a, t) : - (b, \text{writtenBy}, x), (b, \text{hasTitle}, t), (b, \text{writtenBy}, a), \\
(b, \text{publishedIn}, "1949") \]

\[
q(G^\infty) = q^{\text{ref}}(G) = \{(\_:b_1, "El Aleph")\}.
\]
CQ-to-SCQ query reformulation example

\[ q(a, t) : - (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \text{ produces the } q^{\text{ref}} : \]

\[
\begin{align*}
(0) \quad q(b) : & - (b, \text{rdf:type}, \text{Book}) \cup (b, \text{writtenBy}, x) \\
(1) \quad q(b, a) : & - (b, \text{hasAuthor}, a) \cup (b, \text{writtenBy}, a) \\
(2) \quad q(b) : & - (b, \text{publishedIn}, "1949")
\end{align*}
\]
CQ-to-SCQ query reformulation example

\[ q(a, t) :- (b, \text{rdf:type}, \text{Book}), (b, \text{hasTitle}, t), (b, \text{hasAuthor}, a), (b, \text{publishedIn}, "1949") \]

produces the \( q^{ref} \):

1. \( q(b) :- (b, \text{rdf:type}, \text{Book}) \cup (b, \text{writtenBy}, x) \)
2. \( q(b, a) :- (b, \text{hasAuthor}, a) \cup (b, \text{writtenBy}, a) \)
3. \( q(b) :- (b, \text{publishedIn}, "1949") \)

\[ q(G^\infty) = q^{ref}(G) = \{(-:b_1, "El Aleph")\} \]
CQ-to-JUCQ query reformulation

1. **Enlarges the query reformulation language** w.r.t. UCQ/SCQ to have more than one reformulation alternative.
CQ-to-JUCQ query reformulation

1. **Enlarges the query reformulation language** w.r.t. UCQ/SCQ to have more than one reformulation alternative
2. **Uses a cost model** for estimating the cost of evaluating $q^{\text{ref}}$ through an RDBMS
CQ-to-JUCQ query reformulation

1. Enlarges the query reformulation language w.r.t. UCQ/SCQ to have more than one reformulation alternative
2. Uses a cost model for estimating the cost of evaluating $q^{\text{ref}}$ through an RDBMS
3. Chooses the cheapest alternative from the search space.
Optimized reformulation into JUCQs

Query $q$

CQ-to-UCQ ref. algo.

Graph $G$

$q^1$

$c(q^1)$

$q^{ref}$

$c(q^{ref})$

UCQ ref

$q^{ref}$

$q^{best}$

$c(q^{best})$

JUCQ ref

$q^{best}$

RDBMS

Results
CQ-to-JUCQ query reformulation

Given the query

\[ q_1(x, y) :\quad \text{x rdf:type y,} \]
\[ \quad \text{x ub:degreeFrom "http://www.University532.edu"}, \] \[ \quad \text{x ub:memberOf "http://www.Department1.University7.edu"} \]

\[ (t_1) \quad (t_2) \quad (t_3) \]
Given the query

\[
q_1(x, y) :\quad x \text{ rdf:type } y, \\
x \text{ ub:degreeFrom } "http://www.University532.edu", \\
x \text{ ub:memberOf } "http://www.Department1.University7.edu"
\]

and a state-of-the-art CQ-to-UCQ reformulation algorithm \( \text{ref} \):

- the UCQ reformulation is: \((t_1, t_2, t_3)^{\text{ref}}\)
- the SCQ reformulation is: \((t_1)^{\text{ref}} \Join (t_2)^{\text{ref}} \Join (t_3)^{\text{ref}}\)
CQ-to-JUCQ query reformulation

Given the query

\[ q_1(x, y) : - x \text{ rdf:type } y, \]
\[ x \text{ ub:degreeFrom "http://www.University532.edu"}, \]
\[ x \text{ ub:memberOf "http://www.Department1.University7.edu"}, \]

and a state-of-the-art CQ-to-UCQ reformulation algorithm \(ref\), the space of JUCQs is:

1. \((t_1, t_2, t_3)^{ref}\)
2. \((t_1)^{ref} \cap (t_2)^{ref} \cap (t_3)^{ref}\)
3. \((t_1, t_2)^{ref} \cap (t_3)^{ref}\)
4. \((t_1)^{ref} \cap (t_2, t_3)^{ref}\)
5. \((t_1, t_3)^{ref} \cap (t_2)^{ref}\)
6. \((t_1, t_2)^{ref} \cap (t_1, t_3)^{ref}\)
7. \((t_1, t_2)^{ref} \cap (t_2, t_3)^{ref}\)
8. \((t_1, t_3)^{ref} \cap (t_2, t_3)^{ref}\)
CQ-to-JUCQ query reformulation algorithms

Exhaustive algorithm
Impractical since search space size is the number of minimal covers of a set of $n$ elements

Greedy algorithm
Driven by cost; starting from 1-atom fragments and “growing” them

Given the query

$$q_1(x, y) := \quad x \text{ rdf:type } y, \quad (t_1)$$
$$\quad x \text{ ub:degreeFrom } "http://www.University532.edu", \quad (t_2)$$
$$\quad x \text{ ub:memberOf } "http://www.Department1.University7.edu" \quad (t_3)$$

a cost-based greedy exploration is:

$$(t_1)^{ref} \otimes (t_2)^{ref} \otimes (t_3)^{ref}$$
CQ-to-JUCQ query reformulation: greedy algorithm example

Given the query

\[ q_1(x, y) :- x \text{ rdf\:type } y, \]
\[ x \text{ ub\:degreeFrom "http://www.University532.edu"}, \]
\[ x \text{ ub\:memberOf "http://www.Department1.University7.edu"} \]

a cost-based greedy exploration is:

\[ (t_1)^{ref} \Join (t_2)^{ref} \Join (t_3)^{ref} \]
\[ (t_1, t_2)^{ref} \Join (t_3)^{ref} \]
\[ (t_1, t_3)^{ref} \Join (t_2)^{ref} \]
\[ (t_1)^{ref} \Join (t_2, t_3)^{ref} \]
CQ-to-JUCQ query reformulation algorithm

Given the query

\[ q_1(x, y) :- x \text{ rdf:type } y, \]
\[ x \text{ ub:degreeFrom "http://www.University532.edu"}, \]
\[ x \text{ ub:memberOf "http://www.Department1.University7.edu"} \]

a cost-based greedy exploration is:

\[
\begin{align*}
(t_1) & \quad \text{ref} \\
(t_2) & \quad \text{ref} \\
(t_3) & \quad \text{ref}
\end{align*}
\]

\[
\begin{align*}
(t_1, t_2) & \quad \text{ref} \ \Join \ (t_3) \quad \text{ref} \\
(t_1, t_3) & \quad \text{ref} \ \Join \ (t_2) \quad \text{ref} \\
(t_1) & \quad \text{ref} \ \Join \ (t_2, t_3) \quad \text{ref} \\
(t_1, t_3) & \quad \text{ref} \ \Join \ (t_1, t_2) \quad \text{ref} \\
(t_1, t_3) & \quad \text{ref} \ \Join \ (t_2, t_3) \quad \text{ref}
\end{align*}
\]
Part IV

Performance
Two select performance aspects

Chosen for the very stark differences of performance

1. Saturation vs. reformulation
2. Different reformulation methods
How to compare?

**Saturation**

Saturation cost paid before any query arrives.
Maintenance costs at every update.
How to compare?

**Saturation**
Saturation cost paid **before any query arrives**.
Maintenance costs **at every update**.

**Reformulation**
No cost upfront; no cost at update time.
Cost of evaluating large query **at every execution**.
How to compare?

**Saturation**
Saturation cost paid before any query arrives.
Maintenance costs at every update.

**Reformulation**
No cost upfront; no cost at update time.
Cost of evaluating large query at every execution.

**Query threshold**
How many times to run $q$ for the saturation cost to amortize?
The smallest $n$ such that:

$$n \times t(q^{ref}(G)) > t_{sat}(G) + n \times t(q(G^{\infty}))$$

▷ F. Goasdoué, I. Manolescu, and A. Roatiş, “Efficient Query Answering Against Dynamic RDF Databases”, EDBT 2013
Thresholds for comparing saturation and reformulation

**Query threshold** The smallest $n$ such that:

$$n \times t(q^{ref}(G)) > t_{sat}(G) + n \times t(q(G^{\infty}))$$

**Insertion threshold** The smallest $n$ such that:

$$n \times t(q^{ref}(G)) + t(\Delta_+, G) > t_{sat}(G) + t(\Delta_+, G^{\infty}) + n \times t(q(G^{\infty}))$$

**Deletion threshold** The smallest $n$ such that:

$$n \times t(q^{ref}(G)) + t(\Delta_-, G) > t_{sat}(G) + t(\Delta_-, G^{\infty}) + n \times t(q(G^{\infty}))$$
Query, insertion and deletion thresholds

In parenthesis the size of the UCQ ref of $q$
Query, insertion and deletion thresholds

DBLP database

In parenthesis the size of the UCQ ref of $q$

From left to right:
query threshold
threshold for an instance insertion
threshold for an instance deletion
threshold for a schema insertion
threshold for a schema deletion
Saturation vs. reformulation: take-home message

**Saturation**
It takes time to build and space to store.
Query evaluation is very simple.
Maintenance is hard especially for updates to the schema!

**Reformulation**
Oftentimes evaluating reformulated queries is complex.
No overhead on updates.
CQ-to-JUCQ query reformulation performance

Given the query

\[ q_1(x, y) :\ x \text{ rdf:type } y, \]
\[ x \text{ ub:degreeFrom "http://www.University532.edu"}, \]
\[ x \text{ ub:memberOf "http://www.Department1.University7.edu"} \]

and the LUBM 100M benchmark:

<table>
<thead>
<tr>
<th>JUCQ</th>
<th>#reformulations</th>
<th>exec. time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((t_1, t_2, t_3)^{ref})</td>
<td>2,256</td>
<td>6,387</td>
</tr>
<tr>
<td>((t_1)^{ref} \cap (t_2)^{ref} \cap (t_3)^{ref})</td>
<td>195</td>
<td>1,074,026</td>
</tr>
<tr>
<td>((t_1, t_2)^{ref} \cap (t_3)^{ref})</td>
<td>755</td>
<td>1,968</td>
</tr>
<tr>
<td>((t_1)^{ref} \cap (t_2, t_3)^{ref})</td>
<td>200</td>
<td>846,710</td>
</tr>
<tr>
<td>((t_1, t_3)^{ref} \cap (t_2)^{ref})</td>
<td>568</td>
<td>554</td>
</tr>
<tr>
<td>((t_1, t_2)^{ref} \cap (t_1, t_3)^{ref})</td>
<td>1,316</td>
<td>2,734</td>
</tr>
<tr>
<td>((t_1, t_2)^{ref} \cap (t_2, t_3)^{ref})</td>
<td>764</td>
<td>2,289</td>
</tr>
<tr>
<td>((t_1, t_3)^{ref} \cap (t_2, t_3)^{ref})</td>
<td>576</td>
<td>588</td>
</tr>
</tbody>
</table>
Datasets and RDBMS engines

DBLP (8 M) and LUBM (1 M and 100 M) millions triples

- PostgreSQL 9.3.2
- System A
- System B
Reformulation algorithms

Basic CQ-to-UCQ algorithm

Picked that of [Goasdoué et al., EDBT’13] since it handles the largest known fragment of RDF.

Comparison:

1. UCQ reformulation
2. SCQ reformulation
3. Greedy JUCQ reformulation
4. Exhaustive JUCQ reformulation
Performance optimized reformulation experiments on three RDBMSs

Query answering on LUBM 100 M using PostgreSQL

28 queries; 2 to 6 atoms; 1 to 318,089 reformulations

Postgres query answering
LUBM (100M)
Query answering on LUBM 100 M using System A

28 queries; 2 to 6 atoms; 1 to 318,089 reformulations
Query answering on LUBM 100 M using System B

28 queries; 2 to 6 atoms; 1 to 318,089 reformulations
Optimized reformulation: take-home message

1. Equivalent SQL syntaxes are not equal from the RDBMS optimizer perspective (inside or outside well-supported dialect)
2. Choosing the reformulation with the help of textbook cost model formulas makes queries feasible or efficient when they were not
3. This amounts to enlarging the optimizer’s “can-do” dialect, at a very modest performance overhead.
Part V

Conclusion and perspectives
Constraints (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language”
Where we stand

**Constraints** (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language” **RDFS** constraints are among the simplest.
Where we stand

**Constraints** (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language”

**RDFS** constraints are among the simplest.

We considered RDF query answering through **FOL reduction** (i.e., **SQL**).
Where we stand

**Constraints** (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language” RDFS constraints are among the simplest. We considered RDF query answering through **FOL reduction** (i.e., SQL). Not any SQL query resulting from reformulation is handled well by current RDBMSs!
Open issues and perspectives

Where we stand

Constraints (a.k.a. semantics) are crucial for applications, so the push is continuous for choosing “the right constraint language”
RDFS constraints are among the simplest.
We considered RDF query answering through FOL reduction (i.e., SQL).
Not any SQL query resulting from reformulation is handled well by current RDBMSs!
Vast performance differences between:

- Materialization- and saturation-based query answering, depending on the data and update profile
- Different reformulations of the same query; cost-based approach
What is ahead

Continuous push on the **expressivity - efficiency** frontier
Continuous push on the **expressivity - efficiency** frontier

RDBMSs are highly efficient for **some forms** of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints

"Novel" platforms (MapReduce, NoSQL...) will raise the same query evaluation performance problems anyway.

Most attractive languages currently: DL-Lite, Datalog ±

New relevance of query optimization literature and research!
Continuous push on the **expressivity - efficiency** frontier

RDBMSs are highly efficient for **some forms** of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints

RDBMSs have convenient features (indexing, join order, transactions) that make them **attractive back-ends for query answering**.
What is ahead

Continuous push on the **expressivity - efficiency** frontier

RDBMSs are highly efficient for **some forms** of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints.

RDBMSs have convenient features (indexing, join order, transactions) that make them **attractive back-ends for query answering**.

“Novel” platforms (MapReduce, NoSQL...) will raise the **same query evaluation performance problems** anyway.
What is ahead

Continuous push on the **expressivity - efficiency** frontier

RDBMSs are highly efficient for **some forms** of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints

RDBMSs have convenient features (indexing, join order, transactions) that make them **attractive back-ends for query answering**.

“Novel” platforms (MapReduce, NoSQL...) will raise **the same query evaluation performance problems** anyway.

Most attractive languages currently: **DL-Lite, Datalog**

\[\pm\]
Continuous push on the **expressivity - efficiency** frontier

RDBMSs are highly efficient for **some forms** of queries (typically conjunctive queries of medium size), not for all FOL reductions of queries under constraints.

RDBMSs have convenient features (indexing, join order, transactions) that make them **attractive back-ends for query answering**.

“Novel” platforms (MapReduce, NoSQL...) will raise the **same query evaluation performance problems** anyway.

Most attractive languages currently: **DL-Lite, Datalog⁺**

**New relevance of query optimization literature and research!**