Inconsistency-tolerant Query Answering in Ontology-based Data Access

Domenico Lembo

Dipartimento di Ingegneria Informatica, Automatica e Gestionale A. Ruberti
Sapienza Università di Roma, Italy

Lectures from the Ph.D. Course on “Inconsistency Management in Data and Knowledge Bases”

Outline

1 Ontology-based data access
2 $DL$-$Lite_A$: an ontology language for accessing data
   - Syntax and Semantics of $DL$-$Lite_A$
   - Query answering in $DL$-$Lite_A$
   - Ontology satisfiability in $DL$-$Lite_A$
   - Query answering by rewriting in $DL$-$Lite_A$
3 Dealing with inconsistency in ontology based data access
   - ABox Repair Semantics: Intractable Consistent Query Answering
   - Intersection ABox Repair Semantics: Towards first-order rewritable CQA
   - FOL-rewritability of CQA in $DL$-$Lite_A$
   - Closed ABox Repair Semantics
   - Related Literature
4 References
Data integration

- Data integration is the problem of providing a centralized, reconciled access to a set of heterogeneous data sources
- In this setting inconsistency is very frequent
- Indeed, data are managed autonomously by each source, and they are not natively required to be coherent with integrity constraints specified in the centralized schema, a.k.a. mediated or global schema
- Also, data in different sources may easily result mutually incoherent and thus may give rise to inconsistency once merged together
Client’s queries $q$ on the **global schema** is answered by accessing data in the **sources** through suitable **mappings**.

$\sim$ *It achieves “logical transparency” as opposed to simple physical transparency.*

---

**Data integration: a key point**

*It is not assumed that information the global schema has on the data is complete!*

$\sim$ As more sources are added more information and precise data are obtained.
Data integration: a key point

*It is not assumed that information the global schema has on the data is complete!*

\[ \sim \text{As more sources are added more information and precise data are obtained.} \]

Given the global schema, the mapping, and the data at sources, **Query answering** amounts to compute the so-called **certain answers**, i.e., those tuples that are returned by the evaluation of the user query over each model of the system (roughly, each database satisfying the global schema and the mappings).
Ontology-based data access: conceptual layer & data layer

**Ontology-based data access (OBDA)** is a form of data integration based on the idea of specifying the global schema in terms of (the intensional component of an) ontology.

The ontology is a conceptual representation of the domain of interest expressed in a logic-based language, commonly a (tractable) Description Logic terminology (TBox).

Ontology-based data integration: the sources

**Data sources are external, independent, heterogeneous, multiple information systems.**

Based on industrial solutions we can:

- Wrap the sources and see all of them as relational databases.
- Use federated database tools to see the multiple sources as a single one.

**We can see the sources as a single (remote) relational database.**
Ontology-based data access: mappings

Mappings semantically link data at the sources with the ontology.

Several general forms of mappings based on queries have been considered:

- **GAV**: map a query over the source to an element in the global view
  – *most used form of mappings*

- **LAV**: map a relation in the source to a query over the global view
  – *mathematically elegant, but difficult to use in practice*

- **GLAV**: map a query over the sources to a query over the global view
  – *the most general form of mappings*
Ontology-based data access: Query rewriting

The most popular approach to query answering in OBDA systems is through **query rewriting**:

1. The original query is first rewritten with respect to the ontology into a new query over the ontology; we call this step the **ontology rewriting** of the query;

2. The query thus obtained is then rewritten into a source database query using the mapping assertions; we call this step the **mapping rewriting** of the query.

**Note:** Under **GAV mappings**, mapping rewriting can be based on **unfolding**, which amounts to substituting every atom of the query with the corresponding query in the mapping.
Example – query rewriting

Consider the query \( q(x) \leftarrow \text{worksFor}(x, y) \)

the ontology rewriting is

\[
\begin{align*}
\text{r}_{q, T} = q(x) & \leftarrow \text{worksFor}(x, y) \\
q(x) & \leftarrow \text{Employee}(x)
\end{align*}
\]

More on ontology rewriting later

Example – splitting the mapping

To compute the mapping rewriting, we first split \( \mathcal{M} \) as follows (always possible, since queries in the right-hand side of assertions in \( \mathcal{M} \) are without non-distinguished variables):

\[
\begin{align*}
M_{1,1}: \quad & \text{SELECT SSN, PrName} \quad \mapsto \text{Employee}(\text{pers}(SSN)) \\
& \text{FROM D}_1 \\
M_{1,2}: \quad & \text{SELECT SSN, PrName} \quad \mapsto \text{Project}(\text{proj}(PrName)) \\
& \text{FROM D}_1 \\
M_{1,3}: \quad & \text{SELECT SSN, PrName} \quad \mapsto \text{projectName}(\text{proj}(PrName), PrName) \\
& \text{FROM D}_1 \\
M_{1,4}: \quad & \text{SELECT SSN, PrName} \quad \mapsto \text{workFor}(\text{pers}(SSN), \text{proj}(PrName)) \\
& \text{FROM D}_1 \\
M_{2,1}: \quad & \text{SELECT SSN, Salary} \quad \mapsto \text{Employee}(\text{pers}(SSN)) \\
& \text{FROM D}_2, D_3 \\
& \text{WHERE D}_2.\text{Code} = D_3.\text{Code} \\
M_{2,2}: \quad & \text{SELECT SSN, Salary} \quad \mapsto \text{salary}(\text{pers}(SSN), Salary) \\
& \text{FROM D}_2, D_3 \\
& \text{WHERE D}_2.\text{Code} = D_3.\text{Code}
\end{align*}
\]
Example – unfolding

Then, we unify each atom of the query

\[ r_{q,T} = q(x) \leftarrow \text{worksFor}(x,y) \]
\[ q(x) \leftarrow \text{Employee}(x) \]

with the right-hand side of the assertion in the split mapping, and substitute such atom with the left-hand side of the mapping

\[ q(\text{pers}(\text{SSN})) \leftarrow \text{SELECT SSN, PrName} \]
\[ \quad \text{FROM } D_1 \]
\[ q(\text{pers}(\text{SSN})) \leftarrow \text{SELECT SSN, Salary} \]
\[ \quad \text{FROM } D_2, D_3 \]
\[ \quad \text{WHERE } D_2.\text{CODE} = D_3.\text{CODE} \]

The construction of object terms can be pushed into the SQL query, by resorting to SQL functions to manipulate strings (e.g., string concat).

Example – SQL query over the source database

```
SELECT concat(concat('pers (',SSN,')'))
FROM D1
UNION
SELECT concat(concat('pers (',SSN,')'))
FROM D2, D3
```
Ontology-based data access: Getting rid of GAV mappings

GAV mappings are the ones we consider in these slides, since recent research showed that GAV mappings are the only ones that allow query answering to scale on real-world applications (if coupled with a lightweight ontology language).

Thus, in the following we get rid of the mapping, and consider the simplified setting of a stand-alone ontology, and thus concentrate only on the ontology rewriting step.

All techniques we will see naturally extend to the mappings, modulo a further unfolding step.

In considering a stand alone ontology we will refer to its two components: The intensional component (TBox) and an extensional component (ABox).

Taking in mind the OBDA architecture, the ABox can be conceived as the instance of the TBox we can obtain by using the mapping bottom-up, i.e., evaluating each query in the mapping over the underlying database, and instanting the associated TBox elements.

In other terms, such an ABox is a set containing all the instance assertions implied by the mapping and by the data at the sources.

Of course, the answers to a query over the OBDA system can be obtained by evaluating the ontology rewriting on the computed ABox, seen simply as a relational database.
Query answering over ontologies is a form of reasoning under incomplete information, and may be costly in the size of the data, which is not suited for OBDA.

The DL-Lite family is a family of Description Logics (DLs) optimized according to the tradeoff 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 DL-Lite_A, an expressive member of the DL-Lite family, which is at the basis of OWL2 QL, one of the standard profiles of OWL 2.
**DL-Lite\(_A\) ontologies**

TBox assertions:
- Concept (class) inclusion assertions: \( B \sqsubseteq C \), with:
  \[
  B \rightarrow A \mid \exists Q \\
  C \rightarrow B \mid \neg B
  \]
- Role (property) inclusion assertions: \( Q \sqsubseteq R \), with:
  \[
  Q \rightarrow P \mid P^- \\
  R \rightarrow Q \mid Q^-
  \]
- Functionality assertions: \( \text{(funct } Q) \)
- **Proviso:** functional properties cannot be specialized.

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

*Note:* DL-Lite\(_A\) distinguishes also between object and data properties (i.e., roles from attributes), but we ignore this here, for simplicity.

**Correspondence with first-order syntax**

DLs are fragments of first-order logic (FOL), therefore DL-Lite\(_A\) assertions can be rephrased in classical FOL syntax

| \( A_1 \sqsubseteq A_2 \) | \( \forall x. A_1(x) \rightarrow A_2(x) \) |
| \( A_1 \sqsubseteq \neg A_2 \) | \( \forall x. A_1(x) \land A_2(x) \rightarrow \bot \) |
| \( \exists P \sqsubseteq A_1 \) | \( \forall x, y. P(x, y) \rightarrow A_1(x) \) |
| \( \exists P^- \sqsubseteq A_2 \) | \( \forall x, y. P(x, y) \rightarrow A_2(y) \) |
| \( A_1 \sqsubseteq \exists P \) | \( \forall x. A_1(x) \rightarrow \exists y. P(x, y) \) |
| \( A_2 \sqsubseteq \exists P^- \) | \( \forall x. A_2(x) \rightarrow \exists y. P(y, x) \) |
| \( \text{(funct } P) \) | \( \forall x, y, z. P(x, y) \land P(x, z) \rightarrow y = z \) |
| \( \text{(funct } P^-) \) | \( \forall x, y, z. P(y, x) \land P(z, x) \rightarrow y = z \) |
| \( Q_1 \sqsubseteq Q_2 \) | \( \forall x, y. Q_1(x, y) \rightarrow Q_2(x, y) \) |
| \( Q_1 \sqsubseteq \neg Q_2 \) | \( \forall x, y. Q_1(x, y) \land Q_2(x, y) \rightarrow \bot \) |
For the sake of completeness, we report here the formal semantics for \(DL-Lite_A\) ontologies, which is given in terms of interpretations, in the standard FOL sense.

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

- a nonempty set \(\Delta^\mathcal{I}\), the domain of \(\mathcal{I}\)
- an interpretation function \(\cdot^\mathcal{I}\), which maps
  - each individual \(c\) to an element \(c^\mathcal{I}\) of \(\Delta^\mathcal{I}\)
  - each atomic concept \(A\) to a subset \(A^\mathcal{I}\) of \(\Delta^\mathcal{I}\)
  - each atomic role \(P\) to a subset \(P^\mathcal{I}\) of \(\Delta^\mathcal{I} \times \Delta^\mathcal{I}\)

The interpretation function is extended to complex concepts and roles according to their syntactic structure.

\(DL-Lite_A\) (as all DLs of the DL-Lite family) adopts the Unique Name Assumption (UNA), i.e., different individuals denote different objects.
<table>
<thead>
<tr>
<th>Construct</th>
<th>DL-Lite_\mathcal{A}</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA between classes</td>
<td>\mathcal{A}_1 \sqsubseteq \mathcal{A}_2</td>
</tr>
<tr>
<td>Disjointness between classes</td>
<td>\mathcal{A}_1 \sqsubseteq \lnot \mathcal{A}_2</td>
</tr>
<tr>
<td>Domain and range of properties</td>
<td>\exists P \sqsubseteq \mathcal{A}_1 \quad \exists P^- \sqsubseteq \mathcal{A}_2</td>
</tr>
<tr>
<td>Mandatory participation (min card = 1)</td>
<td>\mathcal{A}_1 \sqsubseteq \exists P \quad \mathcal{A}_2 \sqsubseteq \exists P^-</td>
</tr>
<tr>
<td>Functionality of relations (max card = 1)</td>
<td>(\text{funct } P) \quad (\text{funct } P^-)</td>
</tr>
<tr>
<td>ISA between properties</td>
<td>\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2</td>
</tr>
<tr>
<td>Disjointness between properties</td>
<td>\mathcal{Q}_1 \sqsubseteq \lnot \mathcal{Q}_2</td>
</tr>
</tbody>
</table>

**Note:** DL-Lite_\mathcal{A} cannot capture completeness of a hierarchy. This would require disjunction (i.e., OR).
Ontology satisfiability

The following is standard in a FOL theory:

- Given a \( DL-Lite_A \) assertion \( \alpha \), either an ABox and a TBox assertion, and an interpretation \( I \), we denoted with \( I \models \alpha \) the fact that \( I \) satisfies \( \alpha \) (according to the semantics seen before).

- A \( DL-Lite_A \) ontology \( O = (T, A) \) is **Satisfiable** if there exists at least one interpretation that satisfies all its assertions, i.e., there exists \( I \) such that \( I \models \alpha \) for each \( \alpha \) in \( T \) and each \( \alpha \) in \( A \). Such an interpretation is a **model**. We denote with \( Mod(O) \) the set of models of \( O \).
Boolean Unions of Conjunctive Queries

- A boolean conjunctive query (BCQ) $q$ over a DL ontology is a FOL sentence of the form:
  $$\exists \vec{y}. \text{conj}(\vec{t})$$
  where $\vec{y}$ is a set of variables, $\vec{t}$ is a set of terms (i.e., constants or variables) such that each variable in $\vec{t}$ is also in $\vec{y}$, and $\text{conj}(\vec{t})$ is a conjunction of atoms of the form $A(z)$, $P(z, z')$, where $A$ is an atomic concept, $P$ is an atomic role, and $z, z'$ are terms. *Sometimes we use the Datalog notation $q() \leftarrow \text{conj}(\vec{t})$.*

- A boolean union of conjunctive queries (BUCQ), is a FOL sentence of the form
  $$Q = \bigvee_{i=1}^{n} q_i$$
  where each $q_i$ is a BCQ. We will often refer to a BUCQ as a set of BCQs.

*As usual, results on BUCQ carry over non-boolean UCQ in a straightforward way.*

BUCQ entailment

- An interpretation $I$ satisfies a BUCQ $Q = \bigvee_{i=1}^{n} \exists \vec{y}_i. \text{conj}_i(\vec{t}_i)$, denoted $I \models Q$, if the sentence $\bigvee_{i=1}^{n} \exists \vec{y}_i. \text{conj}_i(\vec{t}_i)$ evaluates to true over $I$.

- A DL ontology $O$ satisfies $Q$, or, analogously, $Q$ evaluates to *true* over $O$, denoted $O \models Q$ if $I \models Q$ for every $I \in \text{Mod}(O)$.

The query answering reasoning service is defined as follows: given a DL ontology $O$ and a boolean query $q$ (either a BCQ or a BUCQ) over $O$, verify whether $O \models q$.

A simplified form of query answering is *instance checking*, defined as follows: given a DL ontology $O$ and an ABox assertion $\alpha$, verify whether $O \models \alpha$.
The Notion of FOL-rewritability

The following notion is a crucial one:

**Definition**

Query answering in a DL \( L \) is **FOL-rewritable** if for every BUCQ \( q \) and every TBox \( T \) expressed over \( L \), one can effectively compute a FOL query \( q_r \), over the alphabet of \( T \), such that for every ABox \( A \) such that \( \langle T, A \rangle \) is satisfiable, \( \langle T, A \rangle \models q \) if and only if \( \langle \emptyset, A \rangle \models q_r \). We call such \( q_r \) a **perfect FOL rewriting** (or simply perfect rewriting) of \( q \) w.r.t. \( T \).

Since the evaluation of a FOL query over an ABox is in \( AC^0 \) in data complexity [1], FOL-rewritability of query answering has as an immediate consequence that such a problem is in \( AC^0 \) in data complexity.

Notice that verifying whether \( \langle \emptyset, A \rangle \models q_r \) means evaluating \( q_r \) over \( A \) seen as a database, and therefore can be demanded to a **RDBMS**.

*Note: we will often use \( A \models q \) in place of \( \langle \emptyset, A \rangle \models q \).*

---

Query answering over satisfiable \( DL-Lite_A \) ontologies

- We first take answering of BUCQs over satisfiable \( DL-Lite_A \) ontologies.
- Then, we show how to exploit query answering over satisfiable ontologies to establish ontology satisfiability.
- Next, we show how to solve query answering by rewriting into FOL for BUCQs over \( DL-Lite_A \) ontologies.
- Finally, we deal with the problem of inconsistency-tolerant query answering.

**Remark**

We call **positive inclusions** (PIs) assertions of the form

\[
B_1 \sqsubseteq B_2 \\
Q_1 \sqsubseteq Q_2
\]

whereas we call **negative inclusions** (NIs) assertions of the form

\[
B_1 \sqsubseteq \neg B_2 \\
Q_1 \sqsubseteq \neg Q_2
\]
Query answering over satisfiable $DL$-Lite$_A$ ontologies

- We first tackle answering of BUCQs over satisfiable $DL$-Lite$_A$ ontologies.
- Then, we show how to exploit query answering over satisfiable ontologies to establish ontology satisfiability.
- Next, we show how to solve query answering by rewriting into FOL for BUCQs over $DL$-Lite$_A$ ontologies.
- Finally, we deal with the problem of inconsistency-tolerant query answering.

**Remark**

We call **positive inclusions (PIs)** assertions of the form

$$\begin{align*}
B_1 &\sqsubseteq B_2 \\
Q_1 &\sqsubseteq Q_2
\end{align*}$$

whereas we call **negative inclusions (NIs)** assertions of the form

$$\begin{align*}
B_1 &\sqsubseteq \neg B_2 \\
Q_1 &\sqsubseteq \neg Q_2
\end{align*}$$

**Theorem**

Let $q$ be a boolean UCQs and $T = T_{PI} \cup T_{NI} \cup T_{funct}$ be a TBox s.t.

- $T_{PI}$ is a set of PIs
- $T_{NI}$ is a set of NIs
- $T_{funct}$ is a set of functionalities.

For each ABox $\mathcal{A}$ such that $\langle T, \mathcal{A} \rangle$ is satisfiable, we have that

$$\langle T, \mathcal{A} \rangle \models q \iff \langle T_{PI}, \mathcal{A} \rangle \models q.$$

**Proof [intuition]**

$q$ is a positive query, i.e., it does not contain atoms with negation nor inequality. $T_{NI}$ and $T_{funct}$ only contribute to infer new negative consequences, i.e., sentences involving negation.
Query answering over satisfiable $DL-Lite_\mathcal{A}$ ontologies

**Theorem**

Let $q$ be a boolean UCQs and $\mathcal{T} = \mathcal{T}_{PI} \cup \mathcal{T}_{NI} \cup \mathcal{T}_{funct}$ be a TBox s.t.
- $\mathcal{T}_{PI}$ is a set of PIs
- $\mathcal{T}_{NI}$ is a set of NIs
- $\mathcal{T}_{funct}$ is a set of functionalities.

For each ABox $\mathcal{A}$ such that $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, we have that

$$\langle \mathcal{T}, \mathcal{A} \rangle \models q \iff \langle \mathcal{T}_{PI}, \mathcal{A} \rangle \models q.$$ 

**Proof [intuition]**

$q$ is a positive query, i.e., it does not contain atoms with negation nor inequality. $\mathcal{T}_{NI}$ and $\mathcal{T}_{funct}$ only contribute to infer new negative consequences, i.e., sentences involving negation.

Satisfiability of $DL-Lite_\mathcal{A}$ ontologies

$\langle \mathcal{T}, \emptyset \rangle$ is always satisfiable. That is, inconsistency in $DL-Lite_\mathcal{A}$ may arise only when ABox assertions contradict the TBox.

$\langle \mathcal{T}_{PI}, \mathcal{A} \rangle$, where $\mathcal{T}_{PI}$ contains only PIs, is always satisfiable. That is, inconsistency in $DL-Lite_\mathcal{A}$ may arise only when ABox assertions violate functionalities or NIs.

Example:  

**TBox** $\mathcal{T}$: Professor $\sqsubseteq \lnot$ Student  
$\exists$ teaches $\sqsubseteq$ Professor  
($funct$ teaches$\lnot$)  

**ABox** $\mathcal{A}$: teaches(John,databases)  
Student(John)  
teaches(Mark,databases)

Violations of functionalities and of NIs can be checked separately!
Satisfiability of $DL$-$Lite_A$ ontologies

$\langle T, \emptyset \rangle$ is always satisfiable. That is, inconsistency in $DL$-$Lite_A$ may arise only when ABox assertions contradict the TBox.

$\langle T_{PI}, A \rangle$, where $T_{PI}$ contains only PIs, is always satisfiable. That is, inconsistency in $DL$-$Lite_A$ may arise only when ABox assertions violate functionalities or NIs.

Example:

$\text{TBox } T : \text{Professor } \sqsubseteq \neg \text{Student} \quad \exists \text{teaches } \sqsubseteq \text{Professor} \quad (\text{funct teaches}^-)$

$\text{ABox } A : \text{teaches}(\text{John,databases}) \quad \text{Student}(\text{John}) \quad \text{teaches}(\text{Mark,databases})$

Violations of functionalities and of NIs can be checked separately!
Satisfiability of $DL$-$Lite_A$ ontologies

$\langle T, \emptyset \rangle$ is always satisfiable. That is, inconsistency in $DL$-$Lite_A$ may arise only when ABox assertions contradict the TBox.

$\langle T_{PI}, A \rangle$, where $T_{PI}$ contains only PIs, is always satisfiable. That is, inconsistency in $DL$-$Lite_A$ may arise only when ABox assertions violate functionalities or NIs.

**Example:**

**TBox $T$:**
- Professor $\sqsubseteq \neg$ Student
- $\exists$teaches $\sqsubseteq$ Professor
  (funct teaches$^-$)

**ABox $A$:**
- teaches(John, databases)
- Student(John)
- teaches(Mark, databases)

Violations of functionalities and of NIs can be checked separately!

---

Satisfiability of $DL$-$Lite_A$ ontologies: Checking functs

**Theorem**

Let $T_{PI}$ be a TBox with only PIs, and $(\text{funct } Q)$ a functionality assertion. Then, for any ABox $A$,

$\langle T_{PI} \cup \{ (\text{funct } Q) \}, A \rangle$ is sat iff $A \not\models \exists x, y, z. Q(x, y) \land Q(x, z) \land y \neq z$.

**Proof [sketch]**

$\langle T_{PI} \cup \{ (\text{funct } Q) \}, A \rangle$ is satisfiable iff $\langle T_{PI}, A \rangle \not\models \neg(\text{funct } Q)$. This holds iff $A \not\models \neg(\text{funct } Q)$ (separability property – sophisticated proof). From separability, the claim easily follows, by noticing that $(\text{funct } Q)$ corresponds to the FOL sentence $\forall x, y, z. Q(x, y) \land Q(x, z) \rightarrow y = z$.

For a set of functionalities, we take the union of sentences of the form above (which corresponds to a boolean FOL query).

Checking satisfiability wrt functionalities therefore amounts to evaluate a FOL query over the ABox.
Theorem

Let \( \mathcal{T}_{PI} \) be a TBox with only PIs, and \( (\text{funct } Q) \) a functionality assertion. Then, for any ABox \( \mathcal{A} \),

\[ \langle \mathcal{T}_{PI} \cup \{(\text{funct } Q)\}, \mathcal{A} \rangle \text{ is sat iff } \mathcal{A} \not\models \exists x, y, z. Q(x, y) \land Q(x, z) \land y \neq z. \]

Proof [sketch]

\[ \langle \mathcal{T}_{PI} \cup \{(\text{funct } Q)\}, \mathcal{A} \rangle \text{ is satisfiable iff } \langle \mathcal{T}_{PI}, \mathcal{A} \rangle \not\models \neg(\text{funct } Q). \] This holds iff \( \mathcal{A} \not\models \neg(\text{funct } Q) \) (separability property – sophisticated proof).

From separability, the claim easily follows, by noticing that \( (\text{funct } Q) \) corresponds to the FOL sentence \( \forall x, y, z. Q(x, y) \land Q(x, z) \rightarrow y = z \).

For a set of functionalities, we take the union of sentences of the form above (which corresponds to a boolean FOL query).

Checking satisfiability wrt functionalities therefore amounts to evaluate a FOL query over the ABox.
Example

TBox $T$:  
$\text{Professor} \sqsubseteq \neg \text{Student}$  
$\exists \text{teaches} \sqsubseteq \text{Professor}$  
($\text{funct teaches}^{-}$)

The query we associate to the functionality is:

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

which evaluated over the ABox

ABox $A$:  
$\text{teaches(John, databases)}$  
$\text{Student(John)}$  
$\text{teaches(Mark, databases)}$

returns true.

Satisfiability of $DL$-$Lite_\mathcal{A}$ ontologies: Checking NIs

Theorem

Let $T_{\mathcal{PI}}$ be a TBox with only PIs, and $A_1 \sqsubseteq \neg A_2$ a NI. For any ABox $A$,  
$\langle T_{\mathcal{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, A \rangle$ is sat iff  
$\langle T_{\mathcal{PI}}, A \rangle \models \neg \exists x. A_1(x) \wedge A_2(x)$. 

Proof [sketch]

$\langle T_{\mathcal{PI}} \cup \{A_1 \sqsubseteq \neg A_2\}, A \rangle$ is satisfiable iff  
$\langle T_{\mathcal{PI}}, A \rangle \models \neg (A_1 \sqsubseteq \neg A_2)$. The claim follows easily by noticing that  
$A_1 \sqsubseteq \neg A_2$ corresponds to the FOL sentence  
$\forall x. A_1(x) \rightarrow \neg A_2(x)$. 

The property holds for all kinds of NIs ($A \sqsubseteq \exists Q, \exists Q_1 \sqsubseteq \neg \exists Q_2$, etc.)  
For a set of NIs, we take the union of sentences of the form above (which corresponds to a UCQ). 
Checking satisfiability wrt NIs amounts to answering a UCQ over an ontology with only PIs (this can be reduced to evaluating a UCQ over the ABox – see later).
Satisfiability of $DL$-Lite$_A$ ontologies: Checking NIs

**Theorem**

Let $T_{PI}$ be a TBox with only PIs, and $A_1 \sqsubseteq \neg A_2$ a NI. For any ABox $\mathcal{A}$, 
\[ \langle T_{PI} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A} \rangle \ \text{is satisfiable} \iff \langle T_{PI}, \mathcal{A} \rangle \not\models \exists x. A_1(x) \land A_2(x). \]

**Proof [sketch]**

\[ \langle T_{PI} \cup \{A_1 \sqsubseteq \neg A_2\}, \mathcal{A} \rangle \ \text{is satisfiable} \iff \langle T_{PI}, \mathcal{A} \rangle \not\models \neg (A_1 \sqsubseteq \neg A_2). \]

The claim follows easily by noticing that $A_1 \sqsubseteq \neg A_2$ corresponds to the FOL sentence $\forall x. A_1(x) \rightarrow \neg A_2(x)$.

The property holds for all kinds of NIs ($A \sqsubseteq \exists Q$, $\exists Q_1 \sqsubseteq \exists Q_2$, etc.)

For a set of NIs, we take the union of sentences of the form above (which corresponds to a UCQ).

Checking satisfiability wrt NIs amounts to answering a UCQ over an ontology with only PIs (this can be reduced to evaluating a UCQ over the ABox – see later).

D. Lembo  Inconsistency-tolerant QA in OBDA  Oct 8-9, 2013  (35/90)
Example

**TBox** $\mathcal{T}$:  
$\text{Professor} \sqsubseteq \neg \text{Student}$  
$\exists \text{teaches} \sqsubseteq \text{Professor}$  
($\text{funct teaches}^-$)

The query we associate to the NI is:

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

whose answer over the ontology

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

$\text{teaches}(\text{John}, \text{databases})$

$\text{Student}(\text{John})$

$\text{teaches}(\text{Mark}, \text{databases})$

is true.

---

Checking satisfiability of $DL-Lite_A$ ontologies

Then, we can reduce satisfiability to query answering. As we will see in next slides, query answering of BUCQs in $DL-Lite_A$ is reducible to standard first-order query evaluation. Thus, the following holds.

Satisfiability of a $DL-Lite_A$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is reduced to evaluation of a first order query over $\mathcal{A}$, obtained by uniting

1. the FOL query associated to functionalities in $\mathcal{T}$ to
2. the UCQs produced by a rewriting procedure (depending only on the PIs in $\mathcal{T}$) applied to the query associated to NIs in $\mathcal{T}$.

$\sim$ Ontology satisfiability in $DL-Lite_A$ can be done using RDMBS technology.
Query answering in $DL$-$Lite_{A}$: Query rewriting

To the aim of answering queries, from now on we assume that $T$ contains only PIs.

Given a CQ $q$ and a satisfiable ontology $\mathcal{O} = \langle T, A \rangle$, we verify whether $\mathcal{O} \models q$ as follows:

1. using $T$, reformulate $q$ as a union $r_{q,T}$ of CQs.
2. Evaluate $r_{q,T}$ directly over $A$ managed in secondary storage via a RDBMS.

Correctness of this procedure shows FOL-rewritability of query answering in $DL$-$Lite_{A}$.

$\rightarrow$ Query answering over $DL$-$Lite_{A}$ ontologies can be done using RDMBS technology.
**Intuition:** Use the PIs as basic rewriting rules

\[ q() \leftarrow \text{Student}(x), \text{Professor}(x) \]

\[ \exists \text{teaches} \sqsubseteq \text{Professor} \]

as a logic rule:

\[ \text{Professor}(z) \leftarrow \text{teaches}(z, w) \]

**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 (\( \sigma = \{z/x\} \))

\[ q() \leftarrow \text{Student}(x), \text{teaches}(x, w) \]

We say that the PI \( \exists \text{teaches} \sqsubseteq \text{Professor} \) applies to the atom \( \text{Professor}(x) \).
Query answering in $DL$-$Lite_A$:

**Intuition:** Use the PIs as basic rewriting rules

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

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

as a logic rule:

$$\text{Professor}(z) \leftarrow \text{teaches}(z,w)$$

**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 ($\sigma = \{z/x\}$)

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

We say that the PI $\exists \text{teaches} \sqsubseteq \text{Professor}$ **applies** to the atom $\text{Professor}(x)$.

---

**Answering by rewriting in $DL$-$Lite_A$:**

The algorithm $\text{PerfectRef} \ (q, \mathcal{T}_P)$

1. Rewrite the UCQ $q$ into a new UCQ $q_r$: apply to $q$ in all possible ways the PIs in the TBox $\mathcal{T}$.
2. This corresponds to exploiting ISAs, role typings, and mandatory participations to obtain new queries that could contribute to the answer.
3. Unifying atoms can make applicable rules that could not be applied otherwise.
4. The UCQs resulting from this process is the **perfect rewriting** $r_{q,\mathcal{T}}$.
5. $r_{q,\mathcal{T}}$ is then **encoded into SQL** and evaluated over $\mathcal{A}$ managed in **secondary storage via a RDBMS**, to verify whether $\langle \emptyset, \mathcal{A} \rangle \models r_{q,\mathcal{T}}$, which allows to conclude whether $\langle \mathcal{T}, \mathcal{A} \rangle \models q$. 
Query answering in \( DL-Lite_\mathcal{A} \): Example

**TBox:** \( \text{Professor} \sqsubseteq \exists\text{teaches} \)
\( \exists\text{teaches} \sqsubseteq \sqsubseteq \text{Course} \)

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

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

**ABox:** \( \text{Professor}(\text{Mary}) \)

It is easy to see that the evaluation of \( r_{q,T} \) over \( \mathcal{A} \) returns true, as witnessed by the assertion \( \text{Professor}(\text{Mary}) \).

---

**Outline**

1. Ontology-based data access
2. \( DL-Lite_\mathcal{A} \): an ontology language for accessing data
3. Dealing with inconsistency in ontology based data access
4. References
Example 1: an inconsistent DL-Lite ontology

**TBox**

<table>
<thead>
<tr>
<th>Class</th>
<th>Subsumes</th>
<th>Suppressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedWine</td>
<td>⊑ Wine</td>
<td></td>
</tr>
<tr>
<td>RedWine</td>
<td>⊑ ¬ WhiteWine</td>
<td></td>
</tr>
<tr>
<td>Wine</td>
<td>⊑∃ producedBy</td>
<td></td>
</tr>
<tr>
<td>Wine</td>
<td>⊑¬ Winery</td>
<td></td>
</tr>
<tr>
<td>∃ producedBy</td>
<td>⊑ Winery</td>
<td></td>
</tr>
</tbody>
</table>

**ABox**

- RedWine(grechetto)
- WhiteWine(grechetto)
- Beer(guinnes)
- producedBy(guinnes, sandonna)
- WhiteWine(falanghina)

Techniques seen so far are not adequate. Under standard FOL semantics we can only conclude that every BUCQs posed over the ontology is trivially true \( \Rightarrow \) query answering is meaningless.

---

**Desiderata**

- Define suitable inconsistency-tolerant semantics that allow for meaningful query answering.
- Consider the TBox as a genuine representation of the domain, and restore consistency only at the instance level (intuitively, touch only the ABox to make it consistent with the TBox).
- Take inspiration from the literature on Consistent Query Answering [2, 12, 3], and in particular from those approaches that consider both incomplete and inconsistent databases [8, 9].
- Establish computational complexity of query answering in this setting.
- Look for tractable cases, or even cases in which consistent query answering is first-order rewritable.
The notion of AR-repair

Definition: AR-repair

Given an ontology \( \mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle \) an AR-repair (ABox Repair) of \( \mathcal{O} \) is a set of membership assertions \( \mathcal{A}' \) such that:

- \( \mathcal{A}' \subseteq \mathcal{A} \)
- \( \mathcal{A}' \) is \( \mathcal{T} \)-consistent, i.e., \( \text{Mod}(\mathcal{T}, \mathcal{A}') \neq \emptyset \)
- there does not exist \( \mathcal{A}'' \) s.t. \( \mathcal{A}' \subset \mathcal{A}'' \subseteq \mathcal{A} \) and \( \text{Mod}(\mathcal{T}, \mathcal{A}'') \neq \emptyset \)

In a sentence, an AR-Repair is a maximally consistent subset of the original ABox \( \mathcal{A} \).

The set of all AR-repairs of \( \mathcal{O} \) is denoted with \( \text{AR-Set}(\mathcal{O}) \)

Example 1: AR-repairs

\begin{itemize}
  
  \item **AR-Rep\textsubscript{1}**
  \{WhiteWine(grechetto), WhiteWine(falanghina), Beer(guinnes)\}

  \item **AR-Rep\textsubscript{2}**
  \{RedWine(grechetto), WhiteWine(falanghina), Beer(guinnes)\}

  \item **AR-Rep\textsubscript{3}**
  \{WhiteWine(grechetto), WhiteWine(falanghina), producedBy(guinnes, sandonna)\}

  \item **AR-Rep\textsubscript{4}**
  \{RedWine(grechetto), WhiteWine(falanghina), producedBy(guinnes, sandonna)\}

\end{itemize}
The AR-semantics

The AR-semantics, is based on the notion of AR-repair

**Definition**

The set of AR-models of an ontology \( \mathcal{O} \), denoted \( \text{Mod}_{\text{AR}}(\mathcal{O}) \), is:

\[
\text{Mod}_{\text{AR}}(\mathcal{O}) = \{ I \mid I \in \text{Mod}(\langle T, A_i \rangle), \text{ for some } A_i \in \text{AR-Set}(\mathcal{O}) \}
\]

AR-semantics actually coincides with the *loosely-sound semantics* proposed in [8, 9].

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

**Definition**

Let \( \mathcal{O} \) be a possibly inconsistent ontology, and let \( q \) be a BUCQs. We say that \( q \) is AR-consistently entailed, or simply AR-entailed, by \( \mathcal{O} \), written \( \mathcal{O} \models_{\text{AR}} q \), if \( I \models q \) for every \( I \in \text{Mod}_{\text{AR}}(\mathcal{O}) \).

Intractability of QA under AR-semantics

**DL-Lite**<sub>core</sub> is the least expressive member of the DL-Lite family and its constructs are shared among all logics of such a family.

A **DL-Lite**<sub>core</sub> TBox is a finite set of assertions of the form \( B_1 \sqsubseteq B_2 \) and \( B_1 \sqsubseteq \neg B_2 \) where each \( B_i \) is as follows:

\[
B \quad \rightarrow \quad A \mid \exists Q \quad \quad Q \quad \rightarrow \quad P \mid P^- 
\]

where \( A, P, P^- \), and \( \exists Q \) have the usual meaning.

A **DL-Lite**<sub>core</sub> ABox has the same form of a **DL-Lite**<sub>A</sub> ABox.

**Theorem**

Consistent instance checking for **DL-Lite**<sub>core</sub> is **coNP-hard**

**Proof**

Via reduction from unsatisfiability of a 3-CNF formula.
Intractability of QA under AR-semantics

**Theorem**

Answering BUCQs in $DL-Lite_A$ under the AR-semantics is coNP-complete

**Proof [sketch]**

coNP-hardness follows from the previous theorem. Membership in coNP is proved through the following algorithm

**Algorithm** entailment$(O, Q)$

**Input:** $DL-Lite_A$ ontology $O = \langle T, A \rangle$, BUCQ $Q$

**Output:** true or false

if there exists $A' \subseteq A$ such that

1. $A'$ is $T$-consistent;
2. for each $\alpha \in A \setminus A'$, $A' \cup \{\alpha\}$ is not $T$-consistent;
3. $\langle T, A' \rangle \not|= Q$;

then return false
else return true

---

How can we obtain tractable consistent query answering?

We have seen that even for a very limited ontology language ($DL-Lite_{core}$) and a very limited query languages (single atom ground queries), query answering is intractable under the AR-semantics.

Thus, even though the AR-semantics is a quite natural and reasonable choice for inconsistency-tolerant query answering, there is no hope to find tractable cases that may result interesting for practical purposes.

Towards the identification of tractable settings, we therefore propose a new semantics, which approximate the AR-semantics.

Our new semantics is based on the WIDTIO (When In Doubt Throw It Out) principle, proposed in the area of belief revision and update [23, 13], and allows us to deal with a single repair, rather than multiple ones (potentially exponentially many), as in the AR-semantics.
### The IAR semantics

#### Definition: IAR repair

Let \( O = \langle T, A \rangle \) be a possibly inconsistent \( DL-Lite_A \) KB. The **Intersection ABox Repair** (or \( IAR \)-repair) of \( O \), denoted by \( IAR-Set(O) \), is defined as

\[
IAR-Set(O) = \bigcap_{A_i \in AR-Set(O)} A_i
\]

Then, the set of \( IAR \)-models of \( O \), denoted \( Mod_{IAR}(O) \), is defined as

\[
Mod_{IAR}(O) = Mod(\langle T, IAR-Set(O) \rangle)
\]

A UBCQ \( q \) is \( IAR \)-entailed by \( O \), written \( O \models_{IAR} q \), if \( I \models q \) for every \( I \in Mod_{IAR}(O) \).

---

### Example: IAR semantics

In our ongoing example, we have the following IAR-repair

\[
IAR-Set(O) = \{\text{WhiteWine(falanghina)}\}
\]
IAR semantics vs. AR-semantics

The IAR-semantics is a sound approximation of the AR-semantics, in the sense that, for any ontology $O$, every model of $O$ under AR-semantics is also a model of $O$ under IAR-semantics.

**Theorem**

If $O = \langle T, A \rangle$ is a DL-Lite$_A$ ontology, then, $\text{Mod}_{AR}(O) \subseteq \text{Mod}_{IAR}(O)$.

**Proof**

For each $A_i \in \text{AR-Set}(O)$, it holds that $\text{IAR-Set}(O) \subseteq A_i$, from which it follows that $\text{Mod}(\langle T, A_i \rangle) \subseteq \text{Mod}(\langle T, \text{IAR-Set}(O) \rangle)$, since DL-Lite$_A$ is monotonic. Then, $\bigcup_{A_i \in \text{AR-Set}(O)} \text{Mod}(\langle T, A_i \rangle) = \text{Mod}_{AR}(O) \subseteq \text{Mod}_{IAR}(O)$.

The above theorem implies that, given a BUCQs $q$, $O \models_{IAR} q$ implies that $O \models_{AR} q$. The converse is instead not true: In the example, $O \models_{AR} \text{Wine(grechetto)}$, but $O \not\models_{IAR} \text{Wine(grechetto)}$.

Computing IAR-repair

**Algorithm** Compute-IAR-Repair($O$)

**input:** DL-Lite$_A$ ontology $O = \langle T, A \rangle$

**output:** DL-Lite$_A$ ABox $A'$

**begin**

let $D = \emptyset$

for each fact $\alpha \in A$ do

if $\langle T, \{\alpha\} \rangle$ unsatisfiable

then let $D = D \cup \{\alpha\}$;

for each pair of facts $\alpha_1, \alpha_2 \in A - D$ do

if $\langle T, \{\alpha_1, \alpha_2\} \rangle$ unsatisfiable

then let $D = D \cup \{\alpha_1, \alpha_2\}$;

return $A - D$

**end**

**Theorem**

Let $O$ be a DL-Lite$_A$ ontology. $\text{IAR-Set}(O) = \text{Compute-IAR-Repair}(O)$. Then, computing $\text{IAR-Set}(O)$ is polynomial in data complexity.
The above theorem has as immediate consequence that answering a BUCQ $q$ is in PTIME in data complexity. Indeed, verifying whether $\mathcal{O} \models_{IAR} q$ amounts to verifying whether $\langle T, \text{IAR-Set}(\mathcal{O}) \rangle \models q$, and we have just shown that $\text{Compute-IAR-Repair}(\mathcal{O}) = \text{IAR-Set}(\mathcal{O})$ runs in polynomial time in data complexity, whereas we already showed that entailment of BUCQs in $DL-Lite_\mathcal{A}$ is in $AC^0$ in data complexity (i.e., less than polynomial).

Can we do better? YES!

In what follows we show that the following theorem holds

**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, i.e., $\mathcal{O} \models_{IAR} q$, is in $AC^0$ in data complexity.

---

**FOL-rewritability of consistent query answering**

**Definition**

Query answering in a DL $\mathcal{L}$ is **FOL-rewritable under IAR-semantics**, if for every TBox $T$ expressed in $\mathcal{L}$ and every query $q$ over $T$, one can effectively compute a FOL query $q_r$ over $T$ such that, for every ABox $\mathcal{A}$, $\langle T, \mathcal{A} \rangle \models_{IAR} q$ iff $\langle \emptyset, \mathcal{A} \rangle \models q_r$. We call such $q_r$ the IAR-perfect FOL rewriting (or simply IAR-perfect rewriting) of $q$ w.r.t. $T$. 
Query Rewriting: intuition

Let us for the moment disregard PIs in the TBox, and concentrate only on the evaluation of a query over the ABox seen as a database.

Let $q$ be a BCQ over a $DL-Lite_A$ ontology $\mathcal{O} = \langle T, A \rangle$, and let $\sigma$ be a substitution from variables in $q$ to constants in $A$. If $\sigma(q)$ (considered as a set of atoms) is contained in $A$, we say that $\sigma(q)$ is an image of $q$ in $A$, and that $q$ evaluates to true over $A$.

Not all images of a query over the ABox are “good” images, since they can be involved in inconsistency (which in $DL-Lite_A$ means a violation of a NI or a functionality assertion). Good images are those belonging to all AR-repairs of $\mathcal{O}$.

Roughly speaking, we reformulate $q$ so as to encode into a FOL formula all violations that can involve ABox assertions belonging to images of $q$ on any ABox $A$. Indeed, this can be done by reasoning only on the TBox, and considering each query atom separately.

Example 2

Consider the query $q : \exists x. \text{WhiteWine}(x)$, and the following ontology

TBox

RedWine $\sqsubseteq \neg$ WhiteWine
WhiteWine $\sqsubseteq \neg$ Beer

RedWine $\sqsubseteq \neg$ WhiteWine
RedWine $\sqsubseteq \neg$ Beer

ABox

RedWine(grechetto)
WhiteWine(falanghina)

Both WhiteWine(grechetto) and WhiteWine(falanghina) are images of $q$ in $A$, but only WhiteWine(falanghina) is a good image.

The following query “filter out” bad images:

$\exists x. \text{WhiteWine}(x) \land \neg \text{RedWine}(x) \land \neg \text{Beer}(x)$
Query Rewriting: Dealing with negative inclusions between concepts

We denote with $B(t)$, where $B$ is a basic concept and $t$ is a term, any atom of the form $A(t)$, $P(t, t')$, $P(t', t)$ if $B = A$, or $B = \exists P$, or $B = \exists P^-$, respectively.

Associated to $B(t)$, we define $\text{NotDisjClash}^T_B(t)$ as the following FOL formula:

$$\bigwedge_{A \in DC(B, T)} \neg A(t) \bigwedge_{P \in DRD(B, T)} \neg \exists y. P(t, y) \bigwedge_{P \in DRR(B, T)} \neg \exists y. P(y, t)$$

where

$DC(B, T) = \{A \mid A$ is an atomic concept s.t. $T \models B \sqsubseteq \neg A\}$

$DRD(B, T) = \{P \mid P$ is an atomic role s.t. $T \models B \sqsubseteq \neg \exists P\}$

$DRR(B, T) = \{P \mid P$ is an atomic role s.t. $T \models B \sqsubseteq \neg \exists P^-\}$

Example 1 (cont.)

Let us consider Example 1:

\begin{align*}
\text{TBox} \\
\text{RedWine} \sqsubseteq \text{Wine} & \quad \text{WhiteWine} \sqsubseteq \text{Wine} \\
\text{RedWine} \sqsubseteq \neg \text{WhiteWine} & \quad \text{Beer} \sqsubseteq \neg \text{Wine} \\
\text{Wine} \sqsubseteq \exists \text{producedBy} & \quad \exists \text{producedBy} \sqsubseteq \neg \text{Wine} \\
\exists \text{producedBy}^- \sqsubseteq \neg \text{Winery} & \quad \text{Beer} \sqsubseteq \neg \exists \text{producedBy} \\
\text{Wine} \sqsubseteq \neg \text{Winery} & \quad \text{Beer} \sqsubseteq \neg \text{Winery} \\
\text{(funct} \text{ producedBy)} &
\end{align*}

We have that:

$$\text{NotDisjClash}^T_{\text{RedWine}}(t) = \neg \text{WhiteWine}(t) \land \neg \text{Winery}(t) \land \neg \text{Beer}(t)$$

and

$$\text{NotDisjClash}^T_{\text{Beer}}(t) = \neg \exists y. \text{producedBy}(t, y) \land \neg \text{Wine}(t) \land \neg \text{RedWine}(t) \land \neg \text{WhiteWine}(t) \land \neg \text{Winery}(t)$$
Query Rewriting: Dealing with negative inclusions between roles

Given an atom $P(t, t')$, where $P$ is an atomic role and $t, t'$ are terms, we define the formula $\text{NotDisjClash}^T_P(t, t')$ as follows:

$$\bigwedge_{S \in DR(P, T)} \neg S(t, t') \land \text{NotDisjClash}^T_{\exists P}(t) \land$$
$$\bigwedge_{S \in DIR(P, T)} \neg S(t', t) \land \text{NotDisjClash}^T_{\exists P-}(t')$$

where the sets $DR(P, T)$ and $DIR(P, T)$ are defined as follows:

$$DR(P, T) = \{ S \mid S \text{ is an atomic role and } T \models P \sqsubseteq \neg S \}$$
$$DIR(P, T) = \{ S \mid S \text{ is an atomic role and } T \models P \sqsubseteq \neg S^- \}.$$

Query Rewriting: Dealing with functionalities

Given a functionality assertion $(\text{funct } P)$ 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 $(\text{funct } P^-)$ 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 an atom $P(t, t')$, where $P$ is an atomic role and $t, t'$ are terms, we define $\text{NotFunctClash}^T_P(t, t')$ as the FOL formula:

$$\neg(\exists y. P(t, y) \land y \neq t')$$

if $(\text{funct } P)$ exists in the ontology, and

$$\neg(\exists y. P(y, t') \land y \neq t)$$

if $(\text{funct } P^-)$ exists in the ontology (we add both conditions if both the functionalities are present).
An atomic concept $A$ is unsatisfiable if $\mathcal{T} \models A \sqsubseteq \neg A$.

An ABox assertion $A(a)$ over an unsatisfiable concept $A$, violating a NI $A \sqsubseteq \neg A'$ together with $A'(a)$, is not a real inconsistency because $A(a)$ does not belong to any repair of $\mathcal{A}$.

**Example 2**

In our ongoing example 2, assume to have also that $\mathcal{T} \models \text{RedWine} \sqsubseteq \neg \text{RedWine}$.

The ABox assertion RedWine(grechetto) is $\mathcal{T}$-inconsistent. It is therefore not contained in any Repair, which means that $\text{IAR-Set} = \{\text{WhiteWine(grechetto)},\ \text{WhiteWine(falangina)}\}$

Thus, in this case both WhiteWine(grechetto) and WhiteWine(falangina) are good images of $q : \exists x.\text{WhiteWine}(x)$ in $\mathcal{A}$. But the query $\exists x.\text{WhiteWine}(x) \land \neg \text{RedWine}(x) \land \neg \text{Beer}(x)$ we constructed above filters out WhiteWine(grechetto) as well!

In order to skip false inconsistencies, for an atom $B(t)$ we define the condition

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

Analogously, we define $ConsAtom^T_P(t,t')$ for an atom of the form $P(t,t')$.

Such conditions are suitably put in conjunction with $NotDisjClash$ and $NotFunctClash$ formulas.
Skipping false inconsistencies

Considering the ConsAtom conditions, we obtain:

\[
\text{NotDisjClash}^T_B(t) = \bigwedge_{A \in DC(B,T)} \neg (A(t) \land \text{ConsAtom}^T_A(t)) \\
\bigwedge_{P \in DRD(B,T)} \neg (\exists y.P(t, y) \land \text{ConsAtom}^T_P(t, y)) \\
\bigwedge_{P \in DRR(B,T)} \neg (\exists y.P(y, t) \land \text{ConsAtom}^T_P(y, t))
\]

Similarly for \(\text{NotDisjClash}^T_P(t, t')\).

As for functionalities (both \((\text{funct } P)\) and \((\text{funct } P^-)\)) we have:

\[
\text{NotFunctClash}^T_P(t, t') = \neg (\exists y.P(t, y) \land y \neq t' \land \text{ConsAtom}^T_P(t, y)) \land \\
\neg (\exists y.P(y, t') \land y \neq t \land \text{ConsAtom}^T_P(y, t))
\]

Put it all together: IncRewriting\(_{IAR}(q, \mathcal{T})\)

Let \(q\) be a CQ of the form

\[
\exists x_1, \ldots, x_k. \bigwedge_{i=1}^n B_i(t_1^i) \land \bigwedge_{i=1}^m P_i(t_2^i, t_3^i)
\]

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

\[
\text{NotClash}^T_B(t) = \text{NotDisjClash}^T_B(t) \\
\text{NotClash}^T_P(t, t') = \text{NotDisjClash}^T_P(t, t') \land \text{NotFunctClash}^T_P(t, t')
\]

and use them to build the rewriting:

\[
\exists x_1, \ldots, x_k. \bigwedge_{i=1}^n B_i(t_1^i) \land \text{ConsAtom}^T_{B_i}(t_1^i) \land \text{NotClash}^T_{B_i}(t_1^i) \land \\
\bigwedge_{i=1}^m P_i(t_2^i, t_3^i) \land \text{ConsAtom}^T_{P_i}(t_2^i, t_3^i) \land \text{NotClash}^T_{P_i}(t_2^i, t_3^i)
\]
Example 2 (cont.)

\begin{block}{TBox}
\[
\begin{align*}
\text{RedWine} & \sqsubseteq \neg \text{RedWine} \\
\text{RedWine} & \sqsubseteq \neg \text{Beer} \\
\text{WhiteWine} & \sqsubseteq \neg \text{Beer} \\
\text{RedWine} & \sqsubseteq \neg \text{RedWine}
\end{align*}
\]
\end{block}

The query \( q : \exists x. \text{WhiteWine}(x) \) that we rewrote (for the case without \( \text{RedWine} \sqsubseteq \neg \text{RedWine} \) ) in

\[
\exists x. \text{WhiteWine}(x) \land \neg \text{RedWine}(x) \land \neg \text{Beer}(x)
\]

is now rewritten into

\[
\exists x. \text{WhiteWine}(x) \land \neg (\text{RedWine}(x) \land \text{false}) \land \neg \text{Beer}(x)
\]

which obviously is equivalent to

\[
\exists x. \text{WhiteWine}(x) \land \neg \text{Beer}(x)
\]

Final Rewriting Algorithm

\begin{algorithm}
\textbf{Algorithm} IAR-PerfectRef\((Q, \mathcal{T})\)
\textbf{input:} BUCQs \( Q \), \( DL-Lite_A \) TBox \( \mathcal{T} \)
\textbf{output:} FOL query over \( \mathcal{T} \)
\begin{algorithmic}
\State \( Q' = \text{PerfectRef}(Q, \mathcal{T}) \);
\State \( \phi = \bigvee_{q \in Q'} \text{IncRewriting}_{\text{IAR}}(q, \mathcal{T}) \);
\State \textbf{return} \( \phi \)
\end{algorithmic}
\end{algorithm}

Notice that the first step rewrites the query according to PIs, thus the algorithm takes into account all TBox assertions.
FOL-rewritability of QA in $DL$-$\text{Lite}_A$

Theorem

Let $\mathcal{T}$ be a $DL$-$\text{Lite}_A$ TBox and let $Q$ be a BUCQ. Then, $\text{IAR-PerfectRef}(Q, \mathcal{T})$ is a IAR-perfect FOL rewriting of $Q$ with respect to $\mathcal{T}$.

As a consequence, we obtain that consistent query answering of BUCQs is in $AC^0$ in data complexity.

CAR and ICAR semantics

We propose next two new inconsistency-tolerant semantics aimed at preserving logical consequences of maximal consistent subsets of $\mathcal{A}$.

**Definition: CAR-repair**

Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ a CAR-repair (Closed ABox repair) of $\mathcal{O}$ is a set of ABox assertions $\mathcal{A}'$ such that $\mathcal{A}'$ is an AR-repair of the ontology $\langle \mathcal{T}, clc(\mathcal{O}) \rangle$.

Where $clc(\mathcal{O}) = \{ \alpha \mid \alpha \in HB(\mathcal{O}) \text{ and there exists } S \subseteq A \text{ s.t. } Mod(T, S) \neq \emptyset \text{ and } \langle T, S \rangle \models \alpha \}$ are the consistent logical consequences and $HB(\mathcal{O})$ is the Herbrand Base of $\mathcal{O}$.

**Definition: ICAR-repair**

Intersection Closed ABox repair for $\mathcal{O}$ is the set $\mathcal{A}'$ of ABox assertions belonging to all the CAR-repairs of $\mathcal{O}$.
Example 1 (contd)

TBox

<table>
<thead>
<tr>
<th>Class Relation</th>
<th>Class Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedWine ⊑ Wine</td>
<td>WhiteWine ⊑ Wine</td>
</tr>
<tr>
<td>RedWine ⊑ ¬ WhiteWine</td>
<td>Wine ⊑ ¬ Beer</td>
</tr>
<tr>
<td>Wine ⊑ ∃ producedBy</td>
<td>∃ producedBy ⊑ Wine</td>
</tr>
<tr>
<td>Wine ⊑ ¬ Winery</td>
<td>Beer ⊑ ¬ Winery</td>
</tr>
<tr>
<td>∃ producedBy ⊑ Winery</td>
<td>(funct producedBy)</td>
</tr>
</tbody>
</table>

ABox

RedWine(grechetto)
WhiteWine(grechetto)
Beer(guinnes)
producedBy(guinnes, sandonna)
WhiteWine(falanghina)

Example 1: $clc(\mathcal{O})$

$clc(\mathcal{O})$

RedWine(grechetto)
WhiteWine(grechetto)
Beer(guinnes)
producedBy(guinnes, sandonna)
WhiteWine(falanghina)
Wine(falanghina)
Wine(grechetto)
Wine(guinnes)
Winery(sandonna)
Example 1: CAR-repairs

<table>
<thead>
<tr>
<th>CAR-Rep₁</th>
<th>{WhiteWine(grechetto), WhiteWine(falanghina), Beer(guinness), Wine(grechetto), Wine(falanghina), Winery(sandonna)}</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAR-Rep₂</td>
<td>{RedWine(grechetto), WhiteWine(falanghina), Beer(guinness), Wine(grechetto), Wine(falanghina), Winery(sandonna)}</td>
</tr>
<tr>
<td>CAR-Rep₃</td>
<td>{WhiteWine(grechetto), WhiteWine(falanghina), producedBy(guinness, sandonna), Wine(grechetto), Wine(falanghina), Wine(guinness), Winery(sandonna)}</td>
</tr>
<tr>
<td>CAR-Rep₄</td>
<td>{RedWine(grechetto), WhiteWine(falanghina), producedBy(guinness, sandonna), Wine(grechetto), Wine(falanghina), Wine(guinness), Winery(sandonna)}</td>
</tr>
</tbody>
</table>

Example 1: ICAR-repair

| ICAR-Rep | {WhiteWine(falanghina), Wine(grechetto), Wine(falanghina), Winery(sandonna)} |
AR-semantics vs CAR-semantics

- Let \( \mathcal{O} = \langle T, A \rangle \) and \( \mathcal{O}' = \langle T, A \rangle \) be two consistently equivalent (C-equivalent) ontologies, i.e., such that \( clc(\mathcal{O}) = clc(\mathcal{O}') \). Notably, we have that \( Mod_{CAR}(\mathcal{O}') = Mod_{CAR}(\mathcal{O}) \), but \( Mod_{AR}(\mathcal{O}') \neq Mod_{AR}(\mathcal{O}) \). Thus there can be \( \alpha \) such that \( \mathcal{O} \models_{AR} \alpha \) but \( \mathcal{O}' \not\models_{AR} \alpha \).

- AR-semantics is a sound approximation of the CAR-semantics, i.e., everything that is implied under AR-semantics is also implied under CAR-semantics.

- The converse does not hold. In our ongoing Example 1, let \( q \) be the BCQ \( \exists x. \text{Winery}(x) \), we have that \( \mathcal{O} \models_{CAR} q \), but \( \mathcal{O} \not\models_{AR} q \).

Instance checking under CAR semantics

The instance checking problem under CAR-semantics coincides with instance checking under the ICAR-semantics.

**Theorem**

Let \( \mathcal{O} \) be a DL-Lite\(_A\) ontology, and let \( \alpha \) be an ABox assertion. Then, \( \mathcal{O} \models_{CAR} \alpha \) iff \( \mathcal{O} \models_{ICAR} \alpha \).

**Proof**

\( \mathcal{O} \models_{CAR} \alpha \) iff \( \mathcal{K} \models_{ICAR} \alpha \) follows from the fact that the ICAR-semantics is a sound approximation of the CAR-semantics. As for the converse, since every CAR-repair is deductively closed, it follows that \( \mathcal{O} \models_{CAR} \alpha \) iff \( \alpha \) belongs to the intersection of all the CAR-repairs of \( \langle T, A \rangle \), i.e., to the ICAR-repair of \( \mathcal{O} \).
CAR-semantics vs ICAR-semantics

- Analogously to AR- and IAR-semantics, ICAR- is a sound approximation of CAR-semantics.
- Analogously to AR-semantics, consistent query answering for BUCQs in $DL-Lite_A$ is coNP-complete.
- Analogously to IAR-semantics, computation of the ICAR-repair is in PTIME in data complexity (similar algorithm but with $clc(O)$ instead of $A$).
- Analogously to IAR-semantics, consistent answering of BUCQs under ICAR semantics is FOL-rewritable.

UCQ rewriting under ICAR semantics

The algorithm is essentially the same for the IAR case.

We only have to define $IncRewriting_{ICAR}(q,T)$ as

$$\exists x_1, \ldots, x_k, \bigwedge_{i=1}^n \text{PerfectRef}(B_i(t_i^1),T) \land \text{ConsAtom}_{B_i}(t_i^1) \land \text{NotClash}_{B_i}(t_i^1) \land \bigwedge_{i=1}^m \text{PerfectRef}(P_i(t_i^2, t_i^3),T) \land \text{ConsAtom}_{P_i}(t_i^2, t_i^3) \land \text{NotClash}_{P_i}(t_i^2, t_i^3)$$
Summary of the complexity results for $DL-$Lite$_A$

<table>
<thead>
<tr>
<th></th>
<th>$AR$</th>
<th>$CAR$</th>
<th>$IAR$</th>
<th>$ICAR$</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst. checking</td>
<td>coNP-complete</td>
<td>in AC$^0$</td>
<td>in AC$^0$</td>
<td>in AC$^0$</td>
</tr>
<tr>
<td>UCQ entail.</td>
<td>coNP-complete</td>
<td>coNP-complete</td>
<td>in AC$^0$</td>
<td>in AC$^0$</td>
</tr>
</tbody>
</table>

*Note: the results on inconsistency-tolerance presented in these slides appeared in [16, 17]*

Further work on inconsistency-tolerance in ontologies

There is a bunch of work in belief revision and knowledge representation that specifically deals with inconsistency and incoherence in ontologies (see, e.g., [14, 15, 22]).

However, such work does not distinguish between inconsistency at the ABox and TBox level, as it is instead needed in OBDA, and are mainly focused on inconsistencies at the intensional level, rather than at the instance level.
Further work on inconsistency-tolerance in ontologies

The following works are instead focused on instance-level inconsistency-tolerance in ontology

- In [21], Rosati studies complexity of CQA under $AR$- and $IAR$-semantics for various DL languages (notably, also for tractable DLs like $\mathcal{EL}$, CQA results inherently intractable)
- In [4, 5, 6], Bienvenue studies complexity of CQA under AR-semantics for $DL-Lite$ for sub-classes of CQs, with the aim of identifying tractable cases. Also, she presents a semantics that resembles the ICAR-semantics, but under which instance checking for $DL-Lite_A$ is intractable.
- Bienvenue and Rosati in [7] present two parameterized inconsistency-tolerant semantics for DLs which may converge, depending on the parameter, to the $AR$- or $IAR$-semantics.
- In [18, 19], Lukasievicz et al. propose an application of the $IAR$-semantics to the framework of Datalog+/-

Acknowledgements

joint work with
- Diego Calvanese
- Giuseppe De Giacomo
- Maurizio Lenzerini
- Antonella Poggi
- Riccardo Rosati
- Marco Ruzzi
- Domenico Fabio Savo
Outline

1. Ontology-based data access
2. DL-Lite$_A$: an ontology language for accessing data
3. Dealing with inconsistency in ontology based data access
4. References

References

*Foundations of Databases.*  

Consistent query answers in inconsistent databases.  

*Database Repairing and Consistent Query Answering.*  

First-order expressibility results for queries over inconsistent DL-Lite knowledge bases.  
References II

Inconsistency-tolerant conjunctive query answering for simple ontologies.

On the complexity of consistent query answering in the presence of simple ontologies.
In Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI 2012), 2012.

Tractable approximations of consistent query answering for robust ontology-based data access.
In Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013), 2013.

References III

On the decidability and complexity of query answering over inconsistent and incomplete databases.

Query rewriting and answering under constraints in data integration systems.

Tractable reasoning and efficient query answering in description logics: The DL-Lite family.
References IV

Path-based identification constraints in description logics.

Consistent query answering: Five easy pieces.

On the complexity of propositional knowledge base revision, updates and counterfactuals.

A framework for handling inconsistency in changing ontologies.

References V

Reasoning with inconsistent ontologies.

Inconsistency-tolerant semantics for description logics.

Query rewriting for inconsistent DL-Lite ontologies.

Inconsistency-tolerant query rewriting for linear datalog+-/-.
Complexity of inconsistency-tolerant query answering in datalog+-.

Linking data to ontologies.

On the complexity of dealing with inconsistency in description logic ontologies.
In *Proc. of the 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI 2011)*, pages 1057–1062, 2011.

Non-standard reasoning services for the debugging of description logic terminologies.

*Updating Logical Databases*.