KR Querying & Reasoning for Large Knowledge Graphs
Part I
Sebastian Rudolph, TU Dresden
KR Querying & Reasoning for Large Knowledge Graphs

Part II

Axel Polleres, WU Vienna
Are OWL and RDFS entailment enough?

• Determining Satisfiability and Consistency and Entailments in KGs is one thing...

• But:
  • Mostly you actually want to **retrieve** information from a KG
  • You also need to deal with contextualized information
  • Existing KGs aren’t consistent 😞
Mostly you actually want to retrieve information from a KG

- E.g. from DBpedia

- One of the central datasets of the Linked Open Data-Cloud
- RDF extracted from Wikipedia-Infoboxes
- SPARQL endpoint, e.g.:
  - „Cities in the UK with more than 1M population“:

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT DISTINCT ?city ?pop WHERE {
  ?city a yago:City108524735 .
  ?city dbo:populationTotal ?pop
  FILTER ( ?pop > 1000000 )
}
```

Besides OWL, RDF, RDFS, we need query languages!

Structured queries (SPARQL):

http://yasgui.org/short/UVOyhX8ft
You also need to deal with contextualized information

- E.g. from DBpedia

„Cities in the Italy with more than 1M population“:

Structured queries (SPARQL):

```sparql
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT DISTINCT ?city ?pop WHERE {
  ?city a yago:City108524735 .
  ?city dbo:country:Italy .
  ?city dbo:populationTotal ?pop
  FILTER ( ?pop > 1000000 )
}
```

Doesn’t work!
Existing KGs aren’t consistent 😞 [1]

• E.g. DBpedia

Dbpedia Ontology:

dbo:Agent owl:disjointWith dbo:Place.

dbo:Country rdfs:subClassOf dbo:Place.

dbo:Organisation rdfs:subClassOf dbo:Agent.

Querying and Reasoning for KGs?

• Querying KGs with SPARQL (10min)
• Reasoning by Querying (Rewriting vs. Materialisation) (10min)
• Querying and Reasoning over Contextualised graph data (10min)
• Other issues: (15min)
  • Federation, Path Queries over Linked Data
  • SPARQL for KG Construction (scalability issues)
  • Updates
Querying KGs with SPARQL

Find some more detailed material to introduce SPARQL on my Webpage: http://polleres.net/presentations/
SPARQL

• “Just like SQL, only for matching patterns on (directed labelled) Graphs ...”

```sparql
SELECT ?X
WHERE {
}
```

• Standard protocol to access RDF data over the Web (SPARQL Protocol)

Try it!
Conjunction (.), disjunction (UNION), optional (OPTIONAL) patterns and filters (FILTER)

*Names of scientists from places in Italy?*

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>    
PREFIX dbr: <http://dbpedia.org/resource/>     
PREFIX dbo: <http://dbpedia.org/ontology/>    

SELECT ?N
WHERE
{
  ?X a dbo:Scientist; foaf:name ?N ;
  dbo:birthPlace [ dbo:country dbr:Italy] 
}
ORDER BY ?N
LIMIT 10
```

- Shortcuts for namespace prefixes and to group several triple patterns
- Slicing and dicing (ORDER BY, LIMIT/OFFSET ...)

<table>
<thead>
<tr>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>
Conjunction (.), disjunction (UNION), optional (OPTIONAL) patterns and filters (FILTER)

Names of scientists or writers born in Bologna?

```
SELECT ?N
WHERE
{
  { ?X a dbo:Scientist }
  UNION
  { ?X a dbo:Writer }
  ?X dbo:birthPlace dbr:Bologna; foaf:name ?N
}
ORDER BY ?N
```
Conjunction (.), disjunction (UNION), optional (OPTIONAL) patterns and filters (FILTER)

*Names of scientists born in Bologna and optionally their deathPlace?*

```sparql
SELECT ?N ?D
WHERE
{
  ?X a dbo:Scientist ; dbo:birthPlace dbr:Bologna; foaf:name ?N
  OPTIONAL {?X dbo:deathPlace ?D }
}
ORDER BY ?N
```

Note: variables can be unbound in a result!
Conjunction (.), disjunction (UNION), optional (OPTIONAL) patterns and filters (FILTER), ...

People born in Bologna called “Valentina”

```
SELECT ?X ?N
WHERE
{
  ?X dbo:birthPlace dbr:Bologna ;
    foaf:name ?N .
  FILTER( CONTAINS( ?N, "Valentina") )
}
```

<table>
<thead>
<tr>
<th>X</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dbpedia.org/resource/Valentina_Fago">http://dbpedia.org/resource/Valentina_Fago</a></td>
<td>&quot;Valentina Fago&quot;</td>
</tr>
</tbody>
</table>

SPARQL has lots of FILTER functions to filter text with regular expressions (REGEX), filter numerics (<,>,=,+,-,...), dates, etc.)
CONSTRUCT Queries to create new triples (or to transform one Knowledge Graph to another)

• **Bologna scientists, their birth and death places:**

```turtle
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX : <http://example.org/myKG/>

CONSTRUCT { ?X a :BolgnaScientist . ?Y a :BolgonaAuthor . }
WHERE { { ?X a dbo:Scientist }
  UNION
  { ?Y a dbo:Writer } }
```

```
dbr:Francesco_Maria_Grimaldi :bornIn :Bologna ; :diedIn dbr:Bologna ;
  :type ns2:BolgnaScientist .
dbr:Lamberto_Cesari dbo:born  :Bologna ; :diedIn dbr:Ann_Arbor ;
  :type :BolgnaScientist .
dbr:Amico_Bignami :born :Bologna ; :diedIn dbr:Rome ;
  :type ns2:BolgnaScientist .
dbr:Giovanni_Aldini :born :Bologna ; :diedIn dbr:Milan ;
  :type :BolgnaScientist .
dbr:Guglielmo_Marconi :born :Bologna ; dbo:diedIn dbr:Rome ;
  :type :BolgnaScientist .
...
dbr:Mario_Finzi :type :BolgonaAuthor .
```
Reasoning by Querying

- **Materialisation** *(can be done by rules/queries)* [2]
- **Rewriting** [1]

Reasoning by Querying – Materialisation:

```
SELECT ?X WHERE {
}
```

No answer 😞

DBpedia instance data:

dbr:Marta_Grandi a dbo:Entomologist ;
dbo:birthPlace dbr:Bologna .

dbr:Costanzo_Varolio a dbo:Medician;
  dbo:birthPlace dbr:Bologna .

Ontology (schema data):

dbo:Entomologist rdfs:subClassOf dbo:Scientist.
dbo:Medician rdfs:subClassOf dbo:Scientist.
dbo:Scientist rdfs:subClassOf dbo:Person.
dbo:Person rdfs:subClassOf dbo:Agent.
dbo:Organisation rdfs:subClassOf dbo:Agent.
dbo:birthPlace rdfs:domain dbo:Person .
...
RDFS deduction rules:

cf. https://www.w3.org/TR/rdf11-mt/

Could be read as Datalog deduction rules, e.g.:

\[
\begin{align*}
\text{triple}(U, \text{rdfs:subClassOf}, S) & \quad \leftarrow \quad \text{triple}(U, \text{rdfs:subClassOf}, V) \quad , \quad \text{triple}(V, \text{rdfs:subClassOf}, S) \\
\text{triple}(V, \text{rdfs:type}, S) & \quad \leftarrow \quad \text{triple}(U, \text{rdfs:subClassOf}, S) \quad , \quad \text{triple}(V, \text{rdf:type}, U)
\end{align*}
\]
RDFS deduction rules:

cf. https://www.w3.org/TR/rdf11-mt/

Could be read as Datalog deduction rules, e.g.:

\[
\text{triple}(U, \text{rdfs:subClassOf}, S) \quad :- \quad \text{triple}(U, \text{rdfs:subClassOf}, V) \quad , \quad \text{triple}(V, \text{rdfs:subClassOf}, S)
\]

\[
\text{triple}(V, \text{rdfs:type}, S) \quad :- \quad \text{triple}(U, \text{rdfs:subClassOf}, S) \quad , \quad \text{triple}(V, \text{rdf:type}, U)
\]
RDFS deduction rules:
cf. https://www.w3.org/TR/rdf11-mt/

... and Datalog deduction rules could be read/written as **SPARQL Construct** statements:

```
```
Reasoning by Querying – Materialisation:

```
SELECT ?X WHERE
{
}
```

Applying the rules of the previous slides exhaustively (until a fixpoint), will yield additional implicit KG edges (i.e., RDF triples):

**instance data:**

- dbr:Marta_Grandi a dbo:Entomologist ;
  dbo:birthPlace dbr:Bologna .
- dbr:Costanzo_Varolio a dbo:Medician;
  dbo:birthPlace dbr:Bologna .
- dbr:Marta_Grandi a dbo:Scientist,
  dbo:Person, dbo:Agent.
- dbr:Costanzo_Varolio a
  dbo:Scientist, dbo:Person, dbo:Agent.

**Ontology (schema data):**

- dbo:Entomologist rdfs:subClassOf dbo:Scientist.
- dbo:Medician rdfs:subClassOf dbo:Scientist.
- dbo:Sienticst rdfs:subClassOf dbo:Agent.
- dbo:Person rdfs:subClassOf dbo:Agent.
- dbo:Organisation rdfs:subClassOf dbo:Agent.
- dbo:birthPlace rdfs:domain dbo:Person .
Reasoning by Querying – Query Rewriting:

```
SELECT ?X WHERE
{
  { {?X  a dbo:Scientist } UNION {?X  a dbo:Medician } UNION {?X a dbo:Entomologist } }
}
```

instance data:

```
dbr:Marta_Grandi  a  dbo:Entomologist ;
dbo:birthPlace  dbr:Bologna .

dbr:Costanzo_Varolio  a  dbo:Medician;
dbo:birthPlace  dbr:Bologna .
```

Alternatively, the rules can be used “backwards” to rewrite the original query to yield a more generic query!

Ontology (schema data):

```
dbo:Entomologist rdfs:subClassOf dbo:Scientist.
dbo:Medician rdfs:subClassOf dbo:Scientist.
dbo:Scientist rdfs:subClassOf dbo:Scientist.
dbo:Person rdfs:subClassOf dbo:Agent.
dbo:Organisation rdfs:subClassOf dbo:Agent.
dbo:birthPlace rdfs:domain dbo:Person .
```
Reasoning by Querying – Query Rewriting:

```
SELECT ?X WHERE
{
  { ?X a subclassOf* dbo:Scientist}
}
```

Instance data:

```
dbr:Marta_Grandi a dbo:Entomologist ;
  dbo:birthPlace dbr:Bologna .

dbr:Costanzo_Varolio a dbo:Medician;
  dbo:birthPlace dbr:Bologna .
```

Alternatively, the rules can be used “backwards” to rewrite the original query to yield a more generic query!

You can also use SPARQL 1.1 path expressions in this query rewriting! [1]
Reasoning by Querying – Query Rewriting:

SELECT {?X ?C1 ?C2}
WHERE { ?X a/subClassOf* ?C1;
    a/subClassOf* ?C2.

instance data:

dbr:Marta_Grandi a dbo:Entomologist ;
dbo:birthPlace dbr:Bologna .

dbr:Costanzo_Varolio a dbo:Medician;
dbo:birthPlace dbr:Bologna .

Ontology (schema data):

dbo:Entomologist rdfs:subClassOf dbo:Scientist.
dbo:Medician rdfs:subClassOf dbo:Scientist.
dbo:Scientist rdfs:subClassOf dbo:Person.
dbo:Person rdfs:subClassOf dbo:Agent.
dbo:Organisation rdfs:subClassOf dbo:Agent.
dbo:birthPlace rdfs:domain dbo:Person .
dbo:Organisation owl:disjointWith dbo:Place.

Similarly, RDFS and OWL QL inconsistency checking can be done by querying! [1] (simplified)
Querying and Reasoning over Contextualised graph data

- Example: Wikidata
- Reification techniques and Property Graphs
- Some of our own work in this space:
  - AnQL [3]
  - Querying temporal information: BEAR [4]
  - Stefan Bischof’s thesis work [5]


Often, you also need to deal with contextualized information

• E.g. from "Cities in the Italy with more than 1M population":

Structured queries (SPARQL):

PREFIX : <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT DISTINCT ?city ?pop WHERE {
  ?city a yago:City108524735 .
  ?city dbo:country :Italy .
  ?city dbo:populationTotal ?pop
  FILTER ( ?pop > 1000000 )
}

Doesn’t work!
Contextualised Information is better modeled in another Open Knowledge Graph: **Wikidata**

- Wikidata can also be queried as RDF with SPARQL!
Wikidata as RDF ... can be queried by SPARQL

• “Simple” surface query:

```
SELECT DISTINCT ?city WHERE {
  FILTER (?population > 1000000) }
```

• What’s this?
Wikidata as RDF ... can be queried by SPARQL

- However, Wikidata has more complex info: (temporal context, provenance,...)
  - Rome:
  - https://www.wikidata.org/wiki/Q220

... Can I query that with SPARQL? Yes!

What do we learn?
- Data and meta-data (context/provenance) at the same level → one RDF graph, mixing reification and plain data, cf. [Hernandez et al. 2015]
- Quite some Knowledge about the ontology required!
Reification/Property Graphs:

• How to (best) describe statements about triples in RDF is a bit open… various options, inter-translateable but affect performance of querying:

• Different Graph data models/Graph databases:

  • Labeled Directed graphs (plain RDF) - supported by RDF triple stores:

  ![Labeled Directed graph example](image)

  • Property graph – supported by Graph DBs, e.g. Neo4J, BlazeGraph, etc.:
Reification/Property Graphs:

• How to (best) describe statements about triples in RDF is a bit open... various options, inter-translatable but affect performance of querying:

  Annotated RDF [3]: :Rome :capitalOf :Italy. [1861, [  

  RDF reification: [ a rdf:Statement;  
                    rdf:subject :Rome;  
                    rdf:predicate :capitalOf;  

  “Named Graphs” :G1 { :Rome :capitalOf :Italy. }  
                 :G1 :yearBegins 1861 .

  “Singleton” properties: :Rome :p1 :Italy.  
                         :p1 :subPropertyOf :capitalOf;  
                         :yearBegins 1861 .

Other issues (time allowed)

- Federation, Path Queries over Linked Data [6,7,8,9]
- SPARQL for KG Construction (scalability issues) [10]
- Updates [11]
- Compressed & Queryable RDF-format HDT [12]


8. Vadim Savenkov, Qaiser Mehmood, Jürgen Umbrich, and Axel Polleres. Counting to k, or how SPARQL 1.1 could be efficiently enhanced with top k shortest path queries. In *13th International Conference on Semantic Systems (SEMANTiCS)*, pages 97--103, Amsterdam, the Netherlands, September 2017. ACM. [pdf](#)


Federation/Path Queries

Common problem in graphs, not doable with SPARQL, but with extensions [8]: “Give me the (k) shortest paths between two nodes?”

You can solve this by extending SPARQL [8] with bidirectional bfs over HDT [12]
https://bitbucket.org/vadim_savenkov/topk-pfn/

But how to do this effectively in a Federated setting? Open Research question!!!
Scalability of SPARQL endpoints?

Problem occurred for us when constructing another KG from Wikidata [10] You can solve this by:
1. extracting relevant triples to answer the query via HDT [12] and
2. executing targeted CONSTRUCT queries to the full SPARQL endpoint for specific sub-queries in order to materialize path expressions. Details cf. here
References:


8. Vadim Savenkov, Qaiser Mehmood, Jürgen Umbrich, and Axel Polleres. Counting to k, or how SPARQL 1.1 could be efficiently enhanced with top k shortest path queries. In *13th International Conference on Semantic Systems (SEMANTiCS)*, pages 97--103, Amsterdam, the Netherlands, September 2017. ACM. [pdf]


