Ontology-based Data Management

Maurizio Lenzerini

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

SAPIENZA
UNIVERSITÀ DI ROMA

20th ACM Conference on Information and Knowledge Management

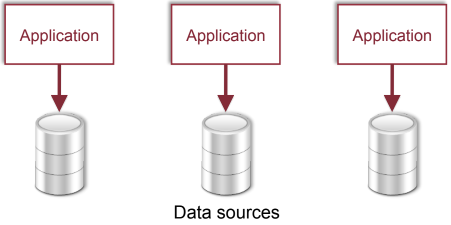
Glasgow, UK, October 24 – 28, 2011

Outline

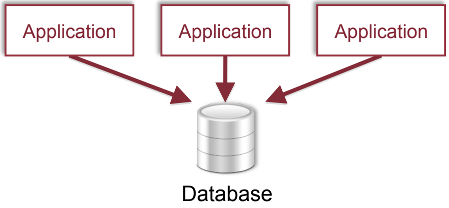
Outline

Information system architecture enabled by DBMS

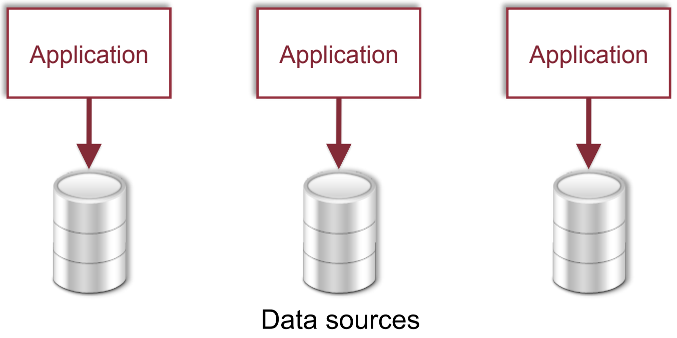
Pre-DBMS architecture (need of a unified data storage):



"Ideal information system architecture" with DBMS ('80s):



Actual information system structure in many organizations



- Distributed, redundant, application-dependent, and mutually incoherent data
- Desperate need of a coherent, conceptual, unified view of data

Information integration

From [Bernstein & Haas, CACM Sept. 2008]:

- Large enterprises spend a great deal of time and money on information integration (e.g., 40% of information-technology shops' budget).
- Market for information integration software estimated to grow from \$2.5 billion in 2007 to \$3.8 billion in 2012 (+8.7% per year) [IDC. Worldwide Data Integration and Access Software 2008-2012 Forecast. Doc No. 211636 (Apr. 2008)]
- Data integration is a large and growing part of software development, computer science, and specific applications settings, such as scientific computing, semantic web, etc..

Basing the information system on a clean, rich and abstract conceptual representation of the data has always been both a goal and a challenge [Mylopoulos et al 1984]

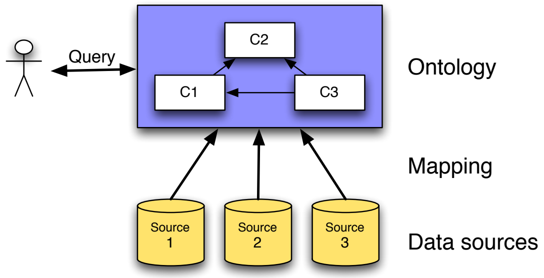
Outline

Ontology-based data management: basic idea

Use Knowledge Representation and Reasoning principles and techniques for a new way of managing data.

- Leave the data where they are
- Build a conceptual specification of the domain of interest, in terms of knowledge structures (**semantic transparency**)
- Map such knowledge structures to concrete data sources
- Express all services over the abstract representation
- Automatically translate knowledge services to data services

Ontology-based data management: architecture



Based on three main components:

- **Ontology**, used as the conceptual layer to give clients a unified conceptual specification of the domain.
- **Data sources**, representing external, independent, heterogeneous, storage (or, more generally, computational) structures.
- **Mappings**, used to semantically link data at the sources to the ontology.

Ontology-based data management (OBDM): topics

- *Ontology-based data access and integration (OBDA)*
- *Ontology-based privacy-aware data access (OBDP)*
- *Ontology-based data quality (OBDQ)*
- *Ontology-based data and service governance (OBDG)*
- *Ontology-based data restructuring (OBDR)*
- *Ontology-based data update (OBDU)*
- *Ontology-based service management (OBDS)*
- *Ontology-based data coordination (OBDC)*

General requirements:

- large data collections
- efficiency with respect to size of data (data complexity)

Formalization of ontology-based data access

An ontology-based data access system is a triple $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{O}$ is the ontology, expressed as TBox in OWL 2 DL (or its logical counterpart $\mathcal{SROIQ}(D)$)
- $\mathcal{S}$ is a (federated) relational database representing the sources
- $\mathcal{M}$ is a set of GLAV mapping assertions, each one of the form

$$\Phi(\vec{x}) \leadsto \Psi(\vec{x})$$

  where

  - $\Phi(\vec{x})$ is a FOL query over $\mathcal{S}$, returning values for $\vec{x}$
  - $\Psi(\vec{x})$ is a FOL query over $\mathcal{O}$, whose free variables are from $\vec{x}$.

Semantics

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation for the ontology $\mathcal{O}$.

Def.: **Semantics**

$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a **model** of $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ if:

- $\mathcal{I}$ is a model of $\mathcal{O}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ wrt $\mathcal{S}$, i.e., satisfies every assertion in $\mathcal{M}$ wrt $\mathcal{S}$.

Def.: **Mapping satisfaction**

We say that $\mathcal{I}$ satisfies $\Phi(\vec{x}) \rightsquigarrow \Psi(\vec{x})$ wrt a database $\mathcal{S}$, if the sentence

$$\forall \vec{x}\ (\Psi(\vec{x}) \rightarrow \Psi(\vec{x}))$$

is true in $\mathcal{I} \cup \mathcal{S}$.

Def.: The **certain answers** to a UCQ $q(\vec{x})$ over $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$

$$cert(q, \mathcal{K}) = \{\ \vec{c}^{\,\mathcal{I}} \in q^{\mathcal{I}} \mid \text{for every model } \mathcal{I} \text{ of } \mathcal{K}\ \}$$

Ontology-based data access: queries

In principle, we are interested in First-order logic (FOL), which is the standard query language for databases. Mostly, we consider **conjunctive queries (CQ)**, i.e., queries of the form (Datalog notation)

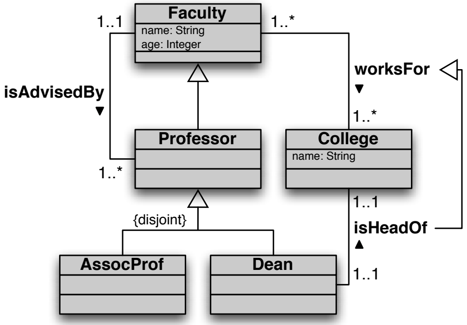
$$q(\vec{x}) \leftarrow R_1(\vec{x}, \vec{y}), \ldots, R_k(\vec{x}, \vec{y})$$

where the lhs is the query head, the rhs is the body, and each $R_i(\vec{x}, \vec{y})$ is an atom using (some of) the free variables $\vec{x}$, the existentially quantified variables $\vec{y}$, and possibly constants.

- CQs contain no disjunction, no negation, no universal quantification.
- Correspond to SQL/relational algebra **select-project-join (SPJ) queries** – the most frequently asked queries.
- They can also be written as **SPARQL** queries.
- A Union of CQs (UCQ) is a set of CQs with the same head predicate.

Example of query

Consider the following ontology (represented as a UML class diagram).



$$q(nf, af, nd) \leftarrow \text{worksFor}(f, c) \wedge \text{isHeadOf}(d, c) \wedge \text{name}(f, nf) \wedge \text{name}(d, nd) \wedge \text{age}(f, x) \wedge \text{age}(d, x)$$

Query: return name, age, and name of dean of all faculty that have the same age as their dean.

Outline

Which languages?

- Which language for expressing queries over the ontology?
- Which language for the mappings?
- Which language for the ontology?

Challenge: optimal compromise between expressive power and data complexity.

Query language for user queries

- Answering FOL queries is **undecidable**, even if the ontology is empty, and the set of mappings is empty.
- Unions of conjunctive queries (UCQs) do not suffer from this problem.
- We can go beyond unions of conjunctive queries without falling into undecidability, but we get intractability in data complexity very soon.

Which languages?

- Which language for expressing queries over the ontology?
  - Essentially UCQs
- Which language for the mappings?
- Which language for the ontology?

Challenge: optimal compromise between expressive power and data complexity.

Query languages for the mappings

$\mathcal{O}$	lhs of $\mathcal{M}$	rhs of $\mathcal{M}$	Query language	Query answering
$\emptyset$	single atom	FOL	single atom	undecidable (1)
$\emptyset$	single atom	UCQ	single atom	NP-complete (2)
$\emptyset$	FOL	CQ	UCQ	AC^0 (3)

(1) *(Abiteboul & Duschka, PODS'98)*.
(2) *(van Der Meyden, TCS'93; Abiteboul & Duschka, PODS'98)*
(3) *(Duschka & Genesereth, PODS'97; Pottinger & Levy VLDBJ 2001)*.

We measure the computational complexity of query answering with respect to the size of the data at $\mathcal{S}$ (data complexity)

Note: $AC^0 \subseteq$ LogSpace, and going beyond LogSpace means going beyond relational databases

Impedance mismatch problem

The impedance mismatch problem

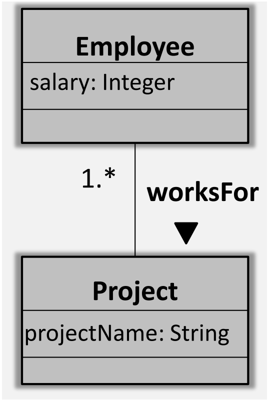
- In **relational databases**, information is represented in forms of tuples of **values**.
- In **ontologies** (or more generally object-oriented systems or conceptual models), information is represented using both **objects** and values ...
  - ... with objects playing the main role, ...
  - ... and values a subsidiary role as fillers of object's attributes.

$\leadsto$ How do we reconcile these views?

Solution: We need **constructors** to create objects of the ontology out of tuples of values in the database.

Note: from a formal point of view, such constructors can be simply Skolem functions!

Impedance mismatch – Example



Actual data is stored in a DB:

$D_1[\mathit{SSN}: \text{String}, \mathit{PrName}: \text{String}]$
Employees and Projects they work for

$D_2[\mathit{Code}: \text{String}, \mathit{Salary}: \text{Int}]$
Employee's Code with salary

$D_3[\mathit{Code}: \text{String}, \mathit{SSN}: \text{String}]$
Employee's Code with SSN

...

From the domain analysis it turns out that (**pers** and **proj** Skolem functions):

- An employee should be created from her *SSN*: **pers**(*SSN*)
- A project should be created from its *Name*: **proj**(*PrName*)

If VRD56B25 is a *SSN*, then **pers**(VRD56B25) is an **object term** denoting a person.

Impedance mismatch: the technical solution

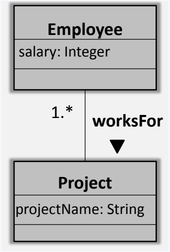
Creating object identifiers

To denote objects, i.e., instances of concepts in the ontology, we use **object terms** of the form $\mathbf{f}(d_1, \ldots, d_n)$, where **f** is a function symbol of arity $n > 0$, and each d_i is value constant retrieved from the sources.

$\rightsquigarrow$ *No confusion between the values stored in the database and the terms denoting objects.*

Ontology-based data access system – Example

Ontology $\mathcal{O}$ (UML)



federated schema of the DB $\mathcal{S}$

$D_1[\mathit{SSN}\colon \text{String}, \mathit{PrName}\colon \text{String}]$
Employees and Projects they work for

$D_2[\mathit{Code}\colon \text{String}, \mathit{Salary}\colon \text{Int}]$
Employee's Code with salary

$D_3[\mathit{Code}\colon \text{String}, \mathit{SSN}\colon \text{String}]$
Employee's Code with SSN

$D_4[\mathit{SSN}\colon \text{String}, \mathit{Tel}\colon \text{String}]$
Employees of the production department: they work for at least one Project

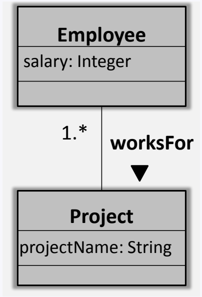
Mapping $\mathcal{M}$

M_1:
```
SELECT SSN, PrName
FROM D1
```
$\leadsto$ Employee(**pers**(*SSN*)),
Project(**proj**(*PrName*)),
projectName(**proj**(*PrName*), *PrName*),
worksFor(**pers**(*SSN*), **proj**(*PrName*))

M_2:
```
SELECT SSN, Salary
FROM D2, D3
WHERE D2.Code = D3.Code
```
$\leadsto$ Employee(**pers**(*SSN*)),
salary(**pers**(*SSN*), *Salary*)

Ontology-based data integration system – Example

Ontology $\mathcal{O}$ (UML)



federated schema of the DB $\mathcal{S}$

D_1[*SSN*: String, *PrName*: String]
Employees and Projects they work for

D_2[*Code*: String, *Salary*: Int]
Employee's Code with salary

D_3[*Code*: String, *SSN*: String]
Employee's Code with SSN

D_4[*SSN*: String, *Tel*: String]
Employees of the production department: they work for at least one Project

Mapping $\mathcal{M}$

M_3: `SELECT SSN` $\rightsquigarrow$ worksFor(**pers**(*SSN*), y)
`FROM` D_4

Which languages?

- Which language for expressing queries over the ontology?
  - Essentially UCQs
- Which language for the mappings?
  - FOL-to-CQ, with object constructors
- Which language for the ontology?

Challenge: optimal compromise between expressive power and data complexity.

Ontologies with large number of instances

- The best current ontology reasoning systems can deal with a moderately large instance level. $\rightsquigarrow 10^4$ individuals *(and this is a big achievement of the last years)!*
- But data of interests in typical information systems (and in data integration) are much **larger**
  $\rightsquigarrow 10^6 - 10^9$ individuals

Question

How can we use ontologies together with large amounts of instances?

Query answering in Description Logic Ontologies

To address these questions, we proceed in two steps

1. we fist deal with the problem of answering queries posed to a "stand-alone" DL ontology.
   A **"stand-alone" DL ontology** $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is constituted by a TBox $\mathcal{T}$ (general axioms on concepts and roles) and ABox $\mathcal{A}$ (facts).
2. We then tackle the problem of ansering queries to an ontology-based data integration system $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ where the TBox is now considered "the ontology" ($\mathcal{O}$), and the ABox $\mathcal{A}$ is replaced by $\mathcal{S}$ and $\mathcal{M}$.

Query answering in Description Logic Ontologies

DL	Data complexity of query answering
$\mathcal{SROIQ}(D)$	? (1)
$\mathcal{SHIQ}(D)$	coNP-complete (2)
?	AC^0 (3)

(1) It is in fact open whether answering CQs over OWL 2 DL (i.e., $\mathcal{SROIQ}(D)$) ontologies is decidable.

(2) *(Hustadt & al., IJCAI'05; Glimm & al., JAIR'08; Ortiz & al., JAIR'08)*. In fact, *(Calvanese & al., KR'06)* show coNP-hardness for very simple languages (fragments of OWL 2 DL) allowing for union.

(3) Question: Are there significative fragments of OWL 2 DL for which answering CQs has the same complexity as SQL query evaluation over a database instance?

The *DL-Lite* family

- A family of Description Logics (DLs) optimized according to the trade-off between expressive power and **complexity** of query answering, with emphasis on **data**.
- Carefully designed to have nice computational properties for answering UCQs (i.e., computing certain answers):
  - The same complexity as relational databases.
  - In fact, query answering can be delegated to a relational DB engine.
  - The DLs of the *DL-Lite* family are essentially the maximally expressive ontology languages enjoying these nice computational properties.
- We present $\textit{DL-Lite}_{\mathcal{R}}$, a member of the *DL-Lite* family.
- $\textit{DL-Lite}_{\mathcal{R}}$ essentially corresponds to **OWL 2 QL**, one of the three candidates **OWL 2 Profiles**.
- Extends (the DL fragment of) the ontology language **RDFS**.

DL-Lite$_{\mathcal{R}}$ ontologies

TBox assertions:

- Concept inclusion assertions: $Cl \sqsubseteq Cr$, with:

$$\begin{array}{lcl} Cl & \longrightarrow & A \mid \exists Q \\ Cr & \longrightarrow & A \mid \exists Q \mid \neg A \mid \neg\exists Q \\ Q & \longrightarrow & P \mid P^{-} \end{array}$$

- Property inclusion assertions: $Q \sqsubseteq R$, with:

$$R \longrightarrow Q \mid \neg Q$$

ABox assertions: $A(c)$, $P(c_1, c_2)$, with c_1, c_2 constants

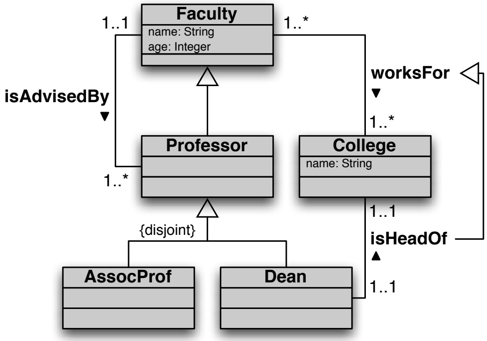
Note: *DL-Lite*$_{\mathcal{R}}$ can be straightforwardly adapted to distinguish also between object and data properties (attributes).

Semantics of *DL-Lite*$_\mathcal{R}$

Construct	Syntax	Example	Semantics
atomic conc.	A	Doctor	$A^\mathcal{I} \subseteq \Delta^\mathcal{I}$
exist. restr.	$\exists Q$	∃child⁻	$\{d \mid \exists e. (d,e) \in Q^\mathcal{I}\}$
at. conc. neg.	$\neg A$	¬Doctor	$\Delta^\mathcal{I} \setminus A^\mathcal{I}$
conc. neg.	$\neg \exists Q$	¬∃child	$\Delta^\mathcal{I} \setminus (\exists Q)^\mathcal{I}$
atomic role	P	child	$P^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$
inverse role	P^-	child⁻	$\{(o,o') \mid (o',o) \in P^\mathcal{I}\}$
role negation	$\neg Q$	¬manages	$(\Delta^\mathcal{I} \times \Delta^\mathcal{I}) \setminus Q^\mathcal{I}$
conc. incl.	$Cl \sqsubseteq Cr$	Father ⊑ ∃child	$Cl^\mathcal{I} \subseteq Cr^\mathcal{I}$
role incl.	$Q \sqsubseteq R$	hasFather ⊑ child⁻	$Q^\mathcal{I} \subseteq R^\mathcal{I}$
mem. asser.	$A(c)$	Father(bob)	$c^\mathcal{I} \in A^\mathcal{I}$
mem. asser.	$P(c_1, c_2)$	child(bob, ann)	$(c_1^\mathcal{I}, c_2^\mathcal{I}) \in P^\mathcal{I}$

DL-Lite$_\mathcal{R}$ (as all DLs of the *DL-Lite* family) adopts the Unique Name Assumption (UNA), i.e., different individuals denote different objects. However, reasoning in *DL-Lite*$_\mathcal{R}$ would have been the same even without UNA.

Example



$$
\begin{array}{rcl}
\mathsf{Professor} & \sqsubseteq & \mathsf{Faculty} \\
\mathsf{AssocProf} & \sqsubseteq & \mathsf{Professor} \\
\mathsf{Dean} & \sqsubseteq & \mathsf{Professor} \\
\mathsf{AssocProf} & \sqsubseteq & \neg\mathsf{Dean} \\
\exists\mathsf{worksFor} & \sqsubseteq & \mathsf{Faculty} \\
\exists\mathsf{worksFor}^- & \sqsubseteq & \mathsf{College} \\
\mathsf{Faculty} & \sqsubseteq & \exists\mathsf{worksFor} \\
\mathsf{College} & \sqsubseteq & \exists\mathsf{worksFor}^- \\
\exists\mathsf{isHeadOf} & \sqsubseteq & \mathsf{Dean} \\
\exists\mathsf{isHeadOf}^- & \sqsubseteq & \mathsf{College} \\
\mathsf{Dean} & \sqsubseteq & \exists\mathsf{isHeadOf} \\
\mathsf{College} & \sqsubseteq & \exists\mathsf{isHeadOf}^- \\
\mathsf{isHeadOf} & \sqsubseteq & \mathsf{worksFor} \\
 & \vdots &
\end{array}
$$

UML attributes can be captured considering the extension of *DL-Lite*$_{\mathcal{R}}$ to data properties.

Technical properties of *DL-Lite*: no finite model property

DL-Lite does **not** enjoy the **finite model property**.

Example

TBox $\mathcal{T}$: $\mathsf{Nat} \sqsubseteq \exists\mathsf{succ}$ $\exists\mathsf{succ}^- \sqsubseteq \mathsf{Nat}$

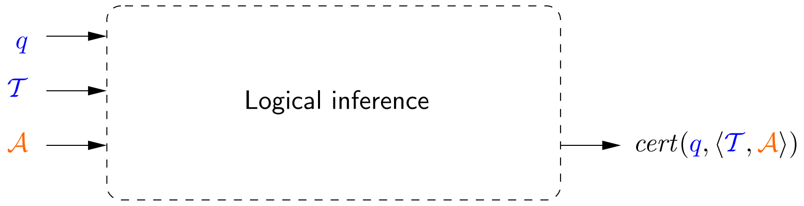
$\mathsf{Zero} \sqsubseteq \mathsf{Nat}$ $\mathsf{Zero} \sqsubseteq \neg\exists\mathsf{succ}^-$ $(\textbf{funct}\ \mathsf{succ}^-)$

ABox $\mathcal{A}$: $\mathsf{Zero}(0)$

$\mathcal{K} = \langle\mathcal{T},\mathcal{A}\rangle$ admits only infinite models.
Hence, it is satisfiable, but **not finitely satisfiable**.

Hence, reasoning w.r.t. arbitrary models is different from reasoning w.r.t. finite models only.

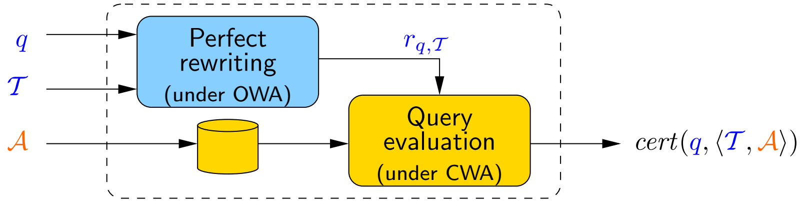
Inference in query answering



To be able to deal with data efficiently, we need to separate the contribution of $\mathcal{A}$ from the contribution of q and $\mathcal{T}$.

$\leadsto$ Query answering by **query rewriting**.

Query rewriting


q
$\mathcal{T}$
$\mathcal{A}$
Perfect rewriting (under OWA)
$r_{q,\mathcal{T}}$
Query evaluation (under CWA)
$cert(q, \langle\mathcal{T}, \mathcal{A}\rangle)$

Query answering can **always** be thought of as done in two phases:

1. **Perfect rewriting**: produce from q and the TBox $\mathcal{T}$ a new query $r_{q,\mathcal{T}}$ (called the perfect rewriting of q w.r.t. $\mathcal{T}$).
2. **Query evaluation**: evaluate $r_{q,\mathcal{T}}$ over the ABox $\mathcal{A}$ seen as a complete database (and without considering the TBox $\mathcal{T}$).
   $\leadsto$ Produces $cert(q, \langle\mathcal{T}, \mathcal{A}\rangle)$.

Note: The "always" holds if we pose no restriction on the language in which to express the rewriting $r_{q,\mathcal{T}}$.

Language of the rewriting

The expressiveness of the ontology language affects the **query language into which we are able to rewrite UCQs**

Complexity of rewriting language	We need at least
AC^0	FOL/SQL (1)
NLogSpace-hard	Linear Datalog
PTime-hard	Datalog
coNP-hard	Disjunctive Datalog

(1) **FOL-rewritability**: relational database technology (SQL) suffices

Query answering in $DL\text{-}Lite_{\mathcal{R}}$

Given an (U)CQ Q and a consistent* ontology $\langle \mathcal{T}, \mathcal{A} \rangle$:

1. **Compute its perfect rewriting**, PerfectRef$(Q, \mathcal{T})$, which turns out to be a UCQ.
2. **Evaluate the perfect rewriting** on the ABox seen as a DB.

To compute the perfect rewriting, starting from the original (U)CQ, iteratively get a CQ to be processed and either:

- **expand** positive inclusions, i.e., $Cl \sqsubseteq A \mid \exists Q$ or $Q \sqsubseteq Q'$, or
- **unify** atoms in the CQ to obtain a more specific CQ to be further expanded

ensuring termination, by carefully choosing new variables in the rewriting.
Each result of the above steps is added to the queries to be processed.

* *We will come back to the case of inconsistent ontology*

Query answering in $\mathit{DL\text{-}Lite}_{\mathcal{R}}$ – Example

TBox: $\mathsf{Professor} \sqsubseteq \exists\mathsf{worksFor}$
$\exists\mathsf{worksFor}^{-} \sqsubseteq \mathsf{College}$

Query: $\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x,y), \mathsf{College}(y)$

Perfect Reformulation:
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x,y), \mathsf{College}(y)$
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x,y), \mathsf{worksFor}(z,y)$
$\mathsf{q}(x) \leftarrow \mathsf{worksFor}(x,z)$
$\mathsf{q}(x) \leftarrow \mathsf{Professor}(x)$

ABox: $\mathsf{worksFor}(\mathtt{john}, \mathtt{collA})$ $\mathsf{Professor}(\mathtt{john})$
$\mathsf{worksFor}(\mathtt{mary}, \mathtt{collB})$ $\mathsf{Professor}(\mathtt{nick})$

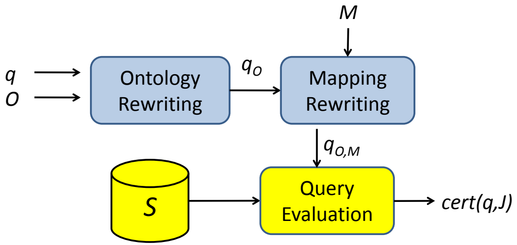
Evaluating the last two queries over the ABox (seen as a DB) produces as answer $\{\mathtt{john}, \mathtt{nick}, \mathtt{mary}\}$.

Using *DL-Lite*$_{\mathcal{R}}$ in ontology-based data integration

We go back to an OBDA system $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ such that

- $\mathcal{O}$ is a *DL-Lite*$_{\mathcal{R}}$ TBox
- $\mathcal{S}$ is a relational database
- $\mathcal{M}$ is a set of GLAV mapping assertions of the form that we have seen before

We extend the notion of perfect rewriting to such a setting.



$q_{\mathcal{O},\mathcal{M}}$ is the **perfect reformulation** of q w.r.t. $\mathcal{K}$

$$q_{\mathcal{O},\mathcal{M}} = \text{MapRewriting}_{\mathcal{M}}(\text{PerfectRef}(Q, \mathcal{O}))$$

Computational complexity of query answering

Theorem

Query answering in an ontology-based data integration system $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ of the kind considered so far is

1. **NP-complete** in the size of the query.
2. **PTime** in the size of the **ontology** $\mathcal{O}$ and the **mappings** $\mathcal{M}$.
3. **AC^0** in the size of the **database** $\mathcal{S}$, in fact FOL-rewritable.

Note: In fact, we can we adopt a *DL-Lite* logic with functionalities and identification assertions (*DL-Lite*$_{\mathcal{A},id}$) to specify $\mathcal{O}$, but only coupled with GAV mappings, otherwise query answering becomes NLogSpace-hard *(Calvanese & al., SKDB'08)*.

Can we extend the framework?
Essentially no, if we want to stay in AC^0.

Beyond *DL-Lite*$_\mathcal{R}$: results on data complexity

	lhs	rhs	funct.	Prop. incl.	Data complexity of query answering
0	*DL-Lite*$_{\mathcal{A},id}$		–	$\surd$	in AC^0
1	$A \mid \exists P.A$	A	–	–	NLogSpace-hard
2	A	$A \mid \forall P.A$	–	–	NLogSpace-hard
3	A	$A \mid \exists P.A$	$\surd$	–	NLogSpace-hard
4	$A \mid \exists P.A \mid A_1 \sqcap A_2$	A	–	–	PTime-hard
5	$A \mid A_1 \sqcap A_2$	$A \mid \forall P.A$	–	–	PTime-hard
6	$A \mid A_1 \sqcap A_2$	$A \mid \exists P.A$	$\surd$	–	PTime-hard
7	$A \mid \exists P.A \mid \exists P^-.A$	$A \mid \exists P$	–	–	PTime-hard
8	$A \mid \exists P \mid \exists P^-$	$A \mid \exists P \mid \exists P^-$	$\surd$	$\surd$	PTime-hard
9	$A \mid \neg A$	A	–	–	**coNP-hard**
10	A	$A \mid A_1 \sqcup A_2$	–	–	**coNP-hard**
11	$A \mid \forall P.A$	A	–	–	**coNP-hard**

- Giving up property inclusions from *DL-Lite*$_\mathcal{R}$ allows for having functional roles, remaining in AC^0 (cf. *DL-Lite*$_\mathcal{F}$). Prop. incl. and funct. can be also used together (cf. *DL-Lite*$_\mathcal{A}$), *provided that functional properties are not specialized*.
- NLogSpace and PTime hardness holds already for instance checking.
- For coNP-hardness in line 10, a TBox with a single assertion $A_L \sqsubseteq A_T \sqcup A_F$ suffices! $\leadsto$ **No** hope of including **covering constraints**.

Which languages?

- Which language for expressing queries over the ontology?
  - Essentially UCQs
- Which language for the mappings?
  - FOL-to-full-CQ (GAV), with object constructors
- Which language for the ontology?
  - $\textit{DL-Lite}_{\mathcal{A},id}$

Challenge: optimal compromise between expressive power and data complexity.

Outline

The problem

One popular approach to dealing with inconsistency in data management is data cleaning

However, even with data cleaning, inconsistencies may remain, and we would like our system to provide meaningful answers to queries.

The problem is that query answering based on classical logic becomes meaningless in the presence of inconsistency (ex falso quodlibet)

Question

How to handle classically-inconsistent ontologies in a more meaningful way?

Example: an inconsistent *DL-Lite* ontology

$\mathcal{O}$

RedWine $\sqsubseteq$ Wine	WhiteWine $\sqsubseteq$ Wine
RedWine $\sqsubseteq \neg$ WhiteWIne	Wine $\sqsubseteq \neg$ Beer
Wine $\sqsubseteq \exists$producedBy	$\exists$producedBy $\sqsubseteq$Wine
Wine $\sqsubseteq \neg$ Winery	Beer $\sqsubseteq \neg$ Winery
$\exists$producedBy$^-$ $\sqsubseteq$ Winery	(*funct* producedBy)

$\mathcal{M}$

R1(x,y,'white') $\rightsquigarrow$ WhiteWine(x)   R1(x,y,'red') $\rightsquigarrow$ RedWine(x)
R2(x,y) $\rightsquigarrow$ Beer(x)   R1(x,y,z) $\vee$ R2(x,y) $\rightsquigarrow$ producedBy(x,y)

$\mathcal{S}$

R1(grechetto,p1,'white')	R1(grechetto,p1,'red')
R2(guinnes,p2)	R1(falanghina,p1,'white')

Inconsistent-tolerant semantics

The semantics we propose for inconsistent OBDA systems is based on the following principles:

- We assume that $\mathcal{O}$ and $\mathcal{M}$ are always consistent (this is true if $\mathcal{O}$ is expressed in *DL-Lite*$_{\mathcal{A},id}$)
- Inconsistencies are caused by the interaction between the data at $\mathcal{S}$ and the other components of the system
- We resort to the notion of *repair* [Arenas, Bertossi, Chomicki, PODS 1999]. Intuitively, a repair for $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ is an ontology $\langle \mathcal{O}, \mathcal{A} \rangle$ that is consistent, and "minimally" differs from $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$.

The notion of “minimally different”

What does it mean to be “minimally different” from $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$? Since $\mathcal{O}$ and $\mathcal{M}$ cannot change, one might be tempted to base the notion on the difference with $\mathcal{S}$. However, this would neglect the impact of $\mathcal{S}$ on the models of the OBDI systems (which are based on $\mathcal{O}$).

So, we base our concept of distance on a new notion, namely $\mathcal{M}(\mathcal{S})$.

Definition ($\mathcal{M}(\mathcal{S})$)

Given $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$, $\mathcal{M}(\mathcal{S})$ is the ABox obtained by computing the tuples obtained by evaluating the queries in the lhs of the mappings, and “transferring” such tuples to the rhs.

The notion of “minimally different”

$\mathcal{M}$

R1(x,y,‘white’) $\rightsquigarrow$ WhiteWine(x) R1(x,y,‘red’) $\rightsquigarrow$ RedWine(x)
R2(x,y) $\rightsquigarrow$ Beer(x) R1(x,y,z) $\vee$ R2(x,y) $\rightsquigarrow$ producedBy(x,y)

$\mathcal{S}$

R1(grechetto,p1,‘white’) R1(grechetto,p1,‘red’)
R2(guinnes,p2) R1(falanghina,p1,‘white’)

$\mathcal{M}(\mathcal{S})$

WhiteWine(grechetto) RedWine(grechetto)
Beer(guinnes) ProducedBy(guiness,p2)
ProducedBy(grechetto,p1) WhiteWine(falanghina)
ProducedBy(falanghina,p1)

Inconsistent-tolerant semantics

We write $S_1 \oplus S_2$ to denote the symmetric difference between S_1 and S_2, i.e.,

$$S_1 \oplus S_2 = (S_1 \setminus S_2) \cup (S_1 \setminus S_2)$$

Definition (Repair)

Let $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system. A **repair** of $\mathcal{K}$ is a pair $\langle \mathcal{O}, \mathcal{M}(S') \rangle$ such that:

1. $\mathit{Mod}(\langle \mathcal{O}, \mathcal{M}(S') \rangle) \neq \emptyset$,
2. no set of facts $\mathcal{A}$ exists such that
   - $\mathit{Mod}(\langle \mathcal{O}, \mathcal{A} \rangle) \neq \emptyset$,
   - $\mathcal{A} \oplus \mathcal{M}(\mathcal{S}) \subset \mathcal{M}(\mathcal{S}') \oplus \mathcal{M}(\mathcal{S})$

The set of repairs for $\mathcal{K}$ is denoted by $\mathit{Rep}(\mathcal{K})$.

Example: Repairs

Rep_1

{WhiteWine(grechetto), Beer(guinnes), WhiteWine(falanghina)}

Rep_2

{RedWine(grechetto), Beer(guinnes), WhiteWine(falanghina)}

Rep_3

{WhiteWine(grechetto), producedBy(guinnes, p2),
WhiteWine(falanghina)}

Rep_4

{RedWine(grechetto), producedBy(guinnes, p2),
WhiteWine(falanghina)}

Inconsistent-tolerant semantics

Definition (Repair model)

Let $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system. An interpretation $\mathcal{I}$ is a **repair model**, or simply an R-*model*, of $\mathcal{K}$ if there exists $\langle \mathcal{T}, \mathcal{A} \rangle \in$ *Rep*$(\mathcal{K})$ such that $\mathcal{I} \models \langle \mathcal{T}, \mathcal{A} \rangle$. The set of repair models is denoted by *R-Mod*$(\mathcal{K})$.

The following notion of consistent entailment is the natural generalization of classical entailment to the repair semantics.

Definition (AR-entailment)

Let ϕ be a first-order sentence. We say that ϕ is r-**consistently entailed**, or simply R-*entailed*, by $\mathcal{K}$, written $\mathcal{K} \models_R \phi$, if $\mathcal{I} \models \phi$ for every $\mathcal{I} \in$ *R-Mod*$(\mathcal{K})$.

Inconsistent-tolerant semantics

Problems:

- Many repairs in general
- What is the complexity of reasoning about all such repairs?

Theorem

Let $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system, and let α be a ground atom. Deciding whether $\mathcal{K} \models_R \alpha$ is coNP-complete with respect to data complexity.

Idea

Consider the "intersection of all repairs", and consider the set of models of such intersection as the semantics of the system (When in Doubt, Throw It Out).

Inconsistent-tolerant semantics

Definition

Let $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system. An **Intersection Repair (IR)** for $\mathcal{K}$ is a pair $\langle \mathcal{O}, \mathcal{A} \rangle$ such that

$$\mathcal{A} = \bigcap_{\mathcal{A}' \in \textit{Rep}(\mathcal{K})} \mathcal{A}'$$

The set of all IR-repairs for $\mathcal{K}$ is denoted by *IR-Rep*$(\mathcal{K})$.

Example (IR Semantics)

IR-Rep$(\mathcal{K})$ is the singleton formed by the ABox
$Rep_1 \cap Rep_2 \cap Rep_3 \cap Rep_4 = \{\text{WhiteWine(falanghina)}\}$.

Inconsistent-tolerant semantics

Definition (IAR-repair model)

Let $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system. An interpretation $\mathcal{I}$ is an **Intersection Repair model**, or simply an IR*-model*, of $\mathcal{K}$ if there exists $\langle \mathcal{T}, \mathcal{S}', \mathcal{M} \rangle \in$ *IR-Rep*$(\mathcal{K})$ such that $\mathcal{I} \models \langle \mathcal{T}, \mathcal{S}', \mathcal{M} \rangle$. The set of Intersection Repair models is denoted by *IR-Mod*$(\mathcal{K})$.

Definition

Let $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA system, and let ϕ be a first-order sentence. We say that ϕ is IR**-consistently entailed**, or simply IR*-entailed*, by $\mathcal{K}$, written $\mathcal{K} \models_{IR} \phi$, if $\mathcal{I} \models \phi$ for every $\mathcal{I} \in$ *IR-Mod*$(\mathcal{K})$.

Inconsistent-tolerant query answering

Two possible methods for answering queries posed to $\mathcal{K} = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ according to the inconsistency-tolerant semantics:

- Compute $\langle \mathcal{O}, \mathcal{A} \rangle \in$ *IR-Rep*$(\mathcal{K})$, and then compute $\vec{t}$ such that $\langle \mathcal{O}, \mathcal{A} \rangle \models q(\vec{t})$
- Rewrite the query q into q' in such a way that, for all $\vec{t}$, we have that $\mathcal{K} \models_{IR} q(\vec{t})$ is equivalent to $\vec{t} \in q'(\mathcal{S})$. Then, evaluate q' over $\mathcal{S}$.

Rewriting technique

We provide a rewriting technique which encodes a UCQ Q into a FOL query Q' which evaluated against the original $\mathcal{S}$ retrieves only the certain answers of Q w.r.t the IR semantics

Rewriting technique

Given a UCQ $Q = q_1 \vee q_2 \vee \ldots \vee q_n$ over $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$

- we compute $\mathsf{PerfectRef}_{IR}(Q, \mathcal{O}, \mathcal{M})$ as
  $$\mathsf{MapRewriting}_{\mathcal{M}}(\mathsf{IncRewritingUCQ}_{IR}(\mathsf{PerfectRef}(Q, \mathcal{O}), \mathcal{O}))$$
- we evaluate $\mathsf{PerfectRef}_{IR}(Q, \mathcal{O}, \mathcal{M})$ over $\mathcal{S}$

where

- $\mathsf{PerfectRef}(Q, \mathcal{O})$ rewrites Q taking care of $\mathcal{O}$
- $\mathsf{IncRewritingUCQ}_{IR}(Q, \mathcal{O}) = \bigvee_{i=1}^{n} \mathsf{IncRewriting}(q_i, \mathcal{O})$ rewrites Q taking care of inconsistencies
- $\mathsf{MapRewriting}_{\mathcal{M}}(Q)$ rewrites Q taking care of $\mathcal{M}$

Inconsistency on disjointness assertions

Given a disjointness assertion $B \sqsubseteq \neg C$ implied by $\mathcal{O}$, an inconsistency may arise if both $B(a)$ and $C(a)$ are in $\mathcal{M}(\mathcal{S})$, for some constant a. Analogously, given $B \sqsubseteq \neg\exists P$ (resp. $B \sqsubseteq \neg\exists P^-$), an inconsistency may arise if $B(a)$ and $P(a,b)$ (resp. $P(b,a)$) belong to $\mathcal{M}(\mathcal{S})$.
In order to characterize such inconsistencies, given a concept B we define $\mathsf{NotDisjClash}_B^{\mathcal{O}}(t)$ as the following FOL formula:

$$\bigwedge_{C \in DC(B,\mathcal{O})} \neg C(t) \quad \bigwedge_{P \in DRD(B,\mathcal{O})} \neg\exists y.P(t,y) \quad \bigwedge_{P \in DRR(B,\mathcal{O})} \neg\exists y.P(y,t)$$

where
$DC(B,\mathcal{O}) = \{C \mid C \text{ is an atomic concept s.t. } \mathcal{O} \models B \sqsubseteq \neg C\}$
$DRD(B,\mathcal{O}) = \{P \mid P \text{ is an atomic role s.t. } \mathcal{O} \models B \sqsubseteq \neg\exists P\}$
$DRR(B,\mathcal{O}) = \{P \mid P \text{ is an atomic role s.t. } \mathcal{O} \models B \sqsubseteq \neg\exists P^-\}$

Inconsistency on disjointness assertions: example

$\mathcal{O}$

RedWine $\sqsubseteq$ Wine	WhiteWine $\sqsubseteq$ Wine
RedWine $\sqsubseteq \neg$ WhiteWine	Beer $\sqsubseteq \neg$ Wine
Wine $\sqsubseteq \exists$producedBy	$\exists$producedBy $\sqsubseteq$Wine
$\exists$producedBy$^-$ $\sqsubseteq$ Winery	Beer $\sqsubseteq \neg$ $\exists$producedBy
Wine $\sqsubseteq \neg$ Winery	Beer $\sqsubseteq \neg$ Winery
(*funct* producedBy)	

Due to RedWine $\sqsubseteq$ $\neg$RedWine we have that:

$$\mathsf{NotDisjClash}^{\mathcal{O}}_{\mathsf{RedWine}}(t) = \neg\mathsf{WhiteWine}(t) \wedge \neg\mathsf{Winery}(t) \wedge \neg\mathsf{Beer}(t)$$

and due to Beer $\sqsubseteq$ $\neg\exists$ProducedBy we have that:

$$\mathsf{NotDisjClash}^{\mathcal{O}}_{Beer}(t) = \neg\exists y.\mathsf{producedBy}(t,y) \wedge \neg\mathsf{Wine}(t)\wedge$$
$$\neg\mathsf{RedWine}(t) \wedge \neg\mathsf{WhiteWine}(t) \wedge \neg\mathsf{Winery}(T)$$

Inconsistency on functionalities

Given a functionality assertion $(funct\ P)$ over a role P, an inconsistency may arise if the assertions $P(a, b)$ and $P(a, c)$ belong to the ABox A.

Analogously, given a functionality assertion $(funct\ P^-)$ over a role P^-, an inconsistency may arise if $P(a, b)$ and $P(a, c)$ belong to the ABox A.

In order to detect such inconsistencies, given a role P we define NotFunctClash$_P^{\mathcal{O}}(t, t')$ as the FOL formula:

- $\neg(\exists y.P(t, y) \land y \neq t')$, if $(funct\ P)$ exists in the ontology, and
- $\neg(\exists y.P(y, t') \land y \neq t)$, if $(funct\ P^-)$ exists in the ontology
- $\neg(\exists y.P(t, y) \land y \neq t') \land \neg(\exists y.P(y, t') \land y \neq t)$, if both the functionalities are present.

Other inconsistency causes

Other less intuitive cases which the algorithm takes into account:

- inconsistency deriving form irreflexive roles (that is, roles for which the ontology implies $P \sqsubseteq \neg P^-$)
- inconsistency deriving from roles such that $T \models \exists P \sqsubseteq \neg\exists P^-$

$$P(a, a)$$

- inconsistencies deriving from attribute assertion in which the type of the range is not respected

$$T = \{\rho(U) \sqsubseteq xsd : string\} \qquad A = \{U(a, 12)\}$$

- "false" inconsistencies.

Skipping false inconsistencies

An atomic concept B is called empty (unsatisfiable) in $\mathcal{O}$ if $\mathcal{O} \models B \sqsubseteq \neg B$.

An ABox assertion $B(a)$ over an empty concept B, violating a disjointness assertion $B \sqsubseteq \neg C$ together with $C(a)$, is not a real inconsistency because it does not belong to any repair.

In order to skip false inconsistencies we define the condition

$$\mathsf{ConsAtom}_B^{\mathcal{O}}(t) = \begin{cases} \textit{false} & \text{if } \mathcal{O} \models B \sqsubseteq \neg B \\ \textit{true} & \text{otherwise} \end{cases}$$

and put it in conjunction with NotDisjClash and NotFunctClash formulas.
A similar conditions holds for roles, which leads to the definition of the formula $\mathsf{ConsAtom}_P^{\mathcal{O}}(t, t')$ for every empty role P.

Skipping false inconsistencies (2)

Considering the ConsAtom conditions, the check on disjointness becomes:

$$\mathsf{NotDisjClash}_B^{\mathcal{O}}(t) = \bigwedge_{C \in DC(B,T)} \neg C(t) \wedge \mathsf{ConsAtom}_C^{\mathcal{O}}(t)$$

$$\bigwedge_{P \in DRD(B,\mathcal{O})} \neg \exists y.P(t,y) \wedge \mathsf{ConsAtom}_P^{\mathcal{O}}(t,y)$$

$$\bigwedge_{P \in DRR(B,\mathcal{O})} \neg \exists y.P(y,t) \wedge \mathsf{ConsAtom}_P^{\mathcal{O}}(y,t)$$

and the check on functionalities (with both $(funct\ P)$ and $(funct\ P^-)$) becomes:

$$\mathsf{NotFunctClash}_P^{\mathcal{O}}(t,t') = \neg(\exists y.P(t,y) \wedge y \neq t' \wedge \mathsf{ConsAtom}_P^{\mathcal{O}}(t,y)) \wedge$$

$$\neg(\exists y.P(y,t') \wedge y \neq t \wedge \mathsf{ConsAtom}_P^{\mathcal{O}}(y,t))$$

Put it all together: $\mathsf{IncRewriting}_{IR}(q, \mathcal{O})$

Let q be a CQ of the form

$$\exists x_1, \ldots, x_k. \bigwedge_{i=1}^{n} B_i(t_i^1) \wedge \bigwedge_{i=1}^{m} P_i(t_i^2, t_i^3)$$

For every concept B_i and every role P_i appearing in q we define the following conditions:

$$\mathsf{NotClash}_B^{\mathcal{O}}(t) = \mathsf{NotDisjClash}_B^{\mathcal{O}}(t)$$

$$\mathsf{NotClash}_P^{\mathcal{O}}(t, t') = \mathsf{NotDisjClash}_P^{\mathcal{O}}(t) \wedge \mathsf{NotFunctClash}_P^{\mathcal{O}}(t, t')$$

and use them to build the rewriting:

$$\exists x_1, \ldots, x_k. \bigwedge_{i=1}^{n} B_i(t_i^1) \wedge \mathsf{ConsAtom}_{B_i}^{\mathcal{O}}(t_i^1) \wedge \mathsf{NotClash}_{B_i}^{\mathcal{O}}(t_i^1) \wedge$$
$$\bigwedge_{i=1}^{m} P_i(t_i^2, t_i^3) \wedge \mathsf{ConsAtom}_{P_i}^{\mathcal{O}}(t_i^2, t_i^3) \wedge \mathsf{NotClash}_{P_i}^{\mathcal{O}}(t_i^2, t_i^3)$$

Example

Let us consider the CQ

$$q = \exists x.\mathsf{RedWine}(x)$$

We have that $\mathsf{IncRewriting}_{IR}(q, \mathcal{O})$ is

$$\exists x.\mathsf{RedWine}(x) \land \neg\mathsf{WhiteWine}(x) \land \neg\mathsf{Beer}(x) \land \neg\mathsf{Winery}(x) \land$$

$$\neg(\exists y.\mathsf{producedBy}(y, x) \land x \neq y)$$

Notice that $\mathsf{ConsAtom}^{\mathcal{O}}_{\mathsf{RedWine}} = true$ because RedWine is not an empty concept.

Contribution

Theorem

Let $\mathcal{O}$ be a *DL-Lite*$_{\mathcal{A}}$ ontology, and let Q be a UCQ, deciding whether $\mathcal{O}$ entails Q under IAR semantics is in AC^0 in data complexity.

The above result can be extended to *DL-Lite*$_{\mathcal{A},id}$.

problem	R-semantics	IR-semantics
instance checking	coNP-complete	in AC$_0$
UCQ answering	coNP-complete	in AC$_0$

Outline

MASTRO

- We have designed **MASTRO**, a $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$-based system for OBDM, and we are experimenting the system in real world settings.
- Is "first-order rewritability" a real limit that cannot be surpassed by data-intensive ontologies? $\rightsquigarrow$ real issue (open research problem).
- Our opinion:
  - FOL rewritability = reuse of relational database technology for query processing
  - more expressive ontology/query languages necessarily require support for (at least linear) recursion
  - currently, there is no available technology for recursive queries (notwithstanding with negation interpreted under the stable model semantics) that is comparable to SQL technology
  - more research is needed!
- Many research challenges remain (i.e., updates, data quality, service and process management, etc.)

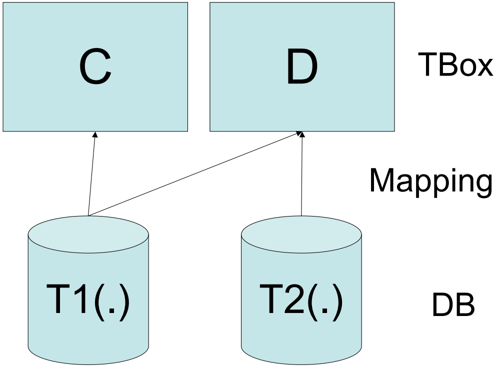
Ongoing work on OBDM

- *Ontology-based data access (OBDA)*
- *Ontology-based data integration (OBDI)*
- *Ontology-based privacy-aware data access (OBDP)*
- *Ontology-based data quality (OBDQ)*
- *Ontology-based data and service governance (OBDG)*
- *Ontology-based data restructuring (OBDR)*
- *Ontology-based data update (OBDU)*
- *Ontology-based service management (OBDS)*
- *Ontology-based data coordination (OBDC)*

Updates and erasures: challenges

- Which is a reasonable semantics for updates expressed over an ontology?
- How to “push” updates espressed over the ontology to updates over the sources?

Updates and erasures: how to push updates

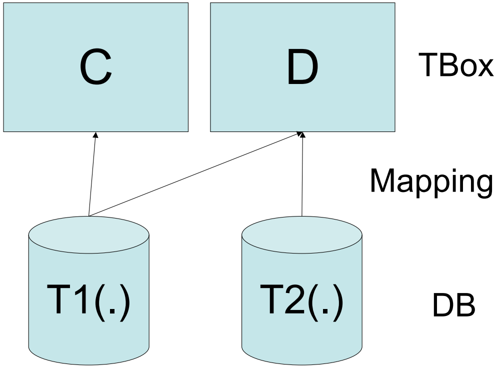


Suppose $C(a), D(a)$ are not logically implied by $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$.

Update $\{\ C(a)\ \}$:

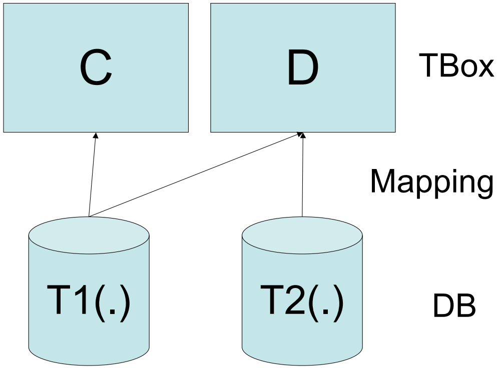
not realizable

Updates and erasures: how to push updates



Suppose $C(a), D(a)$ are not logically implied by $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$.
Update $\{\ C(a), D(a)\ \}$:
realizable by inserting $T1(a)$ in DB, or by inserting $T1(a), T2(a)$ in DB

Updates and erasures: how to push updates



Suppose $C(a), D(a)$ are logically implied by $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$.
Erase $\{\ C(a)\ \}$:
realizable by removing $T1(a)$ and inserting $T2(a)$ in DB

Acknowledgements

People involved in this work:

- Sapienza Università di Roma
  - Giuseppe De Giacomo
  - Floriana Di Pinto
  - Domenico Lembo
  - Maurizio Lenzerini
  - Antonella Poggi
  - Riccardo Rosati
  - Marco Ruzzi
  - Domenico Fabio Savo
- Libera Università di Bolzano
  - Diego Calvanese
  - Mariano Rodriguez Muro
- Many students