# Ontology-based Data Management

*Maurizio Lenzerini*

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza Università di Roma

Part III

*3rd International Winter School on Big Data*
*Bari, Italy, February 13-17, 2017*

# The course

- Part I
  - Ontology-based data management: The framework
  - Queries in OBDM
  - The nature of query answering in OBDM
- Part II
  - Ontology languages
  - Modeling the domain through the ontology
  - Modeling the mapping with the data sources
- Part III
  - Algorithms for query answering
  - Beyond classical first-order queries

# Outline

1. Algorithms for query answering
2. Beyond classical first-order queries

# *DL-Lite*$_{A,id}$

*DL-Lite*$_{A,id}$ is the most expressive logic in the *DL-Lite* family

Expressions in *DL-Lite*$_{A,id}$:

$$\begin{array}{lll} B \longrightarrow A \mid \exists Q \mid \delta(U) & E \longrightarrow \rho(U) & C \longrightarrow B \mid \neg B \\ Q \longrightarrow P \mid P^- & V \longrightarrow U \mid \neg U & R \longrightarrow Q \mid \neg Q \\ T \longrightarrow \top_D \mid T_1 \mid \cdots \mid T_n & & \end{array}$$

Assertions in *DL-Lite*$_{A,id}$:

| | | | |
|---|---|---|---|
| $B \sqsubseteq C$ | (*concept inclusion*) | $E \sqsubseteq T$ | (*value-domain inclusion*) |
| $Q \sqsubseteq R$ | (*role inclusion*) | $U \sqsubseteq V$ | (*attribute inclusion*) |
| $(id\ B\ \pi_1, ..., \pi_n)$ | (*identification assertions*) | (**funct** $Q$) | (*role functionality*) |
| (**funct** $U$) | (*attribute functionality*) | | |

In identification and functional assertions, roles and attributes cannot specialized, and each $\pi_i$ denotes a *path* (with at least one path with length 1), which is an expression built according to the following syntax rule:

$$\pi \longrightarrow S \mid B? \mid \pi_1 \circ \pi_2$$

# Semantics of *DL-Lite*$_{A,id}$

| Construct | Syntax | Example | Semantics |
|---|---|---|---|
| atomic conc. | $A$ | Doctor | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| exist. restr. | $\exists Q$ | $\exists$child$^-$ | $\{d \mid \exists e.\,(d,e) \in Q^{\mathcal{I}}\}$ |
| at. conc. neg. | $\neg A$ | $\neg$Doctor | $\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$ |
| conc. neg. | $\neg \exists Q$ | $\neg\exists$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 | $\neg Q$ | $\neg$manages | $(\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus Q^{\mathcal{I}}$ |
| conc. incl. | $B \sqsubseteq C$ | Father $\sqsubseteq$ $\exists$child | $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ |
| role incl. | $Q \sqsubseteq R$ | hasFather $\sqsubseteq$ child$^-$ | $Q^{\mathcal{I}} \subseteq R^{\mathcal{I}}$ |
| funct. asser. | (**funct** $Q$) | (**funct** succ) | $\forall d,e,e'.(d,e) \in Q^{\mathcal{I}} \wedge (d,e') \in Q^{\mathcal{I}} \rightarrow e = e'$ |
| 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*$_{A,id}$ (as all DLs of the *DL-Lite* family) adopts the Unique Name Assumption (UNA), i.e., different individuals denote different objects.

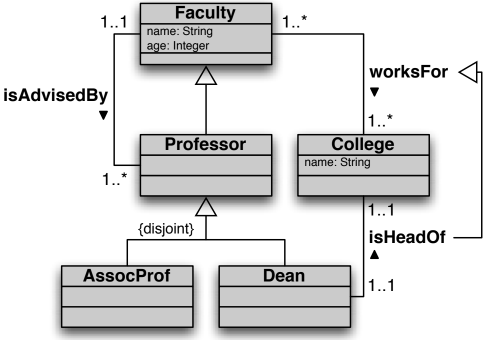
# Capturing basic ontology constructs in *DL-Lite*$_{A,id}$

| | |
|---|---|
| ISA between classes | $A_1 \sqsubseteq A_2$ |
| Disjointness between classes | $A_1 \sqsubseteq \neg A_2$ |
| Domain and range of properties | $\exists P \sqsubseteq A_1$  $\exists P^- \sqsubseteq A_2$ |
| Mandatory participation *(min card = 1)* | $A_1 \sqsubseteq \exists P$  $A_2 \sqsubseteq \exists P^-$ |
| Functionality of relations *(max card = 1)* | $(\textbf{funct } P)$  $(\textbf{funct } P^-)$ |
| ISA between properties | $Q_1 \sqsubseteq Q_2$ |
| Disjointness between properties | $Q_1 \sqsubseteq \neg Q_2$ |

*Note 1:* *DL-Lite*$_{A,id}$ cannot capture completeness of a hierarchy. This would require disjunction (i.e., OR).

*Note 2:* *DL-Lite*$_{A,id}$ can be extended to capture also min cardinality constraints ($A \sqsubseteq \leq n\ Q$), max cardinality constraints ($A \sqsubseteq \geq n\ Q$) [Artale et al, JAIR 2009], $n$-ary relations, and denial assertions (not considered here for simplicity).

# Example of $DL\text{-}Lite_{A,id}$ ontology




$$
\begin{array}{rcl}
\text{Professor} & \sqsubseteq & \text{Faculty} \\
\text{AssocProf} & \sqsubseteq & \text{Professor} \\
\text{Dean} & \sqsubseteq & \text{Professor} \\
\text{AssocProf} & \sqsubseteq & \neg\text{Dean} \\
\\
\text{Faculty} & \sqsubseteq & \exists \mathit{age} \\
\exists \mathit{age}^- & \sqsubseteq & \texttt{xsd:integer} \\
& (\textbf{funct } \mathit{age}) & \\
\\
\exists \text{worksFor} & \sqsubseteq & \text{Faculty} \\
\exists \text{worksFor}^- & \sqsubseteq & \text{College} \\
\text{Faculty} & \sqsubseteq & \exists \text{worksFor} \\
\text{College} & \sqsubseteq & \exists \text{worksFor}^- \\
\\
\exists \text{isHeadOf} & \sqsubseteq & \text{Dean} \\
\exists \text{isHeadOf}^- & \sqsubseteq & \text{College} \\
\text{Dean} & \sqsubseteq & \exists \text{isHeadOf} \\
\text{College} & \sqsubseteq & \exists \text{isHeadOf}^- \\
\text{isHeadOf} & \sqsubseteq & \text{worksFor} \\
& (\textbf{funct } \text{isHeadOf}) & \\
& (\textbf{funct } \text{isHeadOf}^-) & \\
& \vdots &
\end{array}
$$

# Satisfiability and query answering in $DL\text{-}Lite_{A,id}$

Possible approaches to satisfiability checking and query answering:

- the chase (used in database theory for reasoning about data dependencies [Maier 1983], and in data exchange for computing universal solutions [Fagin et al 2003], see [Greco et al 2012])
- resolution-based methods
- ...

None of the existing approaches directly works for our purpose.

$\rightsquigarrow$ So, we designed our own algorithm, called *PerfectRef*, implemented in our OBDM tool, MASTRO

**Remark**

We call

- positive inclusions (PIs) or positive axioms assertions of the form
$$B_1 \sqsubseteq B_2, \quad Q_1 \sqsubseteq Q_2$$
- negative axioms the other assertions, i.e.,
  - negative inclusions (NIs) assertions of the form
$$B_1 \sqsubseteq \neg B_2, \quad Q_1 \sqsubseteq \neg Q_2$$
  - identification assertions, i.e., assertions of the form
$$(id\ B\ \pi_1, ..., \pi_n), (\textbf{funct}\ Q), (\textbf{funct}\ U)$$

### Theorem

*Let* $\mathcal{O} = \mathcal{O}_{\mathrm{PI}} \cup \mathcal{O}_{\mathrm{NI}} \cup \mathcal{O}_{id}$ *be a TBox s.t.*

- $\mathcal{O}_{\mathrm{PI}}$ *is a set of PIs,*
- $\mathcal{O}_{\mathrm{NI}}$ *is a set of NIs,*
- $\mathcal{O}_{id}$ *is a set of identification assertions.*

*Then*

- *There is a boolean query* $q_v$ *such that* $\langle\mathcal{O}, \mathcal{S}, \mathcal{M}\rangle$ *is satisfiable if and only if* $\langle\mathcal{O}_{\mathrm{PI}}, \mathcal{S}, \mathcal{M}\rangle \not\models q_v$.
- *For each* $\mathcal{S}$ *such that* $\langle\mathcal{O}, \mathcal{S}, \mathcal{M}\rangle$ *is satisfiable, and for each UCQ* $q$*, we have that*
$$cert(q, \langle\mathcal{O}, \mathcal{S}, \mathcal{M}\rangle) = cert(q, \langle\mathcal{O}_{\mathrm{PI}}, \mathcal{S}, \mathcal{M}\rangle).$$

To the aim of answering queries, from now on we assume that $\mathcal{O}$ contains only PIs. Also, we denote by $\mathcal{A}$ the ABox $\mathcal{M}(\mathcal{S})$.

- There is a model $can(\langle\mathcal{O},\mathcal{A}\rangle)$ of $\langle\mathcal{O},\mathcal{A}\rangle$ such that, for each model $M'$ of $\langle\mathcal{O},\mathcal{A}\rangle$, there is a homomorphism from $can(\langle\mathcal{O},\mathcal{A}\rangle)$ to $M'$, i.e., a function $h$ from the domain of $can(\langle\mathcal{O},\mathcal{A}\rangle)$ to the domain of $M'$ such that $a^{can(\langle\mathcal{O},\mathcal{A}\rangle)} \in C^{can(\langle\mathcal{O},\mathcal{A}\rangle)}$ implies $h(a)^{M'} \in C^{M'}$, and $a(^{can(\langle\mathcal{O},\mathcal{A}\rangle)}, b^{can(\langle\mathcal{O},\mathcal{A}\rangle)}) \in R^{can(\langle\mathcal{O},\mathcal{A}\rangle)}$ implies $(h(a)^{M'}, h(b)^{M'}) \in R^{M'}$.
- In principle, $can(\langle\mathcal{O},\mathcal{A}\rangle)$ can be constructed by means of a procedure, called *chase*, that starting from $\mathcal{A}$, "apply" the various PIs, by adding the facts sanctioned by the PIs. In general, $can(\langle\mathcal{O},\mathcal{A}\rangle)$ is infinite.
- Thus, instead of trying to build $can(\langle\mathcal{O},\mathcal{A}\rangle)$, we adopt a top-down approach: by using $\mathcal{O}$, we rewrite a CQ $q$ into a UCQ $q'$ in such a way that evaluating $q'$ over $\mathcal{A}$ is equivalent to evaluate $q$ over $can(\langle\mathcal{O},\mathcal{A}\rangle)$ (keeping only answers consituted by constants).

Correctness of this procedure shows FOL-rewritability of query answering in *DL-Lite*$_{A,id}$.

# Query answering in $DL\text{-}Lite_{A,id}$: Query rewriting (cont'd)

Intuition: Use the PIs as basic rewriting rules

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

$$\mathsf{AssocProfessor} \sqsubseteq \mathsf{Professor}$$

as a logic rule: $\forall z\ \mathsf{AssocProfessor}(z) \rightarrow \mathsf{Professor}(z)$

Basic rewriting step:

when the atom unifies with the **head** of the rule (with mgu $\sigma$).

substitute the atom with the **body** of the rule (to which $\sigma$ is applied).

Towards the computation of the perfect rewriting, we add to the input query above the following query ($\sigma = \{z/x\}$)

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

We say that the PI AssocProfessor $\sqsubseteq$ Professor applies to the atom Professor$(x)$.

Consider now the query

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

$$\text{Professor} \sqsubseteq \exists\text{teaches}$$

$$\text{as a logic rule:} \quad \forall z\ \text{Professor}(z) \rightarrow \exists y\ \text{teaches}(z, y)$$

We add to the reformulation the query ($\sigma = \{z_1/x, z_2/y\}$)

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

Analogously, for the query

$$\mathsf{q}(x) \leftarrow \mathsf{teaches}(x, \mathtt{databases})$$

$$\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$$
$$\text{as a logic rule:} \quad \forall z\ \mathsf{Professor}(z) \rightarrow \exists y \mathsf{teaches}(z, y)$$

$\mathsf{teaches}(x, \mathtt{databases})$ does not unify with $\mathsf{teaches}(z, y)$, since the existentially quantified variable $z_2$ in the head of the rule does not unify with the constant $\mathtt{databases}$.

In this case the PI does not apply to the atom $\mathsf{teaches}(x, \mathtt{databases})$.

The same holds for the following query, where $y$ is distinguished

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

# Query answering in *DL-Lite*$_{A,id}$: Query rewriting (cont'd)

Conversely, for the following query with join variables

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

$$\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches}$$

$$\text{as a logic rule:} \quad \forall z \ \mathsf{Professor}(z) \rightarrow \exists y \ \mathsf{teaches}(z, y)$$

The PI above does not apply to the atom $\mathsf{teaches}(x, y)$.

Conversely, the PI

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

$$\text{as a logic rule:} \quad \forall z_1 \forall z_2 \ \mathsf{teaches}(z_1, z_2) \rightarrow \mathsf{Course}(z_2)$$

applies to the atom $\mathsf{Course}(y)$.

We add to the perfect rewriting the query ($\sigma = \{z_2/y\}$)

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

# Query answering in *DL-Lite*$_{A,id}$: Query rewriting (cont'd)

We now have the query

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

The PI $\quad$ Professor $\sqsubseteq \exists$teaches

as a logic rule: $\forall z \; \text{Professor}(z) \rightarrow \exists y \text{teaches}(z, y)$

does not apply to teaches$(x, y)$ nor teaches$(z, y)$, since $y$ is a join variable. However, we can transform the above query by unifying the atoms teaches$(x, y)$, teaches$(z, y)$. This rewriting step is called reduce, and produces the following query

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

We can now apply the PI above ($\sigma\{z_1/x, z_2/y\}$), and add to the reformulation the query

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

# Answering by rewriting in *DL-Lite*$_{A,id}$: The algorithm

1. Rewrite the CQ $q$ into a UCQs: apply to $q$ in all possible ways the PIs in the TBox $\mathcal{O}$.
2. Treat CQs as sets, so as to block infinite loops
3. This corresponds to exploiting ISAs, role typings, and mandatory participations to obtain new queries that could contribute to the answer.
4. Unifying atoms can make applicable rules that could not be applied otherwise.

**Algorithm** *PerfectRef*$(q, \mathcal{O}_P)$
**Input:** conjunctive query $q$, set of *DL-Lite*$_{A,id}$ PIs $\mathcal{O}_P$
**Output:** union of conjunctive queries $PR$
$PR := \{q\}$;
**repeat**
  $PR' := PR$;
  **for each** $q \in PR'$ **do**
  (a) **for each** $g$ in $q$ **do**
    **for each** PI $I$ in $\mathcal{O}_P$ **do**
      **if** $I$ is applicable to $g$
      **then** $PR := PR \cup \{\, q[g/(g, I)] \,\}$
  (b) **for each** $g_1, g_2$ in $q$ **do**
    **if** $g_1$ and $g_2$ unify
    **then** $PR := PR \cup \{\tau(\textit{reduce}(q, g_1, g_2))\}$;
**until** $PR' = PR$;
**return** $PR$

# Answering by rewriting in *DL-Lite*$_{A,id}$: The algorithm

**Theorem (Calvanese et al, JAR 2007)**

*The query resulting from the above process is a UCQ, and is the perfect rewriting $r_{q,\mathcal{O}}$, i.e., evaluating $r_{q,\mathcal{O}}$ over $\mathcal{M}(\mathcal{S})$ computes the certain answers to $q$ wrt $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$.*

Note that the same algorithm can be used to check satisfiability of $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$, where answers to query $q_v$ correspond to violations of negative inclusions.

**TBox:** $\text{Professor} \sqsubseteq \exists\text{teaches}$
$\exists\text{teaches}^- \sqsubseteq \text{Course}$

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

**Perfect Rewriting:**
$\text{q}(x) \leftarrow \text{teaches}(x, y), \text{Course}(y)$
$\text{q}(x) \leftarrow \text{teaches}(x, y), \text{teaches}(z, y)$
$\text{q}(x) \leftarrow \text{teaches}(x, z)$
$\text{q}(x) \leftarrow \text{Professor}(x)$

$\mathcal{M}(\mathcal{S})$**:** $\text{teaches}(\texttt{John}, \texttt{databases})$
$\text{Professor}(\texttt{Mary})$

It is easy to see that the evaluation of $r_{q,\mathcal{O}}$ over $\mathcal{M}(\mathcal{S})$ in this case produces the set $\{\texttt{John}, \texttt{Mary}\}$.

# Complexity

$n$ : query size

$m$ : number of predicate symbols in $\mathcal{O}$ or query $q$

The number of distinct conjunctive queries generated by the algorithm is less than or equal to $(m \times (n+1)^2)^n$, which corresponds to the maximum number of executions of the repeat-until cycle of the algorithm.

Query answering for CQs and UCQs is:

- PTime in the size of TBox.
- $\mathrm{AC}^0$ in the size of the $\mathcal{M}(\mathcal{S})$.
- Exponential in the size of the query.

**Can we go beyond $\textit{DL-Lite}_{A,id}$ and remain in $\mathrm{AC}^0$?**

By adding essentially any other DL construct (without limitations) we lose these computational properties.

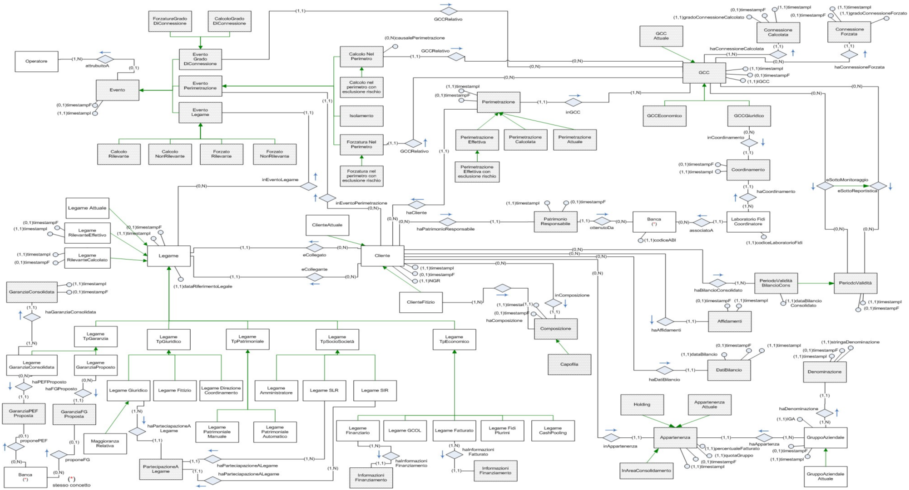
# Beyond *DL-Lite*$_{A,id}$: results on data complexity

| | 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 |

- *DL-Lite*$_{\mathcal{A},id}$ is the most expressive DL of the *DL-Lite* family
- 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.

# Complexity matters

A portion of an ontology for the Italian Public Debt:



# Sources of complexity

For realistic ontologies, systems based on *PerfectRef* works for queries with at most 7-8 atoms.

Two sources of complexity wrt query:

- conjunctive query evaluation is NP-complete – complexity comes from the need of matching the query and the data
  $\rightsquigarrow$ unavoidable!
- the rewritten query has exponential size wrt the original query – complexity comes from the need of "expanding" the query w.r.t. the ontology
  $\rightsquigarrow$ avoidable?

**Example**

TBox $\mathcal{T}$:




$$q(x) \leftarrow A(x), P(x,y), A(y), P(y,z), A(z)$$

UCQ rewriting of $q$ w.r.t. $\mathcal{T}$ contains 729 CQs i.e., it is a UNION of 729 SPJ SQL queries

# Sources of complexity

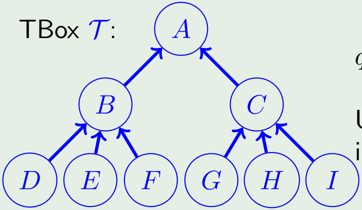
For realistic ontologies, systems based on *PerfectRef* works for queries with at most 7-8 atoms.

Two sources of complexity wrt query:

- conjunctive query evaluation is $\mathrm{NP}$-complete – complexity comes from the need of matching the query and the data
  $\leadsto$ unavoidable!
- the rewritten query has exponential size wrt the original query – complexity comes from the need of "expanding" the query w.r.t. the ontology
  $\leadsto$ avoidable?

## Example

TBox $\mathcal{T}$:




$$q(x) \leftarrow A(x), P(x,y), A(y), P(y,z), A(z)$$

UCQ rewriting of $q$ w.r.t. $\mathcal{T}$ contains 729 CQs i.e., it is a UNION of 729 SPJ SQL queries

# Avoiding exponential blow-up: first attempt

Idea: avoid rewriting whatsoever!

Unfortunately, this idea does not work:

**Theorem (Calvanese et al, JAR 2007)**

*Given $\mathcal{M}(\mathcal{S})$, there is no finite database $B$ such that for every query $q$,* $cert(q, \langle\mathcal{O}, \mathcal{S}, \mathcal{M}\rangle) = q^B$

# Avoiding exponential blow-up: second attempt

Idea: try to apply the chase only partially, so as to obtain a finite database (possibly with variables) such that only "small rewritings" of conjunctive queries are needed [Kontchakov et al, KR 2010]

Two problems:

- it works only without role inclusions
- (partial) materialization not always appropriate in OBDM

# Avoiding exponential blow-up: third attempt

REQUIEM [Pérez-Urbina et al, 2009] is a query rewriting algorithm based on resolution for $\mathcal{ELHIO}$ that rewrites in UCQ for *DL-Lite*$_{A,id}$, and limits the number of "reduce" steps wrt *PerfectRef*. Still the size of the rewritten query is exponential in the worst case.

Presto [Rosati, KR 2010] is the current rewriting algorithm used in MASTRO, based on the following ideas for improving the performance of *PerfectRef*:

- centering the rewriting around the query variables rather than the query atoms – this allows for
  - collapsing sequences of rewriting steps into single steps and
  - dramatically pruning the solution space of the algorithm
- going beyond the disjunctive normal form (UCQ) of the rewritten query – nonrecursive datalog queries (UCQ with an additional unfolding step)

# Avoiding exponential blow-up: third attempt

Presto replaces the "atom-rewrite" and "reduce" rules of *PerfectRef* with a rule (based on MGS) that eliminates existential join variables, where elimination of an existential join variable means that the variable turns into a non-join existential variable (through unification steps)

This makes the rewriting produced by Presto exponential with respect to the number of eliminable existential join variables in the query (notice: in practice, often the majority of existential join variables in a CQ are not eliminable)

$\leadsto$ dramatic reduction of the size of the query generated. Presto can handle queries with about 30 atoms.

# Avoiding exponential blow-up: other attempts

- [Calvanese et al, KR 2012] shows how to exploit knowledge about inclusion dependencies on $\mathcal{M}(\mathcal{S})$ in order to produce more compact rewritings (See ONTOP, an OBDM tool developed at the Free University of Bolzano)
- Prexto [Rosati, ESWC 2012] applies this idea to Presto.

In the worst case, Prexto still rewrites into a union of conjunctive queries of exponential size wrt the original query. Is it avoidable?

**Theorem (Kikot et al, DL 2011)**

*Cheking whether $\vec{t} \in cert(q, \langle \mathcal{O}, \mathcal{M}, \mathcal{S} \rangle)$ is NP-complete in combined complexity, even if $\mathcal{O}$ is in DL-Lite$_{\mathcal{R}}$, and $\mathcal{M}(\mathcal{S})$ is constituted by just one atomic assertion $A(c)$.*

Since answering a FOL query over a database $A(c)$ can be done in linear time, it follows that no algorithm can construct a FOL rewriting in polynomial time, unless P = NP.

# Avoiding exponential blow-up: other attempts

- Polynomial FOL rewritings exist in the case of *DL-Lite*$_{core}$ (*DL-Lite*$_{\mathcal{R}}$ without role inclusions) [Kikot et al, DL 2011].
- Polynomial rewritings exist if we allow rewriting to be expressed in nonrecursive Datalog queries using additional symbols [Gottlob and Schwentick, DL 2011].
- Many recent papers carry out deep investigations on the problem from various points of view [Kikot et al, AAAI 2011], [Gottlob et al, ICDE 2011], [Kikot et al, KR 2012], [Kikot et al, LICS 2014], [Bienvenu et al DL 2015], [Bienvenu et al DL 2016] etc.

# Exploiting (virtual) ABox dependencies

As in databases, we can exploit dependencies on the data and on the ABox to optimize query processing.

ABox dependencies express conditions on the data in the (virtual) ABox.

- Syntax: $C_1 \sqsubseteq_{\mathcal{A}} C_2$ $\qquad$ $R_1 \sqsubseteq_{\mathcal{A}} R_2$
- Meaning: constrain the assertions present in the ABox
  - $\mathcal{A} \vdash A_1 \sqsubseteq_{\mathcal{A}} A_2$ iff $A_1(d) \in \mathcal{A}$ implies $A_2(d) \in \mathcal{A}$
  - $\mathcal{A} \vdash A \sqsubseteq_{\mathcal{A}} \exists P$ iff $A(d) \in \mathcal{A}$ implies $P(d, d') \in \mathcal{A}$ for some $d'$
  - $\mathcal{A} \vdash P_1 \sqsubseteq_{\mathcal{A}} P_2$ iff $P_1(d, d') \in \mathcal{A}$ implies $P_2(d, d') \in \mathcal{A}$
  - $\cdots$

*Note:* ABox dependencies are fundamentally different from TBox assertions. They constrain the syntactic structure of the ontology (the ABox itself), and not the models.

# Exploiting ABox/data dependencies in OBDM

In an OBDM system, ABox/data dependencies have an impact that spans all system components:

- Dependencies on the sources $\mathcal{S}$ induce via the mapping $\mathcal{M}$ dependencies on the virtual ABox $\mathcal{M}(\mathcal{S})$.
- The mapping itself in general induces additional dependencies on $\mathcal{M}(\mathcal{S})$ (that do not directly depend on $\mathcal{S}$).
- Dependencies on $\mathcal{M}(\mathcal{S})$ interact with the TBox $\mathcal{T}$, and such interaction can be exploited for optimization.

# Eliminating redundant TBox assertions

TBox optimization is based on a characterization of assertions in a TBox $\mathcal{T}$ that are redundant wrt a set $\Sigma$ of ABox dependencies.

### Example (Direct redundancy)

Let $\mathcal{T}$ be:

$$\begin{tikzcd} & \text{Human} \\ \text{Person} \arrow[ur] & \\ & \exists\text{hasFather} \arrow[ul, dotted] \end{tikzcd}$$

Let $\Sigma$ be:

$$\begin{tikzcd} & \text{Human} \\ \text{Person} \arrow[ur, twoheadrightarrow] & \\ & \exists\text{hasFather} \arrow[ul, dotted, twoheadrightarrow] \end{tikzcd}$$

Note: $\Sigma$ enforces e.g., that

$$\text{hasFather}(\text{luisa}, \text{franz}) \in \mathcal{A} \quad \text{implies} \quad \text{Human}(\text{luisa}) \in \mathcal{A}.$$

Then Person $\sqsubseteq$ Human is redundant in $\mathcal{T}$.

The overall characterization of redundant TBox assertions is more involved.

# Eliminating redundant TBox assertions

TBox optimization is based on a characterization of assertions in a TBox $\mathcal{T}$ that are redundant wrt a set $\Sigma$ of ABox dependencies.

**Example (Direct redundancy)**

Let $\mathcal{T}$ be:

$$\begin{tikzcd} \mathsf{Human} \\ \mathsf{Person} \arrow[u] \\ \exists\mathsf{hasFather} \arrow[u, dotted] \end{tikzcd}$$

Let $\Sigma$ be:

$$\begin{tikzcd} \mathsf{Human} \\ \mathsf{Person} \arrow[u, twoheadrightarrow] \\ \exists\mathsf{hasFather} \arrow[u, dotted, twoheadrightarrow] \end{tikzcd}$$

Note: $\Sigma$ enforces e.g., that

$$\mathsf{hasFather}(\mathsf{luisa}, \mathsf{franz}) \in \mathcal{A} \quad \text{implies} \quad \mathsf{Human}(\mathsf{luisa}) \in \mathcal{A}.$$

Then $\mathsf{Person} \sqsubseteq \mathsf{Human}$ is redundant in $\mathcal{T}$.

The overall characterization of redundant TBox assertions is more involved.

# Computing an optimized TBox

Given a TBox $\mathcal{T}$ and a set $\Sigma$ of ABox dependencies:

1. Compute the deductive closure $\mathcal{T}_{cl}$ of $\mathcal{T}$ (at most quadratic in size of $\mathcal{T}$).
2. Compute the deductive closure $\Sigma_{cl}$ of $\Sigma$ (at most quadratic in size of $\Sigma$).
3. Eliminate from $\mathcal{T}_{cl}$ all TBox assertions redundant wrt $\Sigma_{cl}$, obtaining $\mathcal{T}_{opt}$.

*Notes:*

- $\mathcal{T}_{opt}$ can be computed in polynomial time in the size of $\mathcal{T}$ and $\Sigma$.
- $\mathcal{T}_{opt}$ might be much smaller than $\mathcal{T}$.

**Theorem**

*For every (virtual) ABox $\mathcal{A}$ satisfying $\Sigma$ and for every UCQ $q$, we have that*

$$cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(q, \langle \mathcal{T}_{opt}, \mathcal{A} \rangle).$$

Hence, $\mathcal{T}_{opt}$ can be used instead of $\mathcal{T}$ independently of the adopted query rewriting method (provided the ABox satisfies $\Sigma$).

# Computing an optimized TBox

Given a TBox $\mathcal{T}$ and a set $\Sigma$ of ABox dependencies:

1. Compute the deductive closure $\mathcal{T}_{cl}$ of $\mathcal{T}$ (at most quadratic in size of $\mathcal{T}$).
2. Compute the deductive closure $\Sigma_{cl}$ of $\Sigma$ (at most quadratic in size of $\Sigma$).
3. Eliminate from $\mathcal{T}_{cl}$ all TBox assertions redundant wrt $\Sigma_{cl}$, obtaining $\mathcal{T}_{opt}$.

*Notes:*

- $\mathcal{T}_{opt}$ can be computed in polynomial time in the size of $\mathcal{T}$ and $\Sigma$.
- $\mathcal{T}_{opt}$ might be much smaller than $\mathcal{T}$.

**Theorem**

*For every (virtual) ABox $\mathcal{A}$ satisfying $\Sigma$ and for every UCQ $q$, we have that*

$$cert(q, \langle \mathcal{T}, \mathcal{A} \rangle) = cert(q, \langle \mathcal{T}_{opt}, \mathcal{A} \rangle).$$

Hence, $\mathcal{T}_{opt}$ can be used instead of $\mathcal{T}$ independently of the adopted query rewriting method (provided the ABox satisfies $\Sigma$).

# Computing an optimized TBox

Given a TBox $\mathcal{T}$ and a set $\Sigma$ of ABox dependencies:

1. Compute the deductive closure $\mathcal{T}_{cl}$ of $\mathcal{T}$ (at most quadratic in size of $\mathcal{T}$).
2. Compute the deductive closure $\Sigma_{cl}$ of $\Sigma$ (at most quadratic in size of $\Sigma$).
3. Eliminate from $\mathcal{T}_{cl}$ all TBox assertions redundant wrt $\Sigma_{cl}$, obtaining $\mathcal{T}_{opt}$.

*Notes:*

- $\mathcal{T}_{opt}$ can be computed in polynomial time in the size of $\mathcal{T}$ and $\Sigma$.
- $\mathcal{T}_{opt}$ might be much smaller than $\mathcal{T}$.

**Theorem**

*For every (virtual) ABox $\mathcal{A}$ satisfying $\Sigma$ and for every UCQ $q$, we have that*

$$cert(q, \langle\mathcal{T}, \mathcal{A}\rangle) = cert(q, \langle\mathcal{T}_{opt}, \mathcal{A}\rangle).$$

Hence, $\mathcal{T}_{opt}$ can be used instead of $\mathcal{T}$ independently of the adopted query rewriting method (provided the ABox satisfies $\Sigma$).

# Deriving ABox dependencies

Derived from dependencies on the data in $\mathcal{S}$:

## Example

Suppose we have two tables in $\mathcal{S}$:

- $\mathsf{ResT}[\mathit{SSN}\colon \mathsf{String}, \ldots]$ stores data about researchers
- $\mathsf{ManT}[\mathit{SSN}\colon \mathsf{String}, \ldots]$ stores data about managers

Consider the following mapping $\mathcal{M}$:

$m_1$: `SELECT SSN FROM ResT` $\rightsquigarrow$ $\mathsf{Researcher}(\mathbf{pers}(\mathit{SSN}))$
$m_2$: `SELECT SSN FROM ManT` $\rightsquigarrow$ $\mathsf{Manager}(\mathbf{pers}(\mathit{SSN}))$

If $\mathcal{S}$ satisfies the inclusion dependency $\mathsf{ManT}[\mathit{SSN}] \subseteq \mathsf{ResT}[\mathit{SSN}]$,
then $\mathcal{M}(\mathcal{S})$ satisfies the dependency $\mathsf{Manager} \sqsubseteq_{\mathcal{M}(\mathcal{S})} \mathsf{Researcher}$.

# Deriving ABox dependencies

Derived from dependencies on the data in $\mathcal{S}$:

## Example

Suppose we have two tables in $\mathcal{S}$:

- ResT[*SSN*: String, ...] stores data about researchers
- ManT[*SSN*: String, ...] stores data about managers

Consider the following mapping $\mathcal{M}$:

$m_1$: `SELECT SSN FROM ResT` $\rightsquigarrow$ Researcher(**pers**(*SSN*))
$m_2$: `SELECT SSN FROM ManT` $\rightsquigarrow$ Manager(**pers**(*SSN*))

If $\mathcal{S}$ satisfies the inclusion dependency ManT[*SSN*] $\subseteq$ ResT[*SSN*],
then $\mathcal{M}(\mathcal{S})$ satisfies the dependency Manager $\sqsubseteq_{\mathcal{M}(\mathcal{S})}$ Researcher.

# Deriving ABox dependencies (cont'd)

Induced by the form of the data in $\mathcal{S}$ and the mapping $\mathcal{M}$:

**Example**

Suppose that in $\mathcal{S}$ we have one table: ResT[*SSN*: String, *Level*: Boolean, ...]

- Stores data about researchers (including managers).
- For managers the value of *Level* is `true`, otherwise it is `false`.

Consider the following mapping $\mathcal{M}$:

$m_1$: `SELECT SSN FROM ResT` $\rightsquigarrow$ Researcher(**pers**(*SSN*))

$m_2$: `SELECT SSN FROM ResT WHERE Level='true'` $\rightsquigarrow$ Manager(**pers**(*SSN*))

We have that $\mathcal{M}(\mathcal{S})$ satisfies the dependency Manager $\sqsubseteq_{\mathcal{M}(\mathcal{S})}$ Researcher. This holds since the lhs query of $m_2$ is contained in the lhs query of $m_1$ (in the traditional sense of query containment in DBs).

This situation corresponds to a natural way of constructing mappings, and is very common in OBDM systems.

# Deriving ABox dependencies (cont'd)

Induced by the form of the data in $\mathcal{S}$ and the mapping $\mathcal{M}$:

**Example**

Suppose that in $\mathcal{S}$ we have one table: ResT[*SSN*: String, *Level*: Boolean, ...]

- Stores data about researchers (including managers).
- For managers the value of *Level* is `true`, otherwise it is `false`.

Consider the following mapping $\mathcal{M}$:

$m_1$: `SELECT SSN FROM ResT` $\rightsquigarrow$ Researcher(**pers**(*SSN*))

$m_2$: `SELECT SSN FROM ResT WHERE Level='true'` $\rightsquigarrow$ Manager(**pers**(*SSN*))

We have that $\mathcal{M}(\mathcal{S})$ satisfies the dependency Manager $\sqsubseteq_{\mathcal{M}(\mathcal{S})}$ Researcher. This holds since the lhs query of $m_2$ is contained in the lhs query of $m_1$ (in the traditional sense of query containment in DBs).

This situation corresponds to a natural way of constructing mappings, and is very common in OBDM systems.

# Deriving ABox dependencies (cont'd)

Induced by the form of the data in $\mathcal{S}$ and the mapping $\mathcal{M}$:

**Example**

Suppose that in $\mathcal{S}$ we have one table: ResT[*SSN*: String, *Level*: Boolean, ...]

- Stores data about researchers (including managers).
- For managers the value of *Level* is `true`, otherwise it is `false`.

Consider the following mapping $\mathcal{M}$:

$m_1$: `SELECT SSN FROM ResT` $\rightsquigarrow$ Researcher(**pers**(*SSN*))

$m_2$: `SELECT SSN FROM ResT WHERE Level='true'` $\rightsquigarrow$ Manager(**pers**(*SSN*))

We have that $\mathcal{M}(\mathcal{S})$ satisfies the dependency Manager $\sqsubseteq_{\mathcal{M}(\mathcal{S})}$ Researcher. This holds since the lhs query of $m_2$ is contained in the lhs query of $m_1$ (in the traditional sense of query containment in DBs).

This situation corresponds to a natural way of constructing mappings, and is very common in OBDM systems.

# Deriving ABox dependencies (cont'd)

We can use the mapping to enforce dependencies "corresponding" to the TBox assertions:

## Example

Suppose that in $\mathcal{S}$ we have one table: ResT[*SSN*: String, *Type*: Char, ...]

- Stores data about researchers of all types.
- The value of *Type* encodes the type or researcher: 'm' for managers, 'p' for principal investigators, 'c' for coordinators, and 'r' for other researchers.

We can define a mapping $\mathcal{M}$ that induces suitable dependencies:

$m_1$: `SELECT SSN FROM ResT` $\leadsto$ Researcher(**pers**(*SSN*))

$m_2$: `SELECT SSN FROM ResT WHERE Type='p'` $\leadsto$ PrincInv(**pers**(*SSN*))

$m_3$: `SELECT SSN FROM ResT WHERE Type='c'` $\leadsto$ Coordinator(**pers**(*SSN*))

$m_4$: `SELECT SSN FROM ResT WHERE Type='m' OR Type='p' OR Type='c'` $\leadsto$ Manager(**pers**(*SSN*))

We have that $\mathcal{M}(\mathcal{S})$ satisfies e.g., the dependency PrincInv $\sqsubseteq_{\mathcal{M}(\mathcal{S})}$ Manager.

# Deriving ABox dependencies (cont'd)

We can use the mapping to enforce dependencies "corresponding" to the TBox assertions:

**Example**

Suppose that in $\mathcal{S}$ we have one table: ResT[*SSN*: String, *Type*: Char, . . . ]

- Stores data about researchers of all types.
- The value of *Type* encodes the type or researcher: 'm' for managers, 'p' for principal investigators, 'c' for coordinators, and 'r' for other researchers.

We can define a mapping $\mathcal{M}$ that induces suitable dependencies:

$m_1$: `SELECT SSN FROM ResT` $\rightsquigarrow$ Researcher(**pers**(*SSN*))

$m_2$: `SELECT SSN FROM ResT WHERE Type='p'` $\rightsquigarrow$ PrincInv(**pers**(*SSN*))

$m_3$: `SELECT SSN FROM ResT WHERE Type='c'` $\rightsquigarrow$ Coordinator(**pers**(*SSN*))

$m_4$: `SELECT SSN FROM ResT WHERE Type='m' OR Type='p' OR Type='c'` $\rightsquigarrow$ Manager(**pers**(*SSN*))

We have that $\mathcal{M}(\mathcal{S})$ satisfies e.g., the dependency PrincInv $\sqsubseteq_{\mathcal{M}(\mathcal{S})}$ Manager.

# Outline

1. Algorithms for query answering
2. Beyond classical first-order queries

# SPARQL queries over OWL 2 QL ontologies

The *DL-Lite* family includes *DL-Lite*$_{\mathcal{R}}$, that is the basis of OWL 2 QL, a profile of OWL 2. Does the *DL-Lite*$_{A,id}$ technique provide a solution to the problem of answering SPARQL queries posed to OWL 2 QL ontologies?

The answer is no, for the following main reasons:

- OWL 2 QL has some features on relations that *DL-Lite*$_{\mathcal{A},id}$ does not have (i.e., reflexive, irreflexive relations)
- In OWL 2 QL one can assert inequality between individuals, while *DL-Lite*$_{\mathcal{A},id}$ does not have inequality; SPARQL queries posed to OWL 2 QL ontologies can also use the inequality predicate, contrary to pure CQs
- In OWL 2 QL an ontology entity can belong to different categories (e.g., an individual and a class), while *DL-Lite*$_{\mathcal{A},id}$ is a pure FOL language (metamodeling)
- Variables appearing in both predicate and argument positions (metaquerying) are allowed in SPARQL queries, contrary to pure CQs (or, the Direct Semantics Entailment Regime of SPARQL)

# Metamodeling and metaquerying

Up to now, we have assumed that the TBox and the ABox were first-order, with a strict separation between individuals and classes/relations.

- Metamodeling: specifying
  - metaclasses (classes whose instances can be themselves classes), and
  - metaproperties (relationships between metaclasses)

- Metaquerying: expressing queries with
  - variables both in predicate and object position, and
  - TBox atoms

# Enriching the mapping languages: mapping intensional knowledge

Source $\mathcal{S}$:

`T-CarTypes`

| Code | Name |
|---|---|
| T1 | Coupé |
| T2 | SUV |
| T3 | Sedan |
| T4 | Estate |

`T-Cars`

| CarCode | CarType | EngineSize | BreakPower | Color | TopSpeed |
|---|---|---|---|---|---|
| AB111 | T1 | 2000 | 200 | Silver | 260 |
| AF333 | T2 | 3000 | 300 | Black | 200 |
| BR444 | T2 | 4000 | 400 | Grey | 220 |
| AC222 | T4 | 2000 | 125 | Dark Blue | 180 |
| BN555 | T3 | 1000 | 75 | Light Blue | 180 |
| BP666 | T1 | 3000 | 600 | Red | 240 |

# Example

Ontology $\mathcal{O}$: $\texttt{Car} \sqsubseteq \texttt{Vehicle}$

Source $\mathcal{S}$:

T-CarTypes

| Code | Name |
|---|---|
| T1 | Coupé |
| T2 | SUV |
| T3 | Sedan |
| T4 | Estate |

T-Cars

| CarCode | CarType | EngineSize | BreakPower | Color | TopSpeed |
|---|---|---|---|---|---|
| AB111 | T1 | 2000 | 200 | Silver | 260 |
| AF333 | T2 | 3000 | 300 | Black | 200 |
| BR444 | T2 | 4000 | 400 | Grey | 220 |
| AC222 | T4 | 2000 | 125 | Dark Blue | 180 |
| BN555 | T3 | 1000 | 75 | Light Blue | 180 |
| BP666 | T1 | 3000 | 600 | Red | 240 |

Mapping $\mathcal{M}$:

- $\{(y) \mid \texttt{T-CarTypes}(x,y)\} \rightsquigarrow \texttt{TypeOfCar}(y), y \sqsubseteq \texttt{Car}$
- $\{(x,v,z) \mid \texttt{T-Cars}(x,y,t,u,v,q) \land \texttt{T-CarTypes}(y,z)\} \rightsquigarrow z(x)$
- $\{(x,y) \mid \texttt{T-CarTypes}(z_1,x) \land \texttt{T-CarTypes}(z_2,y) \land x \neq y\} \rightsquigarrow x \sqsubseteq \neg y$

The ontology $\mathcal{O}$ is enriched through $\mathcal{M}$ and $\mathcal{S}$.

# Metamodeling and metaquerying

With metaclasses and metaproperties in the ontology, metaqueries become natural, e.g.:

**Example**

Interesting queries that can be posed to $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$ exploit the higher-order nature of the system:

- Return all the instances of *Car*, each one with its own type:
  $q(x, y) \leftarrow y(x), \mathtt{Car}(x), \mathtt{TypeOfCar}(y)$
- Return all the concepts which *AB111* is an instance of:
  $q(x) \leftarrow x(\mathtt{AB111})$

# Example of metaquerying

Consider querying an ontology about the "pizza" domain, including

Classes: `margherita, ortolana, vegeterian`
Object properties: `ate`, `liked`, `dislike`

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y)\ \}$

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y), \texttt{dislike}(x, \texttt{margherita})\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z)\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}, z \neq \texttt{ortolana}\ \}$

# Example of metaquerying

Consider querying an ontology about the "pizza" domain, including

Classes: `margherita, ortolana, vegeterian`
Object properties: `ate`, `liked`, `dislike`

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y)\ \}$

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y), \texttt{dislike}(x, \texttt{margherita})\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z)\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian},$
$z \neq \texttt{ortolana}\ \}$

# Example of metaquerying

Consider querying an ontology about the "pizza" domain, including

Classes: margherita, ortolana, vegeterian
Object properties: ate, liked, dislike

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y)\ \}$

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y), \texttt{dislike}(x, \texttt{margherita})\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z)\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}, z \neq \texttt{ortolana}\ \}$

# Example of metaquerying

Consider querying an ontology about the "pizza" domain, including

Classes: `margherita, ortolana, vegeterian`
Object properties: `ate, liked, dislike`

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y)\ \}$

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y), \texttt{dislike}(x, \texttt{margherita})\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z)\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}, z \neq \texttt{ortolana}\ \}$

# Example of metaquerying

Consider querying an ontology about the "pizza" domain, including

Classes: `margherita, ortolana, vegeterian`
Object properties: `ate`, `liked`, `dislike`

$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y)\ \}$

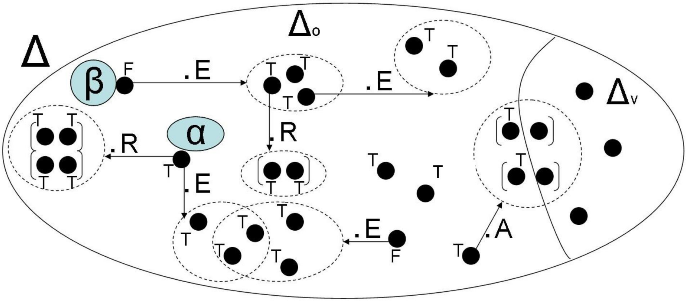
$\{\ (x) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), \texttt{margherita}(y), \texttt{dislike}(x, \texttt{margherita})\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z)\ \}$

$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}\ \}$

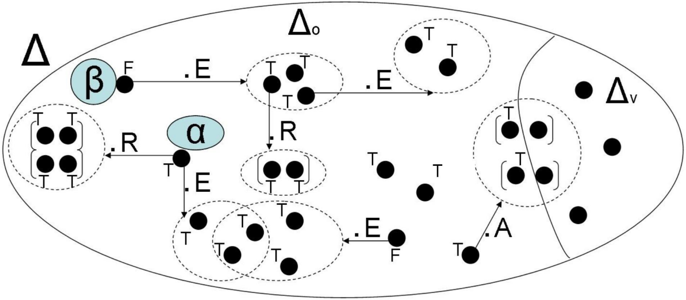
$\{\ (x,z) \mid \texttt{ate}(x,y), \texttt{liked}(x,y), z(y), \texttt{dislike}(x,z), z \sqsubseteq \texttt{vegeterian}, z \neq \texttt{ortolana}\ \}$

- Interpretation structure $\Sigma$:




- Interpretation function $\mathcal{I}_0$, assigning semantics (i.e., a domain object) to expressions.

Note: a domain object is not forced to be an individual (e.g., see $\beta$), or to have a concept or relation extension




Suitable conditions state when an axiom is satisfied by an interpretation $\langle \Sigma, \mathcal{I}_0 \rangle$:

- for concepts: $\langle \Sigma, \mathcal{I}_0 \rangle \models e_1 \sqsubseteq e_2$ if $(e_1^{\mathcal{I}_0})^E \subseteq (e_1^{\mathcal{I}_0})^E$
- for individuals: $\langle \Sigma, \mathcal{I}_0 \rangle \models e_1(e_2)$ if $(e_2^{\mathcal{I}_0})^E \in (e_1^{\mathcal{I}_0})^E$
- for relations: $\langle \Sigma, \mathcal{I}_0 \rangle \models e_1(e_2, e_3)$ if $(e_2^{\mathcal{I}_0}, e_3^{\mathcal{I}_0}) \in (e_1^{\mathcal{I}_0})^R$
- for attributes: $\langle \Sigma, \mathcal{I}_0 \rangle \models e_1(e_2, e_3)$ if $(e_2^{\mathcal{I}_0}, e_3^{\mathcal{I}_0}) \in (e_1^{\mathcal{I}_0})^A$
- $\ldots$

# The "metagrounding" technique

Let $Q$ be a query over an ontology $\mathcal{O}$ (without identification assertions).

- a metagrounding of $Q$ is a query $Q'$ obtained from $Q$ by substituting the metavariables occurring in $Q$ in class, object property or data property positions with a class, object property and data property expression over $\mathcal{O}$, respectively
  - e.g., if $\mathcal{O}_1$ contains the classes $A, B, C$ and the object property $R$, and $Q$ is the query
    $$Q_1() \leftarrow A \sqsubseteq \neg x, B(y), R(x,z), z(y)$$
    then a metagrounding of $Q$ is the query $Q'$ obtained by applying the substitution $\{x \leftarrow C, z \leftarrow C\}$, i.e.,
    $$Q_1() \leftarrow A \sqsubseteq \neg C, B(y), R(C,C), C(y)$$
- Answering $Q$ through metagrounding means computing the union of the answers to all metagroundings of $Q$

# Does metagrounding work?

## Example

- $\mathcal{O}_1 : \{B(F), C(F), A \sqsubseteq \neg C, R(C, A), R(B, C), A(E)\}$
- $Q_1() \leftarrow A \sqsubseteq \neg x, B(y), R(x, z), z(y)$

Although no metagrounding of $Q_1$ is true, one can show that $Q_1$ is indeed true, by partitioning the models of $\mathcal{O}_1$ into

1. those for which $A$ and $B$ are disjoint, and
2. those for which $A$ and $B$ are not disjoint

and showing that the metagrounding $(x \leftarrow B, z \leftarrow C, y \leftarrow F)$ makes $Q_1$ true in (1), and the metagrounding $(x \leftarrow C, z \leftarrow A, y \leftarrow F)$ makes $Q_1$ true in (2).

## Metagrounding does not suffice

In general, answering metaqueries cannot be done through metagrounding. Note that in the above example, the "culprit" is the uncertainty of the axiom $A \sqsubseteq \neg B$.

# Uncertain axioms and TBox-complete ontologies

**Definition**

- An axiom $\alpha$ over the alphabet of $\mathcal{O}$ is certain if either $\mathcal{O} \models \alpha$, or $\mathcal{O} \models \neg\alpha$.
- $\mathcal{O}$ is TBox-complete if there exists no negative axiom that can be expressed over the alphabet of $\mathcal{O}$ that is not certain

- TBox-completeness can be checked in quadratic time w.r.t. the size of the ontology alphabet
- a methodology can be devised to obtain a TBox-complete ontology from an ontology that is not TBox-complete, keeping the same "intuitive intended models"
- TBox-complete ontologies are common in practice. For example, every ontology designed following the traditional methodology for designing ER schemas can be naturally turned into a TBox-complete ontology

# Uncertain axioms and TBox-complete ontologies

- There is a model $can(\langle\mathcal{O},\mathcal{A}\rangle)$ of $\langle\mathcal{O},\mathcal{A}\rangle$ such that, for each model $M'$ of $\langle\mathcal{O},\mathcal{A}\rangle$, there is an extended homomorphism from $can(\langle\mathcal{O},\mathcal{A}\rangle)$ to $M'$. Extended means that the homomorphism preseves also TBox assertions, e.g., if $c_1^{can(\langle\mathcal{O},\mathcal{A}\rangle)} \subseteq c_2^{can(\langle\mathcal{O},\mathcal{A}\rangle)}$, then $h(c_1)^{M'} \subseteq h(c_2)^{M'}$.
- It can be shown that in $can(\langle\mathcal{O},\mathcal{A}\rangle)$ there are exactly the same classes and relations we have in $\langle\mathcal{O},\mathcal{A}\rangle$, and the TBox assertions that are satisfied in $can(\langle\mathcal{O},\mathcal{A}\rangle)$ are exactly those that are logically implied by $\langle\mathcal{O},\mathcal{A}\rangle$.
- This allows us to show that given a TBox-complete ontology $\mathcal{O}$ and a query $Q$, $Q$ can be answered by applying the metagrounding technique, i.e. $Q$ is true if at least one of its metagrounding is true.

# Answering metaqueries over TBox-complete ontologies through metagrounding

**Query answering algorithm**

**input** ontology $\mathcal{O}$, query $Q$
**if** there exists a metagrounding $Q'$ such that $\mathcal{O} \models int(Q')$ and $\mathcal{O} \models ext(Q')$, where

- $int(Q')$ denotes the TBox atoms of $Q'$, and
- $ext(Q')$ denotes the ABox atoms of $Q'$

**then return** *true*
**else return** *false*

- the algorithm is sound and complete for TBox-complete ontologies
- $\mathcal{O} \models int(Q')$ and $\mathcal{O} \models ext(Q')$ can be checked by using any off-the-shelf OBDM inference and querying systems

# The general case

Let $\mathcal{U}^{\mathcal{O}}$ be the set of negative assertions that can be expressed over the alphabet of $\mathcal{O}$ and are uncertain in $\mathcal{O}$

**Definition**

- If $\alpha \in \mathcal{U}^{\mathcal{O}}$, then a violation set of $\alpha$ w.r.t. $\mathcal{O}$ is a minimal set $\mathcal{V}_{\alpha,\mathcal{O}}$ of ABox axioms over the predicates of $\mathcal{O}$ and a set of fresh individuals not in $\mathcal{O}$, such that $\alpha \cup \mathcal{V}_{\alpha,\mathcal{O}}$ is unsatisfiable
- If $\sigma \subseteq \mathcal{U}^{\mathcal{O}}$, then the $\sigma$-completion of $\mathcal{O}$, denoted $\mathcal{O}^{\sigma}$, is the ontology $\mathcal{O} \cup \sigma \cup \mathcal{C}^{\mathcal{U}^{\mathcal{O}} \setminus \sigma}$, where $\mathcal{C}^{\mathcal{U}^{\mathcal{O}} \setminus \sigma}$ is the union of the violation sets of axioms in $\mathcal{U}^{\mathcal{O}}$ that are not in $\sigma$

Note: Intuitively, $\mathcal{O}^{\sigma}$ is obtained from $\mathcal{O}$ by adding all axioms in $\sigma$, and suitable axioms in such a way that all axioms in $\mathcal{U}^{\mathcal{O}} \setminus \sigma$ are violated
Note: $\mathcal{O}^{\sigma}$ is TBox-complete

# Violation sets and ontology completion – example

**Example**

For the following ontology $\mathcal{O}_2$:

$\{B(F), C(F), A \sqsubseteq \neg C, R(E,E), R(F,F), R(C,A), R(B,C), A(E)\}$

We have $\mathcal{U}^{\mathcal{O}_2} = \{A \sqsubseteq \neg B\}$, and

- for $\sigma_1 = \{A \sqsubseteq \neg B\}$, we have $\mathcal{O}_2^{\sigma_1} = \mathcal{O}_2 \cup \{A \sqsubseteq \neg B\}$.
- for $\sigma_2 = \emptyset$, we have $\mathcal{O}_2^{\sigma_2} = \mathcal{O}_2 \cup \{A(s), B(s)\}$

# Algorithm for answering metaqueries over general ontologies

By exploiting the notions of violation set and ontology completion, we are able to derive the following sound and complete algorithm:

## Query answering algorithm

input: ontology $\mathcal{O}$, query $Q$
if there exists $\sigma \subseteq \mathcal{U}^{\mathcal{O}}$ such that $\mathcal{O}^{\sigma} \not\models Q$
then return *false*
else return *true*

## Complexity

| | **ABox complexity** | **TBox complexity** | **Combined complexity** |
|---|---|---|---|
| **TBox-complete ontologies** | $\mathrm{AC}^0$ | PTime | NP-complete |
| **General ontologies** | $\mathrm{AC}^0$ | coNP-complete | $\Pi_2^p$-complete |

# The notion of dynamic classification

Metamodeling can be used to express important modeling patterns, such as dynamic classification.

- Static classifications:
  Person is classified by sex into Male and Female. This means that Person(John), and sex(John,Male) implies Male(John).
- Dynamic classifications:
  Product is classified by HasProductType.

$$\mathtt{HasProductType} \sqsubseteq \mathtt{rdf{:}type}$$
$$\exists\mathtt{HasProductType} \sqsubseteq \mathtt{Product}$$
$$(\textbf{funct}\ \mathtt{HasProductType})$$
$$\forall x\ (\exists y \mathtt{HasProductType}(y,x)) \leftrightarrow x \sqsubseteq \mathtt{Product}$$
$$\forall x \forall y\ (\mathtt{rdf{:}type}(x,y) \wedge \exists z\ \mathtt{HasProductType}(z,y)) \rightarrow \mathtt{HasProductType}(x,y)$$

Note that the following is implied:

$$\forall x \forall y \forall z \forall w\ \mathtt{HasProductType}(x,z) \wedge \mathtt{HasProductType}(y,w) \wedge z \neq w \rightarrow z \sqsubseteq \neg w$$

Note that such metamodeling pattern requires to go beyond OWL 2 QL

# Outline

# The problem of inconsistency

Up to now, we have implicitly assumed to deal with satisfiable OBDM systems, but in practice the OBDM system can be unsatisfiable.

**Problem**

Query answering based on classical logic becomes meaningless in the presence of inconsistency (ex falso quodlibet).

# Example: an inconsistent *DL-Lite* ontology

**$\mathcal{O}$**

RedWine $\sqsubseteq$ Wine
RedWine $\sqsubseteq$ $\neg$WhiteWIne
Wine $\sqsubseteq$ $\exists$producedBy
Wine $\sqsubseteq$ $\neg$Winery
$\exists$producedBy$^-$ $\sqsubseteq$Winery

WhiteWine $\sqsubseteq$ Wine
Wine $\sqsubseteq$ $\neg$Beer
$\exists$producedBy $\sqsubseteq$ Wine
Beer $\sqsubseteq$ $\neg$ 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

**Problem**

To handle classically-inconsistent OBDM systems in a more meaningful way, one needs to change the semantics.

The semantics proposed in [Lembo et al, RR 2010] for inconsistent OBDM 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*$_{\mathcal{A},id}$), so that inconsistencies are caused by the interaction between the data at $\mathcal{S}$ and the other components of the system, i.e., between $\mathcal{M}(\mathcal{S})$ and $\mathcal{O}$
- We resort to the notion of *repair* [Arenas et al, 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 $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$.

  See [Leopoldo Bertossi, "Database Repairing and Consistent Query Answering", *Synthesis Lectures on Data Management*, Vol. 3, No. 5, Morgan and Claypool].

# Inconsistent-tolerant semantics

What does it mean for $\mathcal{A}$ to be "minimally different" from $\langle\mathcal{O}, \mathcal{S}, \mathcal{M}\rangle$? We base this concept on the notion of symmetric difference.

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

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

**Definition (Repair)**

Let $\mathcal{K} = \langle\mathcal{O}, \mathcal{S}, \mathcal{M}\rangle$ be an OBDM system. A repair of $\mathcal{K}$ is an ABox $\mathcal{A}$ such that:

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

# 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)}

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 OBDM system, and let $\alpha$ be a ground atom. Deciding whether $\alpha$ is logically implied by every repair of $\mathcal{K}$ is coNP-complete with respect to data complexity.*

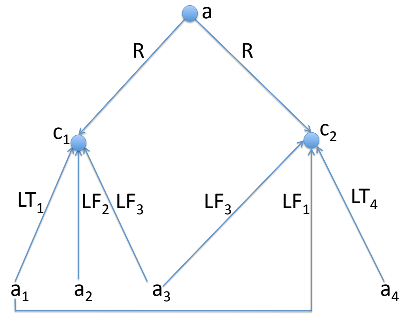
# coNP hardness of the AR-semantics

## Ontology $\mathcal{O}$

$$\exists R \sqsubseteq Unsat$$
$$\exists R^- \sqsubseteq \neg \exists LT_1^-$$
$$\exists R^- \sqsubseteq \neg \exists LF_1^-$$
$$\exists R^- \sqsubseteq \neg \exists LT_2^-$$
$$\exists R^- \sqsubseteq \neg \exists LF_2^-$$
$$\exists R^- \sqsubseteq \neg \exists LT_3^-$$
$$\exists R^- \sqsubseteq \neg \exists LF_3^-$$
$$\exists LT_1 \sqsubseteq \neg \exists LF_1$$
$$\exists LT_1 \sqsubseteq \neg \exists LF_2$$
$$\exists LT_1 \sqsubseteq \neg \exists LF_3$$
$$\exists LF_1 \sqsubseteq \neg \exists LT_2$$
$$\exists LF_1 \sqsubseteq \neg \exists LT_3$$
$$\exists LT_2 \sqsubseteq \neg \exists LF_2$$
$$\exists LT_2 \sqsubseteq \neg \exists LF_3$$
$$\exists LF_2 \sqsubseteq \neg \exists LT_3$$
$$\exists LT_3 \sqsubseteq \neg \exists LF_3$$

## 3-CNF formula $\phi$: $(a_1 \vee \neg a_2 \vee \neg a_3) \wedge (\neg a_3 \vee a_4 \vee \neg a_1)$

## ABox $\mathcal{A}$ corresponding to $\phi$




## $\phi$ satisfiable iff $\langle \mathcal{O}, \mathcal{A} \rangle \not\models_{AR} Unsat(a)$

# When in doubt, throw it out: the IAR semantics

Other intractability results of the AR semantics, even for simpler languages (e.g., [Bienvenu et al 2012-2015])

**Idea: The IAR semantics**

We consider the "intersection of all repairs", and the set of models of such intersection under $\mathcal{O}$ as the semantics of the system (When in Doubt, Throw It Out).

Note that the IAR semantics is an approximation of the AR semantics

# 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 the intersection $\mathcal{A}$ of all repairs of $\mathcal{K}$, 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 $\mathcal{K} \models_{IAR} q(\vec{t})$ is equivalent to $\vec{t} \in q'(\mathcal{M}(\mathcal{S}))$. Then, evaluate $q'$ over $\mathcal{M}(\mathcal{S})$.

We have devised a rewriting technique which encodes a UCQ $q$ into a FOL query $q'$ which, evaluated against the original $\mathcal{M}(\mathcal{S})$, retrieves only the certain answers of $q$ w.r.t the IAR semantics [Lembo et al, JSW 2015].

# 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 $\text{PerfectRef}_{IAR}(Q, \mathcal{O}, \mathcal{M})$ as

  $$\text{MapRewriting}_{\mathcal{M}}(\text{IncRewritingUCQ}_{IAR}(\text{PerfectRef}(Q, \mathcal{O}), \mathcal{O}))$$

- we evaluate $\text{PerfectRef}_{IAR}(Q, \mathcal{O}, \mathcal{M})$ over $\mathcal{S}$

where

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

# Example

Let us consider the CQ

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

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

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

# Results

## Theorem

*Let $Q$ be a UCQ over $\langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle$. Deciding whether $\vec{t} \in cert_{IAR}(Q, \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle)$ is in $AC^0$ in data complexity.*

## Complexity

| problem | $AR$-semantics | $IAR$-semantics |
|---|---|---|
| instance checking | coNP-complete | in $AC_0$ |
| UCQ answering | coNP-complete | in $AC_0$ |

# Outline

# Conclusions: Many challenges for OBDM

- Query answering in *OBDM systems* (ONTOP, MASTRO, etc.)
  - More optimizations in query answering
  - Rewriting wrt mapping (even GAV mapping are problematic)
  - More powerful metamodeling and optimized metaquerying
- *Inconsistency-tolerant query answering*
  - More semantics?
  - Optimizations
  - Preferences over repairs
  - More powerful (meta)querying
- *Ontology-based update*
  - Semantics
  - Pushing the updates to the data sources
  - Updates in the presence of inconsistencies
- Experimenting OBDM in real applications

SAPIENZA UNIVERSITÀ DI ROMA