Virtual Knowledge Graphs for Data Integration

Diego Calvanese

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

Department of Computing Science
Umeå University, Sweden

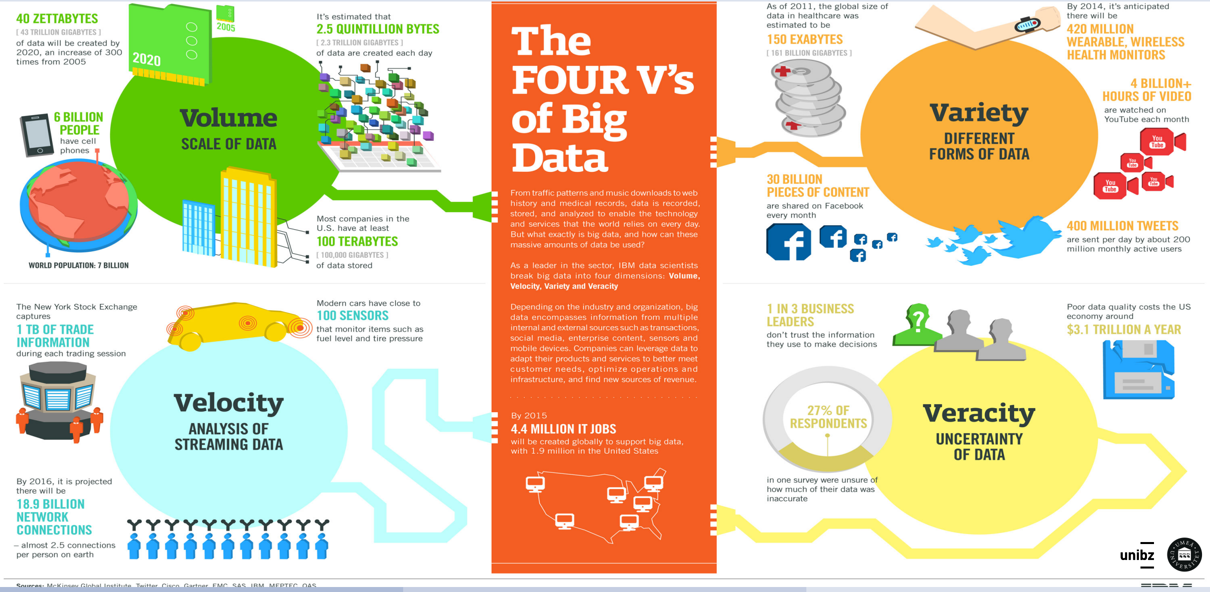
6th International Winter School on Big Data (BigDat 2020)

13–17 January 2020 – Ancona, Italy

Typical view of Big Data

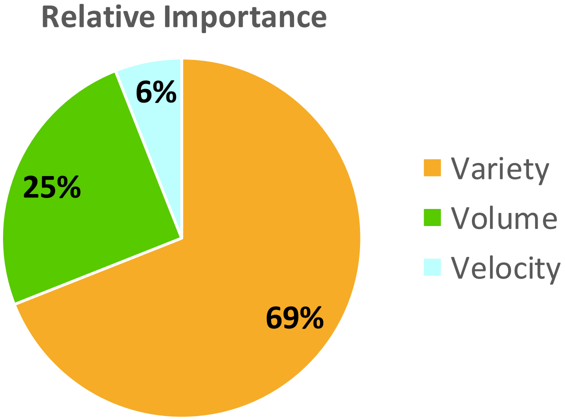
But data has a lot of structure

Challenges in the Big Data era



Variety, not volume, is driving Big Data initiatives

MIT Sloan Management Review (28 March 2016)



http://sloanreview.mit.edu/article/variety-not-volume-is-driving-big-data-initiatives/

How much time is spent searching for the right data?

Important problem: searching for data and establishing its quality

Example: in oil&gas, engineers spend 30–70% of their time on this (Crompton, 2008)

Challenge: Accessing heterogeneous data

Statoil (now Equinor) Exploration

Geologists at Statoil, prior to making decisions on drilling new wellbores, need to gather relevant information about previous drillings.

Slegge relational database:

- Terabytes of relational data
- 1,545 tables and 1727 views
- each with dozens of attributes
- consulted by 900 geologists

Problem: Translating information needs

Information need expressed by geologists

In my geographical area of interest, return all pressure data tagged with key stratigraphy information with understandable quality control attributes, and suitable for further filtering.

To obtain the answer, this needs to be translated into SQL

- main table for wellbores has 38 columns (with cryptic names)
- to obtain pressure data requires a 4-table join with two additional filters
- to obtain stratigraphic information requires a join with 5 more tables

Problem: Translating information needs

We would obtain the following SQL query:

```
SELECT WELLBORE.IDENTIFIER, PTY_PRESSURE.PTY_PRESSURE_S,
       STRATIGRAPHIC_ZONE.STRAT_COLUMN_IDENTIFIER, STRATIGRAPHIC_ZONE.STRAT_UNIT_IDENTIFIER
FROM WELLBORE,
     PTY_PRESSURE,
     ACTIVITY FP_DEPTH_DATA
        LEFT JOIN (PTY_LOCATION_1D FP_DEPTH_PT1_LOC
           INNER JOIN PICKED_STRATIGRAPHIC_ZONES ZS
              ON ZS.STRAT_ZONE_ENTRY_MD <= FP_DEPTH_PT1_LOC.DATA_VALUE_1_O AND
                 ZS.STRAT_ZONE_EXIT_MD >= FP_DEPTH_PT1_LOC.DATA_VALUE_1_O AND
                 ZS.STRAT_ZONE_DEPTH_UOM = FP_DEPTH_PT1_LOC.DATA_VALUE_1_OU
           INNER JOIN STRATIGRAPHIC_ZONE
              ON   ZS.WELLBORE = STRATIGRAPHIC_ZONE.WELLBORE AND
                 ZS.STRAT_COLUMN_IDENTIFIER = STRATIGRAPHIC_ZONE.STRAT_COLUMN_IDENTIFIER AND
                 ZS.STRAT_INTERP_VERSION = STRATIGRAPHIC_ZONE.STRAT_INTERP_VERSION   AND
                 ZS.STRAT_ZONE_IDENTIFIER = STRATIGRAPHIC_ZONE.STRAT_ZONE_IDENTIFIER)
           ON FP_DEPTH_DATA.FACILITY_S = ZS.WELLBORE AND
              FP_DEPTH_DATA.ACTIVITY_S  = FP_DEPTH_PT1_LOC.ACTIVITY_S,
     ACTIVITY_CLASS FORM_PRESSURE_CLASS
WHERE WELLBORE.WELLBORE_S = FP_DEPTH_DATA.FACILITY_S AND
      FP_DEPTH_DATA.ACTIVITY_S = PTY_PRESSURE.ACTIVITY_S AND
      FP_DEPTH_DATA.KIND_S = FORM_PRESSURE_CLASS.ACTIVITY_CLASS_S AND
      WELLBORE.REF_EXISTENCE_KIND = 'actual' AND
      FORM_PRESSURE_CLASS.NAME = 'formation pressure depth data'
```

Problem: Translating information needs

We would obtain the following SQL query:

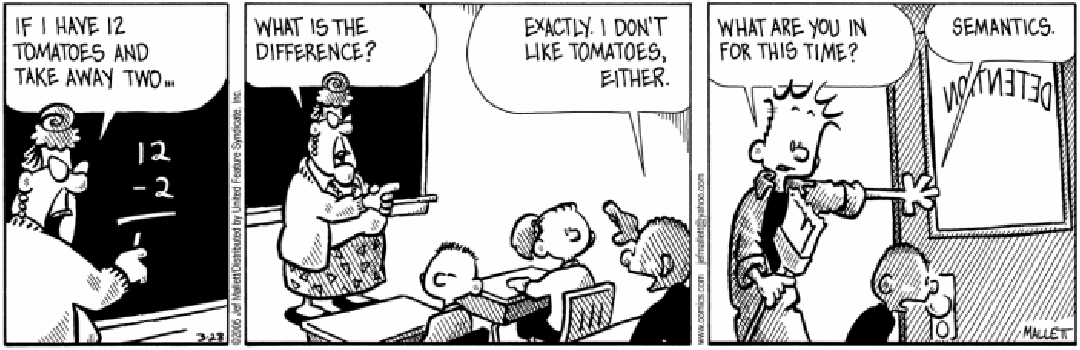
```
SELECT WELL
       STRA
FROM  WELLBO
      PTY_PR
      ACTIVI
         LEF

           INNER JOIN STRATIGRAPHIC_ZONE
              ON    ZS.WELLBORE = STRATIGRAPHIC_ZONE.WELLBORE AND

      ACTIVI
WHERE WELLE
       FP_DE
       FP_DEPTH_DATA.KIND_S = FORM_PRESSURE_CLASS.ACTIVITY_CLASS_S AND
       WELLBORE.REF_EXISTENCE_KIND = 'actual' AND
       FORM_PRESSURE_CLASS.NAME = 'formation pressure depth data'
```

This can be very time consuming, and requires knowledge of the domain of interest, a deep understanding of the database structure, and general IT expertise.

This is also very costly!

Equinor loses **50.000.000€** per year only due to this problem!!

Idea: Exploit semantics of data



FRAZZ: © Jeff Mallett/Dist. by United Feature Syndicate, Inc.

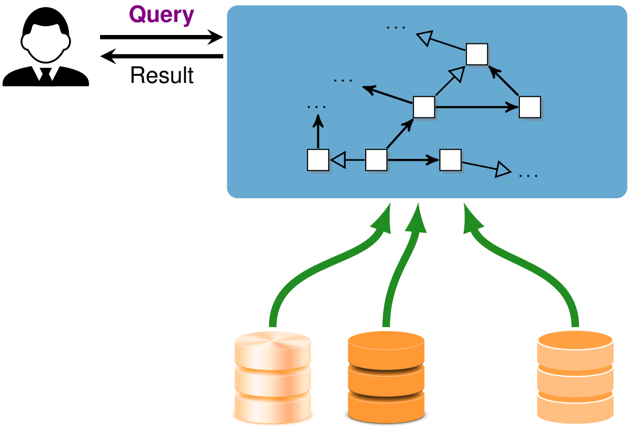
Spring 2015 issue of AI Magazine is devoted to Semantics for Big Data.

unibz

Outline

Outline

Solution: Virtual Knowedge Graphs (VKGs)



Ontology $\mathcal{O}$
conceptual view of data, convenient vocabulary

Mapping $\mathcal{M}$
how to populate the ontology from the data

Data Sources $\mathcal{S}$
autonomous and heterogeneous

Using an ontology makes it simpler for users to formulate their information needs, which are then automatically translated into a query over the data sources.
This approach is also known as **ontology-based data access** (OBDA).

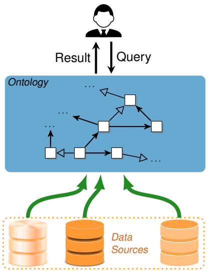
Challenges in VKGs for data access

In VKGs, the ontology exposes through the mapping a view of the underlying data in terms of a graph that stays virtual (i.e., is not materialized).

Such a setting poses significant challenges:

- How to instantiate the abstract framework?
- How to execute queries over the ontology by accessing data in the sources?
- How to address the expressivity – efficiency tradeoff?
- How to optimize performance with big data and large ontologies?

Incomplete information



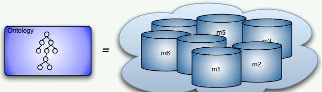
We are in a setting of **incomplete information**!!!

Incompleteness is introduced:

- by data sources, in general assumed to be incomplete;
- by domain constraints encoded in the ontology.

Plus:

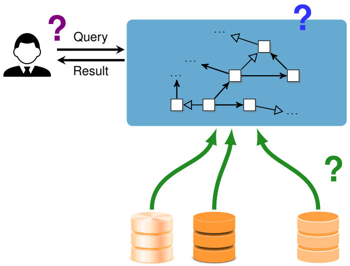
Ontologies are logical theories, and hence perfectly suited to deal with incomplete information!



Minus:

Query answering amounts to **logical inference**, and hence is significantly more challenging.

VKG framework – Which languages to use?



The choice of the right languages needs to take into account the tradeoff between expressive power and efficiency of query answering.

Note: We are in a setting where data plays a prominent role, so **efficiency with respect to the data** is the key factor.

The W3C has standardized languages that are suitable for VKGs:

1. Knowledge graph: expressed in **RDF** [W3C Rec. 2014] (v1.1)
2. Ontology $\mathcal{O}$: expressed in **OWL 2 QL** [W3C Rec. 2012]
3. Mapping $\mathcal{M}$: expressed in **R2RML** [W3C Rec. 2012]
4. Query: expressed in **SPARQL** [W3C Rec. 2013] (v1.1)

Outline

Outline

Resource Description Framework (RDF)

- RDF is a language standardized by the W3C for representing information [W3C Rec. 2004] (v1.0) and [W3C Rec. 2014] (v1.1).
- RDF is a **graph-based data model**, where information is represented as (labeled) nodes connected by (labeled) edges.
- Nodes have three different forms:
  - literal: denotes a constant value, with an associated datatype;
  - IRI (for *internationalized resource identifier*): denotes a resource (i.e., an object), for which the IRI acts as an identifier;
  - blank node: represents an anonymous object.
- And IRI might also denote a **property**, connecting an object to a literal, or connecting two objects.

See also `https://www.w3.org/TR/rdf11-concepts/` for details.

RDF triples

RDF provides a description of the domain of interest in terms of **triples**:

Subject | Predicate | Object

`<http://unibz.inf.di/data#person/2>`

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

`"John"^^xsd:string`

Triple elements: resources denoted by **global identifiers** (IRIs)

1. Subject: IRI of the described resource
2. Predicate: IRI of the property
3. Object: attribute value or IRI of another resource

Prefixes: useful abbreviations and/or references to external information

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix : <http://unibz.inf.di/data#>
@base <http://unibz.inf.di/>
```

RDF triples

RDF provides a description of the domain of interest in terms of **triples**:

Subject	Predicate	Object
`<:person/2>`	`foaf:name`	`"John"^^xsd:string`

Triple elements: resources denoted by **global identifiers** (IRIs)

1. Subject: IRI of the described resource
2. Predicate: IRI of the property
3. Object: attribute value or IRI of another resource

Prefixes: useful abbreviations and/or references to external information

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix : <http://unibz.inf.di/data#>
@base <http://unibz.inf.di/>
```

RDF triples

RDF provides a description of the domain of interest in terms of **triples**:

Subject → `<data#person/2>`

Predicate → `foaf:name`

Object → `"John"^^xsd:string`

Triple elements: resources denoted by global identifiers (IRIs)

1. Subject: IRI of the described resource
2. Predicate: IRI of the property
3. Object: attribute value or IRI of another resource

Prefixes: useful abbreviations and/or references to external information

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix : <http://unibz.inf.di/data#>
@base <http://unibz.inf.di/>
```

RDF – Examples

Class membership:

Fact	Prof(*uni2/person/1*)
RDF triple	`<uni2/person/1> rdf:type :Prof`

Note: This is typically abbreviated as

RDF triple	`<uni2/person/1> a :Prof`

Attribute of an individual:

Fact	lastName(*uni2/person/1*, "Lane")
RDF triple	`<uni2/person/1> :lastName "Lane"`

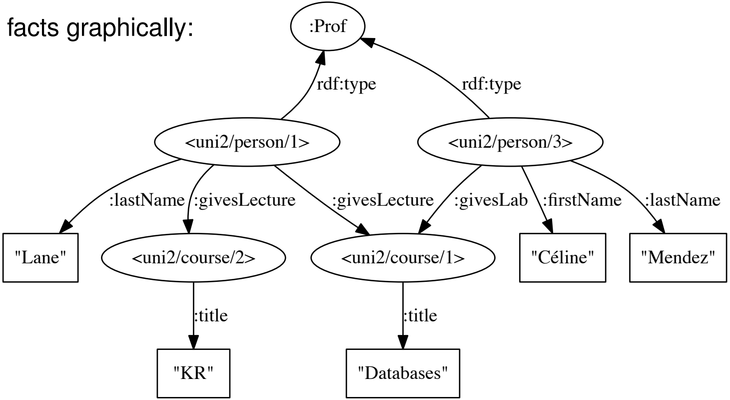
Property of an individual:

Fact	givesLecture(*uni2/person/1*, *uni2/course/2*)
RDF triple	`<uni2/person/1> :givesLecture <uni2/course/2>`

RDF graph – Example

```
<uni2/person/1> rdf:type :Prof
<uni2/person/1> foaf:lastName "Lane"
<uni2/person/1> :givesLecture <uni2/course/1>
...
```

We can represent such a set of facts graphically:



RDF datatypes

- Datatypes are used with RDF literals to represent values such as strings, numbers, and dates.
- Each datatype is itself denoted by an IRI. E.g., the XML Schema built-in datatypes have IRIs of the form `http://www.w3.org/2001/XMLSchema#xxx`
- Each datatype associates to elements in a **lexical space** (i.e., unicode strings) elements from a **value space**.
  Example:
  - datatype: `xsd:boolean`
  - lexical space: { "`true`", "`false`", "`1`", "`0`" }
  - value space: $\{true, false\}$
- To explicitly associate a datatype to a literal, we use the notation *literal^^datatype*.
  Example: `12.5^^xsd:double`, `1^^xsd:integer`

XML Schema built-in datatypes (recommended)

	Datatype	Value space (informative)
Core types	`xsd:string`	Character strings
	`xsd:boolean`	true, false
	`xsd:decimal`	Arbitrary-precision decimal numbers
	`xsd:integer`	Arbitrary-size integer numbers
IEEE floating-point numbers	`xsd:float`	32-bit floating point numbers incl. ±Inf, ±0, NaN
	`xsd:double`	64-bit floating point numbers incl. ±Inf, ±0, NaN
Time and date	`xsd:date`	Dates (yyyy-mm-dd) with or without timezone
	`xsd:time`	Times (hh:mm:ss.sss. . .) with or without timezone
	`xsd:datetime`	Date and time with or without timezone
Limited-range integer numbers	`xsd:byte`	8 bit integers (-128, . . . , +127)
	`xsd:short`	16 bit integers
	`xsd:int`	32 bit integers
	`xsd:long`	64 bit integers
	`xsd:unsignedByte`	8 bit non-negative integers (0, . . . , 255)
	`xsd:unsignedShort`	16 bit non-negative integers
	. . .	

Additional RDF features

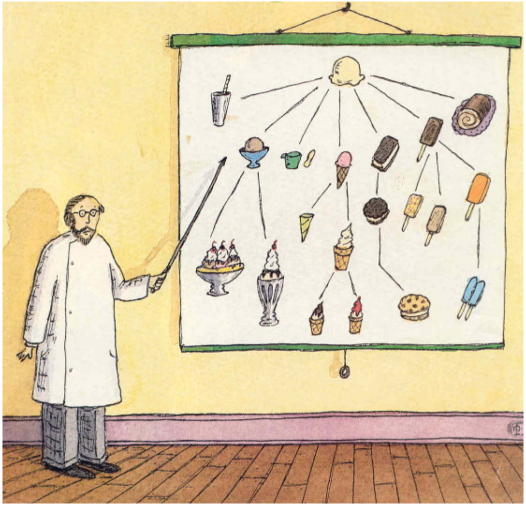
RDF has additional features that we do not cover here:

- blank nodes
- named graphs

Outline

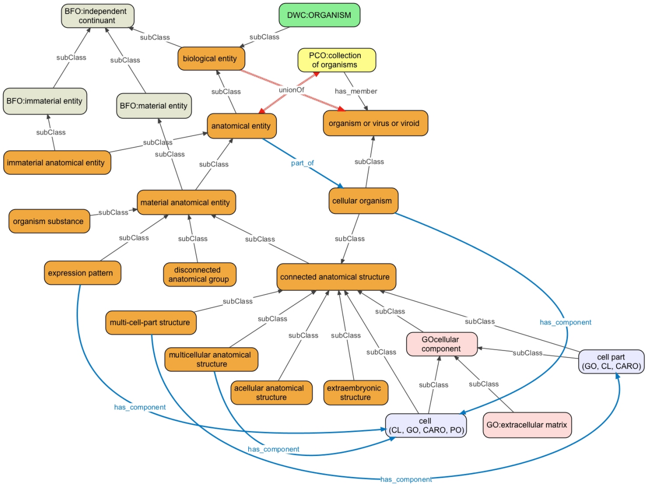
What is an ontology?

- An ontology conceptualizes a domain of interest in terms of **concepts/classes**, (binary) **relations**, and their **properties**.
- It typically organizes the concepts in a hierarchical structure.
- Ontologies are often represented as graphs.
- However, an ontology is actually a **logical theory**, expressed in a suitable fragment of first-order logic, or better, in **description logics**.



What is an ontology?

- An ontology conceptualizes a domain of interest in terms of **concepts/classes**, (binary) **relations**, and their **properties**.
- It typically organizes the concepts in a hierarchical structure.
- Ontologies are often represented as graphs.
- However, an ontology is actually a **logical theory**, expressed in a suitable fragment of first-order logic, or better, in **description logics**.



What is an ontology?

- An ontology conceptualizes a domain of interest in terms of **concepts/classes**, (binary) **relations**, and their **properties**.
- It typically organizes the concepts in a hierarchical structure.
- Ontologies are often represented as graphs.
- However, an ontology is actually a **logical theory**, expressed in a suitable fragment of first-order logic, or better, in **description logics**.

$\forall x.\, \mathsf{Actor}(x) \rightarrow \mathsf{Staff}(x)$
$\forall x.\, \mathsf{SeriesActor}(x) \rightarrow \mathsf{Actor}(x)$
$\forall x.\, \mathsf{MovieActor}(x) \rightarrow \mathsf{Actor}(x)$
$\forall x.\, \mathsf{SeriesActor}(x) \rightarrow \neg\mathsf{MovieActor}(x)$

$\forall x.\, \mathsf{Staff}(x) \rightarrow \exists y.\, \mathit{ssn}(x, y)$
$\forall y.\, \exists x.\, \mathit{ssn}(x, y) \rightarrow \mathtt{xsd:int}(y)$
$\forall x, y, y'.\, \mathit{ssn}(x, y) \wedge \mathit{ssn}(x, y') \rightarrow y = y'$

$\forall x.\, \exists y.\, \mathsf{playsIn}(x, y) \rightarrow \mathsf{MovieActor}(y)$
$\forall y.\, \exists x.\, \mathsf{playsIn}(x, y) \rightarrow \mathsf{Movie}(x)$
$\forall x.\, \mathsf{MovieActor}(x) \rightarrow \exists y.\, \mathsf{playsIn}(x, y)$
$\forall x.\, \mathsf{Movie}(x) \rightarrow \exists y.\, \mathsf{playsIn}(y, x)$
$\forall x, y.\, \mathsf{playsIn}(x, y) \rightarrow \mathsf{actsIn}(x, y)$
$\ldots$

What is an ontology?

- An ontology conceptualizes a domain of interest in terms of **concepts/classes**, (binary) **relations**, and their **properties**.
- It typically organizes the concepts in a hierarchical structure.
- Ontologies are often represented as graphs.
- However, an ontology is actually a **logical theory**, expressed in a suitable fragment of first-order logic, or better, in **description logics**.

$$
\begin{array}{rcl}
\mathsf{Actor} & \sqsubseteq & \mathsf{Staff} \\
\mathsf{SeriesActor} & \sqsubseteq & \mathsf{Actor} \\
\mathsf{MovieActor} & \sqsubseteq & \mathsf{Actor} \\
\mathsf{SeriesActor} & \sqsubseteq & \neg\mathsf{MovieActor} \\[1ex]
\mathsf{Staff} & \sqsubseteq & \exists \mathit{ssn} \\
\exists \mathit{ssn}^- & \sqsubseteq & \mathtt{xsd{:}int} \\
\multicolumn{3}{c}{(\mathbf{funct}\ \mathit{ssn})} \\[1ex]
\exists \mathsf{playsIn} & \sqsubseteq & \mathsf{MovieActor} \\
\exists \mathsf{playsIn}^- & \sqsubseteq & \mathsf{Movie} \\
\mathsf{MovieActor} & \sqsubseteq & \exists \mathsf{playsIn} \\
\mathsf{Movie} & \sqsubseteq & \exists \mathsf{playsIn}^- \\
\mathsf{playsIn} & \sqsubseteq & \mathsf{actsIn} \\
 & \cdots &
\end{array}
$$

The OWL 2 QL ontology language

- **OWL 2 QL** is one of the three standard profiles of OWL 2. [W3C Rec. 2012]
- Derived from the *DL-Lite*$_{\mathcal{R}}$ description logic [Baader et al. 2003] of the *DL-Lite*-family:
  - Groups the domain into classes of objects with common properties.
  - Binary relations between objects are called object properties.
  - Binary relations from objects to values are called data properties.
- Is considered a lightweight ontology language:
  - controlled expressive power
  - efficient inference
- Optimized for accessing large amounts of data
  - Queries over the ontology can be rewritten into SQL queries over the underlying relational database (First-order rewritability).
  - Consistency of ontology and data can also be checked by executing SQL queries.

OWL 2 QL ontologies

- An OWL 2 QL ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ consists of:
  - a so-called **TBox** (for terminological box) $\mathcal{T}$, modeling the schema level information (i.e., axioms), and
  - a so-called **ABox** (for assertional box) $\mathcal{A}$, modeling the extensional level information (i.e., facts).
- In the VKG setting, the ABox is (usually) implicitly defined through the database(s) and the mappings.
- Therefore, in the following, we use the term "ontology" to refer to the TBox only.

RDF Schema (RDFS)

Class hierarchy: `rdfs:subClassOf` $(C_1 \sqsubseteq C_2)$

Example: `:MovieActor rdfs:subClassOf :Actor .`

Inference: `<person/2> rdf:type :MovieActor .`

$\Longrightarrow$ `<person/2> rdf:type :Actor .`

Property hierarchy: `rdfs:subPropertyOf` $(P_1 \sqsubseteq P_2)$

Example: `:playsIn rdfs:subPropertyOf :actsIn .`

Inference: `<person/2> :playsIn <movie/3> .`

$\Longrightarrow$ `<person/2> :actsIn <movie/3> .`

Domain of properties: `rdfs:domain` $(\exists P \sqsubseteq C_1)$

Example: `:playsIn rdfs:domain :MovieActor .`

Inference: `<person/2> :playsIn <movie/3> .`

$\Longrightarrow$ `<person/2> rdf:type :MovieActor .`

Range of properties: `rdfs:range` $(\exists P^- \sqsubseteq C_2)$

Example: `:playsIn rdfs:range :Movie .`

Inference: `<person/2> :playsIn <movie/3> .`

$\Longrightarrow$ `<movie/3> rdf:type :Movie .`

Other constructs of OWL 2 QL (1/2)

Inverse properties: `owl:inverseOf` $(P_1 \sqsubseteq P_2^- \text{ and } P_2 \sqsubseteq P_1^-)$

Example: `:actsIn owl:inverseOf :hasActor .`

Inference: `<person/2> :actsIn <movie/3> .`

$\Longrightarrow$ `<movie/3> :hasActor <person/2> .`

Mandatory participation: `owl:someValuesFrom` in the superclass expression $(C_1 \sqsubseteq \exists P.C_2)$

Example:

```
:SeriesActor rdfs:subClassOf
    [ rdf:type owl:Restriction ;
      owl:onProperty :actsIn ;
      owl:someValuesFrom  :Series ] .
```

Inference: `<person/5> rdf:type :SeriesActor .`

$\Longrightarrow$

```
<person/5> rdfs:type
  [ rdfs:type owl:Restriction ;
    owl:onProperty :actsIn ;
    owl:someValuesFrom :Series ] .
```

Other constructs of OWL 2 QL (2/2)

Class disjointness: `owl:disjointWith` $(C_1 \sqsubseteq \neg C_2)$

Example: `:Actor owl:disjointWith :Movie .`

Inference:

```
<uni1/person/2> rdf:type :Actor  .
<uni1/person/2> rdf:type :Movie  .
```

$\Longrightarrow$ Inconsistent RDF graph

Semantics of an OWL 2 QL ontology

The **formal semantics** of OWL 2 QL is given in terms of first-order interpretations.

An **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of:

- a nonempty set $\Delta^{\mathcal{I}}$, called the interpretation domain (of $\mathcal{I}$), and
- an interpretation function $\cdot^{\mathcal{I}}$, which maps
  - each class nane A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$
  - each property name P to a subset $P^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
- The interpretation function is then extended to cover the OWL 2 QL constructs:

$$(P^-)^{\mathcal{I}} = \{(y, x) \mid (x, y) \in P^{\mathcal{I}}\} \qquad \exists P^{\mathcal{I}} = \{x \mid \text{there is some } y \text{ such that } (x, y) \in P^{\mathcal{I}}\}$$

The semantics of an ontology is given by specifying when $\mathcal{I}$ **satisfies** an assertion α, denoted $\mathcal{I} \models \alpha$:

$$\mathcal{I} \models C_1 \sqsubseteq C_2 \quad \text{if} \quad C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}; \qquad \mathcal{I} \models R_1 \sqsubseteq R_2 \quad \text{if} \quad R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}};$$

$\mathcal{I}$ satisfies an ABox fact, if the fact holds in $\mathcal{I}$.

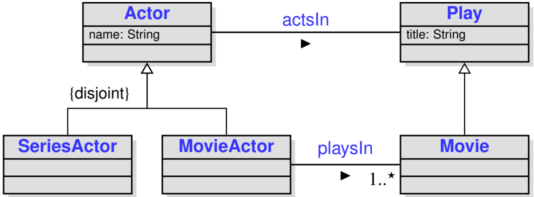
An interpretation that satisfies all assertions of the ontology, is called a **model** of the ontology.

Representing OWL 2 QL ontologies as UML class diagrams/ER schemas

There is a close correspondence between OWL 2 QL and conceptual modeling formalisms [Berardi et al. 2005; Bergamaschi and Sartori 1992; Borgida 1995; Borgida and Brachman 2003; C., Lenzerini, et al. 1999; Lenzerini and Nobili 1990; Queralt et al. 2012].

$\mathsf{SeriesActor} \sqsubseteq \mathsf{Actor}$		subclass
$\mathsf{SeriesActor} \sqsubseteq \neg\mathsf{MovieActor}$		disjointness
$\exists\mathsf{actsIn} \sqsubseteq \mathsf{Actor}$		domain
$\exists\mathsf{actsIn}^- \sqsubseteq \mathsf{Play}$		range
$\mathsf{MovieActor} \sqsubseteq \exists\mathsf{playsIn}$		mandatory participation
$\mathsf{playsIn} \sqsubseteq \mathsf{actsIn}$		sub-association
$\dots$		

An OWL 2 QL ontology can be visualized naturally as a UML class diagram or as an ER schema.



Capturing UML class diagrams/ER schemas in OWL 2 QL

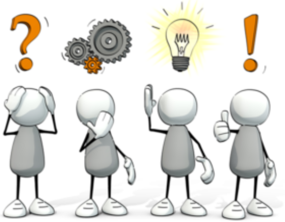
Modeling construct	*DL-Lite*	**FOL formalization**
ISA on classes	$A_1 \sqsubseteq A_2$	$\forall x(A_1(x) \rightarrow A_2(x))$
... and on relations	$R_1 \sqsubseteq R_2$	$\forall x, y(R_1(x, y) \rightarrow R_2(x, y))$
Disjointness of classes	$A_1 \sqsubseteq \neg A_2$	$\forall x(A_1(x) \rightarrow \neg A_2(x))$
... and of relations	$R_1 \sqsubseteq \neg R_2$	$\forall x, y(R_1(x, y) \rightarrow \neg R_2(x, y))$
Domain of relations	$\exists P \sqsubseteq A_1$	$\forall x(\exists y(P(x, y)) \rightarrow A_1(x))$
Range of relations	$\exists P^- \sqsubseteq A_2$	$\forall x(\exists y(P(y, x)) \rightarrow A_2(x))$
Mandatory participation *(min card = 1)*	$A_1 \sqsubseteq \exists P$ $A_2 \sqsubseteq \exists P^-$	$\forall x(A_1(x) \rightarrow \exists y(P(x, y)))$ $\forall x(A_2(x) \rightarrow \exists y(P(y, x)))$

OWL 2 QL/ *DL-Lite*$_\mathcal{R}$ **cannot capture**:

- covering constraints – This would require **disjunction**.
- identity between individuals – This would `owl:sameAs`.
- functionality of roles – This would require number restrictions.

Outline

Query answering – Which query language to use



Querying under incomplete information

Query answering is not simply query evaluation, but a form of logical inference, and requires reasoning.

Two borderline cases for choosing the language for querying ontologies:

1. Use the **ontology language** as query language.
   - Ontology languages are tailored for capturing intensional relationships.
   - They are quite poor as query languages.
2. Use **Full SQL** (or equivalently, first-order logic).
   - Problem: in a setting with incomplete information, query answering is undecidable (FOL validity).

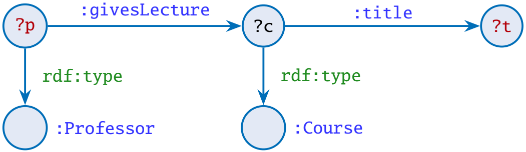
Conjunctive queries – Are concretely represented in SPARQL

A good tradeoff is to use conjunctive queries (CQs) or unions of CQs (UCQs), corresponding to SQL/relational algebra (union) select-project-join queries.

SPARQL query language

- Is the standard query language for RDF data. [W3C Rec. 2008, 2013]
- Core query mechanism is based on **graph matching**.

```
SELECT ?p ?t
WHERE { ?p rdf:type :Professor .
        ?p :givesLecture ?c .
        ?c rdf:type :Course .
        ?c :title ?t
      }
```



Additional language features (SPARQL 1.1):

- `UNION`: matches one of alternative graph patterns
- `OPTIONAL`: produces a match even when part of the pattern is missing
- complex `FILTER` conditions
- `GROUP BY`, to express aggregations
- `MINUS`, to remove possible solutions
- property paths (regular expressions)
- ...

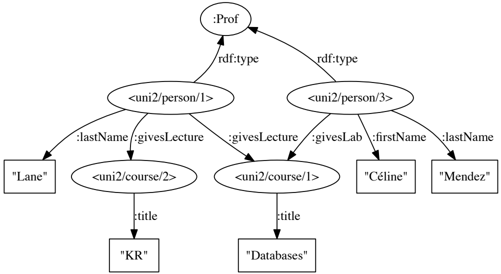
SPARQL Basic Graph Patterns (BGPs)

Basic Graph Patterns are the simplest form of SPARQL query, asking for a pattern in the RDF graph.

Example: BGP

```
SELECT ?p ?ln ?c ?t
WHERE {
  ?p :lastName ?ln .
  ?p :givesLecture ?c .
  ?c :title ?t .
}
```

When evaluated over the RDF graph



… the query returns:

p	ln	c	t
<uni2/person/1>	"Lane"	<uni2/course/1>	"Databases"
<uni2/person/1>	"Lane"	<uni2/course/2>	"KR"

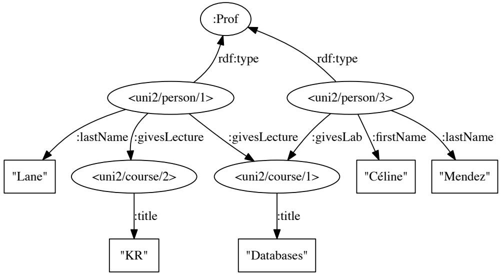
Projecting out variables in a SPARQL query

A query may also return only a subset of the variables used in the BGP.

Example: BGP with projection

```
SELECT ?ln ?t
WHERE {
  ?p :lastName ?ln .
  ?p :givesLecture ?c .
  ?c :title ?t .
}
```

When evaluated over the RDF graph



. . . the query returns:

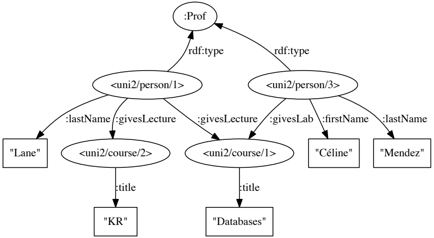
ln	t
"Lane"	"Databases"
"Lane"	"KR"

Union of Basic Graph Patterns

Example: BGPs with UNION

```
SELECT ?p ?ln ?c
WHERE {
  { ?p :lastName ?ln .  ?p :givesLecture ?c . }
  UNION
  { ?p :lastName ?ln .  ?p :givesLab ?c . }
}
```

When evaluated over



. . . the query returns:

p	ln	c
<uni2/person/1>	"Lane"	<uni2/course/1>
<uni2/person/1>	"Lane"	<uni2/course/2>
<uni2/person/3>	"Mendez"	<uni2/course/1>

BGPs vs. conjunctive queries

We can write queries using a more compact and abstract syntax, borrowed from database theory.

Example: BGP

```
SELECT ?p ?ln ?c ?t
WHERE {
  ?p :lastName ?ln .
  ?p :givesLecture ?c .
  ?c :title ?t .
}
```

vs. conjunctive query

$$\boldsymbol{q}(p, ln, c, t) \leftarrow \text{lastName}(p, ln), \text{givesLecture}(p, c), \text{title}(c, t)$$

A conjunctive query $\boldsymbol{q}$ has the form $\boldsymbol{q}(\vec{x}) \leftarrow p_1(\vec{y}_1), \ldots, p(\vec{y}_k)$ **where**

- $\boldsymbol{q}(\vec{x})$ is called the head of $\boldsymbol{q}$,
- $p_1(\vec{y}_1), \ldots, p(\vec{y}_k)$ is a conjunction of atoms called the body of $\boldsymbol{q}$,
- all variables $\vec{x}$ in the head are among $\vec{y}_1, \ldots, \vec{y}_k$, and
- the variables in $\vec{y}_1, \ldots, \vec{y}_k$ that are not among $\vec{x}$ are existentially quantified.

BGPs vs. conjunctive queries (cont.)

Example: BGP with projection

```
SELECT ?ln ?t
WHERE {
  ?p :lastName ?ln .
  ?p :givesLecture ?c .
  ?c :title ?t .
}
```

vs. conjunctive query

$$q(ln, t) \leftarrow \text{lastName}(p, ln), \text{givesLecture}(p, c), \text{title}(c, t)$$

But there is a difference in semantics when we have an ontology:

- In a SPARQL query, all variables, including those that are projected out, must match nodes of the RDF graph.
- In a conjunctive query, the existentially quantified variables can also match nodes that are existentially implied by the axioms of the ontology.

SPARQL UNION vs. unions of CQs

Example: BGP with UNION

```
SELECT ?p ?ln ?c
WHERE {
  { ?p :lastName ?ln .
    ?p :givesLecture ?c .
  }
  UNION
  { ?p :lastName ?ln .
    ?p :givesLab ?c .
  }
}
```

vs. union of CQs (UCQ)

$$\boldsymbol{q}(p, ln, c) \leftarrow \mathsf{lastName}(p, ln), \mathsf{givesLecture}(p, c)$$
$$\boldsymbol{q}(p, ln, c) \leftarrow \mathsf{lastName}(p, ln), \mathsf{givesLab}(p, c)$$

A UCQ is written as a set of CQs, all with the same head.

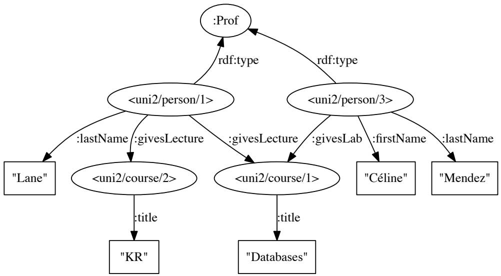
Extending BGPs with OPTIONAL

We might want to add information when available, but **not reject** a solution **when some part of the query does not match**.

Example: BGP with OPTIONAL

```
SELECT ?p ?fn ?ln
WHERE {
  ?p :lastName ?ln .
  OPTIONAL {
    ?p :firstName ?fn .
  }
}
```

When evaluated over the RDF graph



. . . the query returns:

p	fn	ln
<uml2/person/1>		"Lane"
<uml2/person/3>	"Céline"	"Mendez"

SPARQL algebra

We have seen the following features of the SPARQL algebra:

- Basic Graph Patterns
- `UNION`
- `OPTIONAL`

The overall algebra has additional features:

- more complex `FILTER` conditions
- `GROUP BY`, to express aggregations and support aggregation operators
- `MINUS`, to remove possible solutions
- `FILTER NOT EXISTS`, to test for the absence of a pattern

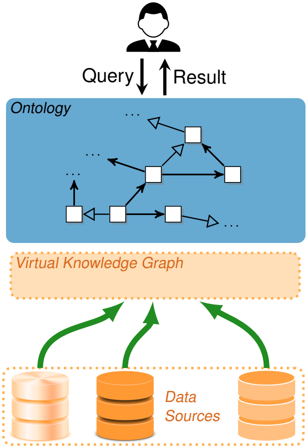
Outline

Use of mappings

In VKGs, the **mapping** $\mathcal{M}$ encodes how the data $\mathcal{D}$ in the sources should be used to create the virtual knowledge graph.

Virtual knowledge graph $\mathcal{V} = \mathcal{M}(\mathcal{D})$

- $\mathcal{V}$ is defined in terms of $\mathcal{M}$ and $\mathcal{D}$.
- Queries are answered with respect to $\mathcal{O}$ and $\mathcal{V}$.
- The data of $\mathcal{V}$ is not materialized (it is virtual!).
- Instead, the information in $\mathcal{O}$ and $\mathcal{M}$ is used to translate queries over $\mathcal{O}$ into queries formulated over the sources.
- Advantage, compared to materialization: the graph is **always up to date** w.r.t. the data sources.



Mismatch between data layer and ontology

Impedance mismatch

- Relational databases store values.
- Ontologies represent both objects and values.

We need to construct the ontology objects from the database values.

Proposed solution

The specification of **how to construct the ontology objects** that populate the virtual data layer from the database values **is embedded in the mapping** between the data sources and the ontology.

Mapping language

The mapping consists of a set of statements of the form

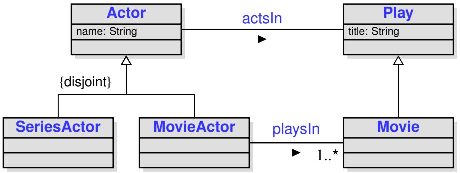
SQL Query $\rightsquigarrow$ Class and Property Membership Assertions

To address the impedance mismatch

In the right-hand side of the mapping, we make use of **iri-templates**, which transform database values into object identifiers (IRIs).

Mapping language – Example

Ontology $\mathcal{O}$:



Mapping $\mathcal{M}$:

m_1: `SELECT mcode, mtitle FROM MOVIE`
`WHERE type = "m"`
$\leadsto$ Movie(**iri**("`pl-`", mcode)),
title(**iri**("`pl-`", mcode), mtitle)

m_2: `SELECT M.mcode, A.acode FROM MOVIE M, ACTOR A`
`WHERE M.mcode = A.pcode AND M.type = "movie"`
$\leadsto$ playsIn(**iri**("`act-`", acode), **iri**("`pl-`", mcode)), ...

Database $\mathcal{D}$:

MOVIE				
mcode	*mtitle*	*myear*	*type*	...
5118	The Matrix	1999	m	...
8234	Altered Carbon	2018	s	...
2281	Blade Runner	1982	m	...

ACTOR			
pcode	*acode*	*aname*	...
5118	438	K. Reeves	...
5118	572	C.A. Moss	...
2281	271	H. Ford	...

The mapping $\mathcal{M}$ applied to database $\mathcal{D}$ generates the virtual knowledge graph $\mathcal{M}(\mathcal{D})$:

Movie(`pl-5118`), title(`pl-5118`, "`The Matrix`")
Movie(`pl-2281`), title(`pl-2281`, "`Blade Runner`")
playsIn(`act-438`, `pl-5118`), playsIn(`act-572`, `pl-5118`), playsIn(`act-271`, `pl-2281`), ...

Concrete mapping languages

Several proposals for concrete languages to map a relational DB to an ontology:

- They assume that the ontology is populated in terms of RDF triples.
- Some template mechanism is used to specify the triples to instantiate.

Examples: D2RQ[1], SML[2], Ontop[3]

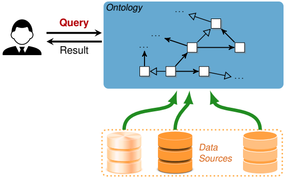
R2RML

- Most popular RDB to RDF mapping language
- W3C Recommendation 27 Sep. 2012, `http://www.w3.org/TR/r2rml/`
- R2RML mappings are themselves expressed as RDF graphs and written in Turtle syntax.

[1] http://d2rq.org/d2rq-language
[2] http://sparqlify.org/wiki/Sparqlification_mapping_language
[3] https://github.com/ontop/ontop/wiki/ontopOBDAModel#Mapping_axioms

Outline

VKGs: Formalization



To formalize VKGs, we distinguish between the intensional and the extensional level information.

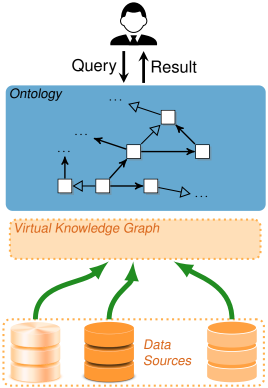
A **VKG specification** is a triple $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, where:

- $\mathcal{O}$ is an ontology (expressed in OWL 2 QL),
- $\mathcal{S}$ is a (possibly federated) relational database schema for the data sources, possibly with integrity constraints,
- $\mathcal{M}$ is a set of (R2RML) mapping assertions between $\mathcal{O}$ and $\mathcal{S}$.

A **VKG instance** is a pair $\langle \mathcal{P}, \mathcal{D} \rangle$, where

- $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$ is a VKG specification, and
- $\mathcal{D}$ is a (possibly federated) relational database compliant with $\mathcal{S}$.

Semantics of VKGs



Remember: The mapping $\mathcal{M}$ generates from the data $\mathcal{D}$ in the sources a **virtual knowledge graph** $\mathcal{V} = \mathcal{M}(\mathcal{D})$.

A first-order interpretation $\mathcal{I}$ of the ontology predicates is a **model** of a VKG instance $\langle \mathcal{P}, \mathcal{D} \rangle$, where $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$, if

- it satisfies all axioms in $\mathcal{O}$, and
- contains all facts in $\mathcal{M}(\mathcal{D})$, i.e., retrieved through $\mathcal{M}$ from $\mathcal{D}$.

Note:

- In general, $\langle \mathcal{P}, \mathcal{D} \rangle$ has infinitely many models, and some of these might be infinite.
- However, for query answering, we do not need to compute such models.

Query answering in VKGs – Certain answers

In VKGs, we want to answer queries formulated over the ontology, by using the data provided by the data sources through the mapping.

Consider our formalization of VKGs and a VKG instance $\mathcal{J} = \langle \mathcal{P}, \mathcal{D} \rangle$.

Certain answers

Given a VKG instance $\mathcal{J}$ and a query q over $\mathcal{J}$, the certain answers to q are those answers that hold in all models of $\mathcal{J}$.

First-order rewritability

To make computing certain answers viable in practice, the VKG setting relies on reducing it to evaluating SQL (i.e., first-order logic) queries over the data.

Consider a VKG specification $\mathcal{P} = \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle$.

First-order rewritability

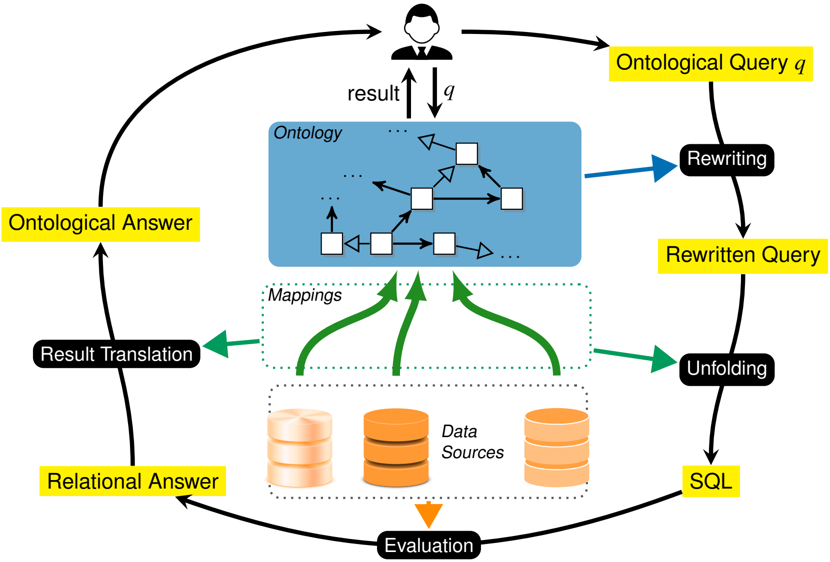
A query $r(\vec{x})$ is a **first-order rewriting** of a query $q(\vec{x})$ with respect to $\mathcal{P}$ if, for every source DB $\mathcal{D}$:

certain answers to $q(\vec{x})$ over $\langle \mathcal{P}, \mathcal{D} \rangle$ = answers to $r(\vec{x})$ over $\mathcal{D}$.

For OWL 2 QL ontologies and R2RML mappings,
(core) SPARQL queries are first-order rewritable.

In other words, **in VKGs, we can compute the certain answers to a SPARQL query by evaluating over the sources its rewriting, which is an SQL query.**

Query answering via query reformulation – Conceptual framework



Outline

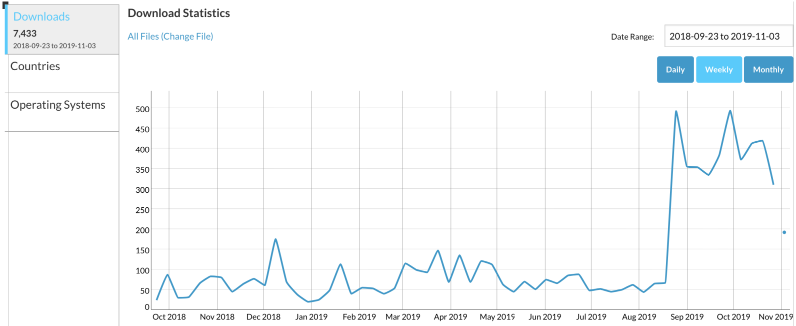
Implementations of VKG systems

- Mastro [C., De Giacomo, Lembo, Lenzerini, Poggi, Rodriguez-Muro, Rosati, et al. 2011] [4], Sapienza Università di Roma & OBDA systems SRL, Italy
- Morph [Priyatna et al. 2014] [5], Technical University of Madrid, Spain
- Ontop [C., Cogrel, et al. 2017][6], Free University of Bolzano, Italy
- Stardog[7], Stardog Union, US
- Ultrawrap [Sequeda and Miranker 2013] [8], Capsenta, US
- Oracle Spatial and Graph RDF Semantic Graph [9]

---

[4] http://www.obdasystems.com/it/mastro
[5] https://github.com/oeg-upm/morph-rdb
[6] http://ontop.inf.unibz.it
[7] http://www.stardog.com
[8] https://capsenta.com/ultrawrap
[9] http://www.oracle.com/technetwork/database/options/spatialandgraph

- *Ontop* is a VKG platform
- It supports all major database engines (e.g., Oracle, DB2, MS SQL Server, PostgreSQL, MySQL).
- Open source under Apache 2 License



Ontopic SrL

- First spin-off of the Free University of Bozen-Bolzano.
- Incorporated in April 2019.
- Product: Ontopic suite based on the *Ontop* engine.
- Services around *Ontop* and the Ontopic suite.

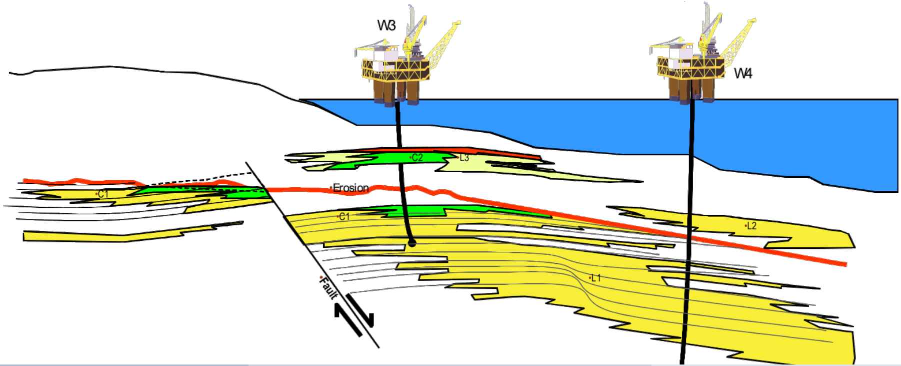
Use cases

- Oil & Gas: Statoil [Kharlamov, Hovland, et al. 2017]
- Turbine Diagnoses: Siemens [Kharlamov, Mailis, et al. 2017]
- Cultural heritage: EPNet project [C., Liuzzo, et al. 2016]
- Maritime security: EMSec project [Brüggemann et al. 2016]
- Manufacturing: case study [Petersen et al. 2017]
- Health care: electronic health records [Rahimi et al. 2014]
- Public debt: Italian Ministry of Economy and Finance [Antonioli et al. 2014]
- Smart cities: IBM Ireland [Lopez et al. 2015]
- Open data publishing: NOI Bolzano
- Development of data integration solutions: SIRIS Academic SL Barcelona

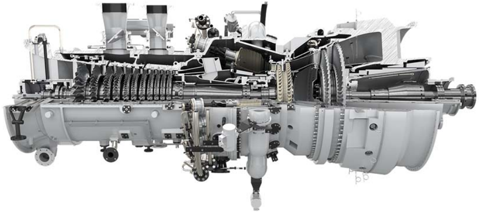
. . . a survey on systems and use cases [Xiao, Ding, et al. 2019]

Statoil (now Equinor) [Kharlamov, Hovland, et al. 2017]

- Statoil (now Equinor) is Norway's largest (oil and gas) company. Statoil has been a use case partner in the EU project Optique.
- Exploration domain: analyse existing relevant data in order to find exploitable accumulations of oil or gas.
- Improve the efficiency of the information gathering routine for geologists.
- Efficient, creative data collection from multiple large volume data sources.



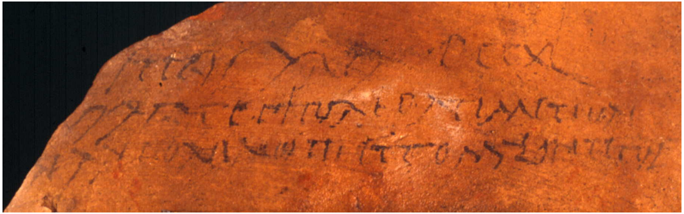
Siemens Energy Services [Kharlamov, Mailis, et al. 2017]



- Use case partner in the EU project Optique.
- Siemens produces huge appliances
  (e.g., gas turbines) and installs them in plants.
- Siemens service centers:
  - over 50 service centers world-wide
  - each center is responsible for several thousands of appliances
  - offer constant monitoring and diagnostics services
- Monitoring and diagnostics tasks
  - reactive and preventive diagnostics: offline, after an issue is detected
  - predictive analyses: real-time, to avoid issues while appliance is functioning

EPNet project [C., Liuzzo, et al. 2016]

- Ontology-based data integration for humanities and archaeologists
- ERC advanced grant EPNet "Production and distribution of food during the Roman Empire: Economics and Political Dynamics".
- Linking three datasets:
  1. the EPNet relational repository
  2. the Epigraphic Database Heidelberg
  3. the Pleiades dataset
- Demo: `http://romanopendata.eu/`

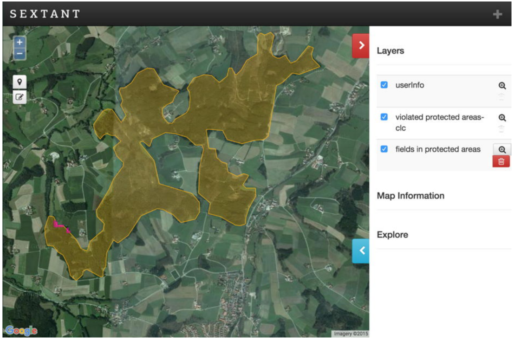


EMSec project [Brüggemann et al. 2016]

- German BMBF project EMSec, collaborated with Airbus.
- Provide real-time services for maritime security.
- Geo-spatial support by *Ontop*-spatial (developed as a fork of *Ontop*).
- SPARQL federation to access different kinds of data sources:
  - SPARQL endpoints of *Ontop* over *in situ* data
  - open SPARQL endpoints: Geonames, DBPedia

Melodies project

- EU FP7 Melodies project: working with Open Data, 16 partners.
- Geospatial extension *Ontop*-spatial used for accessing geospatial data.
- Use cases: urban development, land management, disaster management



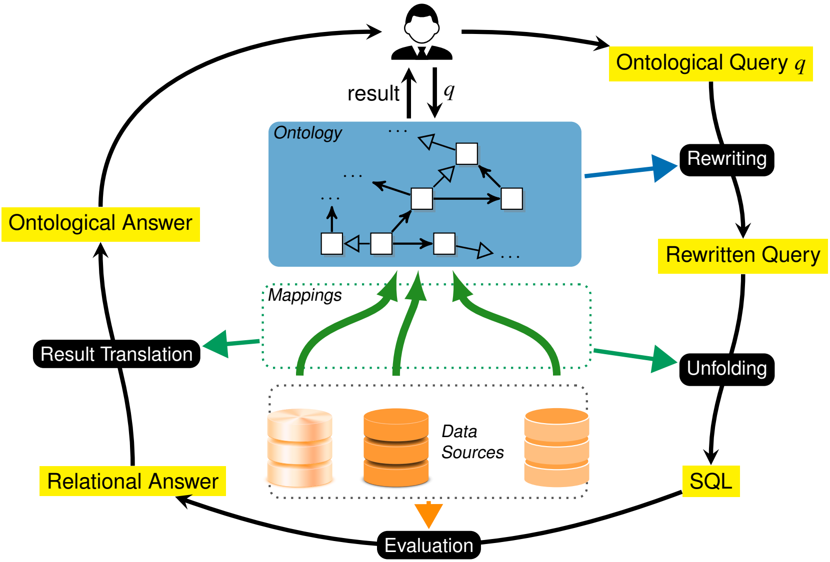
Visualization of violated protected areas in land management.

NOI Tourism Graph

- NOI is a South-Tyrolean company managing a Techpark in Bolzano and providing services to companies and research institutions.
- Ongoing project between Ontopic and NOI.
- Goal: publish tourism related data at the South-Tyrol OpenDataHub as a Knowledge Graph, and make it easily accessible (e.g., from Amazon Alexa).
- The publication of tourism related data serves as a pilot project, to be extended to other forms of open data.
- Demo: `https://sparql.opendatahub.testingmachine.eu/`
- Queries: `https://github.com/noi-techpark/odh-vkg/tree/development/sparql_queries`

Outline

Query answering via query reformulation – Conceptual framework



Query answering via query reformulation – Optimizations needed

The above conceptual framework is realized as follows.

Computing certain answers to a SPARQL query q over a VKG instance $\langle \mathcal{P}, \mathcal{D} \rangle$, with $\mathcal{P} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$:

1. Compute the perfect rewriting of q w.r.t. $\mathcal{O}$.
2. Unfold the perfect rewriting w.r.t. the mapping $\mathcal{M}$.
3. **Optimize** the unfolded query, using database constraints.
4. Evaluate the resulting SQL query over $\mathcal{D}$.

Steps 1 – 3 are collectively called **query reformulation**.

We analyze now more in detail these steps.

Outline

1. Motivation
2. Virtual Knowledge Graphs for Data Access
3. VKG Framework
4. VKG Systems and Usecases
5. Query Answering over VKGs
   - Query rewriting wrt an OWL 2 QL ontology
   - Query unfolding wrt a mapping
   - Mapping saturation
   - Optimization of query reformulation
6. Recent Developments and Future Plans

Rewriting step

The rewriting Step ❶ deals with the knowledge encoded by the axioms of the ontology:

- hierarchies of classes and of properties;
- objects that are existentially implied by such axioms: existential reasoning.

We illustrate the need for dealing with these two aspects with two examples.

Dealing with hierarchies

Suppose that every graduate student is a student, i.e.,

$$\text{GraduateStudent} \sqsubseteq \text{Student}$$

and john is a graduate student: $\text{GraduateStudent}(\texttt{john})$.

What is the answer to the following query, asking for all students?

$$\text{q}(x) \leftarrow \text{Student}(x)$$

In SPARQL: `SELECT ?x WHERE { ?x a Student . }`

The answer should be `john`, since being a graduate student, he is also a student.

Dealing with existential reasoning

Suppose that every student is supervised by some professor, i.e.,

$$\mathsf{Student} \sqsubseteq \exists \mathsf{isSupervisedBy}.\mathsf{Professor}$$

and john is a student: $\mathsf{Student}(\mathtt{john})$.

What is the answer to the following query, asking for all individuals supervised by some professor?

$$\mathsf{q}(x) \leftarrow \mathsf{isSupervisedBy}(x, y),\ \mathsf{Professor}(y)$$

In SPARQL: `SELECT ?x WHERE { ?x isSupervisedBy [ a Professor ] . }`

The answer should be `john`, even though we don't know who is John's supervisor (under existential reasoning).

The query rewriting algorithm

The **query rewriting** algorithm takes into account hierarchies and existential reasoning, by "compiling" the axioms of the ontology into the query.

Example

Consider the ontology axioms:

$$\begin{aligned} \text{Student} &\sqsubseteq \exists \text{isSupervisedBy}.\text{Professor} \\ \text{GraduateStudent} &\sqsubseteq \text{Student} \end{aligned}$$

Using these axioms, the rewriting algorithm rewrites the query

$$\text{q}(x) \leftarrow \text{isSupervisedBy}(x, y),\ \text{Professor}(y)$$

into a union of conjunctive queries (or a SPARQL union query):

$$\begin{aligned} \text{q}(x) &\leftarrow \text{isSupervisedBy}(x, y),\ \text{Professor}(y) \\ \text{q}(x) &\leftarrow \text{Student}(x) \\ \text{q}(x) &\leftarrow \text{GraduateStudent}(x) \end{aligned}$$

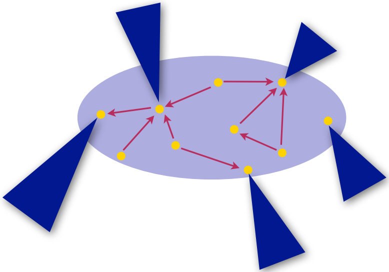
Therefore, over the data GraduateStudent(`john`), the rewritten query returns `john` as an answer.

Note: In *Ontop*, existential reasoning needs to be switched on explicitly, since it affects performance.

Query rewriting and canonical model

Canonical model

Every consistent *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model $\mathcal{I}_{\mathcal{K}}$**, which **gives the right answers to all CQs**, i.e., $\mathrm{cert}(q, \mathcal{K}) = \mathrm{ans}(q, \mathcal{I}_{\mathcal{K}})$



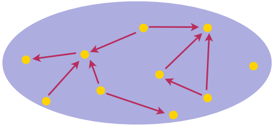
- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **tree-witness rewriting**.

Query rewriting and canonical model

Canonical model

Every consistent *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model** $\mathcal{I}_{\mathcal{K}}$, which **gives the right answers to all CQs**, i.e., $\text{cert}(q, \mathcal{K}) = \text{ans}(q, \mathcal{I}_{\mathcal{K}})$

Core
individuals from $\mathcal{A}$

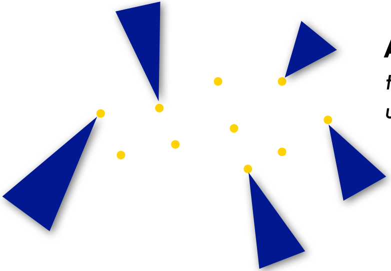


- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **tree-witness rewriting**.

Query rewriting and canonical model

Canonical model

Every consistent *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model $\mathcal{I}_{\mathcal{K}}$**, which **gives the right answers to all CQs**, i.e., $\mathrm{cert}(q, \mathcal{K}) = \mathrm{ans}(q, \mathcal{I}_{\mathcal{K}})$



Anonymous part

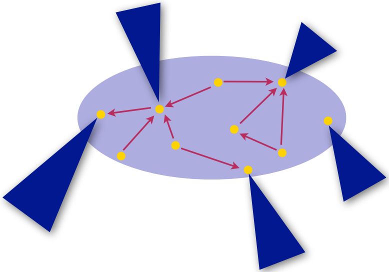
trees rooted at individuals, using unnamed objects

- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **tree-witness rewriting**.

Query rewriting and canonical model

Canonical model

Every consistent *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model** $\mathcal{I}_{\mathcal{K}}$, which **gives the right answers to all CQs**, i.e., $\text{cert}(q, \mathcal{K}) = \text{ans}(q, \mathcal{I}_{\mathcal{K}})$

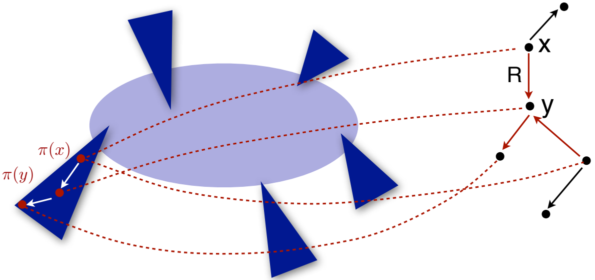


- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **tree-witness rewriting**.

Query rewriting and canonical model

Canonical model

Every consistent *DL-Lite* KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ has a **canonical model $\mathcal{I}_{\mathcal{K}}$**, which **gives the right answers to all CQs**, i.e., $\text{cert}(q, \mathcal{K}) = \text{ans}(q, \mathcal{I}_{\mathcal{K}})$



- The core part can be handled by **saturating the mapping**.
- The anonymous part can be handled by **tree-witness rewriting**.

The *PerfectRef* algorithm for query rewriting

We do not describe here the tree-witness rewriting algorithm, which is rather involved.

Instead, we describe *PerfectRef*, a simple query rewriting algorithm that maintains a set of queries and applies over them two types of transformations:

- rewriting steps that involve inclusion assertions of the ontology, and
- unification of query atoms.

These transformations are applied repeatedly until saturation, i.e., until the set of queries does not change anymore.

Given as input a (core) SPARQL query q, *PerfectRef* computes its **perfect rewriting**, which is still a SPARQL query (involving `UNION`).

Note: Disjointness assertions play a role in ontology satisfiability, but can be ignored during query rewriting. (This is called **separability**.)

Query rewriting step: Basic idea

Intuition: an inclusion assertion corresponds to a logic programming rule.

Basic rewriting step:

When an atom in the query unifies with the **head** of the rule, generate a new query by substituting the atom with the **body** of the rule.

We say that the inclusion assertion **applies to** the atom.

Example

The inclusion assertion $\text{Professor} \sqsubseteq \text{Teacher}$
corresponds to the logic programming rule $\text{Teacher}(z) \leftarrow \text{Professor}(z)$.

Consider the query $\text{q}(x) \leftarrow \text{Teacher}(x)$.

By applying the inclusion assertion to the atom $\text{Teacher}(x)$, we generate:

$$\text{q}(x) \leftarrow \text{Professor}(x).$$

This query is added to the input query, and contributes to the perfect rewriting.

Query rewriting (cont'd)

Example

Consider the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{Course}(y)$

and the inclusion assertion $\exists \mathsf{teaches}^- \sqsubseteq \mathsf{Course}$
as a logic programming rule: $\mathsf{Course}(z_2) \leftarrow \mathsf{teaches}(z_1, z_2)$.

The inclusion applies to $\mathsf{Course}(y)$, and we add to the rewriting the query

$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{teaches}(z_1, y).$$

Example

Consider now the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y)$

and the inclusion assertion $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$
as a logic programming rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$.

The inclusion applies to $\mathsf{teaches}(x, y)$, and we add to the rewriting the query

$$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x).$$

Query rewriting – Constants

Example

Conversely, for the query $\text{q}(x) \leftarrow \text{teaches}(x, \texttt{databases})$

and the same inclusion assertion as before $\text{Professor} \sqsubseteq \exists\text{teaches}$
as a logic programming rule: $\text{teaches}(z, f(z)) \leftarrow \text{Professor}(z)$

$\text{teaches}(x, \texttt{databases})$ does not unify with $\text{teaches}(z, f(z))$, since the **skolem term** $f(z)$ in the head of the rule **does not unify** with the constant $\texttt{databases}$.
Remember: We adopt the **unique name assumption**.

We say that the inclusion does **not** apply to the atom $\text{teaches}(x, \texttt{databases})$.

Example

The same holds for the following query, where y is **distinguished**, since unifying $f(z)$ with y would correspond to returning a skolem term as answer to the query:

$$\text{q}(x, y) \leftarrow \text{teaches}(x, y).$$

Query rewriting – Join variables

An analogous behavior to the one with constants and with distinguished variables holds when the atom contains **join variables** that would have to be unified with skolem terms.

Example

Consider the query $\quad \text{q}(x) \leftarrow \text{teaches}(x, y), \text{Course}(y)$

and the inclusion assertion $\quad \text{Professor} \sqsubseteq \exists\text{teaches}$

as a logic programming rule: $\quad \text{teaches}(z, f(z)) \leftarrow \text{Professor}(z)$.

The inclusion assertion above does **not** apply to the atom $\text{teaches}(x, y)$.

Query rewriting – Reduce step

Example

Consider now the query $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y), \mathsf{teaches}(z, y)$

and the inclusion assertion $\mathsf{Professor} \sqsubseteq \exists\mathsf{teaches}$
as a logic rule: $\mathsf{teaches}(z, f(z)) \leftarrow \mathsf{Professor}(z)$.

This inclusion assertion does not apply to $\mathsf{teaches}(x, y)$ or $\mathsf{teaches}(z, y)$, since y is in join, and we would again introduce the skolem term in the rewritten query.

Example

However, we can transform the above query by unifying the atoms $\mathsf{teaches}(x, y)$ and $\mathsf{teaches}(z, y)$. This rewriting step is called **reduce**, and produces the query

$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, y).$$

Now, we can apply the inclusion above, and add to the rewriting the query

$$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x).$$

Query rewriting – Summary

To compute the perfect rewriting of a query q, start from q, iteratively get a CQ q' to be processed, and do one of the following:

- Apply to some atom of q' an inclusion assertion in the ontology $\mathcal{O}$ as follows:

$$
\begin{array}{rccc}
A_1 \sqsubseteq A_2 & \ldots, A_2(x), \ldots & \rightsquigarrow & \ldots, A_1(x), \ldots \\
\exists P \sqsubseteq A & \ldots, A(x), \ldots & \rightsquigarrow & \ldots, P(x, \_), \ldots \\
\exists P^- \sqsubseteq A & \ldots, A(x), \ldots & \rightsquigarrow & \ldots, P(\_, x), \ldots \\
A \sqsubseteq \exists P & \ldots, P(x, \_), \ldots & \rightsquigarrow & \ldots, A(x), \ldots \\
A \sqsubseteq \exists P^- & \ldots, P(\_, x), \ldots & \rightsquigarrow & \ldots, A(x), \ldots \\
\exists P_1 \sqsubseteq \exists P_2 & \ldots, P_2(x, \_), \ldots & \rightsquigarrow & \ldots, P_1(x, \_), \ldots \\
P_1 \sqsubseteq P_2 & \ldots, P_2(x, y), \ldots & \rightsquigarrow & \ldots, P_1(x, y), \ldots \\
P_1 \sqsubseteq P_2^- & \ldots, P_2(x, y), \ldots & \rightsquigarrow & \ldots, P_1(y, x), \ldots
\end{array}
$$

  ('_' denotes a variable that appears only once)

- Choose two atoms of q' that unify, and apply the unifier to q'.

After each rewriting/unification step, the obtained query is added to the queries still to be processed.

Note: Unifying atoms can make rules applicable that were not so before, and is required for completeness of the method [C., De Giacomo, Lembo, Lenzerini, and Rosati 2007].

The UCQ resulting from this process is the **perfect rewriting** q_r of q w.r.t. the ontology $\mathcal{O}$.

Query rewriting algorithm

Algorithm *PerfectRef*$(Q, \mathcal{T}_P)$
Input: union of conjunctive queries Q, set $\mathcal{T}_P$ of *DL-Lite* inclusion assertions
Output: union of conjunctive queries PR
$PR := Q$;
repeat
  $PR' := PR$;
  for each $q \in PR'$ **do**
    for each g in q **do**
      for each inclusion assertion I in $\mathcal{T}_P$ **do**
        if I is applicable to g **then** $PR := PR \cup \{$ *ApplyPI*(q, g, I) $\}$;
    for each g_1, g_2 in q **do**
      if g_1 and g_2 unify **then** $PR := PR \cup \{\tau($*Reduce*$(q, g_1, g_2))\}$;
until $PR' = PR$;
return PR

Observations:

- Termination follows from having only finitely many different rewritings.
- Disjointness assertions and functionalities do not play any role in the rewriting of the query.

Query answering in *DL-Lite* – Example

Ontology:

$$\mathsf{Professor} \sqsubseteq \mathsf{Teacher}$$
$$\mathsf{Teacher} \sqsubseteq \exists \mathsf{teaches}$$
$$\exists \mathsf{teaches}^- \sqsubseteq \mathsf{Course}$$

Corresponding rules:

$$\mathsf{Teacher}(x) \leftarrow \mathsf{Professor}(x)$$
$$\exists y(\mathsf{teaches}(x,y)) \leftarrow \mathsf{Teacher}(x)$$
$$\mathsf{Course}(x) \leftarrow \mathsf{teaches}(y,x)$$

Query: $\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y), \mathsf{Course}(y)$

Perfect rewriting:

$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y), \mathsf{Course}(y)$$
$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,y), \mathsf{teaches}(\_,y)$$
$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x,\_)$$
$$\mathsf{q}(x) \leftarrow \mathsf{Teacher}(x)$$
$$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$$

ABox: $\mathsf{teaches}(\mathtt{jim}, \mathtt{databases})$ $\mathsf{Professor}(\mathtt{jim})$
$\mathsf{teaches}(\mathtt{julia}, \mathtt{security})$ $\mathsf{Professor}(\mathtt{nicole})$

Evaluating the perfect rewriting over the ABox (seen as a DB) produces as answer $\{\mathtt{jim}, \mathtt{julia}, \mathtt{nicole}\}$.

Query answering in *DL-Lite* – An interesting example

TBox: $\mathsf{Person} \sqsubseteq \exists\mathsf{hasFather}$
$\exists\mathsf{hasFather}^- \sqsubseteq \mathsf{Person}$

ABox: $\mathsf{Person}(\mathtt{john})$

Query: $\mathsf{q}(x) \leftarrow \mathsf{Person}(x), \mathsf{hasFather}(x, y_1), \mathsf{hasFather}(y_1, y_2), \mathsf{hasFather}(y_2, y_3)$

$\mathsf{q}(x) \leftarrow \mathsf{Person}(x), \mathsf{hasFather}(x, y_1), \mathsf{hasFather}(y_1, y_2), \mathsf{hasFather}(y_2, \_)$

$\Downarrow$ **Apply** $\mathsf{Person} \sqsubseteq \exists\mathsf{hasFather}$ to the atom $\mathsf{hasFather}(y_2, \_)$

$\mathsf{q}(x) \leftarrow \mathsf{Person}(x), \mathsf{hasFather}(x, y_1), \mathsf{hasFather}(y_1, y_2), \mathsf{Person}(y_2)$

$\Downarrow$ **Apply** $\exists\mathsf{hasFather}^- \sqsubseteq \mathsf{Person}$ to the atom $\mathsf{Person}(y_2)$

$\mathsf{q}(x) \leftarrow \mathsf{Person}(x), \mathsf{hasFather}(x, y_1), \mathsf{hasFather}(y_1, y_2), \mathsf{hasFather}(\_, y_2)$

$\Downarrow$ **Unify** atoms $\mathsf{hasFather}(y_1, y_2)$ and $\mathsf{hasFather}(\_, y_2)$

$\mathsf{q}(x) \leftarrow \mathsf{Person}(x), \mathsf{hasFather}(x, y_1), \mathsf{hasFather}(y_1, y_2)$

$\Downarrow$

$\ldots$

$\mathsf{q}(x) \leftarrow \mathsf{Person}(x), \mathsf{hasFather}(x, \_)$

$\Downarrow$ **Apply** $\mathsf{Person} \sqsubseteq \exists\mathsf{hasFather}$ to the atom $\mathsf{hasFather}(x, \_)$

$\mathsf{q}(x) \leftarrow \mathsf{Person}(x)$

Outline

1. Motivation
2. Virtual Knowledge Graphs for Data Access
3. VKG Framework
4. VKG Systems and Usecases
5. Query Answering over VKGs
   - Query rewriting wrt an OWL 2 QL ontology
   - Query unfolding wrt a mapping
   - Mapping saturation
   - Optimization of query reformulation
6. Recent Developments and Future Plans

Query unfolding

We consider now Step 2 of reformulation, i.e., the unfolding w.r.t. the mapping $\mathcal{M}$.

In principle, we have two approaches to exploit the mapping:

- bottom-up approach: simpler, but typically less efficient
- top-down approach: more sophisticated, but also more efficient

Both approaches require to first **split** the set of atoms in the target queries of the mapping assertions into the constituent atoms.

Note: In the following, to make notation more compact, we represent terms of the form

$$\textbf{iri}(\text{"xxx"}, v_1, \ldots, v_n) \qquad \text{as} \qquad \textbf{xxx}(v_1, \ldots, v_n).$$

Splitting of mappings

A mapping assertion $\Phi \rightsquigarrow \Psi$, where the target query Ψ is constituted by the atoms $X_1, \ldots, X_k$, can be split into k mapping assertions:

$$\Phi \rightsquigarrow X_1 \qquad \cdots \qquad \Phi \rightsquigarrow X_k$$

This is possible, since Ψ does not contain non-distinguished variables.

Example

m_1: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ Play(**pl**($pcode$)), Actor(**act**($acode$)), name(**act**($acode$), $aname$), actsIn(**act**($acode$), **pl**($pcode$))

is split into

m_1^1: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ Play(**pl**($pcode$))
m_1^2: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ Actor(**act**($acode$))
m_1^3: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ name(**act**($acode$), $aname$)
m_1^4: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ actsIn(**act**($acode$), **pl**($pcode$))

Bottom-up approach to deal with mappings: Materialization

Consists in a straightforward application of the mappings to the data:

1. Propagate the data from $\mathcal{D}$ through $\mathcal{M}$, **materializing** the RDF graph $\mathcal{V} = \mathcal{M}(\mathcal{D})$ (the constants in such an RDF graph are values and object terms obtained from the database values).
2. Apply to $\mathcal{V}$ and to the ontology $\mathcal{O}$, the satisfiability and query answering algorithms developed for *DL-Lite*.

This approach has several drawbacks:

- The technique is no more AC^0 in the size of the data, since the RDF graph $\mathcal{V}$ to materialize is in general polynomial in the size of the data.
- $\mathcal{V}$ may be very large, and thus it may be infeasible to actually materialize it.
- Freshness of $\mathcal{V}$ with respect to the underlying data source(s) may be an issue, and one would need to propagate source updates (cf. Data Warehousing).

Top-down approach to deal with mappings: Unfolding

The top-down approach is realized by computing from the (rewritten) query q_r a new query q_{unf}, by **unfolding** q_r using (the split version of) the mappings $\mathcal{M}$.

Consider the mapping assertions $\Phi_i \rightsquigarrow \Psi_i$.

- Essentially, each atom in q_r that unifies with an atom in some Ψ_i is substituted with the corresponding query Φ_i over the database.
- The unfolded query q_{unf} is such that for each database $\mathcal{D}$ we have that:

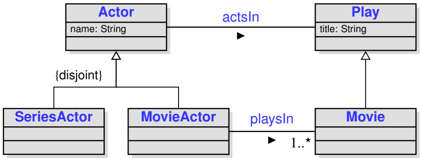
$$q_{unf}(\mathcal{D}) \;=\; Eval_{CWA}(q_r, \mathcal{M}(\mathcal{D})).$$

Unfolding

To unfold a query q_r with respect to a set $\mathcal{M}$ of mapping assertions:

1. For each non-split mapping assertion $\Phi_i(\vec{x}) \rightsquigarrow \Psi_i(\vec{t}, \vec{y})$:
   1. Introduce a **view symbol** Aux_i of arity equal to that of Φ_i.
   2. Add a **view definition** $\mathsf{Aux}_i(\vec{x}) \leftarrow \Phi_i(\vec{x})$.
2. For each split version $\Phi_i(\vec{x}) \rightsquigarrow X_i^j(\vec{t}, \vec{y})$ of a mapping assertion, introduce a **clause** $X_i^j(\vec{t}, \vec{y}) \leftarrow \mathsf{Aux}_i(\vec{x})$.
3. Obtain from q_r in all possible ways queries q_{aux} defined over the view symbols Aux_i as follows:
   1. Find a most general unifier ϑ that unifies each atom $X(\vec{z})$ in the body of q_r with the head of a clause $X(\vec{t}, \vec{y}) \leftarrow \mathsf{Aux}_i(\vec{x})$.
   2. Substitute each atom $X(\vec{z})$ with $\vartheta(\mathsf{Aux}_i(\vec{x}))$, i.e., with the body the unified clause to which the unifier ϑ is applied.
4. The unfolded query q_{unf} is the **union** of all queries q_{aux}, together with the view definitions for the predicates Aux_i appearing in q_{aux}.

Unfolding – Example



m_1: `SELECT pcode, acode, aname FROM ACTOR` $\rightsquigarrow$ Play(**pl**($pcode$)), Actor(**act**($acode$)), name(**act**($acode$), $aname$), actsIn(**act**($acode$), **pl**($pcode$))

m_2: `SELECT mcode, acode, mtitle FROM MOVIE M, ACTOR A WHERE M.mcode = A.pcode AND M.type = "m"` $\rightsquigarrow$ Movie(**pl**($mcode$)), playsIn(**act**($acode$), **pl**($mcode$)), title(**pl**($mcode$), $mtitle$)

We define a view Aux$_i$ for the source query of each mapping m_i.

For each (split) mapping assertion, we introduce a clause:

$$\begin{array}{rcl}
\text{Play}(\textbf{pl}(pcode)) & \leftarrow & \text{Aux}_1(pcode, _, _) \\
\text{Actor}(\textbf{act}(acode)) & \leftarrow & \text{Aux}_1(_, acode, _) \\
\text{name}(\textbf{act}(acode), aname) & \leftarrow & \text{Aux}_1(_, acode, aname) \\
\text{actsIn}(\textbf{act}(acode), \textbf{pl}(pcode)) & \leftarrow & \text{Aux}_1(pcode, acode, _) \\
\text{Movie}(\textbf{pl}(mcode)) & \leftarrow & \text{Aux}_2(mcode, _, _) \\
\text{playsIn}(\textbf{act}(acode), \textbf{pl}(mcode)) & \leftarrow & \text{Aux}_2(mcode, acode, _) \\
\text{title}(\textbf{pl}(mcode), mtitle) & \leftarrow & \text{Aux}_2(mcode, _, mtitle)
\end{array}$$

Unfolding – Example (cont'd)

Query over the ontology: Actors with their name who act in a movie whose title is "The Matrix":

$\boldsymbol{q}(a,n) \leftarrow \mathsf{Actor}(a), \mathsf{name}(a,n), \mathsf{actsIn}(a,p), \mathsf{Movie}(p), \mathsf{title}(p, \texttt{"The Matrix"})$

$$
\begin{array}{rcl}
\mathsf{Actor}(\mathbf{act}(acode)) & \leftarrow & \mathsf{Aux}_1(\_, acode, \_) \\
\mathsf{name}(\mathbf{act}(acode), aname) & \leftarrow & \mathsf{Aux}_1(\_, acode, aname) \\
\mathsf{actsIn}(\mathbf{act}(acode), \mathbf{pl}(pcode)) & \leftarrow & \mathsf{Aux}_1(pcode, acode, \_) \\
\mathsf{Movie}(\mathbf{pl}(mcode)) & \leftarrow & \mathsf{Aux}_2(mcode, \_, \_) \\
\mathsf{title}(\mathbf{pl}(mcode), mtitle) & \leftarrow & \mathsf{Aux}_2(mcode, \_, mtitle)
\end{array}
$$

A unifier ϑ between the atoms in $\boldsymbol{q}$ and the clause heads is:

$\vartheta(a) = \mathbf{act}(acode)$ $\quad \vartheta(n) = aname$

$\vartheta(p) = \mathbf{pl}(pcode)$ $\quad \vartheta(mcode) = pcode$ $\quad \vartheta(mtitle) = \texttt{"The Matrix"}$

After applying ϑ to $\boldsymbol{q}$, we obtain:

$$
\begin{array}{rl}
\boldsymbol{q}(\mathbf{act}(acode), aname) \leftarrow & \mathsf{Actor}(\mathbf{act}(acode)), \;\; \mathsf{name}(\mathbf{act}(acode), aname), \\
& \mathsf{actsIn}(\mathbf{act}(acode), \mathbf{pl}(pcode)), \;\; \mathsf{Movie}(\mathbf{pl}(pcode)), \\
& \mathsf{title}(\mathbf{pl}(pcode), \texttt{"The Matrix"})
\end{array}
$$

Substituting the atoms with the bodies of the clauses (after having applied the unifier), we obtain:

$$
\begin{array}{rl}
\boldsymbol{q}(\mathbf{act}(acode), aname) \leftarrow & \mathsf{Aux}_1(\_, acode, \_), \;\; \mathsf{Aux}_1(\_, acode, aname), \\
& \mathsf{Aux}_1(pcode, acode, \_), \;\; \mathsf{Aux}_2(pcode, \_, \_), \\
& \mathsf{Aux}_2(pcode, \_, \texttt{"The Matrix"})
\end{array}
$$

Exponential blowup in the unfolding

When there are multiple mapping assertions for each atom, the unfolded query may be exponential in the original one.

Consider a query: $q(y) \leftarrow A_1(y), A_2(y), \ldots, A_n(y)$

and the mappings: $m_i^1 \colon \Phi_i^1(x) \rightsquigarrow A_i(\mathbf{iri}(x))$ (for $i \in \{1,\ldots,n\}$)
$m_i^2 \colon \Phi_i^2(x) \rightsquigarrow A_i(\mathbf{iri}(x))$

We add the view definitions: $\mathsf{Aux}_i^j(x) \leftarrow \Phi_i^j(x)$
and introduce the clauses: $A_i(\mathbf{iri}(x)) \leftarrow \mathsf{Aux}_i^j(x)$ (for $i \in \{1,\ldots,n\}, j \in \{1,2\}$).

There is a single unifier, namely $\vartheta(y) = \mathbf{iri}(x)$, but each atom $A_i(y)$ in the query unifies with the head of two clauses.

Hence, we obtain one unfolded query

$$q(\mathbf{iri}(x)) \leftarrow \mathsf{Aux}_1^{j_1}(x), \mathsf{Aux}_2^{j_2}(x), \ldots, \mathsf{Aux}_n^{j_n}(x)$$

for each possible combination of $j_i \in \{1,2\}$, for $i \in \{1,\ldots,n\}$.
Hence, we obtain 2^n **unfolded queries**.

Implementation of top-down approach to query answering

To implement the top-down approach, we need to generate an SQL query.

We can follow different strategies:

1. Substitute each view predicate in the unfolded queries with the corresponding SQL query over the source:
   - \+ joins are performed on the DB attributes, hence can be done efficiently, e.g., by exploiting indexes;
   - \+ does not generate doubly nested queries;
   - – the number of unfolded queries may be exponential.

2. Construct for each atom in the original query a new view. This view takes the union of all SQL queries corresponding to the view predicates, and constructs also the IRIs based on the IRI templates:
   - \+ avoids exponential blow-up of the resulting query, since the union (of the queries coming from multiple mappings) is done before the joins;
   - – joins are performed on IRIs, i.e., on terms built using string concatenation, hence are highly inefficient;
   - – generates doubly nested queries, which per se the database has difficulty in optimizing.

Which method is better, depends on various parameters, and there is no definitive answer.
In general, one needs a mixed approach that applies different strategies to different parts of the query.

Outline

Contributions of rewriting and unfolding

- We are interested in computing certain answers to SPARQL queries over a VKG instance $\langle\mathcal{P},\mathcal{D}\rangle$, with $\mathcal{P}=\langle\mathcal{O},\mathcal{M},\mathcal{S}\rangle$.
- In practice, by computing the rewriting q_r of q w.r.t. $\mathcal{O}$ and its unfolding w.r.t. $\mathcal{M}$, the resulting query q_{unf} might become very large, and costly to execute over $\mathcal{D}$.

Let us consider the contributions of rewriting and unfolding to the query answers:

- In principle, evaluating the unfolding q_{unf} (of q_r w.r.t. $\mathcal{M}$) over $\mathcal{D}$, gives the same result as evaluating q_r over the RDF graph $\mathcal{G}=\mathcal{M}(\mathcal{D})$ extracted through the mapping $\mathcal{M}$ from the data $\mathcal{D}$.
- Instead, the impact of the rewriting on the query answers consists of two components:
  1. the rewriting w.r.t. class and property hierarchies, i.e., $C \sqsubseteq A$, $P_1 \sqsubseteq P_2$;
  2. the rewriting taking into account existential reasoning, i.e., $C_1 \sqsubseteq \exists R$, $C_1 \sqsubseteq \exists R.C_2$.

Note: Component 1 corresponds to computing the saturation $\mathcal{G}_{\text{sat}}$ of $\mathcal{G}$ w.r.t. class and property hierarchies, while component 2 can be handled only through rewriting.

Tree-witness rewriting and saturated mapping

We want to avoid materializing $\mathcal{G}$ and $\mathcal{G}_{\mathsf{sat}}$, but also computing the query rewriting w.r.t. class and property hierarchies.

Therefore we proceed as follows:

1. We rewrite q only w.r.t. the inclusion assertions that cause existential reasoning (i.e., $C_1 \sqsubseteq \exists R$ and $C_1 \sqsubseteq \exists R.C_2$).
   $\leadsto$ **tree-witness rewriting** q_{tw}
2. We use instead class and property hierarchies (i.e., $C \sqsubseteq A$, $P_1 \sqsubseteq P_2$) to enrich the mapping $\mathcal{M}$.
   $\leadsto$ **saturated mapping** $\mathcal{M}_{\mathsf{sat}}$
3. We unfold the tree-witness rewriting q_{tw} w.r.t. the saturated mapping $\mathcal{M}_{\mathsf{sat}}$.

It is possible to show that the resulting query is equivalent to the perfect rewriting q_{r} (as obtained, e.g., through ordinary rewriting w.r.t. $\mathcal{O}$ and unfolding w.r.t. $\mathcal{M}$).

Saturated mapping

Intuitively, the **saturated mapping** $\mathcal{M}_{\text{sat}}$ is obtained as the composition of $\mathcal{M}$ and the ontology $\mathcal{O}$.

For each mapping assertion in $\mathcal{M}$	and each TBox assertion in $\mathcal{O}$	we add a mapping assertion to $\mathcal{M}_{\text{sat}}$
$\Phi(x) \leadsto A_1(\textbf{iri}(x))$	$A_1 \sqsubseteq A_2$	$\Phi(x) \leadsto A_2(\textbf{iri}(x))$
$\Phi(x,y) \leadsto P(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$\exists P \sqsubseteq A_1$	$\Phi(x,y) \leadsto A_1(\textbf{iri}_1(x))$
$\Phi(x,y) \leadsto P(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$\exists P^- \sqsubseteq A_2$	$\Phi(x,y) \leadsto A_2(\textbf{iri}_2(y))$
$\Phi(x,y) \leadsto P_1(\textbf{iri}_1(x), \textbf{iri}_2(y))$	$P_1 \sqsubseteq P_2$	$\Phi(x,y) \leadsto P_2(\textbf{iri}_1(x), \textbf{iri}_2(y))$

Due to saturation, $\mathcal{M}_{\text{sat}}$ will contain at most $|\mathcal{O}| \cdot |\mathcal{M}|$ many mappings.

Note: The saturated mapping has also been called **T-mapping** in the literature.

Saturated mapping – Exercise

Ontology $\mathcal{O}$

$$
\begin{aligned}
\text{Student} &\sqsubseteq \text{Person}\\
\text{PostDoc} &\sqsubseteq \text{Faculty}\\
\text{Professor} &\sqsubseteq \text{Faculty}\\
\exists\text{teaches} &\sqsubseteq \text{Faculty}\\
\text{Faculty} &\sqsubseteq \text{Person}
\end{aligned}
$$

User-defined mapping assertions $\mathcal{M}$

$\texttt{student}(scode, fn, ln)$	$\leadsto$	Student(**iri1**($scode$))	(1)
$\texttt{academic}(acode, fn, ln, pos),\ pos = 9$	$\leadsto$	PostDoc(**iri2**($acode$))	(2)
$\texttt{academic}(acode, fn, ln, pos),\ pos = 2$	$\leadsto$	Professor(**iri2**($acode$))	(3)
$\texttt{teaching}(course, acode)$	$\leadsto$	teaches(**iri2**($acode$), **iri3**($course$))	(4)
$\texttt{academic}(acode, fn, ln, pos)$	$\leadsto$	Faculty(**iri2**($acode$))	(5)

By **saturating the mapping**, we obtain $\mathcal{M}_{\text{sat}}$, containing additional mapping assertions for the classes Faculty and Person.

$\texttt{student}(scode, fn, ln)$	$\leadsto$	Person(**iri1**($scode$))	(6)
$\texttt{academic}(acode, fn, ln, pos),\ pos = 9$	$\leadsto$	Faculty(**iri2**($acode$))	(7)
$\texttt{academic}(acode, fn, ln, pos),\ pos = 9$	$\leadsto$	Person(**iri2**($acode$))	(8)
$\texttt{academic}(acode, fn, ln, pos),\ pos = 2$	$\leadsto$	Faculty(**iri2**($acode$))	(9)
$\texttt{academic}(acode, fn, ln, pos),\ pos = 2$	$\leadsto$	Person(**iri2**($acode$))	(10)
$\texttt{academic}(acode, fn, ln, pos)$	$\leadsto$	Person(**iri2**($acode$))	(11)
$\texttt{teaching}(course, acode)$	$\leadsto$	Faculty(**iri2**($acode$))	(12)
$\texttt{teaching}(course, acode)$	$\leadsto$	Person(**iri2**($acode$))	(13)

unibz

Properties of saturated mappings

H-complete RDF graph

An RDF graph $\mathcal{G}$ is **H-complete** w.r.t. an ontology $\mathcal{O}$, if, for every RDF triple (s, p, o), we have:

$$\langle \mathcal{O}, \mathcal{G} \rangle \models (s, p, o) \qquad \text{iff} \qquad (s, p, o) \in \mathcal{G}$$

The **saturation** $\mathcal{G}_{\text{sat}}$ of $\mathcal{G}$ w.r.t. $\mathcal{O}$ is the smallest RDF graph that contains $\mathcal{G}$ and is H-complete w.r.t. $\mathcal{O}$.

Intuitively, $\mathcal{G}_{\text{sat}}$ is obtained from $\mathcal{G}$ by applying the class and property inclusions of $\mathcal{O}$, but without introducing new nodes.

Relationship between the saturated mapping $\mathcal{M}_{\text{sat}}$ and the saturation of $\mathcal{M}(\mathcal{D})$

- We have that $\mathcal{M}_{\text{sat}}(\mathcal{D}) = (\mathcal{M}(\mathcal{D}))_{\text{sat}}$ (hence, it is an H-complete RDF graph).
- $\mathcal{M}_{\text{sat}}$ does not depend on the SPARQL query q, hence it can be pre-computed.
- It can be optimized (by exploiting query containment).

Mapping optimization – Exercise

Saturated mapping assertions $\mathcal{M}_{\text{sat}}$

$$
\begin{array}{lcll}
\dots & & & \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow & \textsf{Faculty}(\textbf{iri2}(acode)) & (5)\\
\texttt{student}(scode, fn, ln) & \rightsquigarrow & \textsf{Person}(\textbf{iri1}(scode)) & (6)\\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow & \textsf{Faculty}(\textbf{iri2}(acode)) & (7)\\
\texttt{academic}(acode, fn, ln, pos),\ pos = 9 & \rightsquigarrow & \textsf{Person}(\textbf{iri2}(acode)) & (8)\\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow & \textsf{Faculty}(\textbf{iri2}(acode)) & (9)\\
\texttt{academic}(acode, fn, ln, pos),\ pos = 2 & \rightsquigarrow & \textsf{Person}(\textbf{iri2}(acode)) & (10)\\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow & \textsf{Person}(\textbf{iri2}(acode)) & (11)\\
\texttt{teaching}(course, acode) & \rightsquigarrow & \textsf{Faculty}(\textbf{iri2}(acode)) & (12)\\
\texttt{teaching}(course, acode) & \rightsquigarrow & \textsf{Person}(\textbf{iri2}(acode)) & (13)
\end{array}
$$

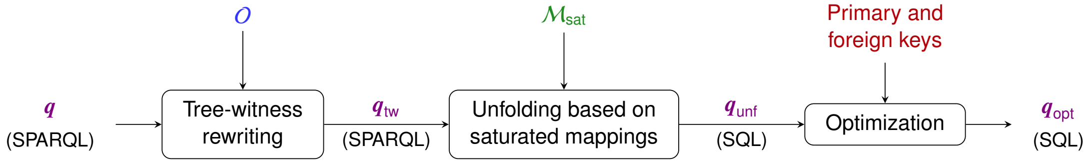
Consider also a foreign key over the database relations

FK: $\exists y_1.\texttt{teaching}(y_1, x) \rightarrow \exists y_2 y_3 y_4.\texttt{academic}(x, y_2, y_3, y_4)$

We can **optimize the mapping** using query containment and the FK. This removes mapping assertions 7, 8, 9, 10, 12, and 13.

$$
\begin{array}{lcll}
\dots & & & \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow & \textsf{Faculty}(\textbf{iri2}(acode)) & (5)\\
\texttt{student}(scode, fn, ln) & \rightsquigarrow & \textsf{Person}(\textbf{iri1}(scode)) & (6)\\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow & \textsf{Person}(\textbf{iri2}(acode)) & (11)
\end{array}
$$

Query reformulation as implemented by the Ontop system



	Step	Input	Output
1.	Tree-witness rewriting	q and $\mathcal{O}$	q_{tw}
2.	Unfolding	q_{tw} and $\mathcal{M}_{\text{sat}}$	q_{unf}
3.	Optimization	q_{unf}, primary and foreign keys	q_{opt}

Let us now consider the optimization step.

Outline

SQL query optimization

Objective : produce SQL queries that are . . .

- similar to manually written ones
- adapted to existing query planners

Structural optimization

- From join-of-unions to union-of-joins
- IRI decomposition to improve performance of joins

Semantic optimization

- Redundant join elimination
- Redundant union elimination
- Using functional constraints

Integrity constraints

- Primary and foreign keys, uniqueness constraints
- Sometimes implicit
- Vital for query reformulation!

Reformulation example – 1. Unfolding

Saturated mapping

$$
\begin{array}{ll}
\texttt{academic}(acode, fn, ln, pos),\ pos \in [1..8] & \\
 & \rightsquigarrow \text{Teacher}(\textbf{iri2}(acode)) \\
\texttt{teaching}(course, acode) & \rightsquigarrow \text{Teacher}(\textbf{iri2}(acode)) \\
\texttt{student}(scode, fn, ln) & \rightsquigarrow \text{firstName}(\textbf{iri1}(scode), fn) \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{firstName}(\textbf{iri2}(acode), fn) \\
\texttt{student}(scode, fn, ln) & \rightsquigarrow \text{lastName}(\textbf{iri1}(scode), ln) \\
\texttt{academic}(acode, fn, ln, pos) & \rightsquigarrow \text{lastName}(\textbf{iri2}(acode), ln)
\end{array}
$$

Query (we assume that the ontology is empty, hence $q_r = q$)

$$\boldsymbol{q}(x, y, z) \leftarrow \text{Teacher}(x),\ \text{firstName}(x, y),\ \text{lastName}(x, z)$$

We apply **query unfolding**, and then **normalization** to make the join conditions explicit.

$$
\begin{array}{rcl}
\boldsymbol{q}_{\text{norm}}(x, y, z) & \leftarrow & \boldsymbol{q}1_{\text{unf}}(x),\ \boldsymbol{q}2_{\text{unf}}(x_1, y), \\
 & & \boldsymbol{q}3_{\text{unf}}(x_2, z),\ x = x_1,\ x = x_2 \\
\boldsymbol{q}1_{\text{unf}}(\textbf{iri2}(acode)) & \leftarrow & \texttt{academic}(acode, fn, ln, pos), \\
 & & pos \in [1..8] \\
\boldsymbol{q}1_{\text{unf}}(\textbf{iri2}(acode)) & \leftarrow & \texttt{teaching}(course, acode) \\
\boldsymbol{q}2_{\text{unf}}(\textbf{iri1}(scode), fn) & \leftarrow & \texttt{student}(scode, fn, ln) \\
\boldsymbol{q}2_{\text{unf}}(\textbf{iri2}(acode), fn) & \leftarrow & \texttt{academic}(acode, fn, ln, pos) \\
\boldsymbol{q}3_{\text{unf}}(\textbf{iri1}(scode), ln) & \leftarrow & \texttt{student}(scode, fn, ln) \\
\boldsymbol{q}3_{\text{unf}}(\textbf{iri2}(acode), ln) & \leftarrow & \texttt{academic}(acode, fn, ln, pos)
\end{array}
$$

Reformulation example – 2. Structural optimization

Unfolded normalized query

$$
\begin{aligned}
q_{\mathsf{norm}}(x,y,z) &\leftarrow q1_{\mathsf{unf}}(x),\ q2_{\mathsf{unf}}(x_1,y),\\
&\qquad q3_{\mathsf{unf}}(x_2,z),\\
&\qquad x = x_1,\ x = x_2\\
q1_{\mathsf{unf}}(\mathbf{iri2}(a)) &\leftarrow \mathtt{academic}(a,f,l,p),\\
&\qquad p \in [1..8]\\
q1_{\mathsf{unf}}(\mathbf{iri2}(a)) &\leftarrow \mathtt{teaching}(c,a)\\
q2_{\mathsf{unf}}(\mathbf{iri1}(s),f) &\leftarrow \mathtt{student}(s,f,l)\\
q2_{\mathsf{unf}}(\mathbf{iri2}(a),f) &\leftarrow \mathtt{academic}(a,f,l,p)\\
q3_{\mathsf{unf}}(\mathbf{iri1}(s),l) &\leftarrow \mathtt{student}(s,f,l)\\
q3_{\mathsf{unf}}(\mathbf{iri2}(a),l) &\leftarrow \mathtt{academic}(a,f,l,p)
\end{aligned}
$$

- While flattening, we can avoid to generate those queries that contain in their body an equality between two terms with incompatible IRI templates.
- This might avoid a potential exponential blowup.

Flattening (URI template lifting) – Part 1/2

$$
\begin{aligned}
q_{\mathsf{lift}}(\mathbf{iri2}(a),y,z) \leftarrow\ & \mathtt{academic}(a,f_1,l_1,p_1),\\
& \mathtt{student}(s,f_2,l_2),\\
& \mathtt{student}(s_1,f_3,l_3),\\
& \mathbf{iri2}(a) = \mathbf{iri1}(s),\\
& \mathbf{iri2}(a) = \mathbf{iri1}(s_1),\\
& p_1 \in [1..8]
\end{aligned}
$$

$$
\begin{aligned}
q_{\mathsf{lift}}(\mathbf{iri2}(a),y,z) \leftarrow\ & \mathtt{academic}(a,f_1,l_1,p_1),\\
& \mathtt{student}(s,f_2,l_2),\\
& \mathtt{academic}(a_2,f_3,z,p_3),\\
& \mathbf{iri2}(a) = \mathbf{iri1}(s),\\
& \mathbf{iri2}(a) = \mathbf{iri2}(a_2),\\
& p_1 \in [1..8]
\end{aligned}
$$

(One sub-query not shown)

$$
\begin{aligned}
q_{\mathsf{lift}}(\mathbf{iri2}(a),y,z) \leftarrow\ & \mathtt{academic}(a,f_1,l_1,p_1),\\
& \mathtt{academic}(a_1,y,l_2,p_2),\\
& \mathtt{academic}(a_2,f_3,z,p_3),\\
& \mathbf{iri2}(a) = \mathbf{iri2}(a_1),\\
& \mathbf{iri2}(a) = \mathbf{iri2}(a_2),\\
& p_1 \in [1..8]
\end{aligned}
$$

Reformulation example – 2. Structural optimization

Unfolded normalized query

$$
\begin{aligned}
q_{\text{norm}}(x,y,z) &\leftarrow q1_{\text{unf}}(x),\ q2_{\text{unf}}(x_1,y),\\
&\quad\ q3_{\text{unf}}(x_2,z),\\
&\quad\ x = x_1,\ x = x_2\\
q1_{\text{unf}}(\mathbf{iri2}(a)) &\leftarrow \texttt{academic}(a,f,l,p),\\
&\quad\ p \in [1..8]\\
q1_{\text{unf}}(\mathbf{iri2}(a)) &\leftarrow \texttt{teaching}(c,a)\\
q2_{\text{unf}}(\mathbf{iri1}(s),f) &\leftarrow \texttt{student}(s,f,l)\\
q2_{\text{unf}}(\mathbf{iri2}(a),f) &\leftarrow \texttt{academic}(a,f,l,p)\\
q3_{\text{unf}}(\mathbf{iri1}(s),l) &\leftarrow \texttt{student}(s,f,l)\\
q3_{\text{unf}}(\mathbf{iri2}(a),l) &\leftarrow \texttt{academic}(a,f,l,p)
\end{aligned}
$$

- While flattening, we can avoid to generate those queries that contain in their body an equality between two terms with incompatible IRI templates.
- This might avoid a potential exponential blowup.

Flattening (URI template lifting) – Part 2/2

$$
\begin{aligned}
q_{\text{lift}}(\mathbf{iri2}(a),y,z) &\leftarrow \texttt{teaching}(c,a),\\
&\quad\ \texttt{student}(s,f_2,l_2),\\
&\quad\ \texttt{student}(s_1,f_3,l_3),\\
&\quad\ \mathbf{iri2}(a) = \mathbf{iri1}(s),\\
&\quad\ \mathbf{iri2}(a) = \mathbf{iri1}(s_1)
\end{aligned}
$$

$$
\begin{aligned}
q_{\text{lift}}(\mathbf{iri2}(a),y,z) &\leftarrow \texttt{teaching}(c,a),\\
&\quad\ \texttt{student}(s,f_2,l_2),\\
&\quad\ \texttt{academic}(a_2,f_3,z,p_3),\\
&\quad\ \mathbf{iri2}(a) = \mathbf{iri1}(s),\\
&\quad\ \mathbf{iri2}(a) = \mathbf{iri2}(a_2)
\end{aligned}
$$

(One sub-query not shown)

$$
\begin{aligned}
q_{\text{lift}}(\mathbf{iri2}(a),y,z) &\leftarrow \texttt{teaching}(c,a),\\
&\quad\ \texttt{academic}(a_1,y,l_2,p_2),\\
&\quad\ \texttt{academic}(a_2,f_3,z,p_3),\\
&\quad\ \mathbf{iri2}(a) = \mathbf{iri2}(a_1),\\
&\quad\ \mathbf{iri2}(a) = \mathbf{iri2}(a_2)
\end{aligned}
$$

Reformulation example – 3. Semantic optimization

We are left with just two queries, which we can simplify by eliminating equalities

$$
\begin{aligned}
q_{\text{struct}}(\textbf{iri2}(a), y, z) \leftarrow\ & \texttt{academic}(a, f_1, l_1, p_1),\ p_1 \in [1..8],\\
& \texttt{academic}(a, y, l_2, p_2),\\
& \texttt{academic}(a, f_3, z, p_3)
\end{aligned}
$$

$$
\begin{aligned}
q_{\text{struct}}(\textbf{iri2}(a), y, z) \leftarrow\ & \texttt{teaching}(c, a),\\
& \texttt{academic}(a, y, l_2, p_2),\\
& \texttt{academic}(a, f_3, z, p_3)
\end{aligned}
$$

We can then exploit database constraints (e.g., primary keys) for semantic optimization of the query.

Self-join elimination (semantic optimization)

PK: $\texttt{academic}(acode, f, l, p) \land \texttt{academic}(acode, f', l', p') \rightarrow (f = f') \land (l = l') \land (p = p')$

$$q_{\text{opt}}(\textbf{iri2}(a), y, z) \leftarrow \texttt{academic}(a, y, z, p_1),\ p_1 \in [1..8]$$

$$q_{\text{opt}}(\textbf{iri2}(a), y, z) \leftarrow \texttt{teaching}(c, a),\ \texttt{academic}(a, y, z, p_2)$$

Outline

Support data analytics in VKGs

Supporting data analytics is currently a top priority for us.

Main challenges:

- Semantics: computing aggregation functions correctly, in particular those depending on cardinalities (`SUM`, `COUNT`, `AVG`) – bag vs. set semantics is an issue.
- Performance: efficient computation of aggregates, ideally by delegating their execution to the database.
- Expressiveness: support user-defined aggregation functions beyond the ones in SPARQL 1.1.

noSQL data sources

Prototype extension of *Ontop* over mongoDB. databases.

MongoDB

- Most popular noSQL DBMS.
- Stores data as collections of **JSON** documents.
- Comes with an expressive (low-level) query language: Mongo Aggregate Queries.

Benefits of VGKs over MongoDB:

- **Interface**: higher-level query language (SPARQL) for the end-user.
- **Performance**: *Ontop* delegates query execution to the MongoDB engine
  $\Rightarrow$ leverages document-based storage.

JSON

Document 23226

```
{ _id:        23226,
 productName: "Olympus OM-D E-M10 Mark II",
 offers: [
   { offerId: 258,
     price:   747.14,
     vendor:  {
       vendorId: 3785,
       name:     "Yeppon Italia"
   }},
   { offerId: 895,
     price:   609.42,
     vendor:  {
       vendorId: 481,
       name:     "amazon.it"
   }},
   { offerId: 922,
     price:   759.99,
     vendor:  {
       vendorId: 481,
       name:     "amazon.it"
 }}]}
```

Document 25887

```
{ _id:        25887,
  productName: "Panasonic Lumix DMC-GX80",
  offers: [
    { offerId: 311,
      price:   500.32,
      vendor:  {
        vendorId: 481,
        name:     "amazon.it"
 }}]}
```

Ontop over MongoDB: higher-level queries

- **Mongo Aggregate Queries**: algebra over collections of JSON documents.
- Can be **complex** to read/manipulate.

Retrieve products offered twice by the same vendor

```
db.product.aggregate([
  {$project: {
     "productName": true, "offer1": "$offers", "offer2": "$offers" }},
  {$unwind: "$offer1"},
  {$unwind: "$offer2"},
  {$project: {
     "productName": true, "offer1": true, "offer2": true,
     "sameVendor": { $and: [
         {$ne: ["$offer1.offerId", "$offer2.offerId"]},
         {$eq: ["$offer1.vendorId", "$offer2.vendorId"]} ]}}},
  {$match: {"sameVendor": true}},
  {$project: {"productName": true, $offer1.vendorId, $offer1.name}}
])
```

Ontop over MongoDB: higher-level queries

- **Mongo Aggregate Queries**: algebra over collections of JSON documents.
- Can be **complex** to read/manipulate. *Ontop* provides a simpler interface through SPARQL.

Retrieve products offered twice by the same vendor

```
SELECT ?productName, ?vendorName
WHERE {
  ?product rdfs:label   ?productName .
  ?offer1  bsbm:product ?product .
  ?offer1  bsbm:vendor  ?vendor .
  ?offer2  bsbm:product ?product .
  ?offer2  bsbm:vendor  ?vendor .
  ?vendor  rdfs:label   ?vendorName .
  FILTER (?offer1 != ?offer2)
}
```

Functional dependencies in JSON

Document 23226

```
{ _id:        23226,
 productName: "Olympus OM-D E-M10 Mark II",
 offers: [
   { offerId: 258,
     price:   747.14,
     vendor:  {
       vendorId: 3785,
       name:     "Yeppon Italia"
   }},
   { offerId: 895,
     price:   609.42,
     vendor:  {
       vendorId: 481,
       name:     "amazon.it"
   }},
   { offerId: 922,
     price:   759.99,
     vendor:  {
       vendorId: 481,
       name:     "amazon.it"
 }}]}
```

Document 25887

```
{ _id:        25887,
  productName: "Panasonic Lumix DMC-GX80",
  offers: [
    { offerId: 311,
      price:   500.32,
      vendor:  {
        vendorId: 481,
        name:     "amazon.it"
 }}]}
```

Functional dependency:
`offers.vendor.vendorId` → `offers.vendor.name`
⇝ The database is **not normalized**.

Brings opportunities for query optimization (taking advantage of "precomputed joins").

Querying JSON data: default solution

- Expose JSON data as a (flat) **relational view**. $\rightsquigarrow$ *Ontop* can be used out-of-the-box.
- May be less efficient.

product

id	productName
23226	Olympus OM-D E-M10 Mark II
25887	Panasonic Lumix DMC-GX80

offer

id	price	product	vendor
258	747.14	23226	3785
311	500.32	25887	481
895	609.42	23226	481
922	759.99	23226	481

vendor

id	name
481	amazon.it
3785	Yeppon Italia

Querying JSON data: MongoDB and beyond

- *Ontop* uses **Nested Relational Algebra** as an internal query representation.
- Optimization: leverages functional dependencies, to avoid joins across JSON documents (identify that the information needed is contained in each document).
- Implemented for MongoDB.

Under development

Extension to other query languages with an "unnest"-like operator:

- SPARK (`explode`)
- PostgreSQL (`json_array_elements`)
- etc.

Provenance and explanation

- The base version of *Ontop*, does not provide any information about how query answers are constructed.
- In many cases, we are interested in:
  - which data from which relation/source has been used to obtain an answer
  - which mappings have been activated
  - which ontology axioms have contributed to the answer
- We have developed a framework for provenance/explanation in VKGs, building on provenance semirings in relational databases [C., Lanti, et al. 2019].
- We have developed a prototype extension of *Ontop* that supports this framework.
- We are currently running experiments, and working on performance improvement.

Geospatial extension

Spatial data play an important role in many scenarios.

- Example: find all transactions from the same account that are in two different locations with a distance greater than 1000 km.

Ontop-spatial (http://ontop-spatial.di.uoa.gr/)

- A prototype extension of *Ontop* for accessing geospatial data.
- Supports GeoSPARQL query language standardized by the Open Geospatial Consortium (OGC).
- Use cases: urban development, land management, disaster management.

Temporal extension

Temporal data play an important role in many scenarios.

- Example 1: find all transactions from the same account that are in two different locations with a distance greater than 1000 km and within 5 min.
- Example 2: find all customers with at least 3 temporal overlapping loans within the last 5 years.

Ontop-temporal [Güzel Kalayci, Brandt, et al. 2019; Güzel Kalayci, Xiao, et al. 2018]

- A prototype extension of *Ontop* for accessing temporal data.
- Can express complex temporal patterns.
- Use cases: turbine diagnoses, medical records.

Outline

Conclusions

- VKGs are by now a mature technology to address the data wrangling and data preparation problems.
- However, it has been well-investigated and applied in real-world scenarios mostly for the case of relational data sources.
- Also in that setting, performance and scalability w.r.t. larger datasets (volume), larger and more complex ontologies (variety, veracity), and multiple heterogeneous data sources (variety, volume) is a challenge.
- Only recently VKGs have been investigated for alternative types of data, such as **temporal data**, **noSQL** and tree structured data, **streaming data** (velocity), **linked open data**, and **geo-spatial data**.
- Performance and scalability are even more critical for these more complex domains.

Further research directions

Theoretical investigations:

- Dealing with data provenance and explanation.
- Dealing with data inconsistency and incompleteness – Data quality!
- Ontology-based update.
- More expressive queries, supporting analytical tasks.
- Coping with evolution of data in the presence of ontological constraints.

From a practical point of view, supporting technologies need to be developed to make the VKG technology easier to adopt:

- Improving the support for multiple, heterogeneous data sources.
- Techniques for (semi-)automatic extraction/learning of ontology axioms and mapping assertions.
- Techniques and tools for efficient management of mappings and ontology axioms, to support design, maintenance, and evolution.
- User-friendly ontology querying modalities (graphical query languages, natural language querying).

Outline

Basics of VKG system modeling and usage

Instructions

We will use part of the material of the Ontop tutorial
`https://ontop-vkg.org/tutorial/`

Program

1. Basics of VKG system modeling and usage
   - Mapping the first data source
   - Mapping the second data source
2. Deploying an Ontop SPARQL endpoint
   - Using Ontop CLI
   - Using Ontop Docker image
   - Using Ontop Tomcat bundle
3. Interacting with an Ontop SPARQL endpoint
   - Command line tools (curl, http)
   - Python and Jupyter Notebook

Thanks

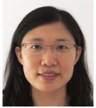
Thank you for your attention!

A great thank you to all my collaborators

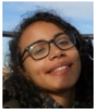

Elena Botoeva

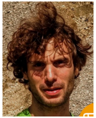

Benjamin Cogrel


Julien Corman


Linfang Ding


Elem Güzel


Sarah Komla Ebri


Davide Lanti


Martin Rezk


Mariano Rodriguez Muro


Guohui Xiao

Univ. Roma “La Sapienza”


Giuseppe De Giacomo


Domenico Lembo


Maurizio Lenzerini


Antonella Poggi


Riccardo Rosati

Birkbeck College London


Roman Kontchakov


Vladislav Ryzhikov


Michael Zakharyaschev

References I

[1] Natalia Antonioli, Francesco Castanò, Spartaco Coletta, Stefano Grossi, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Emanuela Virardi, and Patrizia Castracane. "Ontology-based Data Management for the Italian Public Debt". In: *Proc. of the 8th Int. Conf. on Formal Ontology in Information Systems (FOIS)*. Vol. 267. Frontiers in Artificial Intelligence and Applications. IOS Press, 2014, pp. 372–385.

[2] Franz Baader, Diego C., Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, eds. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[3] Daniela Berardi, Diego C., and Giuseppe De Giacomo. "Reasoning on UML Class Diagrams". In: *Artificial Intelligence* 168.1–2 (2005), pp. 70–118.

[4] Sonia Bergamaschi and Claudio Sartori. "On Taxonomic Reasoning in Conceptual Design". In: *ACM Trans. on Database Systems* 17.3 (1992), pp. 385–422.

[5] Alexander Borgida. "Description Logics in Data Management". In: *IEEE Trans. on Knowledge and Data Engineering* 7.5 (1995), pp. 671–682.

References II

[6] Alexander Borgida and Ronald J. Brachman. “Conceptual Modeling with Description Logics”. In: *The Description Logic Handbook: Theory, Implementation and Applications*. Ed. by Franz Baader, Diego C., Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. Cambridge University Press, 2003. Chap. 10, pp. 349–372.

[7] Stefan Brüggemann, Konstantina Bereta, Guohui Xiao, and Manolis Koubarakis. “Ontology-Based Data Access for Maritime Security”. In: *Proc. of the 13th Extended Semantic Web Conf. (ESWC)*. Vol. 9678. LNCS. Springer, 2016, pp. 741–757. DOI: 10.1007/978-3-319-34129-3_45.

[8] Diego C., Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. “Ontop: Answering SPARQL Queries over Relational Databases”. In: *Semantic Web J.* 8.3 (2017), pp. 471–487. DOI: 10.3233/SW-160217.

[9] Diego C., Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, and Riccardo Rosati. “Ontologies and Databases: The *DL-Lite* Approach”. In: *Reasoning Web: Semantic Technologies for Informations Systems – 5th Int. Summer School Tutorial Lectures (RW)*. Ed. by Sergio Tessaris and Enrico Franconi. Vol. 5689. Lecture Notes in Computer Science. Springer, 2009, pp. 255–356.

References III

[10] Diego C., Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. "The Mastro System for Ontology-Based Data Access". In: *Semantic Web J.* 2.1 (2011), pp. 43–53.

[11] Diego C., Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. "Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family". In: *J. of Automated Reasoning* 39.3 (2007), pp. 385–429.

[12] Diego C., Davide Lanti, Ana Ozaki, Rafael Peñaloza, and Guohui Xiao. "Enriching Ontology-based Data Access with Provenance". In: *Proc. of the 28th Int. Joint Conf. on Artificial Intelligence (IJCAI)*. Int. Joint Conf. on Artificial Intelligence Org., 2019, pp. 1616–1623. DOI: 10.24963/ijcai.2019/224.

[13] Diego C., Maurizio Lenzerini, and Daniele Nardi. "Unifying Class-Based Representation Formalisms". In: *J. of Artificial Intelligence Research* 11 (1999), pp. 199–240.

References IV

[14] Diego C., Pietro Liuzzo, Alessandro Mosca, Jose Remesal, Martin Rezk, and Guillem Rull. "Ontology-Based Data Integration in EPNet: Production and Distribution of Food During the Roman Empire". In: *Engineering Applications of Artificial Intelligence* 51 (2016), pp. 212–229. DOI: 10.1016/j.engappai.2016.01.005.

[15] Elem Güzel Kalayci, Sebastian Brandt, Diego C., Vladislav Ryzhikov, Guohui Xiao, and Michael Zakharyaschev. "Ontology-based Access to Temporal Data with Ontop: A Framework Proposal". In: *Applied Mathematics and Computer Science* 29.1 (2019), pp. 17–30. DOI: 10.2478/amcs-2019-0002.

[16] Elem Güzel Kalayci, Guohui Xiao, Vladislav Ryzhikov, Tahir Emre Kalayci, and Diego C. "Ontop-temporal: A Tool for Ontology-based Query Answering over Temporal Data". In: *Proc. of the 27th ACM Int. Conf. on Information and Knowledge Management (CIKM)*. 2018, pp. 1927–1930. DOI: 10.1145/3269206.3269230.

[17] Evgeny Kharlamov, Dag Hovland, Martin G. Skjæveland, Dimitris Bilidas, et al. "Ontology Based Data Access in Statoil". In: *J. of Web Semantics* 44 (2017), pp. 3–36. DOI: 10.1016/j.websem.2017.05.005.

References V

[18] Evgeny Kharlamov, Theofilos Mailis, Gulnar Mehdi, Christian Neuenstadt, et al. “Semantic Access to Streaming and Static Data at Siemens”. In: *J. of Web Semantics* 44 (2017), pp. 54–74. DOI: 10.1016/j.websem.2017.02.001.

[19] Roman Kontchakov and Michael Zakharyaschev. “An Introduction to Description Logics and Query Rewriting”. In: *Reasoning Web: Reasoning on the Web in the Big Data Era – 10th Int. Summer School Tutorial Lectures (RW)*. Vol. 8714. Lecture Notes in Computer Science. Springer, 2014, pp. 195–244. DOI: 10.1007/978-3-319-10587-1_5.

[20] Maurizio Lenzerini and Paolo Nobili. “On the Satisfiability of Dependency Constraints in Entity-Relationship Schemata”. In: *Information Systems* 15.4 (1990), pp. 453–461.

[21] Vanessa Lopez, Martin Stephenson, Spyros Kotoulas, and Pierpaolo Tommasi. “Data Access Linking and Integration with DALI: Building a Safety Net for an Ocean of City Data”. In: *Proc. of the 14th Int. Semantic Web Conf. (ISWC)*. Vol. 9367. Lecture Notes in Computer Science. Springer, 2015, pp. 186–202.

References VI

[22] Niklas Petersen, Lavdim Halilaj, Irlán Grangel-González, Steffen Lohmann, Christoph Lange, and Sören Auer. "Realizing an RDF-Based Information Model for a Manufacturing Company – A Case Study". In: *Proc. of the 16th Int. Semantic Web Conf. (ISWC)*. Vol. 10588. Lecture Notes in Computer Science. Springer, 2017, pp. 350–366.

[23] Freddy Priyatna, Oscar Corcho, and Juan F. Sequeda. "Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using morph". In: *Proc. of the 23rd Int. World Wide Web Conf. (WWW)*. 2014, pp. 479–490. DOI: 10.1145/2566486.2567981.

[24] Anna Queralt, Alessandro Artale, Diego C., and Ernest Teniente. "OCL-Lite: Finite Reasoning on UML/OCL Conceptual Schemas". In: *Data and Knowledge Engineering* 73 (2012), pp. 1–22.

[25] Alireza Rahimi, Siaw-Teng Liaw, Jane Taggart, Pradeep Ray, and Hairong Yu. "Validating an Ontology-based Algorithm to Identify Patients with Type 2 Diabetes Mellitus in Electronic Health Records". In: *Int. J. of Medical Informatics* 83.10 (2014), pp. 768–778.

References VII

[26] Mariano Rodriguez-Muro, Roman Kontchakov, and Michael Zakharyaschev. "Ontology-Based Data Access: Ontop of Databases". In: *Proc. of the 12th Int. Semantic Web Conf. (ISWC)*. Vol. 8218. Lecture Notes in Computer Science. Springer, 2013, pp. 558–573. DOI: 10.1007/978-3-642-41335-3_35.

[27] Juan F. Sequeda and Daniel P. Miranker. "Ultrawrap: SPARQL Execution on Relational Data". In: *J. of Web Semantics* 22 (2013), pp. 19–39.

[28] Guohui Xiao, Diego C., Roman Kontchakov, Domenico Lembo, Antonella Poggi, Riccardo Rosati, and Michael Zakharyaschev. "Ontology-Based Data Access: A Survey". In: *Proc. of the 27th Int. Joint Conf. on Artificial Intelligence (IJCAI)*. Int. Joint Conf. on Artificial Intelligence Org., 2018, pp. 5511–5519. DOI: 10.24963/ijcai.2018/777.

[29] Guohui Xiao, Linfang Ding, Benjamin Cogrel, and Diego C. "Virtual Knowledge Graphs: An Overview of Systems and Use Cases". In: *Data Intelligence* 1.3 (2019), pp. 201–223. DOI: 10.1162/dint_a_00011.